
Projeto Global da Licenciatura em Informática

# Aplicação Web de Comércio Eletrónico em MEAN Stack

Realizado por:

Fábio Filipe Severino Anselmo, Nº 2040

Coordenador/Orientador:

Professor Doutor Pedro Brandão

Lisboa, Junho de 2017

# Agradecimentos

Gostaria de fazer alguns agradecimentos as pessoas que me apoiaram ao longo da realização do projeto.

Agradeço ao ISTEC e seus professores por me terem passado o conhecimento para poder realizar este projeto.

Agradeço aos meus colegas de licenciatura pelas críticas que foram dando e pela motivação dada ao longo do projeto.

E por último, à minha família por toda a paciência, apoio e motivação ao longo do projeto.

# Resumo

Este relatório dedica-se ao desenvolvimento de uma aplicação *web* de comércio eletrónico ligada ao ramo da informática, onde esta é desenvolvida em HTML, CSS e MEAN *Stack*, com o objetivo de ser uma aplicação *web* escalável.

A aplicação é composta por uma base de dados NoSQL desenvolvida em MongoDB, de um servidor desenvolvido em Node.js, as API's para que o cliente possa fazer pedidos à base de dados são desenvolvidas em Express.js que por sua vez esses pedidos vão ser retornados pelo servidor ao cliente, em que este é desenvolvido em AngularJS.

**Palavras-chave:**

Aplicação web, Comércio eletrónico, HTML, CSS, MEAN Stack, MongoDB, Express.js, Node.js, AngularJS, NoSQL

# Abstract

This report is dedicated to the development of a web-bases ecommerce application in the field of information technology, where it is developed in HTML, CSS and MEAN Stack, with the aim of being a scalable web application.

The application consists of a NoSQL database developed in MongoDB, a server developed in Node.js, the API's from where the client can make the request from the database are developed in Express.js which these requests are return to the client by the server, the client is developed in AngularJS.

**Keywords:**

Web application, Ecommerce, HTML, CSS, MEAN Stack, MongoDB, Express.js, Node.js, AngularJS, NoSQL

# Abreviaturas

HTML – Hyper Text Markup Language

CSS – Cascading Style Sheets

MEAN – MongoDB, Express.JS, AngularJS e Node.js

DOM – Document Object Model

W3C – World Wide Web Consortium

NoSQL – Not Only Structured Query Language

MVC – Model-View-Controller

SPA – Single Page Application

JSON – JavaScript Object Notation

API – Application Programming Interface

BSON – Binary JavaScript Object Notation

RAM – Random-Access Memory

CPU – Central Processing Unit

I/O – Input/output

NPM – Node Package Manager

URL – Uniform Resource Locator

HTTP – Hypertext Transfer Protocol

RWD – Responsive Web Design

JWT – JavaScript Object Notation Web Token

CDN – Content Delivery Network

HMAC – Hash-based Message Authentication Code

# Índice

# Índice de Figuras

# Índice de Tabelas

# 1. Introdução

## 1.1) Âmbito

O presente projeto enquadra-se no âmbito do Projeto Global da licenciatura em informática, de modo ao aluno adquirir experiência e aprofundamento dos conhecimentos adquiridos ao longo dos três anos do curso. Os ensinamentos ao longo do curso não são suficientes para a nossa preparação, desse modo a realização do projeto vem tentar ajudar essa lacuna e para nós alunos é uma experiência importante, pois assim sentimo-nos mais preparados para o nosso futuro que é o mercado de trabalho.

A realização do projeto ajudou a que tenha um sentido de responsabilidade maior, tanto no cumprimento de prazos, tanto na capacidade de definir objetivos. E também a capacidade de realizar o projeto com novas tecnologias e poder adquirir mais conhecimentos sobre as mesmas.

## 1.2) Objetivos

Um dos objetivos fundamentais é a análise das tecnologias ligadas a esta área de desenvolvimento, que neste caso são o HTML, CSS e JavaScript. Outro é o uso de boas práticas no desenvolvimento de aplicações web usando MEAN *stack* (pilha) que é um conjunto de tecnologias, tendo especial atenção à criação da interface, pois é necessário garantir que a aplicação seja intuitiva e com boa usabilidade.

Contudo o objetivo principal deste projeto é desenvolver uma loja de comércio eletrónico, usando simplesmente as linguagens de programação HTML, CSS e MEAN *stack* para desenvolver o lado do cliente e do servidor.

É uma loja onde vai ter disponível produtos ligados à informática, onde os seus clientes vão puder ver os produtos disponíveis e adicionar os desejados ao carrinho de comprar, que posteriormente podem ser comprados através do cartão de crédito ou à cobrança com os devidos dados de envio. Uma vez, a compra realizada com sucesso o cliente receberá um email de confirmação da compra com o devido número de encomenda.

## 1.3) Motivação

A motivação no desenvolvimento do projeto foi sobretudo ter a possibilidade de desenvolver um projeto complexo, poder aprofundar os conhecimentos previamente adquiridos numa área de interesse pessoal e também o uso de novas tecnologias e novas linguagens de programação.

Optei por realizar uma loja de comércio eletrónico, por ter-me sido proposto por um amigo que está a iniciar a sua empresa no ramo da informática, a criação de um *website* onde pudesse disponibilizar produtos e que os seus clientes pudessem fazer a compra dos mesmos. Isto serviu também de motivação no desenvolvimento do projeto.

## 1.4) Estrutura do documento

O documento compreende quatro capítulos para além do presente capítulo, e está organizado da seguinte forma:

- No segundo capítulo é descrito o estado da arte.
- No terceiro capítulo é descrito a contextualização.
- No quarto capítulo é realizado o desenvolvimento da aplicação.
- No quinto e último capítulo são feitas algumas considerações finais sobre o projeto e pontos a melhorar ou acrescentar ao projeto.

# 2. Estado da Arte

## 2.1) Comércio Eletrónico

Com o crescimento da Internet por todo o mundo, a vida da população foi afetada por esta grande tecnologia. Hoje em dia, um dos mais importantes assuntos no mundo *online* é os empregos *online* e o comércio eletrónico a partir de casa (Khoshnampour & Nosrati, 2011, p. 94).

O comércio eletrónico pode ser definido como, o processo de compra e venda de bens ou serviços através de sistemas eletrónicos, como por exemplo, a Internet (Jones, 2009, p. 3). Em que este processo pode ser efetuado entre empresas, entre empresas e consumidores ou entre os setores públicos e privados (Moore, 2004, p. 1).

### *2.1.1) Introdução Histórica*

O comércio eletrónico foi inicialmente desenvolvido para facilitar as transações entres as empresas. No início dos anos 70, foi desenvolvido um sistema de comunicação que permite através de sistemas informáticos o envio e receção de documentos eletrónicos entre empresas, dado pelo nome de *Electronic Data Interchange* (Inci, 2009).

O termo de comércio eletrónico só foi inventado no início dos anos 90, quando a Internet começou a ser comercializada e as pessoas começaram a usar a *World Wide Web*. O primeiro exemplo de comércio eletrónico para consumidores, foi a *Boston Computer Exchange* criada em 1982, que consistia num mercado onde as pessoas podiam vender os seus computadores (Ying, 2008).

### *2.1.2) Vantagens*

As vantagens que o comércio eletrónico trouxe para os consumidores, é que estes podem efetuar compras a qualquer hora do dia durante todo o ano, compras estas que podem ser efetuadas em qualquer parte do mundo, desde que tenham acesso à Internet. Os consumidores através de uma rápida procurar por diferentes websites de comércio eletrónico,

podem comparar os produtos com o objetivo de ver qual o mais barato. O comércio eletrónico trouxe uma grande variedade de produtos que não estão disponíveis no comércio local (Nanehkaran, 2013, p. 191).

Para as empresas o comércio eletrónico fez com que, estes pudessem expandir os seus negócios para mercados internacionais. E para as empresas que estão só ligadas ao comércio eletrónico, só necessitam de ter uma sede, em vez de ter uma sede e várias lojas para ter o negócio a funcionar (Ma'aruf & Abdulkadir, 2012, p. 3071).

### *2.1.3) Desvantagens*

Uma desvantagem que é inevitável em qualquer negócio é a segurança, em que nesta área é a segurança do próprio *website*, para que os hackers não consigam roubar as informações pessoais dos consumidores. Qualquer pessoa pode criar um *website* de comércio eletrónico, onde alguns destes *websites* são criados com o propósito de roubar informações pessoais dos consumidores. Outra desvantagem é não existir nenhuma garantia de que o produto comprado chegue ao destino, e se chegar ao destino se este está ou não danificado (Nanehkaran, 2013, p. 191).

Comparando com uma loja física em que o consumidor efetua a compra na loja e a entrega do produto é feita no momento, numa loja *online* a entrega do produto ao destinatário pode ser demorada, pois esta pode vir do outro lado do mundo ou pode haver atrasos nos meios de envio. Outra grande preocupação ao se efetuar uma compra é o retorno dos produtos, que na maior parte dos *websites* de comércio eletrónico não aceitam o retorno dos mesmos (Niranjanamurthy, Kavyashree, & Chahar, 2013, p. 2366).

# *2.2)* JavaScript

JavaScript é a linguagem de programação mais usada para o lado do cliente em aplicações *web*. Hoje em dia todos os *browsers* modernos usam motores em JavaScript (Eloff & Torstensson, 2012, p. 9). Tem também a capacidade de manipular o *DOM* (Taly, Erlingsson, Mitchell, Miller, & Nagra, 2011, p. 363).

A W3C (2005) define o DOM como uma interface, que permite que programas e *scripts* possam aceder e atualizar dinamicamente o conteúdo dos documentos. Documentos estes que podem ser posteriormente processados e o resultado desse processamento pode ser incorporado de novo na página atual, sem a necessidade de se atualizar a página.

## *2.2.1) Introdução Histórica*

JavaScript foi criado por Brendan Eich na Netscape em que foi apresentado através do *browser* da Netscape. A Microsoft adotou o JavaScript em 1996 com o Internet Explorer *3*. Em 1997 Ecma International, uma organização de normas localizada na Europa, lançou a primeira norma ECMAScript, levando o JavaScript para fora da Netscape. As normas de ECMAScript tem vindo a ser lançadas constantemente até aos dias de hoje (Rider, 2016, p. 68).

## *2.3)* MEAN *Stack*

MEAN é o acrónimo de um aglomerado de componentes da linguagem de programação JavaScript, em que, estes componentes são:

- ***MongoDB*** – base de dados NoSQL;
- ***Express.js*** – uma *framework* do Node.js para aplicações *web*;
- ***AngularJS*** – é uma *framework* de MVC de JavaScript focada para a criação de aplicações SPA;
- ***Node.js*** – é um ambiente de *runtime* de JavaScript para o lado do servidor (Nyati, 2016).

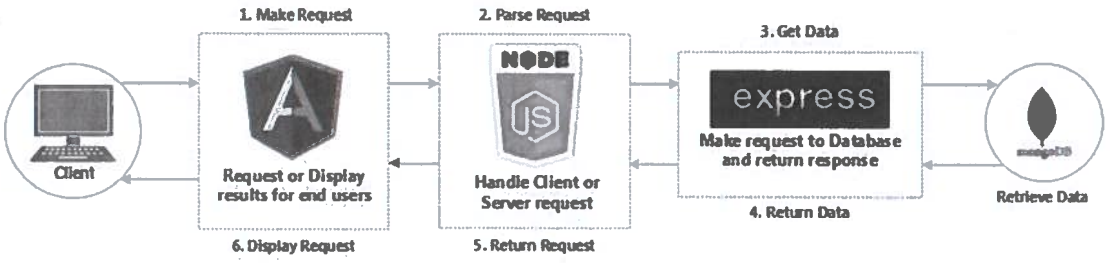


*Figura 1: Comunicação entre os componentes do MEAN* (Nyati, 2016)

Este acrónimo foi inicialmente inventado por Valeri Karpov num blog em 2013, enquanto desenvolvedor de *software* para a empresa MongoDB. Nesse mesmo blog indica que a utilização do MEAN *stack* trás alguns benefícios, tais como, a consulta à base de dados é feita em JavaScript e todos os seus dados são guardados em JSON, em que depois a API do servidor que é desenvolvida em JavaScript envia os dados para o cliente em formato de JSON, ou seja, não é necessário converter, e por último o lado do cliente é também desenvolvido me JavaScript (Karpov, 2013).

# 2.4) MongoDB

MongoDB é uma base de dados orientada a documentos, ou seja, é uma base de dados NoSQL. Por ser NoSQL, evita as estruturas baseadas em tabelas de base de dados relacionais, para adaptar documentos semelhantes ao JSON que contêm esquemas dinâmicos, chamado BSON. Isto facilita e torna mais rápido a integração dos dados para certos tipos de aplicações. MongoDB foi desenvolvido para escalabilidade, alta disponibilidade e alto desempenho, desde um único servidor até infraestruturas grandes e complexas (Techopedia, n.d.).

Em Outubro de 2007 foi desenvolvido pela empresa 10gen o MongoDB, hoje conhecida como MongoDB,Inc, com o objetivo de ser um componente para ser implementado numa plataforma como um serviço. Em 2009 o seu desenvolvimento foi mudado para *open-source* (Techopedia, n.d.).

## *2.4.1) Características*

### *Modelo de Dados*

MongoDB é uma base de dados orientada a documentos, em que armazena a informação em documentos BSON. Em que documentos em BSON consistem numa lista de elementos, em que cada um contém um nome e um tipo de dados e estes podem armazenar subdocumentos, ou seja, documentos dentro de documentos (MongoDB, 2015, p. 4).

Uma base de dados que seja orienta a documentos substitui o conceito de "*linhas*" por um modelo mais flexível "*documento*". E ao permitir que *arrays* sejam incorporados em documentos, torna possível a representação de relações hierárquicas complexas num único registo (Chodorow, 2013, p. 3).

### *Consulta dos Dados*

Segundo o *paper* da empresa (MongoDB, 2015, p. 5), o MongoDB fornece controladores nativos para todas as linguagens de programação e *frameworks* populares. Uma diferença fundamental em relação às bases de dados relacionais é que o modelo de consulta do

MongoDB é implementado como um método ou uma função dentro da API da linguagem de programação específica. A relação entre o modelo de dados orientado a documentos JSON do MongoDB e a estrutura de dados usada na programação orientada a objetos, faz com que a integração com as aplicações seja mais simples.

***Sharding***

Existem dois tipos de escalabilidade: escalabilidade vertical e escalabilidade horizontal. Onde o tipo de escalabilidade vertical é o mais facilitador, pois consiste em aumentar os recursos de uma única máquina, como por exemplo, a RAM ou o CPU, com o objetivo de puder suportar mais processamento. Mas tem as suas desvantagens, pois a partir de um certo ponto aumentar os recursos de uma máquina começa a ficar dispendioso. A escalabilidade horizontal é feita através de várias máquinas com os mesmos recursos, em que cada uma delas controla uma porção do processamento, fazendo com que haja um melhor desempenho (Haviv, 2014, p. 87).

O MongoDB suporta o tipo de escalabilidade horizontal também chamada de *sharding*. O sharding refere-se ao processo de dividir os dados por várias máquinas. Ao contrário das bases de dados relacionais, o *sharding* é automático e incorporado no próprio. Fazendo com que não seja preciso criar código para trabalhar com o *sharding* nas aplicações. Isto faz com que, não seja necessário implementar um *software* de *clustering* adicional ou de uma infraestrutura de discos compartilhados para gerir os processos e a distribuição dos dados ou de recuperação de falhas (MongoDB, 2015, p. 8).

***Replicação***

A replicação da base de dados usada no MongoDB é feita através de uma topologia conhecida como um conjunto de réplicas. Estes conjuntos de réplicas distribuem dados entre as várias maquinas para redundância e *failover* automático no caso de falha do servidor. Além disso, a replicação é usada para escalar a leitura da base de dado, ou seja, se houver uma aplicação de leitura intensiva, é possível espalhar a leitura da base de dados pelas máquinas que estão no cluster do conjunto de réplicas (Haviv, 2014, p. 86).

Os conjuntos de réplicas consistem num nó primário e num ou mais nós secundários, tal como a replicação *master-slave*. O nó primário suporta tanto escrita como leitura, mas os nós secundários só suportam leitura. O que torna os conjuntos de réplicas tão únicos é por estes

suportarem *failover* automático, ou seja, se por ventura o nó primário falhar ou não responder dentro de 10 segundos, o cluster selecionará um nó secundário para automaticamente o promover a nó primário. Quando o antigo nó primário voltar a estar online, esse vai se tornar um nó secundário (Banker, 2011, p. 157).

# 2.5)Node.js

O Node.js é um ambiente de *runtime* em JavaScript, baseado no motor de JavaScript V8 (motor de JavaScript usado no *browser Chrome*) (Foundation, 2017). Em que este usa um modelo direcionado para eventos e *non-blocking* I/O, fazendo com que as operações de I/O sejam efetuadas de forma assíncrona para evitar bloqueios (Iyer, 2013, p. 2).

Node.js é uma plataforma de *software* que permite criar servidores e aplicações *web* em cima dele (Holmes, 2016, p. 7).

## *2.5.1)* Node Package Manager

O NPM, é um repositório *online*[1] para publicar projetos *open-source* de Node.js e é também um utilitário para o Node.js, onde este é usado para procurar e carregar pacotes do repositório *online* através da linha de comandos. NPM torna mais fácil a partilha de código que foi criado por desenvolvedores para resolver um problema em especifico, com a finalidade de que outros desenvolvedores possam reutilizar esse mesmo código nas suas próprias aplicações (Reed, 2011).

## *2.5.2) Programação orientada a eventos*

Node.js usa programação orientada a eventos ou programação assíncrona, ao contrário da maioria das linguagens de programação que usam *blocking* I/O ou I/O síncrono, isto significa que, ao iniciar uma operação de I/O, como por exemplo, leitura do disco ou leitura na base de dados, a aplicação tem de permanecer em espera até que a operação esteja completa (Bretz & Ihrig, 2014, p. 44).

Enquanto que, na programação orientada a eventos, o fluxo de execução é determinado por eventos. Os eventos são manipulados por *callbacks*, em que estes *callbacks* são funções que vão ser invocadas após a conclusão da operação assíncrona. Fazendo com que seja possível realizar múltiplas operações de I/O em paralelo, em que a função *callback* da respetiva

[1] https://www.npmjs.com

operação é invocada quando esta é concluída, isto só é possível graças ao ciclo de eventos (*Event Loop*) (Teixeira, 2013, pp. 16–17).

### *2.5.3) Ciclo de Eventos*

Um conceito muito importante sobre o Node.js, é que este só tem uma *thread*, ou seja, só consegue realizar uma tarefa de cada vez. Mas este usa um ciclo de eventos, que lhe dá a capacidade de realizar múltiplas tarefas em paralelo (Bretz & Ihrig, 2014, p. 41).

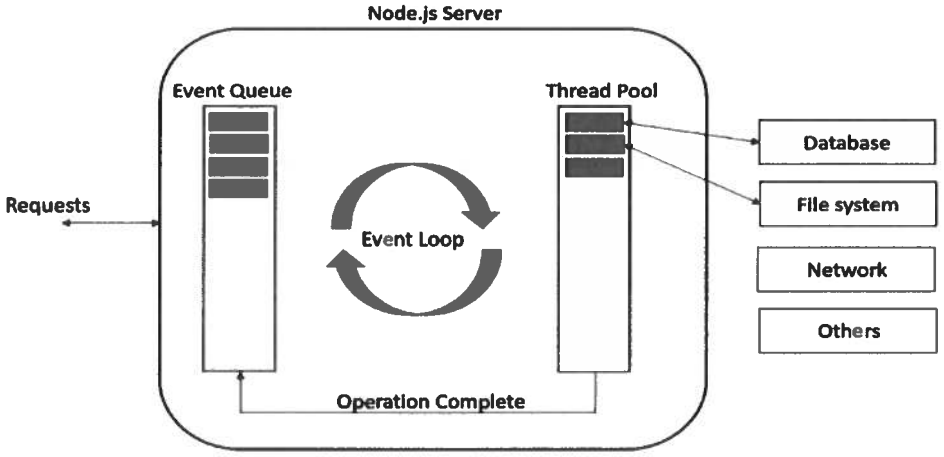


*Figura 2: Representação do ciclo de eventos do NodeJs (Olakara, 2015b)*

O ciclo de eventos tem a função de detetar novos eventos e manipular os eventos que estão no *Event Queue*, através de um ciclo contínuo (Teixeira, 2013, p. 16). Na qual, em cada ciclo, o ciclo de eventos itera sobre o *Event Queue*, se o item que estiver no início do *Event Queue* precisar de realizar operações de I/O, este é delegado para a *Thread Pool*. Após delegar, o ciclo de eventos contínua a executar os eventos que estão no *Event Queue*. Uma vez que a operação de I/O esteja concluída, o *callback* dessa mesma operação vai para o *Event Queue* para posteriormente ser processado. Quando o *callback* é executado pelo ciclo de eventos, este vai fornecer os resultados da operação (Olakara, 2015).

# 2.6) Express.js

O *website* oficial do Express.js, define-o como uma plataforma *web* flexível e minimalista, cuja função é simplificar a criação de aplicações *web* em Node.js. As duas funcionalidades importantes do Express.js são a possibilidade de criar *middlewares* e realizar o endereçamento de pedidos (Express, 2017a).

## *2.6.1)* Middlewares

*Middleware* é uma função que contém três argumentos o pedido (*request*), a resposta (*response*) e o próximo (*next*) (Mardan, 2014, p. 51). Um endereçamento de pedidos pode ter mais do que uma função *middleware* associada, onde estas funções podem ser usadas para verificar se o utilizador está autenticado, colocar um *cookie* ou um *header* (Bretz & Ihrig, 2014, p. 145). Em que este consegue realizar quatro tarefas:

- Executar código.
- Fazer alterações nos objetos *request* e *response*.
- Terminar o ciclo de pedido-resposta.
- Chamar o próximo *middleware* (Express, 2017b).

## *2.6.2) Endereçamento de pedidos*

Como em qualquer servidor *web*, o endereçamento de pedidos é uma parte importante para o seu funcionamento, pois é através do endereçamento de pedidos que o servidor consegue determinar a resposta para o cliente a partir do pedido que foi efetuado, em que este pedido é feito através de um URL ou caminho e um método de solicitação HTTP específico (*GET, POST, PUT, DELETE*) (Bretz & Ihrig, 2014, p. 144).

Um endereçamento de pedidos no servidor é geralmente feito da seguinte maneira: *app.get('/exemplo', function(req,res,next){...});*. O exemplo anterior vai fazer um pedido HTTP do tipo *GET*, cujo URL é "*/exemplo*", onde o servidor vai procurar e executar a função associada ao URL do tipo *GET*. Esta função pode conter dados para serem enviados como resposta para o cliente (Bretz & Ihrig, 2014, p. 144).

# 2.7) AngularJS

AngularJS é uma *framework* de JavaScript *open-source* para o lado do cliente, patrocinada e mantida pela *Google* (Freeman, 2014, p. 3), esta permite o uso do HTML como linguagem de *template* e amplia a sintaxe do HTML para representar os componentes da aplicação de forma mais clara e sucinta (Google, 2016). As aplicações em AngularJS são contruídas em torno de um padrão de arquitetura chamado *Model-View-Controller* (MVC) (Freeman, 2014, p. 3).

## *2.7.1)* Model-View-Controller

*Model-View-Controller* é um padrão de arquitetura que promove a organização do código dividindo as "*preocupações*", separando a interface para o utilizador (***View***) dos dados da aplicação (***Model***) através de um controlador (***Controller***) que controla os *inputs*, este delega as tarefas e coordena com o *model* e a *view* (Panda, 2014, p. 13) (*Figura 3*).

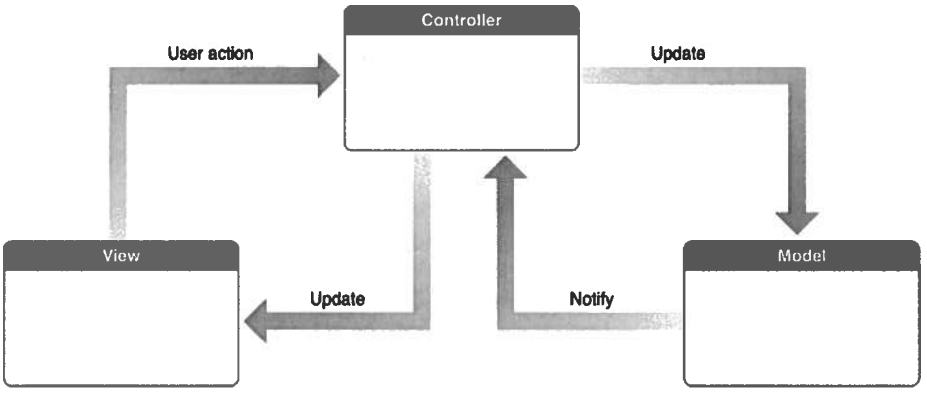


*Figura 3: Representação da arquitetura do Model-View-Controller (Apple Inc., 2015)*

## *2.7.2)* Model

O *model* é a força motriz de uma aplicação que use MVC. Geralmente, este está associado aos dados da aplicação que são obtidos do servidor. Qualquer interface para o utilizador que tenha dados visíveis, ou seja, que os dados sejam vistos pelo utilizador, são derivados do *model* (Seshadri & Green, 2014, p. 2).

### *2.7.3)* View

A *View* é responsável por transformar um ou mais *models* em representações visuais. Em aplicações *web*, isto significa gerar o HTML para o *browser* do utilizador processar (Chadwick, Snyder, & Panda, 2012, p. 6). Quaisquer alterações no *model* subjacente à *view*, faz com que esta seja atualizada automaticamente (Panda, 2014, p. 14).

### *2.7.4)* Controller

O *controller* ou controlador em português, é responsável pelo controlo da lógica da aplicação e da camada de apresentação que é a *view* (Seshadri & Green, 2014, p. 2). Este recebe *inputs* do utilizador através da *view*, e em seguida, trabalha com o *model* para executá-los, para puder enviar os resultados de volta para a *view* (Chadwick et al., 2012, p. 6).

### *2.7.5) Características*

#### *Two-way data binding*

O *Two-way data binding* é a sincronização automática dos dados entre a *view* e o *model*. Assim, sempre que houver uma alteração no valor do *model*, a *view* é automaticamente atualizada, e sempre que houver uma alteração no valor que um componente da *view*, o *model* associado a esse componente é atualizado (Panda, 2014, p. 38).

#### *Injeção de Dependências*

O AngularJS tem um subsistema de injeção de dependências que ajuda os desenvolvedores de código na criação de aplicações, facilitando o seu desenvolvimento, a sua compreensão e os seus testes. A injeção de dependências permite ao desenvolvedores de código pedir as dependências, em vez de ter de ir procurá-las manualmente. Para ter acesso as dependências, simplesmente é necessário adicionar o nome do serviço como parâmetro da função, e o AngularJS vai detetar e fornecer a instância do serviço (Seshadri & Green, 2014, p. 9).

# 3. Contextualização

A Internet tem afetado a maneira como as empresas realizam o seu negocio. E com isso tem surgindo novas maneiras para as empresas puderem promover ou vender os seus produtos, sem que as fronteiras sejam um problema. O MEAN *stack* é um conjunto de tecnologias recentes, em que o seu foco é o desenvolvimento de aplicação *web*.

O projeto global tem como objetivo desenvolver uma aplicação web com este conjunto de tecnologias, MEAN *stack*. Esta aplicação web é uma loja de comércio eletrónico de produtos ligados ao ramo da informática, onde os clientes podem efetuar a compra dos mesmos. Para isso, esta foi criada com uma interface simples de fácil utilização que suporta RWD, para que os clientes possam efetuar as suas compras de maneira fácil e sem complicações.

# 4. Desenvolvimento

## 4.1) Estrutura do Projeto

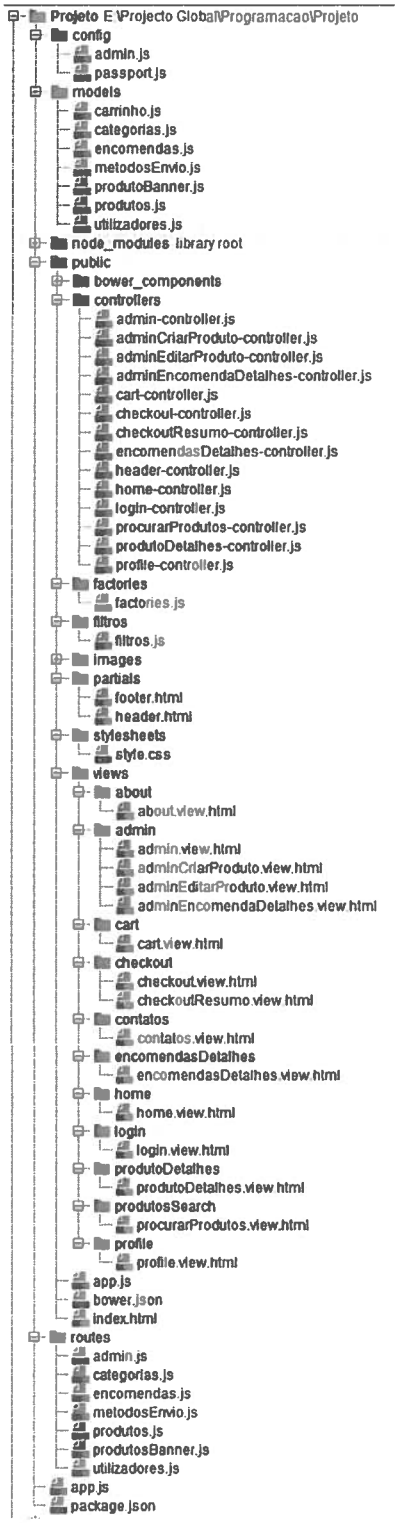


*Figura 4: Estrutura do Projeto*

# 4.2) Instalação das Tecnologias

Neste capítulo é demonstrado como se instala as tecnologias necessárias para que o projeto possa funcionar. Estas tecnologias são o MongoDB e o Node.js. As instalações seguintes vão ser realizadas para o sistema operativo Windows 10.

## *4.2.1) MongoDB*

Para instalar o MongoDB é necessário fazer o donwload do ficheiro de instalação, que esta disponível no *website* oficial do MongoDB[2]. Antes de realizar o download do ficheiro, ter especial atenção à arquitetura do sistema operativo que escolhe. Neste caso, foi realizado o download da versão "*Windows Server 2008 R2 64-bit and later, with SSL support x64*".

Depois do donwload estar concluído e abrir o ficheiro de instalação, vai aparecer uma janela semelhante á Figura 5:

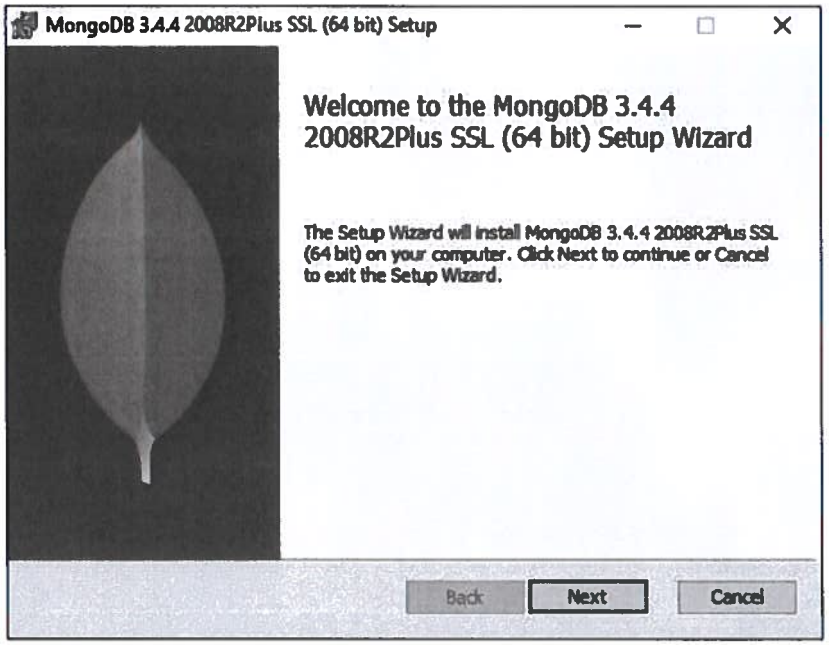


*Figura 5: Janela Inicial da Instalação do MongoDB*

Após aparecer esta janela, carregue no botão *Next* para prosseguir com a instalação, irá aparecer uma janela onde irá pedir o tipo de instalação que pretende, neste caso foi instalado

[2] https://www.mongodb.com

todas as funcionalidades que vêm com o ficheiro de instalação através do botão *Complete*. E por último carregue no botão *Install*, para começar a instalação do *MongoDB*.

Ao completar a instalação, uma janela semelhante à *Figura 6* irá aparecer:

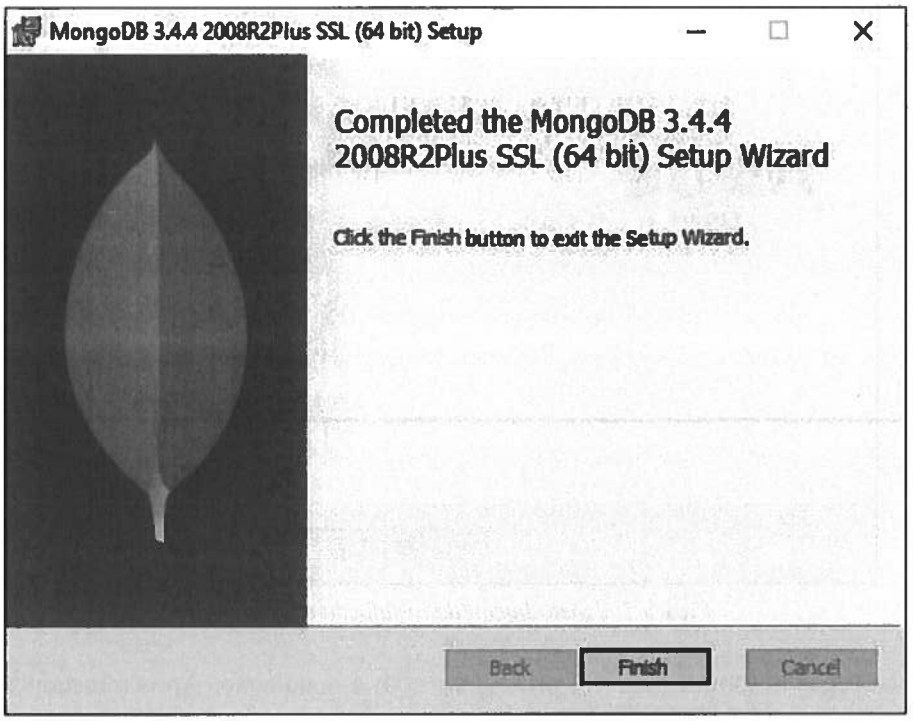


*Figura 6: Janela Final da Instalação do MongoDB*

Uma vez que a instalação esteja completa, é necessário criar a pasta onde a base de dados vai guardar os dados. O caminho que o MongoDB usa por defeito para guardar a base de dados é **C:\data\db**, para isso foram criadas essas pastas através do seguinte comando:

***> md C:\data\db***

### *4.2.2)* Node.js

Para instalar o Node.js é tão simples quanto fazer o download do instalador do *website* oficial do Node.js[3]. Neste caso, foi realizador o download do instalador que tem como descrição "*Recommend for Most Users*" que é a versão *stable*, a outra versão que está disponível no site é uma versão que contém as novas funcionalidades, mas estas funcionalidades nem sempre estão estáveis.

[3] https://nodejs.org/en/

Após o download ter sido feito com sucesso e ter aberto o ficheiro de instalação, uma janela semelhante à *Figura 7* irá aparecer:

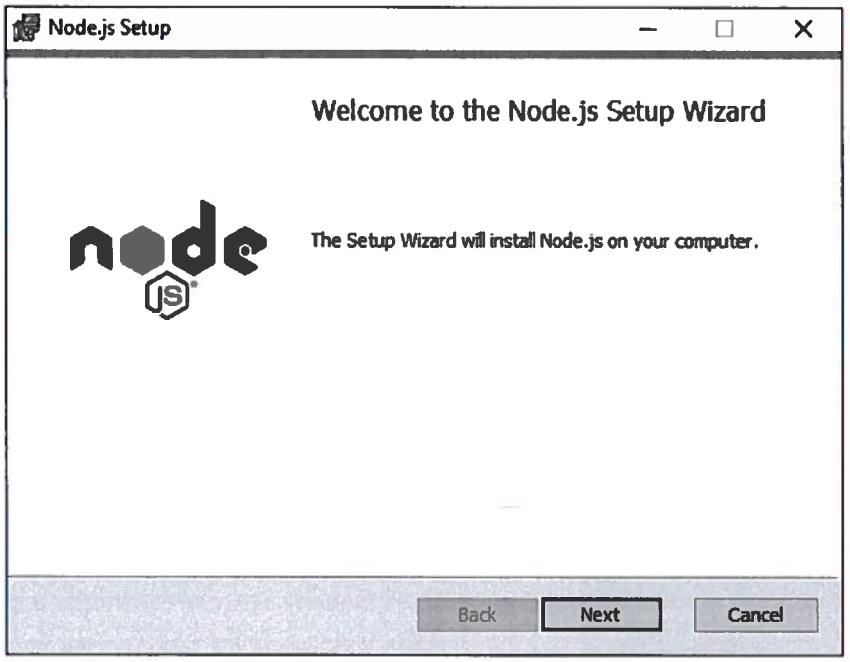


*Figura 7: Janela Inicial da Instalação do Node.js*

Ao carregar no botão *Next* vai prosseguir com a instalação. Após a instalação ter sido realizada com sucesso uma janela semelhante à *Figura 8* irá aparecer:

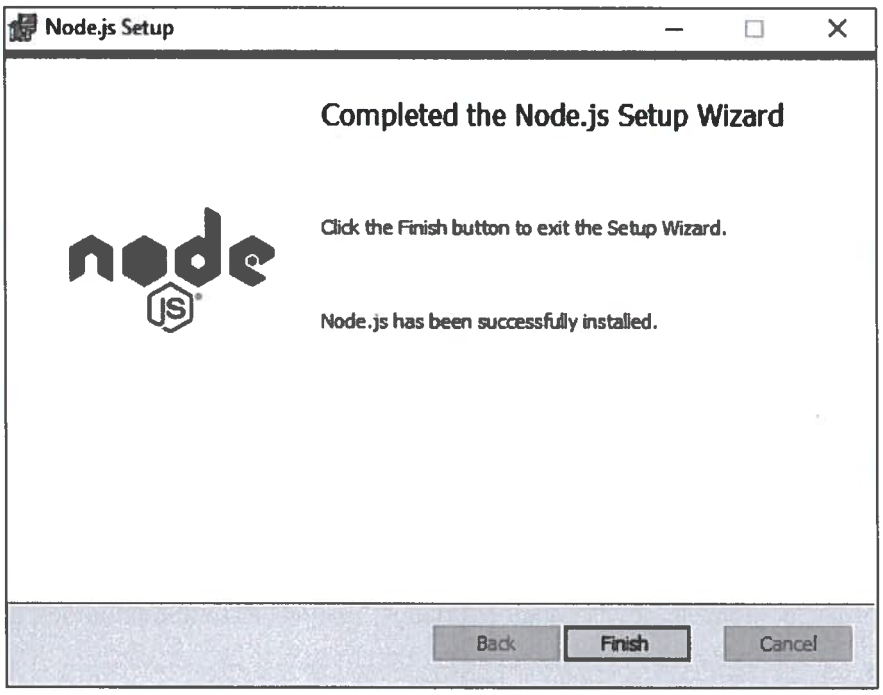


*Figura 8: Janela Final da Instalação do Node.js*

# 4.3) Descrição do funcionamento da aplicação

No presente projeto, ao iniciar o *website* o utilizador é um utilizador não autenticado, onde este é redirecionado para a página principal. Na página principal, estão expostos os produtos em destaque, os *banners* com promoções, uma barra de pesquisa que inclui um filtro para procurar os produtos através da categoria desejada e um *link* no cabeçalho para o utilizador ser redirecionado para a página do *login*. O utilizador anónimo tem a possibilidade de pesquisar todos os produtos associados a uma categoria ou subcategoria através do submenu no cabeçalho e visualizar os produtos em detalhe ao selecionar o produto.

Para utilizadores que tenham o *login* realizado, ou seja, utilizadores autenticados, estes têm todas a funcionalidades descritas acima e mais algumas, como por exemplo, têm a possibilidade de adicionar produtos ao carrinho de compras. Tem também a possibilidade de aceder ao seu perfil, onde podem alterar os seus dados pessoais, ver e remover produtos dos favoritos e ver as encomendas efetuadas. No carrinho de compras, o utilizador tem a capacidade de gerir os produtos adicionados ou efetuar a compra dos mesmos. Ao realizar a compra dos produtos o utilizador tem de preencher um formulário, sobre as informações para onde enviar, o método de envio e o modo de pagamento. O modo de pagamento é possível ser realizado através do cartão de crédito ou à cobrança. Ao efetuar a compra com sucesso o utilizador receberá um email de confirmação.

Os administradores, que são utilizadores autenticados, mas com permissões adicionais, têm as mesmas funcionalidades que um utilizador autenticado mais o acesso ao painel de administrador. O painel de administrador, é onde os administradores gerem o *website*. Têm a possibilidade de gerir os produtos (adicionar, editar e remover), dar privilégios aos utilizadores autenticados de administrador, alterar o estado das encomendas, gerir as subcategorias (adicionar e remover), gerir os métodos de envio (adicionar e remover) e gerir os *banners* (adicionar e remover).

Na *Figura 9*, está representado o caso de uso das funcionalidades que os utilizadores autenticados e os administradores têm no projeto. O caso de uso tem como objetivo mostrar as funcionalidades de um ou mais cenários do sistema (UML Tutorials, 2004, p. 2).

