DESENVOLVIMENTO DE APLICATIVOS NATIVOS ANDROID E IOS PARA O RESTAURANTE UNIVERSITÁRIO DA UFRJ

Felipe Podolan Oliveira

Projeto de Graduação apresentado ao Curso de Engenharia de Computação e Informação da Escola Politécnica da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Computação e Informação.

Orientador: Flávio Luis de Mello

Rio de Janeiro
Setembro de 2018

Não é o conhecimento, mas o
ato de aprender, não a posse
mas o ato de chegar lá, que
concede a maior satisfação.
Carl Friedrich Gauss

Agradecimentos

Quando eu tinha 16 anos, eu resolvi sair de casa para estudar. Foi uma decisão difícil que daria início a uma jornada igualmente difícil. Minha vida mudou completamente sem meus pais e sem minha irmã ao meu lado, sem o conforto que morar com a família proporciona e em uma cidade completamente diferente da cidade em que eu nasci no interior do Paraná.

Em todas minhas decisões, minha mãe e meu pai sempre me apoiaram seja com conselhos, com apoio moral ou financeiro. Durante minha vida acadêmica na UFRJ e fora dela, eu passei por diversas dificuldades e meus pais sempre estiveram do meu lado.

Agradeço a eles pelo suporte, pelo carinho e pela educação que eles me deram. Meus pais sempre serão meus exemplos de vida, de como ser como pessoa.

Eu também agradeço à minha irmã, ao meu cunhado e ao meu sobrinho pelo amor e carinho que sempre me deram e também aos meus amigos que sempre se mostraram dispostos a me ajudar.

Agradeço à UFRJ, como instituição, que foi quase como um segundo lar por alguns anos e me proporcionou diversas oportunidades únicas, como o intercâmbio acadêmico em Normam, Oklahoma e em Miami, Flórida, nos EUA, através do programa Ciências sem Fronteiras, em 2015 e 2016.

Por fim, agradeço aos meus professores que contribuíram para a minha formação e por passarem o conhecimento necessário para eu me tornar um Engenheiro de Computação e Informação.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Computação e Informação.

DESENVOLVIMENTO DE APLICATIVOS NATIVOS ANDROID E IOS PARA O RESTAURANTE UNIVERSITÁRIO DA UFRJ

Felipe Podolan Oliveira

Setembro/2018

Orientador: Flávio Luis de Mello

Curso: Engenharia de Computação e Informação

Há uma tendência de uso de plataformas móveis como meio de acesso aos serviços digitais, tornando-os mais acessíveis. O projeto de graduação que deu origem a esta monografia teve como objetivo criar aplicativos móveis para o sistema de agendamento do Restaurante Universitário da Universidade Federal do Rio de Janeiro (UFRJ). Este serviço é gerenciado pelos funcionários do Restaurante Universitário (RU), consumidos por seus clientes e tem como objetivo virtualizar as filas físicas de acesso às unidades do Restaurante Universitário (RU). Para atingir-se o objetivo foram criados três aplicativos (um Android para os funcionários, um Android para os clientes e um iOS, também, para os clientes) em linguagem nativa e utilizando-se dos SDKs para Android e iOS. As funcionalidades requeridas foram todas entregues levando em consideração a criação de boas User Interfaces (UI) e tendo em mente a User Experience (UX). Tais funcionalidades consistem em criar, editar, excluir de filas virtuais, bem como atendê-las de duas maneiras: por fila ou via QRCode para o aplicativo dos funcionários e, já para as aplicações para clientes, as funcionalidades são visualizar as filas virtuais disponíveis e criar, buscar e excluir agendamentos para cada cliente em uma fila.

Palavras-Chave: Computação móvel, Android, iOS.

Abstract of the Undergraduate Project presented to Poli/COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Computer and Information Engineer.

DEVELOPMENT OF NATIVE APPLICATIONS IN ANDROID AND IOS FOR THE RESTAURANTE UNIVERSITÁRIO OF UFRJ

Felipe Podolan Oliveira

September/2018

Advisor: Flávio Luis de Mello

Course: Computer and Information Engineering

There is a tendency in the use of mobile platforms as a mean to access digital services, making them more accessible. The graduation projected which originated this thesis had as its objective the development of mobile applications for the scheduling system of the Restaurante Universitário from Universidade Federal do Rio de Janeiro (UFRJ). This service is managed by the Restaurante Universitário's staff, consumed by its clients and has as its objective the virtualization of the physical lines (or queues) to access the Restaurante Universitário's establishment unities. To achieve the objective three apps were created (one in Android for the staff, one in Android for the clientes and one in iOS for the clients) in native language and using the SDK's for Android and iOS. The functionalities required were all delivered taking into account the generation of good User Interfaces (UI) and having in mind the User Experience (UX). These functionalities consist in create, edit, delete virtual queues, as well as manage them in two ways: one by one or using QRCodes for the staff app. For the clients apps, the functionalities are consult the available queues and create, edit and delete tickets for each client in a queue.

Keywords: Mobile Computing, Android, iOS.

Felipe Podolan Oliveira

Desenvolvimento de Aplicativos Nativos Android e iOS para o Restaurante Universitário da UFRJ

PROJETO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE COMPUTAÇÃO E INFORMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE COMPUTAÇÃO E INFORMAÇÃO.

Examinadores:

Prof. Flávio Luis de Mello, D.Sc.

Prof. Maria Alice Ferruccio da Rocha, D.Sc.

Prof. Ricardo Guerra Marroquim, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2018

Podolan Oliveira, Felipe

/Felipe Podolan Oliveira. – Rio de Janeiro: UFRJ/POLI – COPPE, 2018.

X, 59 p. 29, 7cm.

Orientador: Flávio Luis de Mello

Projeto (graduação) – UFRJ/ Escola Politécnica/ Curso de Engenharia de Computação e Informação, 2018.

Referências Bibliográficas: p. 58 – 59.

1. Computação móvel. 2. Android. 3. iOS. I. de Mello, Flávio Luis. II. Universidade Federal do Rio de Janeiro, Escola Politécnica/ Curso de Engenharia de Computação e Informação. III. Título.

Sumário

1 Introdução **1**

1.1 Tema . . . 1
1.2 Delimitação . . . 1
1.3 Justificativa . . . 1
1.4 Objetivo . . . 2
1.5 Metodologia . . . 2
1.6 Materiais . . . 3
1.7 Descrição . . . 3

2 Fundamentação **5**

2.1 Kit de Desenvolvimento de Software (SDK) . . . 5
2.2 Interface Gráfica (UI) . . . 5
2.3 Experiência do Usuário (UX) . . . 6
2.4 Web Service . . . 7
2.5 Criação de projetos para Android . . . 9
2.6 Criação de projetos para iOS . . . 14

3 Desenvolvimento Android **17**

3.1 Activity e Fragment . . . 17
3.2 Intent e FragmentManager . . . 22

4 Desenvolvimento iOS **28**

4.1 UIViewController . . . 28
4.2 Segue . . . 33

5 Projeto dos aplicativos para o RU da UFRJ **36**

5.1 Precedentes . . . 36
5.2 Aplicativos móveis . . . 37
5.2.1 Aplicativo Android para funcionários . . . 37
5.2.2 Aplicativo Android para clientes . . . 46
5.2.3 Aplicativo iOS para clientes . . . 50

6 Conclusão e Trabalhos Futuros **56**

Referências Bibliográficas **58**

Lista de Figuras

2.1 Utilização de Internet em 2015 por tipo de dispositivo 7
2.2 Arquitetura Cliente-Servidor 8
2.3 Arquitetura de uma API 8
2.4 Novo projeto no Android Studio - Passo 1 9
2.5 Novo projeto no Android Studio - Passo 2 10
2.6 Novo projeto no Android Studio - Passo 3 10
2.7 Novo projeto no Android Studio - Passo 4 11
2.8 IDE Android Studio 11
2.9 Selecionar dispositivo no Android Studio 12
2.10 Criar dispositivo virtual - Passo 1 12
2.11 Criar dispositivo virtual - Passo 2 13
2.12 Criar dispositivo virtual - Passo 3 13
2.13 Exemplo com Empty Activity 14
2.14 Novo projeto no XCode - Passo 1 14
2.15 Novo projeto no XCode - Passo 2 15
2.16 Novo projeto no XCode - Passo 3 15
2.17 Selecionar dispositivo no XCode 16
2.18 Exemplo com UIViewController único em branco 16

3.1 Ciclo de vida de uma Activity do Android 18
3.2 Exemplo de uso de Fragments 20
3.3 Ciclo de vida de um de um Fragment do Android 20
3.4 Exemplo de UI Android com alguns Views 21

4.1 Arquivos criados pelo XCode 28
4.2 Storyboard com apenas um View em branco 29
4.3 Modificar fundo da UI usando o Attribute Inspector 30
4.4 Procurando por UILabel no Attribute Inspector 30
4.5 Constraints de UIViews 31
4.6 Adicionando UIViewController ao Storyboard 31
4.7 Relacionando UIViewController com arquivo Swift 32
4.8 Botão para mostrar o Assistant editor no XCode 32
4.9 Arrastando UIView para UIViewController no XCode 32

4.10 Ciclo de vida de um UIViewController . . . 33
4.11 Criando um Segue no XCode . . . 34
4.12 Exemplo de aplicação com Segue . . . 35

5.1 Fila de espera para acesso ao RU do CT . . . 36
5.2 LoginActivity do RUAdmin . . . 37
5.3 Fluxo de validação do token JWT . . . 38
5.4 Menu lateral do app RUAdmin . . . 39
5.5 Tela de início (FragmentInitial) do app RUAdmin . . . 39
5.6 ProgressBar no QueuesFragment do RUAdmin . . . 40
5.7 ListView no QueuesFragment do RUAdmin . . . 40
5.8 Criar ou editar fila no RUAdmin - Passo 1 . . . 41
5.9 Criar ou editar fila no RUAdmin - Passo 2.1 . . . 41
5.10 Criar ou editar fila no RUAdmin - Passo 2.2 . . . 41
5.11 Criar ou editar fila no RUAdmin - Passo 3 . . . 42
5.12 Criar ou editar fila no RUAdmin - Passo 4 . . . 42
5.13 Criar ou editar fila no RUAdmin - Passo 5 . . . 42
5.14 Criar ou editar fila no RUAdmin - Passo 6 . . . 42
5.15 Criar ou editar fila no RUAdmin - Confirmação . . . 43
5.16 Excluir fila no RUAdmin - AlertDialog . . . 44
5.17 Excluir fila no RUAdmin - Confirmação . . . 44
5.18 Antender fila no RUAdmin - Tela inicial . . . 45
5.19 Modificar status de agendamento no RUAdmin - AlertDialog . . . 46
5.20 Modificar status de agendamento no RUAdmin - Confirmação . . . 46
5.21 Android - menu lateral . . . 47
5.22 Android - Cardápio . . . 47
5.23 Android - Agendamento . . . 48
5.24 Android - Agendamento - Confirmação . . . 48
5.25 Android - Busca . . . 49
5.26 Android - Agendamento . . . 49
5.27 Android - Excluir agendamento . . . 50
5.28 Android - Agendamentos disponíveis . . . 50
5.29 iOS - Main.Storyboard . . . 51
5.30 iOS - Cardápio . . . 52
5.31 iOS - Agendar . . . 53
5.32 iOS - Buscar . . . 53
5.33 iOS - Agendamento . . . 54
5.34 iOS - Excluir agendamento . . . 54
5.35 iOS - Agendamentos disponíveis . . . 55

Lista de Quadros

2.1 Reclamações mais comuns de usuários de aplicações móveis 6

Lista de Códigos-Fonte

3.1 Classe MainActivity - Exemplo de implementação básica 19
3.2 XML de layout da MainActivity - Exemplo de implementação básica 22
3.3 Classe MainActivity - Exemplo com Intent 23
3.4 XML de layout da MainActivity - Exemplo com Intent 24
3.5 Classe SecondActivity - Exemplo com Intent 25
3.6 Classe Fragment2 - Exemplo com FragmentManager 26
3.7 Classe MainActivity - Exemplo com FragmentManager 27
4.1 Classe MessageViewController - Exemplo com Segue 34
4.2 Classe ViewController - Exemplo com Segue 35

Capítulo 1

Introdução

1.1 Tema

O tema deste trabalho é o desenvolvimento de aplicativos móveis. Neste sentido, os problemas a serem resolvidos são a viabilidade em criar aplicações para plataformas móveis, bem como, a capacidade destas aplicações de ampliar a acessibilidade aos serviços oferecidos e de melhorar a experiência de uso dos usuários.

1.2 Delimitação

O objetivo do projeto é atender aos clientes e aos funcionários do Restaurante Universitário (RU) do Centro de Tecnologia (CT) e de Letras da Universidade Federal do Rio de Janeiro (UFRJ). O escopo de usuários é variável sendo composto pelos alunos, professores e servidores da instituição, bem como por eventuais visitantes que vão à universidade para congressos, semanas esportivas, dentre outros. O projeto também tem capacidade de atender às outras unidades do RU da UFRJ, bem como ser adaptado para outras universidades.

1.3 Justificativa

O Restaurante Universitário da UFRJ tem como objetivo oferecer alimentação de qualidade, equilibrada, e acessível de forma a favorecer a permanência dos estudantes no espaço universitário, permitindo-lhes dedicação integral aos estudos, sendo importante meio de combate à evasão escolar.

Entretanto, apesar de ser benéfico à comunidade universitária, o Restaurante Universitário enfrenta alguns problemas, sendo o principal as filas de espera. Pensando nisso, a Decania do Centro de Tecnologia resolveu criar um sistema de agendamento online cujo objetivo é alocar os horários de entrada no Restaurante Univer-

sitário e, desta forma, tornar as filas, que antes eram físicas e que geravam desgaste aos alunos, em filas virtuais.

Esse projeto funciona desde 2016 na unidade do Restaurante Universitário do Centro de Tecnologia. Os agendamentos são feitos através do website www.ru.ct.ufrj.br. Entretanto, para melhorar ainda mais a experiência dos usuários, bem como para ampliar a acessibilidade aos serviços prestados pelo RU, a Decania do Centro de Tecnologia resolveu desenvolver aplicativos nativos para Smartphone tendo em vista a crescente popularização desta modalidade, notoriamente para as plataformas Android e iOS. O sistema atual também é acessível via aplicações móveis de Web Browser, que por não terem sido desenvolvidas nativamente para as plataformas móveis.

1.4 Objetivo

O objetivo geral é, portanto, desenvolver aplicações móveis nativas em Android (Java) e em iOS (Swift) capazes de fornecer os serviços virtuais do Restaurante Universitário, sendo o principal deles o agendamento de horários para entrar no restaurante, de forma a resolver os problemas de demora e de desorganização das filas do Restaurante Universitário. Desta forma, tem-se como objetivos específicos principais:

- Permitir a criação, edição e exclusão de filas virtuais para o acesso ao RU por parte de seus funcionários;
- Permitir o agendamento em filas virtuais de acesso ao RU, bem como sua exclusão ou verificação de status por parte dos clientes;
- Visualização das filas virtuais vigêntes por parte de todos os usuários do RU.

1.5 Metodologia

Este trabalho utilizou os kits de desenvolvimento de software (SDK, do inglês Software Development Kit) oficiais oferecidos pela Google e pela Apple para suas plataformas Android e iOS, respectivamente.

Atendendo às boas práticas de programação e aos conceitos de orientação a objetos, foram desenvolvidos três aplicativos móveis. São eles:

- Aplicativo Android para clientes do RU;
- Aplicativo iOS para clientes do RU;
- Aplicativo Android para funcionários do RU.

As etapas da criação de cada aplicação consistem em:

- Criação de uma interface para cada funcionalidade;
- Criação da lógica de interação com o usuário para alternar entre as interfaces;
- Criação de métodos para a comunicação com o Back-End;
- Atualização dos dados exibidos na interface ou alternância de interface de acordo com a resposta do Back-End.

A proposta deste trabalho é mostrar como cada etapa do projeto foi desenvolvida para os aplicativos do RU com respaldo na literatura de desenvolvimento de aplicativos para Android e para iOS. O aplicativo para os funcionários foi desenvolvido apenas para a plataforma Android.

Além disso, são expostas as possíveis tecnologias a serem usadas para a comunicação com o Back-End, como por exemplo, os estilos de arquitetura de serviços REST versus SOAP em APIs. O desenvolvimento da API de comunicação e do Back-End, entretanto, não fazem parte do escopo deste projeto, pois os mesmos já estavam em desenvolvimento por outro aluno desde o início do projeto.

O êxito deste trabalho está centrado na entrega de todas as funcionalidades dos aplicativos.

1.6 Materiais

Foi utilizado um notebook pessoal para o desenvolvimento do projeto. Trata-se de um Macbook, pois é necessário para desenvolvimento nativo iOS e atende às demandas para desenvolvimento Android. Os softwares utilizados são gratuitos. Para desenvolvimento iOS, foi utilizado o software XCode, e para desenvolvimento Android, o software Android Studio.

1.7 Descrição

O capítulo 2 deste trabalho apresenta alguns conceitos iniciais como interface gráfica (UI), experiência do usuário (UX) e API, discutindo algumas tecnologias da camada de aplicação da internet em sistemas comumente utilizados por aplicações móveis. Também apresenta as etapas de criação de um projeto em ambos os softwares Android Studio e XCode. O capítulo 3, por sua vez, aprofunda a discussão sobre sistemas Android, com respaldo nos conceitos de Activity e de Fragment, relacionados ao desenvolvimento de UIs e, também nos conceitos de Intent e de FragmentManager, relacionados com a UX. O capítulo 4 faz o mesmo para a discussão sobre desenvolvimento iOS, utilizando os conceitos de UIViewController que está associado ao

desenvolvimento de UIs e de Segue que influencia a UX. Já o capítulo 5 descreve a implementação dos aplicativos do RU explicando cada componente do projeto bem como as decisões que foram tomadas ao longo de suas implementações. O último capítulo conclui o trabalho e discute possíveis trabalhos futuros.

Capítulo 2

Fundamentação

2.1 Kit de Desenvolvimento de Software (SDK)

Para viabilizar o desenvolvimento de uma aplicação móvel é necessário utilizar um Kit de Desenvolvimento de Software, frequentemente abreviado para SDK (Software Development Kit). Um SDK é um pacote que contém ferramentas, bibliotecas, códigos fonte, e outros utilitários pré criados [16]. A Google e a Apple oferecem respectivamente o Android SDK e o iOS SDK, que possibilitam o desenvolvimento de aplicativos para suas plataformas. Com a utilização de SDKs um desenvolvedor pode criar as interfaces gráficas e as funcionalidades da aplicação.

