



ISTEC - Instituto Superior de Tecnologias Avançadas



Bon Appétit App

Licenciatura de Informática

Coordenador: Prof. Dr. Pedro Brandão

Realizado Por: Luís Pedro Leal Ribeiro N° 1901 Turma Noite

Lisboa

2014/2015

ISTEC - Instituto Superior de Tecnologias Avançadas



Bon Appétit App

Luís Pedro Leal Ribeiro

Nº 1901 Turma Noite

Dissertação de Projeto Global apresentado para o comprimento proposto necessário para obtenção do grau de licenciado em Informática no Instituto Superior de Tecnologias Avançadas, realizado sobre a coordenação do Prof. Dr. Pedro Brandão.

O presente trabalho encontra-se ao abrigo do novo acordo autográfico.

Lisboa, 2015

Dedicatória

Dedico este trabalho a minha família, que são o meu bem mais precioso.

Pelo que me ensinam e transmitem.

Pelo amor e carinho incondicional.

Agradecimentos

Ao meu melhor amigo Bruno Grácio, pela orientação, pelo conhecimento, pela disponibilidade e também paciência, sem o seu auxílio o projeto não seria o mesmo.

Ao Prof. Dr. Pedro Brandão, pela disponibilidade demonstrada e orientação do projeto, assim como a exigência que nos incuti-o ao longo do tempo no instituto.

Aos meus colegas de curso pelo incentivo e companheirismo ao longo destes três anos.

Aos restantes professores do ISTECC que contribuíram para enriquecer o meu conhecimento.

A minha família pelo apoio, amor e carinho incondicional.

Resumo

O desenvolvimento de aplicações para *smartphones*, mesmo sendo uma área recente, mas que devido a uma forma natural de evolução e massificação de utilizadores, têm um grande impacto nos dias que decorrem. Através destas aplicações móveis podemos efetuar inúmeras possibilidades, quer a nível de produção, quer a nível de lazer. Um das grandes vertentes passa pela utilização das redes sociais, que por uma forma inata, molda a sociedade de hoje. Com esta tendência a vir cada vez mais evidenciada, surge a ideia do projeto, a realização de uma aplicação móvel para o sistema operativo Android.

Esta dissertação aborda o desenvolvimento de uma aplicação para Android. É primeiro feito uma abordagem, analisando o estado de arte das tecnologias envolvidas na realização do projeto. Numa segunda abordagem, a conceção da aplicação, através de explicação dos conceitos pelos comportamentos apresentados, com a colaboração de figuras ilustrativas, focando os principais temas do trabalho, e enfatizando as três principais características, a aplicação Android, a aplicação Servidor e o *Backoffice*.

Palavras-Chave: Android, Aplicação, Servidor, Funcionalidade.

Abstract

The development of applications for smartphones, even though a recent area, but that due to a natural way of evolution and mass of users, have an impact in these days. Through these mobile applications can make numerous possibilities, both in terms of production, both in terms of leisure. A major plank involves the use of social networks, which in an innate way, shapes society today. With this trend to come increasingly evident, there is the idea behind of the project, the realization of a mobile application for the Android operating system.

This dissertation discusses the development of an application for Android. It first made an approach, analyzing the state of the art about technologies involved in carrying out the project. In a second approach, the application of design through explanation of the concepts for the actions presented, with the help of illustrative figures, focusing on the main themes of the work, and emphasizing the three main topic's, the Android application, server application and the Backoffice.

Keywords: Android, Application, Server, Functionality.

Abreviaturas

QRCode – Quick Response Code.

HTTP – Hipertext Transfer Protocol.

JSON – Javascript Object Notation.

SDK – Software Development Kit.

IDE – Integrated Development Environment.

API – Application Programming Interface.

TI – Tecnologias de Informação.

PaaS – Platform as a Service.

SO – Sistema Operativo.

Índice Geral

Dedicatória	iii
Agradecimentos	iv
Resumo	v
Abstract	vi
Abreviaturas	vii
Índice de Figuras	x
Índice de Quadros	xi
Introdução	1
Estado da Arte	3
1.1 – Sistema operativo <i>Android</i>	3
1.1.1 – Números no Mundo	3
1.1.2 – Código-fonte	3
1.1.3 – Versões	4
1.1.4 – Linguagens de programação	4
1.1.4.1 – Java	4
1.1.5 – Meios para alcançar o resultado final	5
1.1.5.1 - Android Studio	5
1.1.5.2 – Eclipse + Plugin ADT (Android Developer Tools)	5
1.1.5.3 – App Inventor 2	6
1.1.6 – Google Play	7
1.2 – Abordagem ao problema	7
1.2.1 – Aplicação	7
1.2.2 – Tecnologias	7
1.2.2.1 – QRCode (Quick Response Code)	7
1.2.2.2 – NFC (Near Field Communication)	8
1.2.2.3 – Google Places	8
1.2.2.4 – Google Cloud Messaging (GCM)	8
1.2.2.5 – SQLite	8
1.2.2.6 – Facebook	8
1.3 – Aplicações no Mercado	9
Contextualização	10
Desenvolvimento do Projeto	13
Aplicação Android	13

Introdução	13
Primeiro Ecrã	13
Iniciar Pedido.....	21
Verificar Conta Corrente.....	55
Reservar Mesa.....	62
Perto de si.....	78
Aplicação Servidor.....	87
Introdução	87
Cloud Computing.....	87
Heroku.....	88
Estrutura da aplicação	88
Ruby on Rails.....	92
Base de Dados.....	93
Estrutura Aplicacional.....	94
BackOffice	107
Introdução	107
Active Admin.....	107
Conclusão.....	115
Bibliografia	116
Anexos	118

Índice de Figuras

Figura 1 - Estatística de aplicações disponíveis por loja.	11
Figura 2 - Estatística de equipamentos vendidos por Sistema Operativo.	12
Figura 3 - Ecrã de um smartphone com a aplicação Bon Appétit instalada.	14
Figura 4 - Ecrã principal da aplicação	14
Figura 5 - A ler o QRCode.....	22
Figura 6 - QRCode encontrado e lido.	22
Figura 7 - Exemplo de um QRCode utilizado	23
Figura 8 - Menu do Restaurante Mineirão.	24
Figura 9 - Menu de produtos que se encontram dentro da categoria Carne.	30
Figura 10 - Ecrã de visualização de um produto.....	33
Figura 11 - Ecrã de partilha de informação no Facebook	41
Figura 12 - Produto partilhado já na página do Facebook.	41
Figura 13 - Ecrã com os produtos do pedido.	42
Figura 14 - Janela de POP UP de eliminação de um produto ao pedido.	54
Figura 15 - Mensagem a informar que a quantidade foi alterada.	55
Figura 16 - Mensagem ao adicionar um produto.	55
Figura 17 - Ecrã da funcionalidade Conta Corrente.	56
Figura 18 - Ecrã da funcionalidade Reservar Mesa	63
Figura 19 - Ecrã de produto, visto através da opção Ver Ementa.....	64
Figura 20 - Ecrã de envio de e-mail.....	65
Figura 21 - Menu de escolha da aplicação interna de envio de e-mail.	66
Figura 22 - Ecrã da aplicação interna do SO Android para envio de e-mail.	66
Figura 23 - Ecrã inicial da funcionalidade Perto de si.	79
Figura 24 - Restaurantes referenciados no mapa.	79
Figura 25 - IDE RubyMine.	89
Figura 26 - Estrutura da aplicação servidor.	90
Figura 27 - Diagrama da Base de Dados.	93
Figura 28 - Representação gráfica da Estrutura Aplicacional.	94
Figura 29 - Diagrama de Sequência dos pedidos.....	95
Figura 30 - Interface do Active Admin para os Produtos.	108
Figura 31 - Interface do Active Admin para os Restaurantes.	108

Índice de Quadros

Quadro 1 - Mensagem no formato JSON.....	26
Quadro 2 - Código responsável pela mensagem TOAST.....	55
Quadro 3 - Código com as alterações efetuadas ao ficheiro manifesto.	80

Introdução

Atualmente, os *smartphones* estão a mudar a forma como as pessoas interagem entre si e com o mundo. Pode até se dizer que em certa parte, molda os comportamentos da sociedade, criando uma dependência dos mesmos. Podemos pôr dois assuntos em relevância para este facto, o primeiro, as redes sociais, que vieram trazer uma partilha constante de informação acerca de tudo, e em qualquer sítio. Em segundo a facilidade de utilização e inúmeras possibilidades, através das aplicações que existem para estes aparelhos “inteligentes”, que muitas das vezes não têm qualquer custo para a sua utilização.

Este paradigma começou em 2007 e 2008, com o acontecimento de duas situações que catapultaram esta nova era, primeiro o lançamento do iPhone da *Apple*, e segundo o lançamento do SO Android por parte da *Google*, a partir desta altura criou-se uma nova comunidade de programadores de aplicações móveis, que através de um local centralizado, por exemplo a Google Play (loja online para aplicações em Android), massificar a entrega e disponibilidade de aplicações ao utilizador.

O desenvolvimento de aplicações móveis para *smartphones*, veio trazer novas dificuldades, porque é diferente da programação que até a data se fazia para aplicações *desktop*, uma das grandes diferenças é o facto de o ecrã de utilização ser na maioria das vezes mais pequeno, de reduzido tamanho. Outro aspeto importante das aplicações é que têm que ser apelativas e de fácil leitura/utilização. No entanto estes mesmos desafios e dificuldades têm sido amplamente superados, devido a popularidade que alcançam no dia de hoje.

Neste momento podemos afirmar que temos três grandes sistemas operativos que utilizam este meio de interação através de aplicações que são: o Android da *Google*, o IOS da *Apple* e por último o Windows Phone da *Microsoft*. Neste aspeto temos que mencionar o facto que quando o programador desenvolve uma aplicação essa mesma aplicação só irá funcionar num dos sistemas operativos. esse é um dos grandes problemas a nível de aplicações móveis, ou seja, a fragmentação de conteúdos.

Com este propósito podemos afirmar que o presente trabalho, consiste num grande objetivo, que passa por criar uma aplicação móvel para o sistema operativo Android, com uma interface gráfica de fácil utilização/leitura para o utilizador, que permita efetuar pedidos de produtos a um estabelecimento de restauração, entre outras funcionalidades.

Este trabalho encontra-se estruturado sobre um plano previamente fornecido, para o tipo de projeto a apresentar. Tendo em conta esse mesmo plano, o trabalho é composto por cinco capítulos. No primeiro capítulo é apresentado uma breve introdução ao tema do projeto, objetivo e estrutura do trabalho apresentado. No segundo capítulo é debatido o estado de arte, os fundamentos teóricos que estão adjacentes ao projeto como um todo. No terceiro capítulo, é feito uma explicação da contextualização do projeto enquanto ideia. No quarto capítulo é demonstrado todo o desenvolvimento do projeto com três grandes subcapítulos, que são: aplicação Android, aplicação Servidor e *Backoffice*. No último capítulo reside a conclusão e apreciação final deste trabalho.

O método de realização deste projeto, foi através de um planeamento por fases, com isto conseguiu-se, de forma evolutiva alcançar o objetivo proposto. Numa primeira fase, recaiu sobre a escolha do tema proposto pelo Prof. Dr. Pedro Brandão (Coordenador de Projeto), passando depois para a ideia dentro desse mesmo tema, e assim com isto estava fundado o projeto. Na segunda fase foi feita toda a pesquisa de literatura, e procura de conhecimentos que permitissem a realização deste trabalho. Na terceira fase foi a realização do trabalho prático do projeto, mais propriamente a concessão da aplicação móvel. Na quarta e última fase foi enfatizado o trabalho escrito em relação ao projeto, sendo que esta vertente de trabalho foi sendo feita ao longo das últimas duas fases, mas apenas na quarta fase foi realizado na íntegra.

Estado da Arte

Neste capítulo é elaborado uma revisão do estado de arte, estudo bibliográfico em relação ao sistema operativo *Android* e as suas tecnologias, para que seja possível a resolução do problema pelo trabalho proposto. Vai ser abordado outros trabalhos que neste momento estão próximos do objetivo final da aplicação a elaborar, e projetos em desenvolvimento para o mesmo propósito. Ainda nesta secção são também identificados as diferentes abordagens, vantagens e desvantagens e os meios para alcançar o resultado final.

1.1 – Sistema operativo *Android*

O *Android* é um sistema operativo móvel baseado no núcleo *Linux*, que atualmente está a ser desenvolvido pela Open Handset Alliance, liderada pela Google Inc. e outras empresas. Com uma interface de utilizador baseada na manipulação direta, o *Android* é pensado e orientado principalmente para dispositivos móveis com um ecrã sensível ao toque para interagir, tais como *smartphones* ou *tablets*. Nós dias de hoje e devido a sua versatilidade já consegue abranger outros tipos de dispositivos sensíveis ao toque tais como, televisões (*Android TV*), auto radio (*Android Auto*), relógios de pulso (*Android Wear*), óculos (*Google Glass*) e outros eletrodomésticos (por exemplo, Frigorífico). (Queirós, 2013)

1.1.1 – Números no Mundo

Através da sua conferência anual intitulada de *Google I/O* em 2014, apresentou a informação que existe mais de 1 bilhão de utilizadores *Android* por mês, a isto muito se deve a diversidade de equipamentos vendidos com este sistema operativo, porque não só a Google disponibiliza equipamentos com o mesmo, mas como outras grandes fabricantes de dispositivos móveis como por exemplo, Samsung, HTC e LG entre muito outros. (Kahn, 2014)

1.1.2 – Código-fonte

Uma das suas características é ser *open source*, ou seja, é de código-fonte aberto e disponibilizado pela própria Google assim que as versões estão prontas, o que permite as várias empresas que utilizam o sistema operativo nos seus dispositivos de o alterarem sem quaisquer custos a nível de licenciamento, ao invés de por exemplo do sistema operativo *iOS* que é

desenvolvido e comercializado pela Apple, em equipamentos que e apenas dispõem. (Alliance, 2007)

1.1.3 – Versões

Desde o seu nascimento, com a versão 1.0 que foi lançada em setembro de 2008 (1^a versão comercial), a sua evolução apareceu sobre a forma de versões com nomes característicos de doces (por ordem alfabética), em que cada versão lançada além de apresentar melhorias de performance e/ou segurança, acrescenta sempre mais funções nativas, como por exemplo, acesso a censores de batimento cardíaco. Algumas versões do *Android* até ao momento:

- 1.5 *Cupcake*.
- 1.6 *Donut*.
- 2.0 *Eclair*.
- 2.2 *Froyo*.
- 2.3 *Gingerbread*.
- 3.0 *Honeycomb*.
- 4.0 *Ice Cream Sandwich*.
- 4.1 *KitKat*.
- 5.0 *Lollipop* (Versão Atual).
- 6.0 *Marshmallow* (Próxima Versão). (Queirós, 2014) (Android, s.d.)

1.1.4 – Linguagens de programação

Apesar do *Android*, na sua base tendo o núcleo *Linux* que é 90% escrito em *C*, e o restante em *Assembly* e outras, as aplicações para o mesmo são na sua maioria, ou quase todas, escritas em *Java* que será a linguagem que vai ser utilizada para programar a aplicação que irá ser criada. (Clare, 2012)

1.1.4.1 – Java

Java é uma linguagem de programação orientada aos objetos, que teve o seu nascimento na década de 1990, por uma equipa de engenheiros da Sun MicroSystems, que foi chefiada por James Gosling, estando o seu código-fonte escrito em *C++*, sendo uma das linguagens de programação mais populares nos dias de hoje. As suas principais características assentam em:

- Orientação a objetos – Baseado no modelo de simular.

- Portabilidade – Independência de plataforma – “*escreve uma vez, execute em qualquer lugar*”.
- Recursos de rede – Possui uma extensa biblioteca de rotinas que facilitam a cooperação com protocolos de rede (por exemplo, *TCP/IP, HTTP e FTP*).
- Segurança – Pode executar programas via rede com restrições de execução. (Jesus, 2013) (Carvalho, 2012)

1.1.5 – Meios para alcançar o resultado final

Tendo em conta o sistema operativo, e as suas características, e em prol da linguagem de programação utilizada para efetuar a aplicação, neste momento existem pelo menos 3 plataformas possíveis para o desenvolvimento da mesma (*IDE*), que são:

1.1.5.1 - Android Studio

O *Android Studio* é uma plataforma de desenvolvimento para *Android*, criada pela Google. A plataforma é semelhante ao popular e existente *Eclipse*, com *ADT Plugin*, oferecendo as melhores ferramentas e funcionalidades aos programadores, a ideia por-de-trás deste *IDE* desenvolvido recentemente foi a fácil migração de quem vêm do *eclipse* dai as suas parecenças serem muitas. Segundo a própria Google, com o *Android Studio* a programação para *Android* ficou mais simples e rápida. No dia 8 de dezembro de 2014 foi lançada a versão 1.0 deste *software* (Versão estável), as suas principais características e/ou vantagens são:

- Instalação automatizada.
- Importação de amostras de programas (*samples*) e importação de código externo.
- Análise de código e interface gráfico melhorado.
- Análise da performance das aplicações através do recurso *Memory Monitor*.
- Interação com os serviços da Google, como por exemplo o *Google Cloud*.
- Migração de projetos diretamente do *Eclipse*. (Protalinski, 2014)

1.1.5.2 – Eclipse + Plugin ADT (Android Developer Tools)

O *Eclipse* é outra plataforma de desenvolvimento em *Java*, mas que através de *plugins* permite o desenvolvimento em outras linguagens de programação, como por exemplo, *C/C++*, *PHP*, *ColdFusion*, *Python* entre outras. Tal acontece com a plataforma *Android*, através do *plugin ADT*. Este projeto de *IDE* foi iniciado pela IBM que desenvolveu a primeira versão do

software e doou-o como programa livre para a comunidade, hoje em dia o *Eclipse* é o *IDE Java* mais utilizado no mundo. Possui uma característica marcante a nível do seu interface gráfico usando o *SWT* em vés do *Swing*, tendo uma forte orientação para utilização de *plugins* para acrescentar funcionalidades conforme o desejo/necessidades do programador.

Vantagens/desvantagens em relação ao *Android Studio*:

- Difícil instalação, devido ter que se instalar vários componentes destintos como por exemplo o *Android SDK* (kit de desenvolvimento para aplicações *Android*) e o próprio *eclipse + o plugin ADT*.
- Documentação Extensa – Devido a sua maturidade existe muita documentação e de fácil acesso.
- Tal como o *Android Studio* é livre, não tendo qualquer licença para se usar.
- Uma comunidade Gigante – Devido a sua versatilidade a nível das linguagens de programação suportadas. (Queirós, 2013) (Queirós, 2014)

1.1.5.3 – App Inventor 2

App Inventor é uma plataforma de programação baseada em blocos que permite que todos, mesmo os mais novos da programação, consigam iniciar a programação e criar aplicações totalmente funcionais para dispositivos *Android*. Os recém-chegados ao *App Inventor* podem programar a sua primeira aplicação em menos de uma hora, e sendo a sua principal vantagem, programar aplicações mais complexas em muito menos tempo do que com linguagens mais tradicionais, baseadas em puro código. Inicialmente foi desenvolvido pelo professor Hal Abelson e uma equipa do *Google Education*, o *App Inventor* funciona como um serviço *Web* administrado pela equipa do *MIT's Center For Mobile Learning*. Mesmo sendo uma plataforma recente conta já com uma comunidade de cerca de 2 milhões de utilizadores, representados por 195 países a nível mundial, estando na ordem 85 mil utilizadores *online* por semana, tendo desenvolvido 4.7 milhões de aplicações até ao momento. Principais características da aplicação:

- Construção da aplicação em blocos como se fosse um puzzle.
- Não necessita de instalação – Corre através do *Browser* de internet.
- Não precisa de um dispositivo *Android* ou um emulador.
- Possibilidade de guardar/gerir os projetos *online* - através da plataforma online. (David Wolber, 2014) (Gestwicki, October 2011)

1.1.6 – Google Play

Google Play é a “loja” da Google, a onde se encontram as aplicações como jogos, filmes, musicas, livros e aplicações diversas. Publicadas para aquisição por terceiros, tendo já uma vasta gama (mais de um milhão) de aplicações tanto gratuitas como pagas, até ao momento já foram feitos mais de 50 biliões de *downloads* de aplicações. (Queirós, 2014) (Viennot, June 2014)

1.2 – Abordagem ao problema

A forma como poderia ter sido abordado o problema para chegar a uma solução podia ter sido de várias formas, como por exemplo uma pagina *web*, uma revista, uma notícia distribuída de várias formas (televisão, jornal, rádio), a utilização de outros equipamentos que se utilizam atualmente. Mas nos dias de hoje e devido a massificação de dispositivos móveis, nomeadamente *smartphones*, com a capacidade de interação através das aplicações que os sistemas atuais permitem, sensores e tecnologias avançadas de contacto, localização ou partilha de informação foi escolhido este meio, mesmo em certa parte a inovação, da relação Vendedor/Comerciante-Cliente, proposta pelo trabalho.

1.2.1 – Aplicação

Uma aplicação é um conjunto de código desenvolvido com um propósito/função, que dispõem de uma interface gráfica para que o utilizador consiga a interação com a mesma.

1.2.2 – Tecnologias

Neste momento e depois de a sua evolução o sistema operativo *Android* permite utilizar várias tecnologias/sensores para os diferentes fins, neste capítulo vai se descrever algumas das que vão ser possível utilizadas para o desenvolvimento da aplicação. As tecnologias são as seguintes (Queirós, 2014):

1.2.2.1 – QRCode (Quick Response Code)

QRCode é um código de barras bidimensional que é lido através da câmara de filmar do dispositivo móvel e depois é convertido em texto (interativo), um endereço *URI*, um numero de telefone, uma localização georreferenciada, um *e-mail*, um contato ou uma SMS. (Tong, 2014) (Queirós, 2014)

1.2.2.2 – NFC (Near Field Communication)

NFC é um conjunto de tecnologias sem fio, de curto alcance, exigindo normalmente uma distância muito curta (centímetros) para iniciar uma comunicação entre dois dispositivos permitindo assim a partilha de pequenas porções de dados entre os mesmos. O seu uso é variado desde controlar o acesso a locais, fazer pagamentos, lançar sites ou aplicações, ou usar como bilhetes virtuais. (Kelkka, 2009) (Queirós, 2014)

1.2.2.3 – Google Places

Google Places é um serviço que permite obter informações sobre localizações geográficas ou pontos de interesse conhecidos via *HTTP*. Os pedidos especificam locais representados por coordenadas de latitude/longitude, estando disponíveis quatro pedidos básicos que são: Pesquisas de local; Pedidos de detalhes do lugar; Ações no local e preenchimento automático do *Google Places*. (Queirós, 2014) (Van Canneyt, 2012)

1.2.3.4 – Google Cloud Messaging (GCM)

GCM é um serviço que permite no modelo cliente-servidor, através de 2 técnicas básicas de notificação, que permitem as aplicações cliente manterem-se atualizadas com dados vindos de um servidor. As técnicas são o *pulling* e *pushing*, na primeira técnica consiste em o dispositivo cliente consultar regularmente o servidor a fim de obter as informações atualizadas do servidor, na segunda técnica o servidor é que toma a iniciativa e notifica o dispositivo cliente assim que tiver novos dados para atualizar. Neste momento para aplicações registadas este serviço é gratuito. (Queirós, 2014) (Li, 2014)

1.2.3.5 – SQLite

SQLite é o sistema de armazenamento de dados em base de dados relacionais do *Android*, é uma pequena biblioteca *open source* escrita em *C*, que suporta as características típicas de um *SGBD* (Sistema de Gestão de Base de Dados), tais como o suporte para a linguagem de consulta *SQL* e transações. (Lin, 2009) (Queirós, 2014)

1.2.3.6 – Facebook

O *Facebook* é uma rede social a onde as pessoas podem partilhar informações sobre si, ou o seu estado, de diferentes maneiras ou seja, pode ser através de texto, fotos ou

musicas/vídeos com os seus amigos. Isto poderá ser utilizado para a divulgação ou publicidade ao estabelecimento comercial.

1.3 – Aplicações no Mercado

Neste momento e tendo em conta um estudo prévio, no mercado já existem aplicações para o qual esta aplicação está vocacionada, mas nenhuma das abordadas consegue agrupar o conjunto de ideias numa só aplicação, nem abrange o principal intuito desta nova aplicação que é a interação entre a relação Vendedor/Comerciante-Cliente do qual é suposto enfatizar com o trabalho proposto, como por exemplo reservar uma mesa num restaurante e solicitar um certo tipo de prato que está na ementa do dia, que se encontra disponível na aplicação, fazendo com que o cliente chegue ao restaurante com o pedido efetuado do que vai querer, otimizando o seu tempo, e depois através da mesma aplicação interagir com a rede social *Facebook* para partilha de informações com os seus amigos e possibilitando a tal referência de publicidade, com o objetivo de fornecer algumas opções *premium* tais como o possível pagamento da conta através do dispositivo móvel, ou consulta do que foi consumido em tempo real.

Algumas das aplicações que foram objeto de estudo, e algumas ainda em estado de desenvolvimento foram as seguintes:

- *EatOut*.
- *Appetite*.
- *AirMenu*.
- *BestTables*.
- *Zomato*.
- *MB Way*.
- *MB Phone*.
- *Meo Wallet*.
- *MobiPag*.

Contextualização

No mundo das aplicações para sistemas operativos móveis, podemos afirmar que temos três “gigantes” a combater entre si, Android (Google), IOS (Apple) e Windows Phone (Microsoft), quer pela inovação propriamente dita, que por resultados. Tendo isto, será importante fazer a análise a onde este projeto se encontra inserido e também alguns factos de relevância que fazem prevalecer a conceção do projeto.

“Não se possui o que não se comprehende.”

Goethe, Johann

Um dos “gigantes” é a *Google*, com o seu sistema operativo Android desenvolvido sobretudo para dispositivos móveis começou com a Android, Inc., empresa fundada em Palo Alto, Califórnia em outubro de 2003 por Andy Rubin (cofundador da Danger); Rich Miner (co-fundador da Wildfire Communications, Inc.); Nick Sears (ex-vice-presidente da T-Mobile) e Chris White (Responsável pelo projeto de desenvolvimento de design e interface da WebTV) para desenvolver, nas palavras de Rubin, "dispositivos móveis mais inteligentes que estejam mais cientes das preferências e da localização do seu dono". Em julho de 2005 a Google adquiriu a Android, Inc., mais tarde e depois de um conjunto de acontecimentos, surgiu o primeiro telemóvel (T-Mobile's G1 HTC Dream) com o sistema operativo Android (versão 1.0). (Queirós, 2013)

A partir deste acontecimento o sistema operativo tem evoluído, através de várias versões, sempre com uma nomenclatura original, fazendo referência a doces. Todas as versões vieram trazer melhorias, quer a nível de segurança, quer a nível de novas funcionalidades e opções. O Android por ser livre de código aberto, mesmo sendo relativamente novo, conta com uma comunidade de programadores bastante grande, e daí se deve ao facto de existir tantas aplicações para este sistema operativo. Com os dados recolhidos em julho de 2015 a Google Play (Loja de aplicações da Google) detém cerca de 1.6 milhões de aplicações para *download*, sendo a loja que mais aplicações têm neste momento, como demonstra a figura em baixo com o gráfico referente as aplicações disponíveis por loja, através do site de estatísticas. www.statista.com.

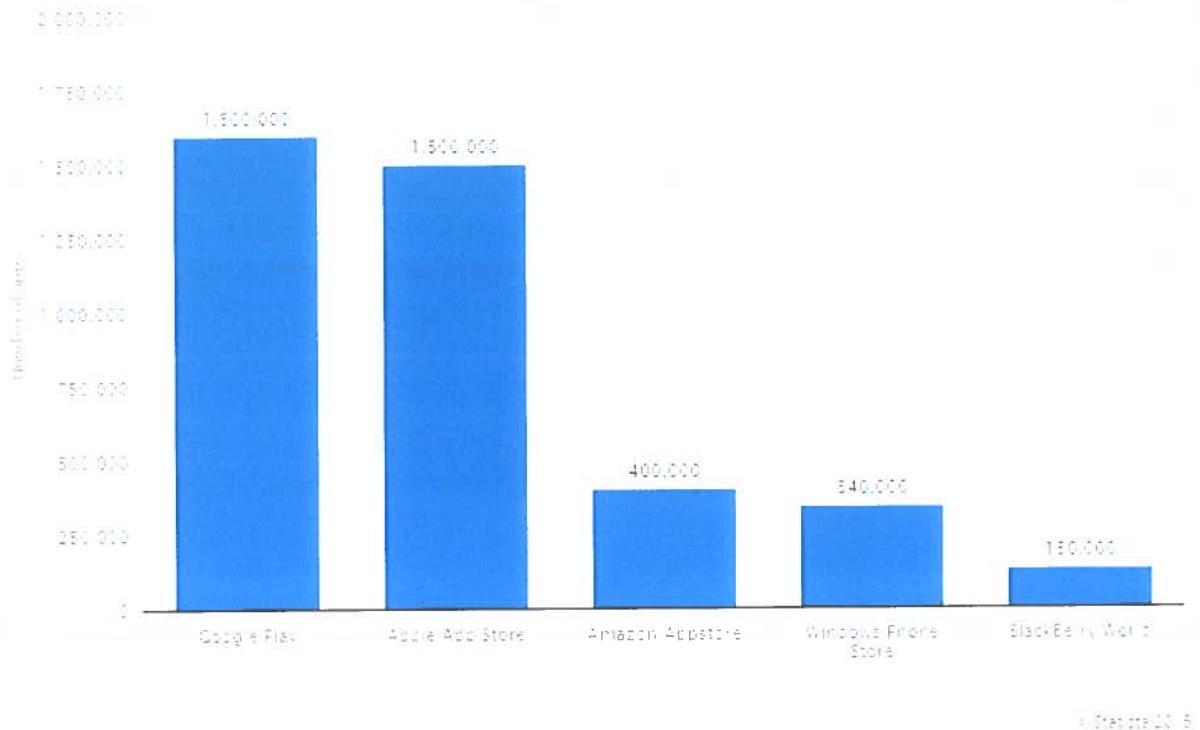


Figura 1 - Estatística de aplicações disponíveis por loja.