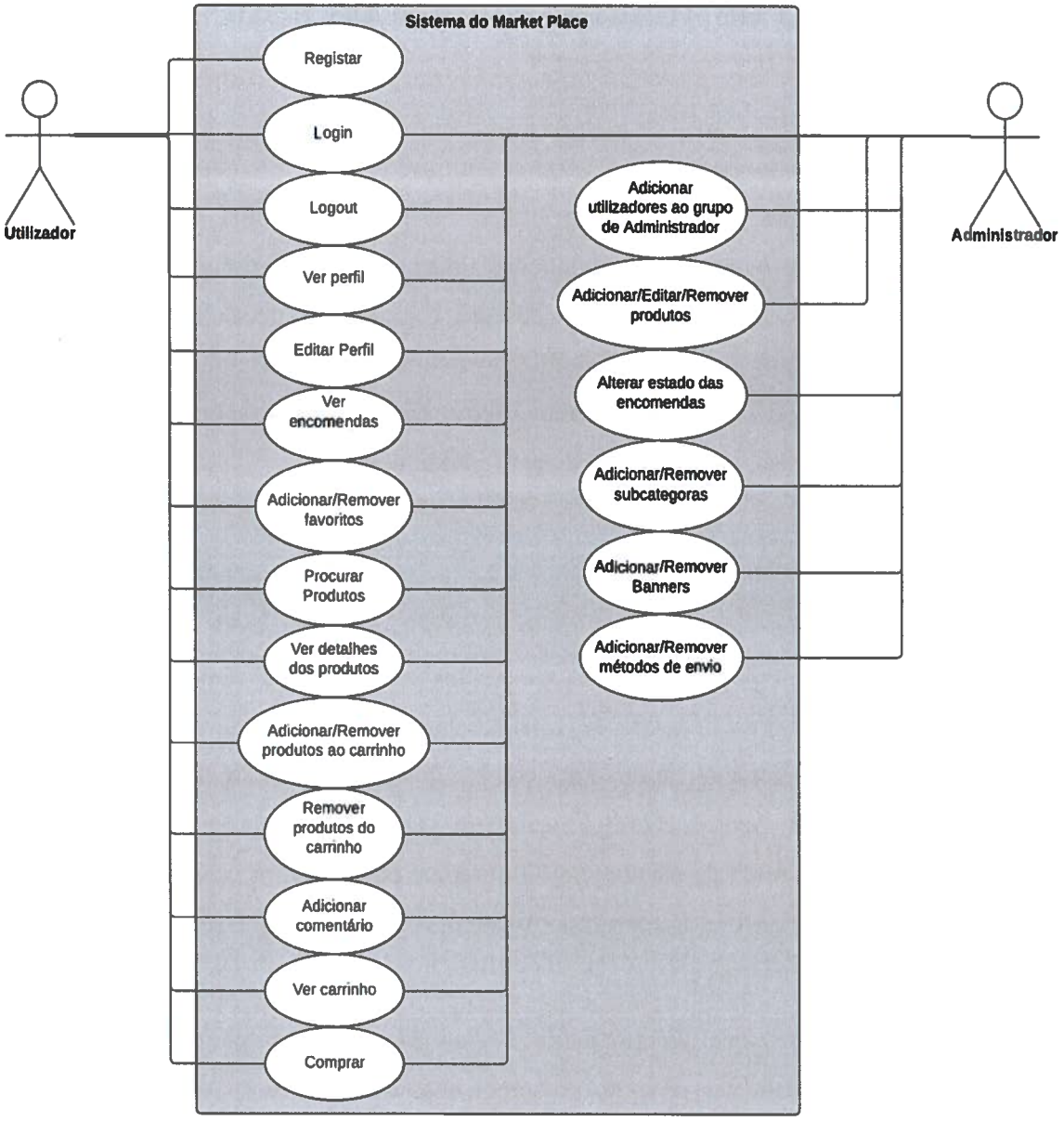


*Figura 9: Caso de Uso das funcionalidades dos utilizadores e dos administradores*

# 4.4) Módulos utilizadas

Neste capitulo são abordados os módulos fundamentais para o funcionamento do projeto. Estes estão divididos por servidor e cliente.

## *4.4.1) Servidor*

### *Mongoose*

Mongoose é um módulo do Node.js que permite o acesso ao MongoDB, este é uma ferramenta de modelação de objetos, em que esses objetos são depois guardados na base de dados com um documento. Para puder guardar os objetos na base de dados, é preciso primeiro criar um *schema*. Cada *schema* mapeia para uma coleção no MongoDB e define a configuração dos documentos dentro dessa coleção. Os *schemas* não só definem a estrutura dos documentos dentro da coleção, mas também permitem criar métodos (Teixeira, 2013, p. 314).

Um *model* representa todos os documentos associados a um *schema*, e é também o *model* que faz a conexão à base de dados para guardar ou obter os documentos da coleção a que este está associado (Teixeira, 2013, p. 314).

No código abaixo, está representado um exemplo de como se define um *schema* e de como se associa um *schema* a um *model*.

```
1.  var UserSchema = new mongoose.Schema({
2.          username: String,
3.          name: String,
4.          password: String
5.  });
6.
7.  var User = mongoose.model('User', UserSchema);
```

### *JsonWebToken*

*Json Web Token* ou JWT[4], é um módulo de transmissão de informação segura entre duas partes através de objetos JSON. A informação pode ser verificada pois esta contém uma assinatura digital. O JWT pode ser assinado através de um segredo, que usa o algoritmo HMAC

[4] https://github.com/auth0/node-jsonwebtoken

ou então usar um par de chaves pública/privada usando o algoritmo de encriptação RSA (JWT, 2017).

Este módulo é normalmente usado em conjunto com o *express-jwt*[5] que é um *middleware* que valida o JWT. Em que este permite autenticar os pedidos HTTP usando *tokens* em JWT, para proteger as API's do servidor.

### *Passport.js*

Passport.js[6] é um módulo para o Node.js. Onde este é implementado no Express.js como um *middleware*. O propósito deste módulo é de autenticar os pedidos vindo do cliente. Existem mais de 300 *strategies*, onde estas são módulos para autenticar os pedidos, autenticação esta que pode ser realizada localmente ou através das credenciais da Google, Facebook, Twitter entre outras (Hanson, 2015).

Para este projeto é usada a ***strategie* local**, em que esta é mais um módulo para o Node.js, que é implementado sobre o Passport.js. Esta *strategie* tem o nome de **passport-local**[7].

### *Stripe*

O Stripe[8] é mais um módulo para o Node.js, me que este fornece acesso à API da empresa Stripe. A Stripe é uma plataforma que permite realizar pagamentos *online*. O pagamento pode ser realizado através de cartão de crédito, apple pay, bitcoins entre outros (Stripe, 2017).

[5] https://github.com/auth0/express-jwt
[6] https://github.com/jaredhanson/passport
[7] https://github.com/jaredhanson/passport-local
[8] https://github.com/stripe/stripe-node

### *4.4.2) Cliente*

#### *Angular-ui-router*

O angular-ui-router[9] é uma *framework* para o AngularJS, esta é responsável por fazer o endereçamento das SPA's no lado do cliente. Esta *framework* fornece diferentes funcionalidades do que o endereçamento que vem com o AngularJS, em que as *views* são baseadas em estados e não simplesmente em URL. Em que cada estado está associado a um nome, um URL, a uma ou mais *views* e a um controlador que contém a lógica da *view*. Ao se alterar a *view*, é possível passar informação para a outra *view* através de argumentos nos estados (Ui-Router, 2017).

#### *Angular-jwt*

Angular-jwt[10] é uma biblioteca para o AngularJS que é usada especialmente para descodificar e verificar a data de expiração de um JWT. No projeto, esta é usada para descodificar a informação que é enviada pelo servidor quando um utilizador faz o *login* (Auth0, n.d.).

[9] https://github.com/angular-ui/ui-router
[10] https://github.com/auth0/angular-jwt

# 4.5) Implementação

## *4.5.1)* Base de dados

Na implementação da base de dados, o único requisito necessário é que o MongoDB esteja instalado no sistema operativo e a correr. Pois quem vai ser responsável por criar os *schemas* e criar os métodos para puder criar, alterar ou apagar os dados que estão na base de dados vai ser o servidor através do módulo Mongoose. No subcapítulo seguinte vão estar representados os *schemas* que foram usados e também os seus métodos.

## *4.5.2)* Servidor

Os módulos que são usados pelo servidor estão referenciados no ficheiro package.json, em que o objetivo deste ficheiro é conter o nome e a versão dos módulos que vão ser usados na aplicação, para que seja feito o download dos mesmos quando se executa o comando *npm install* na linha de comandos dentro da pasta do projeto.

### *4.5.2.1) Criação do servidor*

O primeiro passo é criar o servidor, este vai ser criado com a ajuda da *framework* Express.js. O código abaixo representa o ficheiro ***app.js***, que é responsável pela criação do servidor e configuração dos módulos.

```
1.  var express = require('express');
2.  var path = require('path');
3.  var favicon = require('serve-favicon');
4.  var logger = require('morgan');
5.  var bodyParser = require('body-parser');
6.  var passport = require('passport');
7.  require('./config/passport');
8.  var mongo = require("mongodb");
9.  var mongoose = require("mongoose");
10. var db = mongo.connection;
11. var multer = require('multer');
12. mongoose.connect('mongodb://localhost/projetoGlobal');
13.
14. var utilizadores = require('./routes/utilizadores');
15. var categorias = require('./routes/categorias');
16. var produtos = require('./routes/produtos');
17. var produtosBanner = require('./routes/produtosBanner');
18. var encomendas = require('./routes/encomendas');
19. var metodosEnvio = require('./routes/metodosEnvio');
20. var admin = require('./routes/admin');
```

```
21.
22. var app = express();
23.
24. app.use(logger('dev'));
25. app.use(bodyParser.json({ limit: '20mb' }));
26. app.use(bodyParser.urlencoded({ extended: false }));
27. app.use(passport.initialize());
28. app.use(express.static(path.join(__dirname, 'public')));
29. app.use(favicon(path.join(__dirname, 'public/images/', 'Logo.png')));
30.
31. app.use('/utilizadores', utilizadores);
32. app.use('/categorias', categorias);
33. app.use('/produtos', produtos);
34. app.use('/produtosBanner', produtosBanner);
35. app.use('/encomendas', encomendas);
36. app.use('/admin', admin);
37. app.use('/metodosEnvio', metodosEnvio);
38.
39. app.use(function(req, res, next) {
40.     var err = new Error('Not Found');
41.     err.status = 404;
42.     next(err);
43. });
44.
45. app.use(function(err, req, res, next) {
46.     res.status(err.status || 500);
47.     res.render('error');
48. });
49.
50. app.listen(3000);
```

Para uma melhor compreensão do código acima, este vai ser descrito em pormenor:

- **Linha 1 - 11:** Importação dos módulos para a aplicação;
- **Linha 12:** Conexão à base de dados local, que neste caso é a base de dados com o nome de **projetoGlobal**.
- **Linha 14 - 20:** Importação dos ficheiros que contêm os endereçamentos de pedidos para uma variável.
- **Linha 22:** É criado uma instância da *framework* Express.js.
- **Linha 24 - 29:** É realizada a configuração de alguns módulos anteriormente importados. Onde o *logger* é para realizar o *debug*, o *bodyParser* está associado ao corpo do pedido em que este está configurado para que o limite do corpo do pedido seja *20 mb* e para poder haver objetos dentro de objetos, o módulo *passport* é o módulo responsável por autenticar os utilizadores, o *express.static* é para dizer ao servidor que pode fornecer ficheiros que estejam dentro da pasta public para o cliente, e por último o módulo *favicon* é responsável por definir o ícone que aparece na aba do *browser*.
- **Linha 31 - 37:** São definidos os URL's da aplicação e os ficheiros de endereçamento de pedidos que estarão associados ao URL.

- **Linha 39 - 48:** São funções para apanhar os erros que surgem na aplicação, que não têm *handlers* associados.
- **Linha 50:** Responsável por criar o servidor e pô-lo à escuta de pedidos provenientes do porte 3000.

Com o ficheiro app.js criado já é possível iniciar o servidor usando o comando *npm start* na pasta do projeto, ter em atenção que a base de dados precisa de estar também a correr. Ao iniciar o servidor, este fica à espera de pedidos provenientes da porta 3000.


*Figura 10: Servidor a correr na linha de comandos*

### *4.5.2.2) Schemas*

Como foi mencionado anteriormente, o Mongoose é o módulo responsável por permitir o acesso ao MongoDB. Este é usado para obter ou guardar os dados na base de dados, mas para isso é necessário criar *schemas* e *models*. Em que *schemas* definem a configuração dos documentos dentro da própria coleção e os *models* representam os documentos do *schema* na qual está associado, e é também o *model* que faz a conexão á base de dados para guardar ou obter os documentos da coleção que está associado.

Para o projeto foi necessário criar *schemas* para as categorias, as encomendas, os métodos de envio, os *banners*, os produtos e os utilizadores. Alguns destes *schemas* têm métodos associados. Foram criadas funções que estão associadas ao ficheiro em que estas estão definidas, para depois serem chamadas noutros ficheiros pelo *model*. O objetivo era não haver código repetido, pois estas funções são chamadas mais do que uma vez. Em seguida vão ser mostrados os ficheiros que estão na pasta **models**, em que cada ficheiro contém um *schema*.

- ***categorias.js***

O ficheiro categoria.js contém o *schema* das categorias. Este contém também uma função com o nome de **getCategoriaPorNome**. O ficheiro pode ser consultado na íntegra no Apêndice A.1 deste documento.

```
1. var mongoose = require('mongoose');
2. var Schema = mongoose.Schema;
3.
4. var CategoriasSchema = new Schema({
5.     nome: { type: String, required: true },
6.     subCategorias: [{
7.         nome: { type: String }
8.     }]
9. });
10.
11. var Categorias = module.exports = mongoose.model('Categorias', CategoriasSchema);
12. (...)
```

A função **getCategoriaPorNome** é responsável por procurar a categoria que tenha o nome igual ao valor da variável que é passada como argumento.

- ***encomendas.js***

O ficheiro encomendas.js contém o *schema* das encomendas. O campo utilizador está a referenciar para a coleção Utilizador através do ID que é guardado.

```
1. var mongoose = require('mongoose');
2. var Schema = mongoose.Schema;
3.
4. var EncomendasSchema = new Schema({
5.     utilizador: { type: Schema.Types.ObjectId, ref: 'Utilizador' },
6.     items: { type: Object },
7.     moradaShipping: {
8.         morada: { type: String, required: true },
9.         localidade: { type: String, required: true },
10.        codigoPostal: { type: String, required: true },
11.        telefone: { type: Number, required: true }
12.    },
13.    estadoEncomenda: { type: String, default: 'Em Processamento' },
14.    data: { type: Date, default: Date.now },
15.    precoTotal: { type: Number, required: true },
16.    tipoDePagamento: { type: String, required: true },
17.    pagamentoID: { type: String, required: true },
18.    portes:{
19.        nome:{type:String, required:true},
20.        preco: {type:Number, required:true}
21.    }
22. });
23.
24. module.exports = mongoose.model('Encomendas', EncomendasSchema);
```

- ***metodosEnvio.js***

  O ficheiro metodosEnvio.js contém o *schema* dos métodos de envio.

```
1.  var mongoose = require('mongoose');
2.  var Schema = mongoose.Schema;
3.
4.  var MetodoEnvioSchema = new Schema({
5.      nome: {type: String, require: true},
6.      preco: {type: Number , require:true},
7.      model:{type: String, required: true}
8.  });
9.
10. module.exports = mongoose.model('MetodoEnvio', MetodoEnvioSchema);
```

- ***produtoBanners.js***

  O ficheiro produtoBanners.js contém o *schema* dos *banners*.

```
1.  var mongoose = require('mongoose');
2.  var Schema = mongoose.Schema;
3.
4.  var ProdutoBannerSchema = new Schema({
5.      nome: {type: String, require: true},
6.      imagem: {type: Object },
7.      active: {type: String, default: ""}
8.  });
9.
10. module.exports = mongoose.model('ProdutoBanner', ProdutoBannerSchema);
```

- **produtos.js**

O ficheiro produtos.js contém o *schema* dos produtos e funções que vão ser descritas a seguir. O ficheiro pode ser consultado na íntegra no Apêndice A.2 deste documento. O campo categoria está a referenciar para a coleção Categorias através do ID que é guardado e o username está a referenciar para a coleção Utilizador.

```
1.  var mongoose = require('mongoose');
2.  var Schema = mongoose.Schema;
3.
4.  var ProdutoSchema = new Schema({
5.      nome: {type: String, required: true},
6.      marca: {type: String, required: true},
7.      categoria: {type: Schema.Types.ObjectId, ref: 'Categorias'},
8.      subCategoria: {type: String},
9.      destaque: {type: Boolean, default: false},
10.     data: {type: Date, default: Date.now},
11.     preco: {type: Number, required: true},
12.     quantidade: {type: Number, required: true},
13.     descricao: {type: String},
```

```
14.     imagem: {
15.         nome: {type: String},
16.         file: {type: Object}
17.     },
18.     comentarios: [{
19.         username: {type: Schema.Types.ObjectId, ref: 'Utilizador'},
20.         conteudo: {type: String},
21.         introduzido: {type: Date, default: Date.now}
22.     }, {_id: false}]
23. });
24.
25. var Produto = module.exports = mongoose.model('Produto', ProdutoSchema);
26. (...)
```

As funções que estão associadas ao ficheiro são:

- ***findAllCategorias*** - Procura todos os produtos dentro da coleção e depois preenche o campo **categoria**. O campo **categoria** sem ser preenchido contem simplesmente os ID's e não os nomes das categorias.
- ***getProdutoPorId*** – Procura o produto que tenha o ID igual ao argumento que a função recebe do construtor, e depois preenche os campos **categoria** e **comentarios.username**.
- ***getProdutoPorNome*** – Procura os produtos que tenham o campo **nome** igual ou que contenha uma sequência de letras igual à variável que é passada como argumento no construtor. E depois preenche o campo **categoria**.
- ***getProdutoPorDestaque*** – Procura os produtos que tenham o campo **destaque** true.
- ***getProdutoPorNomeComCategoria*** – Procura os produtos que tenham o campo **nome** igual ou que contenha uma sequência de letras igual ao da variável **nome** que é passada como argumento e que tenham também o campo **categoria** igual ao da variável categoria que a passada como argumento no construtor, e depois preenche o campo **categoria**.
- ***getProdutoPorCategoria*** - Procura os produtos que tenham o campo **categoria** igual ao da variável categoria que é passada como argumento no construtor, e depois preenche o campo **categoria**.
- ***getProdutoPorNomeComSubCategoria*** - Procura os produtos que tenham o campo **nome** igual ou que contenha uma sequência de letras igual ao da variável **nome** que é passada como argumento e que tenham o campo **subCategoria** igual ao da variável **subCategoria** que é passada como argumento no construtor, e depois preenche o campo **categoria**.

- ***getProdutoPorSubCategoria*** - Procura os produtos que tenham o campo **subCategoria** igual ao da variável **subCategoria** que é passada como argumento no construtor, e depois preenche o campo **categoria**.
- ***AdicionarComentarios*** – Adiciona o comentário que é passado como argumento ao *array* já existente do produto cujo ID é passado como argumento no construtor.

- **utilizadores.js**

O ficheiro utilizadores.js contém o *schema* dos utilizadores e métodos associados ao *schema* dos utilizadores. Este contém também uma função com o nome de ***findFavoritos***. O ficheiro pode ser consultado na íntegra no Apêndice A.3 deste documento. O campo favoritos está a referenciar para a coleção Produto através do ID que é guardado no *array*.

```
1.  var mongoose = require('mongoose');
2.  var Schema = mongoose.Schema;
3.  var bcrypt = require('bcrypt-nodejs');
4.  var jwt = require('jsonwebtoken');
5.
6.  var UtilizadorSchema = new Schema({
7.      nomeCompleto: { type: String },
8.      username: { type: String, required: true },
9.      email: { type: String, required: true },
10.     password: { type: String, required: true },
11.     imagemPerfil: {
12.         nome: String,
13.         file: Object
14.     },
15.     morada: {
16.         morada: { type: String },
17.         localidade: { type: String },
18.         codigoPostal: { type: Number },
19.         telefone: { type: Number }
20.     },
21.     favoritos: [{ type: Schema.Types.ObjectId, ref: 'Produto' }],
22.     role: { type: String, default: 'user' },
23.     carrinho: { type: Object }
24. });
25.
26. UtilizadorSchema.methods.encryptPassword = function(password) {
27.     return bcrypt.hashSync(password, bcrypt.genSaltSync(8), null);
28. };
29.
30. UtilizadorSchema.methods.validPassword = function(password) {
31.     return bcrypt.compareSync(password, this.password);
32. };
33.
34. UtilizadorSchema.methods.gerarJwt = function() {
35.     return jwt.sign({
36.         _id: this._id,
37.         email: this.email,
38.         username: this.username,
39.         nomeCompleto: this.nomeCompleto,
40.         imagemPerfil: this.imagemPerfil || "",
```

```
41.         morada: this.morada || "",
42.         favoritos: this.favoritos || "",
43.         role: this.role,
44.         carrinho: this.carrinho,
45.         exp: Math.floor(Date.now() / 1000) + (60 * 60), //(1 hora)
46.     }, "secret");
47. };
48.
49. var Utilizador = module.exports = mongoose.model('Utilizador', UtilizadorSchema);
50. (...)
```

Os métodos que estão associados ao *schema* são:

- ***encryptPassword*** – Este método quando invocado encripta a variável que é a passada como argumento no construtor. No projeto este método é usado para encriptar a password quando o utilizador se regista. Este método usa o módulo *bcrypt*[11].
- ***validPassword*** – Este método quando invocado compara o valor da variável que é a passada como argumento no construtor com a password do utilizador que invoca este método. Na aplicação este método é usado para comparar a password que o utilizador introduzido no login com a password guardada na base de dados do utilizador que este pretende aceder. Este usa também o módulo *bcrypt*.
- ***gerarJwt*** – Este método quando invocado gera um JWT, ou seja, através do módulo *JsonWebToken*, cria um JWT com os dados do utilizador e uma data de expiração que está definido para uma hora. O JWT é codificado com uma palavra chave, que neste caso é "**secret**".

A função **findFavoritos** é responsável por procurar o ID do utilizador que é passado como argumento no construtor e preencher o campo favoritos. Este campo está associado aos produtos que o utilizador marcou como favoritos.

- **carrinho.js**

O carrinho.js não é um *schema*, mas sim um ficheiro que contém funções para criar um carrinho de compras, adicionar ou remover produtos ao carrinho de compras já existente. O ficheiro pode ser consultado na íntegra no Apêndice A.4 deste documento.

A função principal chama-se **Carrinho**, em que esta tem como argumento oldCart, que é para passar o carrinho de compras já existente. Se não for passado nenhum argumento, este

[11] https://www.npmjs.com/package/bcrypt-nodejs

cria um carrinho de compras novo. Onde o carrinho de compras é um objeto que vai conter os produtos adicionados pelo utilizador. Dentro da função, existem outras três funções, que são **adicionar**, **removerUm** e **removerItem**.

A função **adicionar** recebe como argumento item, id e quantidade. Primeiro a função vai verificar se o produto passado como argumento já existe ou não no *array items* que está associado aos produtos do carrinho de compras, se o produto já existir no *array* é adicionado a quantidade que foi passada como argumento na quantidade já existente do produto em questão, senão cria um objeto dentro do *array* com os dados do produto e aumenta o preço total e a quantidade total dependendo do preço do produto e da quantidade passada.

A função **removerUm** recebe como argumento o ID do produto que o utilizador pretende remover. Esta vai ao array *items* procurar o ID que é passado, e remover uma unidade no campo quantidade e reduzir o preço, também reduz o preço total e quantidade total. Se a quantidade do produto existente no *array items* for igual a 0 o produto é removido.

A função **removerItem** recebe como argumento o ID do produto que o utilizador pretende remover. Esta em vez de remover uma unidade do produto como é na função removerUm, remove o produto por completo do carrinho de compras. E reduz o preço total e a quantidade total dependendo do preço do produto e da quantidade removida.

#### *4.4.2.3) Middlewares*

Como foi visto anteriormente, *middlewares* são funções que podem ser chamadas pelo endereçamento de pedidos para realizar operações secundárias antes de ser realizada a operação principal.

Os *middlewares* criados foram:

- **local-login**

O código abaixo representa o código referente ao ficheiro passport.js que está na pasta config, que pode ser consultado na íntegra nos Anexos deste documento. Este *middleware* usa o módulo **passport-local** para criar uma nova *Strategy* com o nome de local-login.

```
1.  var passport = require('passport');
2.  var LocalStrategy = require('passport-local').Strategy;
3.  var Utilizador = require('../models/utilizadores.js');
4.
```

```
passport.use('local-login', new LocalStrategy({
        usernameField: 'email',
        passwordField: 'password',
    },
    function(email, password, done) {
        Utilizador.findOne({
            email: email
        }, function(err, user) {
            if (err) { return done(err); }
            if (!user) { return done(null, false); }
            if (!user.validPassword(password)) { return done(null, false); }
            user.password = undefined;
            token = user.gerarJwt();
            return done(null, token);
        });
    }
));
```

Este *middleware* recebe como parâmetros o usernameField e passwordField. O usernameField é o parâmetro que contém o email e o passwordField é o campo que contém a password.

A função recebe como argumentos os parâmetros anteriores mais o *done* que é o argumento responsável por terminar a função. Dentro da função, é feito uma buscar na coleção dos Utilizadores para verificar se o email que o utilizador introduziu existe ou não na base de dados. Se ocorrer algum erro é chamado o *done* passando esse mesmo erro como argumento, se o utilizador não existir ou se a password que o utilizador introduziu não é igual à password que esta guardada na base de dados é chamado o *done* passando o argumento *false* para indicar que as credenciais não estão corretas. Se o email e a password estiverem corretas, a password que está no objeto *user* é definida como *undefined* para esta não seja enviada para o cliente. Em seguida o objeto ***user*** é codificado utilizando o método **gerarJwt** que foi criado no *schema* do Utilizadores. Após realizar a codificação é chamado o *done* passando o *token* que é o objeto ***user*** codificado como argumento, em que este é enviado para a função do endereçamento de pedidos que chamou este *middleware*.

- **authenticateUser**

O authenticateUser é simplesmente uma instância do módulo *expressjwt* em que é passado o argumento secret. O valor do argumento secret tem de ser igual ao secret definido para codificar o *token* no *login*, que neste caso é na mesma "**secret**".

Este *middleware* é usado para validar o *token* enviado no *header* do pedido efetuado pelo cliente. Se o *token* estiver válido este permite aceder ás funções que estão dentro do

endereçamento, se não estiver válido é enviado para o cliente um erro com a mensagem "Token inválido".

O código abaixo representa o código necessário para criar este *middleware*, onde este é criado em todos os ficheiros que o usem.

```
1. (...)
2. var expressjwt = require('express-jwt');
3. var authenticateUser = expressjwt({secret: 'secret'});
4. (...)
```

- ***autenticacaoAdmin***

O *middleware* autenticacaoAdmin tem como objetivo verificar se o utilizador tem o campo role igual ao argumento role que é passado pelo construtor. Na aplicação, este *middleware* é usado para verificar se o utilizador tem a role de **admin**.

```
1.  var Utilizadores = require('../models/utilizadores');
2.
3.  function hasRole(role) {
4.      return hasRole[role] = function (req, res, next) {
5.              Utilizadores.findById({
6.                  _id: req.user._id
7.              }, function (err, user) {
8.                  user.password = undefined;
9.                  if (role === user.role) next();
10.                 else res.status(401, {message: 'Não Autorizado'}).end();
11.             });
12.         };
13. }
14.
15. module.exports = hasRole;
```

Este consiste numa função de nome **hasRole** que tem um argumento, onde dentro da função é realizada uma buscar na coleção dos Utilizadores para verificar se o utilizador existe. Se existir, é verificado se esse mesmo utilizador tem o campo role igual ao argumento role que foi passado, se tiver passa para o próximo *middleware* usando o next(), se não tiver a role igual é enviado para o cliente uma mensagem a dizer "Não Autorizado" e é chamado o end() para terminar o *middleware* e não permitir avançar para o próximo *middleware*.

Na aplicação é usado mais um *middleware*. *Middleware* este que é uma função existente num módulo para o Node.js, chamado de *multer*[12]. Em que este é usado para manipular o

[12] https://github.com/expressjs/multer

"***multipart/form-data***" para fazer *upload* de ficheiros, em que estes são guardados numa pasta que é especificada no início de cada ficheiro que use este módulo.

#### *4.5.2.4) Endereçamento de pedidos*

Endereçamento de pedidos refere-se à maneira de como a servidor vai responder ao pedido efetuado pelo cliente para um *endpoint* específico, em que este é um URL em conjunto com um método específico para solicitar o HTTP.

```
1.  app.Método(URL, Middleware, function (req, res) {
2.          Função
3.  })
```

O código acima representa a estrutura base de um endereçamento de pedidos. Onde **Método** é um método para solicitar o HTTP (*GET, POST, DELETE*, etc..), **URL** é o caminho para onde o cliente faz o pedido, **Middleware** é uma função intermédia que pode ou não existir, e **Função** é a função que é executada quando o endereçamento respetivo é chamado pelo cliente.

A pasta que contém os endereçamentos é a pasta **routes**, onde estes estão divido em diferentes ficheiros.

Como foi visto anteriormente, os URL's que a aplicação vai usar são definidos em conjunto com os ficheiros de endereçamento de pedidos, isto quer dizer que, os URL's dos ficheiros de endereçamentos vão ser agregados ao URL's base que foi definido em conjunto com o ficheiro. Por exemplo, se o URL definido na criação do servidor for "**/istec**" e o ficheiro com que este foi definido tenha um endereçamento com o URL "**/Lisboa**", o cliente vai conseguir aceder ao URL "**/istec/Lisboa**" onde a função associada ao endereçamento do URL "**/Lisboa**" vai ser executada.

Em seguida, vão ser definidos os endereçamentos de pedidos de cada ficheiro que está na pasta **routes**. Vai ser usado a estrutura base do código acima, para definir os endereçamentos através de uma tabela.

- ***admin.js***

Este ficheiro de endereçamento de pedidos é responsável por responder aos pedidos provenientes do painel de administrador. O URL associado a este ficheiro é o "/admin", em que este é definido no ficheiro app.js, que pode ser consultado na íntegra no Apêndice A.5 deste documento.

Todos os endereçamentos que estão neste ficheiro têm os *middlewares* **authenticateUser** e **autenticacaoAdmin("admin")**, então só os utilizadores com o campo role igual a "**admin**" é que podem fazer pedidos para os URL's deste ficheiro e estes têm de ter o *token* válido.

Em seguida vão ser mostrados todos os endereçamentos que estão no ficheiro admin.js.

*Tabela 1: Estrutura do endereçamento '/criarSubCategoria/:categoriaID' do ficheiro admin.js*

| **Método** | GET |
|---|---|
| **URL** | "/criarSubCategoria/:categoriaID" |
| **Middleware** | authenticateUser, autenticacaoAdmin('admin') |
| **Função** | Responsável por adicionar uma subcategoria que vem no corpo do pedido à categoria que vem com parâmetro no URL, que é categoriaID. |

*Tabela 2: Estrutura do endereçamento '/removerSubCategoria/:categoriaID' do ficheiro admin.js*

| **Método** | POST |
|---|---|
| **URL** | "/removerSubCategoria/:categoriaID" |
| **Middleware** | authenticateUser, autenticacaoAdmin('admin') |
| **Função** | Responsável por remover a subcategoria cujo nome vem no corpo do pedido, da categoria que vem com parâmetro no URL, que é categoriaID. |

*Tabela 3: Estrutura do endereçamento '/criarProduto' do ficheiro admin.js*

| **Método** | GET |
|---|---|
| **URL** | "/criarProduto" |
| **Middleware** | authenticateUser, autenticacaoAdmin('admin') |
| **Função** | Responsável por criar um produto com as informações que vêm no corpo do pedido. |

*Tabela 4: Estrutura do endereçamento '/editarProduto' do ficheiro admin.js*

| | |
|---|---|
| **Método** | POST |
| **URL** | "/editarProduto" |
| **Middleware** | authenticateUser, autenticacaoAdmin('admin') |
| **Função** | Responsável por editar o produto cujo o ID e as informações vêm no corpo do pedido. |

*Tabela 5: Estrutura do endereçamento '/alterarImagemProduto/:produtoID' do ficheiro admin.js*

| | |
|---|---|
| **Método** | POST |
| **URL** | "/alterarImagemProduto/:produtoID" |
| **Middleware** | authenticateUser, autenticacaoAdmin('admin'),upload.single('file') |
| **Função** | Responsável por alterar a imagem do produto cujo ID vem como parâmetro no URL, que é produtoID, através do *middleware* **upload.single('file')**. |

*Tabela 6: Estrutura do endereçamento '/removerProduto/:produtoID' do ficheiro admin.js*

| | |
|---|---|
| **Método** | POST |
| **URL** | "/removerProduto/:produtoID" |
| **Middleware** | authenticateUser, autenticacaoAdmin('admin') |
| **Função** | Responsável por remover o produto cujo ID vem como parâmetro no URL, que é produtoID. |

*Tabela 7: Estrutura do endereçamento '/encomendaAlterar/:encomendaID' do ficheiro admin.js*

| | |
|---|---|
| **Método** | POST |
| **URL** | "/encomendaAlterar/:encomendaID" |
| **Middleware** | authenticateUser, autenticacaoAdmin('admin') |
| **Função** | Responsável por alterar o estado da encomenda cujo ID vem como parâmetro no URL, que é encomendaID, em que esta altera o estado da encomenda para enviada. |

*Tabela 8: Estrutura do endereçamento '/utilizadoresRole/:userID' do ficheiro admin.js*

| **Método** | POST |
|---|---|
| **URL** | "/utilizadoresRole/:userID" |
| **Middleware** | authenticateUser, autenticacaoAdmin('admin') |
| **Função** | Responsável por alterar a role do utilizador cujo ID vem como parâmetro no URL, que é userID, onde o nome da role vem no corpo do pedido. |

*Tabela 9: Estrutura do endereçamento '/metodosEnvio/adicionar' do ficheiro admin.js*

| **Método** | POST |
|---|---|
| **URL** | "/metodosEnvio/adicionar" |
| **Middleware** | authenticateUser, autenticacaoAdmin('admin') |
| **Função** | Responsável por criar um método de envio com as informações que vêm no corpo do pedido. |

*Tabela 10: Estrutura do endereçamento '/metodosEnvio/remover/:metodoID' do ficheiro admin.js*

| **Método** | POST |
|---|---|
| **URL** | "/metodosEnvio/remover/:metodoID" |
| **Middleware** | authenticateUser, autenticacaoAdmin('admin') |
| **Função** | Responsável por remover o método de envio cujo ID vem como parâmetro no URL, que é metodoID. |

- ***categorias.js***

Este ficheiro de endereçamento de pedidos é responsável por enviar as informações sobre as categorias existentes na base de dados para o cliente, este não necessita de nenhum *middleware*, pois não é preciso nenhum tipo de permissão para aceder a este ficheiro. O URL associado a este ficheiro é o "**/categorias**", em que este é definido no ficheiro app.js, que pode ser consultado na íntegra no Apêndice A.6 deste documento.

Em seguida vão ser mostrados todos os endereçamentos que estão no ficheiro categorias.js.

*Tabela 11: Estrutura do endereçamento '/' do ficheiro categoria.js*

| **Método** | GET |
|---|---|
| **URL** | "/" |
| **Middleware** | - |
| **Função** | Responsável por enviar para o cliente todas as categorias existentes na base de dados. |

*Tabela 12: Estrutura do endereçamento '/:categoria' do ficheiro categoria.js*

| **Método** | GET |
|---|---|
| **URL** | "/:categoria" |
| **Middleware** | - |
| **Função** | Responsável por enviar para o cliente todas as categorias existentes na base de dados se o parâmetro categoria for igual a "Todas as Categorias", senão envia simplesmente a categoria que vem como parâmetro. |

- ***encomendas.js***

Este ficheiro de endereçamento de pedidos é responsável por enviar as informações sobre as encomendas existentes na base de dados para o cliente. O URL associado a este ficheiro é o "/encomendas", em que este é definido no ficheiro app.js, que pode ser consultado na íntegra no Apêndice A.7 deste documento.

Em seguida vão ser mostrados todos os endereçamentos que estão no ficheiro encomendas.js.

*Tabela 13: Estrutura do endereçamento '/' do ficheiro encomendas.js*

| **Método** | GET |
|---|---|
| **URL** | "/" |
| **Middleware** | authenticateUser |
| **Função** | Responsável por enviar para o cliente todas as encomendas existentes na base de dados. |

*Tabela 14: Estrutura do endereçamento '/:userID' do ficheiro categoria.js*

| **Método** | GET |
|---|---|
| **URL** | "/:userID" |
| **Middleware** | authenticateUser |
| **Função** | Responsável por enviar para o cliente todas as encomendas que estejam associadas ao utilizador cujo ID vem como parâmetro no URL, que é userID. |

*Tabela 15: Estrutura do endereçamento '/encomendaDetalhe/:encomendaID' do ficheiro categoria.js*

| **Método** | GET |
|---|---|
| **URL** | "/encomendaDetalhe/:encomendaID" |
| **Middleware** | authenticateUser |
| **Função** | Responsável por enviar para o cliente a encomenda cujo ID vem como parâmetro no URL, que é encomendaID. |

- ***metodosEnvio.js***

Este ficheiro de endereçamento de pedidos é responsável por enviar os métodos de envio existentes na base de dados para o cliente. O URL associado a este ficheiro é o "**/metodosEnvio**", em que este é definido no ficheiro app.js, que pode ser consultado na íntegra no Apêndice A.8 deste documento.

Em seguida vão ser mostrados todos os endereçamentos que estão no ficheiro metodosEnvio.js.

*Tabela 16: Estrutura do endereçamento '/' do ficheiro metodosEnvio.js*

| **Método** | GET |
|---|---|
| **URL** | "/" |
| **Middleware** | authenticateUser |
| **Função** | Responsável por enviar para o cliente todas os métodos de envio existentes na base de dados. |

- ***produtos.js***

Este ficheiro de endereçamento de pedidos é responsável por manipular os produtos e realizar a compra dos mesmos. O URL associado a este ficheiro é o "**/produtos**", em que este é definido no ficheiro app.js, que pode ser consultado na íntegra no Apêndice A.9 deste documento.

Em seguida vão ser mostrados todos os endereçamentos que estão no ficheiro produto.js.

*Tabela 17: Estrutura do endereçamento '/' do ficheiro produtos.js*

| **Método** | GET |
|---|---|
| **URL** | "/destaque" |
| **Middleware** | - |
| **Função** | Responsável por enviar para o cliente todos os produtos que tenham o campo destaque igual a *true*. |

*Tabela 18: Estrutura do endereçamento '/Categorias' do ficheiro produtos.js*

| **Método** | GET |
|---|---|
| **URL** | "/Categorias" |
| **Middleware** | - |
| **Função** | Responsável por enviar para o cliente todas os produtos existentes na base de dados. |

*Tabela 19: Estrutura do endereçamento '/detalhes/:id' do ficheiro produtos.js*

| **Método** | GET |
|---|---|
| **URL** | "/detalhes/:id" |
| **Middleware** | - |
| **Função** | Responsável por enviar para o cliente o produto cujo ID vem como parâmetro no URL, que é id. E envia em conjunto uma mensagem que contém "Disponível" se a quantidade do produto for maior que 5, "Stock Limitado" se a quantidade do produto for maior que 0 e menor que 5 e "Indisponível" se a quantidade do produto for igual a 0. |

*Tabela 20: Estrutura do endereçamento '/procurar/:categoria/:nome' do ficheiro produtos.js*

| **Método** | GET |
|---|---|
| **URL** | "/procurar/:categoria/:nome" |
| **Middleware** | - |
| **Função** | Responsável por enviar para o cliente o produto cujo nome vem como parâmetro no URL, que é nome e a categoria seja igual à que categoria que vem como parâmetro no URL, que é categoria. |

*Tabela 21: Estrutura do endereçamento '/procurar/:categoria' do ficheiro produtos.js*

| **Método** | GET |
|---|---|
| **URL** | "/procurar/:categoria" |
| **Middleware** | - |
| **Função** | Responsável por enviar para o cliente os produtos que tenham a categoria igual ao parâmetro do URL categoria. |

*Tabela 22: Estrutura do endereçamento 'subCategoria/procurar/:subCategoria' do ficheiro produtos.js*

| **Método** | GET |
|---|---|
| **URL** | "/subCategoria/procurar/:subCategoria" |
| **Middleware** | - |
| **Função** | Responsável por enviar para o cliente os produtos que tenham a subcategoria igual ao parâmetro do URL subCategoria. |

*Tabela 23: Estrutura do endereçamento '/subCategoria/procurar/:subCategoria/:produtonome' do ficheiro produtos.js*

| **Método** | GET |
|---|---|
| **URL** | "/subCategoria/procurar/:subCategoria/:produtonome" |
| **Middleware** | - |
| **Função** | Responsável por enviar para o cliente o produto cujo nome vem como parâmetro no *URL*, que é nome e a subcategoria seja igual ao parâmetro do URL subCategoria. |

*Tabela 24: Estrutura do endereçamento '/add-Carrinho/:userID/:produtoID' do ficheiro produtos.js*

| **Método** | POST |
|---|---|
| **URL** | "/add-Carrinho/:userID/:produtoID" |
| **Middleware** | authenticateUser |
| **Função** | Responsável por adicionar o produto cujo ID vem como parâmetro no URL, que é produtoID, no carrinho de compras do utilizador cujo ID vem como parâmetro no URL, que é userID. O produto só é adicionado se a quantidade que o utilizador pretende adicionar for menor ou igual á quantidade de produtos existentes. |

*Tabela 25: Estrutura do endereçamento '/removerUm-Carrinho/:userID/:produtoID' do ficheiro produtos.js*

| **Método** | POST |
|---|---|
| **URL** | "/removerUm-Carrinho/:userID/:produtoID" |
| **Middleware** | authenticateUser |
| **Função** | Responsável por reduzir a quantidade do produto cujo ID vem como parâmetro no URL, que é produtoID, do carrinho de compras do utilizador cujo ID vem como parâmetro no *URL*, que é o userID. |

*Tabela 26: Estrutura do endereçamento '/remover-Carrinho/:userID/:produtoID' do ficheiro produtos.js*

| **Método** | POST |
|---|---|
| **URL** | "/remover-Carrinho/:userID/:produtoID" |
| **Middleware** | authenticateUser |
| **Função** | Responsável por remover o produto cujo ID vem como parâmetro no URL, que é o produtoID, do carrinho de compras do utilizador cujo ID vem como parâmetro no URL, que é userID. |

*Tabela 27: Estrutura do endereçamento '/checkout/cartaoCredito/:userID' do ficheiro produtos.js*

| **Método** | POST |
|---|---|
| **URL** | "/checkout/cartaoCredito/:userID" |
| **Middleware** | authenticateUser |
| **Função** | Responsável por efetuar a compra dos produtos existente no carrinho de compras através do cartão de crédito. A transação é realizada com o auxílio do módulo stripe. Após a transação ser efetuada com sucesso é feito uma encomenda, onde esta é guardada com o ID do utilizador que vem como parâmetro no URL, que é userid. E o corpo da encomenda contém os dados introduzidos pelo cliente, as informações do carrinho de compras e as informações da transação. E por último é enviado para o email do utilizador uma confirmação de que a compra foi efetuada com sucesso. |

*Tabela 28: Estrutura do endereçamento '/checkout/cobranca/:userID' do ficheiro produtos.js*

| **Método** | POST |
|---|---|
| **URL** | "/checkout/cobranca/:userID" |
| **Middleware** | authenticateUser |
| **Função** | Responsável por realizar a compra dos produtos existente no carrinho de compras, não é efetuada nenhuma transação. É simplesmente realizado uma encomenda, onde esta é guardada com o ID do utilizador que vem como parâmetro no URL, que é userid. E o corpo da encomenda contém as informações do cliente e do carrinho de compras. E por último é enviado para o email do utilizador uma confirmação que a compra foi efetuada com sucesso. |

*Tabela 29: Estrutura do endereçamento '/comentarios/adcionar/:produtoID/:userID' do ficheiro produtos.js*

| **Método** | POST |
|---|---|
| **URL** | "/comentarios/adcionar/:produtoID/:userID" |
| **Middleware** | authenticateUser |
| **Função** | Responsável por adicionar um comentário ao produto cujo ID vem como parâmetro no URL, que é produtoID, em que a informação do comentário vem no corpo do pedido e o ID do utilizador que introduziu o comentário vem como parâmetro, que é userID. |

- ***produtosBanner.js***

Este ficheiro de endereçamento de pedidos é responsável pelos *banners*. O URL associado a este ficheiro é o "**/produtosBanner**", em que este é definido no ficheiro app.js, que pode ser consultado na íntegra no Apêndice A.10 deste documento.

