Universidade do Minho
Escola de Engenharia

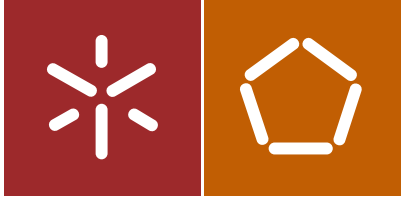
Bruno Emanuel da Silva Dias Mendes

D-Lecture: Controlo de projetores e apresentações através de um Android Smartphone

D-Lecture: Controlo de projetores e apresentações através de um Android Smartphone
Bruno Emanuel da Silva Dias Mendes

UMinho | 2016

outubro de 2016



Universidade do Minho
Escola de Engenharia

Bruno Emanuel da Silva Dias Mendes

D-Lecture: Controlo de projetores e apresentações através de um Android Smartphone

Dissertação de Mestrado
Ciclo de Estudos Integrados Conducentes ao Grau de
Mestre em Engenharia Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação do
Professor Doutor Nuno Filipe Gomes Cardoso

outubro de 2016

Declaração do Autor

Nome: Bruno Emanuel da Silva Dias Mendes
Endereço eletrónico: a62931@alunos.uminho.pt
Telemóvel: 926801939
Cartão do Cidadão: 14024798
Título da dissertação: D-Lecture: Controlo de projetores e apresentações através de um Android Smartphone

Orientador: Professor Doutor Nuno Filipe Gomes Cardoso

Ano de conclusão: 2016
Mestrado Integrado em Engenharia Eletrónica Industrial e Computadores

Universidade do Minho, _____ / _____ / _____

Assinatura:

Acknowledgements

I would like to extend my deepest thanks to my mother Laura Dias, to my father José Mendes, to my brother Sérgio Mendes, to my grandfather Joaquim Dias, and my grandmother Isaurina Silva, for being always present to give me support during my academic life as well as personal life.

To my laboratory colleagues, for the friendship, trust and help that they gave me, during not only the duration of the dissertation, but during my academic path as well.

Finally, to my counselor, Professor Doctor Nuno Filipe Gomes Cardoso for the support, the trust deposited in me, and for the help given to me during difficulties found during the implementation of this project.

To everyone, thank you very much!

Agradecimentos

Em primeiro lugar, gostaria de agradecer à minha mãe Laura Dias, ao meu pai José Mendes, ao meu irmão Sérgio Mendes, ao meu avô Joaquim Dias e à minha avó Isaurina Silva, por estarem sempre presentes para me dar apoio durante a minha vida, tanto a nível pessoal como académico.

Aos meus colegas de laboratório, pela amizade, confiança e ajuda que me deram, não só durante a duração da dissertação, mas também durante o meu percurso académico.

Por fim, ao meu orientador Professor Doutor Nuno Filipe Gomes Cardoso pelo apoio, pela confiança depositada em mim, e pela ajuda que me deu com dificuldades encontradas durante a realização deste projeto.

A todos, muito obrigado!

Abstract

In modern days, whenever there is a need to make a presentation using a video projector, it is frequently necessary the use of auxiliary equipment that will serve as image projection source, for instance, a laptop or a Smartphone. Despite the existence of various brands, which present multiple models of video projectors capable of realizing more complex tasks (on a technology level), a lot of them possess a main disadvantage of being highly expensive. Furthermore, there are various locations where presentations are made, namely schools and companies, which possess obsolete image projection equipment. Therefore, it does not make sense to substitute all of the old equipment for new generation equipment, just to give support to some new functions, which is why in most cases it is not justified the investment needed. To minimize the associated cost to support these new functions on old equipment, this current work aims to implement a low-cost system, capable of adding those new functions to the equipment, with said system being constituted by both an embedded system and an Android mobile application. Using this system it is possible to use the mobile application to control the video projector, manipulating in a simple and interactive way how presentations are made. For the supported functions, the most important are: advancing and going backwards, changing to a new file for presentation, as well as the possibility to draw and even the use of a virtual laser pointer. Taking the embedded system into consideration, it will be connected to the video projector using a HDMI plug, thereby serving as an image source. Thus, the system can receive commands sent by the Smartphone using Wi-Fi connection to control the desired presentation, being said presentation provided by a repository such as DropBox, Google Drive or OneDrive.

Keywords: Embedded Systems, Low Power Cost, Android, Smartphone, Presentations, Web Programming

Resumo

Nos dias de hoje, sempre que existe a necessidade de fazer uma apresentação usando um projetor, é frequentemente necessário a utilização de equipamento auxiliar, que serve de fonte de projeção de imagem, como, por exemplo, um computador portátil ou um *Smartphone*.

Apesar de existirem várias marcas de projetores no mercado, que apresentam múltiplos modelos capazes de realizar tarefas avançadas a nível tecnológico, vários modelos possuem a principal desvantagem de serem dispendiosos. Além disso, existem múltiplos locais onde apresentações são realizadas, nomeadamente escolas e empresas, que usam equipamento de projeção de imagem antiquado. Portanto, não faz sentido substituir todo o equipamento antigo por equipamento mais moderno, simplesmente para suportar algumas funcionalidades novas, sendo em muitos casos um investimento que não se justifica.

Para minimizar o custo associado para suportar as novas funcionalidades em equipamento mais antigo, o trabalho atual pretende implementar um sistema de baixo custo, capaz de adicionar as novas funcionalidades ao equipamento, sendo este constituído por um sistema embebido e uma aplicação para um telemóvel Android. Ao utilizar este sistema é possível controlar o projetor através de uma aplicação móvel, manipulando de forma simples e interativa a maneira como apresentações são realizadas. O novo sistema proposto suporta as seguintes funcionalidades: avançar e retroceder, modificar o ficheiro a ser visualizado, assim como capacidade de desenho e uso de um apontador laser virtual.

Desta forma, o sistema embebido é conectado ao projetor utilizando um cabo HDMI, que serve como fonte de imagem. Assim, o sistema poderá receber comandos enviados pelo *Smartphone* através de conexão Wi-Fi para controlar a apresentação desejada, que é fornecida por um repositório *online*, tais como: DropBox, Google Drive ou OneDrive.

Palavras-Chave: Sistemas Embebidos, Baixo Consumo de Energia, Android, *Smartphone*, Apresentações, Programação *Web*

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Acrónimos

SO	Sistema Operativo
OS	Operating System
GB	Gigabyte
DDR	Double Data Rate
HDMI	High-Definition Multimedia Interface
WLAN	Wireless Local Area Network
UWP	Universal Windows Platform
API	Application Programming Interface
WEP	Wired Equivalent Privacy
WAP	Wireless Application Protocol
SDK	Software Development Kit
SoC	System-on-a-Chip
IDE	Integrated Development Environment
ID	Identification
GPU	Graphics Processing Unit
GPIO	General Purpose Input/Output
SD	Secure Digital
SSD	Solid State Drive
LCD	Liquid Crystal Display
URL	Uniform Resource Locator
ISA	Instruction Set Architecture
USB	Universal Serial Bus
GPS	Global Positioning System
MIMO	Multiple-input and Multiple-output
SQL	Structured Query Language
HTML	HyperText Markup Language
XAML	Extensible Application Markup Language
TVML	TV Markup Language

Capítulo 1

Introdução

Hoje em dia no ensino, o uso de componentes auxiliares, tais como projetores e computadores, tem-se tornado essencial, não só para facilitar o processo de ensino aos professores, como também para facilitar o processo de aprendizagem dos alunos.

Com o tempo tornou-se comum os professores e os alunos usarem computadores e projetores para as suas apresentações, visto que computadores portáteis são cada vez mais comuns no ensino [1] e pelo facto de que é mais simples colocar as apresentações utilizadas por professores em repositórios, sendo depois só necessário fazer *download* para as visualizar.

Outro fator que se tornou comum na vida das pessoas é a utilização de telemóveis, com os modelos mais recentes (*Smartphones*) a serem capazes de executar mais funcionalidades, desde as mais comuns (como fazer chamadas e mandar mensagens), a novas funções que antigamente eram só encontradas em computadores (como edição de documentos e imagens) [2].

Nos dias de hoje é possível trocar o computador pelo *Smartphone* quando é preciso fazer apresentações, deixando assim de ser necessário utilizar um computador quando um *Smartphone* é capaz de o substituir.

No entanto, existe um fator impeditivo, nomeadamente o facto de que os projetores encontrados em locais comuns onde se fazem apresentações, não possuem forma de estabelecer conexão com o *Smartphone*, sendo assim necessário o uso de um sistema embebido auxiliar.