Com este facto apresentado chega-se a outra conclusão, que passa pela utilização do próprio sistema operativo, que mesmo sendo mais recente que os outros, não só conseguiu acompanhar os outros sistemas operativos, como os ultrapassou em utilização. Um dos grande fatores para este acontecimento prende-se com o facto de o SO Android, ser utilizado por vários fabricantes de equipamentos móveis, ao invés do IOS por exemplo, que apenas está disponível nos equipamentos da Apple. Estes fatores contribuíram para a escolha e desenvolvimento do presente trabalho, com a construção de uma aplicação para este sistema operativo, e assim, conseguir não só abranger um maior números de dispositivos, mas como o maior número de utilizadores. Em todas as provisões de utilização ou de vendas de equipamentos para este ano, ou para os seguintes, o SO Android, encontra-se e segundo as previsões na linha da frente, e em muitos casos, destacado. Em baixo podemos verificar um gráfico que demonstra o numero de vendas por sistema operativo de dispositivos móveis, através da *International Data Corporation (IDC)*¹, que as vendas no primeiro quarto de ano de dispositivos Android representam cerca de 78% em 334.4 milhões de equipamentos.

¹ Visitar para mais informação - <http://www.idc.com/>.

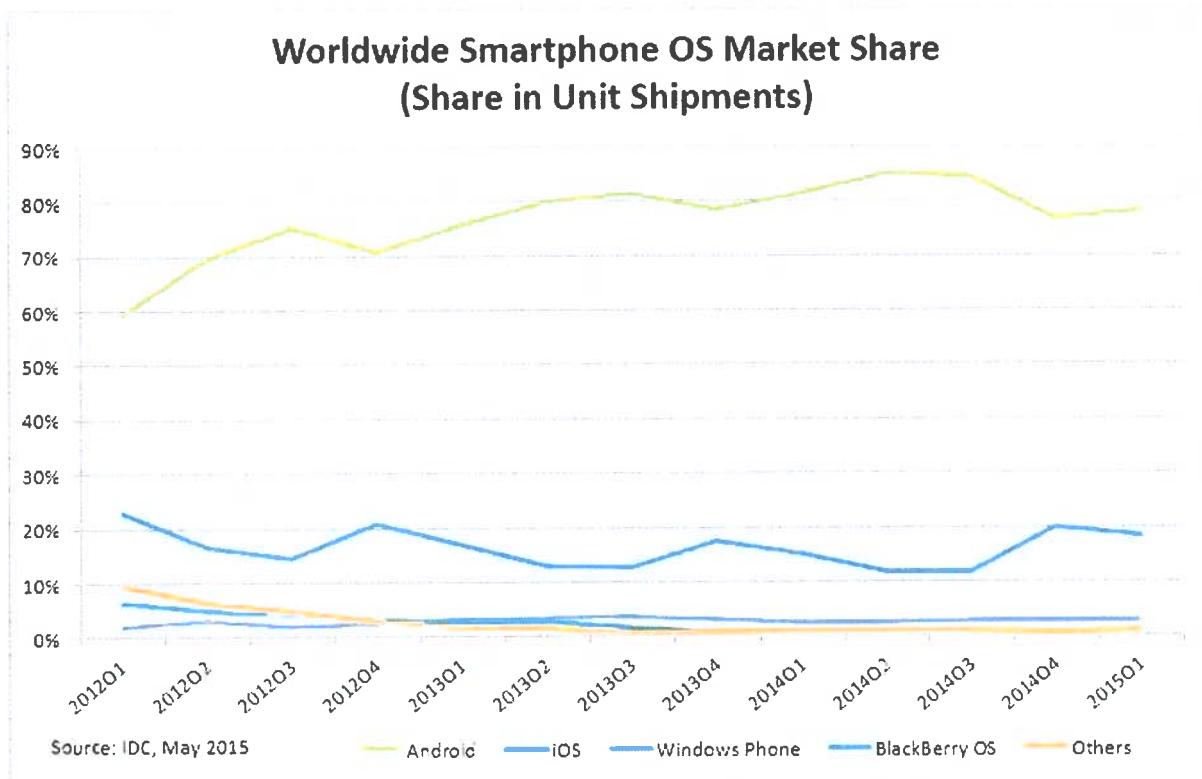


Figura 2 - Estatística de equipamentos vendidos por Sistema Operativo.

Tendo em conta todos estes fatores o presente trabalho visa de ir ao encontro destes mesmos dados, ou seja, efetuar uma aplicação para o sistema operativo Android. Esta aplicação, através das novas *API's* da Google, e de novos conceitos tais como *Cloud Computing*, irá trazer funcionalidades para a execução de pedidos de produtos a um determinado estabelecimento de restauração, a partir de um dispositivo móvel.

Desenvolvimento do Projeto

"A parte que ignoramos é muito maior que tudo quanto sabemos."

Platão

Aplicação Android

Introdução

Neste capítulo vamos ver como a aplicação Android foi realizada, como funciona, as tecnologias responsáveis pelos comportamentos obtidos, através de figuras ilustrativas, assim como a sua explicação pormenorizada da sua utilização, e as principais funções que o utilizador pode utilizar. Vai também ser demonstrado todo o código de programação referente as atividades da aplicação (ecrãs), e a parte gráfica da aplicação. Todo o código estará comentado para uma melhor percepção do que cada função faz e quando atua. Os comentários ao código começam sempre por “//”, e encontram-se escritos a verde.

Primeiro Ecrã

Depois de instalada a aplicação num dispositivo com o sistema operativo Android, irá aparecer um ícone com o símbolo do Bon Appétit em fundo azul, é o ícone da aplicação para aceder a ela (aplicação), basta clicar nele. O ícone é igual ao referenciado na figura em baixo no quadrado amarelo:



Figura 3 - Ecrã de um smartphone com a aplicação Bon Appétit instalada.

Depois de acedermos á aplicação vamos encontrar o ecrã principal da aplicação. Este ecrã inicial é o ponto de partida da aplicação “Bon Appétit”, onde vamos encontrar as funcionalidades principais, que são quatro: Iniciar Pedido; Verificar Conta Corrente; Reservar Mesa e Perto de Si. De seguida ira ser explicado ao pormenor estas quatro funcionalidades, em baixo vamos poder ver como é o aspeto do ecrã inicial, como demonstra a seguinte figura.



Figura 4 - Ecrã principal da aplicação

Neste ecrã vamos ter acesso as quatro principais funções/opções da aplicação que são as seguintes:

1. Iniciar Pedido
2. Verificar Conta Corrente
3. Reservar Mesa
4. Perto de Si

Para obter este aspeto gráfico do primeiro ecrã foi necessário o seguinte código:

```
// Interface gráfico da atividade principal.  
<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:id="@+id/container"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:background="@drawable/fundo1blue"  
    android:orientation="vertical"  
    tools:context="com.javacodegeeks.androidstartactivityforresultexample.ActivityOne">  
    <Button  
        android:id="@+id/scanner"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:layout_alignParentLeft="true"  
        android:layout_alignParentTop="true"  
        android:layout_margin="10dp"  
        android:alpha="0.6"  
        android:background="#CFD8DC"  
        android:gravity="center"  
        android:onClick="scanQR"  
        android:text="Iniciar Pedido"  
        android:textColor="#FFFFFF"  
        android:textSize="18sp" >  
    </Button>  
    <Button  
        android:id="@+id/scanner2"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:layout_below="@+id/scanner"  
        android:layout_centerHorizontal="true"  
        android:layout_gravity="center"  
        android:layout_margin="10dp"  
        android:alpha="0.6"  
        android:background="#CFD8DC"  
        android:gravity="center"  
        android:text="Verificar Conta Corrente"  
        android:textColor="#FFFFFF"  
        android:textSize="18sp" >  
    </Button>  
    <Button  
        android:id="@+id/botaoReservar"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:layout_below="@+id/scanner2"  
        android:layout_centerHorizontal="true"
```

```

        android:layout_gravity="center"
        android:layout_margin="10dp"
        android:alpha="0.6"
        android:background="#CFD8DC"
        android:gravity="center"
        android:text="Reservar Mesa"
        android:textColor="#FFFFFF"
        android:textSize="18sp" >
    </Button>
    <Button
        android:id="@+id/botaoPertoDeSi"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/botaoReservar"
        android:layout_margin="10dp"
        android:alpha="0.6"
        android:background="#CFD8DC"
        android:gravity="center"
        android:text="Perto de si"
        android:textColor="#FFFFFF"
        android:textSize="18sp" >
    </Button>
</RelativeLayout>

```

Para obter os comportamentos dos botões do primeiro ecrã, temos o código que corresponde a atividade principal do programa que é o seguinte:

```

//Código Referente a atividade principal da aplicação Android.
package com.bonappetitapp;
// Imports necessários as classes utilizadas
import android.app.Activity;
import android.app.AlertDialog;
import android.content.ActivityNotFoundException;
import android.content.DialogInterface;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
//Classe principal da aplicação.
public class BonAppetitActivity extends Activity
{
    //Declaração de variáveis globais.
    private static final String ACTION_SCAN = "com.google.zxing.client.android.SCAN";
    private static final String SECURE_CODE = "02068647";
    public Button btnPertoDeSi,btnReservas,btnCcurrent;
    //Classe onCreate que só corre uma vez no início da atividade.
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //Referencia aos objetos da parte gráfica.
        theTextView = (TextView) findViewById(R.id.textView1);
        theTextView2 = (TextView) findViewById(R.id.textView2);
        theTextView3 = (TextView) findViewById(R.id.textView3);
        btnPertoDeSi = (Button) findViewById(R.id.botaoPertoDeSi);

```

```

btnReservas = (Button) findViewById(R.id.botaoReservar);
btnCcurrent = (Button) findViewById(R.id.scanner2);
//Método Responsável pelo click do botão Perto de Si.
btnPertoDeSi.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //Chama a LocaisActivity
        Intent it = new Intent(BonAppetitActivity.this, LocaisActivity.class);
        startActivity(it);
    }
}); //Fim do btnPertoDeSi
//Método Responsável pelo click do botão Reservar Mesa.
btnReservas.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //Chama a ReservasActivity
        Intent it = new Intent(BonAppetitActivity.this, ReservasActivity.class);
        startActivity(it);
    }
}); //Fim do btnReservas
//Método Responsável pelo click do botão Conta Corrent.
btnCcurrent.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //Chama a ContaCorrentActivity
        Intent itt = new Intent(BonAppetitActivity.this, ContaCorrentActivity.class);
        startActivity(itt);
    }
}); //Fim do btnReservas
//Fim do onCreate
//Classe responsável pela Leitura do QRCode.
public void scanQR(View v)
{
    try
    {
        Intent intent = new Intent(ACTION_SCAN);
        intent.putExtra(SCAN_MODE, "QR_CODE_MODE");
        startActivityForResult(intent, 0);
    }
    catch (ActivityNotFoundException anfe)
    {
        showDialog(BonAppetitActivity.this, "No Scanner Found", "Download a scanner code activity?", "Yes", "No").show();
    }
}
//Método Responsável pela leitura do QRCode.
private static AlertDialog showDialog(final Activity act, CharSequence title, CharSequence message,
CharSequence buttonYes, CharSequence buttonNo)
{
    AlertDialog.Builder downloadDialog = new AlertDialog.Builder(act);
    downloadDialog.setTitle(title);
    downloadDialog.setMessage(message);
    downloadDialog.setPositiveButton(buttonYes, new DialogInterface.OnClickListener()
    {
        public void onClick(DialogInterface dialogInterface, int i) {
            Uri uri = Uri.parse("market://search?q=pname:" +
                "com.google.zxing.client.android");
            Intent intent = new Intent(Intent.ACTION_VIEW, uri);
            try {
                act.startActivity(intent);
            } catch (ActivityNotFoundException anfe) {

```

```

        }
    });
    downloadDialog.setNegativeButton(buttonNo, new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialogInterface, int i) {
        }
    });
    return downloadDialog.show();
}

//Classe que recebe o valor do QRCode depois de lido, e abre a nova atividade MenuActivity, passado os dados por Intent.
public void onActivityResult(int requestCode, int resultCode, Intent intent)
{
    if (requestCode == 0)
    {
        if (resultCode == RESULT_OK)
        {
            String contents = intent.getStringExtra("SCAN_RESULT");
            String format = intent.getStringExtra("SCAN_RESULT_FORMAT");
            if(contents.contains(SECURE_CODE))
            {
                //Separar a String Lida, para verificar o código de segurança.
                String[] parts = contents.split("-");
                String secureCode = parts[0];
                String restaurantId = parts[1]; // Valor do restaurante
                String numeroMesa = parts[2]; // Valor da mesa
                //Chamar a atividade MenuActivity.
                if( secureCode.equals(SECURE_CODE)){
                    Intent i = new Intent (this,MenuActivity.class);
                    i.putExtra("QRcode", contents);
                    startActivity(i);
                }
            }
            else{
                Toast toast = Toast.makeText(this, "QRCode Invalido" ,
                Toast.LENGTH_LONG);
                toast.show();
            }
        }
    }
}
//Fim da classe Principal.

```

Depois de demonstrado o primeiro ecrã com as principais funcionalidades, através do código da interface gráfica e do código da atividade, é importante mencionar dois ficheiros que fazem parte da estrutura de uma aplicação Android, que são o ficheiro de manifesto e o ficheiro de String's.

O Ficheiro `AndroidManifest.xml` é um ficheiro no formato XML, criado automaticamente na pasta raiz, e que contém informação essencial para a execução da aplicação: descreve os componentes da aplicação, define os nomes para as atividades, enumera os modos de orientação do ecrã (vertical, horizontal ou ambos), declara permissões para acesso a recursos como o *global positioning system* (GPS), ou a internet, por exemplo, lista de

bibliotecas que a aplicação vai usar e qual atividade que iniciará primeiro quando a aplicação for aberta, etc. (Queirós, 2013)

O código referente ao ficheiro de manifesto da aplicação é o seguinte.

```
// Código referente ao ficheiro de Manisfesto da aplicação Android
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bonappetitapp"
    android:versionCode="1"
    android:versionName="1.0" >
    //Permissões de utilização de recursos
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <permission
        android:name="com.bonappetitapp.permission.MAPS_RECEIVE"
        android:protectionLevel="signature" />
    <uses-permission android:name="com.bonappetitapp.permission.MAPS_RECEIVE" />
    <uses-feature
        android:glEsVersion="0x00020000"
        android:required="true" />
    //SDK requerido para utilização da aplicação Android. e SDK targuet.
    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="19" />
    // Dados sobre a aplicação Android.
    <application
        android:allowBackup="true"
        android:icon="@drawable/logoapp2"
        android:label="Bon Appétit" >
        // Todas as atividades que fazem parte da aplicação Android.
        <activity
            android:name=".bonappetitapp"
            android:icon="@drawable/logoapp2"
            android:label="Bon Appétit"
            android:screenOrientation="portrait" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".MainActivity"
            android:label="Menu"
            android:screenOrientation="portrait" >
        </activity>
        <activity
            android:name=".ProdutosActivity"
            android:label="Produtos"
            android:screenOrientation="portrait" >
        </activity>
        <activity
            android:name=".ProdutoActivity"
```

```

        android:label="Produto"
        android:screenOrientation="portrait">
    </activity>
<activity
    android:name=".LocaisActivity"
    android:label="Perto De Si"
    android:screenOrientation="portrait">
    </activity>
// Metadata necessária para utilização do Google Maps para a atividade Perto de Si.
<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="AIzaSyCqCMVV8zMYWc8l-oskI-5XWFoDtEhejoc" />
//Metadata necessária para utilização de partilha de informação através do SDK do Facebook.
<meta-data
    android:name="com.facebook.sdk.ApplicationId"
    android:value="@string/facebook_app_id" />
<activity
    android:name="com.facebook.FacebookActivity"
    android:configChanges="keyboard|keyboardHidden|screenLayout|screenSize|orientation"
    android:label="@string/app_name"
    android:theme="@android:style/Theme.Translucent.NoTitleBar" />
<provider
    android:name="com.facebook.FacebookContentProvider"
    android:authorities="com.facebook.app.FacebookContentProvider429349730602080"
    android:exported="true" />
//Restantes atividades da aplicação Android.
<activity
    android:name=".PedidoActivity"
    android:label="Pedido"
    android:screenOrientation="portrait"
    android:windowSoftInputMode="adjustPan">
    </activity>
<activity
    android:name=".ReservasActivity"
    android:label="Reservar Mesa"
    android:screenOrientation="portrait">
    </activity>
<activity
    android:name=".EmentaMenuActivity"
    android:label="Ementa - Menu"
    android:screenOrientation="portrait">
    </activity>
<activity
    android:name=".ReservarMesaActivity"
    android:label="Reservar Mesa - E-mail"
    android:screenOrientation="portrait">
    </activity>
<activity
    android:name=".ContaCorrenteActivity"
    android:label="Conta Corrente"
    android:screenOrientation="portrait">
    </activity>
</application>
</manifest>

```

O outro ficheiro mencionado é o strings.xml, é também um ficheiro em XML, em que a sua principal função é guardar todos os valores de String's. O ficheiro da aplicação contém o seguinte código.

```
//Código referente ao ficheiro Strings.xml da aplicação Android.  
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <string name="app_name">Bom Appétit</string>  
    <string name="action_settings">Settings</string>  
    <string name="str_btn_find">Procurar</string>  
    <string name="facebook_app_id">429349730602080</string>  
    <!-- Strings utilizadas para a atividade Perto de si -->  
    <string-array name="place_type">  
        <item>airport</item>  
        <item>atm</item>  
        <item>bank</item>  
        <item>bar</item>  
        <item>bus_station</item>  
        <item>hospital</item>  
        <item>church</item>  
        <item>police</item>  
        <item>restaurant</item>  
    </string-array>  
    <string-array name="place_type_name">  
        <item>Aeroporto</item>  
        <item>ATM</item>  
        <item>Banco</item>  
        <item>Bar</item>  
        <item>Bus</item>  
        <item>Hospital</item>  
        <item>Igreja</item>  
        <item>Policia</item>  
        <item>Restaurante</item>  
    </string-array>  
    <!-- Strings utilizadas para a atividade Perto de si -->  
    <string name="title_activity_menu">MenuActivity</string>  
    <string name="title_activity_single_list_item">SingleListItem</string>  
    <string name="title_activity_produtos">ProdutosActivity</string>  
    <string name="title_activity_produto">ProdutoActivity</string>  
    <string name="title_activity_locais">LocaisActivity</string>  
    <string name="title_activity_pedido">PedidoActivity</string>  
    <string name="title_activity_reservas">ReservasActivity</string>  
    <string name="title_activity_menu_teste">MenuTesteActivity</string>  
    <string name="title_activity_ementa_menu">EmentaMenuActivity</string>  
    <string name="title_activity_reservar_mesa">ReservarMesaActivity</string>  
    <string name="title_activity_conta_corrent">ContaCorrentActivity</string>  
</resources>
```

Iniciar Pedido

Esta opção é a funcionalidade principal da aplicação, é através dela que podemos efetuar o nosso pedido com os produtos que desejamos, que se encontram no menu, do respetivo restaurante. O processo de pedido começa pela leitura de um QRcode, de seguida é nos mostrado o menu do restaurante com as categorias possíveis, por consequência temos os

produtos referentes a cada uma dessas categorias, ou seja, se selecionamos a categoria Carne no menu, a seguir, irá mostrar todos os pratos de carne que esse restaurante têm. Depois de selecionarmos o prato em específico, vamos ter o ecrã que nos vai mostrar informação acerca do produto selecionado, assim como uma descrição de como é composto, o preço, e uma foto ilustrativa no topo. Neste mesmo ecrã, vamos ter a opção de introduzir a quantidade que queremos desse produto, assim como, adicioná-lo ao nosso pedido. Ao carregar na opção finalizar pedido é então visualizado outro ecrã, a onde, poderá ver todos os produtos adicionados ao seu pedido, assim como, atualizar a quantidade dos produtos selecionados, ou mesmo, eliminá-los do seu pedido. Por ultimo temos o botão de concluir pedido, que efetua o pedido final voltando depois ao ecrã principal da aplicação.

De seguida vai ser demonstrado através de figuras ilustrativas, todo o processo, de efetuar um pedido assim como, o código responsável pelos comportamentos para alcançar esse tipo de função. Depois de selecionar a opção de Iniciar Pedido, a aplicação desperta a camara fotográfica para a leitura do QRCode, como demonstra as figuras em baixo.



Figura 5 - A ler o QRCode.



Figura 6 - QRCode encontrado e lido.

A utilização do QRCode

A utilização desta tecnologia foi pensada em minimizar o esforço do utilizador, existia a preocupação de ter que forçar o utilizador a inserir os dados em relação a mesa e o restaurante a onde se encontrava a fazer os pedidos, e assim, a utilização da aplicação não era tão apelativa, mas ainda assim era necessário esses dados. Então, depois de uma pesquisa foi verificado a possível interação através da camara fotográfica do *smartphone*² para a leitura do QRCode e assim evitar a introdução dos dados necessários para os pedidos. Tendo em conta esta opção, em cenário real, iríamos ter um QRCode por mesa, em cada restaurante (estabelecimento de restauração). Com este processo podemos afirmar que é o primeiro passo, para efetuar um pedido com a aplicação Bom Appétit.

Em baixo temos um dos possíveis QRCode's³, como mostra a figura em baixo.

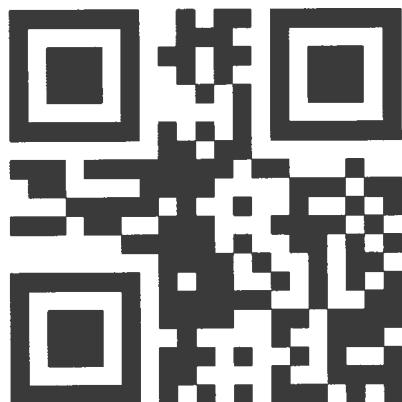


Figura 7 - Exemplo de um QRCode utilizado

Neste exemplo, a informação contida no QRCode é composta por três partes distintas. A primeira parte consiste num código de segurança, composto por 8 dígitos, para poder ultrapassar a questão da leitura de outros QRCode's que não fossem, os que estão na mesa do restaurante e assim provocar erros na aplicação. Na segunda e terceira parte corresponde ao dígito que identifica o restaurante e a mesa, respetivamente. E assim teríamos a seguinte mensagem neste QRCode: 02068647-1-2.

² Palavra para designar um telemóvel inteligente.

³ Um dos possíveis sites para gerar QRCode's - <https://www.the-qrcode-generator.com>

A utilização deste processo de obtenção de dados está a ser feito através de duas funções na atividade (*activity*) principal da aplicação, que são invocadas através do método `onClick`⁴ do botão “Iniciar Pedido”, do ecrã principal. Neste processo é utilizado uma biblioteca pública de seu nome ZXing (Zebra Crossing)⁵. O código das duas funções encontra-se no código da atividade principal anteriormente mostrado.

Depois da leitura do QRCode, vamos então avançar para a leitura do Menu do restaurante, informação essa que se encontra numa plataforma *Cloud*⁶. No capítulo seguinte, “Aplicação Servidor” irá ser demonstrado e explicado a sua configuração, para já vamos nos restringir no lado da aplicação. Para obter o seguinte resultado, vamos ter que demonstrar três pontos fulcrais, que são eles: como está a ser efetuado a comunicação com o servidor, em que formato os dados estão a ser tratados, e por último como são apresentados no ecrã ao utilizador. Uma imagem ilustrativa possível de como é apresentado o Menu do restaurante, encontra-se em baixo.

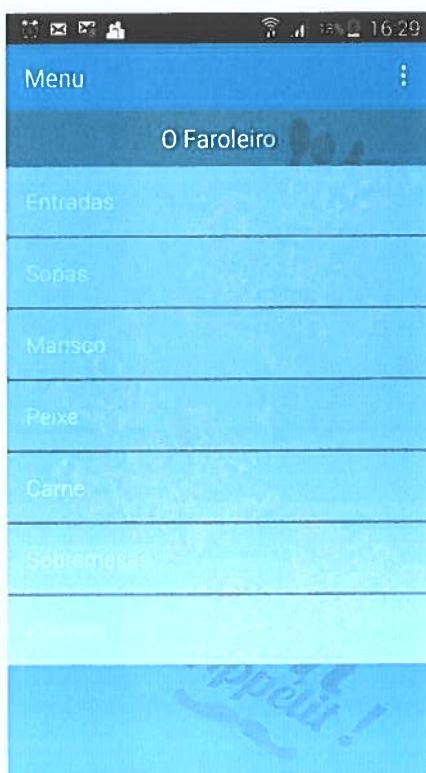


Figura 8 - Menu do Restaurante *O Faroleiro*.

⁴ Método responsável pelo comportamento, quando se “aciona” o objeto em questão.

⁵ Repositório do código para download - <https://github.com/zxing/zxing>

⁶ Definição de Cloud Computing através de NIST – <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>.

Agora vamos descrever o processo de leitura dos dados ao servidor, para isso recorremos a classe RestAPI em java, que trata dos pedidos HttpClient⁷, e assim, devolvendo depois a informação sobre o formato JSON⁸, tendo em conta a programação do servidor na resposta aos pedidos por HTTP. Em baixo o código referente a classe RestAPI.

```
//Classe RestAPI.java, é classe que lida pelos pedidos http feitos ao servidor no formato JSON. É através desta classe que recebemos os dados, e é feito a iteração pela informação dada pelo servidor.
package com.bonappetitapp;
// Imports necessários as classes utilizadas.
import android.os.AsyncTask;
import android.util.Log;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
public class RestAPI extends AsyncTask<String, Void, String>
{
    protected String getResult(String url) throws Exception
    {
        execute(url);
        return get();
    }
    //Método que retorna o URL em background.
    @Override
    protected String doInBackground(String... urls)
    {
        return GET(urls[0]);
    }
    @Override
    protected void onPostExecute(String result)
    {
    }
    //Metodo que devolve o resultado do pedido como String.
    public static String GET(String url)
    {
        InputStream inputStream = null;
        String result = "";
        try
        {
            //Criação do http Client.
            HttpClient httpClient = new DefaultHttpClient();
            // Faz a solicitação GET para a URL dada.
            HttpResponse httpResponse = httpClient.execute(new HttpGet(url));
            // Recebe a resposta como inputStream.
            inputStream = httpResponse.getEntity().getContent();
            // Converte o inputStream para uma string.
            if(inputStream != null)
                result = convertInputStreamToString(inputStream);
            else
        }
```

⁷ Informação adicional sobre o funcionamento do método HttpClient - <http://developer.android.com/reference/org/apache/http/client/HttpClient.html>.

⁸ O que é JSON e como utiliza-lo - <http://developer.android.com/reference/org/json/JSONObject.html>.

```

        result = "Não funcionou!";
    }
    catch (Exception e)
    {
        Log.d("InputStream", e.getLocalizedMessage());
    }
    //Retorna o valor como String.
    return result;
}
//Metodo responsável por guardar e iterar a informação dada pelo pedido de http.
private static String convertInputStreamToString(InputStream inputStream) throws IOException
{
    BufferedReader bufferedReader = new BufferedReader( new InputStreamReader(inputStream));
    String line = "";
    String result = "";
    // Processo de Leitura da informação contida no URL enviado.
    while((line = bufferedReader.readLine()) != null)
        result += line;
    inputStream.close();
    return result;
}
// Fim da classe RestAPI.