2.2 Interface Gráfica (UI)

Interface gráfica, comumente abreviada para UI (User Interface), é a parte da aplicação que o usuário vê e com a qual ele interage [6]. A interface inclui as telas, janelas, controles, menus e etc [6].

De acordo com um estudo publicado pelo Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) [12] sobre os motivos pelos quais os usuários mais reclamam de aplicações móveis, a interface gráfica é um dos casos mais frequentes, o que demonstra a necessidade de empenho no desenvolvimento de interfaces agradáveis e intuitivas.

Não é de se estranhar que as empresas desenvolvedoras dos sistemas operacionais para plataformas móveis ofereçam maneiras de padronizar as interfaces dos aplicativos feitos para suas plataformas.

De fato, a Google e a Apple oferecem dentro de seus SDKs ferramentas de padronização de interfaces. No caso da Google, esse conjunto de ferramentas é chamado de Material Design [10] e, além de ser oferecido para o Android, também pode ser utilizada nas plataformas Web, Flutter e, inclusive, iOS. Já a Apple, oferece seu kit

gráfico, o UIKit [1], apenas para sua própria plataforma.

2.3 Experiência do Usuário (UX)

Um terceiro conceito muito importante quando se trata de aplicações móveis é a Experiência do Usuário, comumente abreviada para UX (User Experience). Apesar de a UX ser um conceito amplamente disseminado e aceito, ele ainda não é bem definido ou entendido. Entretanto, para uma certa compreensão, pode-se utilizar a definição de Hassenzahl & Tractinsky [14]:

> Experiência de usuário é a consequêcia do estado interno de um usuário (predisposição, expectativa, necessidade, motivação, humor, etc), das caracteristicas do sistema designado (complexibilidade, propósito, usabilidade, funcionalidade, etc) e do contexto (ou ambiente) no qual a interação ocorre (configuração organizacional/social, significância da atividade, voluntariedade de uso, etc).

Pode se dizer que a Experiência do Usuário consiste nas sensações que um produto gera em um usuário que, consequentemente, forma opiniões a respeito deste produto com respaldo nessas sensações.

A importância da Experiência do Usuário fica ainda mais clara ao notar que todos os itens mencionados no estudo [12] resumidos no quadro a seguir podem ser relacionados à UX.

Complaint type	Description	Example review
App Crashing	The app often crashed.	"Crashes immediately after starting."
Compatibility	The app had problems on a specific device or an OS version.	"I can't even see half of the app on my iPod Touch."
Feature Removal	A disliked feature degraded the user experience.	"This app would be great, but get rid of the ads!"
Feature Request	The app needed additional features.	"No way to customize alerts."
Functional Error	The problem was app specific.	"Not getting notifications unless you actually open the app."
Hidden Cost	The full user experience entailed hidden costs.	"Great if you weren't forced to buy coins for REAL money."
Interface Design	The user complained about the design, controls, or visuals.	"The design isn't sleek and isn't very intuitive."
Network Problem	The app had trouble with the network or responded slowly.	"New version can never connect to server!"
Privacy and Ethics	The app invaded privacy or was unethical.	"Yet another app that thinks your contacts are fair game."
Resource Heavy	The app consumed too much energy or memory.	"Makes GPS stay on all the time. Kills my battery."
Uninteresting Content	The specific content was unappealing.	"It looks great, but the actual gameplay is boring and weak."
Unresponsive App	The app responded slowly to input or was laggy overall.	"Bring back the old version. Scrolling lags."
Not Specific	The user's comment wasn't useful or didn't point out a problem.	"Honestly the worst app ever."

Quadro 2.1: Reclamações mais comuns de usuários de aplicações móveis

Infere-se do Quadro 1 que um usuário de aplicações móveis espera que a aplicação

- Não trave;
- Seja compatível com seu dispositivo;
- Atenda somente às funcionalidades esperadas;
- Explicite as cobranças utilizadas na aplicação;
- Possua uma boa interface gráfica;
- Possua boa conectividade com a Internet;
- Atenda às condições de privacidade e ética esperadas da aplicação;
- Consuma o mínimo possível de recursos de hardware;
- Seja interessante;
- Seja responsiva.

2.4 Web Service

Um item notório na publicação do Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) [12] previamente citada é a conectividade com a Internet. Apesar de a pesquisa ter sido feita nos Estados Unidos, a realidade no Brasil não é muito diferente. Segundo o Instituto Brasileiro de Geografia e Estatística (IBGE) [11], em 2014, pela primeira vez, o uso do telefone celular para acessar a Internet ultrapassou o uso de microcomputador nos domicílios brasileiros [11]. Em 2015, esse cenário se repetiu no País, [11] conforme o mostra gráfico a seguir:

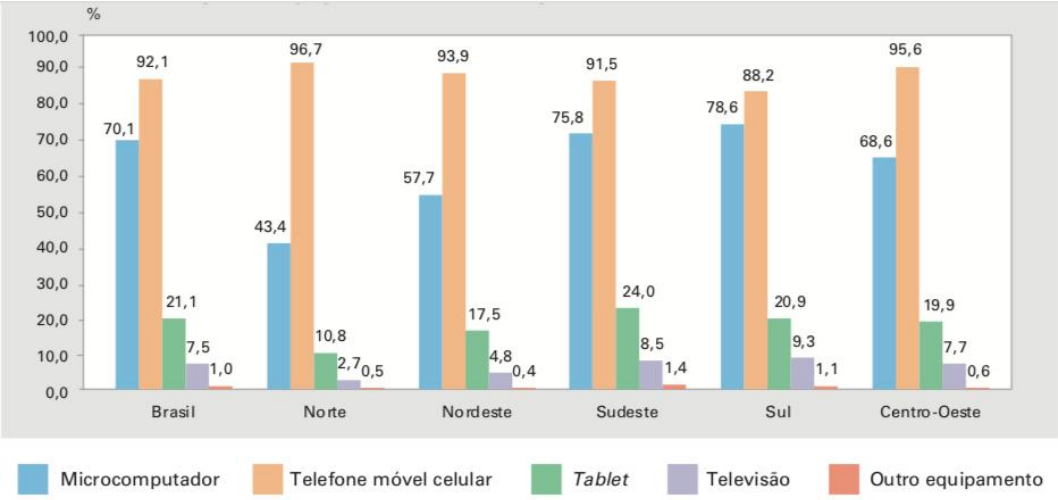

Figura 2.1: Utilização de Internet em 2015 por tipo de dispositivo [11]

Ao comunicar-se com a Internet, um aplicativo móvel passa a utilizar um web service. Mais especificamente, o app passa a ser um cliente da arquitetura cliente-servidor de um web service.

Um web service é um sistema de software designado a suportar interações máquina-com-máquina sobre uma rede de computadores [8]. Num sistema web existem dois agentes: uma máquina que provém um serviço, conhecida como servidor, e uma máquina que requisita o serviço, chamada de cliente, conforme mostra a Figura 2.2. Há diversas arquiteturas possíveis de comunicação entre o cliente e o servidor, uma delas é através de uma Application Programming Interface [15].

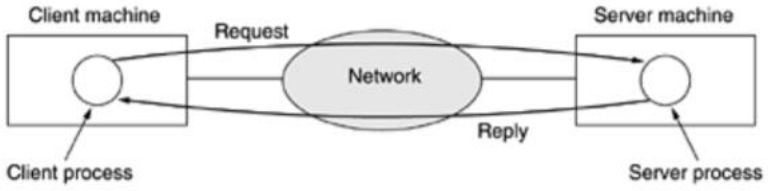

Figura 2.2: Arquitetura Cliente-Servidor [15]

Pessoas usam aplicações via Interfaces Gráficas. Já aplicações usam outras aplicações via Application Programming Interface (API) [2]. APIs provêm as capacidades essenciais para conectar, estender e integrar software [2]. Com uma mesma API é possível acessar os serviços de um servidor em muitos clientes distintos.

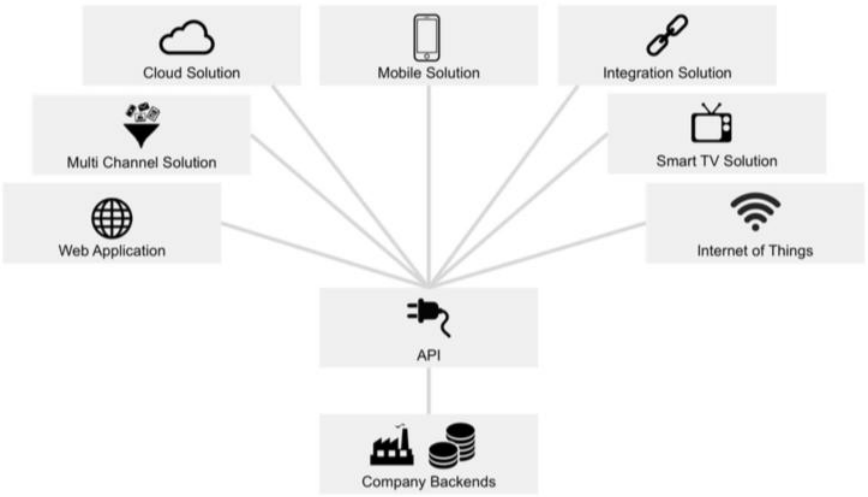

Figura 2.3: Arquitetura de uma API [2]

Existem diversos estilos de arquiteturas de implementação de APIs. Dentre elas, o REST (Representation State Transfer), o RPC (Remote Procedure Call) e o SOAP (Simple Object Access Protocol). Sendo hoje em dia as mais comuns o REST e o SOAP [5].

O REST é um estilo de arquitetura para serviços que define um conjunto de restrições e acordos na arquitetura. Um serviço que cumpre as resgras REST é

chamado de RESTful [2]. O REST foi desenvolvido para fazer uso otimizado de infraestruturas baseadas em HTTP [2].

Mensagens REST possuem pouco payload, o que é adequado para aplicações móveis [17].

Já o SOAP é o protocolo padrão de interface para web services proposto pelo World Wide Web Consortium (W3C) e é aplicável a qualquer protocolo da camada de aplicação (HTTP, SHTTP, FTP, etc) [5].

Entretanto, as mensagens SOAP contém muitos meta dados e apenas suportam estruturas verbosas de XML para solicitações e respostas. Além disso, devido ao seu grande tamanho, serviços SOAP são considerados complexos para ambos os provedores e consumidores de serviços [2].

Consequentemente, o estilo mais adequado de arquitetura de um web service utilizado por aplicações móveis é o REST, pois possui maior facilidade de desenvolvimento, bem como, devido à menor quantidade de payload, tem suas mensagens processadas com maior velocidade.

2.5 Criação de projetos para Android

O Android Studio é a IDE oficial para desenvolvimento de aplicativos para Android.

Depois de fazer o download e de instalar o software, o primeiro passo para criar uma aplicação Android é escolher a opção: "Start a new Android Studio project" ou equivalente caso o idioma de instalação não seja o inglês.

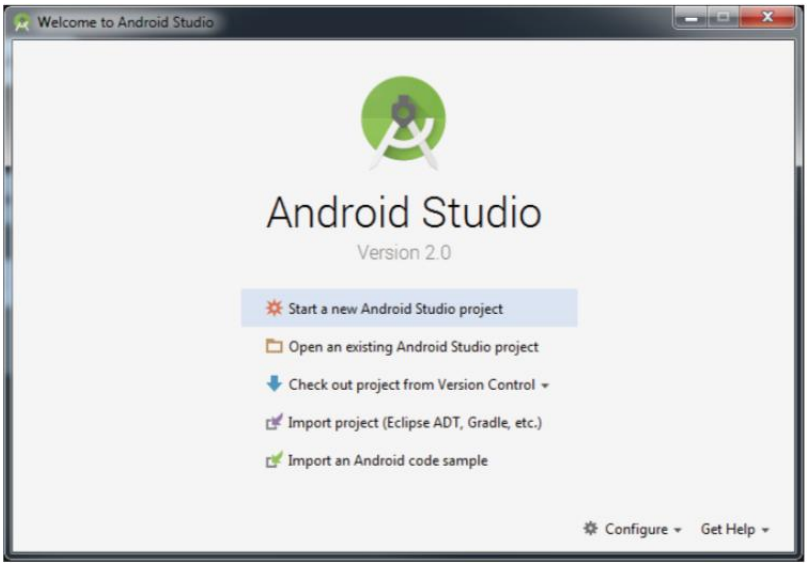

Figura 2.4: Novo projeto no Android Studio - Passo 1 [7]

Em seguida, deve-se nomear a aplicação, bem como escolher o domínio, o diretório onde os arquivos ficarão e o nome do pacote.

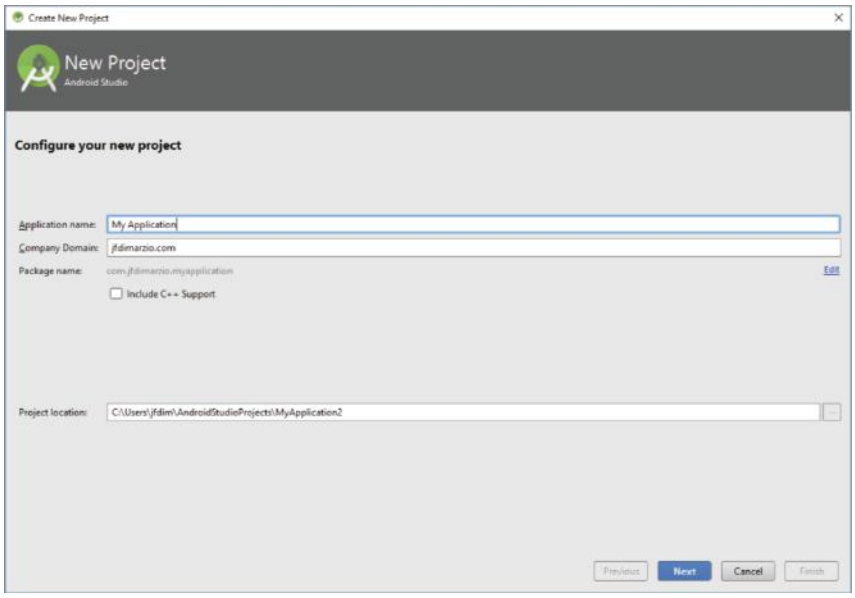

Figura 2.5: Novo projeto no Android Studio - Passo 2 [7]

Após, escolhe-se o SDK mínimo para cada opção que a aplicação será utilizada dentre: smartphones e tablets, wear e etc. O escopo deste trabalho se limita a smartphones e tablets. Quanto menor for a versão do SDK escolhido, maior será o número de dispositivos que suportarão a aplicação.

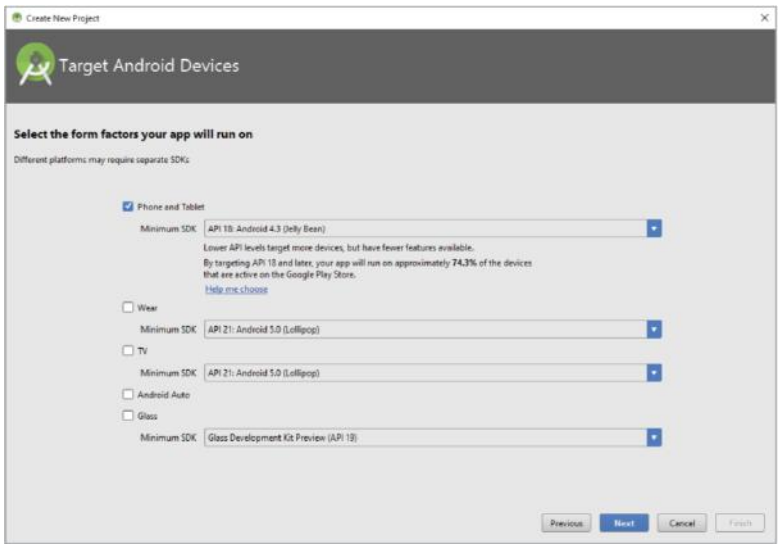

Figura 2.6: Novo projeto no Android Studio - Passo 3 [7]

Feito isso, aparecerão algumas opções de Activity para serem adicionadas à aplicação. O modelo mais básico é o "Empty Activity" e este modelo deve ser o escolhido num primeiro momento até que se obtenha um maior conhecimento da plataforma.

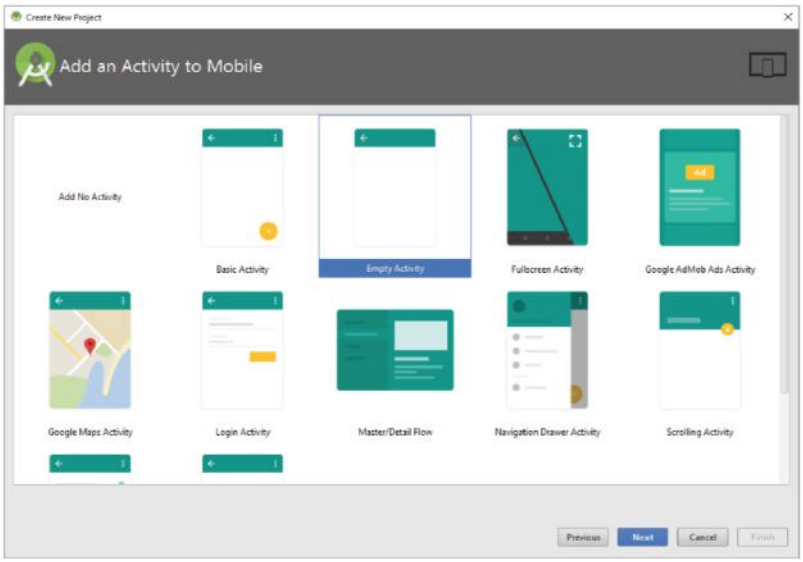

Figura 2.7: Novo projeto no Android Studio - Passo 4 [7]

A IDE será iniciada e alguns arquivos serão criados, podendo ser selecionados no navegador à esquerda.