Esta dissertação pretende implementar uma aplicação para *Smartphone* capaz de estabelecer conexão com um componente auxiliar (sistema embebido),

enviando comandos que refletem as ações do utilizador para manipular uma apresentação, comandos esses que são recebidos pelo programa que é executado no sistema embebido. O sistema embebido é reponsável por fazer o *download* do ficheiro necessário para a apresentação e será também responsável por receber os comandos e executá-los, projetando assim o que o utilizador deseja, sendo possível para o utilizador não só projetar documentos mas também usufruir de um apontador laser virtual e possibilidade de desenhar durante a apresentação.

1.1 Motivação

Com a evolução dos telemóveis apareceu a possibilidade de criar aplicações cada vez mais complexas e úteis, permitindo desempenhar funções antigamente restritas a computadores [2].

De notar o aumento de popularidade que os *Smartphones* têm tido ao longo dos anos, tornando-se num utensílio comum (e até mesmo essencial) na vida das pessoas [3].

Nos locais onde se fazem apresentações, o equipamento de processamento de imagem existente está muitas vezes desatualizado, impedindo assim uso de certas funcionalidades encontradas em projetores modernos. No entanto, não é necessário trocar o equipamento por modelos mais modernos apenas para dar suporte a uma ou duas novas funcionalidades. Com o avanço da tecnologia, nomeadamente nos telemóveis e sistemas embebidos, é possível implementar essas novas funcionalidades, aumentando assim o tempo de vida útil para o equipamento de processamento de imagem.

Com estas ideias em mente e com o facto de sempre que um docente ou aluno faça uma apresentação é necessário levar um computador portátil para ligar ao projetor, surge a ideia de utilizar o telemóvel (com o auxílio de um sistema embebido) para manipular o projetor.

Recorrendo a uma aplicação, não só é possível aproveitar equipamento já existente nos locais onde se fazem apresentações, como também deixa de ser obrigatório o uso do portátil, sendo só preciso usar o telemóvel para controlar o projetor, um sistema embebido (ligado ao projetor), e um repositório *online*

onde as apresentações estão armazenadas de uma forma simples, organizada e segura.

1.2 Objetivos

Esta dissertação tem como objetivo a manipulação de apresentações a serem projetadas através de um telemóvel Android. Para atingir esse objetivo é necessário a utilização de um sistema embebido auxiliar que será ligado ao projetor, servindo assim de intermediário entre o projetor e um telemóvel, que servirá de "comando" para o utilizador. Nesse contexto é possível dividir os objetivos em duas aplicações distintas: (i) a aplicação móvel e (ii) o *software* que é executado no sistema embebido.

O objetivo da aplicação móvel é permitir que o utilizador disponha das funcionalidades comuns em *software* de apresentações (avançar, retroceder, etc) e os comandos comuns que controlam projetores (apontadores laser, desenho, etc), sendo posteriormente possível adicionar mais funcionalidades (como edição de texto), sendo necessário ter em conta a necessidade de comunicar com um repositório *online*, escolhendo o ficheiro que será utilizado para a apresentação.

No caso do *software* que vai ser executado no sistema embebido, é necessário ter a certeza que este recebe os comandos do telemóvel e os executa eficientemente, de forma a controlar o projetor de acordo com o comando recebido. É também necessário garantir a conexão ao repositório *online*, visto que é lá que se encontra o ficheiro que será reproduzido na apresentação.

1. **Requisitos Funcionais**

   Telemóvel

   - Controlar a apresentação através do telemóvel (comunicar com o sistema embebido);
   - Comunicar com o repositório *online*, permitindo ver o conteúdo dentro de pastas e obter os documentos necessários para fazer a apresentação.

Sistema Embebido

- Permitir o controlo através da aplicação móvel;
- Comunicar com o repositório *online*, fazendo o *download* dos documentos para a apresentação.

2. **Requisitos Não-Funcionais**

- Baixar o consumo de energia do sistema móvel;
- Ser robusto;
- Permitir ao utilizador funcionalidades comuns nas apresentações;
- Ser possível adicionar mais funcionalidades em *updates*;
- Executar os comandos recebidos de forma eficiente (sistema embebido);
- Ser de montagem simples.

1.3 Estrutura da Dissertação

Esta dissertação encontra-se dividida em seis capítulos. O primeiro capítulo apresenta a contextualização da dissertação, assim como os seus objetivos e a subsequente estrutura.

O capítulo 2 apresenta o estado da arte, mostrando as abordagens apresentadas pelas empresas em relação ao tema da dissertação, assim como possibilidades para os três atores (plataforma móvel, sistema embebido e repositório).

No capítulo 3 é descrita a arquitetura geral do sistema, apresentando as interações entre os diversos atores do sistema, nomeadamente entre a aplicação que é executada no *Smartphone*, o *software* executado no sistema embebido, o repositório *online* e o projetor.

No capítulo 4 é descrita a fase de implementação, mostrando como a aplicação para o *Smartphone* e o *software* para o sistema embebido foram implementadas.

No penúltimo capítulo são apresentados os resultados do capítulo anterior, mostrando as principais funcionalidades em execução.

No último capítulo são apresentadas as conclusões de todo o trabalho realizado, assim como possíveis melhorias para o sistema implementado.

Capítulo 2

Estado da Arte

Neste capítulo são apresentados alguns exemplos de mercado similares ao tema da dissertação, ou seja, abordagens para com o objetivo de incluir controlo remoto em projetores através de *Smartphones*. De seguida são descritas as plataformas móveis mais populares no mercado, assim como possíveis sistemas embebidos capazes de implementar as funcionalidades comuns em apresentações. Por fim são apresentados alguns repositórios *online* populares com a capacidade de serem integrados em aplicações.

2.1 Projetores

Atualmente os projetores continuam a receber suporte em forma de novos modelos, onde o número de funcionalidades aumenta constantemente. Uma funcionalidade que se tornou bastante utilizada é a possibilidade de visualizar documentos de um *Smartphone* em projetores. Para ser possível esta nova funcionalidade, muitas empresas decidiram embutir nos seus projetores conexão Wi-Fi [4], assim como a capacidade de serem manipulados por aplicações criadas para plataformas móveis.

Para a possível colocação de Wi-Fi [4] nos seus projetores, as empresas optaram por dois métodos: i) módulo Wi-Fi [4] já inserido no projetor, ou ii) a opção de compra de uma Pen Wi-Fi [4] para o projetor. No primeiro caso a necessidade de comprar um novo projetor torna-se imperativo, entretanto na segunda opção uma Pen apresenta um gasto menor. No entanto há que referir que as Pen são desenvolvidas especificamente para a marca do projetor, limitando o seu uso.

A interação com dispositivos móveis traz vantagens, nomeadamente o facto de não ser necessário utilizar um computador, dando ao utilizador o equivalente de um comando remoto, assim como a facilidade de incluir novas funcionalidades nas próprias aplicações para as plataformas móveis (como desenho).

Em relação às aplicações, o número de funcionalidades que elas possuem dependem de empresa para empresa (e por vezes da plataforma móvel), mas tendo sempre uma "base" de funcionalidades, nomeadamente a seleção e projeção de imagens e documentos, conexão Wi-Fi [4], assim como a possibilidade de alternar slides. A figura 2.1 apresenta o conceito geral da interação entre o telemóvel e o projetor.

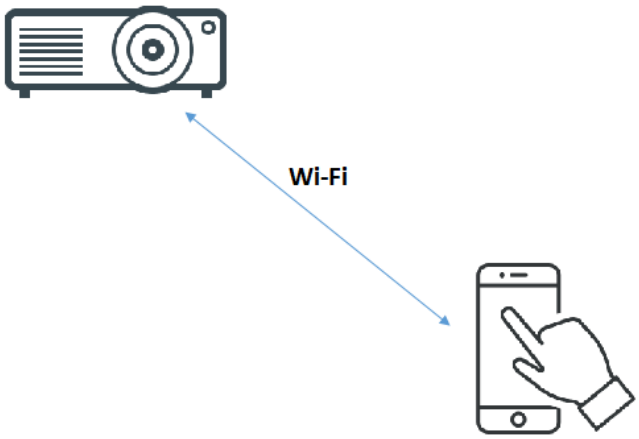


FIGURA 2.1: Conceito Geral de Interação

2.1.1 Exemplos de Mercado

Aaxa TECNHOLOGIES

A aaxa TECNOLOGIES desenvolveu um projetor com um sistema operativo Android, nomeadamente a versão Android 4.2 (Jellybean) com processador *dual core*, 1 GB de memória DDR e 4GB de memória *onboard*. Este projetor permite ao utilizador conectar-se ao projetor via Bluetooth [4] 2.0 e Wi-Fi (802.11n) [4], dando assim flexibilidade ao utilizador no caso de o *router* não funcionar.