```

O formato JSON

O JSON é um acrónimo para *JavaScript Object Notation*, e trata-se de um formato universal para a troca de informação de dados. É baseado num subconjunto da linguagem de JavaScript, sendo que o formato é totalmente em texto e independente da linguagem, sendo assim ideal para a troca de informação com serviços Web. Em baixo temos um exemplo de um pedido HTTP, retornando a mensagem no formato JSON. (Queirós, 2014)

Quadro 1 - Mensagem no formato JSON.

```
{
produtos: [
{
idP: 3,
foto: "https://bon-appetit-server.herokuapp.com/Produtos/preco.jpg",
nome: "Prego",
descricao: "O Prego é um prato típico da culinária de Portugal e pode ser comido no pão ou no prato. Normalmente é temperado com mostarda ou molho picante.",
preco: 12
},
{
idP: 2,
foto: "https://bon-appetit-server.herokuapp.com/Produtos/bitoque.jpg",
nome: "Bitoque",
descricao: "CULINÁRIA prato típico português composto por um bife grelhado ou frito, ovo estrelado por cima. geralmente acompanhado de batatas fritas, arroz e, por vezes, salada",
preco: 34
}
]
```

Depois de ter sido verificado a classe responsável pela comunicação com o servidor para obter os dados dos pedidos http, é agora mostrado o código da atividade Menu, que é o seguinte.

```
//Código Referente a atividade Menu da aplicação Android.
package com.bonappetitapp;
// Imports necessários as classes utilizadas
import java.util.ArrayList;
import org.json.JSONArray;
import org.json.JSONObject;
import android.app.ActionBar;
import android.app.Activity;
import android.app.ListActivity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
//Classe Principal da atividade Menu.
public class MenuActivity extends ListActivity {
    //Declaração de variáveis globais.
    private TextView theTextView;
    private ListView lista;
    public String nomeRes = "";
    public String[] idsMenus = new String[40];
    //Classe onCreate que só corre uma vez no inicio da atividade.
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_menu);
        //Referencia aos objetos da parte gráfica.
        theTextView = (TextView) findViewById(R.id.textView1);
        //Receber os dados enviados pela a atividade, que chamou este ecrã.
        Intent intent = getIntent();
        String contents = intent.getExtras().getString("QRecode");
        //Separar a String Lida, para verificar o código de segurança.
        String[] parts = contents.split("-");
        String secureCode = parts[0];
        String restaurantId = parts[1]; // Valor do restaurante
        String numeroMesa = parts[2]; // Valor da mesa
        //Lista de um array de string que vai servir a lista de items.
        ArrayList<String> listItems=new ArrayList<String>();
        //Definir um adaptador de string que vai gerir os dados dentro da Listview.
        ArrayAdapter<String> adapter;
        //Referência para onde os dados vão ser carregados no Ambiente gráfico.
        adapter=new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1,listItems);
        setListAdapter(adapter);
        //Carregar o nome do restaurante, utilizando a classe RestAPI, no formato JSON.
        try {
            String result = new RestAPI().getResult("https://bon-appetit-server.herokuapp.com/api/restaurantes/" + restaurantId);
```

```

//Criação do objeto JSON com o resultado obtido do pedido http.
JSONObject json = new JSONObject(result);
//Colocar o nome do restaurante dentro da TextView.
theTextView.setText(json.getString("nome"));
nomeRes = json.getString("nome");
}
catch (Exception e)
{
Toast.makeText(this, "Link Inválido", Toast.LENGTH_LONG).show();
}
//Carregar o nome do menu, utilizando a classe RestAPI, no formato JSON.
try
{
String result = new RestAPI().getResult("https://bon-appetit-
server.herokuapp.com/api/restaurantes/" + restauranteId + "/menus");
//Criação do objeto JSON com o resultado obtido do pedido http.
JSONObject json = new JSONObject(result);
JSONArray menusJson = json.getJSONArray("menus");
//For responsável pela iteração dos nomes dos menus, guardando o ID no array de
idsMenus.
for(int i=0; i<menusJson.length(); i++)
{
listItems.add(menusJson.getJSONObject(i).getString("nome"));
idsMenus[i] = menusJson.getJSONObject(i).getString("idM");
}//Fim do For.
}
catch (Exception e)
{
Toast.makeText(this, "Link Inválido", Toast.LENGTH_LONG).show();
}
//Fim do onCreate
//Método responsável pelo click de um determinado menu, para poder chamar os respetivos produtos,
chamando a atividade ProdutosActivity.
@Override
protected void onListItemClick(ListView list, View v, final int position, long id) {
super.onListItemClick(list, v, position, id);
Intent intent = getIntent();
String contents = intent.getExtras().getString("QRcode");

String[] parts = contents.split("-");
String secureCode = parts[0];
String restauranteId = parts[1]; // Valor do restaurante
String numeroMesa = parts[2]; //Valor da mesa
//Chamar a nova atividade passando os dados necessários.
Intent i = new Intent (this.ProdutosActivity.class);
i.putExtra("idRestaurante", restauranteId );
i.putExtra("idMenu", idsMenus[position]);
i.putExtra("mesa", numeroMesa);
i.putExtra("nomeRes", nomeRes);
startActivity(i);
}//Fim do onListItemClick
//Fim da classe Principal da atividade.

```

O código referente a interface gráfica da atividade Menu, é o seguinte.

```

//Interface gráfico da atividade Menu.
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"

```

```

    android:layout_height="match_parent"
    android:background="@drawable/fundo2blue"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.javacodegeeks.androidqrcodeexample.MenuActivity" >
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:padding="10dp"
        android:text="TextView"
        android:textColor="#FFFFFF"
        android:textSize="20dp" />
    <ListView
        android:id="@+id/list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView1"
        android:layout_centerHorizontal="true"
        android:alpha="0.6"
        android:background="#CFD8DC"
        android:divider="#80000000"
        android:dividerHeight="2.0sp"
        android:textColor="#FFFFFF"
        android:textSize="25sp" >
    </ListView>
</RelativeLayout>

```

Neste momento, e depois de carregado o menu com as categorias possíveis, e selecionado uma delas vamos entrar nos produtos que estejam contidos nessa mesma categoria como anteriormente explicado, neste caso, todo o processo é idêntico ao anterior apenas alterando os *link's*⁹ dos pedidos HTTP. E assim, vamos obter a informação desejada dos produtos, que estão inseridos em cada categoria (Carne, Peixe, Entradas, etc.) como demonstra a figura em baixo.

⁹ Palavra inglesa para designar uma ligação entre dois elementos.

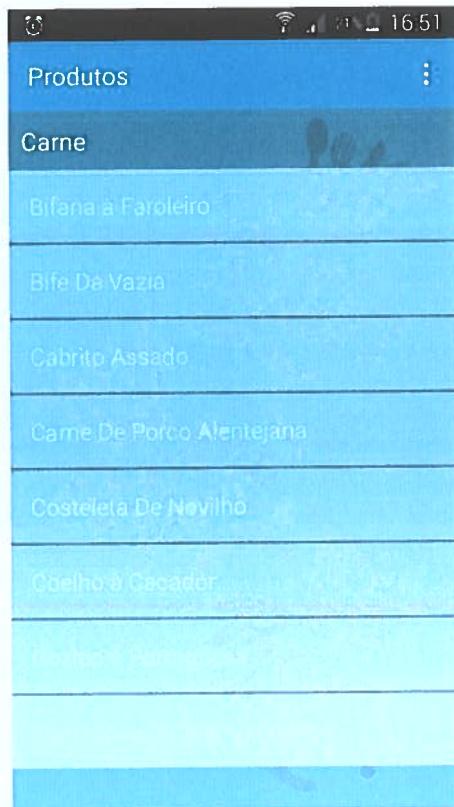


Figura 9 - Menu de produtos que se encontram dentro da categoria Carné.

O código referente a atividade Produtos é o seguinte.

```
//Código Referente a atividade Produtos da aplicação Android.  
package com.bonappetitapp;  
// Imports necessários as classes utilizadas  
import java.util.ArrayList;  
import org.json.JSONArray;  
import org.json.JSONObject;  
import android.app.ListActivity;  
import android.content.Intent;  
import android.os.Bundle;  
import android.view.Menu;  
import android.view.MenuItem;  
import android.view.View;  
import android.widget.ArrayAdapter;  
import android.widget.ListView;  
import android.widget.TextView;  
import android.widget.Toast;  
//Classe Principal da atividade Produtos.  
public class ProdutosActivity extends ListActivity {  
    //Declaração de variáveis globais.  
    private TextView theTextView;  
    public String[] idsProdutos = new String[100];  
    //Classe onCreate que só corre uma vez no inicio da atividade.  
    @Override
```

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_produtos);
    //Referencia aos objetos da parte gráfica.
    theTextView = (TextView) findViewById(R.id.textView1);
    //Receber os dados enviados pela a atividade, que chamou este ecrã.
    Intent intent = getIntent();
    String idMenu = intent.getExtras().getString("idMenu");
    String idRes = intent.getExtras().getString("idRestaurante");
    String numMesa = intent.getExtras().getString("mesa");
    //Lista de um array de string que vai servir a lista de items.
    ArrayList<String> listItems=new ArrayList<String>();
    //Definir um adaptador de string que vai gerir os dados dentro da Listview.
    ArrayAdapter<String> adapter;
    //Referência para onde os dados vão ser carregados no Ambiente gráfico.
    adapter=new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1,listItems);
    setListAdapter(adapter);
    //Carregar o nome do menu, utilizando a classe RestAPI, no formato JSON.
    try
    {
        String result = new RestAPI().getResult("https://bon-appetit-
server.herokuapp.com/api/restaurantes/" + idRes + "/menus/" + idMenu);
        //Criação do objeto JSON com o resultado obtido do pedido http.
        JSONObject json = new JSONObject(result);
        //Colocar o nome do Menu dentro da TextView.
        theTextView.setText(json.getString("nome"));
    }
    catch (Exception e)
    {
        Toast.makeText(this, "Link Inválido", Toast.LENGTH_LONG).show();
    }
    //Carregar o nome dos produtos, utilizando a classe RestAPI, no formato JSON.
    try
    {
        String result = new RestAPI().getResult("https://bon-appetit-
server.herokuapp.com/api/restaurantes/" + idRes + "/menus/" + idMenu + "/produtos");
        //Criação do objeto JSON com o resultado obtido do pedido http.
        JSONObject json = new JSONObject(result);
        JSONArray menusJson = json.getJSONArray("produtos");
        //For responsável pela iteração dos nomes dos produtos, guardando o ID no array de
        idsProdutos.
        for(int i=0; i<menusJson.length(); i++)
        {
            listItems.add(menusJson.getJSONObject(i).getString("nome"));
            idsProdutos[i] = menusJson.getJSONObject(i).getString("idP");
        }
    }
    catch (Exception e)
    {
        Toast.makeText(this, "Link Inválido", Toast.LENGTH_LONG).show();
    }
}
//Fim do onCreate
//Método responsável pelo click de um determinado produto, para poder chamar os respetivo produto
na próxima atividade, chamando a atividade ProdutoActivity.
@Override
protected void onListItemClick(ListView list, View v, final int position, long id) {
    super.onListItemClick(list, v, position, id);
    //Receber de novo os dados, para poder pásálos para a seguinte atividade.
    Intent intent = getIntent();
    String idMenu = intent.getExtras().getString("idMenu");

```

```

String idRes = intent.getExtras().getString("idRestaurante");
String numMesa = intent.getExtras().getString("mesa");
String nomeRes = intent.getExtras().getString("nomeRes");
String Ementa = intent.getExtras().getString("Ementa");
//Chamar a nova atividade passando os dados necessários.
Intent i = new Intent (this,ProdutoActivity.class);
i.putExtra("idMenu", idMenu );
i.putExtra("mesa", numMesa );
i.putExtra("idRestaurante", idRes );
i.putExtra("idProduto", idsProdutos[position]);
i.putExtra("nomeRes", nomeRes);
i.putExtra("Ementa", Ementa);
startActivity(i);
}
}

```

// Fim da classe Principal da atividade.

O código da interface gráfica da atividade Produtos é o seguinte.

```

//Interface gráfico da atividade Produtos.
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/fundo2blue"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.javacodegeeks.androidqrexample.ProdutosActivity" >
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:padding="10dp"
        android:text="TextView"
        android:textColor="#FFFFFF"
        android:textSize="20dp" />
    <ListView
        android:id="@+id/list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView1"
        android:layout_centerHorizontal="true"
        android:alpha="0.6"
        android:background="#CFD8DC"
        android:divider="#80000000"
        android:dividerHeight="2.0sp"
        android:textColor="#DCE775"
        android:textSize="25sp" >
    </ListView>
</RelativeLayout>

```

De seguida, e depois de selecionarmos o produto que queremos, vamos ter várias opções. Neste preciso ponto da aplicação podemos ver uma foto descriptiva do produto, o seu nome, o seu preço e uma breve descrição. Temos também algumas opções mais avançadas como adicionar este produto ao nosso pedido, alterar a quantidade desejada, ou partilhar a informação no Facebook¹⁰, ou depois de adicionar pelo menos um produto ao nosso pedido, temos acesso a opção de Finalizar Pedido, o que nos aponta para a última fase da aplicação para a funcionalidade de efetuar pedidos. Em baixo vemos uma figura ilustrativa de um produto, e as respetivas funcionalidades anteriormente descritas.



Figura 10 - Ecrã de visualização de um produto.

Nesta fase da aplicação, temos várias opções e ações a decorrer, até agora tinha sido apenas mostrado ao utilizador a informação com base em texto, agora temos o carregamento de uma imagem, que esta guardada no servidor e que através de um Link Web, é transformada em uma imagem (Bitmap), e é carregada dentro de um objeto ImageView. De seguida é mostrado o código responsável pela atividade Produto.

¹⁰ Rede Social mais utilizada no mundo – www.facebook.com.

```

//Código Referente a atividade Produto da aplicação Android.
package com.bonappetitapp;
// Imports necessários as classes utilizadas
import java.io.IOException;
import java.io.InputStream;
import java.io.UnsupportedEncodingException;
import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.util.EntityUtils;
import org.json.JSONException;
import org.json.JSONObject;
import com.facebook.CallbackManager;
import com.facebook.FacebookSdk;
import com.facebook.login.LoginManager;
import com.facebook.share.model.ShareLinkContent;
import com.facebook.share.widget.ShareDialog;
import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.net.ParseException;
import android.net.Uri;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.StrictMode;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.ImageView;
import android.widget.RelativeLayout;
import android.widget.TextView;
import android.widget.Toast;
//Classe Principal da atividade Produto.
public class ProdutoActivity extends Activity {
    //Declaração de variáveis globais.
    private TextView txtNome, txtDescricao, txtPreco, txtQuanti;
    public Button btnAdicionar,btnFinalizar;
    public EditText quanti;
    public ImageButton btnFace;
    public static String numPedido = "null";
    public static boolean finalizar = false;
    public String nomePro = "";
    public String proDescri = "";
    public Uri foto;
    //Declaração de variáveis globais para o funcionamento do SDK do Facebook.
    public CallbackManager callbackManager;
    public ShareDialog shareDialog;
    static LoginManager manager;
    //Classe onCreate que só corre uma vez no inicio da atividade.
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_produto);

```

```

//Inicialização do SDK do Facebook.
FacebookSdk.sdkInitialize(getApplicationContext());
callbackManager = CallbackManager.Factory.create();
//Referencia aos objetos da parte gráfica.
txtNome = (TextView) findViewById(R.id.textView1);
txtDescricao = (TextView) findViewById(R.id.textView2);
txtPreco = (TextView) findViewById(R.id.textView3);
txtQuanti = (TextView) findViewById(R.id.textView6);
btnAdicionar = (Button) findViewById(R.id.button1);
btnFinalizar = (Button) findViewById(R.id.button2);
quanti = (EditText) findViewById(R.id.editText1);
btnFace = (ImageButton) findViewById(R.id.BtnFace);
//Receber os dados enviados pela a atividade, que chamou este ecrã.
Intent intent = getIntent();
final String idMenu = intent.getExtras().getString("idMenu");
final String idRes = intent.getExtras().getString("idRestaurante");
final String idPro = intent.getExtras().getString("idProduto");
final String mesa = intent.getExtras().getString("mesa");
final String nomeRes = intent.getExtras().getString("nomeRes");
final String Ementa = intent.getExtras().getString("Ementa");
//Teste de verificação para alterar o aspeto da atividade consoante o necessário. (Se for pela
atividade Reservar Mesa, ao clicar na opção ver ementa, o layout desta atividade muda,
escondendo os objetos e alterando o fundo de ecrã)
if(Ementa != null && Ementa.equals("Ementa")){
    btnAdicionar.setVisibility(View.GONE);
    btnFinalizar.setVisibility(View.GONE);
    txtQuanti.setVisibility(View.GONE);
    quanti.setVisibility(View.GONE);
    btnFace.setVisibility(View.GONE);
    RelativeLayout rl = (RelativeLayout) findViewById(R.id.relativeLayoutId);
    rl.setBackgroundResource(R.drawable.fundo4blue);
}
//Carregar toda a informação acerca do produto (nome, descrição, preço e foto), utilizando a
classe RestAPI, no formato JSON.
try
{
    String result = new RestAPI().getResult("https://bon-appetit-
server.herokuapp.com/api/restaurantes/" + idRes + "/menus/" + idMenu + "/produtos/" + id
Pro);
    //Criação do objeto JSON com o resultado obtido do pedido http.
    JSONObject json = new JSONObject(result);
    //Colocação dos dados nos respetivos objetos na parte gráfica.
    txtNome.setText(json.getString("nome"));
    nomePro = json.getString("nome");
    txtDescricao.setText(json.getString("descricao"));
    proDescri = json.getString("descricao");
    txtPreco.setText(json.getString("preco") + "€");
    //Carregar a foto através da classe DownloadImageTask através do objeto JSON
    new DownloadImageTask((ImageView)
findViewById(R.id.imageView1)).execute(json.getString("foto"));
    //Guardar url da foto para poder partilhar depois no facebook.
    foto = Uri.parse(json.getString("foto"));
}
catch (Exception e)
{
    Toast.makeText(this, "Link Inválido", Toast.LENGTH_LONG).show();
}
//Variável para depois utilizar na partilha de informação pelo Facebook.
final String nomeProduto = nomeRes + ":" + nomePro;

```

```

//Ultrapassar erro de Networking devido a versão do SDK utilizado.
int SDK_INT = android.os.Build.VERSION.SDK_INT;
if (SDK_INT > 8)
{
    StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder()
        .permitAll().build();
    StrictMode.setThreadPolicy(policy);
//Método responsável pelo click do botão Adicionar Produto.
btnAdicionar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //Criar um objeto json e colocar os pares de valor mapeados pelo servidor
        JSONObject jsonobj = new JSONObject();
        try {
            jsonobj.put("pedido_id", numPedido);
            jsonobj.put("produto_id", Integer.parseInt(idPro));
            jsonobj.put("quantidade", Integer.parseInt(quanti.getText().toString()));
        } catch (JSONException e) {
            //TODO Auto-generated catch block
            e.printStackTrace();
        }
        //Crie um cliente http e URL post.
        DefaultHttpClient httpclient = new DefaultHttpClient();
        HttpPost httppostreq = new HttpPost("https://bon-appetit
server.herokuapp.com/api/restaurantes/" + idRes + "/mesas/" + mesa + "/pedido");
        //Criar uma string. A string será anexada ao URL no formato requisitado pelo HTTP
        POST
        try {
            StringEntity se = new StringEntity(jsonobj.toString());
            se.setContentType("application/json; charset=UTF-8");
            //Configurar a entidade na requisição post.
            httppostreq.setEntity(se);
            //Execute a requisição POST.
            HttpResponse httpresponse = httpclient.execute(httppostreq);
            //Para receber a resposta do servidor após a execução do HTTP POST
            String responseText = null;
            try {
                responseText = EntityUtils.toString(httpresponse.getEntity());
            } catch (ParseException e) {
                e.printStackTrace();
            }
            //Receber a string de resposta no novo objeto jSON e os valores dele.
            JSONObject jsonResposta = new JSONObject(responseText);
            Toast.makeText(ProdutoActivity.this, "Pedido Adicionado",
            Toast.LENGTH_LONG).show();
            //Atualização do numPedido através da resposta do post.
            numPedido = jsonResposta.getString("pedido_id");

        } catch (UnsupportedEncodingException e) {
            //TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ClientProtocolException e) {
            //TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            //TODO Auto-generated catch block
            e.printStackTrace();
        } catch (JSONException e) {
            //TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}

```

```

        }

        //Variável que permite aceder a opção de concluir Pedido
        finalizar = true;
    } //fim do try
}); //Fim do método responsável pelo click do botão Adicionar Produto.
//fim do if do stried mode.
//Método responsável pelo click do botão Finalizar Pedido.
btnFinalizar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //Feita uma verificação primeiro de já foi adicionado pelo menos um produto ao
        pedido.
        if(finalizar){
            Intent i = new Intent(ProdutoActivity.this,PedidoActivity.class);
            i.putExtra("idRestaurante", idRes );
            i.putExtra("idPedido", numPedido);
            i.putExtra("nMesa", mesa);
            startActivity(i);
        }
        else{
            Toast.makeText(ProdutoActivity.this,"Por Favor adicione pelo menos um
            produto ao seu pedido, obrigado",Toast.LENGTH_LONG).show();
        }; //Fim do if
    }
}); //Fim do método responsável pelo click do botão Finalizar Pedido.

//Criação do ShareDialog para o Facebook.
shareDialog = new ShareDialog(this);

//Método responsável pelo click do botão Facebook e partilha dos dados.
btnFace.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (ShareDialog.canShow(ShareLinkContent.class)) {
            ShareLinkContent linkContent = new ShareLinkContent.Builder()
                //Informação que é partilhada no facebook.
                .setContentTitle(nomeProduto)
                .setContentUrl(Uri.parse("https://bon-appetit- server.herokuapp.com/"))
                .setImageUrl(foto)
                .setContentDescription(proDescri)
                .build();
            shareDialog.show(linkContent);
        }
    }
}); //Fim do método responsável pelo click do botão Facebook.
//Fim do onCreate.
//Classe responsável por transformar uma foto no formato URL em bitmap, para mostrar a foto do
produto. Este método está a ser chamado quando é carregado os dados no onCreate.
public class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    ImageView bmImage;
    public DownloadImageTask(ImageView bmImage) {
        this.bmImage = bmImage;
    }
    //Classe executada em background por questões de performance, para não subcarregar a
atividade principal.
    protected Bitmap doInBackground(String... urls) {
        String urldisplay = urls[0];
        Bitmap mIcon11 = null;
        try {
            InputStream in = new java.net.URL(urldisplay).openStream();
            mIcon11 = BitmapFactory.decodeStream(in);
        }

```

```

    } catch (Exception e) {
        Log.e("Error", e.getMessage());
        e.printStackTrace();
    }
    //Retornar a imagem em bitmap.
    return mIcon11;
}
//Método para fazer o set da Imagem.
protected void onPostExecute(Bitmap result) {
    bmImage.setImageBitmap(result);
}
//Fim da classe DownloadImageTask
//Fim da classe Principal da atividade.

```

O código referente a interface gráfica da atividade Produto é o seguinte.

```

//Interface gráfico da atividade Produto.
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/relativeLayoutId"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/fundo3blue"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.javacodegeeks.androidqrcodeexample.ProdutoActivity" >
    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:scaleType="fitStart" />
    <TextView
        android:id="@+id/textView2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView4"
        android:layout_centerVertical="true"
        android:paddingLeft="10dp"
        android:paddingRight="10dp"
        android:text="TextView"
        android:textColor="#000000"
        android:textSize="15dp" />
    <TextView
        android:id="@+id/textView4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/textView2"
        android:layout_alignLeft="@+id/textView1"
        android:paddingLeft="10dp"
        android:text="Descrição :"
        android:textAppearance="?android:attr/textAppearanceMedium" />
    <TextView
        android:id="@+id/textView1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_above="@+id/textView4"
        android:paddingLeft="10dp"
        android:text="TextView"

```

```

    android:textColor="#000000"
    android:textSize="25dp"
    android:textStyle="bold" />
<TextView
    android:id="@+id/textView5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView2"
    android:paddingLeft="10dp"
    android:text="Preço :"
    android:textAppearance="?android:attr/textAppearanceMedium" />
<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView5"
    android:paddingLeft="10dp"
    android:text="TextView"
    android:textColor="#000000"
    android:textSize="15dp" />
<TextView
    android:id="@+id/textView6"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/textView3"
    android:layout_marginTop="50dp"
    android:paddingLeft="10dp"
    android:text="Quantidade :"
    android:textAppearance="?android:attr/textAppearanceMedium" />
<EditText
    android:id="@+id/editText1"
    android:layout_width="30dp"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/textView6"
    android:layout_alignBottom="@+id/textView6"
    android:layout_marginLeft="8dp"
    android:layout_toRightOf="@+id/textView6"
    android:ems="10"
    android:inputType="number"
    android:text="1" />
<Button
    android:id="@+id/button1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignTop="@+id/editText1"
    android:layout_marginLeft="8dp"
    android:layout_marginRight="10dp"
    android:layout_toRightOf="@+id/editText1"
    android:alpha="0.6"
    android:background="#CFD8DC"
    android:text="Adicionar"
    android:textColor="#FFFFFF" />
<Button
    android:id="@+id/button2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/button1"

```

```

    android:layout_marginLeft="100dp"
    android:layout_marginRight="10dp"
    android:layout_marginTop="22dp"
    android:alpha="0.6"
    android:background="#CFD8DC"
    android:text="Finalizar Pedido"
    android:textColor="#FFFFFF" />
<ImageButton
    android:id="@+id/BtnFace"
    android:layout_width="50dp"
    android:layout_height="50dp"
    android:layout_alignRight="@+id/button1"
    android:layout_below="@+id/textView5"
    android:adjustViewBounds="true"
    android:padding="0dp"
    android:scaleType="fitCenter"
    android:src="@drawable/facelogo" />
</RelativeLayout>

```

Como descrito anteriormente, e visualizado na figura anterior, temos a funcionalidade de partilhar a informação sobre o produto e o restaurante em causa. Esta “*Feature*” foi idealizada no sentido da publicidade ao estabelecimento de restauração, tendo em conta, a importância que a rede social Facebook têm nos dias de hoje. Para alcançar este feito foi necessário primeiro tratar de alguns aspetos importantes e de relevância para a aplicação.

A utilização do Facebook

O SDK do Facebook para Android permite integrar, numa aplicação Android nativa, todas as funcionalidades da rede social Facebook desde a autenticação até a partilha de conteúdos quer seja fotos, vídeos, estados, ou *sites* entre outros. Para a sua utilização foi necessário fazer *download* do SDK do Facebook¹¹, depois importar o projeto Facebook para o IDE Eclipse, e tornar esse projeto uma biblioteca para a nossa aplicação. Quando concluída esta fase, temos que aceder ao *site* de oficial do Facebook para Programadores¹², e tornar a nossa aplicação referenciada no mesmo site, aqui será necessário gerar uma Hash Key, por uma questão de segurança e em ambiente de desenvolvimento, através da ferramenta Keytool¹³. Todo este processo está descrito de uma forma resumida para mais informação de como fazer, por favor consultar o *site* oficial do Facebook para Programadores. (Queirós. 2014)

Depois de estar configurada a plataforma de desenvolvimento, é necessário alterar o ficheiro de manifesto da nossa aplicação, para podermos usufruir das opções que o SDK do

¹¹ Download do SDK do Facebook para Android - <https://developers.facebook.com/docs/android>.

¹² Site Oficial do Facebook para programadores - <https://developers.facebook.com/>.

¹³ Ferramenta utilizada para gerar a chave Hash.

Facebook nos oferece dentro da nossa aplicação. O código com as alterações encontram-se no ficheiro manifesto anteriormente monstrado.

Em baixo podemos ver várias imagens do processo de partilha de informação na página do Facebook através da aplicação.



Figura 11 - Ecrã de partilha de informação no Facebook

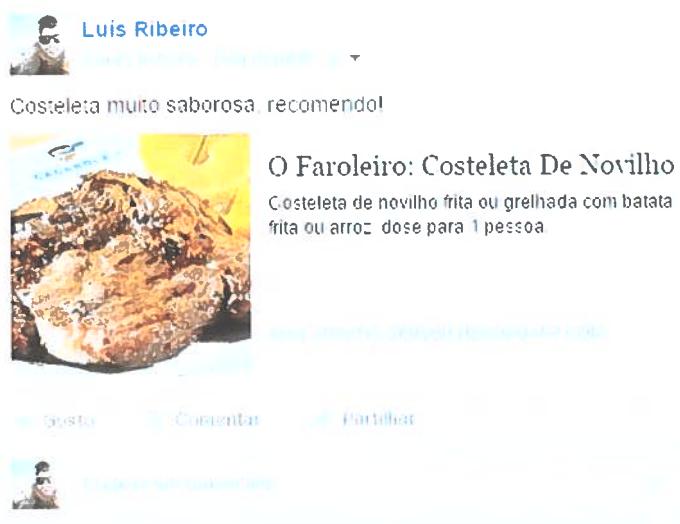


Figura 12 - Produto partilhado já na página do Facebook.

Até agora quando era efetuado pedidos ao servidor, tinha sido do tipo “Get”¹⁴, ou seja, o servidor nos devolve informação apenas. Neste ecrã da aplicação temos a opção como em cima referida de adicionar o produto ao nosso pedido, então neste caso, o pedido HTTP já não vai ser de “Get” mas sim “Post”¹⁵, ou seja ao contrário, sendo assim vai ser a aplicação a enviar os dados para o servidor. Assim sendo os métodos anteriores mencionados já não são iguais para fazer o método “Post”. O código responsável por adicionar o produto selecionado ao nosso pedido, no formato JSON, através dos métodos `HTTPPost`¹⁶, está no código desta atividade anteriormente demostrado.

Depois de ser selecionado a opção de Finalizar Pedido, é então nos mostrado o ultimo ecrã, antes de efetuar o pedido. Neste último ecrã é nos apresentado todos os produtos que fomos adicionando ao nosso pedido, aqui, temos a possibilidade de ver todos os produtos, assim como, as suas quantidades. É neste momento que temos oportunidade de efetuar algumas alterações aos produtos solicitados anteriormente, podemos selecionar a opção atualizar, que prontamente abre uma janela no centro do ecrã para podermos inserir a nova quantidade desejada. Na outra opção temos então a oportunidade de eliminar o produto do nosso pedido. Por último temos a opção de efetuar o Concluir Pedido e assim termina o processo, voltando ao ecrã principal da aplicação. Em baixo podemos ver uma figura do último ecrã com vários produtos adicionados.

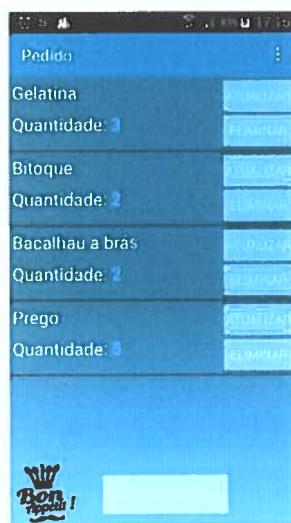


Figura 13 - Ecrã com os produtos do pedido.

¹⁴ Receber dados do servidor.

¹⁵ Enviar dados para o servidor.

¹⁶ O método `HTTPPost` - <http://developer.android.com/reference/org/apache/http/client/methods/HttpPost.html>.

Para obter este comportamento podemos destacar três elementos novos. temos a opção de eliminar o produto ao nosso pedido, que requer neste caso um pedido de HTTP Delete. A visualização dos dados e dos elementos, está a cargo de um ARRAY Adapter¹⁷, que acede a classe Pedido, para aceder aos métodos “get” e “set” respetivamente, com isto, conseguimos a customização da ListView, para a apresentação da informação referente aos produtos e a inclusão de dois botões, para efetuar alterações aos mesmos. Por último temos que falar na característica de ser apresentada uma janela “POP UP”, através do método AlertDialog¹⁸ do Android, que é responsável pela interação do atualizar a quantidade, assim como, o eliminar os produtos.

De seguida é mostrado o código, em relação a atividade, a classe Pedido, a classe MyArrayAdapter, assim como, o código responsável pela interface gráfica desta atividade.