Em seguida vão ser mostrados todos os endereçamentos que estão no ficheiro produtosBanner.js.

*Tabela 30: Estrutura do endereçamento '/' do ficheiro produtosBanner.js*

| **Método** | GET |
|---|---|
| **URL** | "/" |
| **Middleware** | - |
| **Função** | Responsável por enviar para o cliente todos os *banners* existentes na base de dados. |

*Tabela 31: Estrutura do endereçamento '/criarImagem' do ficheiro produtosBanner.js*

| **Método** | POST |
|---|---|
| **URL** | "/criarImagem" |
| **Middleware** | autheticateUser, autenticacaoAdmin('admin'), upload.single('file') |
| **Função** | Responsável por criar um novo *banner*, onde a imagem é retirada do corpo do pedido através do *middleware* **upload.single('file')**. |

*Tabela 32: Estrutura do endereçamento '/remover/:bannerID' do ficheiro produtosBanner.js*

| **Método** | POST |
|---|---|
| **URL** | "/remover/:bannerID" |
| **Middleware** | autheticateUser, autenticacaoAdmin('admin') |
| **Função** | Responsável por remover o *banner* cujo ID vem como parâmetro no URL, que é bannerID. |

- ***utilizadores.js***

Este ficheiro de endereçamento de pedidos é responsável pelos utilizadores. O URL associado a este ficheiro é o "**/utilizadores**", em que este é definido no ficheiro app.js, que pode ser consultado na íntegra no Apêndice A.11 deste documento.

Em seguida vão ser mostrados todos os endereçamentos que estão no ficheiro utilizadores.js.

*Tabela 33: Estrutura do endereçamento '/' do ficheiro utilizadores.js*

| **Método** | GET |
|---|---|
| **URL** | "/" |
| **Middleware** | autheticateUser, autenticacaoAdmin('admin') |
| **Função** | Responsável por enviar para o cliente todos os utilizadores existentes na base de dados. |

*Tabela 34: Estrutura do endereçamento '/registar' do ficheiro utilizadores.js*

| **Método** | GET |
|---|---|
| **URL** | "/registar" |
| **Middleware** | - |
| **Função** | Responsável por criar um novo utilizador na base de dados. |

*Tabela 35: Estrutura do endereçamento '/login' do ficheiro utilizadores.js*

| **Método** | GET |
|---|---|
| **URL** | "/login" |
| **Middleware** | passport.authenticate('local-login', {session: false}) |
| **Função** | Responsável por realizar o login do utilizador através módulo passport, onde este usa o *middleware* criado anteriormente que é o "**local-login**". |

*Tabela 36: Estrutura do endereçamento '/logout' do ficheiro utilizadores.js*

| **Método** | GET |
|---|---|
| **URL** | "/logout" |
| **Middleware** | - |
| **Função** | Responsável por enviar para o cliente uma mensagem a dizer "Logout". |

*Tabela 37: Estrutura do endereçamento '/loggedin' do ficheiro utilizadores.js*

| **Método** | GET |
|---|---|
| **URL** | "/loggedin" |
| **Middleware** | authenticateUser |
| **Função** | Responsável por verificar se o utilizador está autenticado ou não. |

*Tabela 38: Estrutura do endereçamento '/profile' do ficheiro utilizadores.js*

| **Método** | POST |
|---|---|
| **URL** | "/profile" |
| **Middleware** | authenticateUser |
| **Função** | Responsável por alterar o nome do utilizador cuja informação vem no corpo do pedido. |

*Tabela 39: Estrutura do endereçamento '/profile/:id' do ficheiro utilizadores.js*

| **Método** | POST |
|---|---|
| **URL** | "/profile/:id" |
| **Middleware** | authenticateUser |
| **Função** | Responsável por alterar o email do utilizador cujo ID vem como parâmetro no URL, que é id, mas este só é alterado se o email que pretende ainda não esteja em uso. |

*Tabela 40: Estrutura do endereçamento '/profile/morada/:id' do ficheiro utilizadores.js*

| **Método** | POST |
|---|---|
| **URL** | "/profile/morada/:id" |
| **Middleware** | authenticateUser |
| **Função** | Responsável por alterar a morada do utilizador cujo ID vem como parâmetro no URL, que é id. |

*Tabela 41: Estrutura do endereçamento '/profile/imagePerfil/:id' do ficheiro utilizadores.js*

| **Método** | POST |
|---|---|
| **URL** | "/profile/imagePerfil/:id" |
| **Middleware** | authenticateUser, upload.single('file') |
| **Função** | Responsável por alterar a imagem do utilizador cujo ID vem como parâmetro no URL, que é id. A imagem vem no corpo do pedido. |

*Tabela 42: Estrutura do endereçamento '/addFavoritos/:userID/:productID' do ficheiro utilizadores.js*

| **Método** | POST |
|---|---|
| **URL** | "/addFavoritos/:userID/:productID" |
| **Middleware** | authenticateUser |
| **Função** | Responsável por adicionar o produto cujo ID vem como parâmetro no URL, que é produtoID, nos favoritos do utilizador cujo ID vem como parâmetro no URL, que é userID. |

*Tabela 43: Estrutura do endereçamento '/removerFavoritos/:userID/:productID' do ficheiro utilizadores.js*

| **Método** | POST |
|---|---|
| **URL** | "/removerFavoritos/:userID/:productID" |
| **Middleware** | authenticateUser |
| **Função** | Responsável por remover o produto cujo ID vem como parâmetro no URL, que é produtoID, dos favoritos do utilizador cujo ID vem como parâmetro no URL, que é userID. |

*Tabela 44: Estrutura do endereçamento '/favoritos/:userID' do ficheiro utilizadores.js*

| **Método** | GET |
|---|---|
| **URL** | "/favoritos/:userID" |
| **Middleware** | authenticateUser |
| **Função** | Responsável por enviar os produtos que o utilizador cujo ID vem como parâmetro no URL, que é userID, tem nos favoritos. |

### *4.5.2)* Cliente

Para o lado do cliente, a *framework* usada é o AngularJS. E o primeiro passo ao realizar uma aplicação em AngularJS é criar o ficheiro responsável por inicializar a *framework*, configurar os endereçamentos que neste caso é feito através de estados e o ficheiro HTML base.

A pasta public é a pasta que contém todos os ficheiros e módulos necessários para o lado do cliente, por isso todas as pastas que forem mencionadas neste capitulo vão ser subpastas da pasta public.

Os ficheiros base necessários para o funcionamento da aplicação do lado do cliente é o ficheiro de HTML index.html e o ficheiro de JavaScript app.js. Antes de mostrar e explicar estes dois ficheiros, vai ser explicado os filtros e as *factories* que foram criadas.

#### *4.5.2.1) Filtros*

Filtros como o nome indica servem para filtrar algo, que neste caso é usado para filtrar um *array*, em que este seleciona um conjunto de elementos dentro desse *array* para serem retornado como um novo *array*. Os filtros estão disponíveis no ficheiro filtros.js que está na pasta filtros, que pode ser consultado na íntegra no Apêndice A.12 deste documento.

Os filtros criados foram:

- ***startFrom***

```
1.  (…)
2.  app.filter('startFrom', function () {
3.      return function (input, start) {
4.          return input.slice(start);
5.      };
6.  });
7.  (…)
```

O filtro **startFrom** recebe como argumentos o input e o start, em que o input é um *array* e o start é um número. O método *slice* seleciona os elementos a partir de uma certa posição definida pelo start, e retorna esses elementos num novo *array*.

- ***FiltroMarca***

```
1.  (…)
2.  app.filter('FiltroMarca', function () {
3.      return function (produtos, elemento) {
4.          return produtos.filter(function (produto) {
5.              for (var i = 0; i < elemento.length; i++) {
6.                  // se o elemento(marca) selecionda for igual a marca do produto ent
    ao ele devolve o produto
7.                  if (elemento[i].indexOf(produto.marca) != -1) {
8.                      return true;
9.                  }
10.             }
11.             // se não tiver nada selecionado mostra tudo
12.             if (elemento.length == 0) {
13.                 return true;
14.             }
15.             return false;
16.         });
17.     };
18. });
19. (…)
```

O filtro **FiltroMarca** recebe como argumentos produtos e elemento, em que produtos é um *array* e elemento é uma *string*. Este filtro vai fazer retorno dos elementos do *array* produtos que tenham o campo marca igual à *string* passada no argumento elemento. E se não existir nenhuma *string* no argumento elemento este retorna todos os elementos do *array*.

- ***FiltroPreco***

```
1.  (…)
2.  app.filter('FiltroPreco', function () {
3.      return function (produtos, ele) {
4.          return produtos.filter(function (produto) {
5.              // se o produto tiver um preco entre o ele.Max e o ele.Min entao retorn
    a true
6.              if (produto.preco <= ele.max && produto.preco >= ele.min) {
7.                  return true;
8.              }
9.          });
10.     }
11. });
12. (…)
```

O filtro **FiltroPreco** recebe como argumento produtos e ele, em que produtos é um *array* e ele é um objeto com dois campos, um é o campo max e o outro é o campo min em que ambos contêm números. Este filtro faz retorno dos elementos do *array* produtos que tenham o campo preço entre max e min.

### *4.5.2.2) Factory*

Uma *factory* é uma função que retorna um objeto. Esta pode ser acedida em qualquer parte da aplicação desde que o ficheiro onde estas funções estão esteja referenciado no ficheiro HTML base, que neste caso é o index.html.

As *factories* criadas têm funções que são usadas mais do que uma vez ao longo da aplicação, assim ao ter essas funções num só ficheiro faz com que se possa reutilizar essas funções através de uma instância, reduzindo o código noutros ficheiros. O ficheiro que contém estas funções está na pasta factories, em que o nome do ficheiro é factories.js, que pode ser consultado na íntegra no Apêndice A.13 deste documento.

Para a aplicação foram criadas seis *factories*, todas dentro do mesmo ficheiro, e cada *factory* tem as suas próprias funções. As *factories* criadas foram:

- ***Categorias***

```
(...)
    .factory('Categorias', ['$http', function ($http) {
        var Categorias = {};
        Categorias.BreadCrumb = function (produtoId) {
            return $http({
                method: 'GET',
                url: '/produtos/detalhes/' + produtoId
            }).then(function (response) {
                return response.data;
            });
        };
        Categorias.get = function () {
            return $http({
                method: 'GET',
                url: 'categorias/'
            });
        };
        return Categorias;
    }])
(...)
```

Esta *factory* chama-se Categorias e contém duas funções dentro dela. As funções são **BreadCrumb** e **get**. A função **BreadCrumb** retorna a informação proveniente do servidor sobre o produto cujo ID foi passado como parâmetro no URL, que é produtoId e a função **get** retorna todas as categorias existente na base de dados.

- ***Favoritos***

```
(...)
    .factory('Favoritos', ['$http', function ($http) {
        var Favoritos = {};
        Favoritos.get = function (user) {
            return $http({
                method: 'GET',
                url: 'utilizadores/favoritos/' + user
            });
        };
        Favoritos.add = function (user, produtoId) {
            return $http({
                method: 'POST',
                url: 'utilizadores/addFavoritos/' + user + '/' + produtoId
            }).then(function successCallback(response) {
                },
                function errorCallback(response) {
                    console.log(response);
                });
        };
        return Favoritos;
    }])
(...)
```

Esta *factory* chama-se **Favoritos** e contém duas funções dentro dela. As funções são **get** e **add**. A função **get** retorna todos os produtos que o utilizador tem nos favoritos e a função

**add** é utilizada para adicionar o produto passado como parâmetro no URL aos favoritos do utilizador que também é passado como parâmetro.

- ***Encomendas***

```
40. (...)
41.     .factory('Encomendas', ['$http', function ($http) {
42.         var Encomendas = {};
43.         Encomendas.get = function (user) {
44.             return $http({
45.                 method: 'GET',
46.                 url: 'encomendas/' + user
47.             })
48.         };
49.         return Encomendas;
50.     }])
51. (...)
```

Esta *factory* chama-se **Encomendas** e contém a função **get** dentro dela, em que esta retorna todas as encomendas que estejam associadas ao utilizador que é passado como parâmetro no URL.

- ***Carrinho***

```
50. (...)
51.     .factory('Carrinho', ['$http', '$rootScope', '$localStorage', function ($http,
    $rootScope, $localStorage) {
52.         var Carrinho = {};
53.         Carrinho.add = function (user, produtoId, carrinho, qty) {
54.             return $http({
55.                 method: 'POST',
56.                 url: 'produtos/add-Carrinho/' + user + '/' + produtoId,
57.                 data: {
58.                     'carrinho': carrinho,
59.                     'qty': qty
60.                 }
61.             }).then(function successCallback(response) {
62.                     if (response.data.errorMessage) {
63.                         alert(response.data.errorMessage);
64.                     } else {
65.                         $localStorage.currentUser.carrinho = response.data;
66.                         $rootScope.currentUser.carrinho = response.data;
67.                     }
68.                 },
69.                 function errorCallback(response) {
70.                     console.log(response);
71.                 });
72.         };
73.         Carrinho.remover = function (url, user, produtoId, carrinho) {
74.             return $http({
75.                 method: 'POST',
76.                 url: url + user + '/' + produtoId,
77.                 data: {
78.                     'carrinho': carrinho
```

```
79.                 }
80.             }).then(function successCallback(response) {
81.                     if (response.data.errorMessage) {
82.                         alert(response.data.errorMessage);
83.                     }
84.                     $localStorage.currentUser.carrinho = response.data;
85.                     $rootScope.currentUser.carrinho = response.data;
86.                 },
87.                 function errorCallback(response) {
88.                     console.log(response);
89.                 });
90.         };
91.         return Carrinho;
92.     }])
93. (…)
```

Esta *factory* chama-se **Carrinho** e contém duas funções dentro dela. As funções são **add** e **remover**. A função **add** é utilizada para adicionar o produto passado como parâmetro no URL ao carrinho de compras do utilizador que também é passado como parâmetro, em que o carrinho de compras é enviado no corpo do pedido, e ao receber os dados vindos do servidor este guarda o carrinho de compras atualizado no $localStorage e no $rootScope, ou se ocorrer algum erro faz *alert* desse mesmo erro. A função **remover** é utilizada para remover o produto passado como parâmetro no URL do carrinho de compras do utilizador também passado como parâmetro, em que o carrinho de compras é enviado no corpo do pedido. Ao receber os dados vindos do servidor este guarda o carrinho de compras atualizado no $localStorage e no $rootScope ou se ocorrer algum erro faz *alert* desse mesmo erro.

A variável **$localStorage** é a variável responsável por conter todos os dados do utilizador que esteja autenticado, em que estes são guardados no armazenamento local do browser. E a variável **$rootScope** é a variável global responsável por conter todos os dados do utilizador que esteja autenticado, mas esta é usada para mostrar esses dados nas *views*.

- ***Morada***

```
92. (...)
93.     .factory('Morada', ['$http', '$localStorage', '$rootScope', function ($http, $l
    ocalStorage, $rootScope) {
94.         var Morada = {};
95.         Morada.update = function (user, morada) {
96.             return $http({
97.                 method: 'POST',
98.                 url: 'utilizadores/profile/morada/' + user,
99.                 data: morada
100.                }).then(function successCallback(response) {
101.                    $rootScope.currentUser.morada = response.data.morada;
102.                    $localStorage.currentUser.morada = response.data.morada;
103.                }, function errorCallback(response) {
104.                    console.log(response);
105.                });
```

```
106.                };
107.                return Morada;
108.            }])
109.        (...)
```

Esta *factory* chama-se **Morada** e contém a função **update** dentro dela que é utilizada para alterar a morada do utilizador que vem como parâmetro no URL, em que a morada é enviada no corpo do pedido. Ao receber os dados vindos do servidor este guarda a morada atualizado $localStorage e no $rootScope ou se ocorrer algum erro faz *alert* desse mesmo erro.

- ***LogOut***

```
108.        (...)
109.            .factory('LogOut', ['$http', '$rootScope', '$state', '$localStorage', fu
    nction ($http, $rootScope, $state, $localStorage) {
110.                var LogOut = {};
111.                LogOut.logout = function () {
112.                    return $http({
113.                        method: 'POST',
114.                        url: 'utilizadores/logout'
115.                    }).then(function successCallback(response) {
116.                        delete $localStorage.currentUser;
117.                        $http.defaults.headers.common.Authorization = '';
118.                        $rootScope.currentUser = null;
119.                        $state.go('root.home', {
120.                            reload: true
121.                        });
122.                    }, function errorCallback(response) {
123.                        console.log("Failed logout");
124.                    });
125.                };
126.                return LogOut;
127.            }])
```

Esta *factory* chama-se **LogOut** e contém a função **LogOut** dentro dela, que é utilizada para fazer o *logout* do utilizador. Ao receber uma resposta do servidor, esta apaga todos os dados existentes no $localStorage e do $rootScope, e muda para o estado para "**root.home**" que representa a página principal da aplicação.

#### *4.5.2.3) Ficheiros base para o cliente*

Como foi mencionado no início deste capítulo os ficheiros base para o funcionamento do AngularJS são o index.html e o app.js. O index.html é a única página da aplicação, ou seja, o resto dos ficheiros HTML contêm conteúdos específicos que vão ser carregados para dentro de um div do index.html dependendo do estado, este também é responsável por chamar os módulos, controladores, *factories* e filtros que vão ser usados pela aplicação. O app.js é o

ficheiro JavaScript responsável por inicializar o AngularJS, definir os estados, os controladores, as *views* associadas aos estados, e inicializar os módulos.

Todas as variáveis **$scope** que estão são usadas nos controladores são acedíveis no HTML associado a esse controlador. Em que estas variáveis são usadas para passar dados para o HTML.

- ***index.html***

```
1. <!DOCTYPE html>
2. <html ng-app="projetoGlobal" ng-cloak>
3.
4. <head>
5.     <title>Projeto Global</title>
6.     <meta charset="UTF-8">
7.     <meta name="viewport" content="width=device-width, initial-scale=1">
8.     <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-
   awesome/4.7.0/css/font-awesome.min.css">
9.     <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/cs
   s/bootstrap.min.css" integrity="sha384-
   BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anon
   ymous">
10.    <link rel="stylesheet" type="text/css" href="bower_components/angular-
   rangeslider/angular.rangeSlider.css"/>
11.    <link href="stylesheets/style.css" rel="stylesheet">
12. </head>
13.
14. <body>
15. <header ui-view='header'
   onload="categoriaSelecionada = 'Todas as Categorias'"></header>
16.
17. <div class="container">
18.     <div ui-view='container'></div>
19. </div>
20.
21. <footer ui-view='footer'></footer>
22.
23. <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js "></
   script>
24. <script src="bower_components/angular/angular.min.js"></script>
25. <script src="bower_components/angular-ui-router/release/angular-ui-
   router.min.js"></script>
26. <script src="bower_components/angular-
   rangeslider/angular.rangeSlider.js"></script>
27. <script src="bower_components/angular-utils-ui-
   breadcrumbs/uiBreadcrumbs.js"></script>
28. <script src="bower_components/angular-sanitize/angular-sanitize.min.js"></script>
29. <script src="bower_components/ngstorage/ngStorage.min.js"></script>
30. <script src="bower_components/angular-jwt/dist/angular-jwt.min.js"></script>
31. <script src="bower_components/ng-file-upload/ng-file-upload-
   shim.min.js"></script>
32. <script src="bower_components/ng-file-upload/ng-file-upload.min.js"></script>
33. <script src="bower_components/stripe-angular/stripe-angular.js"></script>
34. <script src="bower_components/angular-ckeditor/angular-ckeditor.min.js"></script>
35. <script src="//cdn.ckeditor.com/4.6.2/standard/ckeditor.js"></script>
36. <script src="bower_components/ui-bootstrap/ui-bootstrap-tpls-
   2.3.0.min.js"></script>
```

```
37. <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js "
    integrity="sha384-
    Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa " crossorigin="ano
    nymous "></script>
38. <script type="text/JavaScript" src="https://js.stripe.com/v2/"></script>
39. <script src="app.js"></script>
40. <script src="controllers/home-controller.js"></script>
41. <script src="controllers/header-controller.js"></script>
42. <script src="controllers/produtoDetalhes-controller.js"></script>
43. <script src="controllers/procurarProdutos-controller.js"></script>
44. <script src="controllers/login-controller.js"></script>
45. <script src="controllers/profile-controller.js"></script>
46. <script src="controllers/cart-controller.js"></script>
47. <script src="controllers/checkout-controller.js"></script>
48. <script src="controllers/checkoutResumo-controller.js"></script>
49. <script src="controllers/encomendasDetalhes-controller.js"></script>
50. <script src="controllers/admin-controller.js"></script>
51. <script src="controllers/adminCriarProduto-controller.js"></script>
52. <script src="controllers/adminEditarProduto-controller.js"></script>
53. <script src="controllers/adminEncomendaDetalhes-controller.js"></script>
54. <script src="factories/factories.js"></script>
55. <script src="filtros/filtros.js"></script>
56. </body>
57. </html>
```

O código acima representa o index.html e para uma melhor compreensão do seu funcionamento este vai ser descrito em pormenor.

- **Linha 1 – 2:** Define que tipo de documento é que vai ser utilizado e é definido o nome da aplicação que o AngularJS vai usar.
- **Linha 4 – 12:** O *head* contém o título da aplicação, e faz os importes dos ficheiros de CSS do B*ootstrap* através de uma CDN, do *rangeSlider* que é um módulo usado na aplicação e do CSS principal.
- **Linha 15:** O *header* vai conter a *view* "**header**" e inicializa a variável **categoriaSelecionada** que é uma variável da *view* "**header**" com o valor "Todas as Categorias". em que a *view* é definida no app.js. Esta *view* corresponde ao cabeçalho.
- **Linha 17 – 19:** O div vai conter as *views* que serão associadas à *view container*, em que estas vão depender do estado em que a aplicação esteja.
- **Linha 21:** O *footer* vai conter a *view* "**footer**", em que esta corresponde ao rodapé.
- **Linha 23:** Faz o importe dos *scripts* que vão ser usados pela aplicação. Estes são módulos, controladores, *factories*, filtros e o app.js que é o ficheiro JavaScript base.

- ***app.js***

O ficheiro app.js está divido em três partes, uma para inicializar o AngularJS e os módulos que vão ser usados na aplicação, outra para fazer a configuração do cliente e por último o *run* que é uma função do AngularJS que corre ao se iniciar a aplicação. O ficheiro pode ser consultado na íntegra no Apêndice A.14 deste documento.

Em seguida, vai ser explicado em pormenor as três partes e o código associado.

1. Inicialização do AngularJS

```
1. var app = angular.module('projetoGlobal', ['ui.router', 'ui.bootstrap', 'ui-
   rangeSlider', 'angularUtils.directives.uiBreadcrumbs', 'ngSanitize', 'ngStorage', '
   angular-jwt', 'ngFileUpload', 'stripe', 'ckeditor']);
```

O código acima representa a inicialização do AngularJS chamando o nome que foi definido para a aplicação no index.html que foi "projetoGlobal" e dos módulos que vão ser usados na aplicação, em que é posto entre aspas o nome que foi atribuído ao módulo pelos seus autores.

2. Configuração do cliente

Na configuração do cliente, é feita a configuração do Stripe que é o módulo responsável por realizar o pagamento através do cartão de crédito e é definido os estados da aplicação. A definição dos estados vai ser realizado com o auxílio de tabelas para uma melhor compreensão. Mas primeiro vai ser explicado o estado base ou principal que é o estado que vai conter todos os outros, este não tem nenhum URL associado.

```
1.  .state('root', {
2.      url: '',
3.      abstract: true,
4.      views: {
5.          'header': {
6.              templateUrl: 'partials/header.html',
7.              controller: 'HeaderCtrl'
8.          },
9.          'footer': {
10.             templateUrl: 'partials/footer.html'
11.         }
12.     }
13. })
```

O código acima representa o estado base, que começa com a definição do nome do estado que é "**root**", depois este é definido como ***abstract*** porque não vai ter nenhum URL. As

*views* associadas ao estado, são o *header* e o *footer*. Em que estas *view* vão ser depois associadas ás *tags header* e *footer* do index.html.

Os restantes estados são filhos do estado base, para que estes possam usar as *views* associadas ao pai que são o *header* e o *footer*. É definido nos estados filhos, uma variável com o nome data que vai ser usada pelo módulo *breadcrumb*, estes podem também ter *middlewares*, em que o *middleware* **autheticate:true** verifica se o utilizador está autenticado ou não e o *middleware* **admin:true** verifica se o utilizador tem o campo role igual a **admin**, se estes *middlewares* não retornarem verdade então o utilizador não pode entrar no estado.

*Tabela 45: Estrutura do estado 'root.home'*

| **estado** | | root.home |
|---|---|---|
| **url** | | "/" |
| **views** | **templateUrl** | "views/home/home.view.html" |
| | **controller** | HomeCtrl |
| **data** | | "HomePage" |
| **middleware** | | - |

*Tabela 46: Estrutura do estado 'root.home.produtos'*

| **estado** | | root.home.produtos |
|---|---|---|
| **url** | | "^/produtos/categorias/{categoria}/{nomeProduto}" |
| **views** | **templateUrl** | "views/produtosSearch/procurarProdutos.view.html" |
| | **controller** | ProcurarProdutosCtrl |
| **data** | | "{{id}}" (categoria selecionada) |
| **middleware** | | - |

*Tabela 47: Estrutura do estado 'root.home.produtos.subcategoria'*

| **estado** | | root.home.produtos.subcategoria |
|---|---|---|
| **url** | | "^/subcategorias/{subcategoria}/{nomeProduto}" |
| **views** | **templateUrl** | "views/produtosSearch/procurarProdutos.view.html" |
| | **controller** | ProcurarProdutosCtrl |
| **data** | | "{{id}}" (subcategoria selecionada) |
| **middleware** | | - |

*Tabela 48: Estrutura do estado 'root.home.produtoDetalhes'*

| estado | | root.home.produtoDetalhes |
|---|---|---|
| url | | "^/produto/detalhes/{produtoId}" |
| views | templateUrl | "views/produtoDetalhes/produtoDetalhes.view.html" |
| | controller | ProdutoDetalhesCtrl |
| data | | "{{id.produto.categoria.nome}}" (categoria do produto) |
| middleware | | - |

*Tabela 49: Estrutura do estado 'root.profile'*

| estado | | root.profile |
|---|---|---|
| url | | "/profile" |
| views | templateUrl | "views/profile/profile.view.html" |
| | controller | ProfileCtrl |
| data | | - |
| middleware | | authenticate: true |

*Tabela 50: Estrutura do estado 'root.login'*

| estado | | root.login |
|---|---|---|
| url | | "/login" |
| views | templateUrl | "views/login/login.view.html" |
| | controller | LoginCtrl |
| data | | - |
| middleware | | - |

*Tabela 51: Estrutura do estado 'root.cart'*

| estado | | root.cart |
|---|---|---|
| url | | "/cart" |
| views | templateUrl | "views/cart/cart.view.html" |
| | controller | CartCtrl |
| data | | - |
| middleware | | authenticate: true |

*Tabela 52: Estrutura do estado 'root.checkout'*

| estado | | root.checkout |
|---|---|---|
| url | | "/checkout" |
| views | templateUrl | "views/checkout/checkout.view.html" |
| | controller | CheckoutCtrl |
| data | | - |
| middleware | | authenticate: true |

*Tabela 53: Estrutura do estado 'root.checkoutResumo'*

| estado | | root.checkoutResumo |
|---|---|---|
| url | | "/checkoutResumo/{encomendaId}" |
| views | templateUrl | "views/checkout/checkoutResumo.view.html" |
| | controller | checkoutResumoCtrl |
| data | | - |
| middleware | | authenticate: true |

*Tabela 54: Estrutura do estado 'root.encomendasDetalhes'*

| estado | | root.encomendasDetalhes |
|---|---|---|
| url | | "/encomendasDetalhes/{encomendaId}" |
| views | templateUrl | "views/encomendasDetalhes/encomendasDetalhes.view.html" |
| | controller | EncomendasDetalhesCtrl |
| data | | - |
| middleware | | authenticate: true |

*Tabela 55: Estrutura do estado 'root.admin'*

| estado | | root.admin |
|---|---|---|
| url | | "/admin" |
| views | templateUrl | "views/admin/admin.view.html" |
| | controller | AdminCtrl |
| data | | - |
| middleware | | admin: true |

*Tabela 56: Estrutura do estado 'root.admin.criarProduto'*

| | | |
|---|---|---|
| **estado** | | root.admin.criarProduto |
| **url** | | "/admin/criarProduto" |
| **views** | **templateUrl** | "views/admin/adminCriarProduto.view.html" |
| | **controller** | AdminCriarProdutoCtrl |
| **data** | | - |
| **middleware** | | admin: true |

*Tabela 57: Estrutura do estado 'root.admin.editarProduto'*

| | | |
|---|---|---|
| **estado** | | root.admin.editarProduto |
| **url** | | "/admin/editarProduto/{produtoID}" |
| **views** | **templateUrl** | "views/admin/adminEditarProduto.view.html" |
| | **controller** | AdminEditarProdutoCtrl |
| **data** | | - |
| **middleware** | | admin: true |

*Tabela 58: Estrutura do estado 'root.admin.encomendaDetalhes'*

| | | |
|---|---|---|
| **estado** | | root.admin.encomendaDetallhes |
| **url** | | "/admin/encomendaDetalhes/{encomendaID}" |
| **views** | **templateUrl** | "views/admin/adminEncomendaDetalhes.view.html" |
| | **controller** | AdminEncomendaDetalhesCtrl |
| **data** | | - |
| **middleware** | | admin: true |

*Tabela 59: Estrutura do estado 'root.about*

| | | |
|---|---|---|
| **estado** | | root.about |
| **url** | | "/about" |
| **views** | **templateUrl** | "views/about/about.view.html" |
| | **controller** | - |
| **data** | | - |
| **middleware** | | - |

*Tabela 60: Estrutura do estado 'root.contatos*

| estado | | root.contatos |
|---|---|---|
| url | | "/contatos" |
| views | templateUrl | "views/contatos/contatos.view.html" |
| | controller | - |
| data | | - |
| middleware | | - |

Se o utilizador tentar ir para um estado que não esteja nas tabelas acima, este é redirecionado para estado "**root.home**" que representa a página principal.

*3. run*

No *run* é onde estão os *middlewares* usados nos estados. Estes estão dentro de uma função que é chamada sempre que o estado é alterado. Se o *middleware* authenticate não retornar verdade então é removido os dados contidos nas variáveis $localStorage e $rootScope e o utilizador é redirecionado para o estado "root.login". Se o *middleware* admin não retornar verdade o utilizador é redirecionado para o estado "root.login" se este não estiver autenticado ou é redirecionado para o estado "root.home" se este não tenha o campo role igual a admin.

Existe também uma função no *run* em que é chamada sempre que o utilizador faz *refresh* à pagina. Se o utilizador estiver autenticado as informações não são apagadas e mantêm-no autenticado.

#### *4.5.2.4) Páginas HTML dos estados*

Todas as páginas que irão ser abordadas a seguir contêm as *views header* e *footer*. O HTML das views pode ser consultado na íntegra no Apêndice A.15 e Apêndice A.16. deste documento. Em que estes contêm:

- ***header***
    a. Uma barra de pesquisas onde os utilizadores podem procurar o produto deseja, esta ainda contém um filtro, em que a categoria do produto que deseja pode ser especificada.

b. Botões que permitem:
   *i.* A utilizadores não autenticados ir para o estado do login.
   *ii.* A utilizadores autenticados aceder ao carrinho de compras, ao perfil e fazer o *logout*.
   *iii.* A utilizadores autenticados com o campo role igual a admin aceder ao painel de administrador.
   *iv.*

- ***Footer***
  a. *Links* para os contatos e para o sobre.
  b. Contatos da loja
  c. Informação sobre a empresa

De seguida vai ser explicado as funcionalidades dos estados acima demonstrados em tabelas, com respetivo *screenshot*. Como foi explicado em cima, todos os estados contêm as *views header* e o *footer*, por isso as funcionalidades associadas a estes não vão ser mencionados nos estados abaixo.

- **root.home**


*Figura 11: Estado root.home*

Este estado representa a página principal da aplicação. O controlador responsável por conter a lógica deste estado é o "HomeCtrl" que está contido no ficheiro home-controller.js que por sua vez este está na pasta controllers. O HTML deste estado e respetivo controlador pode ser consultado na íntegra no Apêndice A.17 e no Apêndice A.18 deste documento.

Neste estado é possível:

a. Visualizar os *banners*;
b. Visualizar os produtos em destaque, onde o utilizador pode selecionar o produto e vê-lo em detalhe;
c. Adicionar produtos aos favoritos;
d. Adicionar produtos ao carrinho de compras.

- **root.home.produtos**

*Figura 12: Estado root.home.produtos*

Este estado representa a página principal da aplicação, onde o utilizador pode procurar produtos. Em que este pode ser acedido através de uma procura na barra de pesquisa ou através do submenu Categorias ao selecionar uma categoria. O controlador responsável por conter a lógica deste estado é o "ProcurarProdutosCtrl" que está contido no ficheiro procurarProdutos-controller.js que por sua vez este está na pasta controllers. O HTML deste estado e respetivo controlador pode ser consultado na íntegra no Apêndice A.19 e no Apêndice A.20 deste documento.

Neste estado é possível:

a. Navegar para a página principal através do *breadcrumb*, que é um navegador estrutural;
b. Filtrar os produtos por subcategorias, marcas ou preços, onde estes são dinâmicos, ou seja, os seus valores vão depender dos produtos apresentados;
c. Visualizar os produtos filtrados, onde o utilizador pode selecionar o produto e vê-lo em detalhe;
d. Ordenar os produtos filtrados por Nome, Preço ou Marca, e por ordem crescente ou decrescente;
e. Adicionar produtos aos favoritos;
f. Adicionar produtos ao carrinho de compras.

- **root.home.produtos.subcategoria**

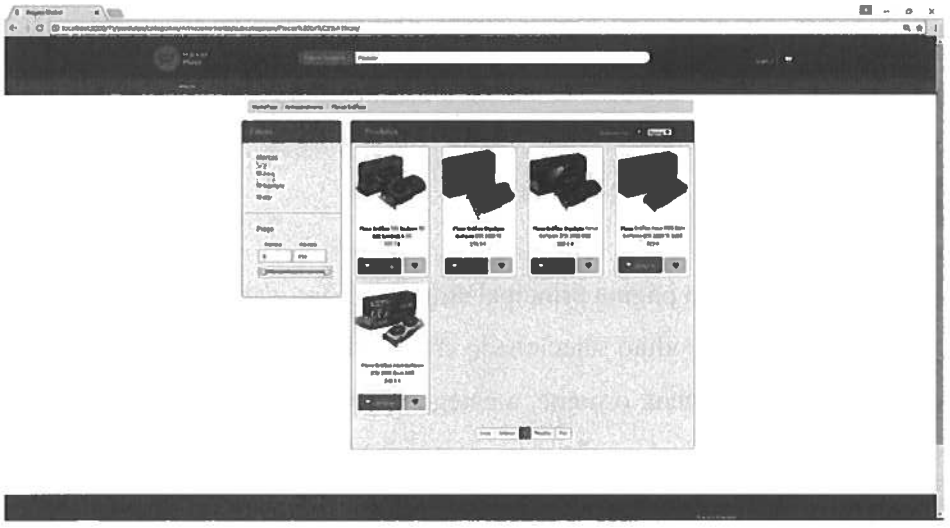

*Figura 13: Estado root.produtos.subcategorias*

Este estado representa a página onde o utilizador pode procurar produtos, este é igual ao estado root.home.produtos, mas este é usado quando o utilizador seleciona uma subcategoria e não uma categoria, logo tem as mesma funcionalidades exceto puder selecionar uma subcategoria. O controlador responsável por conter a lógica deste estado é o "ProcurarProdutosCtrl" que está contido no ficheiro procurarProdutos-controller.js que por sua vez este está na pasta controllers. O HTML deste estado e respetivo controlador pode ser consultado na íntegra no Apêndice A.19 e no Apêndice A.20 deste documento.

- **root.home.produtoDetalhes**

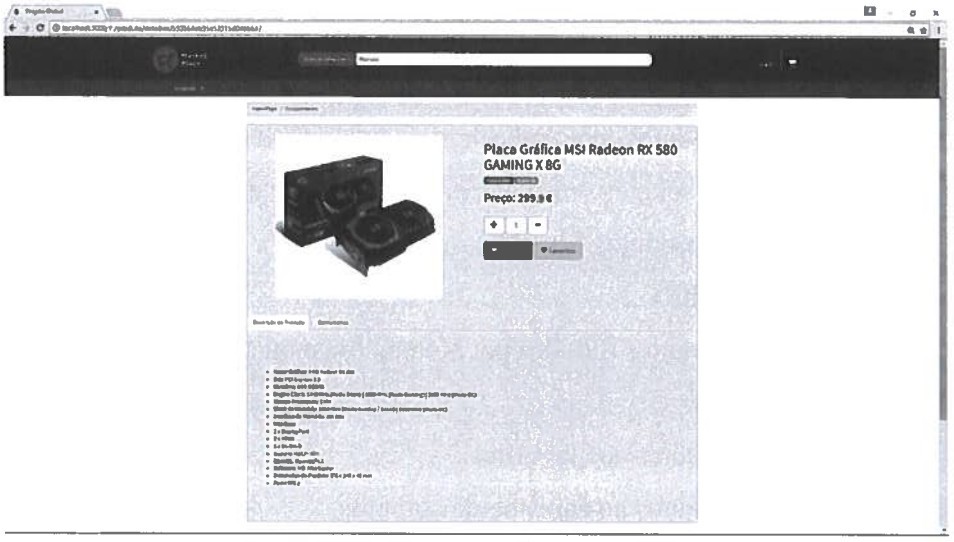


*Figura 14:Estado root.home.produto.Detalhes*

Este estado representa a página de detalhes do produto. O controlador responsável por conter a lógica deste estado é o "ProdutoDetalhesCtrl" que está contido no ficheiro produtoDetalhes-controller.js que por sua vez este está na pasta controllers. O HTML deste estado e respetivo controlador pode ser consultado na íntegra no Apêndice A.21 e no Apêndice A.22 deste documento.

Neste estado é possível:

a. Navegar para a página principal através do *breadcrumb*;
b. Visualizar o produto selecionado em detalhe, onde é possível ver:
    i. A imagem, o nome, a categoria, o preço do produto, se o produto está disponível ou não e a sua descrição.
c. O utilizador autenticado pode na aba comentários, deixar a sua crítica sobre o produto;
d. Adicionar o produto aos favoritos;
e. Adicionar a quantidade que pretender do produto ao carrinho de compras.

O estado do login consiste num div com duas abas, uma para realizar o login e outra para realizar o registo.

- **root.profile**

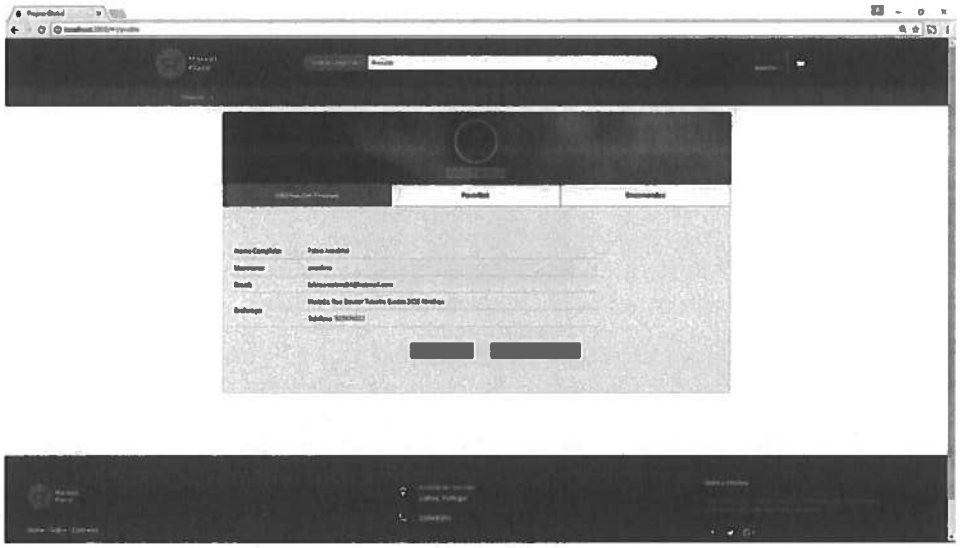

*Figura 15: Estado root.profile*

Este estado representa a página do login. O controlador responsável por conter a lógica deste estado é o "ProfileCtrl" que está contido no ficheiro profile-controller.js que por sua vez este está na pasta controllers. O HTML deste estado e respetivo controlador pode ser consultado na íntegra no Apêndice A.23 e no Apêndice A.24 deste documento.