Este projetor contém um elevado número de funcionalidades, como a capacidade de *stream* de séries e filmes, através de aplicações como Netflix, Youtube ou Flixster. Outra funcionalidade é a possibilidade de jogar diretamente a partir do projetor (fazendo *download* do jogos). Além de aplicações de *stream* e jogos, também é possível utilizar outras aplicações que se encontram na Google *Store*.

A possibilidade de instalação de aplicações da Google *Store* permite ao projetor abrir, criar e até mesmo editar documentos para apresentações. Por fim, o projetor dá a possibilidade ao utilizador de navegar na *Web* e ler *e-mails*. Mais informações sobre o projetor podem ser vistos em [5]. A figura 2.2 apresenta o projetor da aaxa TECNOLOGIES.

FIGURA 2.2: LED Android Pico Projector [5]

BenQ

Com o intuito de simplificar a projeção de imagens de *Smartphones*, a BenQ desenhou uma aplicação Android denominada "QPresenter Pro-Android", que suporta versões de Android desde 2.3.3 até à 4.3.

Para esta aplicação ser útil, é necessário que o modelo do projetor tenha conexão Wi-Fi [4], seja esta embutida no próprio projetor, ou seja necessário comprar uma Pen que permita essa conexão (neste caso tem de ser uma Pen específica da BenQ). Esta aplicação apresenta uma interface simples, mostrando um menu de opções que o utilizador pode escolher, sendo essas opções "*Documents*", "*Photos*", "*Web Pages*", "*Live Camera*" e "*Keypad*".

Ao nível das imagens (*Photos*) a aplicação só permite apresentar imagens no formato JPEG, com os documentos (*Documents*) a possuírem um maior leque de formatos, como o PDF e os do Office, nomeadamente Word, Excel e Powerpoint.

Com a opção "*Web Pages*" o utilizador pode navegar pela *Web* como em qualquer outro navegador, apesar de possuir uma interface mais simples. Com a opção "*Live Camera*" é possível para o utilizador a captura de imagens através da câmara do dispositivo, projetando-as através do projetor para um ecrã. Estas três opções permitem tomar anotações, caso o utilizador deseje.

Por fim, a opção "*Keypad*" permite ao utilizador acesso a uma interface de um comando regular (como de uma televisão), com opções como avançar, retroceder, volume e a opção de modificar as configurações do próprio projetor, como contraste da imagem projetada ou a luminosidade. Mais informações sobre o projetor podem ser vistos em [6].

Epson

Com a criação de novos modelos de projetores com disponibilidade de rede, a Epson decidiu também criar duas aplicações, nomeadamente uma aplicação Android e uma aplicação iOS de forma a simplificar a projeção de documentos e imagens.

Dependendo da versão, é possível o utilizador projetar diferentes tipos de formatos. Para a versão Android da aplicação, só é permitido projetar documentos com o formato PDF e imagens com o formato PNG, TIFF e JPG. No caso da aplicação móvel para o iOS já é suportado um número maior de formatos, incluindo os formatos do Office Word, Excel e PowerPoint e, por último, a inclusão do formato Keynote, uma marca comercial da Apple.

Outra possibilidade para a aplicação é a possibilidade de projetar páginas da *Web*, permitindo assim ao utilizador navegar na *Web*.

Com as possibilidades de projeção anteriormente referidas é também incluído na aplicação os comandos, que permitem o projetor realçar e clarificar qualquer imagem, página da *Web* ou documento projetado, permitindo desfrutar de uma projeção de imagem com maior qualidade.

Por fim, a aplicação permite ao utilizador tomar notas sobre a imagem projetada, sendo essas notas apresentadas pelo projetor. Caso o utilizador deseje, existem opções diferentes para esta funcionalidade, podendo utilizar uma caneta ou marcador com cores diferentes e diferentes níveis de opacidade. Caso seja necessário apagar anotações, foi incluída a opção de um apagador. Mais informações sobre o projetor podem ser vistos em [7].

Panasonic

Com os modelos novos de projetores da Panasonic com conexão Wi-Fi [4] e suporte de controlo, uma aplicação tanto em Android como em iOS foi desenvolvida, sendo capaz de projetar (recorrendo ao projetor) imagens, captura de ecrã e documentos guardados num dispositivo *Smartphone* ou *tablet*, sendo suportados os dispositivos que possuem um SO Android 4.0, 4.1, 4.2, 4.3 e 4.4, abrangendo assim uma vasta gama de dispositivos Android juntamente com os dispositivos iOS.

Ao nível de formato de documentos, a aplicação é compatível com vários formatos, desde formatos populares de imagens (JPEG e PNG) a formatos populares em locais de ensino, tais como: PDF, PowerPoint, Excel e Word.

Uma funcionalidade particular nesta aplicação é a possibilidade de utilizar um marcador para desenhar, sendo válida para imagens, documentos e até mesmo para páginas *Web*, com a opção extra de mudar o marcador de desenho, desde o seu formato (estilo e tamanho) até à sua cor.

Caso o utilizador não possa estar presente fisicamente numa apresentação, existe a possibilidade de controlar o projetor remotamente, conectando-se ao *Network* ID do projetor (necessita de *username* e *password* para autenticação).

Por fim, é possível que os projetores sejam controlados por até dezasseis dispositivos Android/iOS, projetando simultaneamente os documentos desejados, com a possibilidade de alternar qual é o dispositivo (neste caso o documento desse dispositivo) a dar foco. Mais informações sobre o projetor podem ser vistas em [8].

2.2 Plataformas Móveis

Desde o aparecimento de telemóveis que a população tornou-se dependente deles. A facilidade das pessoas comunicarem entre si, era um avanço importante ao nível de desenvolvimento das comunicações, causando o aparecimento de muitas empresas produtoras de telemóveis, muitas ainda presentes na atualidade (como a Nokia e Siemens).

Com o aumento de procura de telemóveis, as empresas começaram a desenvolver tecnologia mais avançada, tentando melhorar os seus telemóveis para ganhar terreno em relação aos seus concorrentes. À medida que os telemóveis foram evoluindo, começaram a introduzir a possibilidade de ligação à *internet* e de consultar o correio eletrónico. Com conexão à *internet*, a troca de informação aumentou, possibilitando até troca de conteúdo multimédia entre utilizadores. Este avanço tecnológico originou a inclusão de diversos periféricos, como câmara fotográfica, GPS e Bluetooth [4], permitindo os telemóveis a executarem um leque de tarefas muito superior ao que era possível originalmente, deixando de estarem restringidos simplesmente à receção e envio de mensagens e chamadas. Isto originou o aparecimento dos primeiros *Smartphones*, que viriam a substituir os até ao momento telemóveis comuns e que tornar-se-iam num dispositivo essencial na vida quotidiana das pessoas.

Com o aparecimento de *Smartphones*, as empresas começaram a desenvolver os seus próprios sistemas operativos, com o intuito de ganhar uma forte presença no mercado. Com o passar dos anos houve um sistema operativo (Android) que ganhou terreno em relação aos outros sistemas operativos, sendo neste momento o SO mais utilizado no mercado. Com o Android em primeiro, aparece em segundo lugar o sistema operativo da Apple, o iOS e em terceiro lugar o sistema operativo da Microsoft para o *Smartphone*, o Windows Phone [9]. A tabela 2.1 apresenta as unidades vendidas de cada plataforma móvel no primeiro trimestre.

SO	Unidades Vendidas	Percentagem Mercado
Android	293,771.2	84.1
iOS	51,629.5	14.8
Windows	2,399.7	0.7
Outros	1451	0.4
TOTAL	349,251.4	100.0

TABELA 2.1: Vendas *Smartphone* no primeiro trimestre 2016 [9]

Com o lançamento da loja *online* para aplicações por parte da Apple, e consequentemente por outros sistema operativos, a popularidade dos *Smartphones* aumentou ainda mais. Com a facilidade de encontrar e distribuir aplicações, as empresas dos respetivos sistemas operativos permitiram o desenvolvimento de aplicações a qualquer programador, sendo assim possível desenvolver e colocar a sua aplicação na loja de *Apps*. Para esse efeito, as empresas, além de colocarem certas regras e etapas para colocar as suas aplicações nas lojas, lançaram os seus *Software Development Kits* (SDK), potenciando um aumento exponencial não só em número de aplicações, mas também em variedade.