```
//Código Referente a atividade Pedido da aplicação Android.  
package com.bonappetitapp;  
// Imports necessários as classes utilizadas  
import java.io.IOException;  
import java.io.UnsupportedEncodingException;  
import java.util.ArrayList;  
import org.apache.http.HttpResponse;  
import org.apache.http.client.ClientProtocolException;  
import org.apache.http.client.methods.HttpPost;  
import org.apache.http.entity.StringEntity;  
import org.apache.http.impl.client.DefaultHttpClient;  
import org.apache.http.util.EntityUtils;  
import org.json.JSONArray;  
import org.json.JSONException;  
import org.json.JSONObject;  
import com.javacodegeeks.androidqrcodeexample.PedidoActivity;  
import com.javacodegeeks.androidqrcodeexample.R;  
import com.javacodegeeks.androidqrcodeexample.MyArrayAdapter;  
import com.javacodegeeks.androidqrcodeexample.Pedido;  
import android.app.Activity;  
import android.content.Intent;  
import android.net.ParseException;  
import android.os.Bundle;  
import android.os.StrictMode;  
import android.util.Log;  
import android.view.Menu;  
import android.view.MenuItem;  
import android.view.View;  
import android.widget.Button;  
import android.widget.ListView;  
import android.widget.Toast;  
//Classe Principal da atividade Pedido.  
public class PedidoActivity extends Activity {  
    //Declaração de variáveis globais.  
    static ListView listview;
```

¹⁷ Como funciona o ArrayAdapter - <http://developer.android.com/reference/android/widget/ArrayAdapter.html>.

¹⁸ Referencia para o método de utilização do AlertDialog -
<http://developer.android.com/reference/android/app/AlertDialog.html>.

```

MyArrayAdapter PedidoArrayAdapter;
ArrayList<Pedido> PedidoArray = new ArrayList<Pedido>();
public Button btnConcluir;
public static String estadoDoPedido = "";
//Arrays que vão guardar o id do produto dentro do idsPedidos, e guardar os ids dos restaurantes dentro
do idsRes, valor esse que depois é acedido pelo get 'api/restaurantes/:idR/pedidos/:idP' =>
'api#getPedido' dentro da atividade Conta Corrente.
public static String[] idsPedidos = new String[40];
public static String[] idsRes = new String[40];
public static int kj = 0;
public static String idRespp;
public static String idPedidopp;
//Classe onCreate que só corre uma vez no inicio da atividade.
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_pedido);
    //Referencia aos objetos da parte gráfica.
    btnConcluir = (Button) findViewById(R.id.button1);
    //Receber os dados enviados pela a atividade, que chamou este ecrã.
    Intent intent = getIntent();
    final String idRes = intent.getExtras().getString("idRestaurante");
    final String idPedido = intent.getExtras().getString("idPedido");
    final String numMesa = intent.getExtras().getString("nMesa");
    idRespp = idRes;
    idPedidopp = idPedido;
    //Carregar toda a informação acerca dos Pedidos efectuados com nome e quantidade. Através da
    Classe Pedido.java. A informação que é acedida depois pelo MyArrayAdpater.java e
    apresentada na atividade Pedido. Foi necessário esta técnica de utilização de um ArrayAdapter
    devido a informação customizada dentro de cada linha da ListView.
    try
    {
        String result = new RestAPI().getResult("https://bon-appetit-
server.herokuapp.com/api/restaurantes/" + idRes + "/pedidos/" + idPedido);
        //Criação do objeto JSON com o resultado obtido do pedido http.
        JSONObject json = new JSONObject(result);
        JSONArray produtosJson = json.getJSONArray("produtos");
        //For responsável pela iteração dos id's de produtos, adicionados ao pedido.
        for(int i=0; i<produtosJson.length(); i++)
        {
            //Criação do objeto JSON com o resultado obtido do pedido http.
            JSONObject obj = produtosJson.getJSONObject(i);
            //Passagem de paramentros ao construtor da classe Pedido.
            PedidoArray.add(new Pedido(obj.getString("nome"),
                obj.getString("quantidade"), obj.getString("id")));
        }
    }
    catch (Exception e)
    {
        Toast.makeText(this, "Link Inválido", Toast.LENGTH_LONG).show();
    }
    //Referência ao objeto para a parte gráfica.
    PedidoArrayAdapter = new MyArrayAdapter(this, R.layout.list_item, PedidoArray);
    listview = (ListView) findViewById(R.id.listView);
    //Colocação dos dados do PedidoArrayAdapter, dentro da Listview da atividade.
    listview.setAdapter(PedidoArrayAdapter);
    //ultrapassar erro de Networking devido a versão do SDK utilizado.
    int SDK_INT = android.os.Build.VERSION.SDK_INT;
    if (SDK_INT > 8)
    {
}

```

```

StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder()
    .permitAll().build();
StrictMode.setThreadPolicy(policy);
//Método responsável pelo click do botão Finalizar Pedido.
btnConcluir.setOnClickListener(new View.OnClickListener() {
    @Override
        public void onClick(View v) {
            //Criar um objeto json e colocar os pares de valor mapeados pelo servidor
            JSONObject jsonobj = new JSONObject();
            try {
                jsonobj.put("pedido_id", idPedido);
            } catch (JSONException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            //Crie um cliente http e URL post.
            DefaultHttpClient httpclient = new DefaultHttpClient();
            HttpPost httppostreq = new HttpPost("https://bon-appetit-
            server.herokuapp.com/api/restaurantes/" + idRes + "/mesas/" + numMesa + "/final
            izar");
            //Criar uma string. A string será anexada ao URL no formato requisitado pelo HTTP
            POST
            try {
                StringEntity se = new StringEntity(jsonobj.toString());
                se.setContentType("application/json; charset=UTF-8");
                //Configurar a entidade na requisição post.
                httppostreq.setEntity(se);
                //Execute a requisição POST.
                HttpResponse httpresponse = httpclient.execute(httppostreq);
                //Para receber a resposta do servidor após a execução do HTTP POST
                String responseText = null;
                try {
                    responseText = EntityUtils.toString(httpresponse.getEntity());
                } catch (ParseException e) {
                    e.printStackTrace();
                    Log.i("Parse Exception", e + "");
                }
                //Guardar a resposta do servidor para depois mostrar ao utilizador.
                estadoDoPedido = responseText;
                //Indicar ao utilizador se o pedido foi feito.
                Toast.makeText(PedidoActivity.this, "O seu pedido foi efetuado
                com " + estadoDoPedido + " Aguarde",
                Toast.LENGTH_LONG).show();
                } catch (UnsupportedEncodingException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (ClientProtocolException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                //Variável utilizada para não poder finalizar pedido. sem ter pelo menos um
                produto adicionado.
                ProdutoActivity.finalizar = false;
                //Variáveis utilizadas na atividade Conta Corrente.
                idsPedidos[kj] = idPedidopp;
                idsRes[kj] = idRespp;
                kj++;
            }
        }
    }
}

```

```

        //Variável que controla a criação de Pedidos novos no servidor.
        ProdutoActivity.numPedido = "null";
        //Chamada do ecrã principal depois de ter finalizado o pedido.
        Intent i = new Intent (PedidoActivity.this, BonAppetitActivity.class);
        startActivity(i);
        finish();
    } //Fim do Try.
} //Fim do método responsável pelo click do botão Finalizar Pedido.
//Fim do if do strited mode
} //Fim do onCreate.
} //Fim da classe principal.

```

```

//Classe Pedido.java responsável pelos Pedidos na atividade Pedido.
package com.bonappetitapp;
public class Pedido {
    private String id;
    private String name;
    private String quantidade;
    //Construtor da classe Pedido.
    public Pedido(String name, String quanti, String id) {
        super();
        this.name = name;
        this.quantidade= quanti;
        this.id = id;
    }
    //Métodos para fazer o "Get" e "Set" dos dados passados por parâmetro.
    public String getQuanti(){
        return quantidade;
    }
    public void setQuanti(String quanti){
        this.quantidade= quanti;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getId()
    {
        return id;
    }
} // Fim da classe Pedido.java

```

```

//Classe MyArrayAdapter.java responsável pelos dados introduzidos na Listview da classe Pedido, utilizando a
classe Pedido.java. Nesta classe encontra-se os métodos pelos botões de atualizar quantidades nos produtos
selecionados, assim como o método do botão de eliminar os produtos ao pedido.
package com.bonappetitapp;
// Imports necessários as classes utilizadas
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;

```

```

import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.util.EntityUtils;
import org.json.JSONException;
import org.json.JSONObject;
import com.bonappetitapp.R;
import com.bonappetitapp.Pedido;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.net.ParseException;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
public class MyArrayAdapter extends ArrayAdapter<Pedido> {
    PedidoActivity context;
    int layoutResourceId;
    ArrayList<Pedido> pedidos = new ArrayList<Pedido>();
    public MyArrayAdapter(PedidoActivity context, int layoutResourceId, ArrayList<Pedido> studs) {
        super(context, layoutResourceId, studs);
        this.layoutResourceId = layoutResourceId;
        this.context = context;
        this.pedidos = studs;
    }
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View item = convertView;
        PedidoWrapper PedidoWrapper = null;
        //Referência aos objetos para a parte gráfica.
        if (item == null) {
            LayoutInflater inflater = ((Activity) context).getLayoutInflater();
            item = inflater.inflate(layoutResourceId, parent, false);
            PedidoWrapper = new PedidoWrapper();
            PedidoWrapper.name = (TextView) item.findViewById(R.id.textName);
            PedidoWrapper.id = (TextView) item.findViewById(R.id.textId);
            PedidoWrapper.edit = (Button) item.findViewById(R.id.btnEdit);
            PedidoWrapper.delete = (Button) item.findViewById(R.id.btnDelete);
            PedidoWrapper.quantidade = (TextView) item.findViewById(R.id.quantidade);
            item.setTag(PedidoWrapper);
        } else {
            PedidoWrapper = (PedidoWrapper) item.getTag();
        }
        //Fim do if
        //Váriaveis utilizadas para saber qual a posição dos objetos.
        Pedido pedido = pedidos.get(position);
        PedidoWrapper.positionOnView = position;
        //Metodos utilizados para colocar os valores nos respetivos campos.
        PedidoWrapper.name.setText(pedido.getName());
        PedidoWrapper.id.setText(pedido.getId());
        PedidoWrapper.quantidade.setText(pedido.getQuanti());
        //Metodo Responsável pelo click do botão atualizar (edit).
        PedidoWrapper.edit.setOnClickListener(new OnClickListener() {
            PedidoWrapper pa;

```

```

    @Override
    public void onClick(View v) {
        //Chamada da classe openAlert2
        openAlert2(v, pa);
    }
    private OnClickListener init(PedidoWrapper pa){
        this.pa = pa;
        return this;
    }
}.init(PedidoWrapper)); //Fim do edit.setOnClickListener
//Metodo Responsável pelo click do botão Eliminar (delete).
PedidoWrapper.delete.setOnClickListener(new OnClickListener() {
    PedidoWrapper pa;
    @Override
    public void onClick(View v) {
        //Chamada da classe openAlert.
        openAlert(v, pa);
    }
    private OnClickListener init(PedidoWrapper pa){
        this.pa = pa;
        return this;
    }
}.init(PedidoWrapper)); //Fim do delete.setOnClickListener
    return item;
}//fim do View getView
//Classe PedidoWrapper.
static class PedidoWrapper
{
    int positionOnView;
    TextView name;
    TextView id;
    TextView quantidade;
    Button edit;
    Button delete;
}//Fim da classe pedidoWrapper
//Classe responsável pela criação de uma janela AlertDialog (popup) para a eliminação de produtos ao pedido.
Esta classe está a ser chamada pelo método de click do botão delete.
void openAlert(View view, PedidoWrapper pa) {
    final PedidoWrapper newPA = pa;
    //Criação de uma janela de AlertDialog (popup).
    AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(context);
    //Mensagem da janela de popup.
    alertDialogBuilder.setMessage("Tem a certeza que pretende eliminar o produto ao seu pedido?");
    //Escolher a resposta sim do popup de eliminação do produto.
    alertDialogBuilder.setPositiveButton("Sim",new DialogInterface.OnClickListener() {
        //Metodo responsável pelo click do botão sim da janela de popup de eliminação do produto.
        @Override
        public void onClick(DialogInterface dialog,int id) {
            //Variaveis utilizadas para fazer o pedido http ao servidor neste caso de Delete.
            String valorIdRes = PedidoActivity.idRespp;
            String valorIdPedido = PedidoActivity.idPedidopp;
            //Processo de pedido http DELETE ao servidor no formato JSON.
            URL url = null;
            try {
                url = new URL("https://bon-appetit-server.herokuapp.com/api/restaurantes/" +
                    valorIdRes + "/pedidos/" + valorIdPedido + "/pedidoproduto/" +
                    newPA.id.getText());
            } catch (MalformedURLException exception) {

```

```

        exception.printStackTrace();
    }
    HttpURLConnection httpURLConnection = null;
    try {
        httpURLConnection = (HttpURLConnection) url.openConnection();
        httpURLConnection.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
        httpURLConnection.setRequestMethod("DELETE");
    } catch (IOException exception) {
        exception.printStackTrace();
    } finally {
        if (httpURLConnection != null) {
            httpURLConnection.disconnect();
        }
    }
    //Método que evoca o onCreate da atividade Pedido, e assim redesenha todos os
    //produtos no ecrã.
    context.recreate();
    //Mensagem enviada ao utilizador a informar que o produto foi eliminado do pedido.
    Toast.makeText(context, "O produto foi eliminado do pedido",
    Toast.LENGTH_LONG).show();
}
});//Fim do método do click do botão sim do eliminar produto.
//Escolher a resposta não do popup de eliminação do produto.
AlertDialogBuilder.setNegativeButton("Não",new DialogInterface.OnClickListener() {
    //Metodo responsável pelo click do botão não da janela de popup de eliminação do produto.
    @Override
    public void onClick(DialogInterface dialog,int id) {
        //Fecha a janela de popup de eliminação do produto.
        dialog.cancel();
        //Mensagem enviado ao utilizador a informar que não eliminou nenhum produto.
        Toast.makeText(context, "Não foi efetuado nenhuma operação",
        Toast.LENGTH_LONG).show();
    }
});
AlertDialog alertDialog = alertDialogBuilder.create();
// show alert
alertDialog.show();
}//Fim da classe openAlert.
//Classe responsável pela criação de uma janela AlertDialog (popup) para a atualização da quantidade do
//produto ao pedido. Esta classe está a ser chamada pelo método de click do botão edit. Esta janela contém uma
//EditText para poder introduzir a nova quantidade.
void openAlert2(View view, PedidoWrapper pa) {
    final PedidoWrapper newPA = pa;
    LayoutInflater li = LayoutInflater.from(context);
    View promptsView = li.inflate(R.layout.prompts,null);
    //Criação de uma janela de AlertDialog (popup).
    AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder (context);
    //Criar prompts.xml com o contrutor alertDialog.
    alertDialogBuilder.setView(promptsView);
    //Guardar o valor introduzido da nova quantidade pelo utilizador
    final EditText userInputQuanti = (EditText)
    promptsView.findViewById(R.id.editTextDialogUserInput);
    //Construir a janela de Dialog.
    alertDialogBuilder
        .setCancelable(false)
        //Caso click no botão de ok.
        .setPositiveButton("OK",new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog,int id) {
                //Váriaveis utilizadas pelo pedido http ao servidor, em formato JSON.

```

```

String valorIdRes = PedidoActivity.idRespp;
String valorIdPedido = PedidoActivity.idPedidopp;
//Criar um objeto json e colocar os pares de valor mapeados pelo servidor.
JSONObject jsonobj = new JSONObject();
try {
    jsonobj.put("quantidade", userInputQuanti.getText());
} catch (JSONException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
//Crie um cliente http e URL post.
DefaultHttpClient httpclient = new DefaultHttpClient();
HttpPost httppostreq = new HttpPost("https://bon-appetit-
server.herokuapp.com/api/restaurantes/" + valorIdRes + "/pedidos/"
+ valorIdPedido + "/pedidoproduto/" + newPA.id.getText());
//Criar uma string. A string será anexada ao URL no formato requisitado pelo HTTP POST
try {
    StringEntity se = new StringEntity(jsonobj.toString());
    se.setContentType("application/json; charset=UTF-8");
    //Configurar a entidade na requisição post.
    httppostreq.setEntity(se);
    //Execute a requisição POST.
    HttpResponse httpresponse = httpclient.execute(httppostreq);
    //Para receber a resposta do servidor após a execução do HTTP POST
    String responseText = null;
    try {
        responseText = EntityUtils.toString(httpresponse.getEntity());
    } catch (ParseException e) {
        e.printStackTrace();
    }
    } catch (UnsupportedEncodingException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (ClientProtocolException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    //Método que evoca o onCreate da atividade Pedido, e assim redesenha todos os produtos no
    //epra.
    context.recreate();
    //Mensagem enviada ao utilizador a informar que a quantidade do produto foi atualizada.
    Toast.makeText(context, "A quantidade foi atualizada", Toast.LENGTH_LONG).show();
}
}
//Caso click no botão de cancelar.
.setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog,int id) {
        //Feixa a caixa de popup.
        dialog.cancel();
    }
});
//Criação do AlertDialog.
AlertDialog alertDialog = alertDialogBuilder.create();
//Mostrar o AlertDialog.
alertDialog.show();
//Fim da classe openAlert2.
//Fim da classe principal.

```

```

//Interface gráfico da atividade Pedido.
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/fundo3blue"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.javacodegeeks.androidqrcodeexample.PedidoActivity" >
    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_marginBottom="15dp"
        android:layout_weight="1"
        android:cacheColorHint="#FFFFFF"
        android:descendantFocusability="beforeDescendants"
        android:divider="#80000000"
        android:dividerHeight="2dp" />
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="15dp"
        android:layout_marginLeft="115dp"
        android:layout_marginRight="10dp"
        android:alpha="0.6"
        android:background="#CFD8DC"
        android:padding="10dp"
        android:text="Concluir Pedido"
        android:textColor="#FFFFFF" />
</LinearLayout>

```

```

//Interface gráfico dos objetos que se encontram dentro da Listview da atividade Pedido.
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:padding="4dp" >
    <TextView
        android:id="@+id/textName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:text="Nome:"
        android:textAppearance="?android:attr/textAppearanceLarge" />
    <Button
        android:id="@+id/btnEdit"
        android:layout_width="90dp"
        android:layout_height="40dp"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:alpha="0.6"
        android:background="#CFD8DC" />

```

```

    android:focusable="false"
    android:focusableInTouchMode="false"
    android:text="Atualizar"
    android:textColor="#FFFFFF" />
<Button
    android:id="@+id/btnDelete"
    android:layout_width="90dp"
    android:layout_height="40dp"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/btnEdit"
    android:layout_marginTop="3dp"
    android:alpha="0.6"
    android:background="#CFD8DC"
    android:focusable="false"
    android:focusableInTouchMode="false"
    android:text="Eliminar"
    android:textColor="#FFFFFF" />
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/textName"
    android:layout_marginTop="10dp"
    android:text="Quantidade: "
    android:textAppearance="?android:attr/textAppearanceLarge" />
<TextView
    android:id="@+id/quantidade"
    android:layout_width="30dp"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/textView1"
    android:layout_alignBottom="@+id/textView1"
    android:layout_toRightOf="@+id/textView1"
    android:focusable="true"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:textColor="#8C9EFF" />
<TextView
    android:id="@+id/textId"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/textName"
    android:layout_marginTop="10dp"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:visibility="invisible" />
</RelativeLayout>

```

```

// Interface gráfico da janela de popup para eliminar o Pedido
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/background_light"
    android:orientation="vertical" >
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="1dp"

```

```

    android:background="@android:color/darker_gray"
    android:orientation="vertical" >
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="20dp"
        android:orientation="vertical" >
        <TextView
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Tem a certeza que quer eliminar este produto ao seu pedido" />
        <Button
            android:id="@+id/ok"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Confirmar" />
        <Button
            android:id="@+id/cancelar"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Cancelar" />
    </LinearLayout>
</LinearLayout>
</LinearLayout>

```

```

// Interface gráfico da janela de popup para atualizar a quantidade dos produtos.
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Insira a nova Quantidade : "
        android:textAppearance="?android:attr/textAppearanceLarge" />
    <EditText
        android:id="@+id/editTextDialogUserInput"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="number" />
    </EditText>
</LinearLayout>

```

Para se alcançar o efeito de “POP UP”, das janelas de visualização, para a alteração da quantidade e supressão dos produtos, recorreu-se a funcionalidade de AlertDialog como mencionado anteriormente.

Em baixo uma figura ilustrativa de uma janela utilizando o objeto AlertDialog.

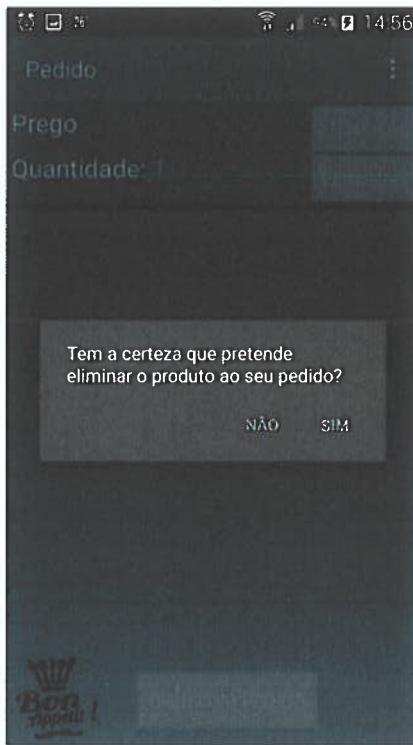


Figura 14 - Janela de POP UP de eliminação de um produto ao pedido.

Como foi mencionado, na visualização da informação foi utilizado mais uma vez o objeto ListView, neste caso, usou-se um ARRAY Adapter para a customização dos elementos que se encontram dentro de cada bloco da ListView. Para este feito, foi necessário também uma classe em java, a classe PEDIDO, que está responsável por nos dar acesso aos métodos “GET” e “SET” dos dados.

Notificação por Toast.

Em todas as ações que o utilizador pode efetuar com a aplicação, é necessário por vezes dar “Feedback”¹⁹ em relação ao resultado dessa mesma ação, neste caso para esse comportamento foi utilizado o objeto Toast²⁰, para a notificação ao utilizador das suas ações sempre que necessário, através de uma mensagem no fundo do ecrã ,que desaparece após alguns segundos, como por exemplo indicar que partilhou algo no Facebook quando a

¹⁹ Significa dar resposta a um determinado acontecimento ou pedido.

²⁰ Referencia do objeto Toast - <http://developer.android.com/reference/android/widget/Toast.html>.

aplicação retorna ao ecrã, ou por exemplo também quando atualiza a quantidade, ou adicionar um produto ao pedido. Em baixo segue o código exemplo responsável por esse comportamento.

Quadro 2 - Código responsável pela mensagem TOAST.

```
Toast.makeText(context, "O produto foi eliminado do pedido", Toast.LENGTH_LONG).show();
```

Em baixo podemos ver duas figuras ilustrativas de como é apresentado a mensagem através do objeto Toast.

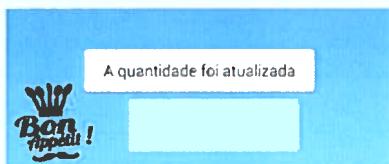


Figura 15 - Mensagem a informar que a quantidade foi alterada.

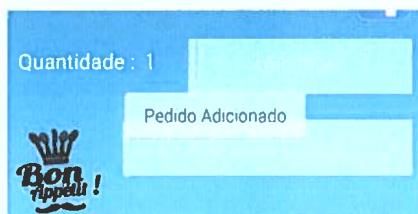


Figura 16 - Mensagem ao adicionar um produto.

Verificar Conta Corrente

Neste segundo tópico, depois de ter sido demonstrado a principal funcionalidade da aplicação Android, vamos também abordar as outras opções ou funcionalidades que o utilizador têm ao seu dispor, nos tópicos seguintes. A segunda funcionalidade é Verificar Conta Corrente, como o nome sugere, é através desta opção que o utilizador vai poder consultar informação acerca dos pedidos efetuados até ao momento.

Nesta funcionalidade, foi pensada e idealizada, para o utilizador ter um melhor controlo dos gastos efetuados com os pedidos que fez, através de uma listagem por pedido, que detalhadamente descreve os produtos, a quantidade, preço por unidade, e preço total por produto e também o preço final do próprio Pedido (o pedido corresponde ao conjunto dos produtos).

A nível da conceção desta tarefa, a programação efetuada ou recursos foram os mesmos que anteriormente foram utilizados, na funcionalidade anterior descrita no presente trabalho, existindo aqui apenas a dificuldade da apresentação dos dados ao utilizador, de forma clara e de fácil leitura. Para este comportamento então, mais uma vez foi usado o objeto ListView para iterar, por cada pedido efetuado, e através de um ARRAY Adapter para a disposição dos dados no ecrã, as opções foram idênticas ao último ecrã do Iniciar Pedido, como demonstra a figura 13.

Para melhor compreensão em baixo vemos uma figura ilustrativa desta funcionalidade, com vários pedidos efetuados.

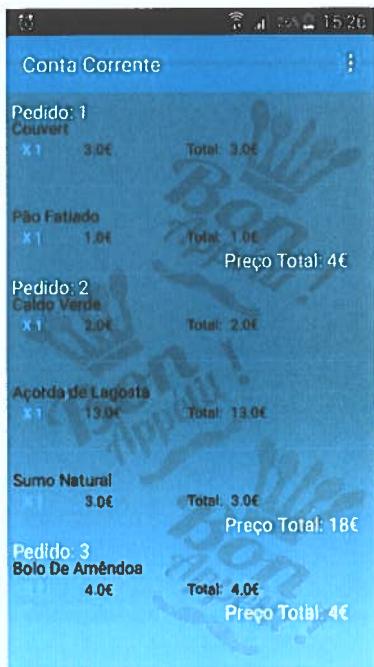


Figura 17 - Ecrã da funcionalidade Conta Corrente.

Como podemos ver pela figura em cima, temos três pedidos efetuados, dentro de cada Pedido temos então os produtos que fazem parte dele (Pedido). Por cada iteração, temos a informação do nome do produto, por de baixo do nome temos a quantidade solicitada, a frente da quantidade o preço unitário do produto, por último temos então o preço total de cada produto, que é obtido pela multiplicação entre a quantidade e o preço unitário. No final de cada Pedido temos então o preço total do mesmo, e assim rapidamente podemos ver que o utilizador efetuou três pedidos, no primeiro com dois produtos, no segundo com três produtos e no ultimo com um produto apenas, e teve um gasto de vinte e seis euros.

O código responsável pela segunda funcionalidade, em relação a atividade, a classe CCorrente, a classe MyArrayAdapter3, assim como, o código responsável pela interface gráfica desta atividade, é o seguinte.