Neste estado é possível:

a. Visualizar ou alterar as informações pessoais do utilizador na aba Informações Pessoais;
b. Visualizar ou remover os produtos dos favoritos na aba Favoritos;
c. Visualizar ou selecionar para ver em detalhe as encomendas realizadas na aba das Encomendas.

- **root.login**

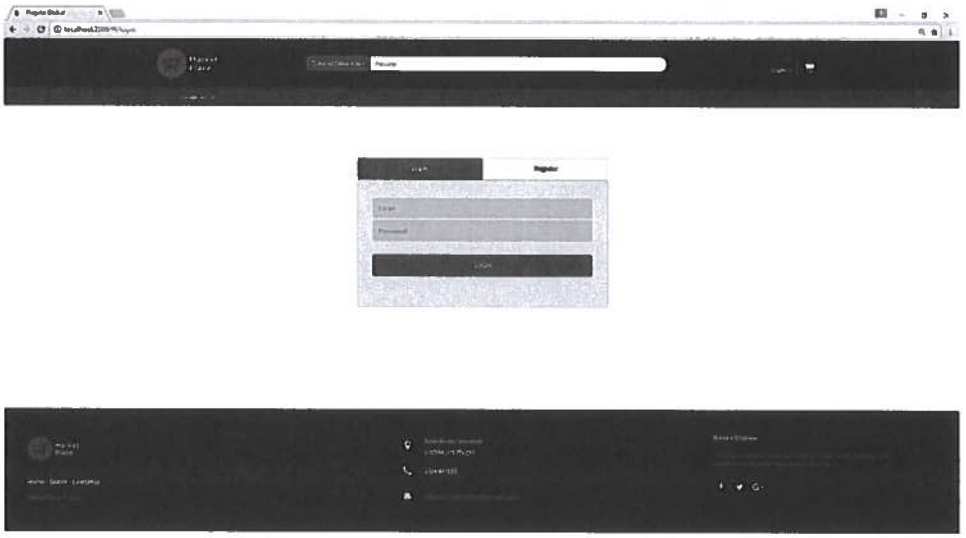

*Figura 16: Estado root.login*

Este estado representa a página do login. O controlador responsável por conter a lógica deste estado é o "LoginCtrl" que está contido no ficheiro login-controller.js que por sua vez este está na pasta controllers. O HTML deste estado e respetivo controlador pode ser consultado na íntegra no Apêndice A.25 e no Apêndice A.26 deste documento.

- **root.cart**

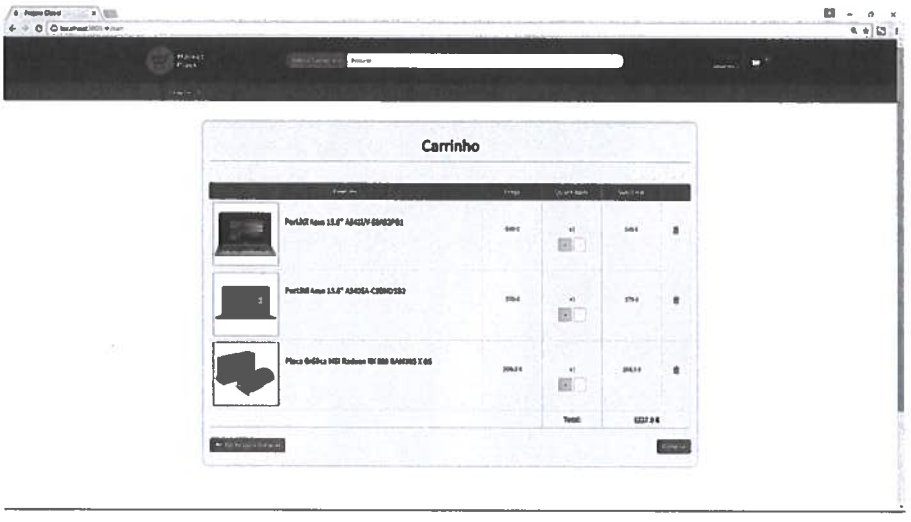


*Figura 17: Estado root.cart*

Este estado representa a página do carrinho de compras. O controlador responsável por conter a lógica deste estado é o "CartCtrl" que está contido no ficheiro cart-controller.js que

por sua vez este está na pasta controllers. O HTML deste estado e respetivo controlador pode ser consultado na íntegra no Apêndice A.27 e no Apêndice A.28 deste documento.

Neste estado é possível:

a. Visualizar os produtos adicionados no carrinho de compras;
b. Aumentar ou reduzir a quantidade dos produtos;
c. Remover os produtos;
d. Visualizar preço total;
e. Retornar para a página principal ao clicar no botão retornar ou avançar para a compra dos produtos no carrinho de compras ao clicar no botão comprar.
f.

- **root.checkout**

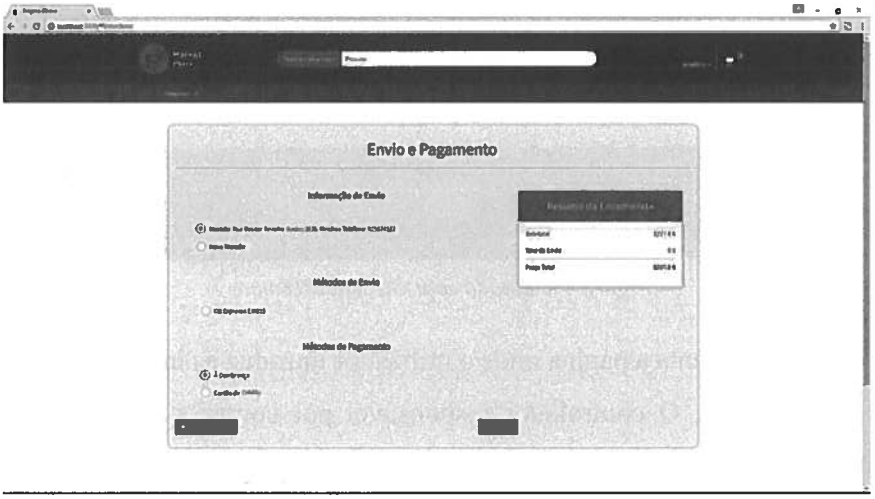


*Figura 18: Estado root.chekout*

Este estado representa a página onde o utilizador introduz as informações sobre o envio e o respetivo pagamento. O controlador responsável por conter a lógica deste estado é o "CheckoutCtrl" que está contido no ficheiro checkout-controller.js que por sua vez este está na pasta controllers. O HTML deste estado e respetivo controlador pode ser consultado na íntegra no Apêndice A.29 e no Apêndice A.30 deste documento

Neste estado é possível:

a. Escolher as informações de Envio, que pode ser a morada já definida no perfil ou uma morada nova;
b. Escolher o método de envio;

c. Escolher o método de pagamento, em que este pode ser à cobrança ou através de cartão de crédito;
d. Visualizar o resumo da compra no div lateral;
e. Retornar para a página do carrinho de compras ao clicar no botão Voltar Atrás ou efetuar a compra ao clicar no botão comprar.

- **root.checkoutResumo**


*Figura 19: Estado root.checkoutResumo*

Este estado representa a página onde o utilizador introduz as informações sobre o envio e o respetivo pagamento. O controlador responsável por conter a lógica deste estado é o "CheckoutResumoCtrl" que está contido no ficheiro checkoutResumo-controller.js que por sua vez este está na pasta controllers. O HTML deste estado e respetivo controlador pode ser consultado na íntegra no Apêndice A.31 e no Apêndice A.32 deste documento.

Após realizar a compra com sucesso o utilizador é redirecionado para este estado, onde é possível ver o número da encomenda. E é possível retornar à pagina principal ao clicar no botão Voltar para a Página Principal.

- **root.encomendasDetalhes**

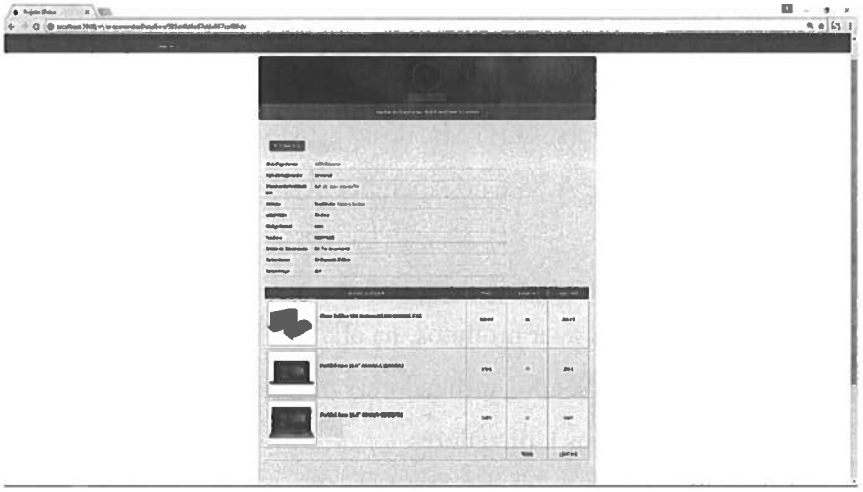

*Figura 20: Estado root.encomendaDetalhes*

Este estado representa a página onde o utilizador pode visualizar em detalhe a encomenda. Em que este pode ser acedido ao selecionar para ver uma encomendas em detalhes no perfil do utilizador na aba Encomendas. O controlador responsável por conter a lógica deste estado é o "EncomendasDetalhesCtrl" que está contido no ficheiro encomendasDetalhes-controller.js que por sua vez este está na pasta controllers. O HTML deste estado e respetivo controlador pode ser consultado na íntegra no Apêndice A.33 e no Apêndice A.34 deste documento.

Neste estado é possível ver todas as informações sobre a encomenda, desde informações sobre o envio, ás informações sobre o pagamento, ao ver os produtos que comprou.

- **root.admin**

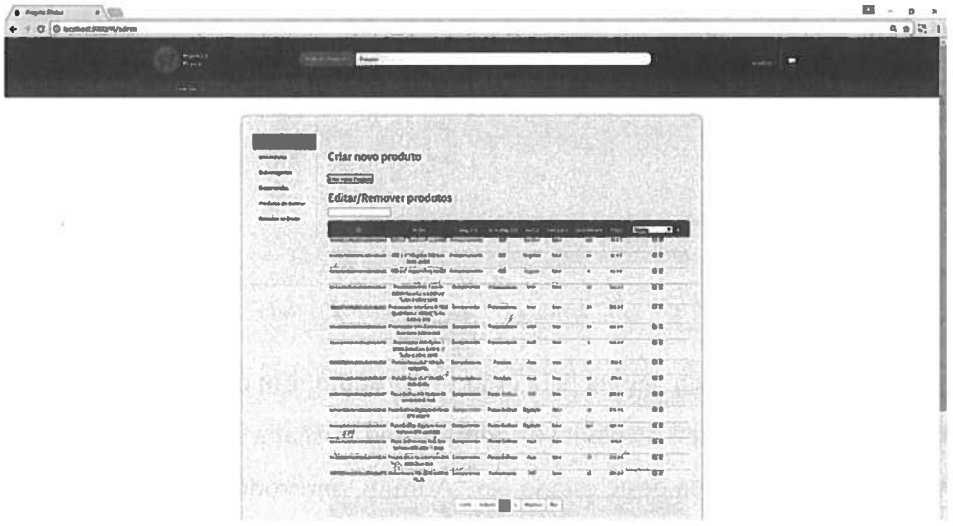


*Figura 21: Estado root.admin*

Este estado representa a página o painel de administrador. Em que este pode ser acedido por utilizadores que tenham o campo role igual a admin. O controlador responsável por conter a lógica deste estado é o "AdminCtrl" que está contido no ficheiro admin-controller.js que por sua vez este está na pasta controllers. O HTML deste estado e respetivo controlador pode ser consultado na íntegra no Apêndice A.35 e no Apêndice A.36 deste documento.

Neste estado é possível:

a. Criar, editar ou remover produtos, na aba dos Produtos;
b. Alterar as permissões dos utilizadores, ou seja, mudar o campo role para user ou admin, na aba dos Utilizadores;
c. Adicionar ou remover subcategorias, na aba das Subcategorias;
d. Selecionar uma encomenda para ir para o estado root.admin.encomendasDetalhes, para ver a encomenda em detalhe e poder alterar o seu estado;
e. Adicionar ou remover os *banners*;
f. Adicionar ou remover métodos de envio.

- **root.admin.criarProduto**

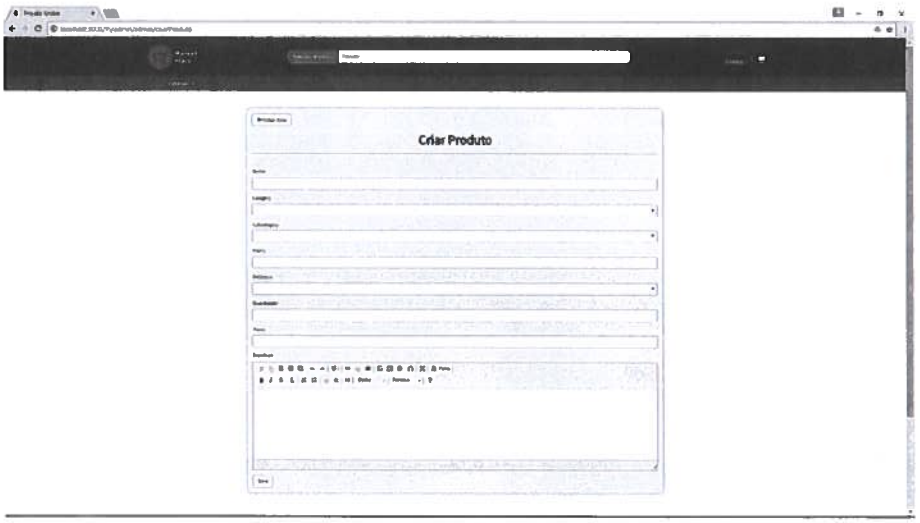


*Figura 22: Estado root.admin.criarProduto*

Este estado representa a página para criar um produto. Em que este pode ser acedido ao selecionar o botão criar produto no painel de administrador na aba dos Produtos. O controlador responsável por conter a lógica deste estado é o "AdminCriarProdutoCtrl" que está contido no ficheiro adminCriarProduto-controller.js que por sua vez este está na pasta controllers. O

HTML deste estado e respetivo controlador pode ser consultado na íntegra no Apêndice A.37 e no Apêndice A.38 deste documento.

Neste estado está um formulário com os dados necessários para criar um produto. As categorias que aparecem são as que estão na base de dados, e consoante a categoria selecionada aparece as subcategorias dessa mesma categoria. Na descrição do produto é usado o módulo ckeditor que possibilidade a introdução de código HTML.

- **root.admin.editarProduto**

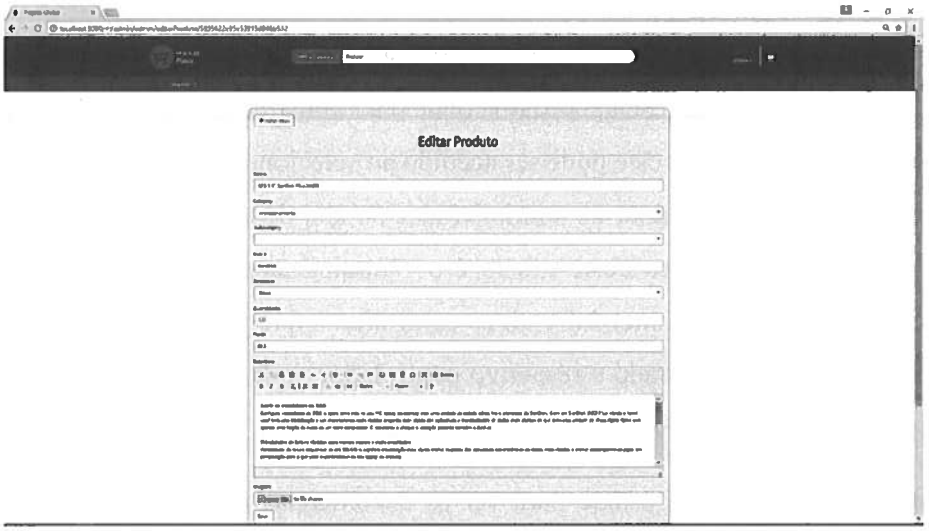


*Figura 23: Estado root.admin.editarProduto*

Este estado representa a página para editar um produto. Em que este pode ser acedido ao selecionar o botão para editar o produto no painel de administrador na aba dos Produtos. O controlador responsável por conter a lógica deste estado é o "AdminEditarProdutoCtrl" que está contido no ficheiro adminEditarProduto-controller.js que por sua vez este está na pasta controllers. O HTML deste estado e respetivo controlador pode ser consultado na íntegra no Apêndice A.39 e no Apêndice A.40 deste documento.

Este estado é semelhante ao estado para criar um produto, mas neste os campos são preenchidos consoante a informação do produto que foi selecionado e tem um campo adicional que para adicionar uma imagem ao produto.

- **root.admin.encomendaDetalhes**

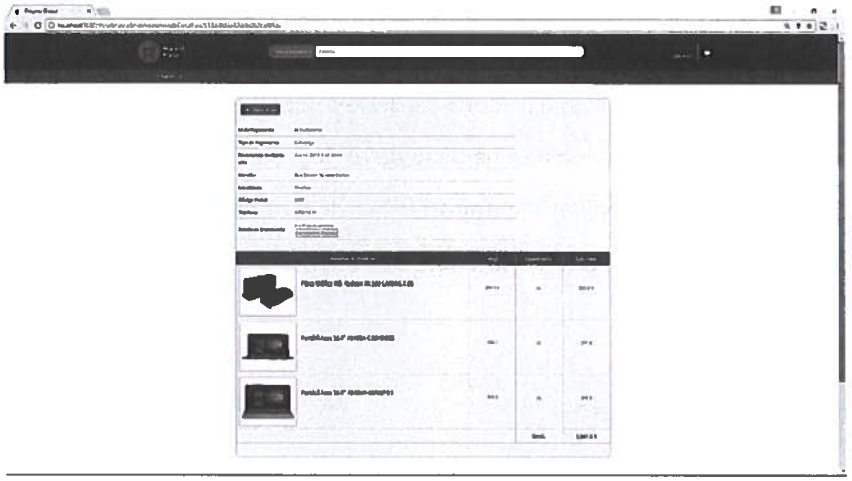

*Figura 24: Estado root.admin.encomendaDetalhes*

Este estado representa a página onde o administrador pode visualizar e alterar o estado da encomenda. Em que este pode ser acedido ao selecionar o botão Ver encomenda em Detalhes da encomenda no painel de administrador na aba dos Encomendas. O controlador responsável por conter a lógica deste estado é o "AdminEncomendaDetalhesCtrl" que está contido no ficheiro adminEncomendaDetalhes-controller.js que por sua vez este está na pasta controllers. O HTML deste estado e respetivo controlador pode ser consultado na íntegra no Apêndice A.41 e no Apêndice A.42 deste documento.

Este estado é igual ao estado para ver as encomendas em detalhes, mas este tem a possibilidade de alterar o estado da encomenda para Enviado ou Em Processamento.

- **root.about**

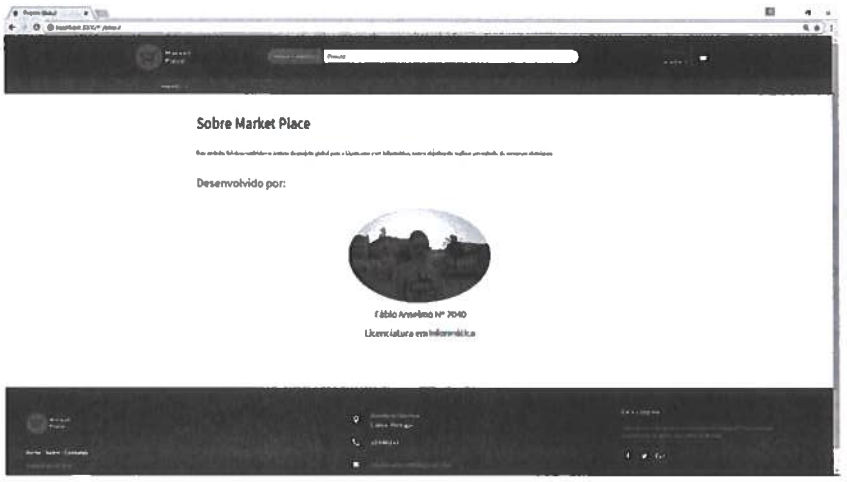


*Figura 25: Estado root.about*

Este estado representa a página que contém informações sobre o âmbito do projeto e do aluno que o desenvolveu. Não tenho nenhum controlador, porque este estado contém uma página estática. O HTML deste estado pode ser consultado na íntegra no Apêndice A.43 deste documento.

- **root.contatos**

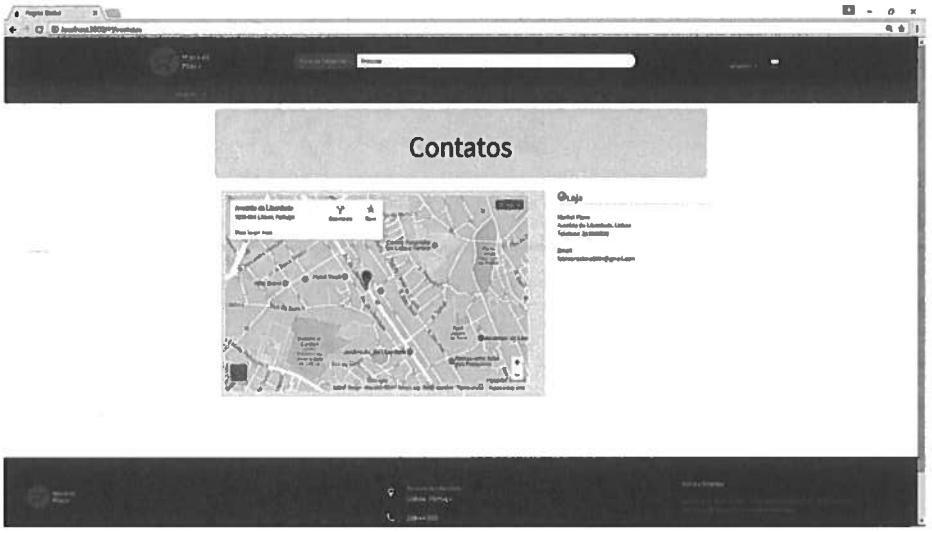


*Figura 26: Estado root.contatos*

Este estado representa a página que contém os contatos da loja física que neste caso é fictício e a localização geográfica da mesma. Não tenho nenhum controlador, porque este estado contém uma página estática. O HTML deste estado pode ser consultado na íntegra no Apêndice A.44 deste documento.

# 5. Conclusão

O objetivo do projeto global em questão era desenvolver uma aplicação *web* de comércio eletrónico em MEAN *Stack* com uma interface que fosse intuitiva e com boa usabilidade. Em que durante o desenvolvimento da aplicação, foram surgindo problemas/desafios. Desafios estes que foram superados com sucesso, uns com mais dificuldades que outros.

A aplicação final serviu para adquirir conhecimentos das tecnologias do MEAN *Stack*, das bases de dados NoSQL e do padrão de arquitetura MVC do AngularJS. Foi também possível entrar em contacto com a criação de API's para que o cliente pudesse realizar pedidos ao servidor.

Concluo então que a utilização do MEAN *Stack* para criação de aplicações *web* é uma ótima escolha. Pois a utilização de JavaScript como linguagem de programação por todas as tecnologias associadas ao MEAN *Stack* facilita o seu desenvolvido. E também o uso de JSON por todas estas tecnologias facilita a interação entre elas.

Para o futuro, espero realizar testes por código e não manualmente como foi realizado, para que possa testar as API's de maneira a que não surgem erros inesperados. Em termos de funcionalidades, espero implementar duas das funcionalidades principais do MongoDB que é o *sharding* e a replicação. Espero também melhorar o painel de administrador, pois este foi desenvolvido com alguma pressão em termos de tempo e não consegui realizar um painel de administrador que seja muito intuitivo.

# 6. Referências

Auth0. (n.d.). angular-jwt. Acesso June 16, 2017, em https://github.com/auth0/angular-jwt

Banker, K. (2011). *MongoDB in action*. Manning. Obtido em http://dl.acm.org/citation.cfm?id=2207997

Bretz, A., & Ihrig, C. (2014). *Full Stack JavaScript Development with MEAN. International Review of Administrative Sciences* (Vol. 43). SitePoint.

Chadwick, J., Snyder, T., & Panda, H. (2012). *Programming ASP.NET MVC 4*.

Chodorow, K. (2013). *MongoDB: The Definitive Guide. Journal of Infectious Diseases* (Vol. 203). O'Reilly. https://doi.org/10.1093/infdis/jir001

Eloff, E., & Torstensson, D. (2012). *An Investigation into the Applicability of Node . js as a Platform for Web Services*. Institutionen för datavetenskap. Obtido em http://www.diva-portal.org/smash/get/diva2:550993/FULLTEXT01.pdf

Express. (2017a). Express - node.js web application framework. Acesso June 16, 2017, em https://expressjs.com/

Express. (2017b). Writing middleware for use in Express apps. Acesso June 16, 2017, em https://expressjs.com/en/guide/writing-middleware.html

Foundation, N. j. (2017). Node.js. Acesso June 16, 2017, em https://nodejs.org/en/

Freeman, A. (2014). Pro AngularJS. *Apress*, 671. https://doi.org/10.1007/978-1-4302-6449-1

Google. (2016). AngularJS: Developer Guide: Dependency Injection. Obtido em https://docs.angularjs.org/guide/di

Hanson, J. (2015). passport-local. Acesso June 16, 2017, em https://github.com/jaredhanson/passport-local

Haviv, A. Q. (2014). *MEAN Web Development*. Packt Publishing Ltd.

Holmes, S. (2016). *Getting MEAN with Mongo, Express, Angular, and Node*. Manning Publications Co. Retirado de http://pdf.th7.cn/down/files/1603/Getting MEAN with Mongo, Express, Angular, and Node.pdf

Inci, D. (2009). History of eCommerce. Acesso June 16, 2017, em http://www.optimum7.com/internet-marketing/ecommerce/history-of-ecommerce.html

Iyer, G. R. (2013). Node.js : Event - driven Concurrency for Web Applications. *Technical Report, ResearchGate*. https://doi.org/10.13140/RG.2.1.2591.9849

Jones, R. (2009). OECD Economics Department Working Papers, *673*(252), 26. https://doi.org/10.1787/226625875038

JWT. (2017). JSON Web Token Introduction - jwt.io. Acesso June 16, 2017, em https://jwt.io/introduction/

Karpov, V. (2013). The MEAN Stack: MongoDB, ExpressJS, AngularJS and Node.js. Acesso June 16, 2017, em https://www.mongodb.com/blog/post/the-mean-stack-mongodb-expressjs-angularjs-and

Khoshnampour, M., & Nosrati, M. (2011). An overview of E-commerce. *World Applied Programming*, *1*(June), 94–99.

Ma'aruf, L. M., & Abdulkadir, K. (2012). An overview of e-commerce implementation in developed and developing country; A case study of United State and Nigeria. *International Journal of Modern Engineering Research (IJMER)*, *2*(5), 3068–3080.

Mardan, A. (2014). *Pro Express.js*. Apress.

MongoDB. (2015). *MongoDB Architecture Guide. MongoDB White Paper*.

Moore, G. (2004). Despite its advantages , the industry has been slow to take-up the e-commerce challenge, (September 2002), 4. Obtido em http://www.unece.lsu.edu/ebusiness/documents/2003-2006/sc03_027.pdf

Nanehkaran, Y. A. (2013). An introduction to electronic commerce. *International Journal of Scientific & Technology Research*, *2*(4), 190–193.

Niranjanamurthy, M., Kavyashree, N., & Chahar, S. J. D. (2013). Analysis of E-Commerce and M-Commerce : Advantages , Limitations and Security issues. *International Journal of Advanced Research in Computer and Communication Engineering*, *2*(6), 2360–2370.

Nyati, A. (2016). React: The UI Paintbrush You've Been Searching For | Tekuma. Acesso June 16, 2017, em http://tekuma.io/tech/react-the-ui-paintbrush-youve-been-searching-for/

Olakara, A. R. (2015). Understanding the Node.js event loop. Obtido em

http://abdelraoof.com/blog/2015/10/28/understanding-nodejs-event-loop

Panda, S. (2014). *AngularJS Novice to Ninja*. *SitePoint*. https://doi.org/10.1016/B978-0-08-036833-7.50001-3

Reed, N. (2011). What is npm? Obtido em https://docs.nodejitsu.com/articles/getting-started/npm/what-is-npm/

Rider, S. (2016). *Introduction to Building Web Apps*. Obtido em https://shawnr.gitbooks.io/introduction-to-building-webapps/content/

Seshadri, S., & Green, B. (2014). *AngularJS Up & Running*. *O'Reilly* (1st ed.).

Stripe. (2017). Stripe. Acesso June 16, 2017, em https://stripe.com/pt

Taly, A., Erlingsson, Úlfar, Mitchell, J. C., Miller, M. S., & Nagra, J. (2011). Automated analysis of security-critical JavaScript APIs. *Proceedings - IEEE Symposium on Security and Privacy*, 363–378. https://doi.org/10.1109/SP.2011.39

Techopedia. (n.d.). MongoDB. Obtido em https://www.techopedia.com/definition/30340/mongodb / Techopedia

Teixeira, P. (2013). *Professional Node.js*. John Wiley & Sons, Inc.

Ui-Router. (2017). UI-Router Features - UI-Router. Acesso June 16, 2017, em https://ui-router.github.io/about/features

UML Tutorials. (2004). Uml Tutorials Uml Tutorial Part 2, 2–4.

W3C. (2005). Document Object Model (DOM). Acesso June 16, 2017, em https://www.w3.org/DOM/

Ying, L. X. (2008). What is E-Commerce? Acesso June 16, 2017, em https://ecommerze.wordpress.com/2008/06/12/the-history-and-evoltion-of-e-commerce/

# Apêndices

## **Apêndice A.1** – Schema das categorias

```
1.  var mongoose = require('mongoose');
2.  var Schema = mongoose.Schema;
3.
4.  var CategoriasSchema = new Schema({
5.      nome: { type: String, required: true },
6.      subCategorias: [{
7.          nome: { type: String }
8.      }]
9.  });
10.
11. var Categorias = module.exports = mongoose.model('Categorias', CategoriasSchema);
12.
13. module.exports.getCategoriasPorNome = function(categoria, callback) {
14.     Categorias
15.         .find({ "nome": { $regex: categoria, $options: 'i' } })
16.         .exec(callback);
17. };
```

## **Apêndice A.2** – Schema dos produtos

```
1.  var mongoose = require('mongoose');
2.  var Schema = mongoose.Schema;
3.
4.  var ProdutoSchema = new Schema({
5.      nome: {type: String, required: true},
6.      marca: {type: String, required: true},
7.      categoria: {type: Schema.Types.ObjectId, ref: 'Categorias'},
8.      subCategoria: {type: String},
9.      destaque: {type: Boolean, default: false},
10.     data: {type: Date, default: Date.now},
11.     preco: {type: Number, required: true},
12.     quantidade: {type: Number, required: true},
13.     descricao: {type: String},
14.     imagem: {
15.         nome: {type: String},
16.         file: {type: Object}
17.     },
18.     comentarios: [{
19.         username: {type: Schema.Types.ObjectId, ref: 'Utilizador'},
20.         conteudo: {type: String},
21.         introduzido: {type: Date, default: Date.now}
22.     }, {_id: false}]
23. });
24.
25. var Produto = module.exports = mongoose.model('Produto', ProdutoSchema);
26.
27. module.exports.findAllCategorias = function (callback) {
28.     Produto
29.         .find()
30.         .populate('categoria')
31.         .exec(callback);
32. };
33.
34. module.exports.getProdutoPorId = function (id, callback) {
35.     Produto.findById(id)
36.         .populate('categoria')
37.         .populate('comentarios.username')
38.         .exec(callback);
39. };
40.
```

```
41. module.exports.getProdutoPorNome = function (nome, callback) {
42.     Produto.find({
43.         "nome": {$regex: nome, $options: 'i'}
44.     })
45.         .populate('categoria')
46.         .exec(callback);
47. };
48. module.exports.getProdutoPorDestaque = function (callback) {
49.     Produto.find({
50.         "destaque": true
51.     })
52.         .exec(callback);
53. };
54.
55. module.exports.getProdutoPorNomeComCategoria = function (nome, categoria, callback)
    {
56.     Produto
57.         .find({
58.             "nome": {$regex: nome, $options: 'i'},
59.             "categoria": categoria
60.         })
61.         .populate('categoria')
62.         .exec(callback);
63. };
64. module.exports.getProdutoPorCategoria = function (categoria, callback) {
65.     Produto
66.         .find({"categoria": categoria})
67.         .populate('categoria')
68.         .exec(callback);
69. };
70. module.exports.getProdutoPorNomeComSubCategoria = function (subCategoria, produtono
    me, callback) {
71.     Produto
72.         .find({
73.             "nome": {$regex: produtonome, $options: 'i'},
74.             'subCategoria': subCategoria
75.         })
76.         .exec(callback);
77. };
78. module.exports.getProdutoPorSubCategoria = function (subCategoria, callback) {
79.     Produto
80.         .find({
81.             'subCategoria': subCategoria
82.         })
83.         .exec(callback);
84. };
85.
86. module.exports.AdicionarComentarios = function (produtoid, comentario, callback) {

87.     Produto
88.         .findOneAndUpdate({_id: produtoid}, {
89.             $push: {
90.                 comentarios: comentario
91.             }
92.         }, {
93.             new: true
94.         })
95.         .exec(callback);
96. }
```

## **Apêndice A.3** – Schema dos utilizadores

```
1. var mongoose = require('mongoose');
```

```
2.  var Schema = mongoose.Schema;
3.  var bcrypt = require('bcrypt-nodejs');
4.  var jwt = require('jsonwebtoken');
5.
6.  var UtilizadorSchema = new Schema({
7.      nomeCompleto: { type: String },
8.      username: { type: String, required: true },
9.      email: { type: String, required: true },
10.     password: { type: String, required: true },
11.     imagemPerfil: {
12.         nome: String,
13.         file: Object
14.     },
15.     morada: {
16.         morada: { type: String },
17.         localidade: { type: String },
18.         codigoPostal: { type: Number },
19.         telefone: { type: Number }
20.     },
21.     favoritos: [{ type: Schema.Types.ObjectId, ref: 'Produto' }],
22.     role: { type: String, default: 'user' },
23.     carrinho: { type: Object }
24. });
25.
26. UtilizadorSchema.methods.encryptPassword = function(password) {
27.     return bcrypt.hashSync(password, bcrypt.genSaltSync(8), null);
28. };
29.
30. UtilizadorSchema.methods.validPassword = function(password) {
31.     return bcrypt.compareSync(password, this.password);
32. };
33.
34. UtilizadorSchema.methods.gerarJwt = function() {
35.     return jwt.sign({
36.         _id: this._id,
37.         email: this.email,
38.         username: this.username,
39.         nomeCompleto: this.nomeCompleto,
40.         imagemPerfil: this.imagemPerfil || "",
41.         morada: this.morada || "",
42.         favoritos: this.favoritos || "",
43.         role: this.role,
44.         carrinho: this.carrinho,
45.         exp: Math.floor(Date.now() / 1000) + (60 * 60), //(1 hora)
46.     }, "secret");
47. };
48.
49. var Utilizador = module.exports = mongoose.model('Utilizador', UtilizadorSchema);
50.
51. module.exports.findFavoritos = function(userID, callback) {
52.     Utilizador
53.         .findById(userID)
54.         .populate('favoritos')
55.         .exec(callback);
56. };
```

## **Apêndice A.4** – Ficheiro do carrinho de compras

```
1.  module.exports = function Carrinho(oldCart) {
2.      this.items = oldCart.items || {};
3.      this.totalQty = oldCart.totalQty || 0;
4.      this.totalPreco = oldCart.totalPreco || 0;
5.
```

```
    this.adicionar = function (item, id, qty) {
        var storeItem = this.items[id];
        if (!storeItem) {
            storeItem = this.items[id] = {item: item, qty: 0, preco: 0};
        }
        storeItem.qty = storeItem.qty + qty;
        storeItem.preco = storeItem.item.preco * storeItem.qty;
        this.totalQty = this.totalQty + qty;
        this.totalPreco = this.totalPreco + storeItem.item.preco * qty;
    };

    this.removerUm = function (id) {
        this.items[id].qty--;
        this.items[id].preco -= this.items[id].item.preco;
        this.totalQty--;
        this.totalPreco -= this.items[id].item.preco;

        if (this.items[id].qty <= 0) {
            delete this.items[id];
        }
    };

    this.removerItem = function (id) {
        if (this.items[id]) {
            this.totalQty -= this.items[id].qty;
            this.totalPreco -= this.items[id].preco;
            delete this.items[id];
        }
    };
};
```

## **Apêndice A.5** – Endereçamento de pedidos do painel de administrador

```
var express = require('express');
var router = express.Router();
var Categorias = require('../models/categorias');
var Produtos = require('../models/produtos');
var Utilizadores = require('../models/utilizadores');
var Carrinho = require('../models/carrinho');
var Encomendas = require('../models/encomendas');
var MetodosEnvio = require('../models/metodosEnvio');
var autenticacaoAdmin = require('../config/admin');
var expressjwt = require('express-jwt');
var authenticateUser = expressjwt({secret: 'secret'});
var multer = require('multer');

// Multer
var storage = multer.diskStorage({
    destination: function (req, file, cb) {
        cb(null, './public/images/uploads/produtos');
    },
    filename: function (req, file, cb) {
        cb(null, req.params.produtoID + '.png');
    }
});
var upload = multer({storage: storage});

router.post('/criarSubCategoria/:categoriaID', authenticateUser, autenticacaoAdmin(
'admin'), function (req, res, next) {
    var subCategoria = {nome: req.body.subCategoria};
    Categorias.findByIdAndUpdate(req.params.categoriaID, {$push: {subCategorias: su
bCategoria}}, {new: true}, function (err, categoria) {
        if (err) console.log(err);
```

```
29.         else res.json(categoria);
30.     });
31. });
32.
33. router.post('/removerSubCategoria/:categoriaID', authenticateUser, autenticacaoAdmi
    n('admin'), function (req, res, next) {
34.     var subCategoria = {nome: req.body.subCategoria};
35.     Categorias.findByIdAndUpdate(req.params.categoriaID, {$pull: {'subCategorias':
    subCategoria}}, {'new': true}, function (err, categoria) {
36.         if (err) console.log(err);
37.         else res.json(categoria);
38.     });
39. });
40.
41. router.post('/criarProduto', authenticateUser, autenticacaoAdmin('admin'), function
     (req, res, next) {
42.     var novoProduto = new Produtos({
43.         nome: req.body.nome,
44.         marca: req.body.marca,
45.         categoria: req.body.categoria,
46.         subCategoria: req.body.subCategoria,
47.         preco: req.body.preco,
48.         quantidade: req.body.quantidade,
49.         descricao: req.body.descricao,
50.         destaque: req.body.destaque,
51.         imagem: {
52.             nome: '',
53.             file: {
54.                 filename: 'produtoSemImagem.png'
55.             }
56.         }
57.     });
58.     novoProduto.save(function (err) {
59.         if (err) console.log(err);
60.         res.json("Criado com sucesso");
61.     });
62. });
63.
64. router.post('/editarProduto', authenticateUser, autenticacaoAdmin('admin'), functio
    n (req, res, next) {
65.     var novoProduto = {
66.         nome: req.body.nome,
67.         marca: req.body.marca,
68.         categoria: req.body.categoria,
69.         subCategoria: req.body.subCategoria,
70.         preco: req.body.preco,
71.         quantidade: req.body.quantidade,
72.         descricao: req.body.descricao,
73.         destaque: req.body.destaque
74.     };
75.     Produtos.findByIdAndUpdate(req.body._id, novoProduto, function (err, result) {

76.         res.json("Editado com sucesso");
77.     })
78.
79. });
80.
81. router.post('/alterarImagemProduto/:produtoID', upload.single('file'), authenticate
    User, autenticacaoAdmin('admin'), function (req, res) {
82.     var novoUpload = {
83.         nome: req.file.originalname,
84.         file: req.file
85.     };
86.     Produtos.findByIdAndUpdate(req.params.produtoID, {$set: {imagem: novoUpload}},
    {'new': true}, function (err, produto) {
87.         if (!produto) res.json({message: "Id nao encontrado"});
```

```
88.         else res.json(produto);
89.     });
90. });
91.
92. router.post('/removerProduto/:produtoID', authenticateUser, autenticacaoAdmin('admi
    n'), function (req, res) {
93.     Utilizadores.find({favoritos: req.params.produtoID}, function (err, user) {
94.         if (err) {
95.             console.log(err);
96.         }
97.         for (var i = 0; i < user.length; i++) {
98.             for (var a = 0; a < user[i].favoritos.length; a++) {
99.                 var favoritos = user[i].favoritos;
100.                    if (favoritos[a] == req.params.produtoID) {
101.                        favoritos.splice(a, 1);
102.                    }
103.                    user[i].favoritos = favoritos;
104.                }
105.                var carrinho = new Carrinho(user[i].carrinho ? user[i].carrinho
    : {});
106.                carrinho.removerItem(req.params.produtoID);
107.                user[i].carrinho = carrinho;
108.                user[i].save();
109.            }
110.            Produtos.findByIdAndRemove(req.params.produtoID, function (err, prod
    uto) {
111.                if (err) console.log(err);
112.                res.json("Removido com sucesso")
113.            });
114.        })
115.    });
116.
117.    router.post('/encomendaAlterar/:encomendaID', authenticateUser, autenticacao
    Admin('admin'), function (req, res) {
118.        var estado = 'Enviada';
119.        Encomendas.findByIdAndUpdate(req.params.encomendaID, {estadoEncomenda: e
    stado}, {new: true}, function (err, encomendas) {
120.            res.json(encomendas);
121.        });
122.    });
123.
124.    router.post('/utilizadoresRole/:userID', authenticateUser, autenticacaoAdmin
    ('admin'), function (req, res) {
125.        var role = req.body.role;
126.        Utilizadores.findByIdAndUpdate(req.params.userID, {role: role}, {new: tr
    ue}, function (err, utilizador) {
127.            res.json(utilizador);
128.        });
129.    });
130.
131.    router.post('/metodosEnvio/adicionar', authenticateUser, autenticacaoAdmin('
    admin'), function (req, res) {
132.        var metodoEnvio = new MetodosEnvio({
133.            nome: req.body.metodo.nome,
134.            preco: req.body.metodo.preco,
135.            model: req.body.metodo.model
136.        });
137.        metodoEnvio.save(function (err) {
138.            res.json({message: 'Método criado com sucesso.'})
139.        });
140.    });
141.
142.
143.    router.post('/metodosEnvio/remover/:metodoID', authenticateUser, autenticaca
    oAdmin('admin'), function (req, res) {
```

```
144.          MetodosEnvio.findByIdAndRemove(req.params.metodoID, function (err, metod
    o) {
145.              res.json("Removido com sucesso")
146.          });
147.      });
148.
149.      module.exports = router;
```