As plataformas móveis possuem os seus prórpios IDEs (*Integrated Development Environments*) para desenvolvimento, com cada IDE a incluir por defeito suporte ao SDK da respetiva plataforma móvel, com o Android a possuir o Android Studio, o Windows Phone a possuir o Visual Studio, e os iOS a possuir o Xcode. OS IDEs geralmente apresentam um ambiente de desenvolvimento, emulador (para ser possível fazer depuração às aplicações), a possibilidade de fazer depuração em *Smartphones* reais (requer ativação da opção de depuração no próprio *Smartphone*), e *application programming interfaces* (API), que introduzem com cada lançamento novas funcionalidades, melhoramento de certas funcionalidades, ou a substituição da forma como certas funcionalidades são escritas (a nível de código).

O SDK mais utilizado é o do Android, lançado pela Google, sendo suportado por diversos IDEs populares, como o Android studio, o Eclipse IDE, e até mesmo IDEs nativos para os outros sistemas operativos, como o Visual Studio da Microsoft e Xcode da Apple (como pluggin). Com a popularidade do

Android, a comunidade que o suporta é enorme, sendo possível para programadores encontrarem exemplos básicos de aplicações, para quem quer aprender ou encontrar formas diferentes de implementar certas funcionalidades, ou simplesmente para tirar dúvidas. O iOS também possui uma grande comunidade, sendo o segundo SO mais popular, enquanto o Windows Phone possui uma comunidade bastante menor [10]. É mostrado na tabela 2.2 as informações com mais detalhe, fazendo uma comparação entre os três ao nível de desenvolvimento de aplicações, do suporte de desenvolvimento, de IDE, e por fim da comunidade.

	Android	**iOS**	**Windows Phone**
Desenvolvimento de Aplicações	•Grande acessibilidade de *debbugers* •Relativamente rápido a fazer *deployment* •Grande tamanho de *deployment* de uma aplicação	•Boa acessibilidade de *debbugers* •Rápido a fazer *deployment* •Tamanho normal de *deployment* uma aplicação	•Grande acessibilidade de *debbugers* •Relativamente rápido a fazer *deployment* •Grande tamanho de *deployment* de uma aplicação
Suporte de Desenvolvimento	•Pequena taxa para venda de aplicações •Grande penetração de mercado	•Pequena taxa para venda de aplicações •Grande penetração de mercado	•Penetração de mercado limitada •Pequena taxa para venda de aplicações
IDE	•Suportado por maior parte dos IDEs populares que existem no mercado	•Bom suporte com IDEs como Apple Xcode e Jetbrains Appcode	•Bastante limitado, sendo utilizado maioritariamente o Visual Studio
Comunidade	•Bastante	•Bastante	•Normal

TABELA 2.2: Comparação entre Sistemas Operativos [10]

Como seria de esperar, o Android acaba por ser o melhor sistema operativo para desenvolvimento.

2.2.1 Android

A Google libertou oficialmente o sistema operativo Android para o mercado em 23 de Setembro 2008, tornando-o *open source* [11].

O SO da Google tinha como objetivo criar uma zona *open source* para desenvolvimento de *software* em *Smartphones* [10]. Desde então o SO tem sido constantemente melhorado, indo da versão 1.0 até à mais recente 6.0 (Marshmallow), sempre com a adição de novas funcionalidades.

O Android é baseado no *Kernel* do Linux mas projetado para dispositivos *touchscreen* como *Smartphones* e *tablets*. A interface de utilizador no Android é focado na manipulação direta, usando gestos *touch*, como *swipping*, *tapping* e *pinching*, para manipular objetos no ecrã, possuindo também um teclado virtual para inserção de texto.

Para além das bibliotecas do *Kernel* do Linux, foram adicionadas várias bibliotecas à plataforma Android para suportar funcionalidades mais complexas. Por fim, a *framework* da aplicação foi criada para providenciar ao sistema bibliotecas para as aplicações *end-user* [12].

Para o desenvolvimento de aplicações, o Android facilita o uso de bibliotecas gráficas 2D e 3D, um motor SQL embutido e customizado com memória persistente e capacidades avançadas de *network* como 3G, 4G e até WLAN [10]. A figura 2.3 apresenta a arquitetura do sistema operativo Android.

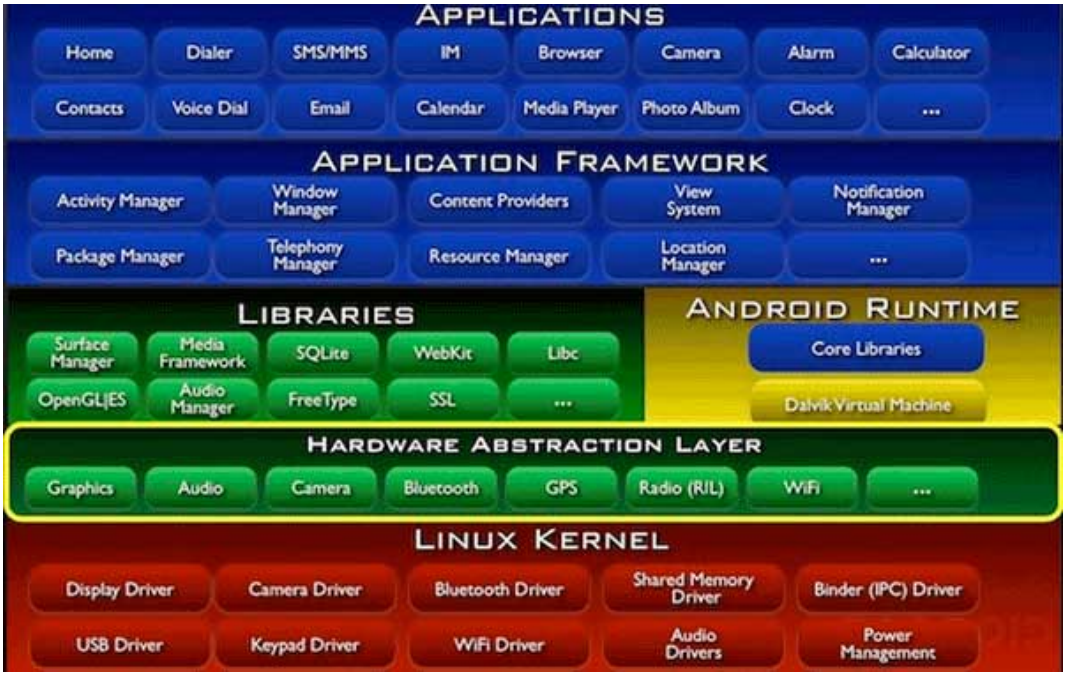


FIGURA 2.3: Arquitetura de Android [13]

2.2.2 iOS

Desenvolvido pela Apple Inc, o iOS é o SO utilizado em dispositivos como o iPhone e o iPad, encontrando-se agora na versão mais recente iOS 9.

Atualmente um dos três SO mais populares do mundo juntamente com o Android e o Windows Phone, este SO tens as suas aplicações desenvolvidas em objetive-C e swift com o auxílio das *frameworks* do sistema, com o SO a ser dividido em uma estrutura simples, encontrando-se dividido em quatro camadas. Nomeadamente: a camada Cocoa *Touch*, a camada *Media*, a camada *Core Services* e a camada *Core* OS [14].

A camada de mais baixo nível é a camada *core* (*Core* OS), que possui as funcionalidades necessárias para a maior parte das tecnologias utilizadas nas outras camadas [15].

A segunda camada (*Core Services*) contém os serviços fundamentais do sistema para as aplicações [16].

A terceira camada (*Media*) possuí as tecnologias de vídeo, áudio e gráficos utilizados para implementar as experiências de multimédia para as aplicações [17].

A última camada (Cocoa *Touch*), e a de mais alto nível, contém as *keys frameworks* para criar as aplicações iOS, que ajudam a definir a aparência das aplicações [18]. A figura 2.4 apresenta a arquitetura do iOS de forma simplificada.