```
//Código Referente a atividade Conta Corrente da aplicação Android.  
package com.bonappetitapp;  
// Imports necessários as classes utilizadas  
import java.util.ArrayList;  
import org.json.JSONArray;  
import org.json.JSONObject;  
import android.app.Activity;  
import android.os.Bundle;  
import android.view.Menu;  
import android.view.MenuItem;  
import android.view.View;  
import android.widget.AdapterView;  
import android.widget.AdapterView.OnItemClickListener;  
import android.widget.ListView;  
import android.widget.TextView;  
import android.widget.Toast;  
//Classe Principal da atividade Conta Corrente.  
public class ContaCorrenteActivity extends Activity {  
    //Declaração de variáveis globais.  
    ListView listview;  
    MyArrayAdapter3 CcorrenteArrayAdapter;  
    ArrayList<CCorrent> cCorrenteArray = new ArrayList<CCorrent>();  
    public TextView txtTotal;  
    public TextView pedidold;  
    public int numPedido = 1;  
    public String preco = "";  
    public int total = 0;  
    public String idres,idpedido;  
    //Classe onCreate que só corre uma vez no inicio da atividade.  
    @Override  
    protected void onCreate(Bundle savedInstanceState)  
    {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_conta_corrente);  
        //Referencia aos objetos da parte gráfica.  
        pedidold = (TextView) findViewById(R.id.pedidold);  
        //Váriaveis Locais.  
        total = 0;  
        cCorrenteArray = new ArrayList<CCorrent>();  
        //For responsável pela iteração dos pedidos efetuados. Com a variável kj vinda da atividade Pedido,  
        assim, como os ids dos pedidos e dos restaurantes.  
        for(int i=0; i<PedidoActivity.kj; i++)  
        //Váriaveis utilizadas para usar na rota dos pedidos http, em relação aos pedidos.  
        idpedido = PedidoActivity.idsPedidos[i];  
        idres = PedidoActivity.idsRes[i];  
        try  
        {  
            String result = new RestAPI().getResult("https://bon-appetit-  
server.herokuapp.com/api/restaurantes/" + idres + "/pedidos/" + idpedido);  
            //Criação do objeto JSON com o resultado obtido do pedido http.  
            JSONObject testes = new JSONObject(result);  
            //Criação do objeto JSONArray com todos os produtos.  
            JSONArray produtosjson = testes.getJSONArray("produtos");  
            //Variável utilizada para o valor total de cada Pedido.
```

```

total = 0;
for(int j=0; j<produtosjson.length(); j++){
    //Calculo do valor total dos Pedidos.
    JSONObject produto = produtosjson.getJSONObject(j);
    total = total + produto.getInt("total");
    //Se o pedido tiver apenas 1 produto e for o primeiro.
    if(j == 0 && produtosjson.length() == 1){
        cCorrenteArray.add(new CCorrente(produto.getString("nome"),
            produto.getString("quantidade"), produto.getString("preco"),
            produto.getString("total"),"Pedido: "+(i+1),"Preço Total: "+total+"€"));
    }
    //Se o produto for o primeiro e existir mais produtos no pedido.
    else if(j == 0){
        cCorrenteArray.add(new CCorrente(produto.getString("nome"),
            produto.getString("quantidade"), produto.getString("preco"),
            produto.getString("total"),"Pedido: "+(i+1),""));
    }
    //Se o produto é ultimo do pedido.
    else if(j == produtosjson.length()-1){
        cCorrenteArray.add(new CCorrente(produto.getString("nome"),
            produto.getString("quantidade"),
            produto.getString("preco").getString("total"),"","Preço Total: "+total+"€"));
    }
    //Se o produto não é o primeiro, e também último.
    else{
        cCorrenteArray.add(new CCorrente(produto.getString("nome"),
            produto.getString("quantidade"), produto.getString("preco"), produto.getString("total"),"","",""));
    }
    //Fim do for que faz a interação pelos produtos.
}
//Fim do Try.
catch (Exception e){
    Toast.makeText(this, "Link Inválido", Toast.LENGTH_LONG).show();
}
//Fim do for que faz a interação pelos pedidos.
//Referencia aos objetos da parte gráfica.
CcorrenteAdapter = new MyArrayAdapter3(ContaCorrenteActivity.this, R.layout.list_item3,
cCorrenteArray);
listview = (ListView) findViewById(R.id.listView);
//Colocação dos dados do CcorrenteAdapter, dentro da Listview da atividade.
listview.setAdapter(CcorrenteAdapter);
listview.setOnItemClickListener(new OnItemClickListener() {
    //Fim do onCreate
});
//Fim da classe Principal da atividade.

```

```

//Classe CCorrente.java responsável pelos Pedidos na atividade Conta Corrente.
package com.bonappetitapp;
public class CCorrente {
    private String name;
    private String quanti;
    private String preco;
    private String total;
    private String pedidoid;
    private String precoTotal;
    //Construtor da classe CCorrente.
    public CCorrente(String name, String quanti, String preco, String total, String pedidoid, String
    precoTotal) {
        super();
        this.name = name;
        this.quanti = quanti;
    }
}

```

```

        this.preco = preco;
        this.total = total;
        this.pedidoId = pedidoId;
        this.precoTotal = precoTotal;
    }
    //Métodos para fazer o "Get" e "Set" dos dados passados por parâmetro.
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getQuanti() {
        return quanti;
    }
    public void setQuanti(String quanti) {
        this.quanti = quanti;
    }
    public String getPreco() {
        return preco;
    }
    public void setPreco(String preco) {
        this.preco = preco;
    }
    public String getTotal() {
        return total;
    }
    public void setTotal(String total) {
        this.total = total;
    }
    public String getPedidoId() {
        return pedidoId;
    }
    public void setPedidoId(String pedidoId) {
        this.pedidoId = pedidoId;
    }
    public String getPrecoTotal() {
        return precoTotal;
    }
    public void setPrecoTotal(String precoTotal) {
        this.precoTotal = precoTotal;
    }
}
//Fim da classe CCorrent.java

```

```

//Classe MyArrayAdapter3.java responsável pelos dados introduzidos na Listview da classe Conta Corrente,
utilizando a classe CCorrent.java. Nesta classe encontra-se os métodos de adicionar a informação a Listview.
package com.bonappetitapp;
// Imports necessários as classes utilizadas
import java.util.ArrayList;
import android.app.Activity;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.TextView;
public class MyArrayAdapter3 extends ArrayAdapter<CCorrent> {
    Context context;

```

```

int layoutResourceId;
//Variaveis Locais.
ArrayList<CCorrent> ccorrentes = new ArrayList<CCorrent>();
public MyArrayAdapter3(Context context, int layoutResourceId,
    ArrayList<CCorrent> studs) {
    super(context, layoutResourceId, studs);
    this.layoutResourceId = layoutResourceId;
    this.context = context;
    this.ccorrentes = studs;
}
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    View item = convertView;
    CCorrentWrapper CCorrentWrapper = null;
    //Referência aos objetos para a parte gráfica.
    if (item == null) {
        LayoutInflator inflater = ((Activity) context).getLayoutInflater();
        item = inflater.inflate(layoutResourceId, parent, false);
        CCorrentWrapper = new CCorrentWrapper ();
        CCorrentWrapper.name = (TextView) item.findViewById(R.id.textName);
        CCorrentWrapper.quanti = (TextView) item.findViewById(R.id.textAge);
        CCorrentWrapper.preco = (TextView) item.findViewById(R.id.textAddr);
        CCorrentWrapper.total = (TextView) item.findViewById(R.id.total);
        CCorrentWrapper.pedidoId = (TextView) item.findViewById(R.id.pedidoId);
        CCorrentWrapper.precoTotal = (TextView) item.findViewById(R.id.textPedidoTotal);
        item.setTag(CCorrentWrapper);
    } else {
        CCorrentWrapper = (CCorrentWrapper) item.getTag();
    }
    //Variaveis utilizadas para saber qual a posição dos objetos.
    CCorrent corrent = ccorrentes.get(position);
    //Metodos utilizados para colocar os valores nos respetivos campos.
    CCorrentWrapper.name.setText(corrent.getName());
    CCorrentWrapper.quanti.setText("X "+ corrent.getQuanti());
    CCorrentWrapper.preco.setText(corrent.getPreco()+"€");
    CCorrentWrapper.total.setText(corrent.getTotal()+"€");
    CCorrentWrapper.pedidoId.setText(corrent.getPedidoId());
    CCorrentWrapper.precoTotal.setText(corrent.getPrecoTotal());
    return item;
}
//Classe CCorrentWrapper.
static class CCorrentWrapper {
    TextView name;
    TextView quanti;
    TextView preco;
    TextView total;
    TextView pedidoId;
    TextView precoTotal;
}
//Fim da classe MyarrayAdapter3.

//Interface gráfico da atividade Conta Corrente
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/fundo2blue"
    android:paddingBottom="@dimen/activity_vertical_margin"

```

```

    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.javacodegeeks.androidqrcodeexample.ContaCorrenteActivity" >
<ListView
    android:id="@+id/listView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_marginTop="5dp"
    android:cacheColorHint="#FFFFFF"
    android:listSelector="@android:color/transparent" />
</RelativeLayout>

```

//Interface gráfico dos objetos que se encontram dentro da Listview da atividade Conta Corrente.

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/ly23"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
<TextView
    android:id="@+id/pedidoId"
    android:layout_width="140dp"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_marginLeft="5dp"
    android:text="Pedido 1"
    android:textColor="#FFFFFF"
    android:textSize="18sp" />
<TextView
    android:id="@+id/textName"
    android:layout_width="140dp"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_marginLeft="5dp"
    android:layout_marginTop="18dp"
    android:text="Name:"
    android:textColor="#000000"
    android:textSize="16sp" />
<TextView
    android:id="@+id/textAge"
    android:layout_width="30dp"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/textName"
    android:layout_marginLeft="15dp"
    android:layout_toRightOf="@+id/textName"
    android:text="Age:"
    android:textColor="#8C9EFF"
    android:textSize="14sp" />
<TextView
    android:id="@+id/textAddr"
    android:layout_width="50dp"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/textAge"
    android:layout_marginLeft="30dp"
    android:layout_toRightOf="@+id/textAge"
    android:text="Address:"
    android:textColor="#000000"
    android:textSize="14sp" />

```

```

<TextView
    android:id="@+id/textTotal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/textAge"
    android:layout_marginLeft="50dp"
    android:layout_toRightOf="@+id/textAddr"
    android:text="Total: "
    android:textColor="#000000"
    android:textSize="14sp" />
<TextView
    android:id="@+id/total"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/textAge"
    android:layout_marginLeft="5dp"
    android:layout_toRightOf="@+id/textTotal"
    android:text="Total"
    android:textColor="#000000"
    android:textSize="14sp" />
<TextView
    android:id="@+id/textPedidoTotal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="60dp"
    android:layout_toRightOf="@+id/textTotal"
    android:text=""
    android:textColor="#FFFFFF"
    android:textSize="18sp" />
</RelativeLayout>

```

Reservar Mesa

Neste tópico vamos discutir a terceira funcionalidade da aplicação. é a opção que permite ao utilizador, dentro de uma lista de restaurantes que estejam registados na aplicação, a possibilidade de ver informações sobre os mesmos, assim como, a sua ementa. Uma das funcionalidades é também a capacidade através de uma interface da própria aplicação, o envio de um *e-mail* ao restaurante selecionado para reserva de mesa.

O objetivo desta funcionalidade é permitir ao utilizador antes de se deslocar ao respetivo restaurante para ver a ementa, ter a possibilidade de a ver atualizada, e assim ponderar a sua escolha sem correr riscos, de não gostar dos serviços prestados pelo restaurante. Para complementar foi adicionado a possibilidade de reservar mesa através dos contatos do restaurante e assim o envio de um *e-mail*, para o efeito.

Em baixo vemos um figura ilustrativa do tipo de menu utilizado para esta funcionalidade, neste caso apenas vemos um restaurante, porque a aplicação ocupa a área do

ecrã na totalidade para apenas um restaurante, mas é através de um menu deslizante, que permite a visualização de todos os outros restaurantes, apenas para isso, basta deslizar com os dedos para baixo, ou cima, para ver os outros.



Figura 18 - Ecrã da funcionalidade Reservar Mesa

Como podemos ver pela figura em cima, é apresentado o restaurante “O Faroleiro”, apresentando o nome, uma foto, tipo de restaurante, e depois a informação relevante a nível dos contatos e a sua localização. Em baixo podemos ver as duas opções que temos, primeiro a de ver a ementa desse restaurante, e segundo a de reservar a mesa.

Ver Ementa

Nesta opção o utilizador vai poder ver a ementa do restaurante selecionado, todo o processo está a reutilizar os mesmos meios do Iniciar Pedido, no entanto, há a destacar duas grandes diferenças entre elas. A primeira é que o utilizador não se encontra ainda no restaurante, logo não têm acesso ao QRCode para passar aos menus das categorias, neste caso não é necessário, visto essa situação ser apenas necessária para efetuar pedidos. Em segundo, e por consequência, o utilizador apenas pode visualizar informação sobre os produtos apenas, quando chegar ao ecrã do produto, a aplicação Android limita as opções do utilizador. Para

melhor compreensão podemos ver o aspeto do ultimo ecrã relativo a um produto quando visto pela opção Ver Ementa.

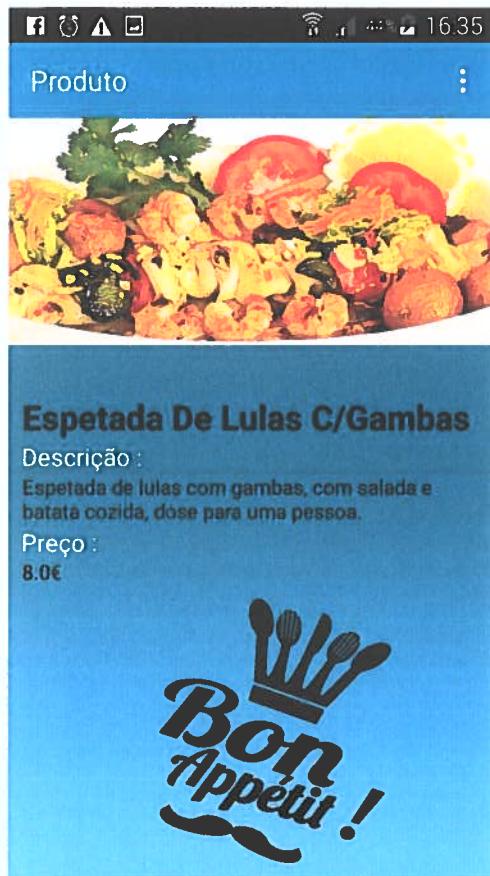


Figura 19 - Ecrã de produto, visto através da opção Ver Ementa.

Como podemos observar pela figura de cima, as opções de efetuar pedidos foi removida, partilha no Facebook, assim como, a alteração da quantidade, foi ainda alterado o fundo de ecrã por motivos estéticos da aplicação.

Reservar Mesa

Nesta opção já foi necessário a utilização de novos recursos para o efeito, anteriormente tinha sido reutilizado funcionalidades já programadas. Aqui o utilizador quando escolher a opção de Reservar Mesa, a aplicação copia automaticamente o endereço de *e-mail* do restaurante selecionado e coloca-o no seguinte ecrã para envio do *e-mail*. Neste caso então, é necessário mencionar o que permitiu a troca/partilha de informação entre Ecrãs diferentes (*Activities*). Esta técnica está a ser utilizada durante todo o projeto, sempre que seja necessário existir essa troca/partilha de informação.

Para melhor compreensão temos que explicar os dois componentes essenciais ao desenvolvimento de aplicações para dispositivos Android:

- Atividades (*Activities*) – as atividades representam os ecrãs de uma aplicação com uma interface gráfica de utilizador.
- Intenções (*Intents*) – uma *intent* é uma mensagem assíncrona que permite que uma aplicação peça funcionalidades a outros componentes do SO Android, tais como atividades, serviços ou receptores de *broadcast*. (Queirós, 2013)

Em baixo temos uma figura ilustrativa do ecrã de envio de *e-mail*.

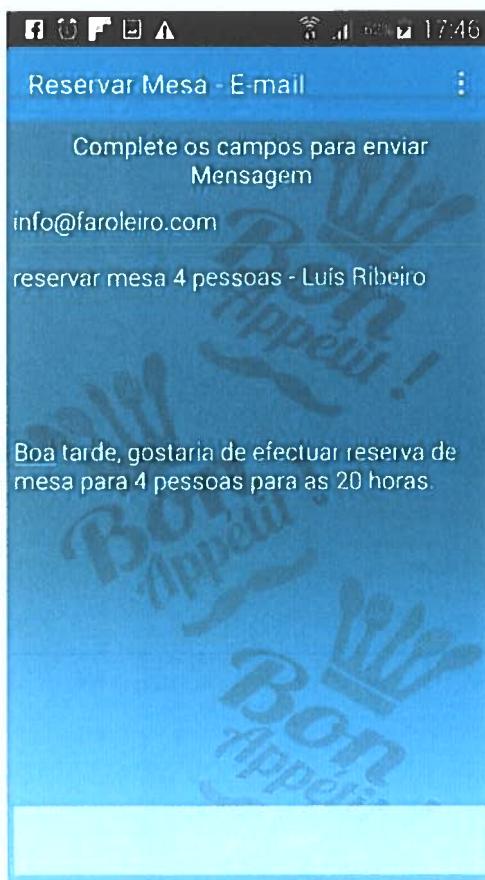


Figura 20 - Ecrã de envio de e-mail.

Neste ecrã, o endereço de correio eletrónico como mencionado anteriormente é escrito automaticamente, pelo processo do componente *Intent*. O seguinte texto é escrito pelo utilizador à sua vontade. Em baixo podemos ver então o botão de Enviar E-mail, a aplicação apresenta um outro ecrã, com as opções válidas para envio de e-mail, copiando todo o texto que o utilizador escreveu anteriormente na aplicação Android (Bom Appétit). Para melhor

compreender segue em baixo duas figuras ilustrativas, uma do menu com as opções de aplicações para envio, e outra com a própria aplicação selecionada para o envio com todo o texto.

Choose an email client from...

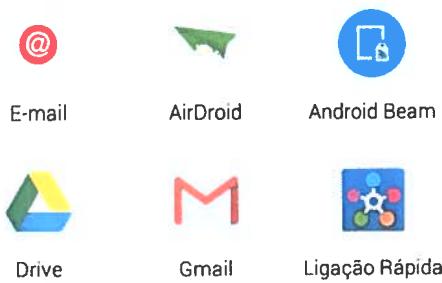


Figura 21 - Menu de escolha da aplicação interna de envio de e-mail.

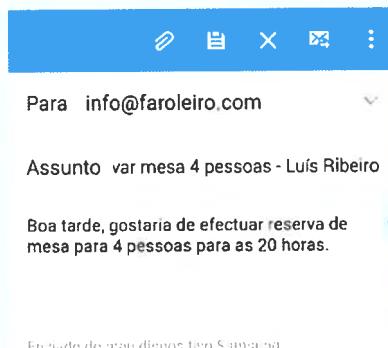


Figura 22 - Ecrã da aplicação interna do SO Android para envio de e-mail.

O código responsável pela terceira funcionalidade, em relação a atividade, a atividade Ementa Menu, a atividade Reservar Mesa – E-mail, a classe Reserva, a classe

MyArrayAdapter2, assim como, o código responsável pela interface gráfica desta atividade, é o seguinte.

```
//Código Referente a atividade Reservar Mesa da aplicação Android.
package com.bonappetitapp;
// Imports necessários as classes utilizadas
import java.io.InputStream;
import java.util.ArrayList;
import org.json.JSONArray;
import org.json.JSONObject;
import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.ImageView;
import android.widget.ListView;
import android.widget.Toast;
//Classe principal da atividade Reservar Mesa.
public class ReservasActivity extends Activity {
    //Declaração de variáveis globais.
    ListView listview;
    MyArrayAdapter2 ReservasArrayAdapter;
    ArrayList<Reserva> reservaArray = new ArrayList<Reserva>();
    //Classe onCreate que só corre uma vez no inicio da atividade.
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_reservas);
        //Tentativa de Pedido de todos os Restaurantes.
        try {
            String result = new RestAPI().getResult("https://bon-appetit-server.herokuapp.com/api/restaurantes");
            //Criação do objeto JSON com o resultado obtido do pedido http.
            JSONObject json = new JSONObject(result);
            //Criação do objeto JSONArray com todos os Restaurantes.
            JSONArray restaurantesJson = json.getJSONArray("restaurantes");
            // For responsável pela iteração dos Vários Restaurantes.
            for(int i=0; i<restaurantesJson.length(); i++){
                //Criação do objeto JSON.
                JSONObject obj = restaurantesJson.getJSONObject(i);
                //Passagem de paramentros ao construtor da classe Reserva.
                reservaArray.add(new Reserva(obj.getString("nome"), obj.getString("tipo"),
                    obj.getString("morada"), obj.getString("telefone"), obj.getString("email"),
                    obj.getString("facebook"), obj.getString("idR"),obj.getString("foto")));
            }
        } catch (Exception e){
            Toast.makeText(this, "Link Inválido", Toast.LENGTH_LONG).show();
        }
        //Referência ao objeto para a parte gráfica.
        ReservasArrayAdapter = new MyArrayAdapter2(ReservasActivity.this,
            R.layout.list_item2, reservaArray);
        listview= (ListView) findViewById(R.id.listView);
        //Colocação dos dados do ReservasArrayAdapter, dentro da Listview da atividade.
        listview.setAdapter(ReservasArrayAdapter);
```

```

//Fim do OnCreate
}//Fim da classe principal da atividade Reservar Mesa.

//Código Referente a atividade Ementa Menu da aplicação Android.
package com.bonappetitapp;
// Imports necessários as classes utilizadas.
import java.util.ArrayList;
import org.json.JSONArray;
import org.json.JSONObject;
import android.app.ListActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
//Classe Principal da atividade Ementa Menu.
public class EmentaMenuActivity extends ListActivity {
    //Declaração de variáveis globais.
    private TextView theTextView;
    public String nomeRes = "";
    public String[] idsMenus = new String[40];
    //Classe onCreate que só corre uma vez no início da atividade.
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_ementa_menu);
        //Referencia aos objetos da parte gráfica.
        theTextView = (TextView) findViewById(R.id.textView1);
        //Receber os dados enviados pela a atividade, que chamou este ecrã.
        Intent intent = getIntent();
        String idRes = intent.getExtras().getString("idRes");
        //Lista de um array de string que vai servir a lista de items.
        ArrayList<String> listItems=new ArrayList<String>();
        //Definir um adaptador de string que vai gerir os dados dentro da Listview.
        ArrayAdapter<String> adapter;
        //Referência para onde os dados vão ser carregados no Ambiente gráfico.
        adapter=new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1,listItems);
        setListAdapter(adapter);
        //Carregar o nome do restaurante, utilizando a classe RestAPI, no formato JSON.
        try
        {
            String result = new RestAPI().getResult("https://bon-appetit-
server.herokuapp.com/api/restaurantes/" + idRes);
            //Criação do objeto JSON com o resultado obtido do pedido http.
            JSONObject json = new JSONObject(result);
            //Colocar o nome do restaurante dentro da TextView.
            theTextView.setText(json.getString("nome"));
            nomeRes = json.getString("nome");
        }catch (Exception e){
            Toast.makeText(this, "Link Inválido", Toast.LENGTH_LONG).show();
        }
    }
}

```

```

        String result = new RestAPI().getResult("https://bon-appetit-
server.herokuapp.com/api/restaurantes/" + idRes + "/menus");
        //Criação do objeto JSON com o resultado obtido do pedido http.
        JSONObject json = new JSONObject(result);
        JSONArray menusJson = json.getJSONArray("menus");
        //For responsável pela iteração dos nomes dos menus, guardando o ID no array de
        idsMenus.
        for(int i=0; i<menusJson.length(); i++)
        {
            listItems.add(menusJson.getJSONObject(i).getString("nome"));
            idsMenus[i] = menusJson.getJSONObject(i).getString("idM");
        }
    } catch (Exception e){
        Toast.makeText(this, "Link Inválido", Toast.LENGTH_LONG).show();
    }
    //Fim do onCreate.
    //Método responsável pelo click de um determinado menu, para poder chamar os respetivos produtos,
    chamando a atividade ProdutosActivity.
    @Override
    protected void onListItemClick(ListView list, View v, final int position, long id) {
        super.onListItemClick(list, v, position, id);
        Intent intent = getIntent();
        String idRes = intent.getExtras().getString("idRes");
        //Chamar a nova atividade passando os dados necessários.
        String Ementa = "Ementa";
        Intent i = new Intent (this,ProdutosActivity.class);
        i.putExtra("idRestaurante", idRes );
        i.putExtra("idMenu", idsMenus[position]);
        i.putExtra("nomeRes", nomeRes);
        i.putExtra("Ementa", Ementa);
        startActivity(i);
    }
    //Fim do onListItemClick
}
// Fim da classe principal da atividade.

```

```

//Código Referente a atividade Reservar Mesa – E-mail da aplicação Android.
package com.bonappetitapp;
// Imports necessários as classes utilizadas
import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
//Classe Principal da atividade Menu.
public class ReservarMesaActivity extends Activity {
    //Declaração de variáveis globais.
    private EditText recipient;
    private EditText subject;
    private EditText body;
    //Classe onCreate que só corre uma vez no inicio da atividade.
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_reservar_mesa);

```

```

//Receber os dados enviados pela a atividade. que chamou este ecrã.
Intent intent = getIntent();
final String email = intent.getExtras().getString("email");
//Referencia aos objetos da parte gráfica.
recipient = (EditText) findViewById(R.id.recipient);
subject = (EditText) findViewById(R.id.subject);
body = (EditText) findViewById(R.id.body);
Button sendBtn = (Button) findViewById(R.id.sendEmail);
//Colocar o endereço de e-mail que foi recebido da outra atividade.
recipient.setText(email);
//Método responsável pelo click do botão Enviar E-mail.
sendBtn.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        sendEmail();
        // Depois de ser enviado o e-mail estes campos são limpos.
        recipient.setText("");
        subject.setText("");
        body.setText("");
    }
}); //Fim do setOnClickListener
} //Fim do onCreate
//Classe responsável pelo envio de e-mail.
protected void sendEmail() {
    String[] recipients = {recipient.getText().toString()};
    Intent email = new Intent(Intent.ACTION_SEND, Uri.parse("mailto:"));
    //Faz o despertar dos clientes de e-mail.
    email.setType("message/rfc822");
    email.putExtra(Intent.EXTRA_EMAIL, recipients);
    email.putExtra(Intent.EXTRA_SUBJECT, subject.getText().toString());
    email.putExtra(Intent.EXTRA_TEXT, body.getText().toString());
    try {
        // Janela de escolha de cliente de e-mail.
        startActivityForResult(Intent.createChooser(email, "Escolha um cliente de E-mail..."));
    } catch (android.content.ActivityNotFoundException ex) {
        Toast.makeText(ReservarMesaActivity.this, "Não existe nenhum cliente de E-mail instalado.", Toast.LENGTH_LONG).show();
    }
}
} // Fim da classe principal da atividade.