## Apêndice A.6 – Endereçamento de pedidos das categorias

```
1.  var express = require('express');
2.  var router = express.Router();
3.  var Categorias = require('../models/categorias');
4.
5.  router.get('/', function (req, res, next) {
6.      Categorias.find(function (err, categorias) {
7.          res.json(categorias);
8.      });
9.  });
10.
11. router.get('/:categoria', function (req, res, next) {
12.     if (req.params.categoria == 'Todas as Categorias') {
13.         Categorias.find(function (err, categorias) {
14.             res.json(categorias);
15.         });
16.     } else {
17.         Categorias.getCategoriasPorNome(req.params.categoria, function (err, catego
    ria) {
18.             res.json(categoria);
19.         });
20.     }
21. });
22.
23. module.exports = router;
```

## Apêndice A.7 – Endereçamento de pedidos das encomendas

```
1.  var express = require('express');
2.  var router = express.Router();
3.  var Encomendas = require('../models/encomendas');
4.  var expressjwt = require('express-jwt');
5.  var authenticateUser = expressjwt({secret: 'secret'});
6.
7.  router.get('/', authenticateUser, function (req, res, next) {
8.      Encomendas.find(function (err, encomendas) {
9.          res.json(encomendas);
10.     })
11. });
12.
13. router.get('/:userID', authenticateUser, function (req, res, next) {
14.     Encomendas.find({'utilizador': req.params.userID}, function (err, encomendas) {

15.         res.json(encomendas);
16.     })
17. });
18. router.get('/encomendaDetalhe/:encomendaID', authenticateUser, function (req, res,
    next) {
19.     Encomendas.findById(req.params.encomendaID, function (err, encomenda) {
20.         res.json(encomenda);
21.     })
22. });
```

```
23.
24. module.exports = router;
```

## **Apêndice A.8** – Endereçamento de pedidos dos métodos de envio

```
1.  var express = require('express');
2.  var router = express.Router();
3.  var MetodosEnvio = require('../models/metodosEnvio');
4.  var expressjwt = require('express-jwt');
5.  var authenticateUser = expressjwt({secret: 'secret'});
6.
7.  router.get('/', authenticateUser, function (req, res, next) {
8.      MetodosEnvio.find(function (err, metodos) {
9.          if (err) console.log(err);
10.         res.json(metodos)
11.     })
12. });
13. module.exports = router;
```

## **Apêndice A.9** – Endereçamento de pedidos dos produtos

```
1.  var express = require('express');
2.  var router = express.Router();
3.  var expressjwt = require('express-jwt');
4.  var authenticateUser = expressjwt({secret: 'secret'});
5.  var stripe = require("stripe")("sk_test_FrklkPL86DJacK5wzOH0NVdR");
6.  var nodemailer = require('nodemailer');
7.  var Produtos = require('../models/produtos');
8.  var Categorias = require('../models/categorias');
9.  var Utilizadores = require('../models/utilizadores');
10. var Carrinho = require('../models/carrinho');
11. var Encomendas = require('../models/encomendas');
12.
13. var transporter = nodemailer.createTransport({
14.     service: 'Gmail',
15.     auth: {
16.         user: 'fabioanselmo1994@gmail.com',
17.         pass: 'nptpaylqvktrfxbj'
18.     }
19. });
20.
21. router.get('/', function (req, res, next) {
22.     Produtos.findAll(function (err, produtos) {
23.         if (err) console.log(err);
24.         res.json(produtos);
25.     });
26. });
27.
28. router.get('/destaque', function (req, res, next) {
29.     Produtos.getProdutoPorDestaque(function (err, produtos) {
30.         if (err) console.log(err);
31.         res.json(produtos);
32.     });
33. });
34. router.get('/Categorias', function (req, res, next) {
35.     Produtos.findAllCategorias(function (err, produtos) {
36.         if (err) console.log(err);
37.         res.json(produtos);
38.     });
39. });
40.
```

```
41. router.get('/detalhes/:id', function (req, res, next) {
42.     Produtos.getProdutoPorId(req.params.id,
43.         function (err, produtos) {
44.             if (err) console.log(err);
45.             for (var i = 0; i < produtos.comentarios.length; i++) {
46.                 produtos.comentarios[i].username.password = null;
47.                 produtos.comentarios[i].username.role = null;
48.                 produtos.comentarios[i].username.morada = null;
49.             }
50.             if (produtos.quantidade == 0) {
51.                 res.json({produto: produtos, stock: "Indisponivel"});
52.             }
53.             else if (produtos.quantidade > 0 && produtos.quantidade < 5) {
54.                 res.json({produto: produtos, stock: "Stock Limitado"});
55.             }
56.             else if (produtos.quantidade >= 5) {
57.                 res.json({produto: produtos, stock: "Disponivel"});
58.             }
59.         });
60. });
61.
62. router.get('/procurar/:categoria/:nome', function (req, res, next) {
63.     Categorias.getCategoriasPorNome(req.params.categoria, function (err, categoria)
    {
64.         if (err) console.log(err);
65.         categoria1 = categoria;
66.         //se for todas as categorias procura só por nome
67.         if (req.params.categoria == 'Todas as Categorias') {
68.             Produtos.getProdutoPorNome(req.params.nome, function (err, produto) {
69.                 if (err) console.log(err);
70.                 res.json(produto);
71.             });
72.         } else {
73.             Produtos.getProdutoPorNomeComCategoria(req.params.nome, categoria1, fun
    ction (err, produto) {
74.                 if (err) console.log(err);
75.                 res.json(produto);
76.             });
77.         }
78.     });
79. });
80.
81. router.get('/procurar/:categoria', function (req, res, next) {
82.     //se for todas as categorias procura por nome
83.     if (req.params.categoria == 'Todas as Categorias') {
84.         Produtos.findAll(function (err, produtos) {
85.             if (err) console.log(err);
86.             res.json(produtos);
87.         });
88.     } else {
89.         Categorias.getCategoriasPorNome(req.params.categoria, function (err, catego
    ria) {
90.             if (err) console.log(err);
91.             Produtos.getProdutoPorCategoria(categoria, function (err, produtos) {
92.                 if (err) console.log(err);
93.                 res.json(produtos);
94.             });
95.         });
96.     }
97. });
98.
99. router.get('/subCategoria/procurar/:subCategoria/:produtonome', function (req, res,
     next) {
100.         Produtos.getProdutoPorNomeComSubCategoria(req.params.subCategoria, req.p
    arams.produtonome, function (err, produto) {
101.             if (err) console.log(err);
```

```
102.                res.json(produto);
103.            })
104.        })
105.
106.        router.get('/subCategoria/procurar/:subCategoria', function (req, res, next)
    {
107.            Produtos.getProdutoPorSubCategoria(req.params.subCategoria, function (er
    r, produto) {
108.                if (err) console.log(err);
109.                res.json(produto);
110.            });
111.        });
112.
113.        router.post('/comentarios/adcionar/:produtoID/:userID', authenticateUser, fu
    nction (req, res, next) {
114.            var novoComentario = {
115.                username: req.params.userID,
116.                conteudo: req.body.comentario
117.            };
118.            Produtos.AdicionarComentarios(req.params.produtoID, novoComentario, func
    tion (err, produto) {
119.                if (err) console.log(err);
120.                if (!produto) console.log("nao user");
121.                Produtos.getProdutoPorId(req.params.produtoID,
122.                    function (err, produtos) {
123.                        if (err) console.log(err);
124.                        for (var i = 0; i < produtos.comentarios.length; i++) {
125.                            produtos.comentarios[i].username.password = null;
126.                            produtos.comentarios[i].username.role = null;
127.                            produtos.comentarios[i].username.morada = null;
128.                        }
129.                        res.json(produtos.comentarios);
130.                    });
131.            });
132.        });
133.
134.
135.        router.post('/add-
    Carrinho/:userID/:produtoID', authenticateUser, function (req, res, next) {
136.            Produtos.findById({_id: req.params.produtoID},
137.                function (err, produto) {
138.                    if (err) console.log(err);
139.                    if (!produto) console.log("Produto nao encontrado");
140.                    if (produto.quantidade > 0 && produto.quantidade >= req.body.qty
    ) {
141.                        var carrinho = new Carrinho(req.body.carrinho ? req.body.car
    rinho : {});
142.                        var produtoQty = produto.quantidade - req.body.qty;
143.                        produto.update({quantidade: produtoQty}, function (err, resu
    lt) {
144.                            if (err) console.log(err);
145.                            carrinho.adicionar(produto, req.params.produtoID, req.bo
    dy.qty);
146.                            Utilizadores.findByIdAndUpdate(req.params.userID, {carri
    nho: carrinho}, function (err, user) {
147.                                if (err) console.log(err);
148.                                req.body.carrinho = carrinho;
149.                                res.json(req.body.carrinho);
150.                            });
151.                        });
152.                    }
153.                    else res.json({errorMessage: "Produto Indisponivel"})
154.                });
155.        });
156.
```

```
157.        router.post('/removerUm-
    Carrinho/:userID/:produtoID', authenticateUser, function (req, res, next) {
158.            var carrinho = new Carrinho(req.body.carrinho ? req.body.carrinho : {});

159.            Produtos.findById({_id: req.params.produtoID},
160.                function (err, produto) {
161.                    if (err) console.log(err);
162.                    if (!produto) console.log("Produto nao encontrado");
163.                    var produtoQty = produto.quantidade + 1;
164.                    produto.update({quantidade: produtoQty}, function (err, result)
    {
165.                        if (err) console.log(err);
166.                        carrinho.removerUm(req.params.produtoID);
167.                        Utilizadores.findByIdAndUpdate(req.params.userID, {carrinho:
     carrinho}, function (err, user) {
168.                            if (err) console.log(err);
169.                            req.body.carrinho = carrinho;
170.                            res.json(req.body.carrinho);
171.                        });
172.                    });
173.                });
174.        });
175.
176.        router.post('/remover-
    Carrinho/:userID/:produtoID', function (req, res, next) {
177.            var carrinho = new Carrinho(req.body.carrinho ? req.body.carrinho : {});

178.            Produtos.findById({_id: req.params.produtoID},
179.                function (err, produto) {
180.                    if (err) console.log(err);
181.                    if (!produto) console.log("Produto nao encontrado");
182.                    var produtoQty = produto.quantidade + carrinho.items[req.params.
    produtoID].qty;
183.                    produto.update({quantidade: produtoQty}, function (err, result)
    {
184.                        if (err) console.log(err);
185.                        carrinho.removerItem(req.params.produtoID);
186.                        Utilizadores.findByIdAndUpdate(req.params.userID, {carrinho:
     carrinho}, function (err, user) {
187.                            if (err) console.log(err);
188.                            req.body.carrinho = carrinho;
189.                            res.json(req.body.carrinho);
190.                        });
191.                    });
192.                });
193.        });
194.
195.        router.post('/checkout/cartaoCredito/:userID', authenticateUser, function (r
    eq, res, next) {
196.            var carrinho = new Carrinho(req.body.carrinho ? req.body.carrinho : {});

197.            stripe.charges.create({
198.                amount: carrinho.totalPreco * 100,
199.                currency: "eur",
200.                source: req.body.stripeToken.id,
201.                description: "Teste"
202.            }, function (err, charge) {
203.                if (err) console.log(err);
204.                var encomendas = new Encomendas({
205.                    utilizador: req.params.userID,
206.                    items: carrinho,
207.                    moradaShipping: {
208.                        morada: req.body.morada.morada,
209.                        localidade: req.body.morada.localidade,
210.                        codigoPostal: req.body.morada.codigoPostal,
211.                        telefone: req.body.morada.telefone
```

```
212.                     },
213.                     precoTotal: carrinho.totalPreco,
214.                     tipoDePagamento: charge.source.brand,
215.                     pagamentoID: charge.id,
216.                     portes: {
217.                         nome: req.body.portes,
218.                         preco: req.body.portesValor
219.                     }
220.                 });
221.                 encomendas.save(function (err, result) {
222.                     carrinho = {};
223.                     Utilizadores.findByIdAndUpdate(req.params.userID, {carrinho: car
    rinho}, function (err, user) {
224.                         if (err) console.log(err);
225.                         var mailOptions = {
226.                             from: 'Market Place <fabioanselmo1994@gmail.com>',
227.                             to: user.email,
228.                             subject: 'Encomenda do Market Place Efectuada com Sucess
    o',
229.                             text: 'Encomenda efetuada com sucesso...Nome: ' + user.n
    omeCompleto + ' Email: ' + user.email + ' Encomenda Nº: ' + result._id + ' Preco To
    tal: $' + result.precoTotal + ' Tipo de Pagamento: ' + result.tipoDePagamento,
230.                             html: '<p>Encomenda efetuada com sucesso.</p><ul><li>Nom
    e: ' + user.nomeCompleto + '</li><li>Email: ' + user.email + '</li><li>Encomenda Nº
    : ' + result._id + '</li><li>Preco Total: $' + result.precoTotal + '</li><li>Tipo d
    e Pagamento: ' + result.tipoDePagamento + '</li></ul>'
231.                         };
232.                         transporter.sendMail(mailOptions, function (err, info) {
233.                             if (err) console.log(err);
234.                             req.body.carrinho = carrinho;
235.                             res.json({carrinho: req.body.carrinho, result: result});

236.                         });
237.
238.                     });
239.                 });
240.             });
241.         });
242.
243.         router.post('/checkout/cobranca/:userID', authenticateUser, function (req, r
    es, next) {
244.             var carrinho = new Carrinho(req.body.carrinho ? req.body.carrinho : {});

245.             var encomendas = new Encomendas({
246.                 utilizador: req.params.userID,
247.                 items: carrinho,
248.                 moradaShipping: {
249.                     morada: req.body.morada.morada,
250.                     localidade: req.body.morada.localidade,
251.                     codigoPostal: req.body.morada.codigoPostal,
252.                     telefone: req.body.morada.telefone
253.                 },
254.                 precoTotal: carrinho.totalPreco,
255.                 tipoDePagamento: "Cobrança",
256.                 pagamentoID: "Id Multibanco",
257.                 portes: {
258.                     nome: req.body.portes,
259.                     preco: req.body.portesValor
260.                 }
261.             });
262.             encomendas.save(function (err, result) {
263.                 carrinho = {};
264.                 Utilizadores.findByIdAndUpdate(req.params.userID, {carrinho: carrinh
    o}, function (err, user) {
265.                     if (err) console.log(err);
266.                     var mailOptions = {
```

```
                    from: 'Market Place <fabioanselmo1994@gmail.com>',
                    to: user.email,
                    subject: 'Encomenda do Market Place Efectuada com Sucesso',
                    text: 'Encomenda efetuada com sucesso...Nome: ' + user.nomeC
ompleto + ' Email: ' + user.email + ' Encomenda Nº: ' + result._id + ' Preco Total:
 $' + result.precoTotal + ' Tipo de Pagamento: ' + result.tipoDePagamento,
                    html: '<p>Encomenda efetuada com sucesso.</p><ul><li>Nome: '
 + user.nomeCompleto + '</li><li>Email: ' + user.email + '</li><li>Encomenda Nº: '
+ result._id + '</li><li>Preco Total: $' + result.precoTotal + '</li><li>Tipo de Pa
gamento: ' + result.tipoDePagamento + '</li></ul>'
                };
                transporter.sendMail(mailOptions, function (err, info) {
                    if (err) console.log(err);
                    req.body.carrinho = carrinho;
                    res.json({carrinho: req.body.carrinho, result: result});
                });
            });
        });
    });

    // Handle invalid JWT
    router.use(function (err, req, res, next) {
        if (err.constructor.name === 'UnauthorizedError') res.status(401).send('
invalid token...');
    });

    module.exports = router;
```

## Apêndice A.10 – Endereçamento de pedidos dos banners

```
var express = require('express');
var router = express.Router();
var ProdutosBanner = require('../models/produtoBanner');
var autenticacaoAdmin = require('../config/admin');
var expressjwt = require('express-jwt');
var authenticateUser = expressjwt({secret: 'secret'});
var multer = require('multer');

// Multer / Save img in the local folder and save with its name
var storage = multer.diskStorage({
    destination: function (req, file, cb) {
        cb(null, './public/images/uploads/produtosBanner');
    },
    filename: function (req, file, cb) {
        cb(null, file.originalname);
    }
});
var upload = multer({
    storage: storage
});

router.get('/', function (req, res, next) {
    ProdutosBanner.find(function (err, produtosBanner) {
        if (err) {
            console.log(err);
        }
        res.json(produtosBanner);
    });
});

router.post('/criarImagem', upload.single('file'), authenticateUser, autenticacaoAd
min('admin'), function (req, res) {
```

```
32.     var novoUpload = {
33.         nome: req.file.filename,
34.         imagem: req.file
35.     };
36.     ProdutosBanner.create(novoUpload, function (err) {
37.         if (err) console.log(err);
38.         res.json("Banner Adicionado")
39.     })
40. });
41.
42. router.post('/remover/:bannerID',  authenticateUser, autenticacaoAdmin('admin'), fu
    nction (req, res) {
43.     ProdutosBanner.findByIdAndRemove(req.params.bannerID, function (err, produto) {

44.         if (err) console.log(err);
45.         res.json("Removido com sucesso")
46.     });
47. });
48.
49. module.exports = router;
```

## **Apêndice A.11–** Endereçamento de pedidos dos utilizadores

```
1.  var express = require('express');
2.  var router = express.Router();
3.  var passport = require('passport');
4.  var expressjwt = require('express-jwt');
5.  var authenticateUser = expressjwt({secret: 'secret'});
6.  var multer = require('multer');
7.  var autenticacaoAdmin = require('../config/admin')
8.  var Utilizadores = require('../models/utilizadores');
9.  var Encomendas = require('../models/encomendas');
10.
11. // Multer, guarda a imagem num pasta local com o nome do id
12. var storage = multer.diskStorage({
13.     destination: function (req, file, cb) {
14.         cb(null, './public/images/uploads/utilizadores');
15.     },
16.     filename: function (req, file, cb) {
17.         cb(null, req.params.id + '.png');
18.     }
19. });
20. var upload = multer({storage: storage});
21.
22. router.get('/', authenticateUser,autenticacaoAdmin('admin'),function (req, res, nex
    t) {
23.     Utilizadores.find(function (err, utilizadores) {
24.         res.json(utilizadores);
25.     });
26. });
27. router.post('/registar', function (req, res, next) {
28.     Utilizadores.findOne({$or: [{email: req.body.email}, {username: req.body.userna
    me}]},
29.         function (err, user) {
30.             if (user) {
31.                 res.json({errorMessage: 'Username ou Email ja em uso.'});
32.             }
33.             else {
34.                 var newUser = new Utilizadores();
35.                 newUser.nomeCompleto = req.body.nomeCompleto;
36.                 newUser.username = req.body.username.toLowerCase();
37.                 newUser.email = req.body.email.toLowerCase();
38.                 newUser.password = newUser.encryptPassword(req.body.password);
```

```
39.                 newUser.imagemPerfil.nome = 'noimage.png';
40.                 newUser.save(function (err) {
41.                     res.json({
42.                         "message": "registo efectuado com sucesso!"
43.                     });
44.                 });
45.             }
46.         });
47. });
48.
49. // route to log in
50. router.post('/login', passport.authenticate('local-
    login', {session: false}), function (req, res, next) {
51.     res.json(req.user);
52. });
53. // route to log out
54. router.post('/logout', function (req, res) {
55.     res.json('Logout');
56. });
57.
58. // route para verificar se o utilizador esta logado ou nao
59. router.get('/loggedin', authenticateUser, function (req, res) {
60.     if (req.isAuthenticated()) {
61.         Utilizadores.findById({
62.             _id: req.user._id
63.         }, function (err, user) {
64.             user.password = null;
65.             res.json(user);
66.         });
67.     } else {
68.         res.json('Nao autenticado.');
69.     }
70. });
71.
72. router.get('/loggedin/role', authenticateUser, autenticacaoAdmin('admin'), function
    (req, res) {
73.     if (req.isAuthenticated()) {
74.         Utilizadores.findById({
75.             _id: req.user._id
76.         }, function (err, user) {
77.             user.password = null;
78.             res.json(user);
79.         });
80.     } else {
81.         res.json('none');
82.     }
83. });
84.
85. router.post('/profile', authenticateUser, function (req, res, next) {
86.     Utilizadores.findOne({
87.         email: req.body.email
88.     }, function (err, user) {
89.         user.update({nomeCompleto: req.body.nomeCompleto}, function (err, raw) {
90.             if (err) {
91.                 console.log(err);
92.             }
93.             Utilizadores.findOne({
94.                 email: req.body.email
95.             }, function (err, user) {
96.                 if (err) {
97.                     console.log(err);
98.                 }
99.                 user.password = undefined;
100.                    res.json(user);
101.                });
102.            });
```

```
103.          });
104.      });
105.
106.      router.post('/profile/:id', authenticateUser, function (req, res, next) {
107.          Utilizadores.findOne({
108.              email: req.body.email
109.          }, function (err, user) {
110.              if (user) {
111.                  res.json({message: "Email ja em uso"});
112.              }
113.              Utilizadores.findById({
114.                  _id: req.params.id
115.              }, function (err, user) {
116.                  if (err) {
117.                      res.json({message: "Id nao encontrado"});
118.                  }
119.                  user.update({email: req.body.email, nomeCompleto: req.body.nomeC
    ompleto}, function (err, user1) {
120.                      if (err) {
121.                          console.log(err);
122.                      }
123.                      res.json(user1);
124.                  });
125.              });
126.          });
127.      });
128.
129.      router.post('/profile/morada/:id', authenticateUser, function (req, res, nex
    t) {
130.          var novaMorada = {
131.              morada: req.body.morada,
132.              localidade: req.body.localidade,
133.              codigoPostal: req.body.codigoPostal,
134.              telefone: req.body.telefone
135.          };
136.          Utilizadores.findByIdAndUpdate(req.params.id, {morada: novaMorada}, {'ne
    w': true}, function (err, user) {
137.              if (err) {
138.                  console.log(err);
139.              }
140.              res.json(user);
141.          });
142.      });
143.
144.      router.post('/profile/imagePerfil/:id', upload.single('file'), authenticateU
    ser, function (req, res) {
145.          var novoUpload = {
146.              nome: req.file.filename,
147.              file: req.file
148.          };
149.          Utilizadores.findByIdAndUpdate(req.params.id, {$set: {imagemPerfil: novo
    Upload}}, {'new': true}, function (err, user) {
150.              if (!user) {
151.                  res.json({message: "Id nao encontrado"});
152.              }
153.              user.password = undefined;
154.              res.json(user);
155.          });
156.      });
157.
158.      router.post('/addFavoritos/:userID/:produtoID', authenticateUser, function (
    req, res) {
159.          Utilizadores.findById(req.params.userID, function (err, user) {
160.              if (!user) {
161.                  res.json({message: "Id nao encontrado"});
162.              }
```

```
163.                //Verificar se o produto ja existe nos favoritos
164.                if (user.favoritos.indexOf(req.params.produtoID) == -1) {
165.                    user.update({$push: {'favoritos': req.params.produtoID}}, functi
    on (err, user1) {
166.                        if (err) {
167.                            console.log(err);
168.                        }
169.                        res.json(user1);
170.                    });
171.                }
172.            });
173.        });
174.
175.        router.post('/removerFavoritos/:userID/:productID', authenticateUser, functi
    on (req, res) {
176.            Utilizadores.findById(req.params.userID, function (err, user) {
177.                if (!user) {
178.                    res.json({message: "Id nao encontrado"});
179.                }
180.                Utilizadores.findByIdAndUpdate(req.params.userID, {$pull: {'favorito
    s': req.params.productID}}, {'new': true}, function (err, user) {
181.                    if (err) {
182.                        console.log(err);
183.                    }
184.                    res.json(user);
185.                });
186.
187.            });
188.        });
189.
190.        router.get('/favoritos/:userID', authenticateUser, function (req, res) {
191.            Utilizadores.findFavoritos(req.params.userID, function (err, user) {
192.                if (err) {
193.                    console.log(err);
194.                }
195.                if (!user) {
196.                    res.json('user nao encontrado');
197.                }
198.                res.json(user);
199.            });
200.        });
201.
202.        // Handle invalid JWT
203.        router.use(function (err, req, res, next) {
204.            console.log(err);
205.            if (err.constructor.name === 'UnauthorizedError') {
206.                res.status(401).send('invalid token...');
207.            }
208.        });
209.
210.        module.exports = router;
```

## Apêndice A.12 – Filtros

```
1.  var app = angular.module('projetoGlobal');
2.
3.  //Onde comeca os artigos na grid (ordem)
4.  app.filter('startFrom', function () {
5.      return function (input, start) {
6.          return input.slice(start);
7.      };
8.  });
9.
```

```
10. app.filter('FiltroMarca', function () {
11.     return function (produtos, elemento) {
12.         return produtos.filter(function (produto) {
13.             for (var i = 0; i < elemento.length; i++) {
14.                 // se o elemento(marca) selecionda for igual a marca do produto ent
    ao ele devolve o produto
15.                 if (elemento[i].indexOf(produto.marca) != -1) {return true;}
16.             }
17.             // se não tiver nada selecionado mostra tudo
18.             if (elemento.length == 0) {return true;}
19.             return false;
20.         });
21.     };
22. });
23.
24. app.filter('FiltroPreco', function () {
25.     return function (produtos, ele) {
26.         return produtos.filter(function (produto) {
27.             // se o produto tiver um preco entre o ele.Max e o ele.Min entao retorn
    a true
28.             if (produto.preco <= ele.max && produto.preco >= ele.min) {return true;
    }
29.         });
30.     }
31. });
```

## Apêndice A.13 – Factories

```
1.  angular.module('projetoGlobal')
2.
3.      .factory('Categorias', ['$http', function ($http) {
4.          var Categorias = {};
5.          Categorias.BreadCrumb = function (produtoId) {
6.              return $http({
7.                  method: 'GET',
8.                  url: '/produtos/detalhes/' + produtoId
9.              }).then(function (response) {
10.                 return response.data;
11.             });
12.         };
13.         Categorias.get = function () {
14.             return $http({
15.                 method: 'GET',
16.                 url: 'categorias/'
17.             });
18.         };
19.         return Categorias;
20.     }])
21.     .factory('Favoritos', ['$http', function ($http) {
22.         var Favoritos = {};
23.         Favoritos.get = function (user) {
24.             return $http({
25.                 method: 'GET',
26.                 url: 'utilizadores/favoritos/' + user
27.             });
28.         };
29.         Favoritos.add = function (user, produtoId) {
30.             return $http({
31.                 method: 'POST',
32.                 url: 'utilizadores/addFavoritos/' + user + '/' + produtoId
33.             }).then(function successCallback(response) {
34.                 },
35.                 function errorCallback(response) {
```

```
36.                     console.log(response);
37.                 });
38.         };
39.         return Favoritos;
40.     }])
41.     .factory('Encomendas', ['$http', function ($http) {
42.         var Encomendas = {};
43.         Encomendas.get = function (user) {
44.             return $http({
45.                 method: 'GET',
46.                 url: 'encomendas/' + user
47.             })
48.         };
49.         return Encomendas;
50.     }])
51.     .factory('Carrinho', ['$http', '$rootScope', '$localStorage', function ($http,
    $rootScope, $localStorage) {
52.         var Carrinho = {};
53.         Carrinho.add = function (user, produtoId, carrinho, qty) {
54.             return $http({
55.                 method: 'POST',
56.                 url: 'produtos/add-Carrinho/' + user + '/' + produtoId,
57.                 data: {
58.                     'carrinho': carrinho,
59.                     'qty': qty
60.                 }
61.             }).then(function successCallback(response) {
62.                     if (response.data.errorMessage) {
63.                         alert(response.data.errorMessage);
64.                     } else {
65.                         $localStorage.currentUser.carrinho = response.data;
66.                         $rootScope.currentUser.carrinho = response.data;
67.                     }
68.                 },
69.                 function errorCallback(response) {
70.                     console.log(response);
71.                 });
72.         };
73.         Carrinho.remover = function (url, user, produtoId, carrinho) {
74.             return $http({
75.                 method: 'POST',
76.                 url: url + user + '/' + produtoId,
77.                 data: {
78.                     'carrinho': carrinho
79.                 }
80.             }).then(function successCallback(response) {
81.                     if (response.data.errorMessage) {
82.                         alert(response.data.errorMessage);
83.                     }
84.                     $localStorage.currentUser.carrinho = response.data;
85.                     $rootScope.currentUser.carrinho = response.data;
86.                 },
87.                 function errorCallback(response) {
88.                     console.log(response);
89.                 });
90.         };
91.         return Carrinho;
92.     }])
93.     .factory('Morada', ['$http', '$localStorage', '$rootScope', function ($http, $l
    ocalStorage, $rootScope) {
94.         var Morada = {};
95.         Morada.update = function (user, morada) {
96.             return $http({
97.                 method: 'POST',
98.                 url: 'utilizadores/profile/morada/' + user,
99.                 data: morada
```

```
100.                    }).then(function successCallback(response) {
101.                        $rootScope.currentUser.morada = response.data.morada;
102.                        $localStorage.currentUser.morada = response.data.morada;
103.                    }, function errorCallback(response) {
104.                        console.log(response);
105.                    });
106.                };
107.                return Morada;
108.            }])
109.            .factory('LogOut', ['$http', '$rootScope', '$state', '$localStorage', fu
    nction ($http, $rootScope, $state, $localStorage) {
110.                var LogOut = {};
111.                LogOut.logout = function () {
112.                    return $http({
113.                        method: 'POST',
114.                        url: 'utilizadores/logout'
115.                    }).then(function successCallback(response) {
116.                        delete $localStorage.currentUser;
117.                        $http.defaults.headers.common.Authorization = '';
118.                        $rootScope.currentUser = null;
119.                        $state.go('root.home', {
120.                            reload: true
121.                        });
122.                    }, function errorCallback(response) {
123.                        console.log("Failed logout");
124.                    });
125.                };
126.                return LogOut;
127.            }])
```

## Apêndice A.14 – Ficheiro JavaScript base

```
1.  var app = angular.module('projetoGlobal', ['ui.router', 'ui.bootstrap', 'ui-
    rangeSlider', 'angularUtils.directives.uiBreadcrumbs', 'ngSanitize', 'ngStorage', '
    angular-jwt', 'ngFileUpload', 'stripe', 'ckeditor']);
2.
3.  app.config(['$stateProvider', '$urlRouterProvider', '$locationProvider', '$httpProv
    ider', function ($stateProvider, $urlRouterProvider) {
4.      Stripe.setPublishableKey('pk_test_xMOnMG8VG56uDC2runKgrf3N');
5.      // Estados/caminhos disponivel na aplicacao
6.      $stateProvider
7.          .state('root', {
8.              url: '',
9.              abstract: true,
10.             views: {
11.                 'header': {
12.                     templateUrl: 'partials/header.html',
13.                     controller: 'HeaderCtrl'
14.                 },
15.                 'footer': {
16.                     templateUrl: 'partials/footer.html'
17.                 }
18.             }
19.         })
20.         .state('root.home', {
21.             url: '/',
22.             views: {
23.                 'container@': {
24.                     templateUrl: 'views/home/home.view.html',
25.                     controller: 'HomeCtrl'
26.                 }
27.             },
28.             data: {
```

```
29.                 displayName: 'HomePage'
30.             }
31.         })
32.         .state('root.about', {
33.             url: '/about',
34.             views: {
35.                 'container@': {
36.                     templateUrl: 'views/about/about.view.html'
37.                 }
38.             }
39.         })
40.         .state('root.contatos', {
41.             url: '/contatos',
42.             views: {
43.                 'container@': {
44.                     templateUrl: 'views/contatos/contatos.view.html'
45.                 }
46.             }
47.         })
48.         .state('root.home.produtos', {
49.             url: '^/produtos/categorias/{categoria}/{nomeProduto}',
50.             views: {
51.                 'container@': {
52.                     templateUrl: 'views/produtosSearch/procurarProdutos.view.html',

53.                     controller: 'ProcurarProdutosCtrl'
54.                 }
55.             },
56.             data: {
57.                 displayName: '{{ id }}'
58.             },
59.             resolve: {
60.                 id: function ($stateParams) {
61.                     $stateParams.categoria = $stateParams.categoria || "Todas as Ca
    tegorias";
62.                     return $stateParams.categoria;
63.                 }
64.             }
65.         })
66.         .state('root.home.produtos.subcategoria', {
67.             url: 'subcategorias/{subcategoria}/{nomeProduto}',
68.             views: {
69.                 'container@': {
70.                     templateUrl: 'views/produtosSearch/procurarProdutos.view.html',

71.                     controller: 'ProcurarProdutosCtrl'
72.                 }
73.             },
74.             data: {
75.                 displayName: '{{ id }}'
76.             },
77.             resolve: {
78.                 id: function ($stateParams) {
79.                     return $stateParams.subcategoria;
80.                 }
81.             }
82.         })
83.         .state('root.home.produtoDetalhes', {
84.             url: '^/produto/detalhes/{produtoId}',
85.             views: {
86.                 'container@': {
87.                     templateUrl: 'views/produtoDetalhes/produtoDetalhes.view.html',

88.                     controller: 'ProdutoDetalhesCtrl'
89.                 }
90.             },
```

```
91.             data: {
92.                 displayName: '{{ id.produto.categoria.nome }}'
93.             },
94.             resolve: {
95.                 id: function ($stateParams, Categorias) {
96.                     return Categorias.BreadCrumb($stateParams.produtoId);
97.                 }
98.             }
99.         })
100.             .state('root.profile', {
101.                 url: '/profile',
102.                 views: {
103.                     'container@': {
104.                         templateUrl: 'views/profile/profile.view.html',
105.                         controller: 'ProfileCtrl'
106.                     }
107.                 },
108.                 authenticate: true
109.             })
110.             .state('root.login', {
111.                 url: '/login',
112.                 views: {
113.                     'container@': {
114.                         templateUrl: 'views/login/login.view.html',
115.                         controller: 'LoginCtrl'
116.                     }
117.                 }
118.             })
119.             .state('root.cart', {
120.                 url: '/cart',
121.                 views: {
122.                     'container@': {
123.                         templateUrl: 'views/cart/cart.view.html',
124.                         controller: 'CartCtrl'
125.                     }
126.                 },
127.                 authenticate: true
128.             })
129.             .state('root.checkout', {
130.                 url: '/checkout',
131.                 views: {
132.                     'container@': {
133.                         templateUrl: 'views/checkout/checkout.view.html',
134.                         controller: 'CheckoutCtrl'
135.                     }
136.                 },
137.                 authenticate: true
138.             })
139.             .state('root.checkoutResumo', {
140.                 url: '/checkoutResumo/{encomendaId}',
141.                 views: {
142.                     'container@': {
143.                         templateUrl: 'views/checkout/checkoutResumo.view.html',
144.                         controller: 'CheckoutResumoCtrl'
145.                     }
146.                 },
147.                 authenticate: true
148.             })
149.             .state('root.encomendasDetalhes', {
150.                 url: '/encomendasDetalhes/{encomendaId}',
151.                 views: {
152.                     'container@': {
153.                         templateUrl: 'views/encomendasDetalhes/encomendasDetalhe
    s.view.html',
154.                         controller: 'EncomendasDetalhesCtrl'
```

```
155.                     }
156.                 },
157.                 authenticate: true
158.             })
159.             .state('root.admin', {
160.                 url: '/admin',
161.                 views: {
162.                     'container@': {
163.                         templateUrl: 'views/admin/admin.view.html',
164.                         controller: 'AdminCtrl'
165.                     }
166.                 },
167.                 admin: true
168.             })
169.             .state('root.admin.criarProduto', {
170.                 url: '/admin/criarProduto',
171.                 views: {
172.                     'container@': {
173.                         templateUrl: 'views/admin/adminCriarProduto.view.html',
174.                         controller: 'AdminCriarProdutoCtrl'
175.                     }
176.                 },
177.                 admin: true
178.             })
179.             .state('root.admin.editarProduto', {
180.                 url: '/admin/editarProduto/{produtoID}',
181.                 views: {
182.                     'container@': {
183.                         templateUrl: 'views/admin/adminEditarProduto.view.html',
184.                         controller: 'AdminEditarProdutoCtrl'
185.                     }
186.                 },
187.                 admin: true
188.             })
189.             .state('root.admin.encomendaDetalhes', {
190.                 url: '/admin/encomendaDetalhes/{encomendaID}',
191.                 views: {
192.                     'container@': {
193.                         templateUrl: 'views/admin/adminEncomendaDetalhes.view.ht
    ml',
194.                         controller: 'AdminEncomendaDetalhesCtrl'
195.                     }
196.                 },
197.                 admin: true
198.             });
199.         $urlRouterProvider.otherwise('/');
200.     }]);
201.
202.     // Ao iniciar a aplicacao
203.     app.run(function ($rootScope, $state, $stateParams, $localStorage, $http) {
204.         // Ao mudar de estado este vai para o topo da pagina
205.         $rootScope.$on('$stateChangeSuccess', function (event, next) {
206.             document.body.scrollTop = document.documentElement.scrollTop = 0;
207.             if (next.authenticate) {
208.                 $http.get('/utilizadores/loggedin')
209.                     .then(function successCallback(user) {
210.                     }, function errorCallback(response) {
211.                         event.preventDefault();
212.                         $rootScope.currentUser = null;
213.                         $localStorage.currentUser = null;
214.                         $state.go('root.login');
215.                     });
216.             }
```

```
217.                else if (next.admin) {
218.                    $http.get('/utilizadores/loggedin')
219.                        .then(function successCallback(user) {
220.                            $http({
221.                                method: 'GET',
222.                                url: 'utilizadores/loggedin/role'
223.                            }).then(function successCallback(response) {
224.                            }, function errorCallback(response) {
225.                                $state.go('root.home');
226.                            });
227.                        }, function errorCallback(response) {
228.                            $state.go('root.login');
229.                        });
230.                }
231.            });
232.            // Deixa o utilizador logado mesmo quando faz refresh
233.            if ($localStorage.currentUser) {
234.                $http.defaults.headers.common.Authorization = 'Bearer ' + $localStor
    age.currentUser.token;
235.                $rootScope.currentUser = $localStorage.currentUser;
236.                $http.get('/utilizadores/loggedin')
237.                    .then(function successCallback(user) {
238.                    }, function errorCallback(response) {
239.                        event.preventDefault();
240.                        $rootScope.currentUser = null;
241.                        $localStorage.currentUser = null;
242.                        $state.go('root.home');
243.                    });
244.            }
245.        });
```

## Apêndice A.15 – HTML da view Header

```
1.  <script type="text/ng-template" id="searchInputTemplate.html">
2.      <a class='searchInputTemplate'>
3.          <img ng-
    src="images/uploads/produtos/{{match.model.imagem.file.filename}}" width="16"> {{ma
    tch.model.nome}}
4.      </a>
5.  </script>
6.
7.  <nav class="navbar header">
8.      <div class="container-fluid">
9.          <div class="row top-Header">
10.             <div class="col-md-2 col-xs-2 header-Logo">
11.                 <div class="col-md-12">
12.                     <a ui-sref="root.home" ui-sref-
    opts="{reload: true}"><img src="images/Logo.png" alt=""></a>
13.                 </div>
14.             </div>
15.             <div class="col-md-3 col-xs-3 header-Search">
16.                 <div class="input-group">
17.                     <div class="input-group-btn header-Categories dropdown">
18.                         <button type="button" class="btn btn-default dropdown-
    toggle1 header-Button-Categories" id="dropdownMenu1" data-toggle="dropdown" aria-
    haspopup="true" aria-expanded="false">
19.                             <span id="header-
    Category">{{categoriaSelecionada}}</span><span class="caret"></span>
20.                             </button>
21.                         <ul class="dropdown-menu" aria-
    labelledby="dropdownMenu1" id="dropdown-menu-Categories">
22.                             <li ng-
    click="select('Todas as Categorias')"><a>Todas as Categorias</a></li>
```

```
23.                         <li ng-
    repeat="categoria in categorias | orderBy: 'nome'" ng-
    click="select(categoria)"><a>{{categoria.nome}}</a></li>
24.                     </ul>
25.                 </div>
26.                 <!-- /btn-group -->
27.                 <form ng-
    submit="procurarProdutoSubmit(categoriaSelecionada, procurarProdutos)">
28.                     <input type="text" ng-
    model="procurarProdutos" placeholder="Procurar" uib-
    typeahead="nome for nome in procurarProduto($viewValue)" typeahead-template-
    url="searchInputTemplate.html" typeahead-min-length='2' typeahead-focus-
    first='false' typeahead-on-select="IrParaProdutoDetalhes($item)"
29.                         class="header-Search-Input" ng-class="{'header-Search-
    Input-Radius' : !procurarProdutos.length}" id="header-Search-
    Input" autocomplete="off">
30.                 </form>
31.                 <div class="input-group-btn">
32.                     <button ng-
    hide="!procurarProdutos.length " class="btn header-Search-Button" ng-
    click="procurarProdutoSubmit(categoriaSelecionada, procurarProdutos)"><span class="
    glyphicon glyphicon-search" aria-hidden="true"></span></button>
33.                 </div>
34.             </div>
35.             <!-- /input-group -->
36.         </div>
37.         <div class="col-md-3 col-xs-3 navbar-right header-Account ">
38.             <div class="col-md-6 col-xs-3 header-Account-SignIn">
39.                 <div class="header-Account-SignIn-Text">
40.                     <a ng-hide='currentUser' ng-
    click="IrPara('root.login')" class="header-Account-Name ">Ola.</a>
41.                     <a ng-show='currentUser' class="header-Account-
    Name bemvido">Bem vindo,</a>
42.                 </div>
43.                 <div class=" header-myAccount dropdown" ng-
    mouseover="hover=true" ng-mouseleave="hover=false">
44.                     <button ng-hide='currentUser' class="header-myAccount-
    Button dropdown-toggle" ng-
    class="{headermyAccountDropdownActive: hover}" id="dropdownMenu2">Login</button>
45.                     <button ng-show='currentUser' class="header-myAccount-
    Button dropdown-toggle" ng-
    class="{headermyAccountDropdownActive: hover}" id="dropdownMenu2" ng-
    click="IrPara('root.profile')">{{currentUser.username}}</button>
46.                     <span class="caret"></span>
47.                     <ul class="dropdown-menu dropdown-content header-myAccount-
    Dropdown" aria-labelledby="dropdownMenu2">
48.                         <li ng-hide='currentUser' ng-
    click="IrPara('root.login')"><a>Iniciar Sessão</a></li>
49.                         <li ng-show="currentUser.role == 'admin'" ng-
    click="IrPara('root.admin')"><a>Painel de Administrador</a></li>
50.                         <li ng-hide='!currentUser' ng-
    click="IrPara('root.profile')"><a>Minha Conta</a></li>
51.                         <li ng-hide='!currentUser' ng-
    click="IrPara('root.cart')"><a>Meu Carrinho</a></li>
52.                         <li ng-hide='!currentUser' ng-
    click="Logout()"><a>Sair</a></li>
53.                     </ul>
54.                 </div>
55.             </div>
56.             <div class="col-md-6 col-xs-3 header-Account-ShoppingCart">
57.                 <span class="glyphicon glyphicon-shopping-cart carrinho" ng-
    click="IrPara('root.cart')"></span>
58.                 <i ng-hide='!currentUser' class="badge hidden-
    xs">{{ currentUser.carrinho.totalQty }}</i>
59.             </div>
60.         </div>
```

```
61.         </div>
62. </nav>
63. <div class="row bottom-Header">
64.     <div class="col-md-2 col-md-offset-2">
65.         <div class="input-group-btn dropdown bottom-Header-Dropdown" ng-
    mouseover="hover1=true" ng-mouseleave="hover1=false">
66.             <button type="button" class="btn btn-default dropdown-toggle bottom-
    Header-Dropdown-Button" id="bottomDropdownMenu" ng-
    class="{bottomDropdownMenuActive: hover1}" data-toggle="dropdown" aria-
    haspopup="true" aria-
    expanded="false">Categorias<span class="caret"></span> </button>
67.             <ul class="dropdown-menu  dropdown-content bottom-Dropdown-Menu" aria-
    labelledby="dropdownMenu1">
68.                 <li class="dropdown-submenu" ng-
    repeat="categoriaSubMenu in categorias | orderBy: 'nome'">
69.                     <a ui-
    sref="root.home.produtos({ categoria: categoriaSubMenu.nome, nomeProduto: ''})" ui-
    sref-opts="{reload: true}">{{categoriaSubMenu.nome}}</a>
70.                     <ul class="dropdown-menu">
71.                         <li ng-
    repeat="subcategoriasSubMenu in categoriaSubMenu.subCategorias | orderBy: 'nome'" n
    g-
    click='IrParaSubCategoria(categoriaSubMenu.nome ,subcategoriasSubMenu.nome)'><a>{{s
    ubcategoriasSubMenu.nome}}</a></li>
72.                     </ul>
73.                 </li>
74.             </ul>
75.         </div>
76.     </div>
77. </div>
```