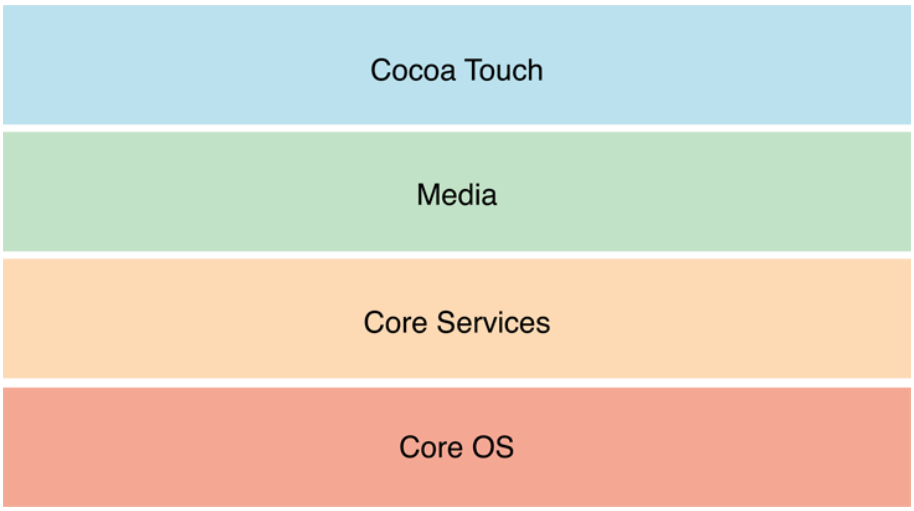


FIGURA 2.4: Arquitetura do iOS [14]

2.2.3 Windows Phone

Com a Apple e a Google a investirem no mercado de *Smartphones* e outros dispositivos com tecnologia *touch screen* (através dos SO iOS e Android), a Microsoft decidiu também lançar o seu próprio SO, sendo assim lançado para o mercado em 21 de Outubro de 2010 o Windows Phone.

Com o tempo, o Windows Phone foi procurando não só melhorar o seu SO, mas também ganhar valor no mercado em relação aos mais populares SO, nomeadamente o Android e o iOS.

Com o lançamento recente do novo Windows 10, a Microsoft decidiu modificar e simplificar a forma como os programadores desenvolvem as suas aplicações. Ao nível de programação, o Windows Phone adicionou com o Windows 10 a *Universal* Windows *Platform* (UWP), válida para Windows 10, permitindo que todas as aplicações para dipositivos Windows possam agora ser desenvolvidas com uma só API universal, adaptando as restantes APIs para o dispositivo correspondente. Com isto é possível criar aplicações com diversas linguagens de programação, desde C#, C++, Javascript com HTML, Visual Basic com XAML, entre outras. Outra possibilidade é a utilização de componentes diferentes numa aplicação, mesmo que esses componentes estejam escritos em linguagens diferentes [19]. Estas opções permitem um programador criar aplicações de baixo nível de dificuldade (para quem está a começar), como de alto nível de dificuldade, como aplicações de uso empresarial e ou jogos.

Infelizmente, isto implica que quem possuí Windows 8.1 e Windows Phone 8.1 (ou versões anteriores), não pode usufruir deste novo método de desenvolvimento, sendo obrigado a utilizar os métodos antigos de *deployment* [19].

2.3 Sistemas Embebidos

Um sistema embebido é, na sua definição mais simples, um sistema de computação dedicado desenhado para desempenhar um número pequeno de funções. Os sistemas embebidos são geralmente parte de um sistema maior, desde televisões, a dispositivos comuns no dia-a-dia, como *Smartphones*, computadores, e impressoras, a sistemas mais complexos, como aviões, intrumentos robóticos em medicina, e até mesmo intrumentos de guerra (como mísseis).

A figura 2.5 apresenta um sistema embebido da Intel, o Intel Atom Z510P.

FIGURA 2.5: Sistema Embebido - Intel Atom Z510P [20]

Um dos primeiros sistemas embebidos a surgir no mundo foi o *Apollo Guidance Computer*, introduzido em Agosto de 1966. Este sistema foi instalado em cada *Apollo Lunar Module* e em cada *Apollo Command Module*, de forma a providenciar tanto computação como interface eletrónica para navegação e controlo da nave espacial.

Com o tempo, os sistemas embebidos começaram ser mais utilizados, e com isso surgiram necessidades maiores. Com o aumento de produção de sistemas embebidos, o preço baixou, possibilitando uma utilização maior em termos de quantidade. Para poderem ser utilizados em outras áreas, assim como serem capazes de executar as suas funções mais rapidamente, foi melhorado o seu desempenho. Por fim, com o aumento de desempenho e o aumento do número de funções que os sistemas embebidos podiam desempenhar, surgiu a necessidade de baixar o consumo de energia. Estas três necessidades (preço, desempenho e consumo de energia) estão sempre presentes na concepção de sistemas embebidos.

O aumento do número de sistemas embebidos originou o aparecimento de diversas arquiteturas para os processadores utilizados nos sistemas embebidos, cada uma com a sua *Instruction Set Architecture* (ISA). Cada arquitetura, apesar de variar apresenta habitualmente semelhanças, como intruções (MOV, ADD, JMP), registos (R0, R1, R2, *etc*) e *flags* (como a *flag* de *carry*). A figura 2.6 apresenta a ISA do microcontrolador AVR de 8 bits.

Mnemonic	Description	Mnemonic	Description	Mnemonic	Description
Flow Control		**Bit Manipulation**		**Load/Store**	
JMP ◆	Jump absolute (24-bit)	SEC/CLC	Set/clear C flag (carry)	MOV	Copy register to register
RJMP	Branch relative (12-bit)	SEH/CLH	Set/clear H flag (half carry)	LD	Load indirect through X/Y/Z
IJMP ●	Jump indirect (Z)	SEN/CLN	Set/clear N flag (negative)	LD ●	Load indirect with postincremnt
RCALL	Call subroutine	SEZ/CLZ	Set/clear Z flag (zero)	LD ●	Load indirect with predecremnt
ICALL ●	Call subroutine indirect (Z)	SEI/CLI	Set/clear I flag (interrupt)	LDD ●	Load indirect with 6-bit offset
RET/RETI	Return/from interrupt	SES/CLS	Set/clear S flag (sign)	LDI	Load 8-bit immediate
CP/CPC	Compare/with carry	SEV/CLV	Set/clear V flag (overflow)	LDS ●	Load from 16-bit address
CPI	Compare with 8-bit immediate	SET/CLT	Set/clear T bit	LPS ●	Load from program space
CPSE	Compare, skip if equal	SBR/CBR	Set/clear bit in register	ST	Store indirect through X/Y/Z
SBRS/SBRC	Skip if register bit set/clear	BSET/BCLR	Set/clear bit in status register	ST ●	Store indirect with postincremnt
SBIS/SBIC	Skip if I/O bit set/clear	SER/CLR	Set/clear entire register	ST ●	Store indirect with predecremnt
BRcc	Conditional branch	SBI/CBI	Set/clear bit in I/O space	STD ●	Store indirect with 6-bit offset
Logical		**Arithmetic**		STS ●	Store to 16-bit address
AND	Logical AND	ADD/ADC	Add/with carry	IN/OUT	Input/output to/from I/O space
ANDI	Logical AND 8-bit immediate	ADIW ●	Add 6-bit immediate	PUSH/POP	Push/pop stack element
OR	Logical OR	SUB/SUBC	Subtract/with borrow	BLD/BST	Load/store T bit
ORI	Logical OR 8-bit immediate	SBIW ●	Subtract 6-bit immediate	**Miscellaneous**	
EOR	Logical exclusive-OR	SUBI/SBCI	Subtract 8-bit imm/w borrow	NOP	No operation
LSL/LSR	Logical shift left/right by 1 bit	INC/DEC	Increment/decrement register	SLEEP	Wait for interrupt
ROL/ROR	Rotate left/right by 1 bit	MUL ◆	Multiply 8 × 8 → 16	WDR	Watchdog reset
ASR	Arithmetic shift right by 1 bit				
COM/NEG	One's/two's complement				
SWAP	Swap nibbles		Can use R16–R31 only	●	Not available on 90S1200, 1220
TST	Test for zero or minus		Can use R24–R31 only	◆	Future enhancement

FIGURA 2.6: ISA do microcontrolador AVR de 8 bits [21]

A adição constante de funcionalidades aos sistemas embebidos aumentou, em muito, a complexidade dos sistemas. Adicionando a necesidade comum da presença de periféricos de entrada como de saída, tornou certos sistemas embebidos em autênticos mini-computadores, que em certos casos até podem possuir os seus próprios sistemas operativos. Certos sistema embebidos (como a raspberry Pi B+ [22]) são capazes de utilizar um leque diverso de sistemas operativos, enquanto que outros (como o Chromecast [23]) possuem um SO próprio.

Com um leque tão grande de utilidade, os sistemas embebidos irão cada vez mais evoluir, sendo inseridos em novas áreas para simplificar tarefas, continuando a assentar a necessidade do ser humano nos sistemas embebidos.