```

```

//Classe Reserva.java responsável pelos dados dos restaurantes na atividade Reservar Mesa.
package com.bonappetitapp;
public class Reserva {
    private String name;
    private String age;
    private String address;
    private String telefone;
    private String email;
    private String face;
    private String id;
    private String resFoto;
    //Construtor da classe Reserva.
    public Reserva(String name, String age, String address, String telefone, String email, String face, String id, String resFoto) {
        super();
        this.name = name;
        this.age = age;
        this.address = address;
        this.telefone = telefone;
    }
}

```

```

        this.email = email;
        this.face = face;
        this.id = id;
        this.resFoto = resFoto;
    }
    //Métodos para fazer o "Get" e "Set" dos dados passados por parâmetro.
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getAge() {
        return age;
    }
    public void setAge(String age) {
        this.age = age;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    public String getTelefone() {
        return telefone;
    }
    public String getEmail() {
        return email;
    }
    public String getFace() {
        return face;
    }
    public String getId() {
        return id;
    }
    public String getresFoto() {
        return resFoto;
    }
}
// Fim da classe Reserva.java

```

//Classe MyArrayAdapter2.java responsável pelos dados introduzidos na Listview da classe Reservar Mesa, utilizando a classe Reserva.java. Nesta classe encontra-se os métodos de adicionar a informação a Listview.

```

package com.bonappetitapp;
// Imports necessários as classes utilizadas
import java.io.InputStream;
import java.util.ArrayList;
import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.AsyncTask;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;

```

```

import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
//Classe onCreate que só corre uma vez no início da atividade.
public class MyArrayAdapter2 extends ArrayAdapter<Reserva> {
    //Declaração de variáveis globais.
    Context context;
    int layoutResourceId;
    ArrayList<Reserva> reservas = new ArrayList<Reserva>();
    public MyArrayAdapter2(Context context, int layoutResourceId,
        ArrayList<Reserva> studs) {
        super(context, layoutResourceId, studs);
        this.layoutResourceId = layoutResourceId;
        this.context = context;
        this.reservas = studs;
    }
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View item = convertView;
        ReservasWrapper ReservasWrapper = null;
        //Referência aos objetos para a parte gráfica.
        if (item == null) {
            LayoutInflater inflater = ((Activity) context).getLayoutInflator();
            item = inflater.inflate(layoutResourceId, parent, false);
            ReservasWrapper = new ReservasWrapper ();
            ReservasWrapper.name = (TextView) item.findViewById(R.id.textName);
            ReservasWrapper.age = (TextView) item.findViewById(R.id.textAge);
            ReservasWrapper.address = (TextView) item.findViewById(R.id.textAddr);
            ReservasWrapper.telefone = (TextView) item.findViewById(R.id.texttele);
            ReservasWrapper.email = (TextView) item.findViewById(R.id.textemail);
            ReservasWrapper.facebook = (TextView) item.findViewById(R.id.textface);
            ReservasWrapper.id = (TextView) item.findViewById(R.id.textId);
            ReservasWrapper.resFoto = (ImageView) item.findViewById(R.id.imageView1);

            ReservasWrapper.edit = (Button) item.findViewById(R.id.btnEdit);
            ReservasWrapper.delete = (Button) item.findViewById(R.id.btnDelete);
            item.setTag(StudentWrapper);
        } else {
            ReservasWrapper = (ReservasWrapper) item.getTag();
        }
        //Váriaveis utilizadas para saber qual a posição dos objetos.
        Reserva Reservar = reservas.get(position);
        //Metodos utilizados para colocar os valores nos respetivos campos.
        ReservasWrapper.name.setText(Reservar.getName());
        ReservasWrapper.age.setText(Reservar.getAge());
        ReservasWrapper.address.setText(Reservar.getAddress());
        ReservasWrapper.telefone.setText(Reservar.getTelefone());
        ReservasWrapper.email.setText(Reservar.getEmail());
        ReservasWrapper.facebook.setText(Reservar.getFace());
        ReservasWrapper.id.setText(Reservar.getId());
        //Carregar as fotos dos restaurantes para dentro da ImageView.
        new DownloadImageTask((ImageView)
            ReservasWrapper.resFoto).execute(Reservar.getresFoto());
        //Método responsável pelo click do botão Ver Ementa.
        ReservasWrapper.edit.setOnClickListener(new OnClickListener() {
            ReservasWrapper pa;
            @Override
            public void onClick(View v) {
                //Chamada da nova atividade EmenteMenu.

```

```

        Intent t = new Intent (getContext(),EmentaMenuActivity.class);
        t.putExtra("idRes", (pa.id.getText()).toString());
        context.startActivity(t);
    }

    private OnClickListener init(ReservasWrapper pa){
        this.pa = pa;
        return this;
    }

    }.init(ReservasWrapper)); // Fim do setOnClickListener do botão Ver Ementa.
    //Método responsável pelo click do botão Reservar Mesa.
    ReservasWrapper.delete.setOnClickListener(new OnClickListener() {
        ReservasWrapper pa;
        @Override
        public void onClick(View v) {
            //Chamada da nova atividade Reservar Mesa – Email..
            Intent k = new Intent (getContext(),ReservarMesaActivity.class);
            k.putExtra("email", (pa.email.getText()).toString());
            context.startActivity(k);
        }

        private OnClickListener init(ReservasWrapper pa){
            this.pa = pa;
            return this;
        }

        }.init(ReservasWrapper)); // Fim do setOnClickListener do botão Reservar Mesa.
        return item;
    }//Fim da classe MyArrayAdapter2.
//Classe ReservasWrapper
static class ReservasWrapper {
    TextView name;
    TextView age;
    TextView address;
    TextView telefone;
    TextView email;
    TextView facebook;
    TextView id;
    ImageView resFoto;
    Button edit;
    Button delete;
} //Fim da classe ReservasWrapper
//Classe responsável por transformar uma foto no formato URL em bitmap. para mostrar a foto do Restaurante. Este método está a ser chamado quando é carregado os dados dentro da ListView.
public class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    ImageView bmImage;
    public DownloadImageTask(ImageView bmImage) {
        this.bmImage = bmImage;
    }
    //Classe executada em background por questões de performance. para não subcarregar a atividade principal.
    protected Bitmap doInBackground(String... urls) {
        String urldisplay = urls[0];
        Bitmap mIcon11 = null;
        try {
            InputStream in = new java.net.URL(urldisplay).openStream();
            mIcon11 = BitmapFactory.decodeStream(in);
        } catch (Exception e) {
            Log.e("Error", e.getMessage());
            e.printStackTrace();
        }
        //Retornar a imagem em bitmap.
        return mIcon11;
    }
}

```

```

    }

    //Método para fazer o set da imagem.
    protected void onPostExecute(Bitmap result) {
        bmImage.setImageBitmap(result);
    }

    //Fim da classe DownloadImageTask
}

```

// Interface gráfico da atividade Reservar Mesa.

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/fundo2blue"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.javacodegeeks.androidqrcodeexample.ReservasActivity" >
    <ListView
        android:id="@+id/listView"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_marginTop="5dp"
        android:cacheColorHint="#FFFFFF"
        android:divider="#80000000"
        android:dividerHeight="5dp"
        android:listSelector="@android:color/transparent" />
</RelativeLayout>

```

//Interface gráfico dos objetos que se encontram dentro da Listview da atividade Reservar Mesa.

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="4dp" >
    <TextView
        android:id="@+id/textId"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:visibility="invisible" />
    <TextView
        android:id="@+id/textName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="5dp"
        android:text="Name:"
        android:textColor="#FFFFFF"
        android:textSize="30dp" />
    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textName"
        android:scaleType="fitStart" />
    <TextView

```

```
    android:id="@+id/tipo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/imageView1"
    android:layout_marginBottom="7dp"
    android:layout_marginTop="245dp"
    android:text="Tipo:"
    android:textColor="#FFFFFF"
    android:textSize="21sp" />
<TextView
    android:id="@+id/textAge"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/tipo"
    android:layout_marginLeft="5dp"
    android:layout_toRightOf="@+id/tipo"
    android:textColor="#000000"
    android:textSize="17sp" />
<TextView
    android:id="@+id/morada"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/imageView1"
    android:layout_below="@+id/tipo"
    android:layout_marginBottom="7dp"
    android:text="Morada:"
    android:textColor="#FFFFFF"
    android:textSize="21sp" />
<TextView
    android:id="@+id/textAddr"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/morada"
    android:layout_marginLeft="5dp"
    android:layout_toRightOf="@+id/morada"
    android:textColor="#000000"
    android:textSize="17sp" />
<TextView
    android:id="@+id/telefone"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/imageView1"
    android:layout_below="@+id/morada"
    android:layout_marginBottom="7dp"
    android:text="Telefone:"
    android:textColor="#FFFFFF"
    android:textSize="21sp" />
<TextView
    android:id="@+id/texttele"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/telefone"
    android:layout_marginLeft="5dp"
    android:layout_toRightOf="@+id/telefone"
    android:textColor="#000000"
    android:textSize="17sp" />
<TextView
    android:id="@+id/email"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```

    android:layout_alignLeft="@+id/imageView1"
    android:layout_below="@+id/telefone"
    android:layout_marginBottom="7dp"
    android:text="Email:"
    android:textColor="#FFFFFF"
    android:textSize="21sp" />
<TextView
    android:id="@+id/textemail"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/email"
    android:layout_marginLeft="5dp"
    android:layout_toRightOf="@+id/email"
    android:textColor="#000000"
    android:textSize="17sp" />
<TextView
    android:id="@+id/facebook"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/imageView1"
    android:layout_below="@+id/email"
    android:text="Facebook:"
    android:textColor="#FFFFFF"
    android:textSize="21sp" />
<TextView
    android:id="@+id/textface"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/facebook"
    android:layout_marginLeft="5dp"
    android:layout_toRightOf="@+id/facebook"
    android:textColor="#000000"
    android:textSize="17sp" />
<Button
    android:id="@+id/btnEdit"
    android:layout_width="160dp"
    android:layout_height="50dp"
    android:layout_alignLeft="@+id/imageView1"
    android:layout_marginTop="480dp"
    android:alpha="0.6"
    android:background="#CFD8DC"
    android:focusable="false"
    android:focusableInTouchMode="false"
    android:text="Ver Ementa"
    android:textColor="#FFFFFF" />
<Button
    android:id="@+id/btnDelete"
    android:layout_width="160dp"
    android:layout_height="50dp"
    android:layout_alignBaseline="@+id/btnEdit"
    android:layout_alignRight="@+id/imageView1"
    android:alpha="0.6"
    android:background="#CFD8DC"
    android:focusable="false"
    android:focusableInTouchMode="false"
    android:text="Reservar Mesa"
    android:textColor="#FFFFFF" />
</RelativeLayout>

```

```

//Interface gráfico da atividade Ementa Menu.
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/fundo2blue"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.javacodegeeks.androidqrcodeexample.EmentaMenuActivity" >
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:padding="10dp"
        android:text="TextView"
        android:textColor="#FFFFFF"
        android:textSize="20dp" />
    <ListView
        android:id="@+id/list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView1"
        android:layout_centerHorizontal="true"
        android:alpha="0.6"
        android:background="#CFD8DC"
        android:divider="#80000000"
        android:dividerHeight="2.0sp"
        android:textColor="#DCE775"
        android:textSize="25sp" >
    </ListView>
</RelativeLayout>

```

```

//Interface gráfico da atividade Reservar Mesa – E-mail.
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/fundo2blue"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.javacodegeeks.androidqrcodeexample.ReservarMesaActivity" >
    <TextView
        android:id="@+id/text1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        android:gravity="center"
        android:text="Complete os campos para enviar Mensagem"
        android:textAppearance="?android:attr/textAppearanceMedium" />
    <EditText
        android:id="@+id/recipient"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/text1" 

```

```

    android:layout_margin="5dp"
    android:ems="10"
    android:hint="Para"
    android:inputType="textEmailAddress" />
<EditText
    android:id="@+id/subject"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/recipient"
    android:layout_margin="5dp"
    android:ems="10"
    android:hint="Assunto" />
<EditText
    android:id="@+id/body"
    android:layout_width="fill_parent"
    android:layout_height="250dp"
    android:layout_below="@+id/subject"
    android:layout_margin="5dp"
    android:ems="10"
    android:hint="Mensagem" />
<Button
    android:id="@+id/sendEmail"
    android:layout_width="fill_parent"
    android:layout_height="50dp"
    android:layout_alignParentBottom="true"
    android:layout_margin="5dp"
    android:alpha="0.6"
    android:background="#CFD8DC"
    android:text="Enviar E-mail"
    android:textColor="#FFFFFF" />
</RelativeLayout>

```

Perto de si

Depois de verificado pormenorizado as três funcionalidades anteriores, resta-nos falar da funcionalidade “Perto de Si”, aqui vamos utilizar a aplicação para procurar pontos de interesse através da nossa localização atual, recorrendo ao Google MAPS que nos fornecerá informação sobre os mesmos. Esta funcionalidade foi idealizada como complemento a funcionalidade principal da aplicação, e por isso, é que têm a opção de por exemplo mostrar as caixas de multibanco (ATM) perto de nós, entre outros pontos de interesse ou uteis, tais como hospitais, polícia, bancos ou paragens de autocarros. Em baixo podemos observar uma figura ilustrativa do primeiro ecrã.

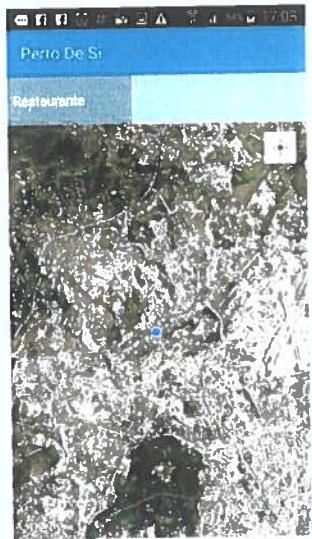


Figura 23 - Ecrã inicial da funcionalidade Perto de si.

Como podemos observar pela figura em cima, temos no canto superior esquerdo as opções de pesquisa dos pontos de interesse, neste caso selecionado Restaurante, no centro o ponto azul indica-nos a nossa posição atual. Ao ser acionado o botão de Procurar irá aparecer os pontos de interesse marcados por um sinalizador a vermelho, indicando alguma informação útil tais como nome, morada, e uma breve descrição, podendo depois a posteriora indicar-nos o caminho para lá chegar quer de veículo motorizado ou a pé. Em baixo encontra-se uma figura ilustrativa das opções mencionadas anteriormente.



Figura 24 - Restaurantes referenciados no mapa.

Para obter esta funcionalidade há que realçar vários fatores que são necessários, para ser possível a utilização da API Google MAPS na aplicação. Foi necessário primeiro importar a biblioteca Google Play Services²¹ para o nosso projeto, depois registar a aplicação no site da Google²² para obter credenciais de autenticação para usar a funcionalidade MAPS.

Em relação ao código necessário, há a destacar a alteração feita no ficheiro de manifesto da aplicação que foi o seguinte.

Quadro 3 - Código com as alterações efetuadas ao ficheiro manifesto.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

<meta-data
    android:name="com.google.android.gms.version" android:value="@integer/google_play_services_version" />

<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="AIzaSyCqCMVV8zMYWc8l-oskI-5XWFoDtEbjoc" />
```

Para obter este comportamento foi necessário o código seguinte.

```
//Código Referente a atividade Perto de si da aplicação Android.
package com.bonappetitapp;
// Imports necessários as classes utilizadas
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.HashMap;
import java.util.List;
import org.json.JSONObject;
import com.javacodegeeks.androidqrcodeexample.PlaceJSONParser;
import com.javacodegeeks.androidqrcodeexample.R;
import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.GooglePlayServicesUtil;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.MapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;
```

²¹ Referência para instalação e configuração - <https://developers.google.com/android/guides/setup>.

²² Site – <https://code.google.com/apis/console>.

```

import android.app.Dialog;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.Spinner;
//Classe Principal da atividade Perto de Si.
public class LocaisActivity extends FragmentActivity implements LocationListener {
//Declaração de variáveis Globais.
    GoogleMap mGoogleMap;
    Spinner mSprPlaceType;
    String[] mPlaceType=null;
    String[] mPlaceTypeName=null;
    double mLatitude=0;
    double mLongitude=0;
    //Classe onCreate que só corre uma vez no inicio da atividade.
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_locais);
    //Array com os tipos de Lugares.
    mPlaceType = getResources().getStringArray(R.array.place_type);
    //Array com os nomes dos tipos de lugares.
    mPlaceTypeName = getResources().getStringArray(R.array.place_type_name);
    //Criar uma array adapter com um array de tipos de lugares para popular o Spinner.
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_spinner_dropdown_item, mPlaceTypeName);
    //Referência ao Spinner na parte gráfica.
    mSprPlaceType = (Spinner) findViewById(R.id.spr_place_type);
    //Configurar o adapter no objeto Spinner para os tipos de lugar.
    mSprPlaceType.setAdapter(adapter);
    Button btnFind;
    //Referência ao botão procurar.
    btnFind = (Button) findViewById(R.id.btn_find);
    //Obter o estado de disponibilidade do Google Play.
    int status = GooglePlayServicesUtil.isGooglePlayServicesAvailable(getApplicationContext());
    if(status!=ConnectionResult.SUCCESS){ // Caso o Google Play Services não estiver disponível.
        int requestCode = 10;
        Dialog dialog = GooglePlayServicesUtil.getErrorDialog(status, this, requestCode);
        dialog.show();
    }else { // Caso o Google Play Services estiver disponível.
        //Obter o Google map e referêncialo para dentro de um fragmento.
        mGoogleMap = ((MapFragment) getSupportFragmentManager().findFragmentById(R.id.map))
            .getMap();
        //Permitir a minha localização no Google map.
        mGoogleMap.setMyLocationEnabled(true);
        mGoogleMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
        //Obter o objeto LocationManager através do serviço de Sistema Location_Service.
        LocationManager locationManager = (LocationManager) getSystemService(LOCATION_SERVICE);
        //Criação de um objeto de critério para receber o provider.
    }
}

```

```

Criteria criteria = new Criteria();
//Obter o nome do melhor provider.
String provider = locationManager.getBestProvider(criteria, true);
//Obter a localização atual através do GPS.
Location location = locationManager.getLastKnownLocation(provider);
if(location!=null){
    onLocationChanged(location);
}
locationManager.requestLocationUpdates(provider, 20000, 0, this);
//Método responsável pelo click do botão procurar.
btnFind.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        int selectedPosition = mSprPlaceType.getSelectedItemPosition();
        String type = mPlaceType[selectedPosition];
        StringBuilder sb = new
            StringBuilder("https://maps.googleapis.com/maps/api/place/nearbysearch/json?");
        sb.append("location=" + mLatitude + "," + mLongitude);
        sb.append("&radius=5000");
        sb.append("&types=" + type);
        sb.append("&sensor=true");
        sb.append("&key=AIzaSyAMXyxtN9fhO8j9NkIgf_-l-tpAZmARAKo");
        //Cria uma nova tarefa para o download dos dados JSON.
        PlacesTask placesTask = new PlacesTask();
        //Invoca o método "doInBackground()" da classe PlaceTask
        placesTask.execute(sb.toString());
    }
});
}
//Fim do onCreate
//Método para fazer download de um url de dados no formato JSON.
private String downloadUrl(String strUrl) throws IOException{
    String data = "";
    InputStream iStream = null;
    HttpURLConnection urlConnection = null;
    try{
        URL url = new URL(strUrl);
        //Criação de uma ligação http para comunicar com o URL.
        urlConnection = (HttpURLConnection) url.openConnection();
        //Conecta com o URL.
        urlConnection.connect();
        // Lê os dados do URL.
        iStream = urlConnection.getInputStream();
        BufferedReader br = new BufferedReader(new InputStreamReader(iStream));
        StringBuffer sb = new StringBuffer();
        String line = "";
        while( ( line = br.readLine()) != null){
            sb.append(line);
        }
        data = sb.toString();
        br.close();
    }catch(Exception e){
        Log.d("Exception while downloading url", e.toString());
    }finally{
        iStream.close();
        urlConnection.disconnect();
    }
    return data;
}
//Fim do downloadUrl

```

```

//Classe Responsável pelo download do Google Places.
public class PlacesTask extends AsyncTask<String, Integer, String>{
    String data = null;
    // Invocado pelo método execute() deste objeto.
    @Override
    protected String doInBackground(String... url) {
        try{
            data = downloadUrl(url[0]);
        }catch(Exception e){
            Log.d("Background Task",e.toString());
        }
        return data;
    }
    // Executado depois do método doInBackground() ter terminado.
    @Override
    protected void onPostExecute(String result){
        ParserTask parserTask = new ParserTask();
        //O Começo do Parsing do Google places no formato JSON.
        //Invoca o método "doInBackground()" da classe ParseTask.
        parserTask.execute(result);
    }
    //Fim do onPostExecute
}//Fim do PlacesTask
//A classe que faz o parse do Google Places no formato JSON.
public class ParserTask extends AsyncTask<String, Integer, List<HashMap<String, String>>>{
    JSONObject jObject;
    // Invocado pelo método execute() deste objeto.
    @Override
    protected List<HashMap<String, String>> doInBackground(String... jsonData) {
        List<HashMap<String, String>> places = null;
        PlaceJSONParser placeJsonParser = new PlaceJSONParser();
        try{
            jObject = new JSONObject(jsonData[0]);
            //Obter os dados como uma contrução de uma lista.
            places = placeJsonParser.parse(jObject);
        }catch(Exception e){
            Log.d("Exception",e.toString());
        }
        return places;
    }
    // Executado depois do método doInBackground() ter terminado.
    @Override
    protected void onPostExecute(List<HashMap<String, String>> list){
        //Limpare todos os sinalizadores no mapa.
        mGoogleMap.clear();
        for(int i=0;i<list.size();i++){
            //Criar um sinalizador.
            MarkerOptions markerOptions = new MarkerOptions();
            //Obter um local através da lista de locais.
            HashMap<String, String> hmPlace = list.get(i);
            //Obter a latitude de um local.
            double lat = Double.parseDouble(hmPlace.get("lat"));
            //Obter a longitude de um local.
            double lng = Double.parseDouble(hmPlace.get("lng"));
            //Obter o nome.
            String name = hmPlace.get("place_name");
            //Obter nos arredores (perto)
            String vicinity = hmPlace.get("vicinity");
            LatLng latLng = new LatLng(lat, lng);
            //Marcar a posição do sinalizador.
            markerOptions.position(latLng);
        }
    }
}

```

```

    //Colocar o nome do sinalizador, que é mostrado quando se clica nele.
    markerOptions.title(name + " : " + vicinity);
    //Colocar um sinalizador a onde é clicado.
    mGoogleMap.addMarker(markerOptions);
}

}

//Classe responsável pela atualização da mudança de localização.
@Override
public void onLocationChanged(Location location) {
    mLatitude = location.getLatitude();
    mLongitude = location.getLongitude();
    LatLng latLng = new LatLng(mLatitude, mLongitude);
    mGoogleMap.moveCamera(CameraUpdateFactory.newLatLng(latLng));
    mGoogleMap.animateCamera(CameraUpdateFactory.zoomTo(12));
}

//Construtores obrigatórios, mesmo sem utilização.
@Override
public void onProviderDisabled(String provider) {
    // TODO Auto-generated method stub
}

@Override
public void onProviderEnabled(String provider) {
    // TODO Auto-generated method stub
}

@Override
public void onStatusChanged(String provider, int status, Bundle extras) {
    // TODO Auto-generated method stub
}

//Fim da classe Principal da Atividade.

//Classe PlaceJSONParser que é responsável pela recolha de informação acerca dos locais da Google Places no formato JSON.
package com.bonappetitapp;
// Imports necessários as classes utilizadas
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
//Classe Principal.
public class PlaceJSONParser {
    //Recebe um JSONObject e retorna uma lista.
    public List<HashMap<String, String>> parse(JSONObject jObject){
        JSONArray jPlaces = null;
        try {
            //Reenvia todos os elementos dentro do array 'places'.
            jPlaces = jObject.getJSONArray("results");
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }

    //Invoca o método getPlaces com um array de objeto JSON a onde cada JSON objeto representa um local.
    return getPlaces(jPlaces);
}

private List<HashMap<String, String>> getPlaces(JSONArray jPlaces){
    int placesCount = jPlaces.length();
    List<HashMap<String, String>> placesList = new ArrayList<HashMap<String, String>>();
    HashMap<String, String> place = null;
}

```

```

//Toma cada lugar, analisa e adiciona à lista de objeto
for(int i=0; i<placesCount;i++){
    try {
        //Chama o método getPlace com um objeto JSON do Local para analisalo.
        place = getPlace((JSONObject)jPlaces.get(i));
        placesList.add(place);
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
return placesList;
}

//Parsing do objeto JSON Local.
private HashMap<String, String> getPlace(JSONObject jPlace){
    HashMap<String, String> place = new HashMap<String, String>();
    String placeName = "-NA-";
    String vicinity="-NA-";
    String latitude="";
    String longitude="";
    try {
        //Estrair o nome do local, se existir.
        if(!jPlace.isNull("name")){
            placeName = jPlace.getString("name");
        }
        //Estrair locais nos arredores, se existir.
        if(!jPlace.isNull("vicinity")){
            vicinity = jPlace.getString("vicinity");
        }
        latitude = jPlace.getJSONObject("geometry").getJSONObject("location").getString("lat");
        longitude = jPlace.getJSONObject("geometry").getJSONObject("location").getString("lng");
        place.put("place_name", placeName);
        place.put("vicinity", vicinity);
        place.put("lat", latitude);
        place.put("lng", longitude);
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
return place;
}

//Fim da classe PlaceJSONParser.

```

```

//Interface gráfico da atividade Perto de Si.
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/fundo3blue"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.javacodegeeks.androidqrcoodeexample.LocaisActivity" >
    <Spinner
        android:id="@+id/spr_place_type"
        android:layout_width="150dp"
        android:layout_height="60dp"
        android:layout_alignParentTop="true"
        android:background="#78909C"
        android:textColor="#FFFFFF" />

```