## Apêndice A.16 – HTML da view Footer

```
1.  <div class="footer-distributed">
2.      <div class="footer-left">
3.          <img src="images/Logo.png" alt="">
4.          <p class="footer-links">
5.              <a href="#">Home</a> ·
6.              <a ui-sref="root.about" ui-sref-opts="{reload: true}">Sobre</a> ·
7.              <a ui-sref="root.contatos" ui-sref-opts="{reload: true}">Contatos</a>
8.          </p>
9.          <p class="footer-company-name">Market Place © 2016</p>
10.     </div>
11.     <div class="footer-center">
12.         <div><i class="fa fa-map-
    marker"></i><p><span>Avenida da Liberdade</span> Lisboa, Portugal</p></div>
13.         <div><i class="fa fa-phone"></i><p>210000333</p></div>
14.         <div><i class="fa fa-
    envelope"></i><p><a href="mailto:fabioanselmo1994@gmail.com">fabioanselmo1994@gmail
    .com</a></p></div>
15.     </div>
16.     <div class="footer-right">
17.         <p class="footer-company-
    about"><span>Sobre a Empresa</span> Lorem ipsum dolor sit amet, consectateur adispi
    cing elit. Fusce euismod convallis velit, eu auctor lacus vehicula sit amet.</p>
18.         <div class="footer-icons">
19.             <a href="#"><i class="fa fa-facebook"></i></a>
20.             <a href="#"><i class="fa fa-twitter"></i></a>
21.             <a href="#"><i class="fa fa-google-plus"></i></a>
22.         </div>
23.     </div>
24. </div>
```

# **Apêndice A.17** – HTML da view do estado 'root.home'

```
1.  <!-- Images Carousel -->
2.  <div class="row carousel-div" id="slides_control">
3.      <div id="carousel-example-generic" class="carousel slide" data-
    ride="carousel">
4.          <div  ng-show="!semProdutoBanner" class="carousel-inner" role="listbox">
5.              <div class="item {{img.active}}" ng-
    repeat="img in produtosBanner"><img ng-
    src="images/uploads/produtosBanner/{{img.nome}}" alt="..."></div>
6.          </div>
7.          <div  ng-show="semProdutoBanner" class="carousel-inner" role="listbox">
8.              <div class="item active"><img ng-
    src="images/uploads/produtosBanner/produtoSemImagem.png" alt="..."></div>
9.          </div>
10.         <!-- Controls -->
11.         <a class="left carousel-control" ng-href="#carousel-example-
    generic" role="button" data-slide="prev" ng-click='$event.preventDefault()'>
12.             <span class="glyphicon glyphicon-chevron-left" aria-
    hidden="true"></span>
13.             <span class="sr-only">Previous</span>
14.         </a>
15.         <a class="right carousel-control" ng-href="#carousel-example-
    generic" role="button" data-slide="next" ng-click='$event.preventDefault()'>
16.             <span class="glyphicon glyphicon-chevron-right" aria-
    hidden="true"></span>
17.             <span class="sr-only">Next</span>
18.         </a>
19.     </div>
20. </div>
21. <!-- FIM Images Carousel -->
22. <!-- Product Carousel -->
23. <div class="row product-Slider-Controllers">
24.     <div class="row">
25.         <div class="col-md-9">
26.             <div class="col-md-6 col-sm-6">
27.                 <h3>Produtos em Destaque</h3>
28.             </div>
29.         </div>
30.         <div class="col-md-3">
31.             <!-- Controls -->
32.             <div class="controls pull-right">
33.                 <a class="left fa fa-chevron-left btn btn-success" ng-
    href="#product-Slider" data-slide="prev" ng-click='$event.preventDefault()'></a>
34.                 <a class="right fa fa-chevron-right btn btn-success" ng-
    href="#product-Slider" data-slide="next" ng-click='$event.preventDefault()'></a>
35.             </div>
36.         </div>
37.     </div>
38.     <div id="product-Slider" class="carousel slide" data-ride="carousel1">
39.         <div class="item itemHome" ng-show="semProdutoDestaque">
40.             <div class="row">
41.                 <ul class="product-Grid">
42.                     <li class="semProdutosHomeH3"><h3>Nao encontrou Produtos</h3></
    li>
43.                 </ul>
44.             </div>
45.         </div>
46.         <div class="carousel-inner" id="product-Slider-Home">
47.             <div class="item" ng-class="{active:$first}" ng-
    repeat="produtos in produtosDestaque">
48.                 <div class="row">
49.                     <div class="col-sm-12 col-md-12 col-lg-3 col-xs-12" ng-
    repeat="produto in produtos">
```

```
50.                         <div class="col-item">
51.                             <div class="photo">
52.                                 <a ui-
    sref="root.home.produtoDetalhes({ produtoId:produto._id })" ui-sref-
    opts="{reload: true}">
53.                                     <img ng-
    src="images/uploads/produtos/{{produto.imagem.file.filename}}" class="img-
    responsive" alt="a" />
54.                                 </a>
55.                             </div>
56.                             <div class="info">
57.                                 <div class="row">
58.                                     <div class="price col-sm-12 col-md-12 col-xs-
    12 col-lg-12">
59.                                         <h5>{{produto.nome}}</h5>
60.                                         <h5 class="price-text-
    color">{{produto.preco}} €</h5>
61.                                     </div>
62.                                 </div>
63.                             </div>
64.                             <div class=" separator clear-left">
65.                                 <p class="btn-add addtocart_button2">
66.                                     <button value="Add to Cart " title="Add to Cart
    " class="addtocart-button cart-click" ng-
    click="addCarrinho(produto._id);$event.preventDefault()">
67.                                         <span class="glyphicon glyphicon-shopping-
    cart" aria-hidden="true"></span>  Comprar</button>
68.                                 </p>
69.                                 <p class="btn-
    add addtocart_button2"><button class="btn btn-lg btn-brand btn-full-
    width addtocart-favoritos-Detalhes btn-fav " ng-
    click="addFavoritos(produto._id);$event.preventDefault() "><span class="glyphicon g
    lyphicon-heart " aria-hidden="true "></span> Favoritos</button></p>
70.                             </div>
71.                             <div class="clearfix "></div>
72.                         </div>
73.                     </div>
74.                 </div>
75.             </div>
76.         </div>
77.     </div>
78. </div>
```

## **Apêndice A.18** – Controlador do estado 'root.home'

```
1.  angular.module('projetoGlobal')
2.      .controller('HomeCtrl', ['$scope', '$state', '$http', '$rootScope', '$localStor
    age', 'Favoritos', 'Carrinho', function ($scope, $state, $http, $rootScope, $localS
    torage, Favoritos, Carrinho) {
3.          $scope.produtosDestaque = [];
4.          $scope.semProdutoDestaque = false;
5.          $scope.semProdutoBanner = false;
6.          iniciarCarousel();
7.          iniciarProductSlider();
8.
9.          // Iniciar Carousel do Image Banner
10.         function iniciarCarousel() {
11.             $('#carousel-example-generic').carousel({interval: 5000});
12.             $http({
13.                 method: 'GET',
14.                 url: 'produtosBanner',
15.             }).then(function (response) {
16.                 if (response.data.length == 0) $scope.semProdutoBanner = true;
```

```
                else {
                    $scope.produtosBanner = response.data;
                    $scope.produtosBanner[0].active = "active";
                }
            });
        };

        // Iniciar o Produto em Destaque Slider
        function iniciarProductSlider() {
            $('#product-Slider').carousel({
                interval: 0
            });
            $http({
                method: 'GET',
                url: 'produtos/destaque'
            }).then(function (response) {
                if (response.data.length == 0) $scope.semProdutoDestaque = true;
                for (var t = 0; t < response.data.length; t += 4) {
                    $scope.produtosDestaque.push(response.data.slice(t, t + 4));
                }
            });
        };

        $scope.addFavoritos = function (productID) {
            if ($rootScope.currentUser) Favoritos.add($rootScope.currentUser._id, p
roductID);
            else $state.go('root.login', {reload: true});
        };

        $scope.addCarrinho = function (productID) {
            if ($rootScope.currentUser) Carrinho.add($rootScope.currentUser._id, pr
oductID, $localStorage.currentUser.carrinho, 1);
            else $state.go('root.login', {reload: true});
        };
    }]);
```

## **Apêndice A.19** – HTML da view dos estados 'root.home.produtos' e 'root.home.produtos.subcategoria'

```
<ui-breadcrumbs displayname-property="data.displayName" template-
url="bower_components/angular-utils-ui-breadcrumbs/uiBreadcrumbs.tpl.html"></ui-
breadcrumbs>
<div class="row ">
    <div class="col-sm-12 col-xs-12 col-lg-3 side-Menu-Container">
        <div class="side-Menu">
            <div class="title-Menu"><h3>Filtros</h3></div>
            <div ng-hide='!TodasAsCategorias' class="side-Menu-Filters col-sm-
4 col-xs-4 col-lg-12">
                <h4>Categorias</h4>
                <ul>
                    <form>
                        <ul class="category-list">
                            <li ng-
repeat="categoria in categorias | orderBy : 'nome'">
                                <a ng-
click='filtroCategoria(categoria.nome)' id="{{categoria.nome}}">{{categoria.nome}}<
/a>
                            </li>
                        </ul>
                    </form>
                </ul>
            </div>
```

```
18.             <div ng-hide='subCategoriaHide' class="side-Menu-Filters col-sm-4 col-
    xs-4 col-lg-12">
19.                 <h4>Sub-Categorias</h4>
20.                 <ul>
21.                     <form>
22.                         <ul class="category-list">
23.                             <li ng-
    repeat="subcategoria in subCategorias | orderBy : 'nome'">
24.                                 <a ng-
    click='filtroSubCategoria(subcategoria.nome)' id="{{subcategoria.nome}}">{{subcateg
    oria.nome}}</a>
25.                             </li>
26.                         </ul>
27.                     </form>
28.                 </ul>
29.             </div>
30.             <div ng-hide='MarcasHide' class="side-Menu-Filters col-sm-4 col-xs-
    3 col-lg-12">
31.                 <h4>Marcas</h4>
32.                 <ul>
33.                     <form>
34.                         <li ng-repeat="marca in Marcas | orderBy: ''">
35.                             <input id="{{marca}}" class="" type="checkbox" name="ch
    eckbox" ng-click="toggle(marca, marcaSelecionada)">
36.                             <label for="{{marca}}" class="">{{marca}}</label>
37.                         </li>
38.                     </form>
39.                 </ul>
40.             </div>
41.             <div ng-hide='PrecoHide' class="side-Menu-Filters col-sm-4 col-xs-
    5 col-lg-12">
42.                 <h4>Preço</h4>
43.                 <ul>
44.                     <li class="spanPrecos">
45.                         <span>Mínimo</span>
46.                         <span>Máximo</span>
47.                     </li>
48.                     <li>
49.                         <input type="text" class="price-Filter" ng-
    model="precoSlider.min">
50.                         <input type="text" class="price-Filter" ng-
    model="precoSlider.max">
51.                     </li>
52.                     <li>
53.                         <div range-slider min="0" max="PrecoMaxFixo" model-
    min="precoSlider.min" model-max="precoSlider.max" disabled="false"></div>
54.                     </li>
55.                 </ul>
56.             </div>
57.             <div ng-hide='!PrecoHide' class="side-Menu-Filters col-sm-4 col-xs-
    5 col-lg-12">
58.             </div>
59.         </div>
60.     </div>
61.     <div class="col-lg-9 col-sm-12 col-xs-12 side-Menu menu-Products">
62.         <div class="title-Menu">
63.             <div class="col-sm-9 col-lg-4 col-xs-9">
64.                 <h3>Produtos</h3>
65.             </div>
66.             <div class="col-xs-1 col-xs-offset-1 col-sm-1 col-sm-offset-1 col-lg-
    1 col-lg-offset-1 navbar-right select-Filter">
67.                 <select name="filter" ng-model="filter" ng-init="filter='nome'">
68.                         <option value="nome">Nome</option>
69.                         <option value="preco">Preço</option>
70.                         <option value="marca">Marca</option>
71.                     </select>
```

```
72.             </div>
73.             <div class="col-sm-1 col-lg-1 col-xs-1 navbar-
    right orderByDirection">
74.                 <span ng-show='OrderByShow' ng-
    click='toggleOrderBy()' class="glyphicon glyphicon-arrow-up" aria-
    hidden="true"></span>
75.                 <span ng-show='!OrderByShow' ng-
    click='toggleOrderBy()' class="glyphicon glyphicon-arrow-down" aria-
    hidden="true"></span>
76.             </div>
77.             <div class="col-sm-1 col-lg-1 col-xs-1 navbar-right select-Filter-
    Title">
78.                 <h5>Ordenar Por:</h5>
79.             </div>
80.         </div>
81.         <div class="item procurarProdutos">
82.             <div class="row">
83.                 <ul class="product-Grid">
84.                     <li ng-
    repeat="item in produtosFiltrados = (produto | FiltroMarca:marcaSelecionada | Filtr
    oPreco: precoSlider) | startFrom:(currentPage-
    1)*pageSize | limitTo:pageSize | orderBy : filter :OrderByShow">
85.                         <div class="col-sm-12 col-lg-3 col-xs-12 ">
86.                             <div class="col-item">
87.                                 <div class="photo">
88.                                     <a ng-click='produtoDetalhes(item._id) '>
89.                                         <img ng-
    src="images/uploads/produtos/{{item.imagem.file.filename}}" class="img-
    responsive" />
90.                                     </a>
91.                                 </div>
92.                                 <div class="info">
93.                                     <div class="row">
94.                                         <div class="price col-sm-12  col-xs-12 col-
    lg-12">
95.                                             <h5>{{item.nome}}</h5>
96.                                             <h5 class="price-text-
    color">{{item.preco}} €</h5>
97.                                         </div>
98.                                     </div>
99.                                 </div>
100.                                    <div class=" separator clear-left">
101.                                        <p class="btn-add addtocart_button2 ">
102.                                            <button value="Add to Cart " title="
    Add to Cart " class="addtocart-button3 cart-click " ng-
    click="addCarrinho(item._id);$event.preventDefault()">
103.                                                <span class="glyphicon glyphicon
    -shopping-cart" aria-hidden="true"></span>  Comprar</button>
104.                                            </button>
105.                                        </p>
106.                                        <p class="btn-add addtocart_button2">
107.                                            <button class="btn btn-lg btn-
    brand btn-full-width addtocart-favoritos-Detalhes btn-fav " ng-
    click="addFavoritos(item._id);$event.preventDefault() "><span class="glyphicon glyp
    hicon-heart " aria-hidden="true "></span></button>
108.                                        </p>
109.                                    </div>
110.                                    <div class="clearfix ">
111.                                    </div>
112.                                </div>
113.                            </div>
114.                        </li>
115.                    </ul>
116.                </div>
117.            </div>
118.            <div class="item semProduto" ng-show="semProduto">
```

```
119.                    <div class="row">
120.                        <ul class="product-Grid">
121.                            <li class="semProdutosH3"><h3>Não encontrou Produtos</h3
    ></li>
122.                        </ul>
123.                    </div>
124.                </div>
125.                <div class="row pagination-Container">
126.                    <div class="col-sm-12  col-xs-12 col-lg-12 pagination ">
127.                        <ul uib-pagination total-
    items="produtosFiltrados.length" ng-change='goToTop() ' ng-
    model="currentPage" items-per-page="pageSize" max-size='5 ' class="pagination-
    sm" data-boundary-links="true" force-ellipses="true" next-text="Próximo" previous-
    text="Anterior" first-text="Início" last-text="Fim"></ul>
128.                    </div>
129.                </div>
130.            </div>
131.        </div>
```

## **Apêndice A.20** – Controlador dos estados 'root.home.produtos' e 'root.home.produtos.subcategoria'

```
1.  angular.module('projetoGlobal')
2.
3.      .controller('ProcurarProdutosCtrl', ['$scope', '$state', '$stateParams', '$http
    ', '$rootScope', '$localStorage', 'Carrinho', 'Favoritos', function ($scope, $state
    , $stateParams, $http, $rootScope, $localStorage, Carrinho, Favoritos) {
4.          $scope.currentPage = 1;
5.          $scope.pageSize = 12;
6.          $scope.produto = [];
7.          $scope.produtosMarcas = [];
8.          $scope.produtosPreco = [];
9.          $scope.Marcas = [];
10.         $scope.MarcasHide = false;
11.         $scope.PrecoHide = false;
12.         $scope.PrecoMax = 0;
13.         $scope.PrecoMin = 0;
14.         $scope.categorias = [];
15.         $scope.subCategorias = [];
16.         $scope.pages = [];
17.         $scope.semProduto = false;
18.         $scope.TodasAsCategorias = false;
19.         $scope.subCategoriaHide = false;
20.         $scope.categoriasSelecionadas = [];
21.         $scope.filtro = {};
22.         $scope.marcaSelecionada = [];
23.         $scope.OrderByShow = true;
24.         GetCategoria($stateParams.categoria);
25.
26.         if ($stateParams.categoria == "Todas as Categorias") {
27.             $scope.subCategoriaHide = true; // Faz hide ao menu das subCategorias
28.             $scope.TodasAsCategorias = true; // Faz hide ao menu das Categorias
29.         } else {
30.             $scope.TodasAsCategorias = false; // Faz show ao menu das Categorias
31.         }
32.
33.         // Se estiver no state das SubCategorias
34.         if ($state.current.name == 'root.home.produtos.subcategoria') {
35.             GetProdutos('produtos/subCategoria/procurar/' + $stateParams.subcategor
    ia + '/' + $stateParams.nomeProduto);
36.             $scope.subCategoriaHide = true; // Faz hide ao menu das subCategorias
37.         } else {
```

```
38.             // Se o nome do produto for '' entao procura so pela categoria (ou seja
    , vem das categorias do bottom header)
39.             if ($stateParams.nomeProduto === '') {
40.                 GetProdutos('produtos/procurar/' + $stateParams.categoria);
41.             } else {
42.                 GetProdutos('produtos/procurar/' + $stateParams.categoria + '/' + $
    stateParams.nomeProduto);
43.             }
44.             // No state das Categorias ao clickar numa subCategoria, este vai ao o
    state da subcategoria clickada
45.             $scope.filtroSubCategoria = function (subCategoria) {
46.                 $state.go('root.home.produtos.subcategoria', {
47.                     subcategoria: subCategoria,
48.                     nomeProduto: $stateParams.nomeProduto
49.                 }, {reload: true});
50.             }
51.         }
52.
53.         // Buscar Categorias e SubCategorias
54.         function GetCategoria(categoria) {
55.             $http({
56.                 method: 'GET',
57.                 url: '/categorias/' + categoria
58.             }).then(function (response) {
59.                 $scope.subCategoria = true;
60.                 angular.forEach(response.data, function (categorias) {
61.                     $scope.categorias.push(categorias);
62.                     angular.forEach(categorias.subCategorias, function (subcategori
    as) {
63.                         $scope.subCategorias.push(subcategorias);
64.                     });
65.                 });
66.             });
67.         };
68.
69.         // Vai buscar os produtos
70.         function GetProdutos(url) {
71.             $http({
72.                 method: 'GET',
73.                 url: url
74.             }).then(function (response) {
75.                 // Se nao houver nenhum produto
76.                 if (response.data.length === 0) {
77.                     $scope.semProduto = true;
78.                     $scope.MarcasHide = true;
79.                     $scope.PrecoHide = true;
80.                     $scope.subCategoriaHide = true;
81.                 } else {
82.                     angular.forEach(response.data, function (produtos) {
83.                         $scope.produto.push(produtos);
84.                         $scope.produtosMarcas.push(produtos.marca);
85.                         $scope.produtosPreco.push(produtos.preco);
86.                     });
87.                     VerificarMarcas();
88.                     VerificarPrecos();
89.                 }
90.             });
91.         }
92.
93.         // Ao mudar de Pagina ir para o topo
94.         $scope.goToTop = function () {
95.             document.body.scrollTop = document.documentElement.scrollTop = 0;
96.         };
97.
98.         //Ir para a pagina de detalhes ao clickar no produto
99.         $scope.produtoDetalhes = function (id) {
```

```
100.                $state.go('root.home.produtoDetalhes', {produtoId: id}, {reload:
    true});
101.            }
102.
103.            //Ir para a pagina de detalhes ao clickar no produto
104.            $scope.filtroCategoria = function (categoria) {
105.                $state.go('root.home.produtos', {
106.                    categoria: categoria,
107.                    nomeProduto: $stateParams.nomeProduto
108.                }, {reload: true});
109.            };
110.
111.            function VerificarMarcas() {
112.                $.each($scope.produtosMarcas, function (i, el) {
113.                    //se nao houver a marca (el) no array marcas entao faz push
    senao nao faz push
114.                    if ($.inArray(el, $scope.Marcas) === -
    1) $scope.Marcas.push(el);
115.                });
116.            }
117.
118.            // Vai buscar o preco max e min dos produtos e define-os no slider
119.            function VerificarPrecos() {
120.                $scope.PrecoMinFixo = Math.min.apply(Math, $scope.produtosPreco)
     - 40;
121.                $scope.PrecoMaxFixo = Math.max.apply(Math, $scope.produtosPreco)
     + 40;
122.                $scope.precoSlider = {
123.                    min: 0,
124.                    max: $scope.PrecoMaxFixo
125.                };
126.            }
127.
128.            // when user clicks on checkbox, change selected list
129.            $scope.toggle = function (categoria, list) {
130.                var idx = list.indexOf(categoria);
131.                if (idx > -1) list.splice(idx, 1);
132.                else list.push(categoria);
133.            };
134.
135.            // Toggle para o OrderBy Directions
136.            $scope.toggleOrderBy = function () {
137.                $scope.OrderByShow = !$scope.OrderByShow;
138.            };
139.
140.            $scope.addFavoritos = function (produtoID) {
141.                if ($rootScope.currentUser) Favoritos.add($rootScope.currentUser
    ._id, produtoID);
142.                else $state.go('root.login', {reload: true});
143.            };
144.
145.            $scope.addCarrinho = function (productID) {
146.                if ($rootScope.currentUser) Carrinho.add($rootScope.currentUser.
    _id, productID, $localStorage.currentUser.carrinho, 1);
147.                else $state.go('root.login', {reload: true});
148.            };
149.        }]);
```

# Apêndice A.21 – HTML da view do estado 'root.home.produtoDetalhes'

```
1.  <ui-breadcrumbs displayname-property="data.displayName" template-
    url="bower_components/angular-utils-ui-breadcrumbs/uiBreadcrumbs.tpl.html"></ui-
    breadcrumbs>
2.  <div class="productDetailDiv">
3.      <div class="row">
4.          <div class="col-lg-6 col-md-7 col-sm-12">
5.              <div class="col-lg-offset-1 col-md-offset-1 col-sm-offset-1 col-xs-
    offset-1 col-lg-10  col-md-10 col-sm-12 col-xs-10 imageDetailContainer ">
6.                  <img ng-
    src="images/uploads/produtos/{{produto.imagem.file.filename}} " alt="Imagem do Prod
    uto " class="image-responsive imageDetail"/>
7.              </div>
8.          </div>
9.          <div class="row visible-xs"></div>
10.         <div class="col-lg-6 col-md-5 col-sm-12 ProdutoDetailContainer">
11.             <div class="row ">
12.                 <div class="col-lg-12 col-md-12 col-sm-12 ">
13.                     <h1>{{produto.nome}}</h1>
14.                 </div>
15.             </div>
16.             <div class="row ">
17.                 <div class="col-lg-12 col-md-12 col-sm-12 produtlabelsDetalhes ">
18.                     <span class="label label-
    primary ">{{produto.categoria.nome}}</span>
19.                     <span ng-show="stock == 'Disponivel'" class="label label-
    success ">Disponível</span>
20.                     <span ng-show="stock == 'Stock Limitado'" class="label label-
    warning ">Stock Limitado</span>
21.                     <span ng-show="stock == 'Indisponivel'" class="label label-
    danger ">Indisponível</span>
22.                 </div>
23.             </div>
24.             <div class="row ">
25.                 <div class="col-md-12 col-sm-12 bottom-rule ">
26.                     <h2 class="product-price ">Preço: {{produto.preco}} €</h2>
27.                 </div>
28.             </div>
29.             <br>
30.             <div class="row add-to-cart ">
31.                 <div class="col-md-7 col-sm-12 product-qty ">
32.                     <span class="btn btn-default btn-lg btn-qty" ng-
    click="incrementarQty()">
33.                                 <span class="glyphicon glyphicon-plus " aria-
    hidden="true"></span>
34.                     </span>
35.                     <input class="btn btn-default btn-lg btn-qty " ng-
    model='quantidadeProduto'/>
36.                     <span class="btn btn-default btn-lg btn-qty" ng-
    click="decrementarQty()">
37.                                 <span class="glyphicon glyphicon-minus " aria-
    hidden="true "></span>
38.                     </span>
39.                 </div>
40.             </div>
41.             <br>
42.             <div class="row ">
43.                 <div class="col-md-7 col-sm-12 ">
44.                     <button class="btn btn-lg btn-brand btn-full-width addtocart-
    button-Detalhes " ng-
```

```
    click="addCarrinho(produto._id);$event.preventDefault()"><span class="glyphicon gly
    phicon-shopping-cart" aria-hidden="true"></span> Comprar  </button>
45.                     </button>
46.                     <button class="btn btn-lg btn-brand btn-full-width addtocart-
    favoritos-Detalhes " ng-
    click="addFavoritos(produto._id);$event.preventDefault() "><span class="glyphicon g
    lyphicon-heart " aria-hidden="true "></span> Favoritos </button>
47.                 </div>
48.             </div>
49.         </div>
50.     </div>
51.     <div class="row ">
52.         <div class="tab-product-Detail ">
53.             <!-- Nav tabs -->
54.             <ul class="nav nav-tabs " role="tablist ">
55.                 <li role="presentation " class="active ">
56.                     <a data-target="#description" aria-
    controls="description" role="tab" data-toggle="tab">Descrição do Produto</a>
57.                 </li>
58.                 <li role="presentation ">
59.                     <a data-target="#reviews" aria-
    controls="reviews" role="tab" data-toggle="tab">Comentários</a>
60.                 </li>
61.             </ul>
62.             <!-- Tab panes -->
63.             <div class="tab-content tab-detail-product ">
64.                 <div role="tabpanel" class="tab-pane active" id="description">
65.                     <div class="container ">
66.                         <p class="top-10 " ng-bind-html="produto.descricao">
67.                         </p>
68.                     </div>
69.                 </div>
70.                 <div role="tabpanel" class="tab-pane" id="reviews">
71.                     <div class="row ">
72.                         <form class="form-
    horizontal " id="commentForm" role="form" ng-submit="adicionarComentario()">
73.                             <div class="form-group ">
74.                                 <label for="email " class="col-sm-2 control-
    label ">Comentário</label>
75.                                 <div class="col-sm-10 ">
76.                                     <textarea class="form-
    control " name="TextAreaComentario" ng-
    model="TextAreaComentario" id="addComment" rows="5"></textarea>
77.                                 </div>
78.                             </div>
79.                             <div class="form-group">
80.                                 <div class="col-sm-offset-2 col-sm-10">
81.                                     <button class="btn btn-success btn-circle text-
    uppercase" type="submit" id="submitComment"><span class="glyphicon glyphicon-
    send"></span> Adicionar Comentário</button>
82.                                 </div>
83.                             </div>
84.                         </form>
85.                     </div>
86.                     <div class="row">
87.                         <ul class="media-list">
88.                             <li class="media"
89.                                 ng-
    repeat=" comentario in comentariosFiltrados = comentarios | orderBy:'-
    introduzido'| startFrom:(currentPageComentarios-
    1)*itemsPerPageComentarios | limitTo:itemsPerPageComentarios">
90.                                 <div class="pull-left " href="">
91.                                     <img class="media-object img-circle" ng-
    src="images/uploads/utilizadores/{{comentario.username.imagemPerfil.nome}}" alt="pr
    ofile ">
92.                                 </div>
```

```
93.                                 <div class="media-body">
94.                                     <div class="well well-lg">
95.                                         <h4 class="media-heading text-
    uppercase reviews">{{comentario.username.username}}</h4>
96.                                         <ul class="media-date text-
    uppercase reviews list-inline">
97.                                             <li class="">{{ comentario.introduzido
    | date:'medium'}}</li>
98.                                         </ul>
99.                                         <p class="media-
    comment">{{comentario.conteudo}}</p>
100.                                    </div>
101.                                </div>
102.                            </li>
103.                        </ul>
104.                    </div>
105.                    <div class="row pagination-
    Container paginationDetalhes">
106.                        <div class="col-sm-12  col-xs-12 col-lg-
    12 pagination ">
107.                            <ul uib-pagination total-
    items="comentarios.length" ng-model="currentPageComentarios"
108.                                items-per-
    page="itemsPerPageComentarios" class="pagination-sm"
109.                                data-boundary-links="true" force-
    ellipses="true"></ul>
110.                        </div>
111.                    </div>
112.                </div>
113.            </div>
114.        </div>
115.    </div>
116. </div>
```

## **Apêndice A.22** – Controlador do estado 'root.home.produtoDetalhes'

```
1.  angular.module('projetoGlobal')
2.
3.      .controller('ProdutoDetalhesCtrl', ['$scope', '$http', '$stateParams', '$rootSc
    ope', '$state', '$localStorage', 'Favoritos', 'Carrinho', function ($scope, $http,
    $stateParams, $rootScope, $state, $localStorage, Favoritos, Carrinho) {
4.          $scope.quantidadeProduto = 1;
5.          var min = 1;
6.          var max = 99;
7.          $scope.currentPageComentarios = 1;
8.          $scope.itemsPerPageComentarios = 5;
9.          $scope.comentarios = [];
10.
11.         $http({
12.             method: 'GET',
13.             url: 'produtos/detalhes/' + $stateParams.produtoId
14.         }).then(function (response) {
15.             $scope.produto = response.data.produto;
16.             $scope.comentarios = response.data.produto.comentarios;
17.             $scope.stock = response.data.stock;
18.         });
19.
20.         // Funcao para aumentar a quantidade
21.         $scope.incrementarQty = function () {
22.             if ($scope.quantidadeProduto >= max) {return;}
23.             $scope.quantidadeProduto++;
24.         }
25.         // Funcao para reduzir a quantidade
```

```
26.         $scope.decrementarQty = function () {
27.             if ($scope.quantidadeProduto <= min) {return;}
28.             $scope.quantidadeProduto--;
29.         };
30.
31.         $scope.adicionarComentario = function () {
32.             if ($rootScope.currentUser) {
33.                 var comentario = JSON.stringify({
34.                     username: $rootScope.currentUser._id,
35.                     comentario: $scope.TextAreaComentario
36.                 });
37.                 if ($scope.TextAreaComentario === undefined || $scope.TextAreaComen
    tario === '') alert('Introduzir Conteudo no Comentario');
38.                 else {
39.                     $http({
40.                         method: 'POST',
41.                         url: 'produtos/comentarios/adcionar/' + $stateParams.produt
    oId + '/' + $rootScope.currentUser._id,
42.                         data: comentario
43.                     }).then(function (response) {
44.                         $scope.comentarios = response.data;
45.                         $scope.TextAreaComentario = '';
46.                     });
47.                 }
48.             }
49.             else {
50.                 $scope.TextAreaComentario = '';
51.                 alert("Faca Login.")
52.             }
53.         }
54.
55.         $scope.addFavoritos = function (productID) {
56.             if ($rootScope.currentUser) Favoritos.add($rootScope.currentUser._id, p
    roductID)
57.             else $state.go('root.login', {reload: true});
58.         };
59.
60.         $scope.addCarrinho = function (productID) {
61.             if ($rootScope.currentUser) Carrinho.add($rootScope.currentUser._id, pr
    oductID, $localStorage.currentUser.carrinho, $scope.quantidadeProduto);
62.             else $state.go('root.login', {reload: true});
63.         };
64.
65.     }
66.     ]);
```

## **Apêndice A.23** – HTML da view do estado 'root.profile'

```
1.  <div class="row">
2.      <div class="col-lg-12 col-sm-12">
3.          <div class="card hovercard" ng-cloak>
4.              <div class="card-background">
5.                  <img class="card-bkimg" alt="" src="images/wallpaperProfile.jpg">
6.              </div>
7.              <div class="useravatar">
8.                  <img alt="Imagem Perfil" src='images/uploads/utilizadores/{{current
    User.imagemPerfil.nome}}'>
9.              </div>
10.             <div class="card-info"> <span class="card-
    title">{{currentUser.nomeCompleto}}</span>
11.             </div>
12.         </div>
```

```
13.         <div class="btn-pref btn-group btn-group-justified btn-group-
    lg" role="group">
14.             <div class="btn-group" role="group">
15.                 <button type="button" class="btn btn-primary" href="#tab1" data-
    toggle="tab">Informações Pessoais</button>
16.             </div>
17.             <div class="btn-group" role="group">
18.                 <button type="button" class="btn btn-default" href="#tab2" data-
    toggle="tab">Favoritos</button>
19.             </div>
20.             <div class="btn-group" role="group">
21.                 <button type="button" class="btn btn-default" href="#tab3" data-
    toggle="tab">Encomendas</button>
22.             </div>
23.         </div>
24.         <div class="well">
25.             <div class="tab-content">
26.                 <!-- Tab Personal Information -->
27.                 <div class="tab-pane fade in active" id="tab1">
28.                     <div class="row">
29.                         <div class="col-md-9 col-lg-9 personal-Information">
30.                             <table class="table table-user-information">
31.                                 <tbody>
32.                                     <tr>
33.                                         <th>Nome Completo:</th>
34.                                         <td>{{currentUser.nomeCompleto}}</td>
35.                                     </tr>
36.                                     <tr>
37.                                         <th>Username:</th>
38.                                         <td>{{currentUser.username}}</td>
39.                                     </tr>
40.                                     <tr>
41.                                         <th>Email:</th>
42.                                         <td>{{currentUser.email}}</td>
43.                                     </tr>
44.                                     <tr>
45.                                         <th rowspan="2">Endereço:</th>
46.                                         <td>Morada: {{currentUser.morada.morada}} {
    {currentUser.morada.codigoPostal}} {{currentUser.morada.localidade}}</td>
47.
48.                                     </tr>
49.                                     <tr>
50.                                         <td>Telefone: {{currentUser.morada.telefone
    }} </td>
51.                                     </tr>
52.                                 </tbody>
53.                             </table>
54.                             <a class="btn btn-primary button-alterar" data-
    toggle="modal" data-target="#myModalHorizontal">Alterar Informações Pessoais</a>
55.                             <a class="btn btn-primary" data-toggle="modal" data-
    target="#myModalMoradas">Alterar Endereço</a>
56.                         </div>
57.                     </div>
58.                 </div>
59.                 <!--END Tab Personal Information -->
60.                 <!-- Tab Favorites -->
61.                 <div class="tab-pane fade in" id="tab2">
62.                     <div class="row product-Slider-Controllers">
63.                         <div id="product-Favorites-
    Slider" class="carousel carouselProfile slide" data-ride="carousel">
64.                             <div class="carousel-inner carouselInner">
65.                                 <div ng-show="favoritos == 0">
66.                                     <div class="row">
67.                                         <ul class="product-Grid">
68.                                             <li class="semProdutosH3">
```

```
69.                                                 <h3>Não tem nenhum produto nos favo
   ritos</h3>
70.                                             </li>
71.                                         </ul>
72.                                     </div>
73.                                 </div>
74.                                 <div class="item" ng-show='favoritos' ng-
   class="{active:$first}" ng-repeat="items in favoritos">
75.                                     <div class="row">
76.                                         <div class="col-sm-10 col-md-10 col-lg-
   3 col-xs-12" ng-repeat="item in items">
77.                                             <div class="col-item">
78.                                                 <div class="photo">
79.                                                     <a ng-
   click='produtoDetalhes(item._id)'>
80.                                                         <img ng-
   src="images/uploads/produtos/{{item.imagem.file.filename}}" class="img-
   responsive" alt="Imagem do Produto" />
81.                                                     </a>
82.                                                 </div>
83.                                                 <div class="info">
84.                                                     <div class="row">
85.                                                         <div class="price col-sm-
   12 col-md-12 col-xs-12 col-lg-12">
86.                                                             <h5>
87.                                                                 {{item.nome}}</h5>

88.                                                             <h5 class="price-text-
   color">
89.                                                                 {{item.preco}} €</h
   5>
90.                                                         </div>
91.                                                     </div>
92.                                                 </div>
93.                                                 <div class="separator clear-
   left">
94.                                                     <p class="btn-
   add addtocart_button2 ">
95.                                                         <button value="Add to Cart
   " title="Add to Cart " class="addtocart-button cart-click" ng-
   click="addCarrinho(item._id);$event.preventDefault()">Add to Cart<span> </span></bu
   tton>
96.                                                     </p>
97.                                                     <p class="btn-
   add addtocart_button2">
98.                                                         <button class="btn btn-
   lg btn-brand btn-full-width addtocart-favoritos-Detalhes btn-fav " ng-
   click="removerFavorito(item._id);$event.preventDefault()">Remover</button>
99.                                                     </p>
100.                                                </div>
101.                                                <div class="clearfix ">
102.                                                </div>
103.                                            </div>
104.                                        </div>
105.                                    </div>
106.                                </div>
107.                            </div>
108.                        </div>
109.                        <div class="row">
110.                            <div class="col-md-9">
111.                                <div class="col-md-6 col-sm-6">
112.                                </div>
113.                            </div>
114.                            <div class="col-md-3">
115.                                <div class="controls pull-right">
```

```
116.                                    <a class="left fa fa-chevron-
    left btn btn-success" href="#product-Favorites-Slider" data-slide="prev" ng-
    click='$event.preventDefault()'></a>
117.                                    <a class="right fa fa-chevron-
    right btn btn-success" href="#product-Favorites-Slider" data-slide="next" ng-
    click='$event.preventDefault()'></a>
118.                                </div>
119.                            </div>
120.                        </div>
121.                    </div>
122.                </div>
123.                <!-- END Tab Favorites -->
124.                <!-- Tab Orders -->
125.                <div class="tab-pane fade in table-responsive" id="tab3">
126.                    <table class="table table-striped table-Orders">
127.                        <thead class="table-Orders-Header">
128.                            <tr>
129.                                <th>Encomenda Nº</th>
130.                                <th>Data</th>
131.                                <th>Total</th>
132.                                <th>Estado da encomenda</th>
133.                                <th> </th>
134.                            </tr>
135.                        </thead>
136.                        <tbody>
137.                            <tr ng-
    repeat="encomenda in encomendasFiltradas = (encomendas |  orderBy:'-
    data' | startFrom:(currentPage-1)*itemsPerPage | limitTo:itemsPerPage)">
138.                                <td>{{encomenda._id}}</td>
139.                                <td>{{encomenda.data | date:'medium'}}</td>

140.                                <td>{{encomenda.items.totalPreco}} €</td>
141.                                <td>{{encomenda.estadoEncomenda}}</td>
142.                                <td>
143.                                    <span><a ng-
    click='encomendasDetalhes(encomenda._id)'>Ver Detalhes da Encomenda</a></span>
144.                                </td>
145.                            </tr>
146.
147.                        </tbody>
148.
149.                    </table>
150.                    <div class="row pagination-
    Container paginationDetalhes">
151.                        <div class="col-sm-12  col-xs-12 col-lg-
    12 paginationProfile ">
152.                            <ul uib-pagination total-
    items="encomendas.length" ng-model="currentPage" items-per-
    page="itemsPerPage" class="pagination-sm" data-boundary-links="true" force-
    ellipses="true" next-text="Próximo" previous-text="Anterior" first-
    text="Início" last-text="Fim"></ul>
153.                        </div>
154.                    </div>
155.                </div>
156.                <!-- END Tab Orders -->
157.            </div>
158.        </div>
159.    </div>
160. </div>
161.
162.  <!-- Modal Personal Information -->
163.  <div class="modal fade" id="myModalHorizontal" tabindex="-
    1" role="dialog" aria-labelledby="myModalLabel" aria-hidden="true">
164.      <div class="modal-dialog">
165.          <div class="modal-content">
166.              <!-- Modal Header -->
```

```
167.                    <div class="modal-header">
168.                        <button type="button" class="close" data-dismiss="modal">
169.                                <span aria-hidden="true">×</span>
170.                                <span class="sr-only" ng-
    click="fecharModalInfo()">Close</span>
171.                            </button>
172.                        <h4 class="modal-
    title" id="myModalLabel">Editar Perfil</h4>
173.                    </div>
174.                    <!-- END Modal Header -->
175.                    <!-- Modal Body -->
176.                    <div class="modal-body">
177.                        <div class="row">
178.                            <form class="form-
    horizontal" role="form" enctype="multipart/form-data">
179.                                <div class="col-md-3">
180.                                    <div class="text-center">
181.                                        <img src="//placehold.it/100" class="avatar
    img-circle" alt="avatar">
182.                                        <h6>Upload de uma fota diferente</h6>
183.                                        <input type="file" name="file" id="file" ng-
    model="upload.file" ngf-select ngf-max-size="10MB" class="form-control" />
184.                                    </div>
185.                                </div>
186.                                <div class="col-md-9 personal-info">
187.                                    <h3>Informacoes Pessoais</h3>
188.                                    <br>
189.                                    <div class="form-group">
190.                                        <label class="col-lg-3 control-
    label">Full Name:</label>
191.                                        <div class="col-lg-8">
192.                                            <input class="form-
    control" type="text" ng-model='userUpdate.nomeCompleto' minlength="2">
193.                                        </div>
194.                                    </div>
195.                                    <div class="form-group">
196.                                        <label class="col-lg-3 control-
    label">Email:</label>
197.                                        <div class="col-lg-8">
198.                                            <input class="form-
    control" type="email" ng-model='userUpdate.email'>
199.                                        </div>
200.                                    </div>
201.                            </form>
202.                            </div>
203.                        </div>
204.                    </div>
205.                    <!-- END Modal Body -->
206.                    <div class="modal-footer">
207.                        <button type="button" class="btn btn-default" ng-
    click="fecharModalInfo()" data-dismiss="modal">Fechar</button>
208.                        <button type="button" class="btn btn-primary btn-
    orange" data-dismiss='modal' ng-
    click="updateProfile()">Guardar Alteracoes</button>
209.                    </div>
210.                </div>
211.            </div>
212.        </div>
213.        <!-- END Modal Personal Information -->
214.        <div class="modal fade" id="myModalMoradas" tabindex="-
    1" role="dialog" aria-labelledby="myModalLabel" aria-hidden="true">
215.            <div class="modal-dialog">
216.                <div class="modal-content">
217.                    <!-- Modal Header -->
218.                    <div class="modal-header">
219.                        <button type="button" class="close" data-dismiss="modal">
```

```
220.                                    <span aria-hidden="true">×</span>
221.                                    <span class="sr-only" ng-
    click="fecharModalMoradas()">Close</span>
222.                                </button>
223.                            <h4 class="modal-
    title" id="myModalMoradasLabel">Editar Morada</h4>
224.                        </div>
225.                        <!-- END Modal Header -->
226.                        <!-- Modal Body -->
227.                        <div class="modal-body">
228.                            <div class="row">
229.                                <div class="col-md-9 col-md-offset-1 personal-info text-
    center">
230.                                    <h3>Morada</h3>
231.                                    <br>
232.                                    <form class="form-horizontal" role="form">
233.                                        <div class="form-group">
234.                                            <label class="col-lg-3 control-
    label">Morada:</label>
235.                                            <div class="col-lg-8">
236.                                                <input class="form-
    control" type="text" ng-model='userUpdate.morada.morada'>
237.                                            </div>
238.                                        </div>
239.                                        <div class="form-group">
240.                                            <label class="col-lg-3 control-
    label">Localidade:</label>
241.                                            <div class="col-lg-8">
242.                                                <input class="form-
    control" type="text" ng-model='userUpdate.morada.localidade'>
243.                                            </div>
244.                                        </div>
245.                                        <div class="form-group">
246.                                            <label class="col-lg-3 control-
    label">Codigo Postal:</label>
247.                                            <div class="col-lg-8">
248.                                                <input class="form-
    control" type="number" ng-model='userUpdate.morada.codigoPostal'>
249.                                            </div>
250.                                        </div>
251.                                        <div class="form-group">
252.                                            <label class="col-lg-3 control-
    label">Telefone:</label>
253.                                            <div class="col-lg-8">
254.                                                <input class="form-
    control" type="number" ng-model='userUpdate.morada.telefone'>
255.                                            </div>
256.                                        </div>
257.                                    </form>
258.                                </div>
259.                            </div>
260.                        </div>
261.                        <!-- END Modal Body -->
262.                        <div class="modal-footer">
263.                            <button type="button" class="btn btn-default" data-
    dismiss="modal">Fechar</button>
264.                            <button type="button" class="btn btn-primary btn-
    orange" data-dismiss='modal' ng-
    click="updateMoradas()">Guardar Alteracoes</button>
265.                        </div>
266.                    </div>
267.                </div>
268.            </div>
```