2.3.1 Chromecast

O Chromecast é um dispositivo de transmissão de conteúdo multimédia em fluxo contínuo capaz de ligar à porta HDMI [24] de uma televisão [23].

Desenvolvido pela Google, esta Pen é controlada via *Smartphones*, *tablets* e até mesmo computadores portáteis, tendo bastante êxito no mercado, sendo o

dispositivo de *streaming* mais vendido nos EUA em 2014 [25].

Em relação à versão mais recente, convém analisar as suas especificações. Possuí duas portas, sendo uma a entrada micro USB, necessária para providenciar energia ao dispositivo (5V, 1A), e uma porta HDMI [24] para conteúdo de *stream*, com resolução máxima de vídeo de 1080p. Com um tamanho de 51.9x51.9x13.49 mm e um peso de 39.1 g, este dispositivo é compacto e leve, excelente para levar em viagens. Para comunicação, a nova versão Chromecast usa 802.11 b/g/n/ac Wi-Fi [4] de 2.4 GHz/5GHz, enquanto que a primeira versão só fornece conexão Wi-Fi [4] 802.11 b/g/n, oferecendo assim um maior leque de opções. Para a segurança da conexão *wireless*, o Chromecast usa WEP e WAP/WAP2 [26].

O Chromecast funciona com Android (versão 4.1 e acima), iOS 7.0 e versões superiores, Windows 7, 8, 8.1 e 10, Mac SO 10.7 e acima e por fim versões ou superiores ao SO Chrome 27 (Chromebooks), apresentado assim um enorme leque de sistemas operativos compatíveis com o Chromecast [26].

Por fim, com o Chromecast SDK, é possível desenvolver aplicações para o Chromecast, sendo necessário criar duas aplicações, uma aplicação de envio (dispositivo de controlo) e outra de receção (Chromecast). O SDK é instalado juntamente com *Google Play Services* (SDK opcional) no Android Studio (caso a aplicação de envio seja para o SO Android) [27]. A figura 2.7 apresenta um Chromecast 2.

FIGURA 2.7: Chromecast 2 [28]

2.3.2 Raspberry Pi B+

A raspberry é uma série de sistemas embebidos considerados mini computadores, apresentando funcionalidades básicas como processamento de texto, *stream* de vídeo e áudio e, até permite, a utilização de jogos [29]. A figura 2.8 apresenta o modelo Raspberry Pi B+.

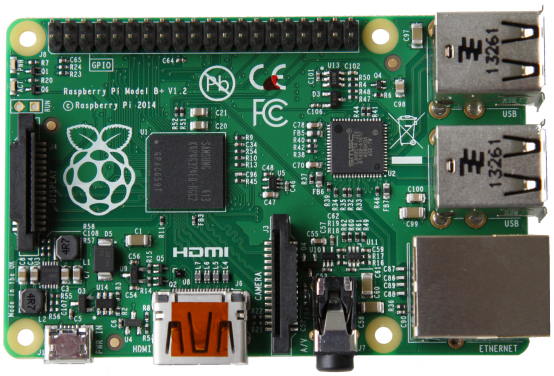


FIGURA 2.8: Raspberry Pi B+ [30]

A nível de tamanho o modelo B+ possuí 85x56x17 mm, sendo relativamente pequeno, principalmente quando comparado a um computador fixo. O modelo B+ possuí um *System-on-a-Chip* (SoC) *Broadcom* BCM2835, incluindo um processador ARM1176JZF-S de 700 MHz, um GPU *Video Core* IV com 512 MB (primeiros modelos B só vinham com 256 MB). Este modelo não incluí um disco rígido embutido nem um *solid state Drive* (SSD), usando alternativamente um cartão micro SD tanto para *booting* como para memória persistente [31].

A nível de *outputs*, o modelo B+ usa para uma porta 3.5mm Jack tanto para vídeo como para áudio, tendo também *output* HDMI [24], uma porta para interface com câmaras, uma porta de interface com display LCD, quatro portas USB 2.0, porta Ethernet 10/100 e um *General Purpose Input/Output* (GPIO) de 26 pinos [31].

Para energia, o raspberry Pi B+ usa um cabo micro USB de 5V, com uma corrente de 650 mA, necessitando assim de 3 W (o modelo B necessita de 3.5 W) [31].

O SO oficialmente suportado pelos criadores da raspberry Pi é o Raspbian. No entanto é possível usar o SO Linux e suas variações, como o Ubuntu *Mate*

e o *Snappy* Ubuntu *Core*, e até variações do Windows 10, como o Windows 10 IOT *Core*, com outros possíveis SO *custom* [22].

Por fim convém referir o custo de um raspberry Pi em relação a um computador. Na tabela 2.3 é apresentado um custo normal de uma raspberry Pi com componentes extras para poder usufruir das funcionalidades básicas. A tabela original pode ser encontrada em [32] (valor do raspberry Pi B+ reduziu).

Item	**Custo**
Cabo de alimentação Micro USB	10 $
Teclado USB	8 $
Rato USB	4 $
Cartão 8 GB microSD	8 $
Cabo Ethernet	2 $
Cabo HDMI	4 $
USB *card Reader* (instalação do SO)	7 $
Caixa para Raspberry pi (opcional)	15 $
Raspberry Pi B+	25 $
CUSTO TOTAL	83 $ = 76.96 €(Aprox.)

TABELA 2.3: Custo para um mini computador

2.3.3 Apple TV

Assim como a Google, a Apple também possuí o seu dispositivo de transmissão de conteúdo de multimédia denominado Apple TV. Este sistema possui quatro gerações de modelos, sendo o primeiro libertado para o mercado em 2007, o segundo em 2010, o terceiro em 2012 e por fim a versão mais recente, que foi lançada em 2015 [33]. A figura 2.9 apresenta o modelo da Apple TV mais recente, a geração 4.

FIGURA 2.9: Apple TV geração 4 [33]

Este dispositivo vem juntamente com um comando para controlo remoto, mas é possível controlar o dispositivo através de um *Smartphone* (iPhone) e também iPad, sendo compatível com com televisores de alta definição com HDMI [24] [34].

A nível de especificações, cada geração foi sendo modificada, coincidente com o avanço tecnológico ao longo dos anos. A geração mais recente (4), possui um processador A8 com arquitetura de 64 bits, com capacidade de 32/64 GB. A nível de portas e interfaces para comunicação, a Apple TV possuí cabo HDMI [24], receptor IV (para o comando remoto), Bluetooth 4.0 [4], Wi-Fi (802.11ac com tecnologia MIMO) [4] e *ethernet* (10/100BASE-T). Para conteúdo multimédia, a Apple TV 4 suporta bastantes formatos, com formatos possíveis para imagens/fotografias em JPEG, TIFF e GIF, formatos de aúdio como AAC (16 a 320 Kbps), MP3 (16 a 320 Kbps), MP3 VBR, *Audible* (formatos 2, 3 e 4), WAV, Dolby *Digital* 5.1 e Dolby *Digital Plus* 7.1, entre outros. Para vídeo suporta MPEG-4 até 2,5 Mbps de 640x480 píxeis, de vídeo H.264 até 1080p e de *Baseline Profile* H.264 de nível 3.0 ou inferior [34].

O sistema operativo da Apple TV 4, ao contrário das versões anteriores, apresenta um novo sistema operativo, o tvOS. Este sistema operativo é derivado do iOS, acrescentando novas *Frameworks* como TVMLJS, uma API utilizada para carregar páginas TVML para visualizar informação em aplicações cliente-servidor e TVMLKit, que incorpora elementos Javascript e TVML nas aplicações, entre outros [35].

Com o tvOS, é possível aos programadores criarem as suas próprias aplicações, desde as tradicionais que utilizam as *Frameworks* do iOS, como as aplicações cliente-servidor. Estas aplicações possuem como objetivo principal o *stream* de conteúdo multimédia através do uso de tecnologias *web* como HTTPS, XMLHTTPRequest, DOM, e JavaScript [35]. Quando as aplicações são colocadas para uso público, é possível usufruir delas através das lojas respetivas (*App Store*, *Apple TV App Store*, ou *Mac App Store*).

2.4 Repositórios *Online*

Com a era informática, a geração de informação digital produzida por utilizadores individuais (como programadores) está cada vez a aumentar, notando-se um aumento por cada ano [36].

Isto gerou a necessidade de empresas e utilizadores que trabalham com grandes quantidades de informação digital, procurar outros meios de guardar informação. Foi assim que o conceito de *Cloud Hosting* surgiu. Os repositórios *online* aumentaram em popularidade, oferecendo formas simples de guardar informação, assim como de partilhar dita informação. A figura 2.10 apresenta o conceito geral de *Cloud Hosting*.