```
<Button  
    android:id="@+id/btn_find"  
    android:layout_width="match_parent"  
    android:layout_height="60dp"  
    android:layout_alignParentTop="true"  
    android:layout_toRightOf="@+id/spr_place_type"  
    android:alpha="0.8"  
    android:background="#CFD8DC"  
    android:text="@string/str_btn_find"  
    android:textColor="#FFFFFF"  
    android:textSize="18sp" />  
<fragment  
    android:id="@+id/map"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_below="@+id/spr_place_type"  
    class="com.google.android.gms.maps.MapFragment" />  
</RelativeLayout>
```

Aplicação Servidor

Introdução

Neste capítulo vamos focar todas as atenções, no servidor, que está alojado numa plataforma “*Cloud*”, e é responsável pela disponibilização de toda a informação que é fornecida ao utilizador na aplicação que foi demonstrada no capítulo anterior, para a sua utilização, quer a nível de efetuar os pedidos, quer a nível da informação acerca dos restaurantes. Irá ser abordado todas as tecnologias usadas, assim como, a sua utilização. Vai-se demonstrar através de imagens ilustrativas algumas configurações do próprio servidor, assim como, a estrutura aplicacional que está responsável pelas comunicações entre o servidor, e a aplicação no Android.

Cloud Computing

Cloud computing podemos afirmar que é um conceito, e não uma tecnologia, que está a revolucionar a forma como as pessoas e as empresas utilizam os recursos disponibilizados pelas TI. É baseado num modelo que permite o acesso ubíquo, conveniente e a pedido através da rede, a um conjunto de recursos de computação partilhados (redes, servidores, armazenamento, aplicações, serviços, etc.), que podem ser rapidamente aprovisionados ou libertados, com um mínimo de esforço e sem interação com o fornecedor. (Ferreira, 2015)

Com isto, temos o propósito da utilização de um servidor para alojar a aplicação, através de um fornecedor de PaaS (Platform as a Service)²³. Este fornecedor têm a capacidade de fornecer ao cliente/utilizador e a possibilidade de colocar na *cloud* as aplicações criadas ou adquiridas pelo utilizador, que usam linguagens de programação, bibliotecas de funções (*libraries*), serviços e ferramentas suportadas pelo fornecedor. Neste caso, o utilizador não gera nem controla a infraestrutura de *cloud* subjacente, ou seja, a rede, servidores, sistemas operativos, armazenamento, mas têm controlo sobre as aplicações instaladas apenas por si. Este modelo é particularmente favorável no caso de sermos programadores e existir a necessidade de ter uma aplicação disponível na internet, sem nos preocuparmos com a infraestrutura responsável por isso. Na escolha do fornecedor de PaaS, temos antes que verificar o seu custo de serviço, e as linguagens de programação que ele permite, através das suas bibliotecas.

²³ Mais informação acerca de PaaS - http://www.microsoft.com/industry/government/guides/cloud_computing/5-PaaS.aspx.

Tendo em conta alguns dos melhores fornecedores do momento, e depois de alguma ponderação, devidos as duas características mencionadas anteriormente, foi escolhido o fornecedor Heroku²⁴, tendo recaído a principal razão a sua escolha o facto de ser grátis, para a utilização a que a aplicação necessita. (Ferreira, 2015)

Heroku

Este foi o fornecedor de serviço, escolhido pela razão anteriormente mencionada, mas também por permitir aplicações na linguagem escolhida, neste caso, Ruby²⁵ e Rails²⁶. Atualmente é um dos fornecedores mais populares, uma das razões é por ser também um dos mais antigos, um dos seus maiores clientes é o serviço DropBox²⁷, que consiste num serviço de alojamento. Para a utilização deste serviço foi necessário instalar alguns programas, que permitissem o desenvolvimento da aplicação e configuração do servidor, os programas são:

- Heroku Toolbelt – é o programa do fornecedor Heroku responsável pela criação, manutenção e comunicação com o servidor a onde se encontra a nossa aplicação. Este programa pode ser descarregado em: <https://toolbelt.heroku.com>.
- Git – é o programa responsável pelo controlo de versões da aplicação e pelo “Deploy” dos dados, para o servidor do fornecedor. Este Programa pode ser descarregado em: <https://git-scm.com/downloads>.

Como mencionado anteriormente a aplicação foi desenvolvida na linguagem Ruby, para isso, foi necessário instalar a sua biblioteca, assim como, um ambiente de desenvolvimento (IDE) para efetuar toda a programação relativa ao servidor. O IDE escolhido foi o RubyMine²⁸.

Estrutura da aplicação

Depois de ter sido escolhido o fornecedor do serviço, que permitia o alojamento da aplicação através da internet, e dos programas necessários para existir essa comunicação entre o desenvolvimento e a plataforma “Cloud”, chegamos então ao ambiente de desenvolvimento RubyMine que é a onde a aplicação e configuração do servidor irá ser propriamente feita. Com isto foi necessário antes a instalação da sua biblioteca RailsInstaller que pode ser descarregada

²⁴ A plataforma Heroku está disponível em <http://www.heroku.com>.

²⁵ Para mais informação acerca de linguagem Ruby visitar - <https://www.ruby-lang.org/pt/>.

²⁶ Para mais informação acerca de Ruby on Rails visitar - <http://rubyonrails.org/>.

²⁷ Mais informação em - <https://www.dropbox.com/>.

²⁸ Mais informação e download em - <https://www.jetbrains.com/ruby/>.

em: <http://railsinstaller.org/en>. Na figura que se encontra em baixo, podemos ver como é o RubyMine e a sua estrutura.

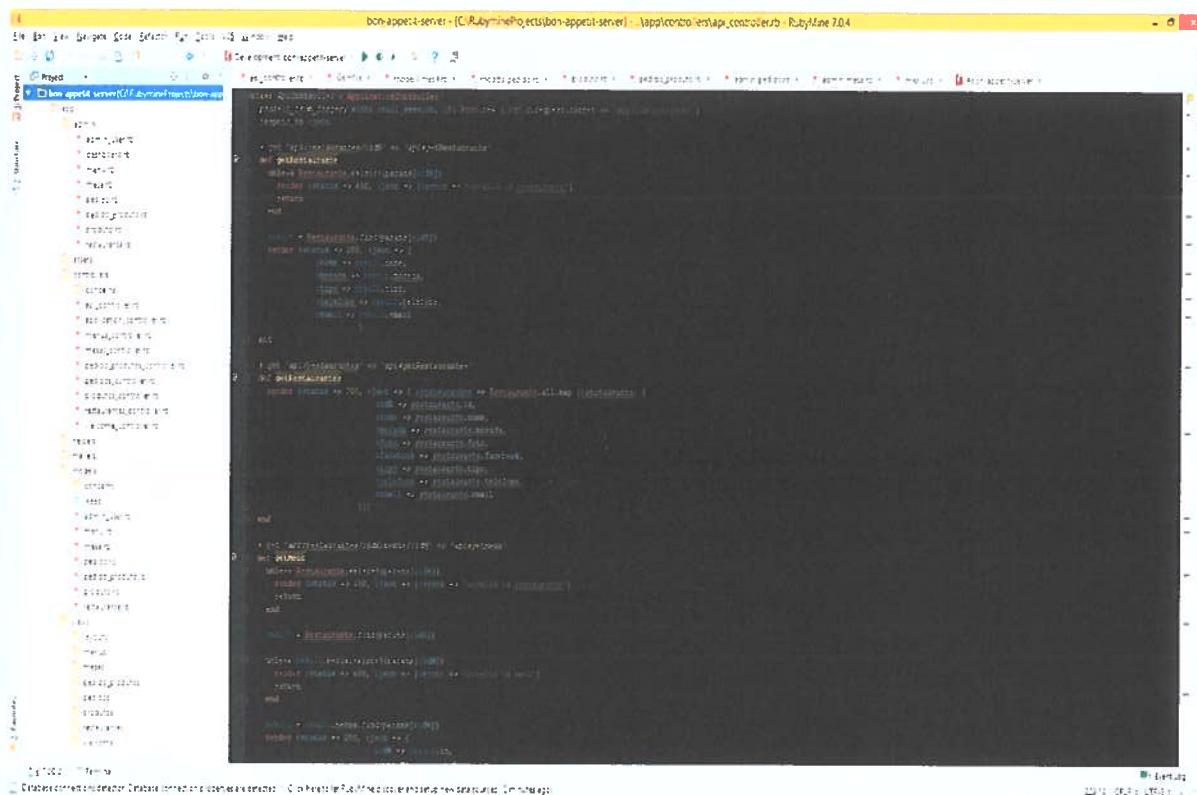


Figura 25 - IDE RubyMine.

Através desta interface foi possível a realização da configuração do servidor para ostentar a base de dados da aplicação, assim como, o controlador responsável pelas operações efetuadas na mesma, e toda a configuração necessária para o seu uso. Através do IDE, foi também possível ter um ambiente de desenvolvimento, tendo uma base de dados local, com isto permitindo efetuar as alterações sem ter que alterar o projeto no servidor *online* e assim evitar os constrangimentos de estar a efetuar operações que levassem a rotura da aplicação propriamente real, e assim podendo efetuar testes de diagnóstico antes de fazer o “*deploy*” dos dados para o ambiente de produção. Esta foi uma das mais-valias verificadas da utilização deste ambiente de desenvolvimento.

Como verificado, todas as alterações e configurações eram efetuadas localmente, e após ser efetuado os testes, era necessário o envio do projeto para o servidor alojado pelo fornecedor do serviço, Heroku. Neste caso era então utilizado o programa anteriormente mencionado, o

Git, para o envio do projeto, que era necessário recorrer a três linhas de comando que são as seguintes:

- git add .
- git commit –am “Alterações efetuadas na BD – Versão 1”
- git push heroku master

Consoante o efetuado, correspondia ao tempo demorado pelo processo em geral, após a mensagem de sucesso, todas as alterações que tivessem sido efetuadas, teriam impacto na aplicação do Android.

Para falar na configuração do servidor, vamos primeiro abordar a estrutura do mesmo, ou seja, como está feita a organização da aplicação servidor e os seus pontos mais importantes. A estrutura da aplicação Servidor está organizada da seguinte forma, como demonstra a figura em baixo.



Figura 26 - Estrutura da aplicação servidor.

Neste caso, temos que destacar vários pontos, mas começando de cima para baixo, são os seguintes:

- App – é aqui que se encontra as pastas mais importantes, é onde estão as pastas referentes a programação de toda a aplicação a nível das opções acedidas pela aplicação no Android.
- Admin – esta pasta é referente a um *plugin* instalado, o *Active Admin*²⁹, que vai fornecer a capacidade de *BackOffice* para a gestão dos dados da aplicação Android. Este tema será abordado no próximo capítulo do trabalho.
- Controllers – é nesta pasta que se encontra o controlador, ou seja, o ficheiro de configuração (*api_controller.rb*) a onde se localiza toda a programação referente aos pedidos que podem ser efetuados sobre a base de dados, através dos métodos elaborados, como por exemplo: *api#getMenus*³⁰ – que nos devolve todos os menus de categorias referentes a um restaurante. É sem dúvida um dos ficheiros mais importantes em toda a configuração da aplicação servidor.
- Models – nesta pasta encontra-se a base de dados, é a onde está a ser efetuado as configurações em relação as tabelas de base dados que a aplicação Android necessita. É aqui também que está a ser feito todo o controlo de inserção dos dados na base de dados, querendo isto dizer, que as regras que ditam por exemplo que tipos de dados podem ser inseridos nos campos para o efeito, dentro do *backoffice*, através do *Active Admin*.
- Views – esta pasta é onde se localiza os ficheiros responsáveis pela renderização dos dados, responsável pela parte gráfica.
- Config – dentro desta pasta encontra-se outro ficheiro de configuração dos mais importantes para a aplicação, que é o ficheiro de rotas (*routes.rb*). Neste ficheiro encontra-se a configuração das rotas disponíveis, para os pedidos HTTP, feitos pela aplicação Android, indicando que tipo de pedido é: *Get*, *Set*, *Post*, *Delete* ou *update*, e também o respetivo caminho (*Link*) assim como o método correspondente para esse pedido, que se encontra configurado no ficheiro *api_controller.rb* anteriormente mencionado.

²⁹ Para mais informação visitar - <http://activeadmin.info/>.

³⁰ Nome de um dos métodos utilizados.

- Public – nesta pasta encontram-se todas as fotos, que são apresentadas pela aplicação Android, dos restaurantes ou dos produtos.

Ruby on Rails

Ruby on Rails (ou simplesmente Rails) é uma *framework* de desenvolvimento web escrito na linguagem de programação Ruby. Desde sua estreia em 2004, Ruby on Rails rapidamente se tornou uma das ferramentas mais poderosas e populares para a construção de aplicações web dinâmicas. Rails é usado por empresas tão diversas como a Airbnb, Basecamp, Disney, GitHub, Hulu, Kickstarter, Shopify, Twiter, entre outras. Além disso, muitas empresas de desenvolvimento web especializaram-se em Rails, como ENTP, Thoughtbot, Pivotal Labs, Hashrocket, e HappyFunCopr, além de inúmeros consultores independentes, formadores e prestadores de serviços na área das TI.

O que faz Rails ser tão bom? Em primeiro lugar, Ruby on Rails é cem por cento *open-source*³¹, disponível sob a licença permissiva do MIT, e como resultado, não tem qualquer custo associado, para fazer o *download* ou usar. Rails também deve muito de seu sucesso ao seu design elegante e compacto, explorando a maleabilidade da linguagem subjacente, o Ruby. Rails efetivamente cria uma linguagem específica de domínio, para escrever aplicações web, como consequência, muitas tarefas de programação web comuns, tais como a geração de HTML, construção de modelos de dados, ou elaborar rotas de *Link's*, são agora mais fácil com Rails, e o código da aplicação é mais conciso e de fácil leitura. (Hartl, 2015) (Metz, 2012)

Alem dos métodos de leitura de livros, consulta de fóruns, páginas de internet referentes a linguagem de programação Ruby on Rails, tenho que destacar um sítio na internet que foi fundamental para a compreensão da linguagem em questão através de vídeos aulas (em inglês), sendo esses mesmos vídeos em formato evolutivo e conciso através do seu autor. O *link* da pagina é o seguinte: <http://railsforzombies.org/>.

A que reafirmar, que toda a programação em relação a aplicação Servidor, foi elaborada em Ruby on Rails, excepto a programação da base de dados que foi feita em SQLite 3, como vamos evidenciar no tópico seguinte.

³¹ Palavra em inglês que refereência ou quer dizer – Livre, aberto a todos.

Base de Dados

Depois de ter sido abordado a estrutura que está responsável pela aplicação servidor, vamos agora mostrar outra das características mais importantes em todo o projeto, que é a base de dados que relaciona todos os dados necessários para o funcionamento da aplicação Android, como por exemplo os nomes dos restaurantes, *link's* para as fotos, pedidos, etc. Toda a informação que está a ser carregada na aplicação Android, está contida nesta base de dados. A base de dados ao longo de todo o processo de desenvolvimento foi alterada, de acordo com as funcionalidades inseridas, e consoante a necessidade de mais dados a serem apresentados ao utilizador, em baixo encontra-se uma figura ilustrativa do aspeto final do diagrama da base de dados e as suas relações.



Figura 27 - Diagrama da Base de Dados.

À que destacar que toda a programação da base de dados foi elaborada na linguagem de programação SQLite 3 a nível de desenvolvimento (local), e que a posteriora, quando era enviado todas as alterações para o ambiente de produção, a base de dados está a ser operada em PostgreSQL³², através das API internas do fornecedor Heroku, como anteriormente mencionado no tópico de *Cloud Computing*, este processo está a cargo do próprio fornecedor

³² Mais informação acerca de PostgreSQL em - <http://www.postgresql.org/>.

e não do programador. Com isto foi conseguido incorporar e utilizar mais duas tecnologias no projeto.

Estrutura Aplicacional

Depois de se ter abordado todos os aspetos relevantes a nível da aplicação servidor, é necessário, demonstrar a estrutura aplicacional que está encarregue de gerir toda a informação e comunicação entre a aplicação Android, e a aplicação Servidor, para se compreender melhor como funciona a interação entre as duas. Em baixo temos uma figura ilustrativa da representação gráfica da estrutura aplicacional e as suas principais tecnologias.



Figura 28 - Representação gráfica da Estrutura Aplicacional.

Através de um modelo de diagrama de sequência podemos observar como os pedidos são feitos pela aplicação Android através da interação do utilizador, quando esses mesmos pedidos são enviados a aplicação Servidor, o servidor processa esse pedido através do seu controlador, que por si, acede depois as bases de dados recolhendo ou efetuando alterações na informação nela contida. A base de dados depois de ter sido alterada ou consultada, envia de volta uma resposta a aplicação Servidor, que por si e através do controlador envia a resposta a aplicação Android, o solicitado. Todo este processo têm tempos de processamento esperado entre as alterações, assim como, dentro da própria aplicação, tendo em conta o solicitado pelo utilizador, como podemos demonstrar pela figura seguinte.

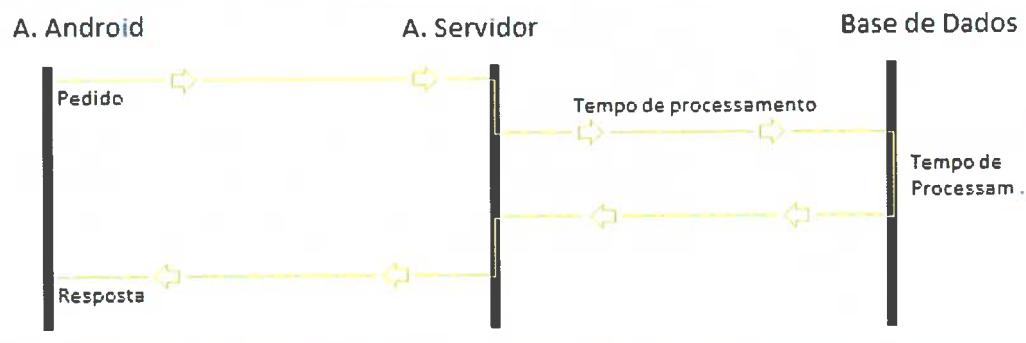


Figura 29 - Diagrama de Sequência dos pedidos.

Em baixo é demonstrado todo o código em relação, aos ficheiros de configuração, da aplicação Servidor. Começando pelo controlador (`api_controller.rb`), ficheiro de rotas (`routes.rb`), ficheiro de GEM, Base de dados e pagina *Welcome* do servidor. No ficheiro `api_controller.rb` devido a sua extensão, utilizou-se uma linha tracejado para separar os diferentes métodos, para uma melhor leitura do código.

//Ficheiro `api_controller.rb` (controlador), contém todos os métodos para operar sobre a base de dados.

```
class ApiController < ApplicationController
  protect_from_forgery with: :null_session, if: Proc.new { |c| c.request.format == 'application/json' }
  respond_to :json //O Controlador apenas responde a pedidos em formato JSON.

  // Método que retorna apenas um restaurante. (Get -- Retorna um valor).
  # get 'api/restaurantes/:idR' => 'api#getRestaurante'
  def getRestaurante
    //Verifica primeiro se o id do restaurante existe, se não existir apresenta mensagem de erro.
    unless Restaurante.exists?(params[:idR])
      render :status => 400, :json => { :error => 'invalid id restaurante' }
      return
    end
    //Se existir envia a informação sobre o restaurante (nome, morada, tipo, telefone, email).
    result = Restaurante.find(params[:idR])
    render :status => 200, :json => {
      :nome => result.nome,
      :morada => result.morada,
      :tipo => result.tipo,
      :telefone => result.telefone,
      :email => result.email
    }
  end

  // Método que retorna todos os restaurantes. (Get -- Retorna um valor).
  # get 'api/restaurantes' => 'api#getRestaurantes'
  def getRestaurantes
    //Aqui quando é feito o pedido, ele devolve todos os restaurantes, com a informação (id,nome, morada, tipo, telefone, email, facebook,foto).
  end
```

```

render :status => 200, :json => { :restaurantes => Restaurante.all.map { |restaurante| {
    :idR => restaurante.id,
    :nome => restaurante.nome,
    :morada => restaurante.morada,
    :foto => restaurante.foto,
    :facebook => restaurante.facebook,
    :tipo => restaurante.tipo,
    :telefone => restaurante.telefone,
    :email => restaurante.email
  }}
}

end



---


// Método que retorna um menu de um determinado restaurante. (Get -- Retorna um valor).
# get 'api/restaurantes/:idR/menus/:idM' => 'api#getMenu'
def getMenu
//Verifica primeiro se o id do restaurante existe, se não existir apresenta mensagem de erro.
unless Restaurante.exists?(params[:idR])
  render :status => 400, :json => { :error => 'invalid id restaurante'}
  return
end
//Coloca o restaurante dentro da variável result , depois de ter feito a verificação que existia.
  result = Restaurante.find(params[:idR])
//Verifica se o id do menu existe, se não existir apresenta mensagem de erro.
unless result.menus.exists?(params[:idM])
  render :status => 400, :json => { :error => 'invalid id menu'}
  return
end

  result = result.menus.find(params[:idM])
//Se o menu existir, mostra o seu id, e nome.
  render :status => 200, :json => {
    :idM => result.id,
    :nome => result.nome
  }
end



---


// Método que retorna todos os menus de um determinado restaurante. (Get -- Retorna um valor).
# get 'api/restaurantes/:idR/menus' => 'api#getMenus'
def getMenus
//Verifica primeiro se o id do restaurante existe, se não existir apresenta mensagem de erro.
unless Restaurante.exists?(params[:idR])
  render :status => 400, :json => { :error => 'invalid id restaurante'}
  return
end
//Colocação de todos os menus do restaurante, dentro da variável result.
  result = Restaurante.find(params[:idR]).menus
//Feito o mapa de todos os menus, mostrando o id, e nome de cada um.
  render :status => 200, :json => { :menus => result.map { |menu| {
    :idM => menu.id,
    :nome => menu.nome
  }}}
end



---


// Método que retorna todas as mesas de um determinado restaurante. (Get -- Retorna um valor).
# get 'api/restaurantes/:idR/mesas' => 'api#getMesas'
def getMesas

```

```

//Verifica primeiro se o id do restaurante existe, se não existir apresenta mensagem de erro.
unless Restaurante.exists?(params[:idR])
  render :status => 400, :json => {error => 'invalid id restaurante'}
  return
end
//Colocação de todas as mesas do restaurante, dentro da variável result.
result = Restaurante.find(params[:idR]).mesas
//Feito o mapa de todas as mesas, mostrando o id, numero e capacidade de cada uma.
render :status => 200, :json => { :mesas => result.map {|mesa| {
  :idM => mesa.id,
  :numero => mesa.numero,
  :capacidade => mesa.capacidade
}}}
end
-----  

// Método que retorna apenas uma mesa de um determinado restaurante. (Get -- Retorna um valor).
# get 'api/restaurantes/:idR/mesas/:nM' => 'api#getMesa'
def getMesa
//Verifica primeiro se o id do restaurante existe, se não existir apresenta mensagem de erro.
unless Restaurante.exists?(params[:idR])
  render :status => 400, :json => {error => 'invalid id restaurante'}
  return
end
//Colocação do restaurante, dentro da variável result.
result = Restaurante.find(params[:idR])
//Verificação se o número de mesa existe, neste restaurante (result), se não existir dá erro.
unless result.mesas.exists?(:numero => params[:nM])
  render :status => 400, :json => {error => 'invalid numero de mesa'}
  return
end
//Se existir coloca a mesa dentro da variável result e depois retorna o seu id, número e capacidade.
result = result.mesas.find_by_numero(params[:nM])
render :status => 200, :json => {
  :idM => result.id,
  :numero => result.numero,
  :capacidade => result.capacidade
}
end
-----  

// Método que retorna todos os produtos, que pertencem a um menu, de um determinado restaurante. (Get -- Retorna um valor).
# get 'api/restaurantes/:idR/menus/:idM/produtos' => 'api#getProdutos'
def getProdutos
//Verifica primeiro se o id do restaurante existe, se não existir apresenta mensagem de erro.
unless Restaurante.exists?(params[:idR])
  render :status => 400, :json => {error => 'invalid id restaurante'}
  return
end
//Colocação do restaurante, dentro da variável result.
result = Restaurante.find(params[:idR])
//Verificação se o id de menu existe, neste restaurante (result), se não existir dá erro.
unless result.menus.exists?(params[:idM])
  render :status => 400, :json => {error => 'invalid id menu'}
  return
end
//Colocação do menu, dentro da variável result.

```

```

result = result.menus.find(params[:idM])
//Colocação dos produtos, que se encontram naquele menu, dentro da variável result.
result = result.produtos
//Retorna todos os produtos, dentro daquele menu, com a informação (id, foto, nome, descrição, preço).
render :status => 200, :json => { :produtos => result.map {|produto| {
    :idP => produto.id,
    :foto => produto.foto,
    :nome => produto.nome,
    :descricao => produto.descricao,
    :preco => produto.preco
  }}}
end

```

// Método que retorna um produto, que pertencem a um menu, de um determinado restaurante. (Get -- Retorna um valor).

```
# get 'api/restaurantes/:idR/menus/:idM/produtos/:idP' => 'api#getProduto'
```

```
def getProduto
```

//Verifica primeiro se o id do restaurante existe, se não existir apresenta mensagem de erro.

```
unless Restaurante.exists?(params[:idR])
```

```
  render :status => 400, :json => { :error => 'invalid id restaurante'}
```

```
  return
```

```
end
```

//Colocação do restaurante, dentro da variável result.

```
result = Restaurante.find(params[:idR])
```

//Verificação se o id de menu existe, neste restaurante (result), se não existir dá erro.

```
unless result.menus.exists?(params[:idM])
```

```
  render :status => 400, :json => { :error => 'invalid id menu'}
```

```
  return
```

```
end
```

//Colocação do menu, dentro da variável result.

```
result = result.menus.find(params[:idM])
```

//Verificação se o id do produto existe, neste menu (result), se não existir dá erro.

```
unless result.produtos.exists?(params[:idP])
```

```
  render :status => 400, :json => { :error => 'invalid id produto'}
```

```
  return
```

```
end
```

//Colocação do produto, dentro da variável result.

```
result = result.produtos.find(params[:idP])
```

//Retorna os dados acerca do produto (id, foto, nome, descrição, preço).

```
render :status => 200, :json => {
```

```
    :idP => result.id,
    :foto => result.foto,
    :nome => result.nome,
    :descricao => result.descricao,
    :preco => result.preco
  }
```

```
}
```

```
end
```

// Método responsável por adicionar um produto ao pedido. (POST – faz um operação, escreve na b. de dados)

```
# post 'api/restaurantes/:idR/mesas/:nM/pedido' => 'api#setPedido'
```

```
# programação em ruby on rails
```

```
def setPedido
```

//Verifica primeiro se o id do produto existe, se não existir apresenta mensagem de erro.

```
unless params[:produto_id]
```

```
  render :status => 400, :json => { :errors => 'invalid id produto'}
```

```
  return
```

```

end
//Verifica se o id do pedido existe, se não existir apresenta mensagem de erro.
unless params[:pedido_id]
  render :status => 400, :json => { :errors => 'invalid id pedido' }
  return
end
//Verifica se o valor da quantidade é correto , se não for apresenta mensagem de erro.
unless params[:quantidade]
  render :status => 400, :json => { :errors => 'invalid quantidade' }
  return
end
//Colocação dos parâmetros recebidos em variáveis.
rest = Restaurante.find(params[:idR])
mesa = rest.mesas.find_by(numero: params[:nM])
//Verifica se o id do pedido for null, cria um novo.
if params[:pedido_id] == 'null'
  pedido = Pedido.new(:restaurante_id => rest.id, :mesa_id => mesa.id, :estado => 'Aberto')
  //Se o pedido não se conseguir-se criar, aparece uma mensagem de erro.
  unless pedido.save
    render :status => 400, :json => { :errors => 'Pedido! not saved! please, check the arguments' }
    return
  end
else //SE for enviado um id valido, que já existe, coloca na variável pedido.
  pedido = Pedido.find(params[:pedido_id])
end

produto = Produto.find(params[:produto_id]) //Procura do produto pelo id.
//Adicionar o produto, criação de um novo PedidoProduto.
pedido_produto = PedidoProduto.new(:pedido_id => pedido.id, :produto_id => produto.id, :preco =>
produto.preco, :quantidade => params[:quantidade])
//Se o PedidoProduto não se conseguir-se gravar, aparece uma mensagem de erro.
unless pedido_produto.save
  render :status => 400, :json => { :errors => 'PedidoProduto! not saved! please, check the arguments' }
  return
end
//Retorno do id do pedido, para poder adicionar os próximos pedidos de produtos, até ser fechado o pedido.
render :status => 200, :json => { :pedido_id => pedido.id}
end
-----  

// Método responsável por finalizar o pedido. (POST – faz um operação, escreve na base de dados).
# post 'api/restaurante/:idR/mesas/:nM/finalizar' => 'api#closePedido'
def closePedido
  unless params[:pedido_id]
    //Verifica primeiro se o id do pedido existe, se não existir apresenta mensagem de erro.
    render :status => 400, :json => { :errors => 'invalid id pedido' }
    return
  end
  //Colocação dos parâmetros recebidos em variáveis.
  rest = Restaurante.find(params[:idR])
  mesa = rest.mesas.find_by(numero: params[:nM])
  //Verifica se o id do pedido for null, apresenta uma mensagem de erro.
  if params[:pedido_id] == 'null'
    render :status => 400, :json => { :errors => 'Pedido! not saved! please, check the arguments' }
    return
  end
  //Colocação dos dados dentro de variáveis.

```

```

pedido = Pedido.find(params[:pedido_id])
pedido_prods = pedido.pedido_produtos
//Reset a variável total.
total = 0
//Método para calcular o valor do pedido.
pedido_prods.each do |pedido_prod|
  total = total + (pedido_prod.quantidade * pedido_prod.preco)
end
//Alteração do valor do pedido no campo estado com a designação Finalizado.
pedido.update(preco_total: total, estado: 'Finalizado')
//Retorno da mensagem a indicar sucesso no processo de finalizar pedido.
render :status => 200, :json => 'sucesso'
end

// Método que retorna todos os produtos pedidos, dentro de um pedido, que pertencem a um determinado restaurante. (Get -- Retorna um valor).
# get 'api/restaurantes/:idR/pedidos/:idP' => 'api#getPedido'
def getPedido
//Verifica primeiro se o id do restaurante existe, se não existir apresenta mensagem de erro.
unless Restaurante.exists?(params[:idR])
  render :status => 400, :json => {error => 'invalid id restaurante'}
  return
end
//Colocação do restaurante, dentro da variável result.
result = Restaurante.find(params[:idR])
//Verifica se o id do pedido existe, se não existir apresenta mensagem de erro.
unless result.pedidos.exists?(:id => params[:idP])
  render :status => 400, :json => {error => 'invalid numero de pedido'}
  return
end
//Colocação do restaurante, dentro da variável result.
result = Pedido.find(params[:idP])
//Retorna a informação acerca de todos os produtos que se encontra pedidos, num determinado Pedido, mostrando a informação (id, nome, quantidade, preço, total).
render :status => 200, :json => { :produtos => result.pedido_produtos.map {|pedido_produto| {
  :id => pedido_produto.id,
  :nome => Produto.find(pedido_produto.produto_id).nome,
  :quantidade => pedido_produto.quantidade,
  :preco => pedido_produto.preco,
  :total => pedido_produto.quantidade * pedido_produto.preco
}}}
end

// Método responsável por eliminar um produto do pedido. (Delete – faz um operação, apaga na base de dados).
#delete 'api/restaurantes/:idR/pedidos/:idP/pedidoproduto/:idPP' => 'api#deletePedidoProduto'
def deletePedidoProduto
//Verifica primeiro se o id do restaurante existe, se não existir apresenta mensagem de erro.
unless Restaurante.exists?(params[:idR])
  render :status => 400, :json => {error => 'invalid id restaurante'}
  return
end
//Colocação do restaurante, dentro da variável result.
result = Restaurante.find(params[:idR])
//Verifica se o id do pedido existe, se não existir apresenta mensagem de erro.
unless result.pedidos.exists?(:id => params[:idP])
  render :status => 400, :json => {error => 'invalid id de pedido'}

```

```

    return
end
//Colocação do id do pedido caso exista, dentro da variável result.
result = Pedido.find(params[:idP])
//Verifica se o id do produto pedido existe, dentro do pedido, se não existir apresenta mensagem de erro.
unless result.pedido_produtos.exists?(:id => params[:idPP])
  render :status => 400, :json => {error => 'invalid id de Pedido produto'}
  return
end
//Se existir coloca o id do produto pedido, dentro da variável result2.
result2 = result.pedido_produtos.find(params[:idPP])
//Destroi o produto do pedido, caso não consiga indica mensagem de erro.
unless result2.destroy
  render :status => 400, :json => {error => 'error on destroy'}
  return
end
//Retorna todos os produtos que ainda se encontram no pedido, mostrando a informação (id, nome, quantidade, preço)
render :status => 200, :json => { :produtos => result.pedido_produtos.map {|pedido_produto| {
  :id => pedido_produto.id,
  :nome => Produto.find(pedido_produto.product_id).nome,
  :quantidade => pedido_produto.quantidade,
  :preco => pedido_produto.preco
}}}
end
-----

---


// Método responsável por atualizar a quantidade de um produto pedido. (POST – faz um operação, escreve na base de dados).
#post 'api/restaurantes/:idR/pedidos/:idP/pedidoproduto/:idPP' => 'api#updateQuantidade'
def updateQuantidade
  unless params[:quantidade]
    //Verifica primeiro se o valor da quantidade é correto, se não for apresenta mensagem de erro.
    render :status => 400, :json => {errors => 'invalid quantidade'}
    return
  end
  //Verifica se o id do restaurante existe, se não existir apresenta mensagem de erro.
  unless Restaurante.exists?(params[:idR])
    render :status => 400, :json => {error => 'invalid id restaurante'}
    return
  end
  //Colocação do restaurante, dentro da variável result.
  result = Restaurante.find(params[:idR])
  //Verifica se o id do pedido existe, se não existir apresenta mensagem de erro.
  unless result.pedidos.exists?(:id => params[:idP])
    render :status => 400, :json => {error => 'invalid id de pedido'}
    return
  end
  //Colocação do id do pedido, dentro da variável result.
  result = Pedido.find(params[:idP])
  //Verifica se o id do produto pedido existe, dentro do pedido, se não existir apresenta mensagem de erro.
  unless result.pedido_produtos.exists?(:id => params[:idPP])
    render :status => 400, :json => {error => 'invalid id de Pedido produto'}
    return
  end
  //Atualização do valor da quantidade no produto pedido, dentro do pedido.
  result2 = result.pedido_produtos.find(params[:idPP])

```

```

result2.quantidade = params[:quantidade]
result2.save
//Mensagem de retorno de sucesso no processo de atualização.
render :status => 200, :json => 'sucesso'
end
end

```

//Ficheiro de routes.rb. Este ficheiro contém todas as rotas disponíveis para os pedidos http feitos pela aplicação Android, indicando o tipo de pedido, caminho da rota, e o nome do método que se encontra dentro do ficheiro api_controller.rb.