# **Apêndice A.24 –** Controlador do estado 'root.profile'

```
1.  angular.module('projetoGlobal')
2.
3.      .controller('ProfileCtrl', ['$scope', '$state', '$http', '$localStorage', '$roo
    tScope', 'Upload', 'Favoritos', 'Encomendas', 'Carrinho', 'Morada', function ($scop
    e, $state, $http, $localStorage, $rootScope, Upload, Favoritos, Encomendas, Carrinh
    o, Morada) {
4.          // copia o conteudo da variavel
5.          $scope.userUpdate = angular.copy($rootScope.currentUser);
6.          $scope.quantidadeProduto = 1;
7.          $scope.currentPage = 1;
8.          $scope.itemsPerPage = 10;
9.          $scope.encomendas = [];
10.         getFavoritos();
11.         getEncomendas();
12.
13.         $(".btn-pref .btn").click(function () {
14.             $(".btn-pref .btn").removeClass("btn-primary").addClass("btn-
    default");
15.             $(this).removeClass("btn-default").addClass("btn-primary");
16.         });
17.
18.         $('#product-Favorites-Slider').carousel({interval: 0});
19.
20.         function getFavoritos() {
21.             Favoritos.get($rootScope.currentUser._id).then(
22.                 function successCallback(response) {
23.                     $scope.favoritos = [];
24.                     for (var t = 0; t < response.data.favoritos.length; t += 4) {
25.                         $scope.favoritos.push(response.data.favoritos.slice(t, t +
    4));
26.                     }
27.                 },
28.                 function errorCallback(response) {
29.                     console.log("registo Failed");
30.                 });
31.         }
32.
33.         function getEncomendas() {
34.             Encomendas.get($rootScope.currentUser._id).then(
35.                 function successCallback(response) {
36.                     $scope.encomendas = [];
37.                     $scope.encomendas = response.data;
38.                 },
39.                 function errorCallback(response) {
40.                     console.log("encomenda Failed");
41.                 });
42.         };
43.
44.         function removeUser() {
45.             $('#modal').modal('hide');
46.             delete $localStorage.currentUser;
47.             $http.defaults.headers.common.Authorization = '';
48.             $rootScope.currentUser = null;
49.             window.location.reload();
50.             $state.go('root.login', {reload: true});
51.         }
52.
53.         $scope.updateProfile = function () {
54.             var user = $scope.userUpdate;
55.             if (user.email == $localStorage.currentUser.email) {
56.                 $http.post('utilizadores/profile/', user)
57.                     .then(function successCallback(response) {
```

```
58.                             $rootScope.currentUser.nomeCompleto = response.data.nom
    eCompleto;
59.                             $localStorage.currentUser.nomeCompleto = response.data.
    nomeCompleto;
60.                             if ($scope.upload) { // se houver imagePerfil faz uploa
    d
61.                                 $scope.upload.file.name = $rootScope.currentUser._i
    d + '.png';
62.                                 Upload.upload({ // uploud da image para a base de d
    ados
63.                                     url: 'utilizadores/profile/imagePerfil/' + $roo
    tScope.currentUser._id,
64.                                     method: 'POST',
65.                                     data: $scope.upload
66.                                 }).then(function (response) {
67.                                     $rootScope.currentUser.imagemPerfil.nome = resp
    onse.data.imagemPerfil.nome;
68.                                     $localStorage.currentUser.imagemPerfil.nome = r
    esponse.data.imagemPerfil.nome;
69.                                     window.location.reload();
70.                                 });
71.                             }
72.                         },
73.                         function errorCallback(response) {
74.                             console.log("registo Failed");
75.                         });
76.             } else {
77.                 $http.post('utilizadores/profile/' + $localStorage.currentUser._id,
     user)
78.                     .then(function successCallback(response) {
79.                         $http.post('utilizadores/logout')
80.                             .then(function successCallback(response) {
81.                                 if ($scope.upload) { // se houver imagePerfil faz u
    pload
82.                                     $scope.upload.file.name = $rootScope.currentUse
    r._id + '.png';
83.                                     Upload.upload({ // uploud da image para a base
    de dados
84.                                         url: 'utilizadores/profile/imagePerfil/' +
    $rootScope.currentUser._id,
85.                                         method: 'POST',
86.                                         data: $scope.upload
87.                                     }).then(function (response) {
88.                                         removeUser();
89.                                     });
90.                                 } else {
91.                                     removeUser();
92.                                 }
93.                             }, function errorCallback(response) {
94.                                 console.log("Failed logout");
95.                             });
96.                     }, function errorCallback(response) {
97.                         console.log("registo Failed");
98.                     });
99.             }
100.            };
101.
102.            $scope.updateMoradas = function () {
103.                Morada.update($rootScope.currentUser._id, $scope.userUpdate.mora
    da);
104.            };
105.
106.            $scope.fecharModalInfo = function () {
107.                $scope.userUpdate = $rootScope.currentUser;
108.            };
109.            //Ir para a pagina de detalhes ao clickar no produto
```

```
110.                $scope.produtoDetalhes = function (id) {
111.                    $state.go('root.home.produtoDetalhes', {produtoId: id}, {reload:
    true});
112.                };
113.                //Ir para a pagina de detalhes ao clickar no produto
114.                $scope.encomendasDetalhes = function (id) {
115.                    $state.go('root.encomendasDetalhes', {encomendaId: id}, {reload:
    true});
116.                };
117.
118.                $scope.removerFavorito = function (userID) {
119.                    $http.post('utilizadores/removerFavoritos/' + $rootScope.current
    User._id + '/' + userID)
120.                        .then(function successCallback(response) {
121.                            getFavoritos();
122.                        }, function errorCallback(response) {
123.                            console.log(response);
124.                        });
125.                };
126.
127.                $scope.addCarrinho = function (productID) {
128.                    Carrinho.add($rootScope.currentUser._id, productID, $localStorag
    e.currentUser.carrinho, 1);
129.                };
130.            }]);
```

## Apêndice A.25 – HTML da view do estado 'root.login'

```
1.  <div class="row">
2.      <div class="col-lg-6 col-sm-6 col-lg-offset-3 col-sm-offset-3 panel-login">
3.          <div class="btn-pref btn-group btn-group-justified btn-group-
    lg" role="group" aria-label="...">
4.              <div class="btn-group btn-login-group" role="group">
5.                  <a id="stars" class="btn btn-primary" data-target="#tab1" data-
    toggle="tab">Login</a>
6.              </div>
7.              <div class="btn-group btn-login-group" role="group">
8.                  <a id="favorites" class="btn btn-default" data-target="#tab2" data-
    toggle="tab">Registar</a>
9.              </div>
10.         </div>
11.         <div class="well">
12.             <div class="tab-content login">
13.                 <!-- Tab Personal Information -->
14.                 <div class="tab-pane fade in active" id="tab1">
15.                     <div class="row">
16.                         <div class="col-md-12 col-lg-12 personal-
    Information2 loginBox">
17.                             <div class="box">
18.                                 <div class="content">
19.                                     <div ng-show="messageLogin" class="alert alert-
    danger">{{messageLogin}}</div>
20.                                     <div class="form loginBox">
21.                                         <form method="post" ng-
    submit="login(user)">
22.                                             <input id="email" class="form-
    control" type="text" placeholder="Email" ng-model="user.email">
23.                                             <input id="password" class="form-
    control" type="password" placeholder="Password" ng-model="user.password">
24.                                             <input class="btn btn-primary btn-
    login" type="submit" value="Login" ng-click="login(user)">
25.                                         </form>
26.                                     </div>
```

```
                                </div>
                            </div>
                        </div>
                    </div>
                    <!--END Tab Personal Information -->
                    <!-- Tab Favorites -->
                    <div class="tab-pane fade in box" id="tab2">
                        <div class="row form">
                            <div class="col-md-12 col-lg-12">
                                <div ng-show="messageRegisto" class="alert alert-
danger">{{messageRegisto}}</div>
                                <form name='registo'>
                                    <input id="nomeCompleto" class="form-
control" type="text" placeholder="Nome Completo" ng-
model="userRegisto.nomeCompleto" minlength="2" required>
                                    <input id="usermane" class="form-
control" type="text" placeholder="Username" ng-
model="userRegisto.username" minlength="5" required>
                                    <input id="emailRegisto" class="form-
control" type="email" placeholder="Email" ng-model="userRegisto.email" required>
                                    <input id="passwordRegisto" class="form-
control" type="password" placeholder="Password" ng-
model="userRegisto.password" pattern=".{5,10}" required title="5 to 10 characters">

                                    <input id="password_confirmation" class="form-
control" type="password" placeholder="Repeat Password" ng-
model="userRegisto.password2" required>
                                    <input class="btn btn-primary btn-
register" type="submit" value="Registar" ng-click="registar(userRegisto)">
                                </form>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
            <!-- END Tab Favorites -->
        </div>
</div>
```

## **Apêndice A.26** – Controlador do estado 'root.login'

```
angular.module('projetoGlobal')

    .controller('LoginCtrl', ['$scope', '$state',  '$http', '$rootScope', '$localSt
orage', 'jwtHelper', function ($scope, $state,  $http, $rootScope, $localStorage, j
wtHelper) {
        // This object will be filled by the form
        $scope.user = {};
        $scope.userRegisto = {};
        $scope.messageLogin;

        // Alterar as Tabs do login e registo
        $(".btn-pref .btn").click(function () {
            $(".btn-pref .btn").removeClass("btn-primary").addClass("btn-
default");
            $(this).removeClass("btn-default").addClass("btn-primary");
        });

        $scope.login = function (user) {
            $http.post('/utilizadores/login', user)
                .then(function successCallback(response) {
                    var token = jwtHelper.decodeToken(response.data);
```

```
                        $localStorage.currentUser = token;
                        $localStorage.currentUser.token = response.data;
                        $http.defaults.headers.common.Authorization = 'Bearer ' + r
esponse.data;
                        $rootScope.currentUser = token;
                        $state.go('root.home', {reload: true});
                    },
                    function errorCallback(response) {
                        $scope.messageLogin = 'Password ou email Incorreto.';
                        $scope.user.email = '';
                        $scope.user.password = '';
                    });
        };
        $scope.registar = function (user) {
            if ($scope.registo.$valid && user.password == user.password2) {
                $http.post('/utilizadores/registar', user)
                    .then(function successCallback(response) {
                        if (response.data.errorMessage) {
                            $scope.messageRegisto = response.data.errorMessage;
                            $scope.userRegisto.password = '';
                            $scope.userRegisto.password2 = '';
                        } else {
                            $scope.userRegisto = '';
                            $state.reload();
                        }
                    }, function errorCallback(response) {
                        $scope.messageRegisto = 'Registo mal sucedido.';
                        $scope.userRegisto.passwordRegisto = '';
                        $scope.userRegisto.password2 = '';
                    });
            } else {
                $scope.messageRegisto = "As passwords nao sao iguais";
                $scope.userRegisto.password = '';
                $scope.userRegisto.password2 = '';
            }
        }
    }]);
```

## **Apêndice A.27** – HTML da view do estado 'root.cart'

```
<div class="row cart">
    <h1 class="info-text">Carrinho</h1>
    <div class="col-md-12 col-lg-12 personal-Information">
        <div class="tab-pane fade in table-responsive" id="tab3">
            <table class="table table-striped table-bordered table-Orders-
Detalhes">
                <colgroup>
                    <col width='45%'>
                    <col width='10%'>
                    <col width='10%'>
                    <col width='10%'>
                    <col width='5%'>
                </colgroup>
                <thead class="table-Orders-Header">
                    <tr>
                        <th>Produto</th>
                        <th>Preço</th>
                        <th>Quantidade</th>
                        <th>Sub-Total</th>
                        <th></th>
                    </tr>
                </thead>
                <tbody>
```

```
23.                     <tr ng-repeat='produto in currentUser.carrinho.items'>
24.                         <td>
25.                             <div class="table-Orders-Detalhes">
26.                                 <div class="col-lg-3 col-md-3 col-sm-3 table-
    Orders-Detalhes-Imagem imagem-cart">
27.                                     <img ng-
    src="images/uploads/produtos/{{produto.item.imagem.file.filename}}" ui-
    sref="root.home.produtoDetalhes({ produtoId:produto.item._id })" ui-sref-
    opts="{reload: true}">
28.                                 </div>
29.                                 <div class="col-lg-8 col-md-4 col-sm-4 table-
    Orders-Detalhes-Produto">
30.                                     <span class="table-Orders-Detalhes-Produto-
    Nome">{{ produto.item.nome }}</span>
31.                                 </div>
32.                             </div>
33.                         </td>
34.                         <td>
35.                             <div class="table-Orders-Detalhes-Produto2">
36.                                 <span>{{produto.item.preco}} €</span>
37.                             </div>
38.                         </td>
39.                         <td>
40.                             <div class="table-Orders-Detalhes-Produto2">
41.                                 <div class="input-append">
42.                                     <div class="row">
43.                                         <span>x{{produto.qty}}</span>
44.                                     </div>
45.                                     <div class="row">
46.                                         <button class="btn btn-
    inverse btn_Decrementar_Qty" type="button" ng-disabled="item.quantity <= 1" ng-
    click="addCarrinho(produto.item._id)">+</button>
47.                                         <button class="btn btn-
    default btn_Incrementar_Qty" type="button" ng-disabled="item.quantity >= 99" ng-
    click="removerUmCarrinho(produto.item._id)">-</button>
48.                                     </div>
49.                                 </div>
50.                             </div>
51.                         </td>
52.                         <td>
53.                             <div class="table-Orders-Detalhes-Produto2">
54.                                 <span>{{produto.preco}} €</span>
55.                             </div>
56.                         </td>
57.                         <td>
58.                             <div class="cart_Trash">
59.                                 <a ng-click="removerCarrinho(produto.item._id)">
60.                                     <span class="glyphicon glyphicon-trash" aria-
    hidden="true"></span>
61.                                 </a>
62.                             </div>
63.                         </td>
64.                     </tr>
65.                 </tbody>
66.                 <tfoot ng-hide="!currentUser">
67.                     <tr>
68.                         <th colspan="2"></th>
69.                         <th class="table-Orders-Detalhes-Produto-
    Total" colspan="1">Total:</th>
70.                         <td class="table-Orders-Detalhes-Produto-
    Total" colspan="2">{{currentUser.carrinho.totalPreco}} €</td>
71.                     </tr>
72.                 </tfoot>
73.             </table>
```

```
74.             <a class="btn btn-default btn-orders-Detalhes" ng-
    click="goToHome()"><span class="glyphicon glyphicon-arrow-left" aria-
    hidden="true"></span> Continuar a Comprar</a>
75.             <input type='button' class='btn btn-default btn-next pull-right btn-
    checkout btn-orders-Detalhes' name='checkout' value='Comprar' ng-
    click='goToCheckout()' />
76.         </div>
77.     </div>
78. </div>
```

## **Apêndice A.28** – Controlador do estado 'root.cart'

```
1.  angular.module('projetoGlobal')
2.
3.      .controller('CartCtrl', ['$scope',  '$rootScope', '$state', '$localStorage', 'C
    arrinho', function ($scope,  $rootScope, $state, $localStorage, Carrinho) {
4.
5.          $scope.goToHome = function () {
6.              $state.go('root.home', {reload: true});
7.          };
8.
9.          $scope.goToCheckout = function () {
10.             if ($localStorage.currentUser.carrinho.totalPreco == 0 || $localStorage
    .currentUser.carrinho.totalPreco == undefined) {
11.                 $state.go('root.home', {reload: true});
12.             } else {
13.                 $state.go('root.checkout', {reload: true});
14.             }
15.         };
16.
17.         $scope.addCarrinho = function (productID) {
18.             Carrinho.add($rootScope.currentUser._id, productID, $localStorage.curre
    ntUser.carrinho, 1);
19.         };
20.
21.         $scope.removerUmCarrinho = function (productID) {
22.             Carrinho.remover('produtos/removerUm-
    Carrinho/', $rootScope.currentUser._id, productID, $localStorage.currentUser.carrin
    ho);
23.         };
24.
25.         $scope.removerCarrinho = function (productID) {
26.             Carrinho.remover('produtos/remover-
    Carrinho/', $rootScope.currentUser._id, productID, $localStorage.currentUser.carrin
    ho);
27.         };
28.     }]);
```

## **Apêndice A.29** – HTML da view do estado 'root.checkout'

```
1.  <div class="row cart checkout">
2.      <div class="checkout">
3.          <h1>Envio e Pagamento</h1>
4.          <div class="col-lg-4 col-md-4 col-sm-4 orderReviewTable_Div pull-right">
5.              <div class="orderReviewTable_Div2">
6.                  <div id="order-summary" class="grey-box order-summary">
7.                      <h3>Resumo da Encomenda</h3>
8.                      <ul class="summary-list list-unstyled">
9.                          <li>Sub-
    total <span>{{currentUser.carrinho.totalPreco}} €</span></li>
10.                         <li>Taxa de Envio<span>{{taxaEnvio}} €</span></li>
```

```
11.                         <li>Preço Total <span>{{PrecoTotal}} €</span></li>
12.                     </ul>
13.
14.                 </div>
15.             </div>
16.         </div>
17.         <div class="col-lg-8 col-md-8 col-sm-8 pull-left shippingAddressCol">
18.             <div class="row">
19.                 <h4 class="info-text">Informação de Envio</h4>
20.                 <div class="col-lg-12 col-md-12 col-sm-12 pull-
    left shippingAddressDiv">
21.                     <ul id='listShippingAddress' class="shippingList">
22.                         <form>
23.                             <li ng-hide="soNovaMorada">
24.                                 <input class="magic-
    radio" name='addressUser' id="addressUser" type="radio" ng-
    model='address.name' value="moradaPredefinida"/>
25.                                 <label for="addressUser"> <b>Morada:</b> {{currentU
    ser.morada.morada}} {{currentUser.morada.codigoPostal}}, {{currentUser.morada.local
    idade}} <b>Telefone:</b> {{currentUser.morada.telefone}}</label>
26.                             </li>
27.                             <li>
28.                                 <input type="radio" id='novaMorada' name='novaMorad
    a' class="magic-radio" ng-model='address.name' value="novaMorada"/>
29.                                 <label for="novaMorada"><b>Nova Morada:</b></label>

30.                             </li>
31.                         </form>
32.                     </ul>
33.                 </div>
34.             </div>
35.             <div class="row" ng-show="address.name =='novaMorada'">
36.                 <div class="col-lg-12 col-md-12 col-sm-12 pull-left">
37.                     <div class="billingAddressDiv">
38.                         <form name='novaMoradaForm' action="">
39.                             <!-- Text input-->
40.                             <div class="form-group textInputCheckout">
41.                                 <label class="col-md-2 col-sm-2 control-
    label addressLabel" for="morada">Morada</label>
42.                                 <div class="col-md-9 col-sm-9">
43.                                     <input id="" name="morada" type="text" class="f
    orm-control input-md" ng-model="novaMorada.morada" required minlength="2">
44.                                 </div>
45.                             </div>
46.                             <!-- Text input-->
47.                             <div class="form-group textInputCheckout">
48.                                 <label class="col-md-2 col-sm-2 control-
    label addressLabel"
49.                                         for="localidade">Localidade</label>
50.                                 <div class="col-md-9 col-sm-9">
51.                                     <input id="" name="localidade" type="text" clas
    s="form-control input-md" ng-
    model="novaMorada.localidade" required minlength="2">
52.                                 </div>
53.                             </div>
54.                             <!-- Text input-->
55.                             <div class="form-group textInputCheckout">
56.                                 <label class="col-md-2 col-sm-2 control-
    label addressLabel" for="codigoPostal">Codigo-Postal</label>
57.                                 <div class="col-md-9 col-sm-9">
58.                                     <input id="" name="codigoPostal" type="number"
    class="form-control input-md" ng-model="novaMorada.codigoPostal" required>
59.                                 </div>
60.                             </div>
61.                             <!-- Text input-->
62.                             <div class="form-group textInputCheckout">
```

```
63.                                 <label class="col-md-2 col-sm-2 control-
   label addressLabel" for="telefone">Telefone</label>
64.                                 <div class="col-md-9 col-sm-9">
65.                                     <input id="NovaMoradatelefone" name="telefone"
   type="number" class="form-control input-md" ng-
   model="novaMorada.telefone" required>
66.                                 </div>
67.                             </div>
68.                         </form>
69.                     </div>
70.                 </div>
71.             </div>
72.             <div class="row billingAddressDiv">
73.                 <h4 class="info-text">Métodos de Envio</h4>
74.                 <div class="col-lg-12 col-md-12 col-sm-12 pull-
   left metodosEnvioDiv">
75.                     <ul id='listMetodosEnvio' class="shippingList">
76.                         <form name='metodosEnvioForm' action="">
77.                             <li ng-repeat='metodos in metodosEnvio'>
78.                                 <input class="magic-
   radio" id="{{ metodos.nome}}" type="radio" name='metodosEnvio' ng-
   model='metodosEnvio.nome' ng-value="metodos"/>
79.                                 <label for="{{metodos.nome}}"> {{ metodos.nome}} </
   label>
80.                             </li>
81.                         </form>
82.                     </ul>
83.                 </div>
84.             </div>
85.
86.             <div class="row billingAddressDiv paymentDiv">
87.                 <h4 class="info-text ">Métodos de Pagamento</h4>
88.                 <div class="col-lg-12 col-md-12 col-sm-12 pull-
   left metodosPagamentoDiv">
89.                     <ul id='listMetodosPagamento' class="shippingList">
90.                         <li ng-repeat='metodos in metodosPagamento'>
91.                             <input class="magic-
   radio" id="{{ metodos.nome}}" type="radio" name='{{ metodos.nome}}' ng-
   model='metodosPagamentos.nome' ng-value="metodos.model"/>
92.                             <label for="{{metodos.nome}}"> {{ metodos.nome}} </labe
   l>
93.                         </li>
94.                     </ul>
95.                     <form id="checkout-
   form" name='comprarStripeForm' stripe:form="comprarStripe" ng-
   show="opcao == 'CartaoDeCredito'">
96.                         <div class="row">
97.                             <div class="col-lg-12 col-md-12 col-sm-12 pull-left">
98.                                 <div class="form-group textInputCheckout">
99.                                     <label class="col-md-2 col-sm-2 control-
   label addressLabel" for="name">Nome Completo</label>
100.                                    <div class="col-md-9 col-sm-9">
101.                                        <input type="text" id="name" class="
   form-control" name="name" ng-model="cartaoCredito.name" required>
102.                                    </div>
103.                                </div>
104.                                <div class="form-group textInputCheckout">
105.                                    <label class="col-md-2 col-sm-2 control-
   label addressLabel" for="address">Endereço</label>
106.                                    <div class="col-md-9 col-sm-9">
107.                                        <input type="text" id="address" clas
   s="form-control" name="address" ng-model="cartaoCredito.address" required>
108.                                    </div>
109.                                </div>
110.                                <div class="form-group textInputCheckout">
```

```
111.                                        <label class="col-md-2 col-sm-2 control-
    label addressLabel" for="card-name">Cartão em nome</label>
112.                                        <div class="col-md-9 col-sm-9">
113.                                            <input type="text" id="card-
    name" class="form-control" ng-model="cartaoCredito.cardName" required>
114.                                        </div>
115.                                    </div>
116.                                    <div class="form-group textInputCheckout">
117.                                        <label class="col-md-2 col-sm-2 control-
    label addressLabel" for="card-number">Número do Cartão de Crédito</label>
118.                                        <div class="col-md-9 col-sm-9">
119.                                            <input type="number" id="card-
    number" class="form-control" ng-model="cartaoCredito.cardNumber" data-
    stripe="number" minlength="16" maxlength="16" required>
120.                                        </div>
121.                                    </div>
122.                                    <div class="form-group textInputCheckout">
123.                                        <div class="col-sm-4 col-md-4">
124.                                            <label class="col-md-3 col-sm-
    3 control-label addressLabel cardExpiration" for="card-expiry-
    month">Mês de Expiração</label>
125.                                        <div class="col-md-1 col-sm-1">
126.                                            <input type="number" id="card-
    expiry-month" class="form-control cardExpirationInput" ng-
    model="cartaoCredito.cardExpiryMonth" data-stripe="exp-
    month" minlength="2" maxlength="2" required>
127.                                        </div>
128.                                    </div>
129.                                    <div class="col-md-4 col-sm-4 ">
130.                                        <label class="col-md-3 col-sm-
    3 control-label addressLabel cardExpiration" for="card-expiry-
    year">Ano de Expiração</label>
131.                                        <div class="col-md-1 col-sm-1">
132.                                            <input type="number" id="card-
    expiry-year" class="form-control cardExpirationInput" ng-
    model="cartaoCredito.cardExpiryYear" data-stripe="exp-
    year" minlength="4" maxlength="4" required>
133.                                        </div>
134.                                    </div>
135.                                    <div class="col-md-4 col-sm-4 ">
136.                                        <label class="col-md-3 col-sm-
    3 control-label addressLabel cardExpiration" for="card-cvc">CVC</label>
137.                                        <div class="col-md-1 col-sm-1">
138.                                            <input type="number" id="card-
    cvc" class="form-control cardExpirationInput" ng-
    model="cartaoCredito.cardCVC" data-
    stripe="cvc" minlength="3" maxlength="3" required>
139.                                        </div>
140.                                    </div>
141.                                </div>
142.                            </div>
143.                        </div>
144.                        <input type='submit' id='btn-finish' class='btn btn-
    default btn-orders-Detalhes' name='finish' value='Comprar' ng-
    disabled="disableButton1"/>
145.                    </form>
146.                </div>
147.            </div>
148.            <div class="row">
149.                <div class="pull-left">
150.                    <a class="btn btn-default btn-previous btn-orders-
    Detalhes" ng-click='goToCart()'>
151.                        <span class="glyphicon glyphicon-arrow-left" aria-
    hidden="true"></span> Voltar Atrás</a>
152.                </div>
153.                <div class="pull-right">
```

```
154.                         <button class="btn btn-default btn-previous btn-orders-
    Detalhes" ng-hide="opcao == 'CartaoDeCredito'" ng-click='comprarCobranca()' ng-
    disabled="disableButton2">Comprar</button>
155.                     </div>
156.                 </div>
157.             </div>
158.
159.         </div>
160.     </div>
```

## **Apêndice A.30** – Controlador do estado 'root.checkout'

```
1.  angular.module('projetoGlobal')
2.
3.      .controller('CheckoutCtrl', ['$scope', '$http', '$rootScope', '$state', '$local
    Storage', function ($scope, $http, $rootScope, $state, $localStorage) {
4.          $scope.metodosEnvio = [];
5.          $scope.novaMorada = {};
6.          $scope.cartaoCredito = {};
7.          $scope.morada = {};
8.          $scope.address = {name: 'moradaPredefinida'};
9.          $scope.metodosPagamentos = {nome: 'Combrança'};
10.         $scope.taxaEnvio;
11.         $scope.PrecoTotal = $rootScope.currentUser.carrinho.totalPreco + $scope.tax
    aEnvio;
12.         $scope.soNovaMorada = false;
13.         $scope.metodosEnvioSelecionado = false;
14.         $scope.metodoEnvio;
15.         $scope.novaMoradaBool = false;
16.         $scope.disableButton1 = false;
17.         $scope.disableButton2 = false;
18.         getMetodosEnvio();
19.
20.         function getMetodosEnvio() {
21.             $http.get('/metodosEnvio/')
22.                 .then(function successCallback(response) {
23.                         $scope.metodosEnvio = response.data;
24.                     },
25.                     function errorCallback(response) {
26.                         console.log("sometingh Failed");
27.                     });
28.         }
29.
30.         $scope.$watch('address.name', function (value) {
31.             if ($rootScope.currentUser.morada.morada == '' || $rootScope.currentUse
    r.morada.codigoPostal == '' || $rootScope.currentUser.morada.localidade == '' || $r
    ootScope.currentUser.morada.telefone == '' || $rootScope.currentUser.morada.morada
    == undefined) {
32.                 $scope.soNovaMorada = true;
33.                 $scope.address = {name: 'novaMorada'};
34.             }
35.             if (value == 'novaMorada') {
36.                 $scope.novaMoradaBool = true;
37.                 $scope.morada = $scope.novaMorada;
38.             } else {
39.                 $scope.novaMoradaBool = false;
40.                 $scope.novaMorada = {};
41.                 $scope.morada = $rootScope.currentUser.morada;
42.             }
43.         });
44.
45.         $scope.$watch('metodosPagamentos.nome', function (value) {
46.             $scope.opcao = value;
```

```
47.         });
48.
49.         $scope.$watch('metodosEnvio.nome', function (value) {
50.             if (value != undefined) {
51.                 $scope.metodosEnvioSelecionado = true;
52.                 $scope.metodoEnvio = value.nome;
53.                 $scope.taxaEnvio = 0;
54.                 $scope.taxaEnvio = parseInt(value.preco);
55.                 $scope.PrecoTotal = $rootScope.currentUser.carrinho.totalPreco + $s
    cope.taxaEnvio;
56.             }
57.             else {
58.                 $scope.taxaEnvio = 0;
59.                 $scope.PrecoTotal = $rootScope.currentUser.carrinho.totalPreco + $s
    cope.taxaEnvio;
60.                 $scope.metodosEnvioSelecionado = false;
61.             }
62.         });
63.         $scope.goToCart = function () {
64.             $state.go('root.cart', {reload: true});
65.         };
66.
67.         $scope.comprarStripe = function (status, response) {
68.             if (response.error) {
69.                 alert(response.error.message);
70.             }
71.             else {
72.                 if ($scope.novaMoradaBool && $scope.novaMoradaForm.$valid && $scope
    .metodosEnvioSelecionado && $scope.comprarStripeForm.$valid) {
73.                     comprar();
74.                 }
75.                 else if (!$scope.novaMoradaBool && $scope.morada && $scope.metodosE
    nvioSelecionado && $scope.comprarStripeForm.$valid) {
76.                     comprar();
77.                 }
78.                 else {
79.                     if ($scope.morada == '') alert("Preencha as Informações de Envi
    o.");
80.                     else if (!$scope.metodosEnvioSelecionado) alert("Selecione um M
    etodo de Envio.");
81.                     else if (!$scope.comprarStripeForm.$valid) alert("Preencha as I
    nformações do Cartao de Credito.");
82.                 }
83.             }
84.             function comprar() {
85.                 $scope.disableButton1 = true;
86.                 $rootScope.currentUser.carrinho.totalPreco += $scope.taxaEnvio;
87.                 $http.post('/produtos/checkout/cartaoCredito/' + $rootScope.current
    User._id, {
88.                     'stripeToken': response,
89.                     'morada': $scope.morada,
90.                     'carrinho': $localStorage.currentUser.carrinho,
91.                     'portes': $scope.metodoEnvio,
92.                     'portesValor': $scope.taxaEnvio
93.                 })
94.                     .then(function successCallback(response) {
95.                             $localStorage.currentUser.carrinho = response.data.carr
    inho;
96.                             $rootScope.currentUser.carrinho = response.data.carrinh
    o;
97.                             $state.go('root.checkoutResumo', {encomendaId: response
    .data.result._id}, {reload: true});
98.                         },
99.                         function errorCallback(response) {
100.                             });
101.                 }
```

```
102.                 };
103.
104.                 $scope.comprarCobranca = function () {
105.                     if ($scope.novaMoradaBool && $scope.novaMoradaForm.$valid && $sc
    ope.metodosEnvioSelecionado) {
106.                         comprar();
107.                     }
108.                     else if (!$scope.novaMoradaBool && $scope.morada && $scope.metod
    osEnvioSelecionado) {
109.                         comprar();
110.                     }
111.                     else {
112.                         if (!$scope.metodosEnvioSelecionado) alert("Selecione um Met
    odo de Envio.");
113.                         else alert("Preencha as Informações de Envio.");
114.                     }
115.                     function comprar() {
116.                         $scope.disableButton2 = true;
117.                         $rootScope.currentUser.carrinho.totalPreco += $scope.taxaEnv
    io;
118.                         $http.post('/produtos/checkout/cobranca/' + $rootScope.curre
    ntUser._id, {
119.                             'morada': $scope.morada,
120.                             'carrinho': $localStorage.currentUser.carrinho,
121.                             'portes': $scope.metodoEnvio,
122.                             'portesValor': $scope.taxaEnvio
123.                         })
124.                             .then(function successCallback(response) {
125.                                     $localStorage.currentUser.carrinho = response.da
    ta.carrinho;
126.                                     $rootScope.currentUser.carrinho = response.data.
    carrinho;
127.                                     $state.go('root.checkoutResumo', {encomendaId: r
    esponse.data.result._id}, {reload: true});
128.                                 },
129.                                 function errorCallback(response) {
130.                                     console.log("sometingh Failed");
131.                                 });
132.                     }
133.                 }
134.
135.
136.                 if (window.scrollY >= 250) { // change target to number
137.                     document.getElementById('order-
    summary').style.position = 'absolute';
138.                     document.getElementById('order-summary').style.top = '250px';
139.                 }
140.                 if (window.scrollY < 249) { // change target to number
141.                     document.getElementById('order-summary').style.top = '';
142.                     document.getElementById('order-
    summary').style.position = 'fixed';
143.                 }
144.
145.                 $scope.metodosPagamento = [{
146.                     nome: "À Combrança",
147.                     model: "Combrança"
148.                 },
149.                     {
150.                         nome: "Cartão de Crédito",
151.                         model: "CartaoDeCredito"
152.                     }
153.                 ];
154.
155.             }]);
```

## Apêndice A.31 – HTML da view do estado 'root. checkoutResumo'

```
<div class="row cart checkout">
    <div class="checkout checkoutResumo">
        <div class="tab-content">
            <div class="row">
                <div class="col-md-12 col-sm-12">
                    <div class="info-text-encomenda-feita-Div">
                        <h4 class="info-text" id="info-text-encomenda-
feita">A sua encomenda foi efetuada com sucesso.</h4>
                    </div>
                    <div class="info-p-encomenda-feita-Div">
                        <p>Obrigado pela sua compra.</p>
                        <p>O número da sua encomenda é: <span class="info-span-
encomenda">{{encomendaID}}</span></p>
                        <p>Vai receber dentro de momentos um email com a confirmaçã
o da sua encomenda.</p>
                    </div>
                </div>
            </div>
            <div class="row div-btn-finish-order">
                <div class="">
                    <a ui-sref="root.home" ui-sref-
opts="{reload: true}" class="btn btn-default btn-previous btn-orders-Detalhes btn-
finish-order">Voltar para a Página Principal</a>
                </div>
            </div>
        </div>
    </div>
</div>
```

## Apêndice A.32 – Controlador do estado 'root.checkoutResumo'

```
angular.module('projetoGlobal')

    .controller('CheckoutResumoCtrl', ['$scope', '$stateParams', function ($scope,
$stateParams) {
        $scope.encomendaID = $stateParams.encomendaId;
    }]);
```

## Apêndice A.33 – HTML da view do estado 'root.encomendasDetalhes'

```
<div class="row">
    <div class="col-lg-12 col-sm-12">
        <div class="card hovercard" ng-cloak>
            <div class="card-background">
                <img class="card-bkimg" alt="" src="images/wallpaperProfile.jpg">
            </div>
            <div class="userávatar">
                <img alt="" src='images/uploads/utilizadores/{{currentUser.imagemPe
rfil.nome}}'>
            </div>
            <div class="card-info"><span class="card-
title">{{currentUser.nomeCompleto}}</span>
            </div>
        </div>
        <div class="orders-Details-h3">
```

```
14.             <h3>Detalhes da Encomenda: {{encomendaID}}</h3>
15.         </div>
16.         <div class="well">
17.             <div class="tab-content">
18.                 <!-- Tab Personal Information -->
19.                 <div class="tab-pane fade in active" id="tab1">
20.                     <a class="btn btn-default btn-orders-Detalhes" ui-
    sref="root.profile" ui-sref-opts="{reload: true}"><span
21.                                 class="glyphicon glyphicon-arrow-left" aria-
    hidden="true"></span> Voltar Atrás</a>
22.                     <div class="row">
23.                         <div class="col-md-9 col-lg-9 personal-Information">
24.                             <table class="table table-user-information">
25.                                 <tbody>
26.                                 <tr>
27.                                     <th>Id do Pagamento:</th>
28.                                     <td>{{encomenda.pagamentoID}}</td>
29.                                 </tr>
30.                                 <tr>
31.                                     <th>Tipo de Pagamento:</th>
32.                                     <td>{{encomenda.tipoDePagamento}}</td>
33.                                 </tr>
34.                                 <tr>
35.                                     <th>Encomenda Realizada em:</th>
36.                                     <td>{{encomenda.data | date:'medium'}}</td>
37.                                 </tr>
38.                                 <tr>
39.                                     <th>Morada:</th>
40.                                     <td>{{encomenda.moradaShipping.morada}}</td>
41.                                 </tr>
42.                                 <tr>
43.                                     <th>Localidade:</th>
44.                                     <td>{{encomenda.moradaShipping.localidade}}</td
    >
45.                                 </tr>
46.                                 <tr>
47.                                     <th>Código Postal:</th>
48.                                     <td>{{encomenda.moradaShipping.codigoPostal}}</
    td>
49.                                 </tr>
50.                                 <tr>
51.                                     <th>Telefone:</th>
52.                                     <td>{{encomenda.moradaShipping.telefone}}</td>

53.                                 </tr>
54.                                 <tr>
55.                                     <th>Estado da Encomenda:</th>
56.                                     <td>{{encomenda.estadoEncomenda}}</td>
57.                                 </tr>
58.                                 <tr>
59.                                     <th>Portes Nome:</th>
60.                                     <td>{{encomenda.portes.nome}}</td>
61.                                 </tr>
62.                                 <tr>
63.                                     <th>Portes Preço:</th>
64.                                     <td>{{encomenda.portes.preco}} €</td>
65.                                 </tr>
66.                                 </tbody>
67.                             </table>
68.                         </div>
69.                     </div>
70.                 </div>
71.                 <div class="row">
72.                     <div class="col-md-12 col-lg-12 personal-Information">
73.                         <div class="tab-pane fade in table-responsive" id="tab3">
```

```
74.                         <table class="table table-striped table-bordered table-
    Orders-Detalhes">
75.                             <colgroup>
76.                                 <col width='50%'>
77.                                 <col width='10%'>
78.                                 <col width='10%'>
79.                                 <col width='10%'>
80.                             </colgroup>
81.                             <thead class="table-Orders-Header">
82.                             <tr>
83.                                 <th>Detalhes do Produto</th>
84.                                 <th>Preço</th>
85.                                 <th>Quantidade</th>
86.                                 <th>Sub-Total</th>
87.                             </tr>
88.                             </thead>
89.
90.                             <tbody>
91.                             <tr ng-repeat="produto in encomenda.items.items">
92.                                 <td>
93.                                     <div class="table-Orders-Detalhes">
94.                                         <div class="col-lg-3 col-md-3 col-sm-
    3 table-Orders-Detalhes-Imagem">
95.                                             <img ng-
    src="images/uploads/produtos/{{produto.item.imagem.file.filename}}">
96.                                         </div>
97.                                         <div class="col-lg-8 col-md-4 col-sm-
    4 table-Orders-Detalhes-Produto">
98.                                             <span class="table-Orders-Detalhes-
    Produto-Nome">{{ produto.item.nome }}</span>
99.                                         </div>
100.                                        </div>
101.                                    </td>
102.                                    <td>
103.                                        <div class="table-Orders-Detalhes-
    Produto2">
104.                                            <span>{{produto.item.preco}} €</
    span>
105.                                        </div>
106.                                    </td>
107.                                    <td>
108.                                        <div class="table-Orders-Detalhes-
    Produto2">
109.                                            <span>x{{produto.qty}}</span>
110.                                        </div>
111.                                    </td>
112.                                    <td>
113.                                        <div class="table-Orders-Detalhes-
    Produto2">
114.
115.                                            <span>{{produto.preco}} €</span>

116.                                        </div>
117.                                    </td>
118.                                </tr>
119.                                </tbody>
120.
121.                                <tfoot>
122.                                <tr>
123.                                    <th colspan="2"></th>
124.                                    <th class="table-Orders-Detalhes-
    Produto-Total" colspan="1">Total:</th>
125.                                    <td class="table-Orders-Detalhes-
    Produto-Total">{{encomenda.precoTotal}} €</td>
126.                                </tr>
127.                                </tfoot>
```

```
                                    </table>
                                </div>
                            </div>
                        </div>
                        </div>
                        <!--END Tab Personal Information -->
                        <!-- Tab Favorites -->
                    </div>
                </div>
            </div>
            </div>
```