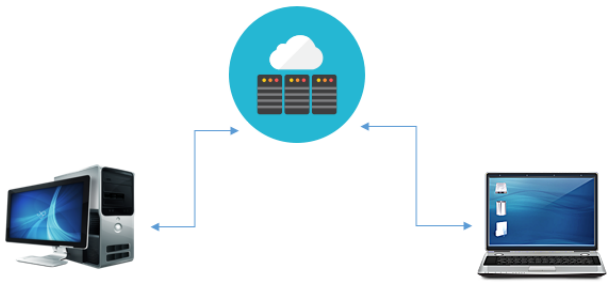


FIGURA 2.10: *Download* e *Upload* de ficheiros para a "*Cloud*"

Muitos serviços de *hosting* aparecerem, como Dropbox, One Drive, Google Drive, 115 SyncDisk, entre outros. Repositórios como o Dropbox foram referidos a possuir mais de 100 milhões de utilizadores e 115 SyncDisk com mais de 30 milhões de utilizadores, para dar a ideia de quão utilizado os repositórios se tornaram populares [37].

No entanto há que referir o facto de que para manter os servidores dos repositórios *online*, é necessário investir muito dinheiro. Isto levou aos criadores dos repositórios a cobrar aos utilizadores uma certa quantia para poderem pagar as despesas dos serviços. De referir que muitos destes repositórios permitem a criação de contas gratuitas (como o Dropbox), mas limitam o espaço de armazenamento aos utilizadores gratuitos [37]. Outros limitam as funcionalidades, como o Github que só permite a criação de pastas privadas para guardar informação para quem possui conta *premium* (utilizador que paga).

Com espaço de armazenamento limitado, muitos utilizadores normalmente optavam por fazer *update* (substituir por versões mais recentes) ao ficheiros que guardavam nos repositórios em vez de guardar todas as versões. Isso causou o aparecimento de *cloud synchronization mechanism*, facilitando o *update* dos ficheiros do lado do utilizador para o servidor (*cloud*) [37].

Sendo assim, com a facilidade de armazenamento e de partilha de informação por parte dos utilizadores, os repositórios continuam o seu aumento, tanto popularidade como de necessidade por parte de utilizadores, levando aos criadores dos repositórios a investirem o seu dinheiro para melhorar e expandir os seus serviços, permitindo assim aos utilizadores mais funcionalidades, espaço e rapidez no *update*/*upload*/*download* de ficheiros.

2.4.1 Dropbox

O Dropbox é um serviço de *hosting* de ficheiros criado pela empresa Dropbox, Inc., serviço que foi lançado inicialmente em Junho de 2007.

Dos serviços de *hosting* mais utilizados no mundo [37], este repositório apresenta a possibilidade de ser acedido pela *internet*, fazendo *login* pelo *website* próprio da Dropbox, assim como através de aplicações que podem ser descarregadas através do seu *website* (para sistemas operativos presentes em computadores), ou através das lojas de *Apps* para os sistemas operativos correspondentes nos *Smartphones* (como Android e iOS), simplificando o acesso às contas pessoais dos utilizadores, assim como a interação com os ficheiros [38].

Sendo um serviço bastante popular, a Dropbox criou a sua própria API, possibilitando a programadores a inserção da Dropbox nas suas aplicações. A primeira API (API V1 ou *Core* API) permitiu a criação de diversas aplicações

com a Dropbox incorporada, apresentando um SDK para diversas linguagens de programação, como PHP, Python, Android, iOS, *etc*. Com os SDKs também veio a necessitada documentação, e em muitos dos SDKs, tutoriais para ajudar a integração da API no projeto [39].

Com o passar do tempo e com a evolução constante da Dropbox, a API V1 deixou de ser atualizada, sendo criado uma nova API, a API V2. Esta nova API, apesar de ser a correntemente a suportada oficialmente, não dá o mesmo suporte a certas linguagens como a antiga API, focando-se em outras linguagens, como objetive-C (linguagem não suportada na API anterior) e Javascript, mas suportando na mesma linguagens como Python e Java [40].

A API V2 utiliza o protocolo OAuth 2.0 [41] para autenticação. Este processo leva a aplicação do utilizador a ser reencaminhada para um url de autenticação, onde o utilizador deve inserir as suas credenciais para validar a sua conta, e sendo feito a dita validação, o OAuth 2.0 volta para a aplicação, oferecendo à aplicação um *token* de acesso [42].

Para integrar a API numa aplicação o programador precisa de criar uma Dropbox *App* primeiro. Com a *App* criada o programador recebe um *client* ID e *App secret*. Ao inserir a API na aplicação o programador precisa de usar pelo menos o *client* ID para poder proceder ao processo de autenticação.

2.4.2 One Drive

Assim como o Dropbox, o One Drive também é um serviço de *hosting* de ficheiros criado pela Microsoft em Agosto de 2007.

Sendo um serviço de *hosting*, é possível para utilizadores guardarem os seus ficheiros, assim como fazer *share* dos ditos ficheiros com outras pessoas. Para aceder a uma conta One Drive, geralmente o utilizador autentica a sua conta no *website* da One Drive. Para quem não quer estar sempre a ligar o *internet browser* para auntenticar a sua conta é possível fazer *download* da aplicação para o computador ou para *Smartphone* através das respetivas *App Stores*.

Uma das vantagens do One drive em relação a outros repositórios é a capacidade de interagir com o Microsoft Office, podendo abrir e guardar ficheiros do One Drive diretamente nas aplicações do Office (Word, Excell, *etc*), e com a

aplicação no computador é possível sincronizar os documentos facilmente e até trabalhar em conjunto com outras pessoas no mesmo documento partilhado [43].

Como o Dropbox, o One Drive também possui a sua própria API. Esta API possui SDKs para integrar em diversas aplicações. Para quem só precisa de abrir e guardar ficheiros, o One Drive possui a *File picker* SDK, disponível para Android, iOS, Windows e Javascript. Para quem precisa de mais controlo/-funcionalidades, utiliza a One Drive API SDK, disponível para Android, iOS, Python e Windows .NET, C# e Xamarin [44].

O One Drive utiliza o protocolo OAuth 2.0 [41] para autenticação (como a Dropbox API). Depois de autenticar a conta, o utilizador recebe um *access token* para poder interagir com a conta através da aplicação [45].

Para integrar o One Drive API numa aplicação, é necessário primeiro criar uma One Drive *App*, obtendo um *client* ID e uma *app secret*. Estes dois deverão ser incluídos posteriormente na aplicação que utiliza a One Drive API (de forma similar à inclusão no Android na imagem anterior) [46].

2.4.3 Google Drive

Assim como a Microsoft, a Google também criou o seu próprio repositório *online*, nomeadamente o Google Drive, que foi lançado para o mundo em 24 de Abril de 2012 [47].

Este repositório oferece 15 GB de armazenamento gratuito [48], sendo efetivamente o repositório que oferece mais espaço de armazenamento gratuito dos três mencionados, com o Dropbox a oferecer 2 GB [49] e o One Drive a oferecer 5 GB [50], mas ambos com opções extras para aumentarem um pouco o espaço de armazenamento sem ter de recorrer a pagamentos/contas *premium*.

Com o Google Drive, é possível partilhar documentos com outras pessoas, fazer *upload* e *download* de documentos (atributo comum entre serviços de *hosting* de ficheiros). O Google Drive funciona juntamente com o Gmail e Google fotos, sendo possível ir buscar ficheiros diretamente ao google drive para anexar ou criar cópias dos ficheiros para o Google Drive [51].

Assim como os repositórios anteriormente referidos, o Google Drive também possui a sua própria API para incorporar o Google Drive em aplicações. As três plataformas onde a Google Drive funciona, são o Android e iOS (aplicações) e HTTP (*web apps* corridas em *browsers*) [52].

Para o Android, pode ser observado uma forma similar de utilizaçã ao Dropbox/One Drive, sendo necessário fazer *download* do Android SDK (para desenvolvimento em Android) e do Google *Play Services* SDK (que contém a API requerida). Depois é necessário registar a aplicação para obter uma *app ID* (similar ao Dropbox e ao One Drive), que será posteriormente utilizado na aplicação em desenvolvimento. Esta API também utiliza o protocolo OAuth 2.0 [41] para autenticação [53].

Para o iOS, o Google Drive utiliza uma API RESTful [54], oferecendo uma Objective-C *client library*, localizada no github, sendo uma *framework* Cocoa com interfaces que cobrém este serviço *web* [55].