```

Rails.application.routes.draw do
  devise_for :admin_users, ActiveAdmin::Devise.config
  ActiveAdmin.routes(self)

//Base de dados acessíveis pelo ficheiro routes.rb (cada tabela é uma “resource”, um recurso).
# resources :pedido_produtos
# resources :pedidos
# resources :produtos
# resources :menus
# resources :mesas
# resources :restaurantes
//Pagina de Welcome do servidor, programada em HTML5.
# You can have the root of your site routed with "root"
root 'welcome#index'

//Rotas dos pedidos http disponíveis.
//Tipo-----Caminho-----Método.
get 'api/restaurantes' => 'api#getRestaurantes'
get 'api/restaurantes/:idR' => 'api#getRestaurante'

get 'api/restaurantes/:idR/mesas' => 'api#getMesas'
get 'api/restaurantes/:idR/mesas/:nM' => 'api#getMesa'

get 'api/restaurantes/:idR/menus' => 'api#getMenus'
get 'api/restaurantes/:idR/menus/:idM' => 'api#getMenu'

get 'api/restaurantes/:idR/menus/:idM/produtos' => 'api#getProdutos'
get 'api/restaurantes/:idR/menus/:idM/produtos/:idP' => 'api#getProduto'

get 'api/restaurantes/:idR/pedidos/:idP' => 'api#getPedido'

post 'api/restaurantes/:idR/mesas/:nM/pedido' => 'api#setPedido'
post 'api/restaurantes/:idR/mesas/:nM/finalizar' => 'api#closePedido'

post 'api/restaurantes/:idR/pedidos/:idP/pedidoproduto/:idPP' => 'api#updateQuantidade'
delete 'api/restaurantes/:idR/pedidos/:idP/pedidoproduto/:idPP' => 'api#deletePedidoProduto'
end

```

//Ficheiro de instalação das GEM , na aplicação servidor. Neste caso foi necessário para a instalação do Active Admin (Plugin de Backoffice).

```
source 'https://rubygems.org'
```

```
# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
gem 'rails', '4.1.8'
# Use sqlite3 as the database for Active Record
```

```

group :development, :test do
  gem 'sqlite3'
end
group :production do
  gem 'pg'
end
# Use SCSS for stylesheets
gem 'sass-rails', '~> 4.0.3'
# Use Uglifier as compressor for JavaScript assets
gem 'uglifier', '>= 1.3.0'
# Use CoffeeScript for .js.coffee assets and views
gem 'coffee-rails', '~> 4.0.0'
# See https://github.com/sstephenson/execjs#readme for more supported runtimes
# gem 'therubyracer', platforms: :ruby
# Use jquery as the JavaScript library
gem 'jquery-rails'
# Turbolinks makes following links in your web application faster. Read more:
https://github.com/rails/turbolinks
gem 'turbolinks'
# Build JSON APIs with ease. Read more: https://github.com/rails/jbuilder
gem 'jbuilder', '~> 2.0'
# bundle exec rake doc:rails generates the API under doc/api.
gem 'sdoc', '~> 0.4.0', group: :doc
# Windows does not include zoneinfo files, so bundle the tzinfo-data gem
gem 'tzinfo-data', platforms: [:mingw, :mswin]
//Gem adicionais instaladas.
# additional gems
gem 'coffee-script-source', '1.8.0'
gem 'activeadmin', github: 'activeadmin'
gem 'devise'
gem 'rails_12factor', group: :production
ruby '2.1.5'

```

//Base de dados Menu

```

class Menu < ActiveRecord::Base
  belongs_to :restaurante //Pertence a um restaurante.
  has_many :produtos, :dependent => :destroy //Tem um ou mais produtos.
//Validações de segurança, quando se está a criar um novo.
  validates :nome, :presence => true //Presença de nome obrigatório
  validate :restaurante_id_exists //Verifica se o id do restaurante existe.
//Método do erro caso o restaurante não exista.
  private
  def restaurante_id_exists
    unless Restaurante.exists?(self.restaurante_id)
      errors.add(:restaurante_id, 'does not exists')
    end
  end
end

```

//Base de dados Mesa

```

class Mesa < ActiveRecord::Base
  belongs_to :restaurante //Pertence a um restaurante.
  has_many :pedidos //Tem um ou mais pedidos.

```

```

//Validações de segurança, quando se está a criar um novo.
validates :numero, :presence => true, :numericality => {:greater_than => 0} //Presença obrigatório de numero de mesa, que é um numero e maior que 0.
validates :capacidade, :numericality => {:greater_than => 0} //que a capacidade é um numero maior que 0.
validate :restaurante_id_exists //Verifica se o id do restaurante existe.
//Método do erro caso o restaurante não exista.
private
def restaurante_id_exists
unless Restaurante.exists?(self.restaurante_id)
errors.add(:restaurante_id, 'does not exists')
end
end
end

//Base de dados Pedido
class Pedido < ActiveRecord::Base
belongs_to :restaurante //Pertence a um restaurante.
belongs_to :mesa //Pertence a uma mesa.
has_many :pedido_produtos, :dependent => :destroy //Tem um ou mais Produtos Pedidos.
//Validações de segurança, quando se está a criar um novo.
validates :estado, :presence => true, :inclusion => %w(Aberto Finalizado Processar Processado Pago Fechado Cancelado) //O estado têm que estar presente, e se pode ser um dos da lista de inclusão.
validate :restaurante_id_exists //Verifica se o id do restaurante existe.
validate :mesa_id_exists //Verifica se o id da mesa existe.
//Método do erro caso o restaurante não exista.
private
def restaurante_id_exists
unless Restaurante.exists?(self.restaurante_id)
errors.add(:restaurante_id, 'does not exists')
end
end
//Método do erro caso a mesa não exista.
def mesa_id_exists
unless Mesa.exists?(self.mesa_id)
errors.add(:mesa_id, 'does not exists')
end
end
end

//Base de dados Pedido_Produto
class PedidoProduto < ActiveRecord::Base
belongs_to :pedido //Pertence a um pedido.
belongs_to :produto //Pertence a um produto.
//Validações de segurança, quando se está a criar um novo.
validates :preco, :presence => true //Obrigatório ter preço.
validates :quantidade, :presence => true, :numericality => {:greater_than => 0} //Origa a quantidade ser de presença obrigatória, tem que ser um numero e maior que 0.
validate :produto_id_exists //Verifica se o id do produto existe.
validate :pedido_id_exists //Verifica se o id do pedido existe.
//Método do erro caso o id do produto não exista.
private
def produto_id_exists
unless Produto.exists?(self.produto_id)
errors.add(:produto_id, 'does not exists')
end

```

```

end
//Método do erro caso o id do pedido não exista.
def pedido_id_exists
  unless Pedido.exists?(self.pedido_id)
    errors.add(:pedido_id, 'does not exists')
  end
end
end

//Base de dados Produto

class Produto < ActiveRecord::Base
  belongs_to :menu //Pertence a um menu.
  has_many :pedido_produtos //Pode ser pedido uma ou várias vezes.
//Validações de segurança, quando se está a criar um novo.
  validates :nome, :presence => true //Nome de presença obrigatória.
  validates :descricao, :presence => true, :length => { :minimum => 2, :maximum => 400 } //Descrição de
  presença obrigatória com um tamanho mínimo de 2 caracteres e máximo 400 caracteres.
  validates :preco, :presence => true //Preço de presença obrigatória.
  validates :foto, :format => { :with => %r{^(gif|jpg|png)\z}i } //Foto, com o formato em acabar .gif.jpg ou .png.
  validate :menu_id_exists //Verifica se o id do menu existe.
//Método do erro caso o id do menu não exista.
  private
  def menu_id_exists
    unless Menu.exists?(self.menu_id)
      errors.add(:menu_id, 'does not exists')
    end
  end
end

//Base de dados Restaurante

class Restaurante < ActiveRecord::Base
  has_many :mesas, :dependent => :destroy //Tem uma ou mais Mesas.
  has_many :menus, :dependent => :destroy //Tem um ou mais Menus.
  has_many :pedidos, :dependent => :destroy //Tem um ou mais Pedidos.
//Validações de segurança, quando se está a criar um novo
  validates :nome, :presence => true, :uniqueness => true //Nome de presença obrigatória e que é único.
  validates :morada, :presence => true, :length => { :minimum => 2, :maximum => 200 } //Morada obrigatória
  com tamanho mínimo de 2 caracteres e máximo 200 caracteres.
  validates :tipo, :presence => true //Tipo de presença obrigatória.
  validates :telefone, :presence => true, :uniqueness => true, :format => { :with => /\A[2-9][0-9]{8}\z/i }
//Telefone de Presença obrigatória, único, e com o formato de [2-9][0-9]{8}.
  validates :email, :uniqueness => true, :format => { :with => /\A([^\@\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})\z/i }
end //Email único, e respeitando o formato de e-mail.

```

//Pagina welcome do servidor. (<https://bon-appetit-server.herokuapp.com/>). Programação feita em HTML. 5.

```

</br>
</br>
<div align="center">
  
  <h2>ISTEC - Instituto Superior de Tecnologias Avançadas</h2>
  </br>
  
  <h1>Bon Appétit Server</h1>
  </br>

```

<h1>Luís Pedro Leal Ribeiro , Nº 1901</h1>

</br>

<h2>Licenciatura de Informática</h2>

</br>

<h3>Projeto Global apresentado para o comprimento proposto necessário para obtenção do grau de licenciado em Informática </h3>

<h3>no Instituto Superior de Tecnologias Avançadas, realizado sobre a coordenação do Prof. Dr. Pedro Brandão.</h3>

</div>

BackOffice

Introdução

Neste ultimo capítulo vamos abordar o *backoffice*, ou seja, como foi feito e as tecnologias adjacentes, assim como, o seu propósito e funcionalidade, através do *plugin Active Admin*. Irá ser demonstrado algumas representações gráficas das principais funcionalidades.

Active Admin

O active admin é um *plugin* em Ruby on Rails, é uma *framework* administrativa para aplicações em Ruby on Rails, que permite geral estilos de interfaces administrativas. A sua maior vertente é a capacidade de ser abstrato em relação ao negócio que está a utilizar este componente, com isto, conseguimos ter uma forma fácil de implementar elegantes e bonitas interfaces de *backoffice*, sem muito esforço. Mais uma vez recorreu-se a um “componente” para o projeto, que é gratuita, sem qualquer custo de utilização.

O propósito deste componente é permitir a inserção dos dados na base de dados, ou seja, fazer todo o trabalho de *backoffice*, como por exemplo introduzir novos produtos, apagar produtos, alterar os menus disponíveis dos restaurantes, entre outros, assim como a interação com a aplicação Android, visionando por exemplo os pedidos efetuados pela aplicação Android.

A sua versatilidade permite através de filtros de informação programados, aceder a informação que necessitamos de uma forma clara e limpa para o utilizador, sendo esta uma das suas mais-valias.

Com isto, e em tempo real, todas as alterações feitas no Active Admin, são sentidas na aplicação Android. Todo este processo está devidamente acautelado através de um conjunto de autenticação, ou seja, através de utilizador e palavra passe.

Todo o processo de instalação e configuração pode ser consultado na documentação, e visionado com exemplos práticos, através do sitio oficial do *plugin Active Admin* em: <http://activeadmin.info/docs/documentation.html>. Em baixo é mostrado algumas figuras do aspeto das interfaces que permitem operar sobre os dados.

The screenshot shows the 'Products' section of the Bon Appétit Server application. At the top, there's a navigation bar with links for 'Bon Appétit Server', 'Dashboard', 'About Us', 'Contact', and 'Logout'. Below the navigation, the title 'Produtos' is displayed. The main area contains a table with 10 rows, each representing a product. Each row includes a small thumbnail image of the food item, followed by the product name and a brief description. To the right of each thumbnail are three small icons: a magnifying glass for 'View', a pencil for 'Edit', and a trash can for 'Delete'. A vertical sidebar on the right side of the table provides filtering options for 'Nome', 'Descrição', 'Preço', 'Custo', 'Estoque', and 'Status'.

Figura 30 - Interface do Active Admin para os Produtos.

Como podemos observar na figura de cima, temos a interface que proporciona a gestão dos produtos, que aparecem separados por linhas, ostentando a informação inerente a cada um deles, e a respetiva foto. Ao lado da foto temos as opções de ver, editar ou apagar o produto. No canto superior direito temos então as opções de filtrar a informação como anteriormente mencionado. Em baixo temos outra figura que ilustra outra interface, neste caso a dos restaurantes.

The screenshot shows the 'Restaurantes' section of the Bon Appétit Server application. The layout is similar to Figure 30, with a top navigation bar and a 'Restaurantes' title. Below the title is a table with multiple rows of restaurant data. Each row includes a thumbnail image, the restaurant name, address, phone number, and website. To the right of each row are three icons: a magnifying glass for 'View', a pencil for 'Edit', and a trash can for 'Delete'. A vertical sidebar on the right side of the table offers filters for 'Nome', 'Endereço', 'Número', 'Telefone', 'Site', and 'Status'.

Figura 31 - Interface do Active Admin para os Restaurantes.

Em baixo é demonstrado a configuração dos ficheiros referentes ao plugin Active Admin, neste caso usou-se o mesmo processo de separar o código através de uma linha a tracejado, para uma melhor leitura do código.

//Active menu.rb

```
//Ficheiro de Configuração da página de backoffice dos Menus.
ActiveAdmin.register Menu do
  menu :parent => 'Estabelecimentos' //Nome do Menu Pai, que contém os menus da barra superior de navegação.
  permit_params :id, :restaurante_id, :nome //Parâmetros acessíveis dentro do Active Admin.
  filter :nome // Parâmetro de condição de filtragem dos dados.
// Definição das colunas que aparecem na tabela menus com os respetivos dados.
  index do
    selectable_column
    column :id //Id do Menu
    column :nome //Nome do Menu
    actions //Acesso aos botões View, Edit e Delete.
  end
//Formulário de criação de um novo Menu.
  form do |f|
    f.inputs 'Detalhes' do //Cabeçalho
      f.input :restaurante_id, :as => :select, :collection => Restaurante.all.map {|restaurante| [restaurante.nome, restaurante.id]} //Serve para definir um campo de input no formulário, do tipo "select", em que os valores que o utilizador pode escolher é de uma lista, da coleção de restaurantes disponíveis.
      f.input :nome //Campo para inserir o nome do menu.
    end
    f.actions //Botões de Create Menu e Cancel.
  end
// Mostrar um elemento da tabela isolado
  show do
    attributes_table do
      row :id
      row :restaurante_id do |menu|
        link_to Restaurante.find(menu.restaurante_id).nome, admin_restaurante_path(menu.restaurante_id) //Mostrar um dos valores fazendo um link para outro Menu, neste caso para o do respetivo restaurante.
      end
    end
  end
end
```

//Active mesa.rb

```
//Ficheiro de Configuração da página de backoffice dos Mesa.
ActiveAdmin.register Mesa do
  menu :parent => 'Estabelecimentos'//Nome do Menu Pai, que contém os menus da barra superior de navegação.
  //Parâmetros acessíveis dentro do Active Admin.
  permit_params :id, :restaurante_id, :numero, :capacidade
  // Parâmetro de condição de filtragem dos dados.
  filter :numero
```

```

link_to Mesa.find(pedido.mesa_id).numero, admin_mesa_path(pedido.mesa_id)
end
column :preco_total // Campo preço total do Pedido.
column :estado // Campo estado do Pedido.
actions // Acesso aos botões View, Edit e Delete.
end
// Formulário de criação de um novo Pedido.
form do |f|
  f.inputs 'Detalhes' do // Cabeçalho
    f.input :restaurante_id, :as => :select, :collection => Restaurante.all.map {|restaurante| [restaurante.nome, restaurante.id]} // Serve para definir um campo de input no formulário, do tipo "select", em que os valores que o utilizador pode escolher é de uma lista, da coleção de restaurantes disponíveis.
    f.input :mesa_id, :as => :select, :collection => Mesa.all.map {|mesa| [mesa.numero, mesa.id]} // Serve para definir um campo de input no formulário, do tipo "select", em que os valores que o utilizador pode escolher é de uma lista, da coleção de mesas disponíveis.
    f.input :preco_total // Campo para inserir o preço total do pedido.
    f.input :estado // Campo para inserir o estado do pedido.
  end
  f.actions // Botões de Create Pedido e Cancel.
end
// Mostrar um elemento da tabela isolado
show do
  attributes_table do
    row :id // Campo id.
    row :restaurante_id do |pedido|
      link_to Restaurante.find(pedido.restaurante_id).nome, admin_restaurante_path(pedido.restaurante_id)
    end // Mostrar um dos valores fazendo um link para outro Menu, neste caso para o do respetivo restaurante.
    row :mesa_id do |pedido|
      link_to Mesa.find(pedido.mesa_id).numero, admin_mesa_path(pedido.mesa_id)
    end // Mostrar um dos valores fazendo um link para outro Menu, neste caso para a respetiva mesa.
    row :preco_total // Campo preço total do Pedido.
    row :estado // Campo estado do Pedido.
  end
end
end

```

```

// Active pedido_produto.rb

// Ficheiro de Configuração da página de backoffice dos Pedido Produtos.
ActiveAdmin.register PedidoProduto do
  menu :parent => 'Estabelecimentos' // Nome do Menu Pai, que contém os menus da barra superior de navegação.
  // Parâmetros acessíveis dentro do Active Admin.
  permit_params :id, :produto_id, :pedido_id
  // Parâmetro de condição de filtragem dos dados.
  filter :preco
  filter :quantidade
  // Definição das colunas que aparecem na tabela Pedido Produtos com os respetivos dados.
  index do
    selectable_column
    column :id // Campo Id.
    column :produto_id do |pedido| // Link para o respetivo Produto.
      link_to Produto.find(pedido.produto_id).nome, admin_produto_path(pedido.produto_id)
    end
    column :pedido_id do |pedido| // Link para o respetivo id do Produto.
  end

```

```

link_to Pedido.find(pedido_pedido_id).id, admin_pedido_path(pedido_pedido_id)
end
column :quantidade // Campo quantidade do produto.
column :preco // Campo Preço unitário do produto.
actions // Acesso aos botões View, Edit e Delete.
end
// Formulário de criação de um novo Pedido Produto.
form do |f|
  f.inputs 'Detalhes' do // Cabeçalho
    f.input :pedido_id, :as => :select, :collection => Pedido.all.map {|pedido| [pedido.id, pedido.id]} // Serve para definir um campo de input no formulário, do tipo "select", em que os valores que o utilizador pode escolher é de uma lista, da coleção de id's disponíveis.
    f.input :produto_id, :as => :select, :collection => Produto.all.map {|produto| [produto.nome, produto.id]} // Serve para definir um campo de input no formulário, do tipo "select", em que os valores que o utilizador pode escolher é de uma lista, da coleção de produtos disponíveis.
    f.input :quantidade // Campo para inserir a quantidade de produto.
    f.input :preco // Campo para inserir o preço do produto.
  end
  f.actions // Botões de Create Pedido Produto e Cancel.
end
// Mostrar um elemento da tabela isolado
show do
  attributes_table do
    row :id // Campo id.
    row :produto_id do |pedido|
      link_to Produto.find(pedido.produto_id).nome, admin_producto_path(pedido.produto_id)
    end // Mostrar um dos valores fazendo um link para outro Menu, neste caso para o do respetivo Produto.
    row :pedido_id do |pedido|
      link_to Pedido.find(pedido.pedido_id).id, admin_pedido_path(pedido.pedido_id) // Mostrar um dos valores fazendo um link para outro Menu, neste caso para o do respetivo id do Pedido.
    end
    row :quantidade // Campo com a quantidade do Produto.
    row :preco // Campo preço do Produto.
  end
end
end

```

```

// Active produto.rb

// Ficheiro de Configuração da página de backoffice do Produto.
ActiveAdmin.register Produto do
  menu :parent => 'Estabelecimentos' // Nome do Menu Pai, que contém os menus da barra superior de navegação.
  // Parâmetros acessíveis dentro do Active Admin.
  permit_params :id, :menu_id, :nome, :descricao, :preco, :foto
  // Parâmetro de condição de filtragem dos dados.
  filter :nome
  filter :descricao
  filter :preco
  // Definição das colunas que aparecem na tabela Produtos com os respetivos dados.
  index do
    selectable_column
    column :id // Campo Id.
    column :nome // Campo nome.
    column :descricao // Campo descrição.
  end
end

```

```

column :preco //Campo preço unitário.
column :foto do |foto| //Campo foto com preview reduzido da mesma.
  image_tag(foto.foto, height: '60', width: '80')
end
actions //Acesso aos botões View, Edit e Delete.
end
//Formulário de criação de um novo Produto.
form do |f|
  f.inputs 'Detalhes' do //Cabeçalho
    f.input :menu_id, :as => :select, :collection => Menu.all.map {|menu| [menu.restaurante.nome + '-' + menu.nome, menu.id]} //Serve para definir um campo de input no formulário, do tipo "select", em que os valores que o utilizador pode escolher é de uma lista, da coleção Restaurantes-Menus disponíveis.
    f.input :nome //Campo para inserir nome do produto.
    f.input :descricao //Campo para inserir a descrição do produto.
    f.input :preco //Campo para inserir o preço do produto.
    f.input :foto //Campo para inserir o link da foto do produto.
  end
  f.actions //Botões de Create Produto e Cancel.
end
// Mostrar um elemento da tabela isolado
show do
  attributes_table do
    row :id //Campo id.
    row :menu_id do |produto|
      link_to Menu.find(produto.menu_id).nome, admin_menu_path(produto.menu_id)
    end // Mostrar um dos valores fazendo um link para outro Menu, neste caso para o do respetivo Menu.
    row :nome //Campo id.
    row :descricao //Campo descrição do Produto.
    row :preco //Campo preço unitário do Produto.
    row :foto do |produto| //Campo foto com preview reduzido da mesma.
      image_tag(produto.foto, height: '120', width: '160')
    end
    row :created_at //Campo criado em.
    row :updated_at //Campo atualizado em.
  end
end
end

```

```

//Active restaurante,rb
//Ficheiro de Configuração da página de backoffice do Restaurantes.
ActiveAdmin.register Restaurante do
  menu :parent => 'Estabelecimentos' //Nome do Menu Pai, que contém os menus da barra superior de navegação.
  //Parâmetros acessíveis dentro do Active Admin.
  permit_params :id, :nome, :morada, :tipo, :telefone, :email, :foto, :facebook
  //Parâmetro de condição de filtragem dos dados.
  filter :nome
  filter :morada
  filter :foto
  filter :facebook
  filter :tipo
  filter :telefone
  filter :email
  // Definição das colunas que aparecem na tabela Restaurantes com os respetivos dados.

```

```

index do
  selectable_column
  column :id
  column :nome
  column :morada
  column :foto
  column :facebook
  column :tipo
  column :telefone
  column :email
  actions
end

//Formulário de criação de um novo Restaurante.
form do |f|
  f.inputs 'Detalhes' do
    f.input :nome
    f.input :morada
    f.input :foto
    f.input :facebook
    f.input :tipo
    f.input :telefone
    f.input :email
  end
  f.actions
end

// Mostrar um elemento da tabela isolado
show do
  attributes_table do
    row :id
    row :nome
    row :morada
    row :foto
    row :facebook
    row :tipo
    row :telefone
    row :email
    row :created_at
    row :updated_at
  end
end
end

```

Conclusão

Do desenvolvimento do projeto inerente a esta dissertação e do estudo realizado na área do desenvolvimento de aplicações para *smartphones* podemos retirar várias conclusões, sendo algumas de aspeto de fácil compreensão e outras apenas com o aprofundar do tema, se tornam visíveis. É uma área do presente, com um passado relativamente pequeno mas vasto, mas sem dúvida alguma que é um tema de futuro, não só pela constante evolução que têm vindo a acontecer, mas pela utilização massiva por parte dos utilizadores através destes aparelhos moveis “inteligentes”.

Com esta referência, a escolha da plataforma de desenvolvimento escolhida para o projeto, o SO Android, consistiu numa mais-valia, devido a panóplia de equipamentos de vários fabricantes que se consegue abranger, dai também a versatilidade alcançada pela aplicação desenvolvida. A questão principal recaiu sobre as provisões e estudos, que denunciam ser o SO móvel mais utilizado já no presente, e será o mais utilizado no futuro devido aos equipamentos que o utilizam.

Desta forma, podemos indicar que o principal objetivo proposto pelo trabalho foi realizado com sucesso na íntegra, com a aplicação desenvolvida, assim como, a programação do servidor, e com a integração de uma plataforma de *backoffice*. Mesmo sendo um projeto multifacetado, conseguiu-se obter resultados positivos, e para isso, é importante enfatizar o facto de ter sido utilizado vários conceitos e tecnologias, como: Android, Java, XML, API's da Google, Ruby on Rails, HTML 5, SQLite 3, PostGreSQL e um conceito relativamente novo Cloud Computing (PaaS), entre outros.

Devido a este desafio iminente, e pessoalmente, tendo em conta que nenhum destes conceitos ou tecnologias mencionados eram do meu conhecimento, foi muito gratificante a sua realização, porque me “obrigou” a levantar muitas questões sobre o desconhecido, ao meu intelectual.

Pondo isto resta-me afirmar que existe várias ideias de melhorias e propostas para adicionar ao projeto que serão adicionadas de futuro, como por exemplo: *Backoffice* para os estabelecimentos de restauração ou um sistema de *Login* através do Facebook.

Bibliografia

- Alliance, O. H. (5 de Novembro de 2007). Industry Leaders Announce Open Platform for Mobile Devices.
- Android. (s.d.). *Codenames, Tags, and Build Numbers*. Obtido de Source Android: <http://source.android.com/source/build-numbers.html>
- Carvalho, A. (2012). *Exercícios de Java - Algoritmia e Programação Estruturada*. Lisboa: FCA.
- Clare, A. (2012). *The Rough Guide to Android Phones and Tablets*. Penguin Books.
- David Wolber, H. A. (2014). *App Inventor 2*. O'Reilly Media.
- Ferreira, A. M. (2015). *Introdução ao Cloud Computing*. Lisboa: FCA.
- Gestwicki, P. a. (October 2011). App Inventor for Android with Studio-based Learning. *J. Comput. Sci. Coll.*, 55--63.
- Hartl, M. (2015). *Ruby on Rails Tutorial: Learn WEB Development with Rails*. Michigan: Addison-Wesley.
- Jesus, C. (2013). *Curso Prático de Java*. Lisboa: FCA.
- Kahn, J. (2014). Google shows off new version of Android, announces 1 billion active monthly users. *Google I/O*. TechSpot.
- Kelkka, R. a. (2009). Remote Identification and Information Processing with a Near Field Communication Compatible Mobile Phone. *CompSysTech '09*, 49:1--49:6.
- Li, T. a. (2014). Mayhem in the Push Clouds: Understanding and Mitigating Security Hazards in Mobile Push-Messaging Services. Em T. a. Li, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (pp. 978--989). New York, NY, USA: ACM.
- Lin, D.-Y. a. (2009). Collateral Evolution of Applications and Databases. Em D.-Y. a. Lin, *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops* (pp. 31--40). ACM: New York, NY, USA.
- Metz, S. (2012). *Practical Object-Oriented Design Ruby*. Michigan: Addison - Wesley.
- Protalinski, E. (8 de Dezembro de 2014). Google releases Android Studio 1.0, the first stable version of its IDE.
- Queirós, R. (2013). *Android - introdução ao desenvolvimento de Aplicações*. Lisboa: FCA.
- Queirós, R. (2014). *Desenvolvimento de Aplicações Profissionais em Android*. Lisboa: FCA.
- Tong, L. a. (2014). *QR Code Detection Based on Local Features*. Xiamen, China: ACM.
- Van Canneyt, S. a. (2012). Detecting Places of Interest Using Social Media. Em S. a. Van Canneyt, *Proceedings of the The 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology - Volume 01* (pp. 447--451). Washington, DC, USA: IEEE Computer Society.

Viennot, N. a. (June 2014). A Measurement Study of Google Play. *SIGMETRICS Perform. Eval. Rev.*, 221--233.

Anexos

Como anexo a este trabalho, apresenta-se um DVD com o seguinte conteúdo:

- Dissertação em formato digital (PDF).
- Aplicação para instalar em sistema operativo Android.
- Video a demonstrar o funcionamento da aplicação Android.
- Video a demonstrar a utilização do backoffice (Active Admin).
- Dois QRCodes para utilização da aplicação Bon Appétit.