## Apêndice A.34 – Controlador do estado 'root.encomendasDetalhes'

```
angular.module('projetoGlobal')

.controller('EncomendasDetalhesCtrl', ['$scope', '$state', '$http', '$stateParams',
 function ($scope, $state, $http, $stateParams) {
    $scope.encomendaID = $stateParams.encomendaId;
    $scope.encomenda;
    // Procura o produto que vem passado como parametro do estado anterior
    $http({
        method: 'GET',
        url: 'encomendas/encomendaDetalhe/' + $scope.encomendaID
    }).then(function(response) {
      $scope.encomenda = response.data;;
    });
}]);
```

## Apêndice A.35 – HTML da view do estado 'root.admin'

```
<div id="adminPanel" class="row cart">
    <div class="col-sm-2 col-md-2 col-lg-2 ">
        <nav class="nav-sidebar">
            <ul class="nav tabs">
                <li class="active"><a data-target="#tabProdutos" data-
toggle="tab">Produtos</a></li>
                <li class=""><a data-target="#tabUtilizadores" data-
toggle="tab">Utilizadores</a></li>
                <li class=""><a data-target="#tabCategorias" data-
toggle="tab">Subcategorias</a></li>
                <li class=""><a data-target="#tabEncomendas" data-
toggle="tab">Encomendas</a></li>
                <li class=""><a data-target="#tabBanner" data-
toggle="tab">Produtos do Banner</a></li>
                <li class=""><a data-target="#tabMetodosEnvio" data-
toggle="tab">Métodos de Envio</a></li>
            </ul>
        </nav>
    </div>
    <!-- tab content -->
    <div class="col-md-10 col-sm-10 col-lg-10 tab-content">
        <div class="tab-pane active text-style" id="tabProdutos">
            <h2>Criar novo produto</h2>
            <br>
            <input type="button" name="criarProduto" id="criarProduto" ng-
click="IrParaCriarProduto()"
                   value="Criar novo Produto">
            <br>
            <h2>Editar/Remover produtos</h2>
            <div class="tab-pane fade in table-responsive" id="tab3">
```

```
24.                 <table class="table table-striped table-Orders">
25.                     <thead class="table-Orders-Header">
26.                     <tr>
27.                         <th>ID</th>
28.                         <th>Nome</th>
29.                         <th>Categoria</th>
30.                         <th>Subcategoria</th>
31.                         <th>Marca</th>
32.                         <th>Destaque</th>
33.                         <th>Quantidade</th>
34.                         <th>Preço</th>
35.                         <th>
36.                             <div class="navbar-right">
37.                                 <div class="select-Filter">
38.
39.                                     <select name="filter" ng-
    model="filterProduto" ng-init="filterProduto='nome'">
40.                                         <option value="nome">Nome</option>
41.                                         <option value="preco">Preço</option>
42.                                         <option value="categoria.nome">Categoria</o
    ption>
43.                                         <option value="subCategoria">Subcategoria</
    option>
44.                                         <option value="marca">Marca</option>
45.                                         <option value="destaque">Destaque</option>

46.                                         <option value="quantidade">Quantidade</opti
    on>
47.                                     </select>
48.
49.                                     <span ng-show='OrderByShowProduto' ng-
    click='toggleOrderByProduto()' class="glyphicon glyphicon-arrow-
    up orderByDirectionEncomendas" aria-hidden="true"></span>
50.                                     <span ng-show='!OrderByShowProduto' ng-
    click='toggleOrderByProduto()' class="glyphicon glyphicon-arrow-
    down orderByDirectionEncomendas" aria-hidden="true"></span>
51.                                 </div>
52.                             </div>
53.
54.                         </th>
55.                     </tr>
56.                     </thead>
57.                     <tbody>
58.                     <input type="text" name=procurarProduto" id="procurarProdutoInp
    ut" ng-model="searchProduto">
59.                     <tr ng-
    repeat="produto in produtosFiltrados = (produtos |filter:searchProduto | orderBy :
    filterProduto :!OrderByShowProduto |startFrom:(currentPage-
    1)*itemsPerPage | limitTo:itemsPerPage)">
60.                         <td>{{produto._id}}</td>
61.                         <td>{{produto.nome}}</td>
62.                         <td>{{produto.categoria.nome}}</td>
63.                         <td>{{produto.subCategoria}}</td>
64.                         <td>{{produto.marca}}</td>
65.                         <td>{{produto.destaque}}</td>
66.                         <td>{{produto.quantidade}}</td>
67.                         <td>{{produto.preco}} €</td>
68.                         <td>
69.                             <a ng-
    click="IrParaEditarProduto(produto._id)"><span class="glyphicon glyphicon-edit"
70.                                                                                 ar
    ia-hidden="true"></span></a>
71.                             <a ng-
    click="removerProduto(produto._id)"><span class="glyphicon glyphicon-trash"
72.                                                                             aria-
    hidden="true"></span></a>
```

```
73.                         </td>
74.                     </tr>
75.
76.                     </tbody>
77.
78.                 </table>
79.                 <div class="row pagination-Container paginationDetalhes">
80.                     <div class="col-sm-12  col-xs-12 col-lg-
    12 paginationProfile ">
81.                         <ul uib-pagination total-items="produtos.length" ng-
    model="currentPage"
82.                             items-per-page="itemsPerPage" class="pagination-
    sm" data-boundary-links="true"
83.                             force-ellipses="true" next-text="Próximo" previous-
    text="Anterior" first-text="Início"
84.                             last-text="Fim"></ul>
85.                     </div>
86.                 </div>
87.             </div>
88.         </div>
89.         <div class="tab-pane text-style" id="tabUtilizadores">
90.             <h2>Utilizadores</h2>
91.             <table class="table table-striped table-Orders">
92.                 <thead class="table-Orders-Header">
93.                 <tr>
94.                     <th>ID</th>
95.                     <th>Email</th>
96.                     <th>Username</th>
97.                     <th>Nome Completo</th>
98.                     <th>Role</th>
99.                     <th>
100.                            <div class="navbar-right">
101.                                <div class="select-Filter">
102.                                    <select name="filter" ng-
    model="filterUtilizadores" ng-init="filterUtilizadores='_id'">
103.                                        <option value="_id">ID</option>
104.                                        <option value="email">Email</option>
105.                                        <option value="username">Username</optio
    n>
106.                                        <option value="nomeCompleto">Nome Comple
    to</option>
107.                                        <option value="role">Role</option>
108.                                    </select>
109.
110.                                    <span ng-show='OrderByShowUtilizadores' ng-
    click='toggleOrderByUtilizadores()'
111.                                          class="glyphicon glyphicon-arrow-
    up orderByDirectionUtilizadores"
112.                                          aria-hidden="true"></span>
113.                                    <span ng-show='!OrderByShowUtilizadores' ng-
    click='toggleOrderByUtilizadores()'
114.                                          class="glyphicon glyphicon-arrow-
    down orderByDirectionUtilizadores"
115.                                          aria-hidden="true"></span>
116.                                </div>
117.                            </div>
118.
119.                        </th>
120.                    </tr>
121.                    </thead>
122.                    <tbody>
123.                    <input type="text" name=procurarUtilizadores" id="procurarUt
    ilizadoresInput"
124.                           ng-model="searchUtilizadores">
125.                    <tr ng-
    repeat="utilizador in utilizadoresFiltrados = (utilizadores |filter:searchUtilizado
```

```
   res | orderBy : filterUtilizadores :!OrderByShowUtilizadores |startFrom:(currentPag
   eUtilizadores-1)*itemsPerPageUtilizadores | limitTo:itemsPerPageUtilizadores)">
126.                            <td>{{utilizador._id}}</td>
127.                            <td>{{utilizador.email}}</td>
128.                            <td>{{utilizador.username}}</td>
129.                            <td>{{utilizador.nomeCompleto}}</td>
130.                            <td>{{utilizador.role}}</td>
131.                            <td>
132.                                <div ng show="utilizador.role == 'user'">
133.                                    <input type="button" ng-
   click="adicionarRole(utilizador._id, 'admin')" value="Admin"/>
134.                                </div>
135.                                <div ng-show="utilizador.role == 'admin'">
136.                                    <input type="button" ng-
   click="adicionarRole(utilizador._id, 'user')" value="User"/>
137.                                </div>
138.                            </td>
139.                        </tr>
140.
141.                        </tbody>
142.
143.                    </table>
144.                    <div class="row pagination-Container paginationDetalhes">
145.                        <div class="col-sm-12  col-xs-12 col-lg-
   12 paginationProfile ">
146.                            <ul uib-pagination total-items="utilizadores.length" ng-
   model="currentPageUtilizadores"
147.                                items-per-
   page="itemsPerPageUtilizadores" class="pagination-sm" data-boundary-links="true"
148.                                force-ellipses="true" next-text="Próximo" previous-
   text="Anterior" first-text="Início"
149.                                last-text="Fim"></ul>
150.                        </div>
151.                    </div>
152.                </div>
153.
154.                <div class="tab-pane text-style" id="tabCategorias">
155.
156.                    <!-- Nav tabs -->
157.                    <ul class="nav nav-tabs categoriasTab" role="tablist">
158.                        <li role="presentation" class="active"><a data-
   target="#home" role="tab" data-toggle="tab"
159.                                                                    ng-
   click="atualizarCategorias()">Adicionar SubCategorias</a>
160.                        </li>
161.                        <li role="presentation"><a data-
   target="#profile" role="tab" data-toggle="tab"
162.                                                   ng-
   click="atualizarCategorias()">Remover SubCategorias</a></li>
163.                    </ul>
164.
165.                    <!-- Tab panes -->
166.                    <div class="tab-content">
167.                        <div role="tabpanel" class="tab-pane active" id="home">
168.                            <h2>Adicionar SubCategoria</h2>
169.                            <div class="panel-heading">
170.                                Selecione uma Categoria: <select class="selectCatego
   rias"
171.                                                                ng-
   options="item as item.nome for item in categorias track by item.nome"
172.                                                                ng-
   model="selectedCategoria"></select>
173.                                <div class="adicionarSubCategoria" ng-
   show='selectedCategoria'>
174.                                    <input type="text" name="adcionarSubCategoria" i
   d="" ng-model="SubCategoriaInput">
```

```
175.                                <input ng-
    show='SubCategoriaInput' type="button" name="" id=""
176.                                       value="Adicionar SubCategoria"
177.                                       ng-
    click="adicionarSubCategoria(selectedCategoria._id)">
178.                                </div>
179.
180.                            </div>
181.                        </div>
182.                        <div role="tabpanel" class="tab-pane" id="profile">
183.                            <h2>Remover SubCategoria</h2>
184.                            <div class="panel-heading">
185.                                Selecione uma Categoria: <select class="selectCatego
    rias"
186.                                                                ng-
    options="item as item.nome for item in categorias track by item.nome"
187.                                                                ng-
    model="selectedCategory1"></select>
188.                            </div>
189.                            <div class="adicionarSubCategoria" ng-
    show='selectedCategory1'>
190.                                <ul>
191.                                    <div ng-
    repeat="subCategory1 in selectedCategory1.subCategorias">
192.                                        <li>{{subCategory1.nome}}
193.                                            <a ng-
    click="removerSubCategoria(selectedCategory1 ,subCategory1.nome)"><span
194.                                                    class="glyphicon glyphicon-
    trash" aria-hidden="true"></span></a>
195.                                        </li>
196.                                    </div>
197.                                    <li ng-
    show="selectedCategory1.subCategorias == ''">Não tem Sub-Categorias</li>
198.                                </ul>
199.                            </div>
200.                        </div>
201.                    </div>
202.
203.                    <br>
204.
205.                </div>
206.
207.                <div class="tab-pane text-style" id="tabEncomendas">
208.                    <h2>Encomendas</h2>
209.                    <table class="table table-striped table-Orders">
210.                        <thead class="table-Orders-Header">
211.                        <tr>
212.                            <th>Encomenda N.</th>
213.                            <th>Data</th>
214.                            <th>Total</th>
215.                            <th>Estado da encomenda</th>
216.                            <th>
217.                                <div class="navbar-right">
218.                                    <div class="select-Filter">
219.
220.                                        <select name="filter" ng-model="filter" ng-
    init="filter='data'">
221.                                            <option value="data">Data</option>
222.                                            <option value="precoTotal">Preço</option
    >
223.                                            <option value="estadoEncomenda">Marca</o
    ption>
224.                                        </select>
225.
226.                                        <span ng-show='OrderByShow' ng-
    click='toggleOrderBy()'
```

```
227.                                                    class="glyphicon glyphicon-arrow-
    up orderByDirectionEncomendas"
228.                                                    aria-hidden="true"></span>
229.                                          <span ng-show='!OrderByShow' ng-
    click='toggleOrderBy()'
230.                                                    class="glyphicon glyphicon-arrow-
    down orderByDirectionEncomendas"
231.                                                    aria-hidden="true"></span>
232.                                      </div>
233.                                  </div>
234.
235.                              </th>
236.                          </tr>
237.                          </thead>
238.                          <tbody>
239.                          <input type="text" name="procurarEncomendas" id="procurarEnc
    omendasInput" ng-model="searchEncomendas">
240.                          <tr ng-
    repeat="encomenda in encomendasFiltradas = (encomendas | filter:searchEncomendas |
    orderBy : filter :!OrderByShow | startFrom:(currentPage-
    1)*itemsPerPageEncomendas | limitTo:itemsPerPageEncomendas)">
241.                              <td>{{encomenda._id}}</td>
242.                              <td>{{encomenda.data | date:'medium'}}</td>
243.                              <td>{{encomenda.items.totalPreco}} €</td>
244.                              <td>{{encomenda.estadoEncomenda}}</td>
245.                              <td>
246.                                  <span><a ng-
    click='IrParaEncomendaDetalhes(encomenda._id)'>Ver Detalhes da Encomenda</a></span>

247.                              </td>
248.                          </tr>
249.
250.                          </tbody>
251.
252.                      </table>
253.                      <div class="row pagination-Container paginationDetalhes">
254.                          <div class="col-sm-12  col-xs-12 col-lg-
    12 paginationProfile ">
255.                              <ul uib-pagination total-items="encomendas.length" ng-
    model="currentPageEncomendas"
256.                                  items-per-
    page="itemsPerPageEncomendas" class="pagination-sm" data-boundary-links="true"
257.                                  force-ellipses="true" next-text="Próximo" previous-
    text="Anterior" first-text="Início"
258.                                  last-text="Fim"></ul>
259.                          </div>
260.                      </div>
261.                  </div>
262.                  <div class="tab-pane text-style" id="tabBanner">
263.                      <h2>Adicionar Banner</h2>
264.                      <div class="panel-heading">
265.                          <div class="adicionarSubCategoria imagemBanner">
266.                              <label>Imagem:</label>
267.                              <input type="file" name="file" id="file" ng-
    model="upload.file" ngf-select ngf-max-size="10MB"
268.                                      class="form-control uploadBanner "/>
269.                              <input type="button" value="Criar Banner" ng-
    click="criarBanner()">
270.                          </div>
271.                      </div>
272.
273.                      <h2>Remover Banner</h2>
274.                      <div class="panel-heading">
275.                          Selecione o Banner: <select class="selectCategorias"
276.                                                  ng-
    options="item as item.nome for item in produtosBanner"
```

```
277.                                                    ng-
    model="selectedBanner"></select>
278.                    </div>
279.                    <ul>
280.                        <li class="adminLiBanner" ng-show="selectedBanner">
281.                            <a ng-click="removerBanner(selectedBanner._id)"><span
282.                                    class="glyphicon glyphicon-trash" aria-
    hidden="true"></span></a>
283.                            <img class="adminImagemBanner" ng-
    src="images/uploads/produtosBanner/{{selectedBanner.nome}}">
284.                        </li>
285.                    </ul>
286.                </div>
287.                <div class="tab-pane text-style" id="tabMetodosEnvio">
288.                    <h2>Adicionar Método de Envio</h2>
289.                    <div class="panel-heading">
290.                        <div class="adicionarSubCategoria">
291.                            <form action="" name="MetodoEnvioForm">
292.                                <label>Nome:</label>
293.                                <input type="text" ng-
    model="metodoEnvio.nome" required>
294.                                <label>Preço:</label>
295.                                <input type="number" ng-
    model="metodoEnvio.preco" required>
296.                                <input type="button" value="Criar Metodo" ng-
    click="criarMetodo()">
297.                            </form>
298.                        </div>
299.                    </div>
300.
301.                    <h2>Remover Método de Envio</h2>
302.                    <ul>
303.                        <li class="adminLiBanner" ng-
    repeat="metodos in metodosEnvio">
304.                            Nome: {{metodos.nome}} | Preço: {{metodos.preco}}€
305.                            <a ng-click="removerMetodo(metodos._id)"><span
306.                                    class="glyphicon glyphicon-trash" aria-
    hidden="true"></span></a>
307.                        </li>
308.                    </ul>
309.                </div>
310.            </div>
311.        </div>
```

## **Apêndice A.36** – Controlador do estado 'root.admin'

```
1.  angular.module('projetoGlobal')
2.
3.      .controller('AdminCtrl', ['$scope', '$http', '$state', 'Upload', function ($sco
    pe, $http, $state, Upload) {
4.          $scope.produtos = [];
5.          $scope.encomendas = [];
6.          $scope.utilizadores = [];
7.          $scope.metodosEnvio = {};
8.          $scope.metodoEnvio = {};
9.          $scope.currentPage = 1;
10.         $scope.itemsPerPage = 15;
11.         $scope.currentPageEncomendas = 1;
12.         $scope.itemsPerPageEncomendas = 15;
13.         $scope.currentPageUtilizadores = 1;
14.         $scope.itemsPerPageUtilizadores = 15;
15.         getCategorias();
16.         getProdutos();
```

```
17.         getEncomendas();
18.         getBanner();
19.         getUtilizadores()
20.         getMetodosEnvio();
21.
22.         // procurar todas categorias
23.         function getCategorias() {
24.             $scope.selectedCategoria = undefined;
25.             $scope.selectedCategory1 = undefined;
26.             $scope.categorias = '';
27.             $http({
28.                 method: 'GET',
29.                 url: 'categorias/'
30.             }).then(function successCallback(response) {
31.                     $scope.categorias = response.data;
32.                 },
33.                 function errorCallback(response) {
34.                     console.log("sometingh Failed");
35.                 });
36.         }
37.
38.         function getBanner() {
39.             $http({
40.                 method: 'GET',
41.                 url: 'produtosBanner/'
42.             }).then(function successCallback(response) {
43.                     $scope.produtosBanner = response.data;
44.                 },
45.                 function errorCallback(response) {
46.                     console.log("sometingh Failed");
47.                 });
48.         }
49.
50.
51.         function getMetodosEnvio() {
52.             $http({
53.                 method: 'GET',
54.                 url: 'metodosEnvio/'
55.             }).then(function successCallback(response) {
56.                     $scope.metodosEnvio = response.data;
57.                 },
58.                 function errorCallback(response) {
59.                     console.log("sometingh Failed");
60.                 });
61.         }
62.
63.         $scope.adicionarSubCategoria = function (categoriaID) {
64.             $http.post('admin/criarSubCategoria/' + categoriaID, {"subCategoria": $
    scope.SubCategoriaInput})
65.                 .then(function successCallback(response) {
66.                     alert('Sub-
    Categoria ' + $scope.SubCategoriaInput + ' Adicionada');
67.                     $scope.SubCategoriaInput = '';
68.                     getCategorias();
69.                 }, function errorCallback(response) {
70.                     console.log(response);
71.                 });
72.         }
73.         $scope.removerSubCategoria = function (categoriaNome, subCategoria) {
74.             $http.post('admin/removerSubCategoria/' + categoriaNome._id, {"subCateg
    oria": subCategoria})
75.                 .then(function successCallback(response) {
76.                     getCategorias();
77.                     alert('Sub-Categoria ' + subCategoria + ' Removida');
78.                 }, function errorCallback(response) {
79.                     console.log(response);
```

```
80.                 });
81.         }
82.         $scope.removerProduto = function (produtoID) {
83.             $http.post('admin/removerProduto/' + produtoID)
84.                 .then(function successCallback(response) {
85.                     getProdutos();
86.                 }, function errorCallback(response) {
87.                     console.log(response);
88.                 });
89.         }
90.         $scope.criarBanner = function () {
91.             if ($scope.upload) {
92.                 Upload.upload({ // uploud da image para a base de dados
93.                     url: 'produtosBanner/criarImagem',
94.                     method: 'POST',
95.                     data: $scope.upload
96.                 }).then(function (response) {
97.                     console.log(response);
98.                     $scope.upload = '';
99.                     alert('Upload com Sucesso');
100.                    });
101.                } else {
102.                    alert('Insira a Imagem');
103.                }
104.            }
105.            $scope.removerBanner = function (bannerID) {
106.                $http.post('produtosBanner/remover/' + bannerID)
107.                    .then(function successCallback(response) {
108.                        getBanner();
109.                    }, function errorCallback(response) {
110.                        console.log(response);
111.                    });
112.            }
113.
114.            $scope.adicionarRole = function (userID, role) {
115.                $http.post('admin/utilizadoresRole/' + userID, {role: role})
116.                    .then(function successCallback(response) {
117.                        getUtilizadores();
118.                    }, function errorCallback(response) {
119.                        console.log(response);
120.                    });
121.            }
122.
123.            $scope.criarMetodo = function () {
124.                if ($scope.MetodoEnvioForm.$valid) {
125.                    $scope.metodoEnvio.model = $scope.metodoEnvio.nome.replace(/
    \s/g, '');
126.                    $http.post('admin/metodosEnvio/adicionar', {metodo: $scope.m
    etodoEnvio})
127.                        .then(function successCallback(response) {
128.                            $scope.metodoEnvio.nome = '';
129.                            $scope.metodoEnvio.preco = '';
130.                            getMetodosEnvio();
131.                            alert("Metodo de Envio adicionado com sucesso.")
132.                        }, function errorCallback(response) {
133.                            console.log(response);
134.                        });
135.                }
136.                else {
137.                    alert('Preencha o Nome e o Preço do método de envio.')
138.                }
139.            }
140.
141.            $scope.removerMetodo = function (userID) {
142.                $http.post('admin/metodosEnvio/remover/' + userID)
143.                    .then(function successCallback(response) {
```

```
144.                         getMetodosEnvio();
145.                     }, function errorCallback(response) {
146.                         console.log(response);
147.                     });
148.             }
149.
150.
151.             $scope.atualizarCategorias = function () {
152.                 getCategorias();
153.             }
154.             $scope.IrParaCriarProduto = function () {
155.                 $state.go('root.admin.criarProduto', {reload: true});
156.             }
157.             $scope.IrParaEditarProduto = function (produtoid) {
158.                 $state.go('root.admin.editarProduto', {produtoID: produtoid}, {r
    eload: true});
159.             }
160.             $scope.IrParaEncomendaDetalhes = function (encomendaid) {
161.                 $state.go('root.admin.encomendaDetalhes', {encomendaID: encomend
    aid}, {reload: true});
162.             }
163.             // Toggle para o OrderBy Directions
164.             $scope.toggleOrderBy = function () {
165.                 $scope.OrderByShow = !$scope.OrderByShow;
166.             };
167.             $scope.toggleOrderByProduto = function () {
168.                 $scope.OrderByShowProduto = !$scope.OrderByShowProduto;
169.             };
170.             $scope.toggleOrderByUtilizadores = function () {
171.                 $scope.OrderByShowUtilizadores = !$scope.OrderByShowUtilizadores
    ;
172.             };
173.
174.             // Vai buscar os produtos
175.             function getProdutos() {
176.                 $http({
177.                     method: 'GET',
178.                     url: 'produtos/Categorias'
179.                 }).then(function (response) {
180.                     $scope.produtos = response.data;
181.                 });
182.             }
183.
184.             function getUtilizadores() {
185.                 $http({
186.                     method: 'GET',
187.                     url: 'utilizadores/'
188.                 }).then(function (response) {
189.                     $scope.utilizadores = response.data;
190.                 });
191.             }
192.
193.             // Vai buscar os produtos
194.             function getEncomendas() {
195.                 $http({
196.                     method: 'GET',
197.                     url: 'encomendas/'
198.                 }).then(function (response) {
199.                     $scope.encomendas = response.data;
200.                 });
201.             }
202.
203.         }]);
```

## **Apêndice A.37** – HTML da view do estado 'root.admin.CriarProduto'

```
1.  <div class="row cart criarProduto">
2.      <a class="btn btn-default" ng-
    click="voltarAtras()"><span class="glyphicon glyphicon-arrow-left" aria-
    hidden="true"></span> Voltar Atras</a>
3.      <h1>Criar Produto</h1>
4.      <ul class="errors"></ul>
5.      <form name="criarProdutoForm">
6.          <div class="form-group">
7.              <label>Nome:</label>
8.              <input name="nome" type="text" class="form-control" ng-
    model="criarProduto.nome" required/>
9.          </div>
10.         <div class="form-group">
11.             <label>Category</label>
12.             <select name="category" class="form-control" ng-
    options="item as item.nome for item in categorias track by item.nome"  ng-
    model="criarProduto.categoria" required></select>
13.         </div>
14.         <div class="form-group">
15.             <label>Subcategory</label>
16.             <select name="subcategory" class="form-control" ng-
    options="item as item.nome for item in criarProduto.categoria.subCategorias track b
    y item.nome"  ng-model="criarProduto.subCategoria"></select>
17.         </div>
18.         <div class="form-group">
19.             <label>Marca</label>
20.             <input name="marca" type="text" class="form-control"  ng-
    model="criarProduto.marca"required />
21.         </div>
22.         <div class="form-group">
23.             <label>Destaque:</label>
24.             <select name="destaque" class="form-control" ng-
    model="criarProduto.destaque" required>
25.                 <option ng-value="true">True</option>
26.                 <option ng-value="false">False</option>
27.             </select>
28.         </div>
29.         <div class="form-group">
30.             <label>Quantidade:</label>
31.             <input name="quantidade" type="number" class="form-control" ng-
    model="criarProduto.quantidade" required/>
32.         </div>
33.         <div class="form-group">
34.             <label>Preco:</label>
35.             <input name="preco" type="number" class="form-control" ng-
    model="criarProduto.preco" required/>
36.         </div>
37.         <div class="form-group">
38.             <label>Descricao:</label>
39.             <div ckeditor="options" ng-model="criarProduto.descricao" ></div>
40.         </div>
41.         <input type="button" value="Save" ng-
    click="criarProdutoInput()" class="btn btn-default"/>
42.     </form>
43. </div>
```

## **Apêndice A.38** – Controlador do estado 'root.admin.CriarProduto'

```
1.  angular.module('projetoGlobal')
2.
```

```
3.      .controller('AdminCriarProdutoCtrl', ['$scope', '$http', '$state', function ($s
    cope, $http, $state) {
4.
5.          $scope.voltarAtras = function () {
6.              $state.go('root.admin', {reload: true});
7.          }
8.          // Editor options.
9.          $scope.options = {
10.             language: 'pt',
11.             allowedContent: true,
12.             entities: false
13.         };
14.         $http({
15.             method: 'GET',
16.             url: '/categorias'
17.         }).then(function (response) {
18.             $scope.categorias = response.data;
19.         });
20.
21.         $scope.criarProdutoInput = function () {
22.             if ($scope.criarProduto.descricao && $scope.criarProdutoForm.$valid) {

23.                 var categoria = $scope.criarProduto.categoria;
24.                 var subCategoria = $scope.criarProduto.subCategoria;
25.                 $scope.criarProduto.categoria = categoria._id;
26.                 $scope.criarProduto.subCategoria = subCategoria.nome;
27.                 $http.post('admin/criarProduto', $scope.criarProduto)
28.                     .then(function successCallback(response) {
29.                             $scope.criarProduto = '';
30.                             $state.go('root.admin', {reload: true});
31.                         },
32.                         function errorCallback(response) {
33.                             console.log("Algo Falhou");
34.                         });
35.             }
36.             else alert("Preencha tudo.")
37.         };
38.     }]);
```

## **Apêndice A.39** – HTML da view do estado 'root.admin.EditarProduto'

```
1.  <div class="row cart editarProduto">
2.      <a class="btn btn-default" ng-
    click="voltarAtras()"><span class="glyphicon glyphicon-arrow-left" aria-
    hidden="true"></span> Voltar Atras</a>
3.      <h1>Editar Produto</h1>
4.      <ul class="errors"></ul>
5.      <form name="editarProdutoForm">
6.          <div class="form-group">
7.              <label>Nome:</label>
8.              <input name="nome" type="text" class="form-control" ng-
    model="editarProduto.nome" required/>
9.          </div>
10.         <div class="form-group">
11.             <label>Category</label>
12.             <select name="category" class="form-control" ng-
    options="item as item.nome for item in categorias track by item.nome"  ng-
    model="editarProduto.categoria" required></select>
13.         </div>
14.         <div class="form-group">
15.             <label>Subcategory</label>
```

```
16.             <select name="subcategory" class="form-control" ng-
    options="item as item.nome for item in editarProduto.categoria.subCategorias track
    by item.nome"  ng-model="editarProduto.subCategoria"></select>
17.         </div>
18.         <div class="form-group">
19.             <label>Marca</label>
20.             <input name="marca" type="text" class="form-control"  ng-
    model="editarProduto.marca"required />
21.         </div>
22.         <div class="form-group">
23.             <label>Destaque:</label>
24.             <select name="destaque" class="form-control" ng-
    model="editarProduto.destaque" required>
25.                 <option ng-value="true">True</option>
26.                 <option ng-value="false">False</option>
27.             </select>
28.         </div>
29.         <div class="form-group">
30.             <label>Quantidade:</label>
31.             <input name="quantidade" type="number" class="form-control" ng-
    model="editarProduto.quantidade" required/>
32.         </div>
33.         <div class="form-group">
34.             <label>Preco:</label>
35.             <input name="preco" type="number" class="form-control" ng-
    model="editarProduto.preco" required/>
36.         </div>
37.
38.         <div class="form-group">
39.             <label>Descricao:</label>
40.             <div ckeditor="options" ng-model="editarProduto.descricao" ></div>
41.         </div>
42.         <div class="form-group">
43.             <label>Imagem:</label>
44.             <input type="file" name="file" id="file" ng-model="upload.file" ngf-
    select ngf-max-size="10MB" class="form-control" />
45.         </div>
46.
47.         <input type="button" value="Save" ng-
    click="editarProdutoInput()" class="btn btn-default"/>
48.     </form>
49. </div>
```

## **Apêndice A.40** – Controlador do estado 'root.admin.EditarProduto'

```
1.  angular.module('projetoGlobal')
2.
3.      .controller('AdminEditarProdutoCtrl', ['$scope', '$http', '$state', '$statePara
    ms', 'Upload', function ($scope, $http, $state, $stateParams, Upload) {
4.          $scope.voltarAtras = function () {
5.              $state.go('root.admin', {reload: true});
6.          }
7.
8.          $http({
9.              method: 'GET',
10.             url: '/categorias'
11.         }).then(function (response) {
12.             $scope.categorias = response.data;
13.         });
14.
15.         $http({
16.             method: 'GET',
17.             url: 'produtos/detalhes/' + $stateParams.produtoID
```

```
        }).then(function (response) {
            $scope.editarProduto = response.data.produto;
        });

        $scope.editarProdutoInput = function () {
            if ($scope.editarProduto.descricao && $scope.editarProdutoForm.$valid)
{
                var categoria = $scope.editarProduto.categoria;
                var subCategoria = $scope.editarProduto.subCategoria;
                $scope.editarProduto.categoria = categoria._id;
                $scope.editarProduto.subCategoria = subCategoria.nome || $scope.edi
tarProduto.subCategoria;
                $http.post('admin/editarProduto', $scope.editarProduto)
                    .then(function successCallback(response) {
                            if ($scope.upload) {
                                Upload.upload({ // uploud da image para a base de d
ados
                                    url: 'admin/alterarImagemProduto/' + $scope.edi
tarProduto._id,
                                    method: 'POST',
                                    data: $scope.upload
                                }).then(function (response) {
                                    $state.go('root.admin', {reload: true});
                                });
                            } else {
                                $state.go('root.admin', {reload: true});
                            }
                        },
                        function errorCallback(response) {
                            console.log("Algo Falhou");
                        });
            }
            else alert("Preencha tudo.");
        }

    }]);
```

## Apêndice A.41 – HTML da view do estado 'root.admin.encomendaDetalhes'

```
<div class="cart adminEncomendaDetalhes">
    <a class="btn btn-default btn-orders-Detalhes" ui-sref="root.admin" ui-sref-
opts="{reload: true}"><span
            class="glyphicon glyphicon-arrow-left" aria-
hidden="true"></span> Voltar Atras</a>
    <div class="row">
        <div class="col-md-9 col-lg-9 personal-Information">
            <table class="table table-user-information">
                <tbody>
                <tr>
                    <th>Id do Pagamento:</th>
                    <td>{{encomenda.pagamentoID}}</td>
                </tr>
                <tr>
                    <th>Tipo de Pagamento:</th>
                    <td>{{encomenda.tipoDePagamento}}</td>
                </tr>
                <tr>
                    <th>Encomenda Realizada em:</th>
                    <td>{{encomenda.data | date:'medium'}}</td>
                </tr>
                <tr>
```

```
21.                     <th>Morada:</th>
22.                     <td>{{encomenda.moradaShipping.morada}}</td>
23.                 </tr>
24.                 <tr>
25.                     <th>Localidade:</th>
26.                     <td>{{encomenda.moradaShipping.localidade}}</td>
27.                 </tr>
28.                 <tr>
29.                     <th>Código Postal:</th>
30.                     <td>{{encomenda.moradaShipping.codigoPostal}}</td>
31.                 </tr>
32.                 <tr>
33.                     <th>Telefone:</th>
34.                     <td>{{encomenda.moradaShipping.telefone}}</td>
35.                 </tr>
36.                 <tr>
37.                     <th>Estado da Encomenda:</th>
38.                     <td>{{encomenda.estadoEncomenda}}
39.                         <div class="encomendaEstadoButton" ng-
    hide="encomenda.estadoEncomenda == 'Enviada'">
40.                             <input  type="button" name="alterarEstado" id="" value=
    "Encomenda Enviada"
41.                                 ng-click="alterarEstadoEncomenda()">
42.                         </div>
43.                     </td>
44.                 </tr>
45.                 </tbody>
46.             </table>
47.         </div>
48.     </div>
49.     <div class="row">
50.         <div class="col-md-12 col-lg-12 personal-Information">
51.             <div class="tab-pane fade in table-responsive" id="tab3">
52.                 <table class="table table-striped table-bordered table-Orders-
    Detalhes">
53.                     <colgroup>
54.                         <col width='50%'>
55.                         <col width='10%'>
56.                         <col width='10%'>
57.                         <col width='10%'>
58.                     </colgroup>
59.                     <thead class="table-Orders-Header">
60.                     <tr>
61.                         <th>Detalhes do Produto</th>
62.                         <th>Preço</th>
63.                         <th>Quantidade</th>
64.                         <th>Sub-Total</th>
65.                     </tr>
66.                     </thead>
67.
68.                     <tbody>
69.                     <tr ng-repeat="produto in encomenda.items.items">
70.                         <td>
71.                             <div class="table-Orders-Detalhes">
72.                                 <div class="col-lg-3 col-md-3 col-sm-3 table-
    Orders-Detalhes-Imagem">
73.                                     <img ng-
    src="images/uploads/produtos/{{produto.item.imagem.file.filename}}">
74.                                 </div>
75.                                 <div class="col-lg-8 col-md-4 col-sm-4 table-
    Orders-Detalhes-Produto">
76.                                     <span class="table-Orders-Detalhes-Produto-
    Nome">{{ produto.item.nome }}</span>
77.                                 </div>
78.                             </div>
79.                         </td>
```

```
                        <td>
                            <div class="table-Orders-Detalhes-Produto2">
                                <span>{{produto.item.preco}} €</span>
                            </div>
                        </td>
                        <td>
                            <div class="table-Orders-Detalhes-Produto2">
                                <span>x{{produto.qty}}</span>
                            </div>
                        </td>
                        <td>
                            <div class="table-Orders-Detalhes-Produto2">

                                <span>{{produto.preco}} €</span>
                            </div>
                        </td>
                    </tr>
                    </tbody>

                    <tfoot>
                        <tr>
                            <th colspan="2"></th>
                            <th class="table-Orders-Detalhes-Produto-
Total" colspan="1">Total:</th>
                            <td class="table-Orders-Detalhes-Produto-
Total">{{encomenda.precoTotal}} €</td>
                        </tr>
                        </tfoot>
                    </table>
                </div>
            </div>
        </div>

    </div>
```

## Apêndice A.42 – Controlador do estado 'root.admin.encomendaDetalhes'

```
angular.module('projetoGlobal')

    .controller('AdminEncomendaDetalhesCtrl', ['$scope', '$http', '$stateParams', f
unction ($scope, $http, $stateParams) {
        $scope.encomendaID = $stateParams.encomendaID;
        $scope.encomenda;
        // Procura o produto que vem passado como parametro do estado anterior
        $http({
            method: 'GET',
            url: 'encomendas/encomendaDetalhe/' + $scope.encomendaID
        }).then(function (response) {
            $scope.encomenda = response.data;
        });

        $scope.alterarEstadoEncomenda = function () {
            $http({
                method: 'POST',
                url: 'admin/encomendaAlterar/' + $scope.encomendaID
            }).then(function (response) {
                $scope.encomenda = response.data;
            });
        }
    }]);
```

## **Apêndice A.43** – HTML da view do estado 'root.about'

```
1.  <div class="row">
2.      <div class="col-lg-12">
3.          <h1 class="page-header">Sobre Market Place</h1><br>
4.          <p>Este website foi desenvolvido no âmbito do projeto global para a Licenci
    atura em Informática, com o objetivo de realizar um website de comércio eletrónico.
    </p>
5.      </div>
6.  </div>
7.  <div class="row">
8.      <div class="col-lg-12"><h2 class="page-
    header">Desenvolvido por:</h2><br></div>
9.      <div class="col-lg-4 col-lg-offset-4 col-sm-6 text-center">
10.         <img class="img-circle img-responsive img-
    center" src="/images/FabioAnselmo.jpg" alt="Fabio Anselmo Imagem">
11.         <h3>Fábio Anselmo Nº 2040</h3>
12.         <h3>Licenciatura em Informática </h3>
13.     </div>
14. </div>
```

## **Apêndice A.44** – HTML da view do estado 'root.contatos'

```
1.  <div class="jumbotron jumbotron-sm jumbotronContatos">
2.      <div class="container">
3.          <div class="row">
4.              <div class="col-sm-12 col-lg-
    12 contatos"><h1 class="h1">Contatos</h1></div>
5.          </div>
6.      </div>
7.  </div>
8.  <div class="container">
9.      <div class="row">
10.         <div class="col-md-8">
11.             <div class="well well-sm" style="overflow: hidden">
12.                 <iframe width="700" height="450" frameborder="0" style="border:0" s
    rc="https://www.google.com/maps/embed/v1/place?q=place_id:EiJBdi4gZGEgTGliZXJkYWRlL
    CBMaXNib2EsIFBvcnR1Z2Fs&key=AIzaSyCUGdKKV4A6nnr4P3DinK_eEzECVa1x3pE"allowfullscreen
    ></iframe>
13.             </div>
14.         </div>
15.         <div class="col-md-4">
16.             <form>
17.                 <legend><span class="glyphicon glyphicon-
    globe"></span>Loja</legend>
18.                 <address>
19.                     <strong>Market Place</strong><br>
20.                     Avenida da Liberdade, Lisboa<br>
21.                     Telefone:
22.                     210000333
23.                 </address>
24.                 <address>
25.                     <strong>Email</strong><br>
26.                     <a href="mailto:fabioanselmo1994@gmail.com">fabioanselmo1994@gm
    ail.com</a>
27.                 </address>
28.             </form>
29.         </div>
30.     </div>
31. </div>
```