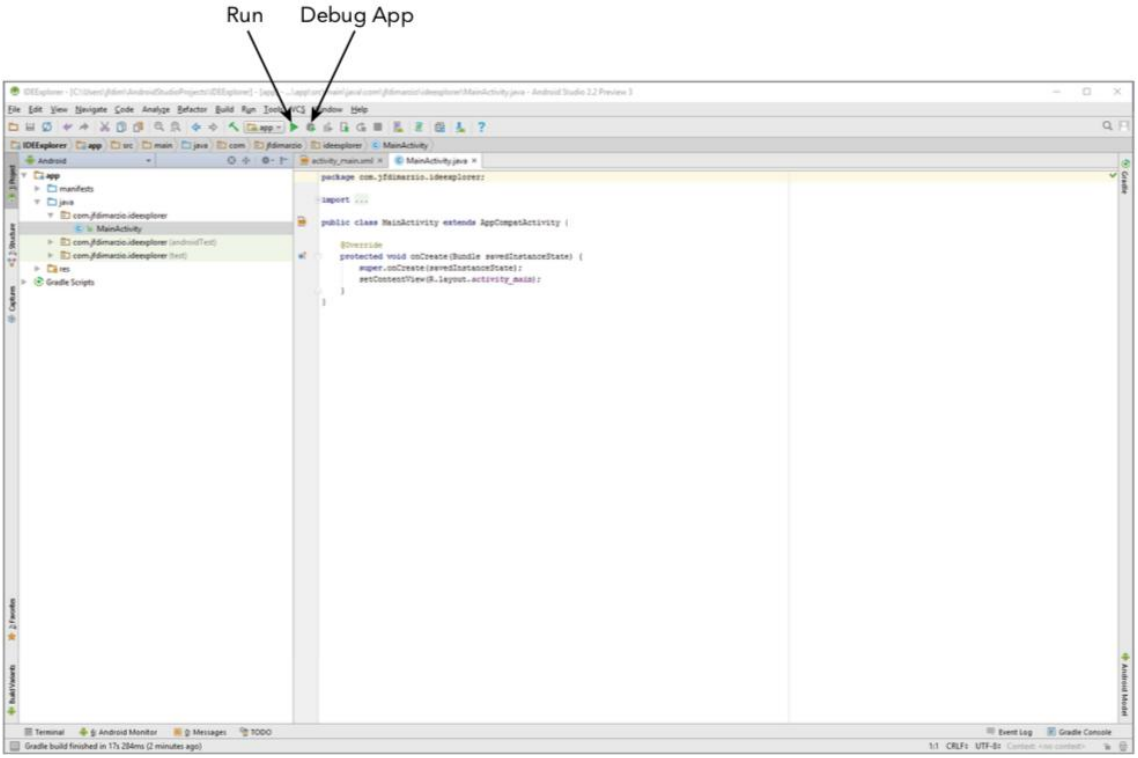

Figura 2.8: IDE Android Studio [7]

O código pré-escrito cria uma Activity vazia. Para visualizá-la basta clicar no botão "Run App". É possível rodar as aplicações tanto em dispositivos físicos, quanto em dispositivos virtuais.

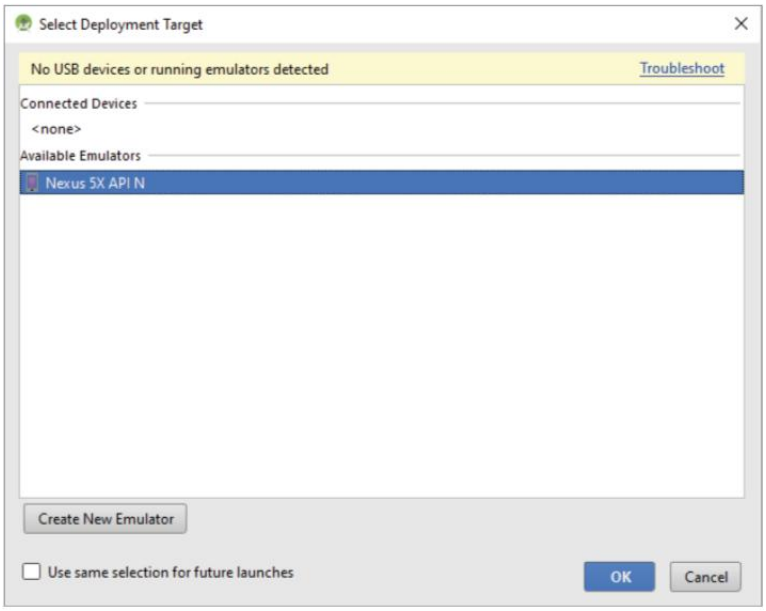

Figura 2.9: Selecionar dispositivo no Android Studio [7]

Para criar um dispositivo virtual, é necessário clicar em "Create New Emulator" escolher o hardware que o dispositivo irá simular.

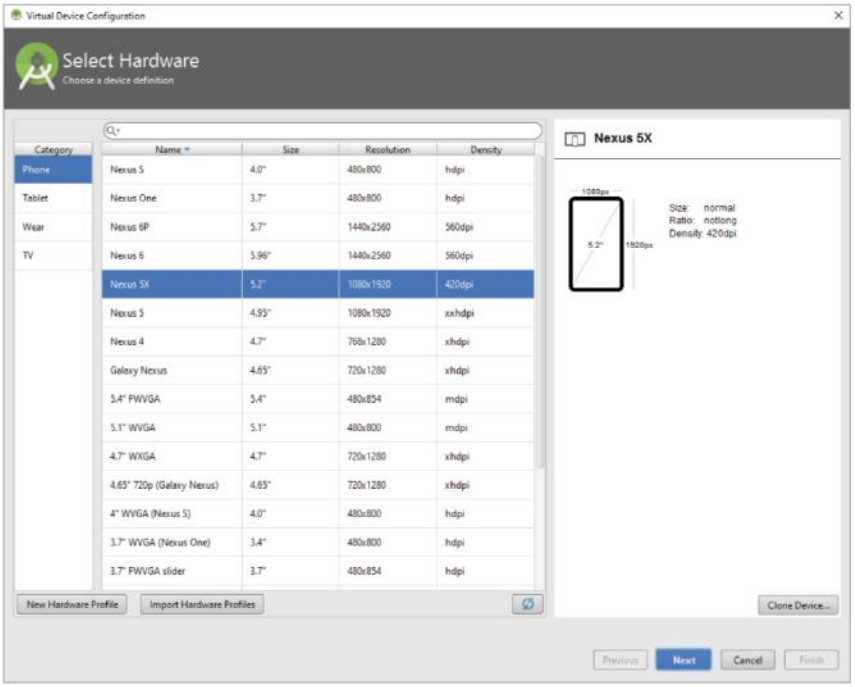

Figura 2.10: Criar dispositivo virtual - Passo 1 [7]

Em seguida, escolhe-se a versão do Android que rodará na máquina.

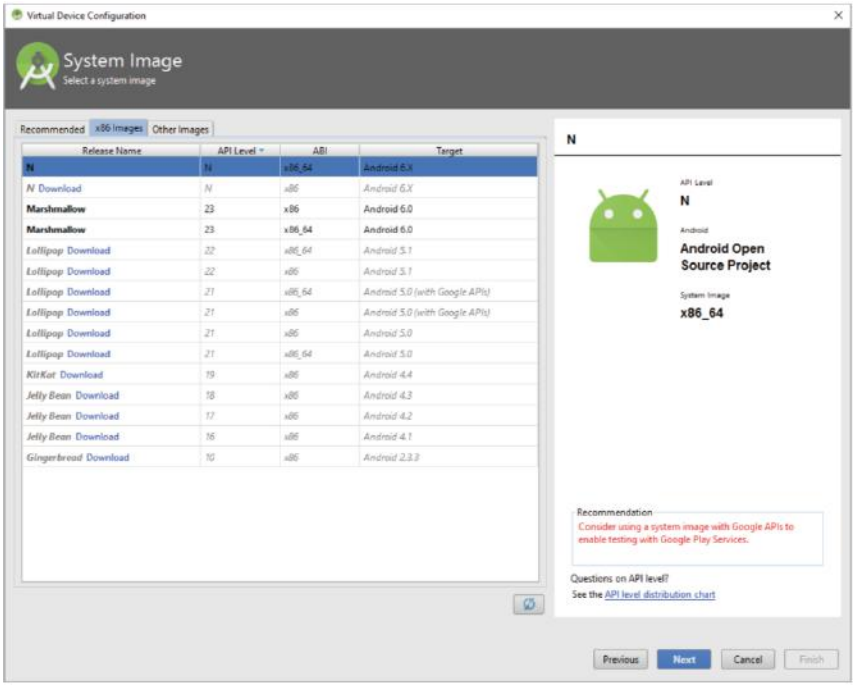

Figura 2.11: Criar dispositivo virtual - Passo 2 [7]

E, por último, setar algumas configurações como o nome do dispositivo virtual, a orientação inicial do dispositivo e se o frame, ou borda, deve ser visível.

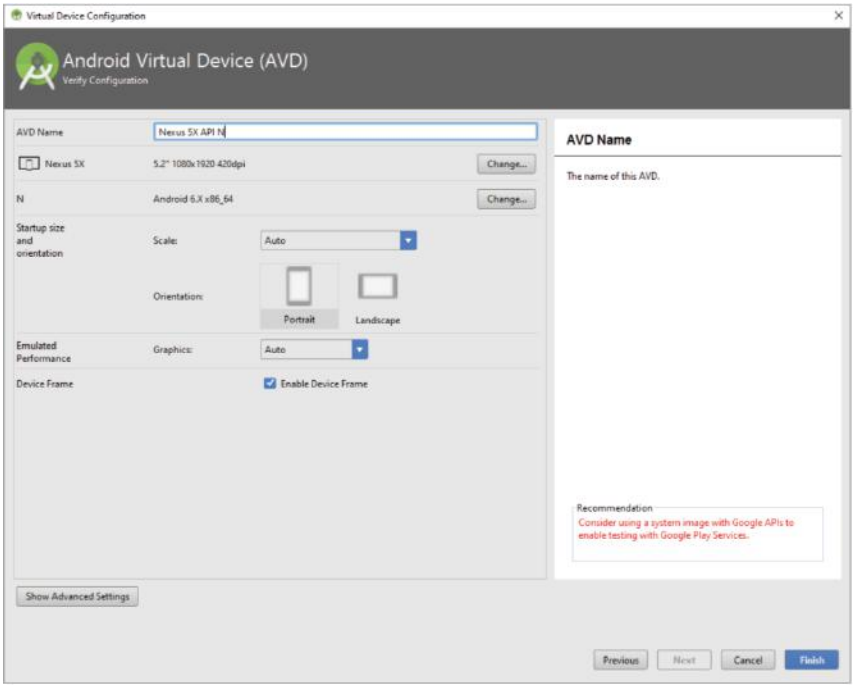

Figura 2.12: Criar dispositivo virtual - Passo 3 [7]

Compilando o exemplo criado em [7] num dispositivo virtual, obteve-se o seguinte resultado:

Figura 2.13: Exemplo com Empty Activity [7]

2.6 Criação de projetos para iOS

O XCode é a IDE oficial para desenvolvimento de aplicativos para iOS.

Depois de fazer o download e de instalar o XCode (recomenda-se o fazer pela Mac App Store), deve-se escolher a opção "Create a New XCode Project".

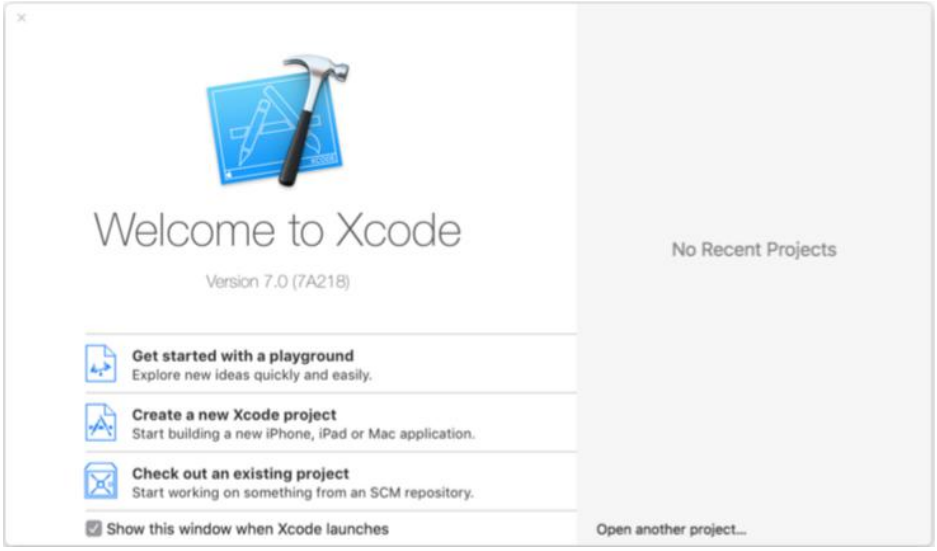

Figura 2.14: Novo projeto no XCode - Passo 1 [4]

Após, algumas opções de template serão apresentadas. Até que se obtenha um

maior conhecimento da IDE e da linguagem, é recomendável escolher a opção “Single View App”.

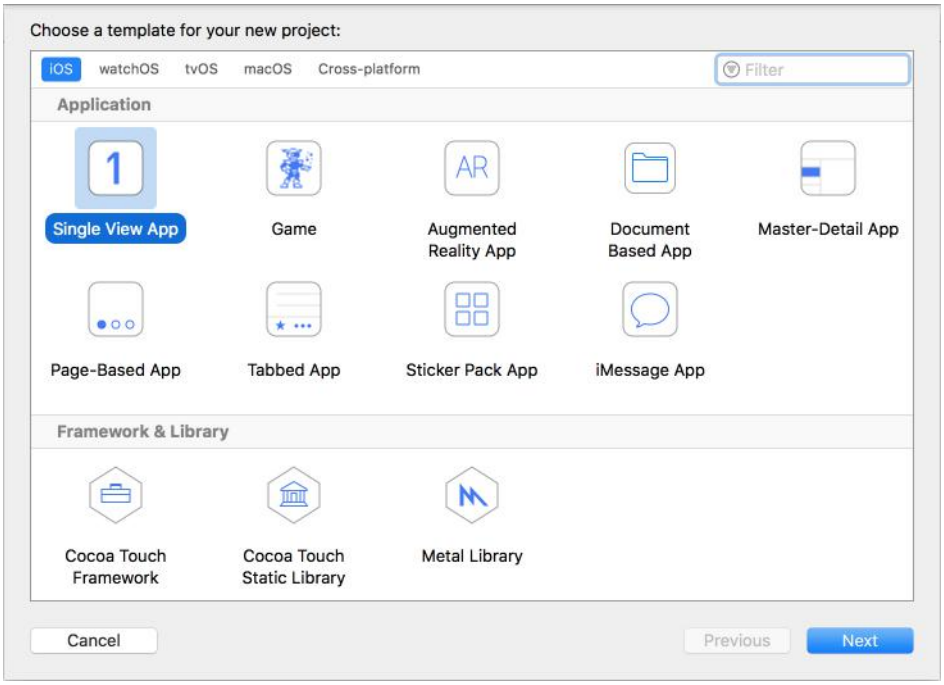

Figura 2.15: Novo projeto no XCode - Passo 2

Sequencialmente, deve-se preencher algumas informações como o nome da aplicação, o nome da companhia, um identificador para a companhia, a linguagem a ser utilizada e para qual dispositivo a aplicação será desenvolvida.

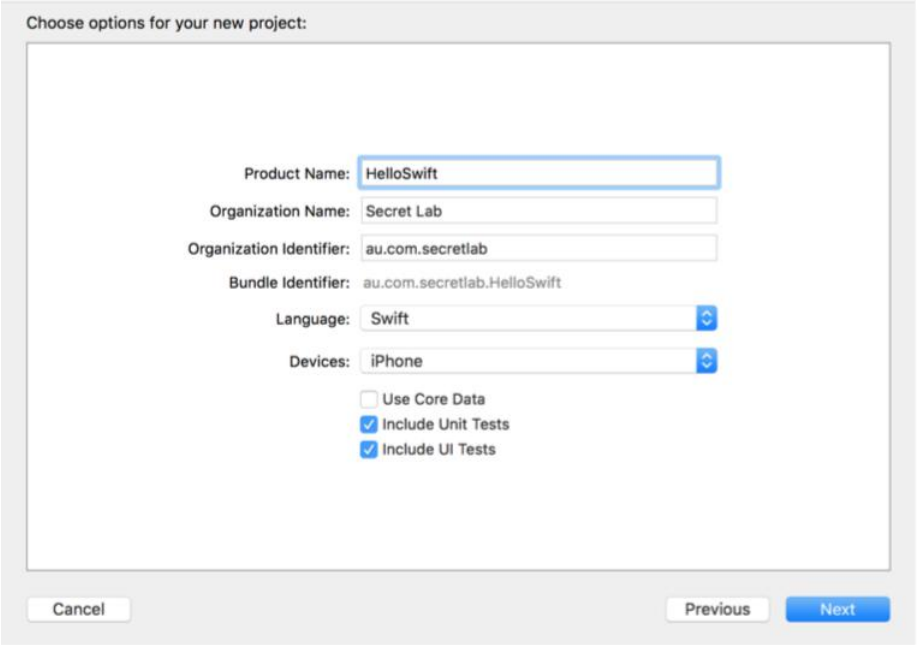

Figura 2.16: Novo projeto no XCode - Passo 3 [4]

De forma similar ao Android, alguns arquivos serão criados no projeto. A classe

equivalente à Activity do Android no iOS é a classe UIViewController do iOS SDK. O projeto criado contêm apenas um UIViewController em branco.

Utilizando o XCode também é possível selecionar um dispositivo físico ou virtual para rodar a aplicação.

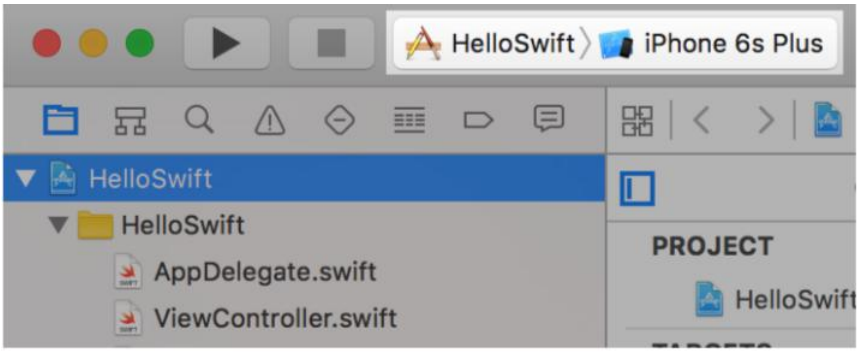

Figura 2.17: Selecionar dispositivo no XCode [4]

A imagem a seguir mostra o resultado que se obtém compilando um projeto com um UIViewController em branco num dispositivo virtual.

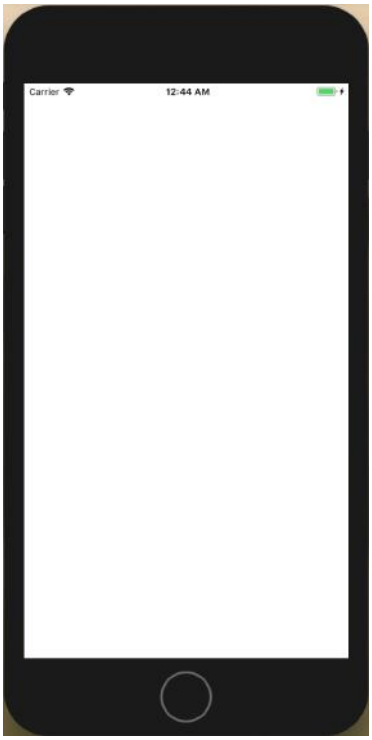

Figura 2.18: Exemplo com UIViewController único em branco

Capítulo 3

Desenvolvimento Android

3.1 Activity e Fragment

Em sistemas operacionais como Windows ou Linux, um usuário pode ter diversas aplicações rodando e visíveis ao mesmo tempo [3]. Diferentemente, no sistema Android existe apenas uma aplicação rodando no primeiro plano [3]. Um usuário pode invocar outras aplicações ou outras telas da mesma aplicação a partir da aplicação visível e todos esses programas e telas são gravados no Activity Manager possibilitando ao usuário retornar à tela anterior clicando no botão de retorno [3].

A classe base Activity define uma série de eventos que governam seu ciclo de vida são eles [7]:

- onCreate(): chamado quando a Activity é criada;
- onStart(): chamado quando a Activity se torna visível para o usuário;
- onResume(): chamado quando a Activity começa a interagir com o usuário;
- onPause(): chamado quando a Activity está sendo pausada e uma outra Activity está sendo resumida ou criada;
- onStop(): chamado quando a Activity não está mais visível para o usuário;
- onDestroy(): chamado quando a Activity está sendo destruída pelo sistema;
- onRestart(): chamado quando a Activity foi parada e está iniciando novamente.

A Figura 3.1 mostra em forma de diagrama o ciclo de vida de uma Activity.

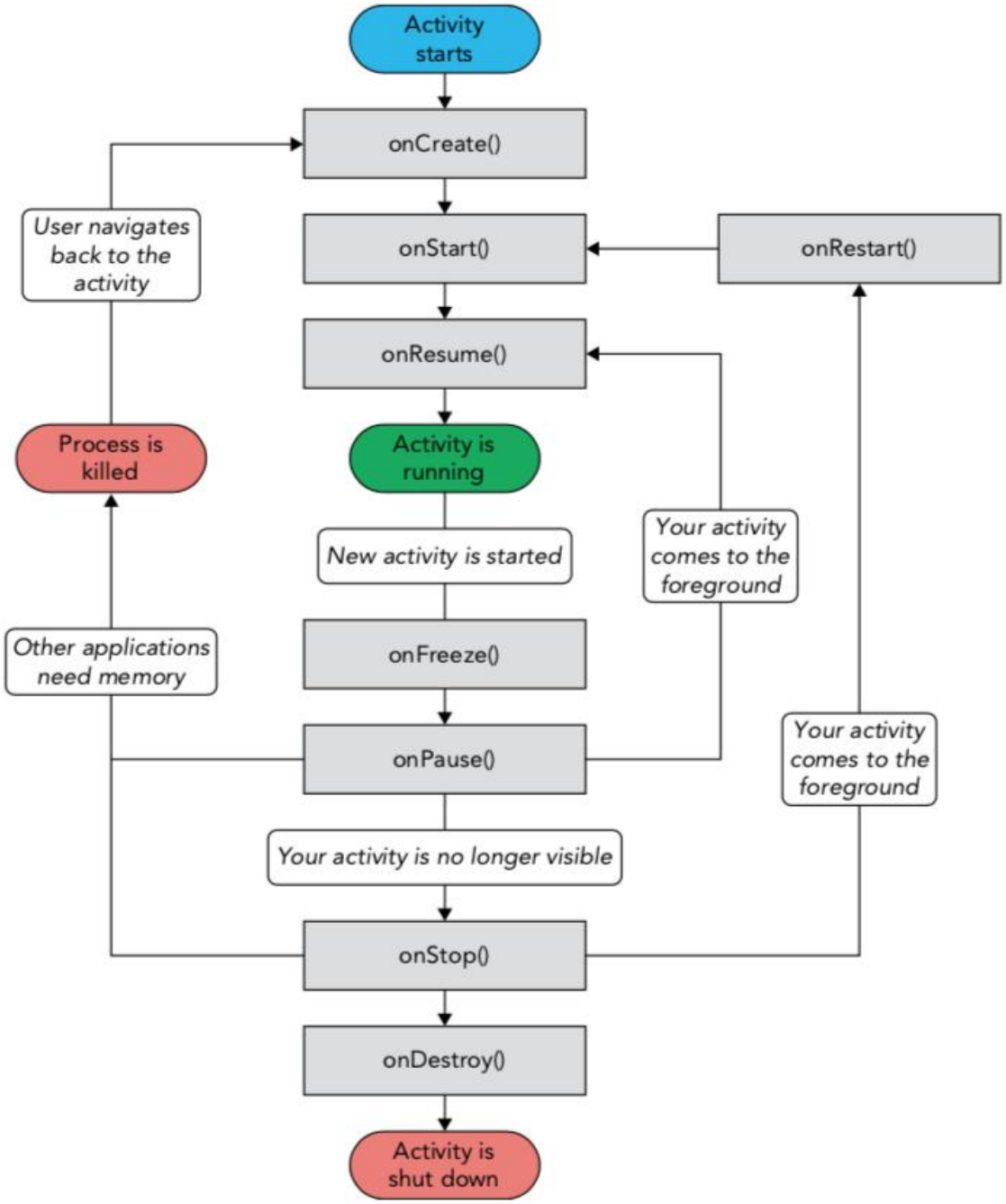

Figura 3.1: Ciclo de vida de uma Activity do Android [7]

O Código-Fonte 3.1 reproduzido de [7] demonstra como sobrescrever o método onCreate() para que ele se comporte da maneira desejada pela MainActivity, ou seja, apenas carrega a UI do arquivo de layout *activity_main.xml* localizado na pasta */res/layout* do projeto.

MainActivity.java

```
package com.jfdimarzio.chapter1helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        //set the layout from the res/layout folder
        setContentView(R.layout.activity_main);
    }
}
```

Código-Fonte 3.1: Classe MainActivity - Exemplo de implementação básica

Conforme mencionado, uma Activity ocupa a tela toda do dispositivo o que pode resultar em uma interface difícil de ser projetada para dispositivos com telas de tamanhos diferentes ou quando o usuário utiliza a aplicação em orientações diferentes. Uma UI ideal para um Smartphone provavelmente não ficará tão agradável em ambas as orientações do dispositivo ou na tela de um Tablet. Pensando nisso, a partir do Android 3.0 para Tablets e do Android 4.0 para Smartphones, também existe suporte para Fragments, que podem ser imaginados como Activities em "miniatura"e que, quando agrupadas, formam uma Activity [7].

Um Fragment representa o comportamento ou uma parte da interface do usuário em uma Activity [9]. É possível combinar vários Fragments em uma única Activity para compilar uma UI de vários painéis e reutilizar um Fragment em diversas Activities [9]. Um Fragment funciona como uma seção modular de uma Activity, que tem o próprio ciclo de vida, recebe os próprios eventos de entrada e que pode ser adicionado ou removido com a Activity em execução (uma espécie de "subActivity" que pode ser reutilizada em diferentes Activities) [9].

Na figura Figura 3.2, quando desenvolvida para um Tablet, a aplicação possui uma Activity (A) com dois Fragments (A e B). Já quando desenvolvida para um Smartphone, a aplicação possui duas Activities (A e B), cada uma delas com um Fragment (A e B, respectivamente).

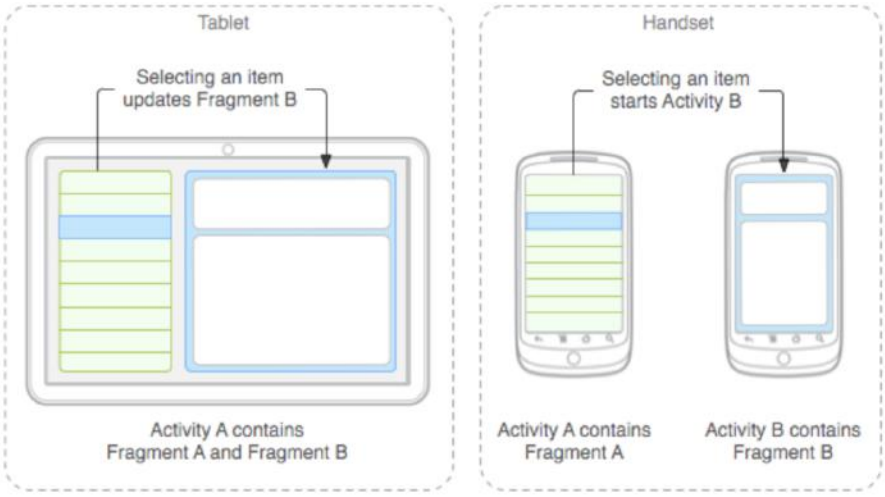

Figura 3.2: Exemplo de uso de Fragments [9]

No caso do Tablet, selecionar um item da lista do Fragment A, causa uma atualização nos dados do Fragment B, sem sair da mesma Activity. Já no Smartphone, selecionar um item da lista do Fragment A invoca a Activity B, que contém um Fragment B. Como consequência, a mesma aplicação oferece a mesma funcionalidade com UIs diferenciadas, dependendo do dispositivo.

Fragments possuem seu próprio ciclo de vida, conforme mostra a Figura 3.3.

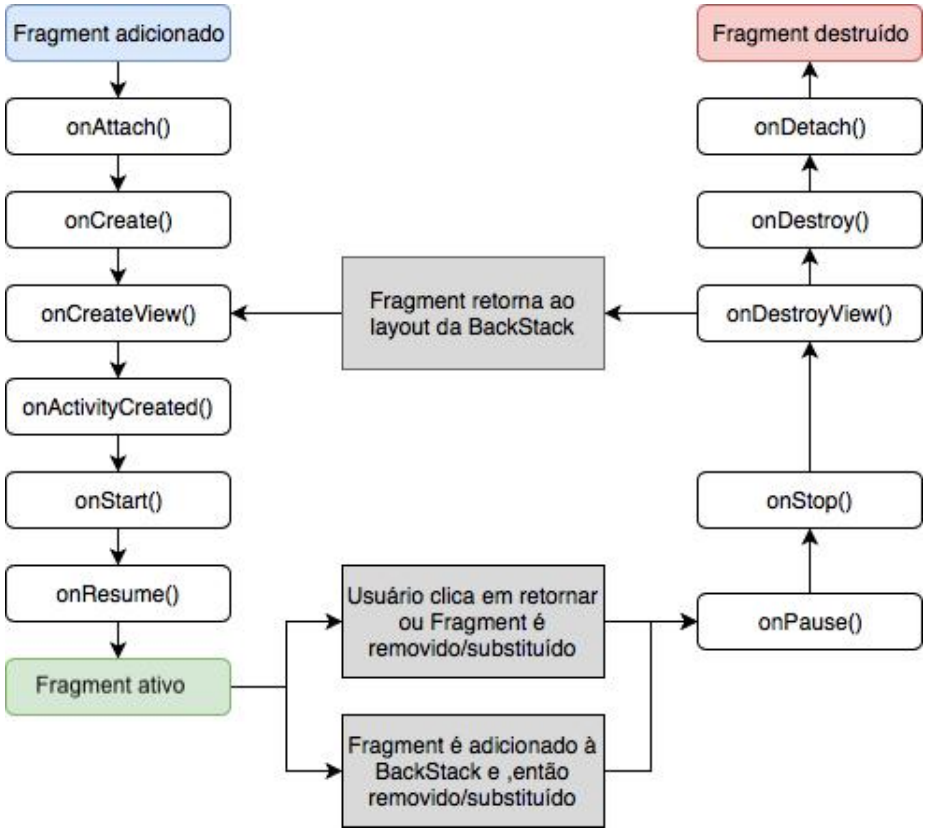

Figura 3.3: Ciclo de vida de um de um Fragment do Android [9]

Uma Activity (ou um Fragment) pode conter Views que são, por exemplo, botões, textos, caixas de texto, etc e ViewGroups, que são um ou mais Views agrupados, por exemplo, RadioGroup, ScrollView, Layout, etc [7]. Layouts usualmente são utilizados para agrupar e ordenar Views em uma UI [7].

Os principais tipos de Layout do SDK Android atual são:

- FrameLayout: desenvolvido para conter apenas um único View adicionado ao canto superior esquerdo do Layout, porém não existe limitação para adição de mais Views (sempre adicionados usando a mesma lógica);
- ScrowView: é um tipo especial de FrameLayout que permite deslizar pela tela;
- LinearLayout: ordena os Views em uma única linha ou em uma única coluna;
- GridLayout: agrupa os Views em linhas e colunas;
- RelativeLayout: possibilita ordenar as Views uma em relação à outra.

Há duas possibilidades para criar as UIs em aplicações Android: proceduralmente ou via XML de layout.

O exemplo do começo deste capítulo carrega a UI a partir do XML *activity_main.xml*. Para adicionar Views à UI, basta modificar este XML da forma desejada. A figura 3.4 mostra o resultado do exemplo que a segue modificando o arquivo *activity_main.xml* para mostrar alguns Views [7].

Figura 3.4: Exemplo de UI Android com alguns Views [7]

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    android:id="@+id/RLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <TextView
        android:id="@+id/lblComments"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, Android!"
        android:layout_alignParentTop="true"
        android:layout_alignParentStart="true" />
    <FrameLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignStart="@+id/lblComments"
        android:layout_below="@+id/lblComments"
        android:layout_centerHorizontal="true">
        <ImageView
            android:src="@mipmap/butterfly"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </FrameLayout>
</RelativeLayout>
```

Código-Fonte 3.2: XML de layout da MainActivity - Exemplo de implementação básica

Neste exemplo, a UI contém um RelativeLayout que, por sua vez, possui um TextView com o texto "Hello, Android!" e um FrameLayout "pai" de um ImageView responsável por carregar a imagem *butterfly.png* contida na pasta *res/mipmap-hdpi*.

O próximo passo para desenvolver uma aplicação Android é entender como navegar entre Activities e, também, entre Fragments.

3.2 Intent e FragmentManager

Para alternar para outra Activity usa-se a classe Intent e é necessário que o projeto possua, ao menos, duas Activities.

A criação de uma Activity é simples: basta seguir o caminho: File, New, Activity e escolher o tipo de Activity desejada. Novamente, recomenda-se começar com uma "Empty Activity". Em seguida, deve-se nomear a Activity e manter a opção de criar automaticamente um XML de layout selecionada, caso assim desejar.

No Código-Fonte 3.3, modificado de [7], foi adicionado um View do tipo Button ao Layout da MainActivity, definindo o método onClick para a atividade de clique no botão.

MainActivity.java

```
package com.jfdimarzio.usingintent;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClick(View view) {
        Intent i = new Intent("com.jfdimarzio.usingintent.SecondActivity");
        i.putExtra("str1", "This is a string");
        i.putExtra("age1", 25);

        Bundle extras = new Bundle();
        extras.putString("str2", "This is another string");
        extras.putInt("age2", 35);

        i.putExtras(extras);

        startActivity(i);
    }
}
```

Código-Fonte 3.3: Classe MainActivity - Exemplo com Intent

activity_main.xml

```
<?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:layout_width="match_parent"
    android:layout_height="match_parent"
    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.jfdimarzio.usingintent.MainActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Main Activity!"
        android:id="@+id/textView" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Display second activity"
        android:onClick="onClick"
        android:id="@+id/button"
        android:layout_below="@+id/textView"
        android:layout_alignParentStart="true"
        android:layout_marginTop="56dp" />
</RelativeLayout>
```

Código-Fonte 3.4: XML de layout da MainActivity - Exemplo com Intent

A classe MainActivity contém o método onClick() esperado pelo View Button definido no XML do Código-Fonte 3.4. Esse método invoca o método startActivity() da classe Activity passando um objeto do tipo Intent que, por sua vez, é criado recebendo uma SecondActivity. O comportamento esperado desta aplicação é instanciar a MainActivity e o clique em seu botão faz com que a primeira Activity (MainActivity) seja destruída e dê lugar à segunda Activity, SecondActivity. Além disso, o objeto Intent carrega dados do tipo chave-valor da MainActivity para a SecondActivity através dos métodos putExtra() e putExtras().

Já a classe SecondActivity, quando criada, recupera o valor da chave "str2" e mostra esse valor na tela através de uma mensagem da classe Toast, conforme mostra

o Código-Fonte 3.5. É notório que a classe SecondActivity deste exemplo ilustrativo nem necessita de um XML de layout, pois a ideia do exemplo é apenas mostrar o funcionamento da classe Intent e ela pode ficar em branco. Entretanto, não seria difícil utilizar a mesma lógica para criar um botão que retornasse à MainActivity, por exemplo.

SecondActivity.java

```
package com.jfdimarzio.usingintent;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class SecondActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);

        Bundle bundle = getIntent().getExtras();
        Toast.makeText(this, bundle.getString("str2"),
    Toast.LENGTH_SHORT).show();
    }
}
```

Código-Fonte 3.5: Classe SecondActivity - Exemplo com Intent

O processo é parecido com Fragments. Um Fragment, assim como uma Activity, pode ter seu layout definido por um XML. No exemplo a seguir foram definidos dois Fragments (Fragment1 e Fragment2), cada um deles com seu respectivo XML de layout ($fragment1.xml$ e $fragment2.xml$). Os arquivos $.java$ que definem o comportamento de cada um deles são bem parecidos, cada qual criando as classes Fragment1 e Fragment2, que herdam da classe base Fragment do SDK Android. Além disso, as classes devem sobrescrever o método $onCreateView()$ do ciclo de vida do Fragment e carregar dentro do método a respectiva UI de cada XML de layout. O Código-Fonte 3.6 define a classe Fragment2. Conforme mencionado, a classe Fragment1 é bastante similar e seu Código-Fonte será omitido.

Fragment2.java

```
package com.jfdimarzio.fragments;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class Fragment2 extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
    container, Bundle savedInstanceState) {
        //---Inflate the layout for this fragment---
        return inflater.inflate(R.layout.fragment2, container, false);
    }
}
```

Código-Fonte 3.6: Classe Fragment2 - Exemplo com FragmentManager

Este exemplo mostra como alternar entre Fragments dependendo da posição da tela: "em pé" (*portrait*) e "deitada" (*landscape*). Esta alternância é feita através da classe FragmentManager no método onCreate() da MainActivity. É importante notar que ao mudar a orientação do dispositivo, a Activity visível é sempre destruída e recriada e, portanto, o método onCreate() será chamado todas as vezes que o usuário realizar esta ação.

O Código-Fonte 3.7 abaixo calcula a largura e a altura do dispositivo utilizando a classe DisplayMetrics do Android SDK e, caso a largura seja maior que a altura, substitui o conteúdo do View com id "content" por um Fragment1. Caso contrário, a substituição é feita por um Fragment2.

MainActivity.java

```
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        FragmentManager fragmentManager = getFragmentManager();
        FragmentTransaction fragmentTransaction =
    fragmentManager.beginTransaction();

        DisplayMetrics display = this.getResources().getDisplayMetrics();
        int width = display.widthPixels;
        int height = display.heightPixels;
        if (width > height) {
            //---landscape mode---
            Fragment1 fragment1 = new Fragment1();
            // android.R.id.content refers to the content view of the
    activity
            fragmentTransaction.replace(android.R.id.content, fragment1);
        }
        else {
            //---portrait mode---
            Fragment2 fragment2 = new Fragment2();
            fragmentTransaction.replace(android.R.id.content, fragment2);
        }
    fragmentTransaction.commit();
    }
}
```

Código-Fonte 3.7: Classe MainActivity - Exemplo com FragmentManager

Capítulo 4

Desenvolvimento iOS

4.1 UIViewController

Seguindo os passos mencionados no capítulo 2, a aplicação iOS criada deve possuir apenas uma UI em branco. Alguns arquivos são criados durante a criação da aplicação conforme mostra a Figura 4.1.

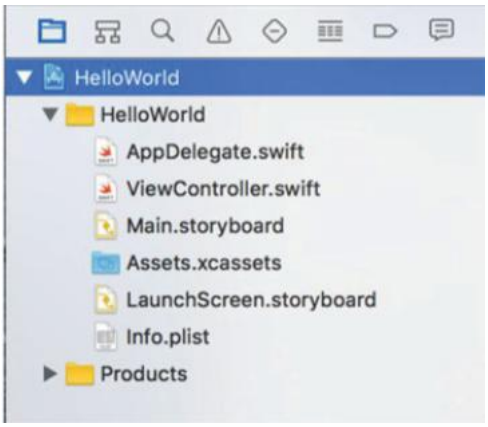

Figura 4.1: Arquivos criados pelo XCode [13]

Dentre eles, há um arquivo chamado Main.storyboard. O Storyboard é a ferramenta de design das UIs da aplicação [13].

A Figura 4.2 mostra a Main.storyboard de uma aplicação com apenas um View em branco.

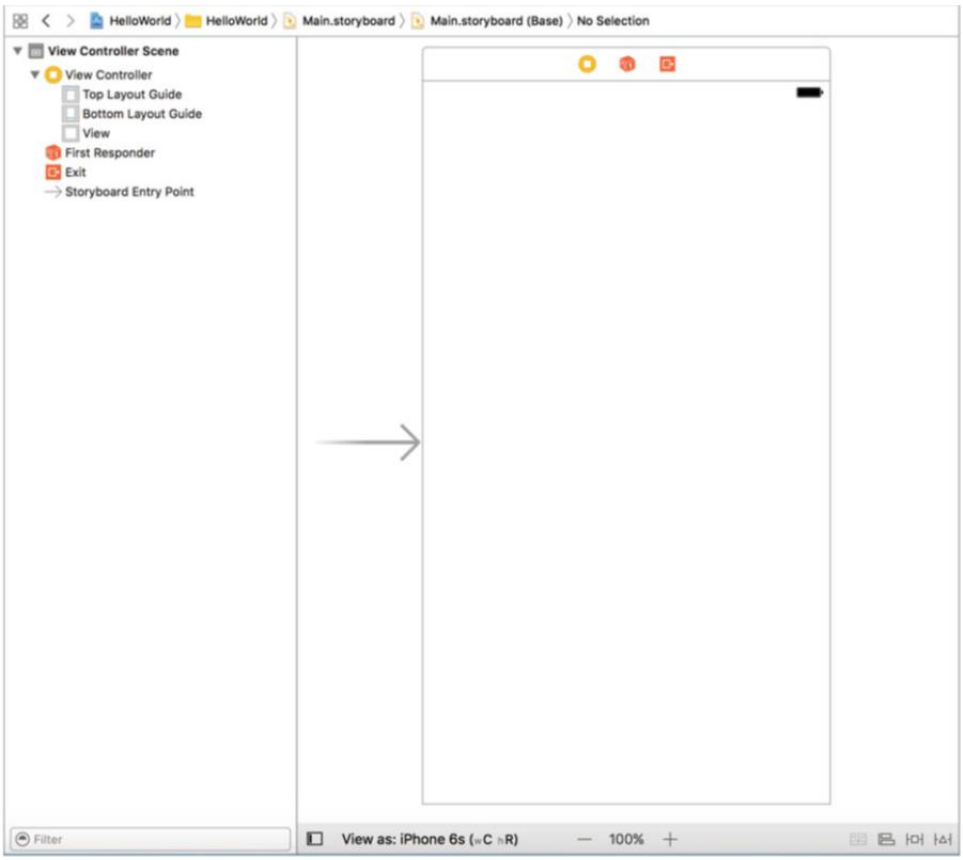

Figura 4.2: Storyboard com apenas um View em branco [13]

No Storyboard é possível alterar o design da UI representada pelo objeto retangular em formato de tela de celular e utilizando-se do Attibute Inspector. Para visualizar o Attibute Inspector, que fica à direita da tela, basta seguir o caminho View, Utilities, View Attribute Inspector. O Attribute Inspector é atualizado com as configurações possíveis para o item selecionado no Storyboard. A Figura 4.3 mostra como modificar a cor de fundo de uma UI e a Figura 4.4 mostra como procurar por UIViews, especificamente um UILabel, que podem ser arrastados para a tela da UI [13].

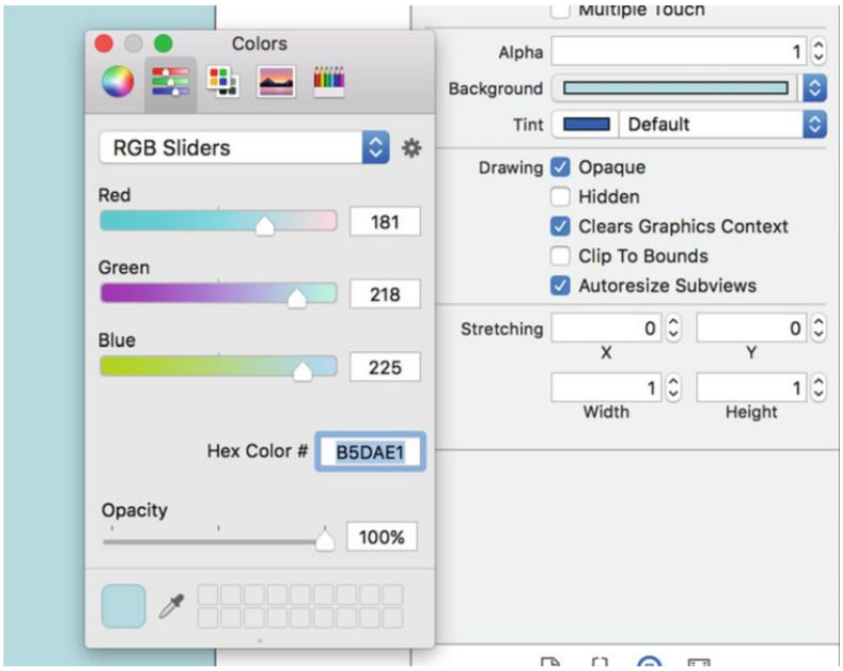

Figura 4.3: Modificar fundo da UI Attribute Inspector [13]

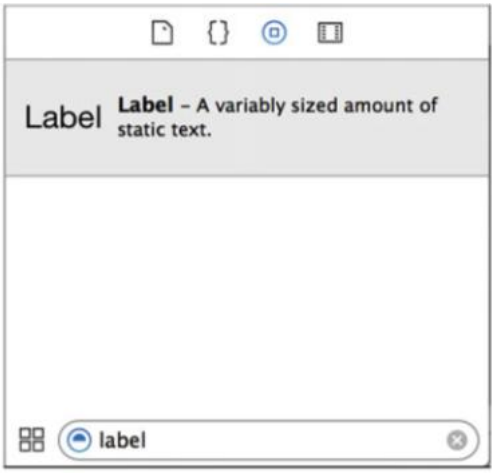

Figura 4.4: Procurando por UILabel no Attribute Inspector [13]

Após arrastar o UIView para a tela do dispositivo, é possível definir sua localização e comportamento na UI atribuindo Constraints a ele, que podem ser em relação ao seu tamanho, ao UIView que o contém, ou a outros UIViews ao seu redor. Para modificar as Constraints do objeto selecionado, basta clicar no botão mostrado na Figura 4.5, neste botão existem algumas opções de Constraints, como por exemplo, as Constraints de margem que, para serem ativadas, deve-se clicar nas linhas vermelhas e, depois de selecionadas, pode-se definir a distância que as margens do UIView em questão terá com relação às margens dos objetos mais próximos ao seu redor (ou que o contém caso não haja nenhum objeto próximo à margem em questão).

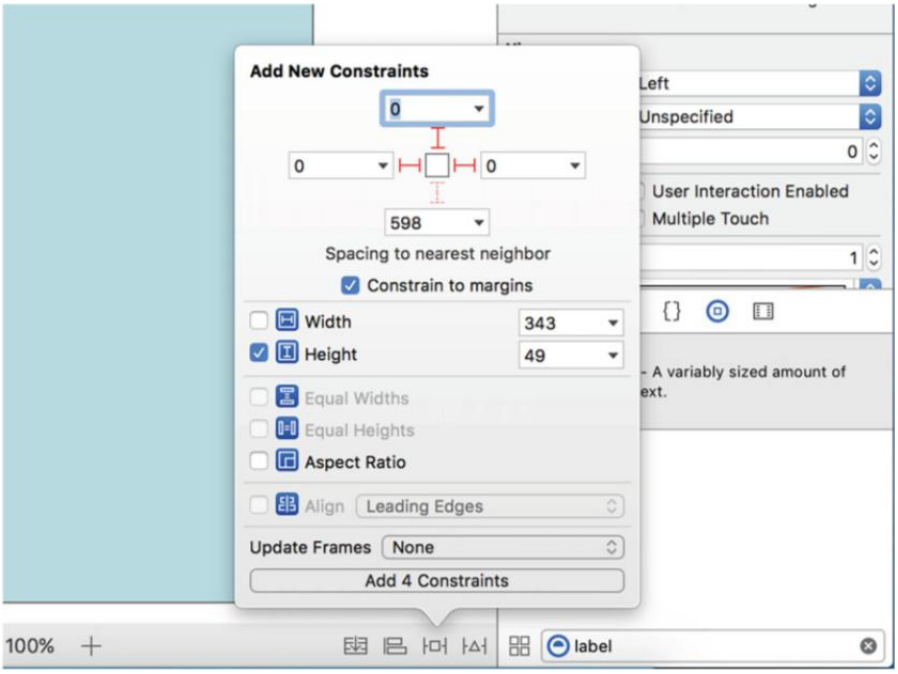

Figura 4.5: Constraints de UIViews [13]

A UI mostrada no Storyboard também é um tipo de UIView, mais especificamente é um UIViewController. A Figura 4.6 mostra como é possível procurar por objetos do tipo UIViewController no Attribute Inspector e arrastá-los para o Storyboard para criar outras UIs.

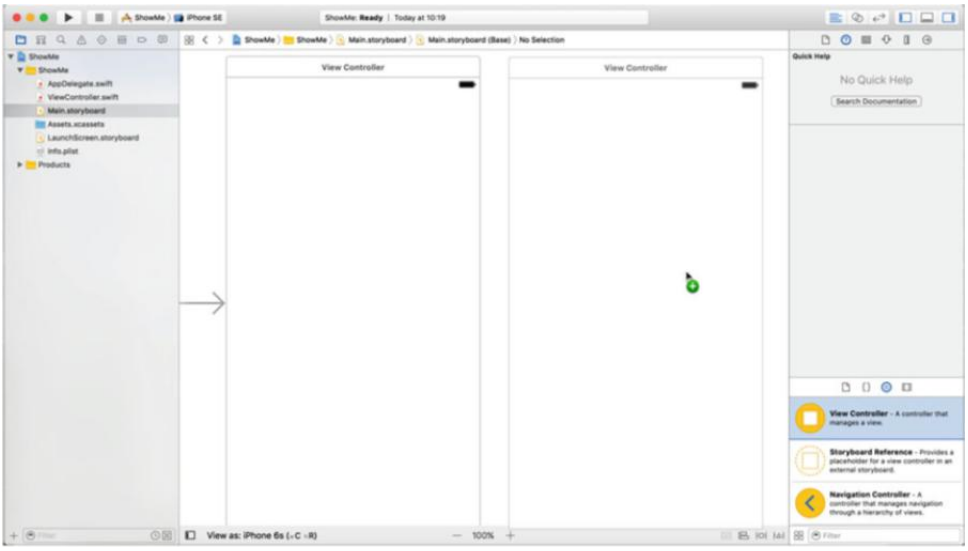

Figura 4.6: Adicionando UIViewController ao Storyboard [13]

Todo UIViewController deve ser relacionado pelo Attribute Inspector com um arquivo Swift, como mostra a Figura 4.7. Este arquivo deve conter uma classe que herda da classe UIViewController do iOS SDK.

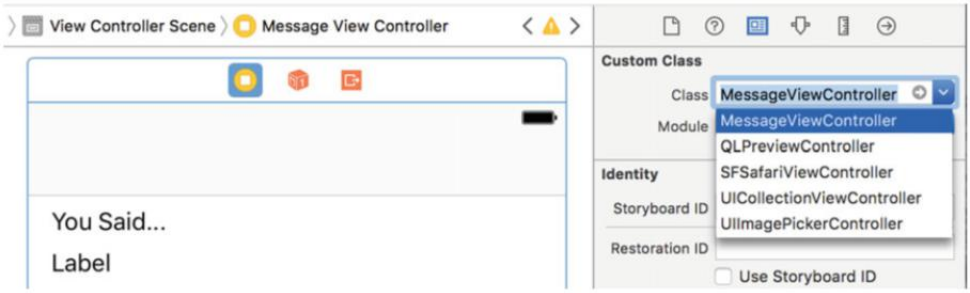

Figura 4.7: Relacionando UIViewController com arquivo Swift [13]

Existem algumas configurações para cada tipo de UIView que podem ser alteradas através do Attribute Inspector como, por exemplo, a cor, a fonte, o texto de um UILabel, porém esses dados ficam estáticos, como acontece com os dados setados nos XMLs de layout do Android. Para modificar os dados de um UIView dinamicamente é necessário conectá-lo ao código Swift que representa o UIViewConstroller "pai" deste UIView. Para isso, é necessário clicar no botão "Show the Assistant editor" que é representado por dois círculos sobrepostos, como mostrado na Figura 4.8.

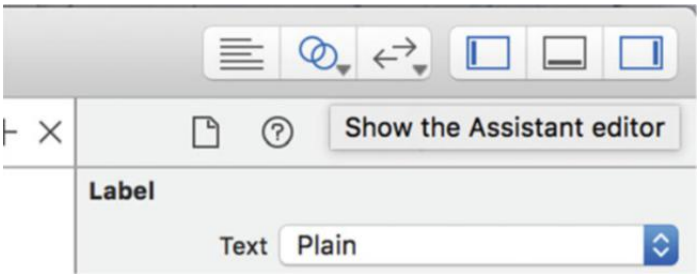

Figura 4.8: Botão para mostrar o Assistant editor no XCode [13]

Este botão dividirá a tela em duas e então é possível arrastar o UIView desejado para o código Swift. Ao ser arrastado, será criada uma variável na classe marcada com uma anotação do tipo @IBOutlet.

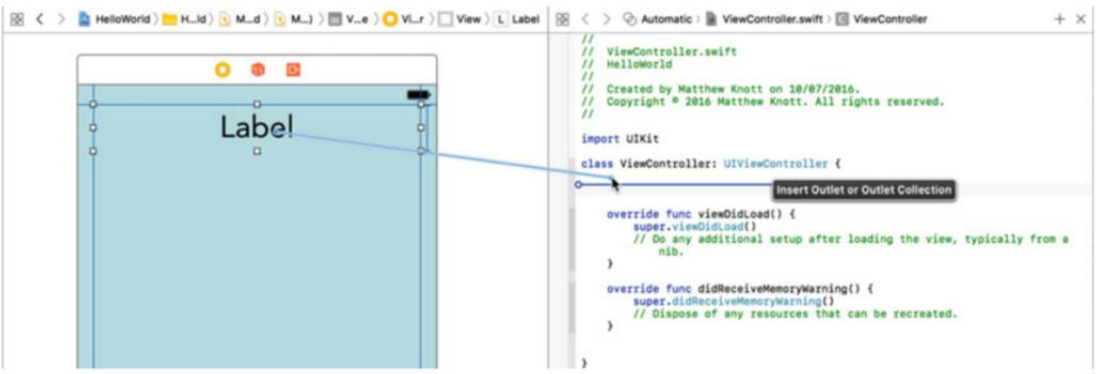

Figura 4.9: Arrastando UIView para UIViewController no XCode [13]

Como acontece com as Activities e Fragments do Android, um UIViewController também possui um ciclo de vida e, consequentemente, métodos que podem ser sobrescritos para definir-se o comportamento desejado em cada fase do seu ciclo de vida.

Os eventos que governam o ciclo de vida de um UIViewController são [1]:

- viewWillAppear(): chamado logo antes de o View aparecer;
- viewDidAppear(): chamado logo depois de o View aparecer;
- viewDidLoad(): chamado quando, além de já ter aparecido, o View tiver sido completamente carregado;
- viewWillDisappear(): chamado logo antes de o View desaparecer;
- viewDidDisappear(): chamado logo após o View desaparecer.

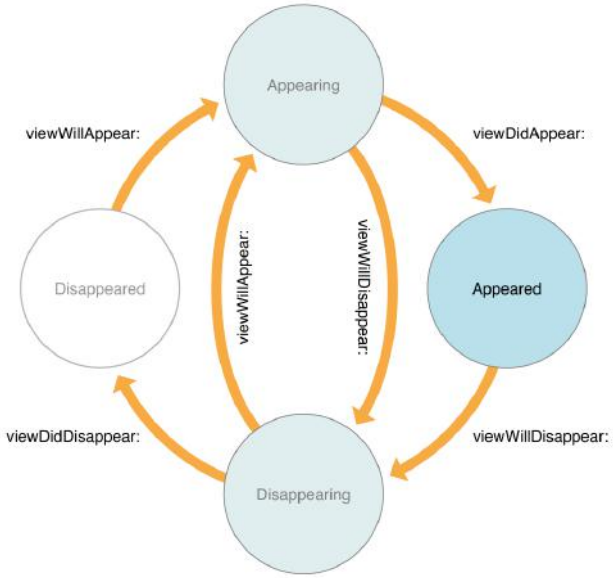

Figura 4.10: Ciclo de vida de um UIViewController [1]

4.2 Segue

A transição entre UIViewControllers é feita através de Segues. Obviamente, é necessário que a aplicação possua, ao menos, dois UIViewConstrollers para poder conter um Segue.

A criação de um Segue também é feita no Storyboard. Para fazê-lo, basta arrastar o UIView do primeiro UIViewController que performará a ação para o segundo ViewController.

A Figura 4.11 mostra como relacionar um UIButton de um UIViewController com um segundo UIViewController. O clique no botão irá performar a transição para o segundo UIViewController.

Figura 4.11: Criando um Segue no XCode [13]

Note que na Figura 4.11 o primeiro UIViewController contém, além do UIButton, um UILabel com o texto "Text to send" e um UITextField.

A Figura 4.7 mostra o segundo UIViewController, nele são adicionados dois UILabels, o superior contendo o texto "You Said..." e o segundo contendo o texto padrão "Label". Este UIViewController está relacionado com uma classe Swift chamada MessageViewController, conforme mostrado na Figura 4.7.

O Código-Fonte 4.1 mostra o código Swift do segundo UIViewController. Neste exemplo, a classe MessageViewController herda da classe UIViewController e, foi criada uma conexão (Outlet) para o UILabel que contém a palavra "Label" mostrado na Figura 4.7 chamado de messageLabel. Além disso, foi criada uma variável do tipo String chamada messageData e, no método viewDidLoad(), carregado após o View tiver sido completamente carregado, o texto do UILabel messageLabel recebe o valor da String messageData.

MessageViewController.swift

```
class MessageViewController: UIViewController {

@IBOutlet weak var messageLabel: UILabel!
var messageData: String?

override func viewDidLoad() {
    super.viewDidLoad()

    // Do any additional setup after loading the view.
    messageLabel.text = messageData
}
```

Código-Fonte 4.1: Classe MessageViewController - Exemplo com Segue

Já a classe relacionada ao primeiro UIViewController, que foi chamada de ViewController, tem seu código Swift modificado de [13], mostrado no Código-Fonte 4.2.

ViewController.swift

```
class ViewController: UIViewController {

    @IBOutlet weak var textToSendField: UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
    }

    override func prepareForSegue(segue: UIStoryboardSegue, sender:
    AnyObject?) {
        let messageController = segue.destination as!
    MessageViewController
        messageController.messageData = textToSendField.text
    }

}
```

Código-Fonte 4.2: Classe ViewController - Exemplo com Segue

Neste exemplo, o UITextField foi associado pelo Storyboard e chamado de textToSendField. O método prepareForSegue() instancia o destino do segue (que é um objeto da classe MessageViewController) e seta o conteúdo da variável messageData com o texto do UITextField textToSendField. Ou seja, com o clique no botão possui a ação de um Segue, a classe ViewController se prepara para esta ação passando o texto do UITextField para a variável messageData do MessageViewController, que ao ser criada, usa o valor desta variável para modificar o texto do UILabel presente nela. O comportamento esperado desta aplicação é mostrado na Figura 4.12.

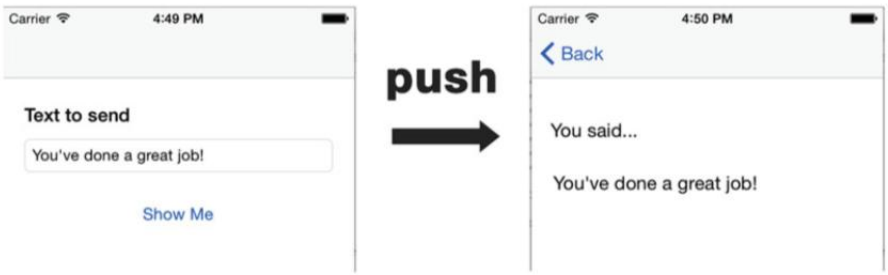

Figura 4.12: Exemplo de aplicação com Segue [13]

Capítulo 5

Projeto dos aplicativos para o RU da UFRJ

5.1 Precedentes

Conforme mencionado no capítulo 1, o Restaurante Universitário da Universidade Federal do Rio de Janeiro enfrenta o problema das filas de espera para o acesso aos seus estabelecimentos.

Figura 5.1: Fila de espera para acesso ao RU do CT

Além disso, as refeições oferecidas pelo Restaurante Universitário devem ser exclusivas à comunidade universitária. Porém o controle manual do acesso, com apresentação de documentos, é propenso a falhas como o empréstimo de documentos, por exemplo, e aumenta o tempo gasto para entrar no restaurante, intensificando o problema das filas.

Pensando nisso, a Decania do Centro de Tecnologia foi pioneira na implementação de um sistema de agendamento online com o objetivo de virtualizar as

filas físicas e de minimizar os problemas causados pela necessidade identificação dos clientes.

O sistema de agendamento online pode ser acessado pelos clientes através do website www.ru.ct.ufrj.br e também possui um website administrativo para os funcionários. Além disso, conta com uma REST API para comunicação com os servidores.

5.2 Aplicativos móveis

5.2.1 Aplicativo Android para funcionários

O aplicativo para os funcionários recebeu o nome de RUAdmin e possui duas Activities (LoginActivity e MainActivity) e vários Fragments que são utilizados pela MainActivity.

Ao iniciar a aplicação, a LoginActivity é criada e seu layout é carregado a partir do XML *activity_login*. Caso haja dados de login salvos no dispositivo, esses dados são colocados nos campos respectivos. É possível alternar entre o ambiente de desenvolvimento e o de produção e há também a possibilidade de salvar ou não os dados.

Todos os métodos de comunicação com a API foram colocados numa classe estática chamada API.

Figura 5.2: LoginActivity do RUAdmin

Quando o usuário clica no botão “Enviar”, o método estático API.getToken()

é chamado e, caso a resposta do servidor seja recebida com sucesso, os dados são salvos se a opção "Lembrar dados" estiver selecionada além de ser feita a alternância para a classe MainActivity através de um Intent.

O método getToken() da classe API merece uma explicação um pouco mais detalhada: do lado do servidor, quando a rota relativa a esse método recebe uma requisição, ela responde não somente com uma resposta HTTP, mas também com uma mensagem push contendo um token JWT. A aplicação Android implementa um serviço de recebimento de mensagens push que, ao receber um push do tipo JWT, guarda o token no dispositivo.

Existe um outro método na classe API que verifica o token JWT guardado no dispositivo chamado checkToken().

Os métodos getToken() e checkToken() invocam um ao outro em um loop até que o token guardado no dispositivo seja verificado ou até que um número fixo de chamados à API tenha sido feito.

Todos os outros métodos da classe API, inicialmente, chamam o método checkToken() e só prosseguem caso a resposta deste método seja positiva.

O fluxo de validação do token JWT é mostrado na Figura 5.3.

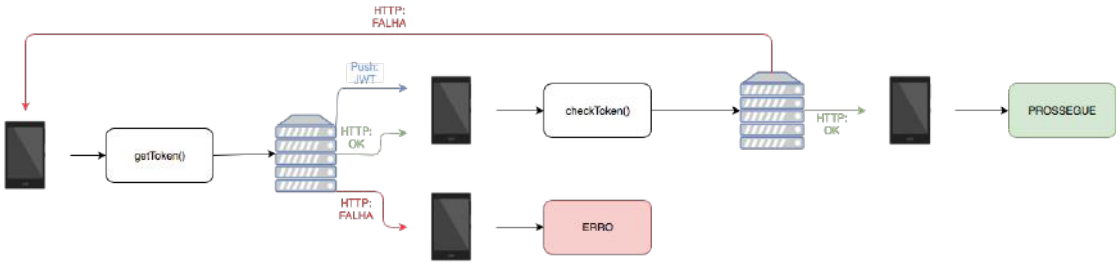

Figura 5.3: Fluxo de validação do token JWT

Uma vez que o método getToken() tenha sido verificado na LoginActivity, a MainActivity é criada. Ela contém a maioria das funcionalidades da aplicação e, além de herdar da classe AppCompactActivity (um tipo de Activity), ela também implementa a interface NavigationView.OnNavigationItemSelectedListener do Android SDK, responsável pelas interações com o menu lateral.

O método onCreate() da MainActivity carrega a UI a partir do XML de layout *activity_main* que contém um FrameLayout e um NavigationView (layout do menu).

Em seguida, ele anexa ao seu conteúdo um Fragment chamado InitialFragment, cujo conteúdo é composto apenas por alguns textos de boas-vindas e algumas imagens.

Neste método também é definido o comportamento do botão "Sair", que utiliza um Intent para retornar para a LoginActivity.

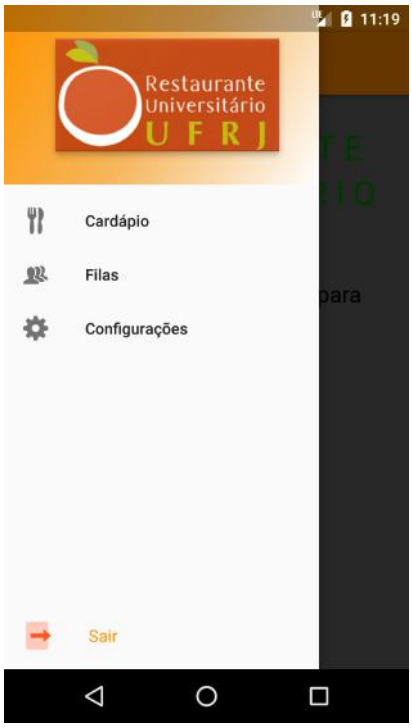

Figura 5.4: Menu lateral do app RUAdmin

Figura 5.5: Tela de início (FragmentInitial) do app RUAdmin

Pelo fato de a MainActivity implementar a interface OnNavigationItemSelectedListener, ela deve sobrescrever os métodos onBackPressed() e onNavigationItemSelected() cujas funções são bem intuitivas: a primeira define a ação que acontece quando o usuário clica no botão "voltar" do seu dispositivo e a segunda define a ação ao selecionar um item do Menu. As implementações correspondentes funcionam da seguinte forma: a primeira já vem implementada pelo SDK para retornar à tela anterior, porém é necessário identificar o Fragment visível na tela e marcar o item do menu correspondente como selecionado, ou o item selecionado do Menu continuará marcado com a última seleção do usuário. Já a segunda faz o oposto, ou seja, cria um Fragment correspondente ao item selecionado e utiliza o FragmentManager para anexar este Fragment na tela.

Ao clicar no item "Filas" no menu lateral, conforme mencionado anteriormente, o método onNavigationItemSelected() da MainActivity utiliza um objeto do tipo FragmentManager para anexar um Fragment, que nesse caso é do tipo QueuesFragment, ao conteúdo da MainActivity.

Ao criar um QueueFragment chamadas ao método estático API.getQueues() são realizada a cada 10 segundos e enquanto uma resposta não é recebida um ProgressBar fica visível. Em caso de resposta positiva do servidor, esse método retorna um ArrayList de objetos do tipo Queue, criado a partir do JSONArray retornado pelo servidor. Essa lista é utilizada para popular um ListView do Android SDK, que utiliza um QueueAdapter.

A classe QueueAdapter é responsável por definir tanto o layout de cada item do ListView associado, bem como implementar as funcionalidades dos itens internos a esse layout.

Cada item do ListView em questão possui tanto as informações da fila virtual do RU quanto três botões: um verde para iniciar o atendimento da fila, um azul para editar a fila e um vermelho para excluir a fila.

Caso o JSONArray retorne vazio, uma texto é exibido na tela informando que não há filas abertas no momento.

Além disso, o QueuesFragment possui dois botões no rodapé um roxo para cadastro de novas filas e, outro, para atendimento de filas via QRCode.

Figura 5.6: ProgressBar no QueuesFragment do RUAdmin

Figura 5.7: ListView no QueuesFragment do RUAdmin

Os botões de criação e edição de filas são bem parecidos. O primeiro anexa ao conteúdo da MainActivity um Fragment do tipo QueueRegisterFragment que utiliza o método API.registerQueue() para se comunicar com o servido e o segundo um Fragment do tipo QueueEditFragment que utiliza o método API.updateQueue() para se comunicar com o servidor. Ambos os Fragments possuem um RelativeLayout contendo um ViewFlipper com seis subviews e um RadioGroup, também com seis subviews, todos do tipo RadioButton.

A primeira tela visível ao usuário é composta de um calendário para o funcionário escolher a data de criação ou edição da fila.

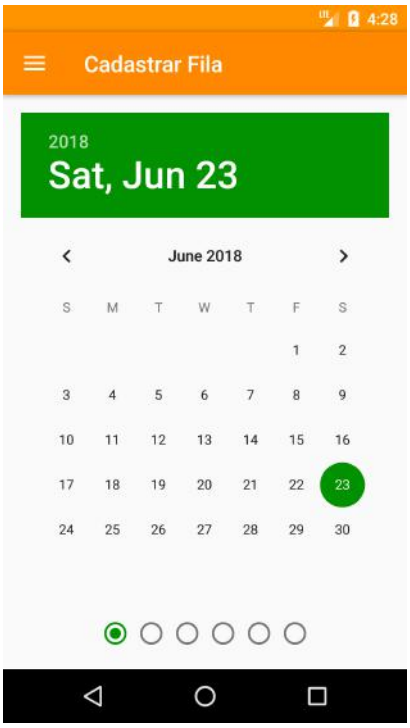

Figura 5.8: Criar ou editar fila no RUAdmin - Passo 1

O clique no RadioButton ao lado, ou o deslizamento do dedo da direita para a esquerda na tela, leva ao segundo item do FlipperView, onde o funcionário deve inserir o cardápio, o restaurante, o status, a refeição, a capacidade do restaurante, a quantidade de tickets, o atraso em minutos e a tolerância também em minutos.

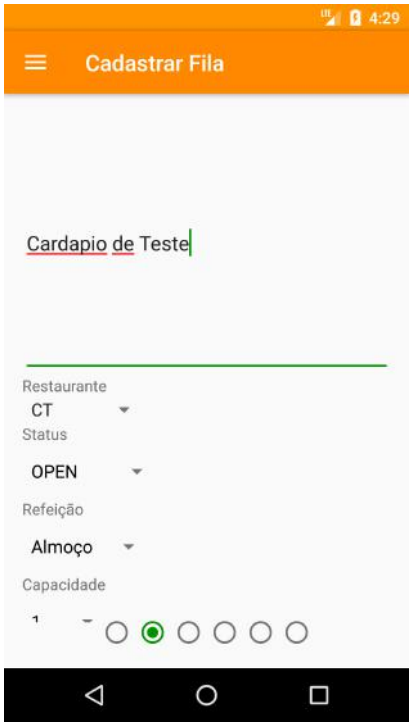

Figura 5.9: Criar ou editar fila no RUAdmin - Passo 2.1

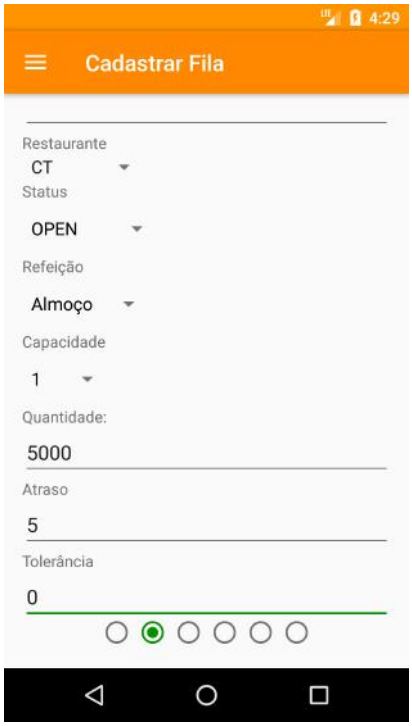

Figura 5.10: Criar ou editar fila no RUAdmin - Passo 2.2

As telas seguintes servem para a escolha dos horários para o início de distribuição dos agendamentos (horário a partir do qual os clientes podem agendar horário), para a abertura da fila e para o fechamento da fila.

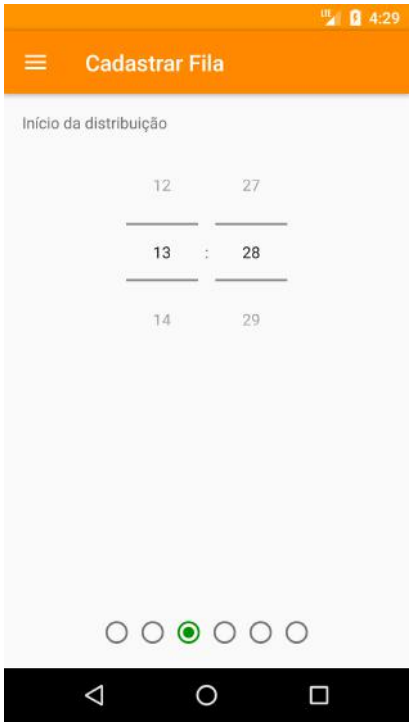

Figura 5.11: Criar ou editar fila no RUAdmin - Passo 3

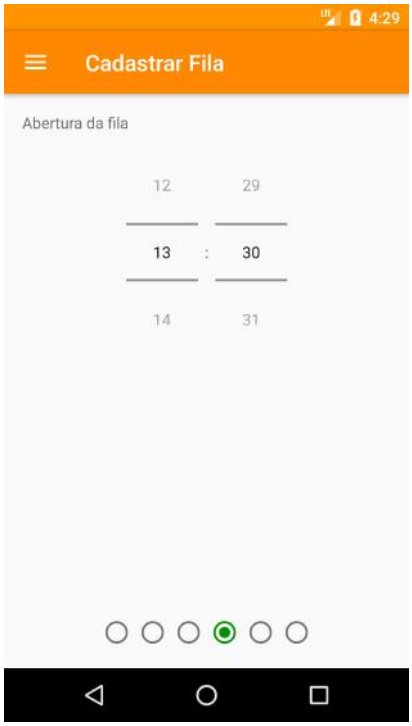

Figura 5.12: Criar ou editar fila no RUAdmin - Passo 4

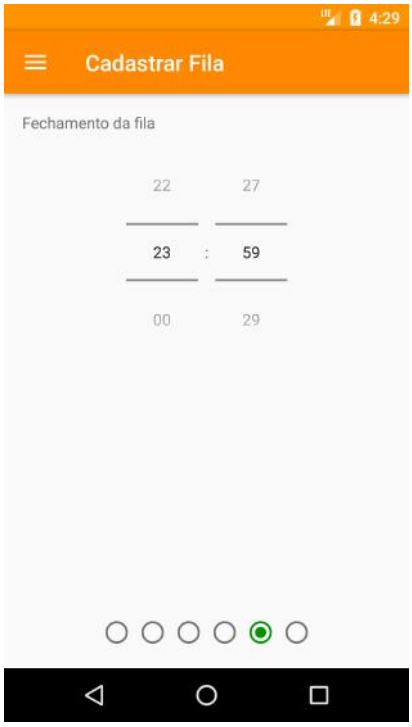

Figura 5.13: Criar ou editar fila no RUAdmin - Passo 5

Figura 5.14: Criar ou editar fila no RUAdmin - Passo 6

Por fim, a última tela exibe um resumo das informações inseridas e o botão "Enviar" que envia as informações pela API para o servidor.

No caso da edição de filas, os dados já vem preenchidos de acordo com o que os dados salvos no servidor e os horários de abertura e de fechamento da fila são imutáveis.

Após finalizar o cadastro ou edição da fila, a aplicação utiliza o FragmentManager para anexar a QueuesFragment novamente e uma mensagem Toast é exibida informando sucesso ou o erro.

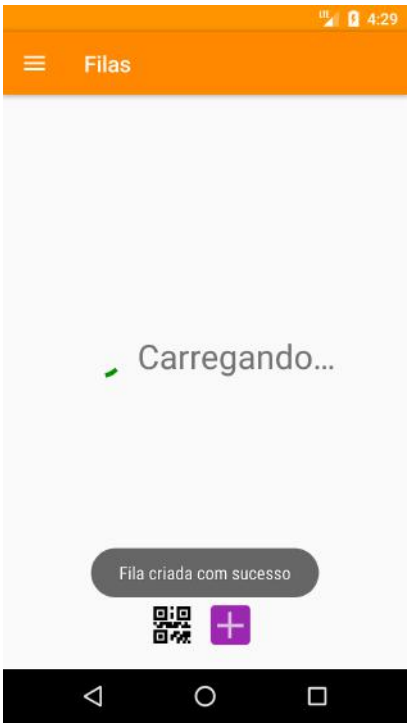

Figura 5.15: Criar ou editar fila no RUAdmin - Confirmação

Uma terceira funcionalidade com relação às filas é a exclusão. Clicando no botão vermelho da fila correspondente, abre-se um AlertDialog para confirmação da exclusão. Cado o usuário optar pela opção "Sim", o método API.updateQueue() é chamado atualizando o status da fila para "CLOSED". Assim como nos demais casos, uma mensagem Toast é exibida mostrando a resposta do servidor informando sucesso ou, caso ocorra algum erro, qual foi o erro.

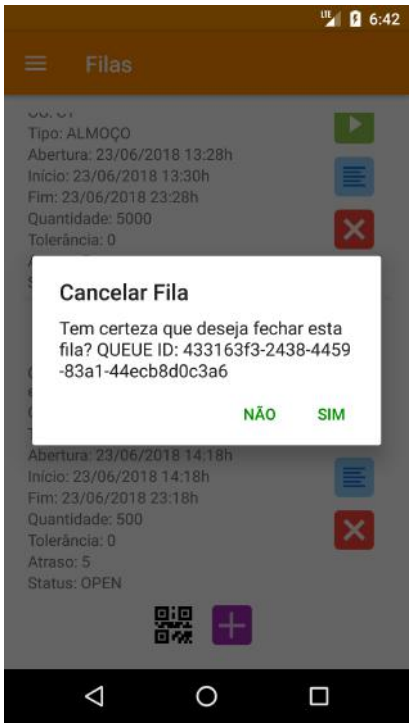

Figura 5.16: Excluir fila no RUAdmin AlertDialog

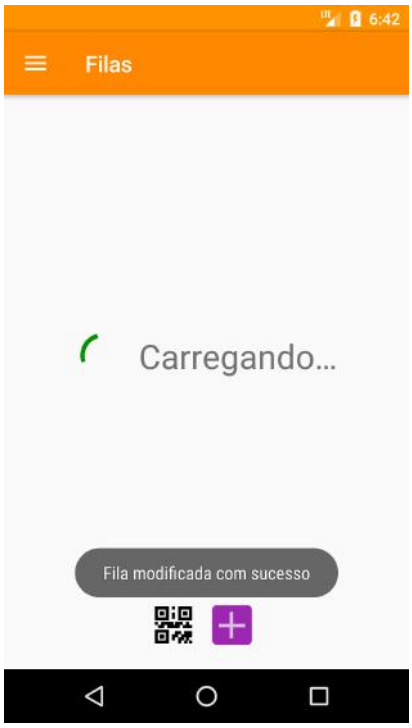

Figura 5.17: Excluir fila no RUAdmin Confirmação

O botão verde para atender uma fila, por sua vez, anexa à MainActivity o Fragment do tipo QueueAttendFragment. Este Fragment possui em seu XML de layout um View do tipo EditText, um Spinner para selecionar o status, um ListView que utiliza o QueueAttendAdapter que define o layout dos itens do ListView e as funcionalidades de cada elemento e, por fim, um Button no rodapé.

Similarmente a como é feito com as filas, o QueueAttendFragment faz com os agendamentos de uma fila. Ao ser criada, a classe mostra um ProgressBar para informar que os dados estão sendo carregados enquanto chama dinamicamente o método API.getTicketsFromQueue() que retorna todos os agendamentos da fila sendo atendida dentro de um determinado período de horários definido pela tolerância da fila em questão. O JSONArray retornado pelo servidor é convertido em um ArrayList de objetos do tipo Ticket, que é utilizado para popular o ListView da tela. O layout de cada elemento deste ListView contém informações dos agendamentos, bem como três botões. Além disso, o ArrayList que contém os agendamentos é filtrado baseado no status de agendamento selecionado no Spinner e também quando algum texto é digitado na caixa do EditText. Conforme mostra a Figura 5.18

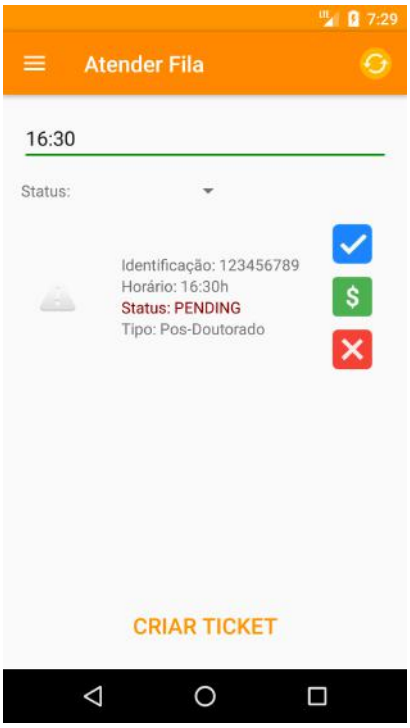

Figura 5.18: Antender fila no RUAdmin - Tela inicial

Além disso, quando há algum texto digitado no EditText, o botão "CRIAR TICKET" do rodapé se torna visível, este botão se comunica com a API através do método API.generateTicket(), e cria um agendamento para o cliente na hora. Já com relação aos botões de cada agendamento, eles são responsáveis por mudar o status de um agendamento através do método API.updateTicket() e seguindo a seguinte lógica: o azul muda o status para "CLOSED", o verde muda o status para "BILLED" e o vermelho para "CANCELED", sempre mostrando um AlertDialog antes para confirmar a operação e uma mensagem Toast depois para informar o sucesso ou falha. Conforme mostram a Figura 5.19 e a Figura 5.20

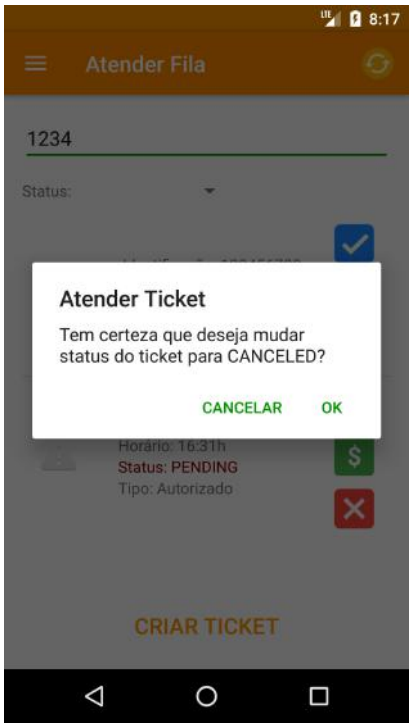

Figura 5.19: Modificar status de agendamento no RUAdmin AlertDialog

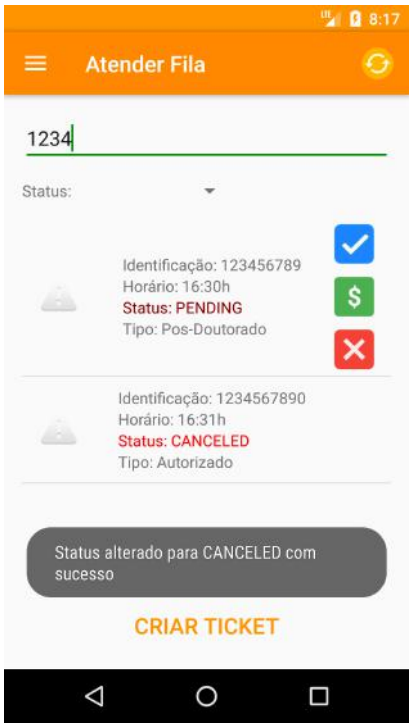

Figura 5.20: Modificar status de agendamento no RUAdmin Confirmação

Voltando ao Fragment QueueFragment, há também o botão para atendimento via QRCode, que instancia um Fragment do tipo QueueAttendQRCodeFragment. Este Fragment abre a câmera do dispositivo e permanece "escutando" por um QR-Code. Ao decodificar o QRCode, as informações do agendamento são obtidas e uma chamada à classe API.updateTicket() é feita para mudar o status do agendamento em questão para "BILLED" após uma confirmação em um AlertDialog.

O último item do menu lateral é o "Configurações" cujas funcionalidades basicamente envolvem mostrar e/ou modificar alguns valores guardados no dispositivo.

5.2.2 Aplicativo Android para clientes

Já o aplicativo Android para clientes do RU contém duas Activities: a MainActivity e a TicketActivity e, similarmente ao app para funcionários, ele também implementa a interface NavigationView.OnNavigationItemSelectedListener para o menu lateral cujas funções são implementadas de maneira bastante similar. Outro ponto similar é a comunicação com a API, que é feita através da classe API e cujos métodos são todos estáticos e seguem o padrão de chamar a API.checkToken() antes de realizarem qualquer atividade. Ao iniciar, o aplicativo abre a Activity MainActivity e em seguida substitui o conteúdo da tela por um MenuFragment através do classe FragmentManager do Android.

A classe MenuFragment é responsável por mostrar o cardápio semanal dos RUs para os alunos. Ela possui três Spinners no topo da tela: um para escolha do dia da semana, um para a escolha do restaurante desejado e, por fim, um para a escolha do tipo de refeição: que podem ser, por exemplo, almoço ou jantar. Logo abaixo, possui alguns TextViews que mostram o cardápio.

A cada seleção de um item de qualquer Spinner, uma nova chamada à API.getMenu() é feita e o conteúdo dos TextViews é atualizado.

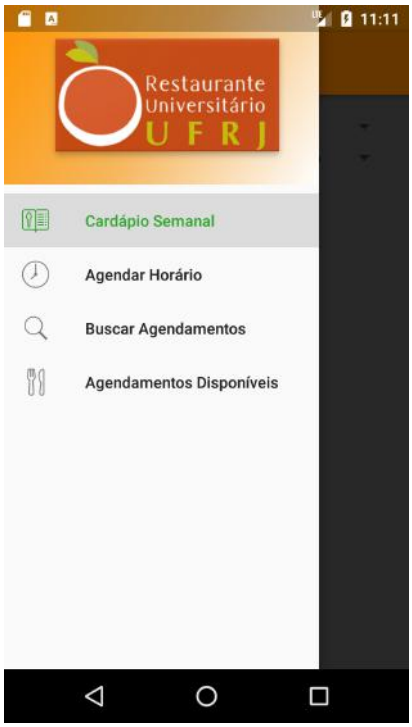

Figura 5.21: Android - Menu lateral

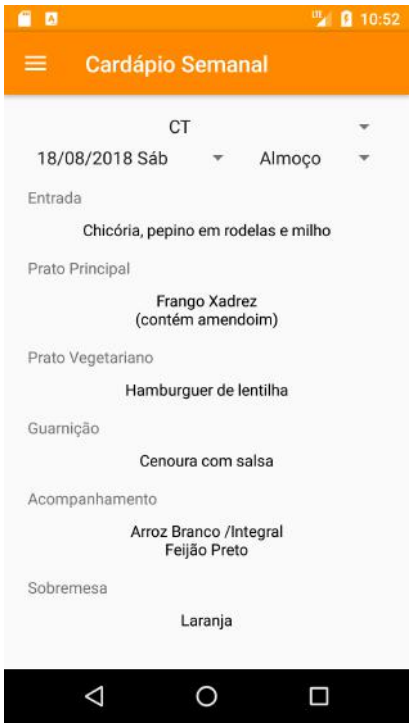

Figura 5.22: Android - Cardápio

O segundo item selecionável no menu lateral é o item “Agendar Horário”. Ao selecionar este item, o Fragment do tipo TicketAllocatorFragment é adicionado à tela pelo método OnNavigationItemSelectedListener e, como já mencionado, utilizando a classe FragmentManager para fazer a transição.

O XML de conteúdo do TicketAllocatorFragment chamado fragment_ticket_allocator.xml contém alguns TextViews de descrição dos itens que os precedem, um EditText para o usuário inserir sua identificação, um Switch que permite memorizar a identificação no dispositivo, um Spinner que possibilita a escolha da fila virtual onde o agendamento será realizado, um TimePicker para seleção do horário pretendido e, por fim, um botão para enviar os dados para a API.

Ao ser instanciada, a classe TicketAllocatorFragment faz chamadas recorrentes, a cada 10 minutos, para a API.getQueues() procurando as filas disponíveis. Ao receber uma resposta, as chamadas à API são interrompidas e o conteúdo do Spinner é atualizado com as filas disponíveis.

A segunda interação possível deste Fragment com a API acontece ao clicar no botão “ENVIAR” que chama o método API.allocateTicekt() responsável por inserir um agendamento e informa através de uma mensagem Toast se o agendamento foi realizado ou o erro ao usuário conforme mostram a Figura 5.23 e a Figura 5.24.

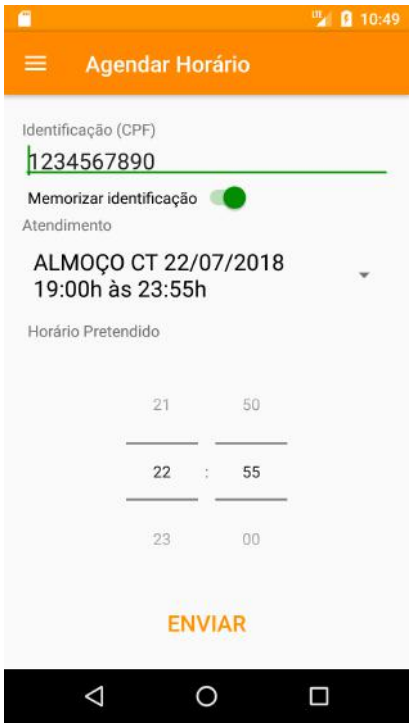

Figura 5.23: Android - Agendamento

Figura 5.24: Android - Agendamento Confirmação

O terceiro item do menu, “Buscar Agendamentos”, é bastante parecido com o “Agendar Horário”. A diferença está no TimePicker que é presente no XML do TicketAllocatorFragment, porém não no XML do TicketSearchFragment, que é o Fragment instanciado para este item do menu.

Da mesma forma, este Fragment faz chamadas recorrentes à API.getQueues() para atualizar o Spinner de seleção da fila virtual. Porém o clique no botão “ENVIAR” chama a API.searchTicket() que returna uma mensagem informando que não há agendamentos para aquele cliente, ou então retorna um agendamento. No primeiro caso, a mensagem é exibida na tela através da classe Toast, já no segundo caso, utiliza-se de um Intent para acessar a Activity TicketActivity.

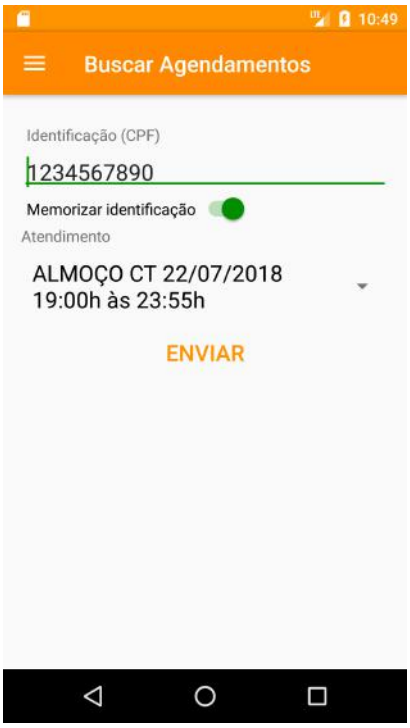

Figura 5.25: Android - Busca

A TicketActivity mostra todas as informações do agendamento encontrado na tela, inclusive um QRCode com informações do agendamento que pode ser lido pelo aplicativo dos funcionários.

Figura 5.26: Android - Agendamento

Ela também permite ao usuário excluir o agendamento em questão com um clique no botão que mostra uma lixeira vermelha. A exclusão precisa de uma confirmação em um AlertDialog.

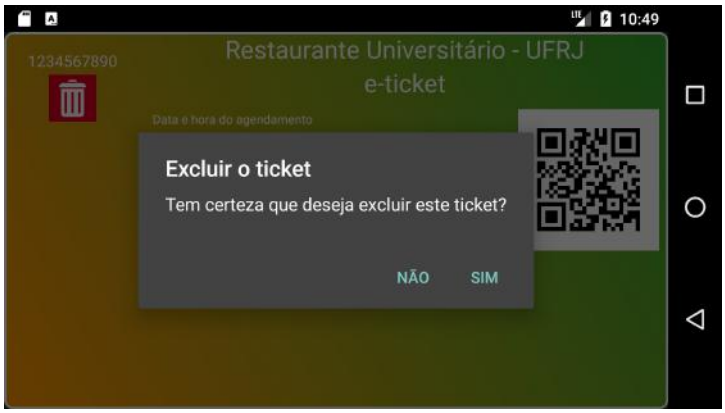

Figura 5.27: Android - Excluir agendamento

Voltando à MainActivity, há um último item do menu lateral possível de ser selecionado, "Agendamentos Disponíveis". Este item instancia a classe QueuesFragment cujo XML possui um GridView do Android SDK, que utiliza um QueueAdapter. Esta classe, assim como a TicketAllocatorFragment e a TicketSearchFragment também faz chamadas recorrentes ao método API.getQueues() e, assim que recebe um resultado da API, atualiza o GridView com informações das filas virtuais disponíveis no momento.

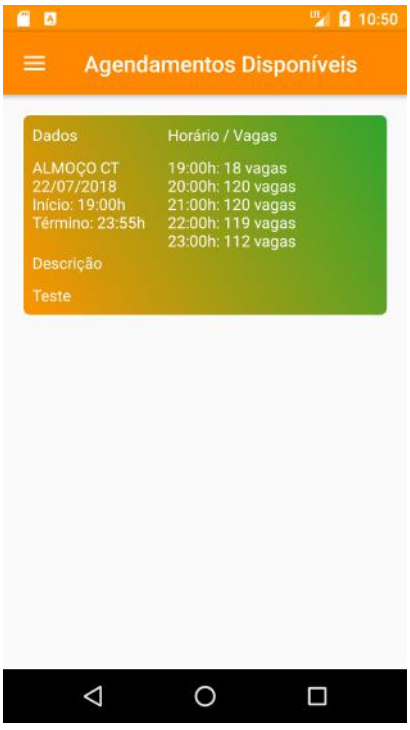

Figura 5.28: Android - Agendamentos disponíveis

5.2.3 Aplicativo iOS para clientes

O aplicativo iOS para clientes do RU possui várias similaridades ao Android. A começar pela Struct API responsável por fazer todos os chamados à API de maneira

estática.

O aplicativo foi criado utilizando-se de um TabBarController com um menu localizado na parte inferior da tela. Cada clique do em um item do menu, utiliza-se de um Segue para acessar o Controller responsável pela UI selecionada. Todos os relacionamentos entre Segues foram definidos diretamente na Main.Storyboard, conforme mostra a Figura 5.29

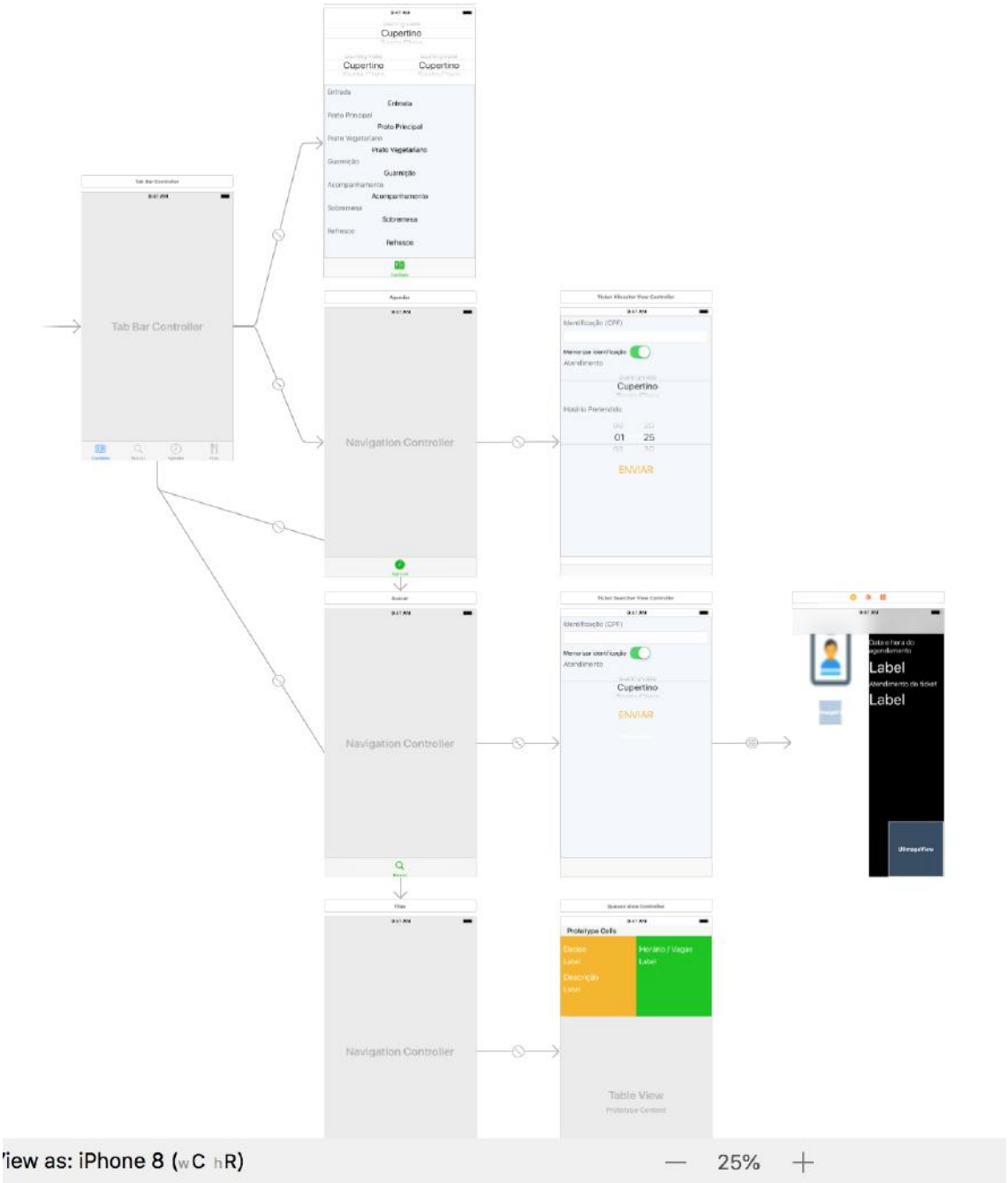

Figura 5.29: iOS - Main.Storyboard

Como no aplicativo para Android, a primeira UI que é aberta ao iniciar a aplicação é tela responsável por mostrar o cardápio semanal e corresponde ao item

mais à esquerda do menu.

A classe instanciada é a MenuViewController que herda das seguintes classes do SDK para o iOS: UIViewController, UIPickerViewDataSource e UIPickerViewDelegate.

Este ViewController possui um ScrollView contendo, dentro dele, três UIPickerViews e vários UILabels. Ao ser instanciada, a classe calcula o número de itens de cada UIPickerView e os popula com os dados relevantes: dias da semana atual, possíveis restaurantes e tipos de refeição ao delegar a origem de dados (DataSource) de cada UIPickerView que nada mais são que Arrays de Strings.

A cada mudança em qualquer item selecionado em qualquer UIPickerView, uma chamada ao método API.getMenu() é realizado e os UILabels que informam o cardápio são atualizados adequadamente.

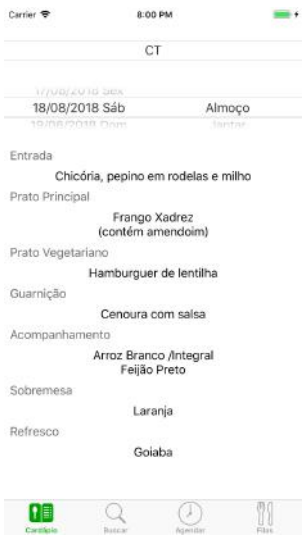

Figura 5.30: iOS - Cardápio

Pulando o segundo item do menu, o terceiro item, “Agendar”, cuja UI está associado ao TicketAllocatorViewController, este Controller herda das mesmas classes que o MenuViewController e também possui um UIScrollView que, por sua vez, contém alguns UILabels informativos e os seguintes views: um UITextField para que o usuário possa digitar sua identificação, um UISwitch que permite memorizar a identificação no dispositivo, um UIPickerView para selecionar a fila digitar em que pretende-se fazer o agendamento, um UIDatePicker para selecionar o horário desejado e, por fim, um UIButton. A implementação é correspondente à implementação Android: chamadas recorrentes são feitas à API.getQueues() e o conteúdo do UIPickerView é atualizado assim que as filas virtuais sejam recebidas da API. O clique no botão “ENVIAR” salva os dados do usuário no dispositivo caso o UISwitch esteja selecionado e envia os dados chamando o método API.allocateTicket(). A resposta do servidor é mostrada para o usuário através de uma mensagem Toast. Vale ressal-

tar aqui que o iOS não possui em seu SDK uma funcionalidade similar ao Toast do Android e para o desenvolvimento deste app, foi utilizada uma biblioteca terceira chamada $Toast\_Swift$.

Figura 5.31: iOS - Agendar

Voltando ao segundo item do menu, "Buscar", sua UI (cujo controller é o TicketSearchViewController) é praticamente idêntica à anterior diferindo da ausência do UIDatePicker. Aqui também ocorrem as chamadas ao método API.getQueues() que atualizam o UIPickerView. O clique no botão, no entanto, chama o método API.searchTicket() que se relaciona com outra rota da API. Esta rota é responsável por encontrar o agendamento do usuário. Quando a resposta é positiva, utiliza-se do NavigationController alternar para a UI relacionada ao controller TicketViewController.

Figura 5.32: iOS - Buscar

A classe TickerViewController herda da classe ViewController. Ela possui apenas

UILabels e UIImageView (sendo a imagem da lixeira vermelha usada como um botão). Os itens são dispostos para mostrar as informações do agendamento.

Figura 5.33: iOS - Agendamento

Já o clique na imagem da lixeira se comunica com a API através do método API.updateStatus() que atualiza o status de um agendamento para CANCELED com a confirmação prévia do usuário requisitada por um UIAlertController.

Figura 5.34: iOS - Excluir agendamento

Voltando ao menu inferior, o item mais à direita é responsável por mostrar todos os agendamentos disponíveis e está associado ao Controller QueuesViewController que herda da classe UITableViewController do iOS SDK.

Cada item deste TableViewController possui um UI relacionado a ela, que é definido por uma classe do Swift chamada QueuesTableViewCell que herda da classe UITableViewCell do iOS SDK. É nesta classe que os UILabels de cada fila virtual são definidos.

Figura 5.35: iOS - Agendamento disponíveis

Capítulo 6

Conclusão e Trabalhos Futuros

A demora e desorganização das filas do Restaurante Universitário da UFRJ, como se disse, levou a Decania do CT a propor a virtualização das filas. O agendamento online, atualmente disponível para web via Desktop, agora conta com aplicações nativas para plataformas móveis.

Quando a Decania enviou o e-mail para a vaga de estágio do desenvolvimento dos aplicativos móveis, eu resolvi responder ao e-mail, pois sempre me interessei por desenvolvimento de aplicações móveis e vi o projeto como uma oportunidade de aprendizado, além de ser um projeto que tem um potencial de gerar um impacto bastante positivo para os clientes do RU, sendo eu um deles. Sequencialmente, eu fui entrevistado pelo professor Fernando Ribeiro, o Decano do Centro de Tecnologia, e comecei a estagiar no setor de informática da Decania.

As funcionalidades requeridas pela Decania foram criação, edição, exclusão de filas virtuais, bem como seu atendimento de duas maneiras: para cada fila ou via QRCode para o aplicativo dos funcionários e, já para as aplicações para clientes, as funcionalidades foram a visualização das filas virtuais disponíveis e a criação, busca e exclusão de agendamentos para cada cliente em uma fila. Todas as funcionalidades foram desenvolvidas seguindo boas práticas de programação, tais como manutenibilidade, documentação e orientação a objetos foram observadas.

A maior dificuldade enfrentada durante o projeto foi a demora para criar-se um ambiente de desenvolvimento que simulasse o ambiente de produção. No início do projeto, todos os testes eram feitos diretamente no ambiente de produção.

Todo o processo de criação de tais aplicações foi descrito neste trabalho, incluindo seus desafios e viabilidade, e o fato de que todos os itens que tornam a UX positiva (ver item 2.3) foram levados em consideração foi um grande aprendizado, mostrando que, no desenvolvimento de uma aplicação, o público alvo deve estar sempre no horizonte do desenvolvedor, e suas necessidades devem ter o maior peso no processo de criação. Quando lançadas, as aplicações terão ainda uso em ambiente restrito que é o RU da UFRJ, e suas funcionalidades têm potencial para expansão.

No curto prazo, uma funcionalidade útil a ser incluída é a possibilidade de efetuar pagamentos virtuais pelo sistema, de modo que o tempo de acesso ao RU seja minimizado ainda mais, facilitando também o trabalho de seus funcionários e, eventualmente, impactando positivamente o orçamento da universidade. No longo prazo, é possível vislumbrar um cenário em que outras universidades que enfrentam problemas com as filas em seus restaurantes adotem o mesmo método de agendamento online por aplicações. Igualmente, esse método pode ser futuramente utilizado em outros ambientes universitários que se utilizam de filas físicas, como hospitais, ou eventos acadêmicos, como congressos, visando facilitar a experiência de todos aqueles que circulam diariamente na universidade.

Referências Bibliográficas

[1] APPLE, 2018. "Apple Developer Documentation". maio. Disponível em: <https://developer.apple.com/documentation>.

[2] BIEHL, M., 2015, *API Architecture: The Big Picture for Building APIs (API University Series) (Volume 2)*. CreateSpace Independent Publishing Platform. ISBN: 9781508676645.

[3] BURNETTE, E., 2010, *Hello, Android: Introducing Google's Mobile Development Platform (Pragmatic Programmers)*. Pragmatic Bookshelf. ISBN: 1-934356-56-5.

[4] BUTTFIELD-ADDISON, P., MANNING, J., NUGENT, T., 2018, *Learning Swift 3*. O'Reilly UK Ltd. ISBN: 149198757X.

[5] CASTILLO, P. A., BERNIER, J. L., ARENAS, M. G., et al., 2011, "SOAP vs REST: Comparing a master-slave GA implementation", *arXiv preprint arXiv:1105.4978*.

[6] CHONG, P. H. J., SO, P. L., SHUM, P., et al., 2004, "Design and implementation of user interface for mobile devices", *IEEE Transactions on Consumer Electronics*, v. 50, n. 4, pp. 1156–1161.

[7] DIMARZIO, J., 2016, *Beginning Android Programming with Android Studio*. John Wiley & Sons.

[8] FERRIS, C., 2004, "Web services architecture", *Standard, W3C World*, p. 10.

[9] GOOGLE, 2018. "Google Android Documentation". . Disponível em: <https://developer.android.com>.

[10] GOOGLE, 2018. "Material Design". maio. Disponível em: <https://material.io>.

[11] IBGE, 2016, *Acesso a Internet e a televisão e posse de telefone móvel celular para uso pessoal, 2015 : Pesquisa Nacional por Amostra de Domicílios*.

Rio de Janeiro : IBGE, Instituto Brasileiro de Geografia e Estatística. ISBN: 9788524044052.

[12] KHALID, H., SHIHAB, E., NAGAPPAN, M., et al., 2015, "What do mobile app users complain about?" *IEEE Software*, v. 32, n. 3, pp. 70–77.

[13] KNOTT, M., 2016, *Beginning Xcode - Swift 3 Edition*. APRESS L.P. ISBN: 1430250046.

[14] LAW, E., ROTO, V., VERMEEREN, A. P. O. S., et al., 2008, "Towards a shared definition of user experience". In: *CHI'08 extended abstracts on Human factors in computing systems*, pp. 2395–2398. ACM.

[15] TANENBAUM, A. S., 2003, *Redes de Computadores, 4 edição*. Campus. ISBN: 857605924X.

[16] THORN, A., 2008, *Cross-Platform Game Development: Making PC Games for Windows, Linux and Mac*. WORDWARE PUB CO. ISBN: 159822056X.

[17] WAGH, K., THOOL, R., 2012, "A comparative study of soap vs rest web services provisioning techniques for mobile host", *Journal of Information Engineering and Applications*, v. 2, n. 5, pp. 12–16.