Para HTTP, a Google Drive API usa uma REST API, que pode ser utilizada juntamente com linguagens e *frameworks* como javascript, .NET, Python e Ruby, entre outros [56]. Assim como a API para o Android e para o iOS, a REST API tambem utiliza o protocolo OAuth 2.0 [41] para autenticação. Quando o utilizador precisa de aceder aos seus documentos, a aplicação pergunta à Google para um certo nível de acesso (um URL que define a liberdade de acesso de dados que o utilizador permite a aplicação aceder). Depois de o utilizador é levado para uma página de consentimento, sendo depois reencaminhado para a sua aplicação, obtendo um *token* de acesso (equivalente ao Dropbox e ao One Drive). [57]

Capítulo 3

Especificação do Sistema

No capítulo 1, foi apresentada a ideia/conceito geral do projeto a desenvolver nesta tese de dissertação, nomeadamente controlar um projetor através de um *Smartphone*. Para isso é necessário recorrer ao auxílio de um sistema embebido para estabelecer ligação com o projetor, e de duas aplicações: uma para ser executada num sistema embebido e outra para executar num *Smartphone*. Para isso é necessário escolher qual o sistema embebido, a plataforma móvel e o repositório a utilizar no projeto.

No capítulo anterior foram apresentados alguns exemplos de mercado similares ao pretendido nesta dissertação. De seguida foram apresentadas as considerações para os três atores descritos anteriormente (sistema embebido, plataforma móvel e repositório), explicando as suas capacidades e utilidades, assim como as suas especificações.

Assim, este capítulo começa por especificar as escolhas feitas, assim como as razões por detrás dessas escolhas. De seguida, é apresentada a arquitetura do sistema "D-Lecture" de uma forma geral e explicado o seu funcionamento, assim como as ferramentas utilizadas no desenvolvimento das duas aplicações (Android e Chromecast).

Explicado o funcionamento do sistema de forma "global", entra-se em maior detalhe nas duas aplicações (sistema embebido e *Smartphone*), mostrando como interagem entre si, como cada uma interage com o repositório, e como o Chromecast interage com o projetor.

3.1 Plataforma Móvel

No capítulo 2 foram apresentados três plataforma móveis, sendo essas três as mais populares no mercado (Tabela 2.1). Como se pode verificar o Android continua a liderar no mercado relativamente ao número de vendas, comparativamente com as duas plataformas móveis seguintes, que nem metade das vendas do Android conseguiram. Isto provém, de certa forma, da popularidade enorme do Android. Isto é um fator de peso, visto que o sistema precisa de poder ser utilizado por diversas pessoas, e com a popularidade do Android, cobre-se logo a maior parte dos *Smartphones* em corrente utilização. A tabela 3.1 compara as três plataformas móveis de uma forma simplificada após uma análise breve à tabela 2.2, com uma escala entre "Mau", "Normal" e "Bom", sendo possível reparar nas vantagens do Android em relação ao resto.

	Android	**iOS**	**Windows Phone**
Desenvolvimento de Aplicações	Bom	Bom	Bom
Suporte de Desenvolvimento	Bom	Bom	Normal
IDE	Bom	Normal	Mau
Comunidade	Bom	Bom	Normal

TABELA 3.1: Comparação entre Plataformas Móveis (simplificado da tabela 2.2)

A nível de desenvolvimento de aplicações o Android é similar aos restantes, no entanto o suporte de desenvolvimento coloca o Windows Phone atrás do Android e iOS, e com variedade enorme de IDEs que suportam o desenvolvimento para Android, este torna-se a escolha mais apropriada para a plataforma móvel a utilizar. Por fim é possível notar que o Android possui uma comunidade enorme por trás, sendo possível encontrar diversos *websites* e mesmo vídeos com tutoriais para quem quer começar a programar para o Android, assim como *websites* para tirar dúvidas.

Sendo assim, a plataforma móvel escolhida para utilização nesta dissertacção foi o Android.

3.2 Sistema Embebido

Para escolher o sistema embebido a utilizar convém ter em conta certos aspetos, nomeadamente: os custos, a simplicidade de programação e a necessidade de componentes extra. Assim, a Raspberry Pi suporta várias linguagens de programação, apresentando também a possibilidade de escolha entre diversos sistemas operativos.

O Chromecast possuí o seu próprio sistema operativo, correndo diversas aplicações, denominadas *web apps*. Adicionalmente o Chromecast não necessita de componentes extra, vindo equipado com os cabos necessários (micro USB para alimentação e HDMI [24] para ligar ao projetor, caso este possua entrada HDMI [24]), enquanto que para a Raspberry Pi é necessário comprar componentes extra. Isto leva a um aumento de preço necessário para usufruir das funcionalidades requeridas para a implementação, colocando o custo da Raspberry Pi superior ao do Chromecast. A tabela 3.2 apresenta a comparação entre os preços de ambos.

Item	**Custo Raspberry Pi B+**	**Custo Chromecast**
Cabo de alimentação Micro USB	10 $	-
Cartão 8 GB microSD	8 $	-
Pen Wi-Fi	3 $	-
Cabo HDMI	4 $	-
USB *card Reader* (instalação do SO)	7 $	-
Caixa para Raspberry pi (opcional)	15 $	-
Raspberry Pi B+	25 $	-
Chromecast	-	39 €
CUSTO TOTAL	72 $ = 65.86 €(Aprox.)	39 €

TABELA 3.2: Comparação entre preços

Um aspeto negativo do Chromecast em relação à Raspberry Pi é o facto de oferecer um canal de comunicação aberto a dispositivos que estejam conectados à mesma rede Wi-Fi [4] que o Chromecast, que em certos casos poderá causar problemas de segurança na comunicação.

Em relaccão à Apple TV, este dispositivo possui um grande ponto negativo para o proposto. Esse ponto é o preço do produto, que custa 179 €, para a

versão de armazenamento de 32 GB e 229 €para a versão de 64 GB. Isto é um preço exagerado para o que se pretende implementar na dissertação, e com o facto de o cabo HDMI [24] não ser incluído, aumentando ainda mais o preço pela falta de componentes necessários [58].

O custo superior, a necessidade de componentes extra, o aumento da complexidade da montagem na Raspberry Pi, e o custo do Apple TV e a necessidade de arranjar material extra, optou-se por utilizar o Chromecast. Como razão adicional para o utilizar, existe a razão de explorar as suas capacidades em maior profundidade, verificando a sua viabilidade como um sistema embebido.

3.3 Repositório *Online*

A escolha entre os repositórios *online* mencionados (Dropbox, One Drive e Google Drive) foi uma escolha difícil.

A nível de armazenamento básico, o Google Drive apresenta o maior espaço de armazenamento, com 15 GB, seguido do One Drive com 5 GB de espaço, e por fim o Dropbox com 2 GB. No entanto, convém referir que o Dropbox apresenta formas de aumentar o espaço de armazenamento sem ter de recorrer a contas *premium*, sendo possível obter um máximo de 16 GB, utilizando o sistema de *referral* a amigos, obtendo 500 MB por cada *referral* [59].

Ao nível das APIs, todos os repositórios referidos apresentam funcionalidades similares, principalmente nas implementações para os *Smartphones*. Neste ponto, não existe diferença relevante que justifique o uso de um repositório em detrimento dos outros.

Por fim, refere-se a popularidade dos repositórios. Como referido no capítulo anterior, o Dropbox é o mais popular dos três, possuindo mais de 100 milhões de utilizadores, sendo o repositório mais utilizado no mundo inteiro. A tabela 3.3 apresenta uma comparação simples entre os serviços de *hosting* de ficheiros.

Critérios	Repositório
Armazenamento	Dropbox
API	Iguais
Popularidade	Dropbox

TABELA 3.3: Tabela de comparação dos repositórios

Assim, decidiu-se utilizar o Dropbox como o repositório *online* para esta dissertação.

3.4 *Overview* do Sistema

O projeto "D-Lecture" propõe uma alternativa *Low-Cost* a projetores Wi-Fi [4] que podem ser controlados por *Smartphones*, ou seja, o desenvolvimento de um sistema capaz de permitir a projetores mais antigos funcionalidades disponibilizadas nos novos projetores, assim como simplificar o processo de fazer apresentações.

Este sistema é composto por um sistema embebido (Chromecast), que está conectado a um projetor com entrada HDMI [24] e corre uma aplicação que comunica com outra aplicação que é executada num *Smartphone*, sendo que ambas as aplicações são capazes de comunicar com o repositório (Dropbox).

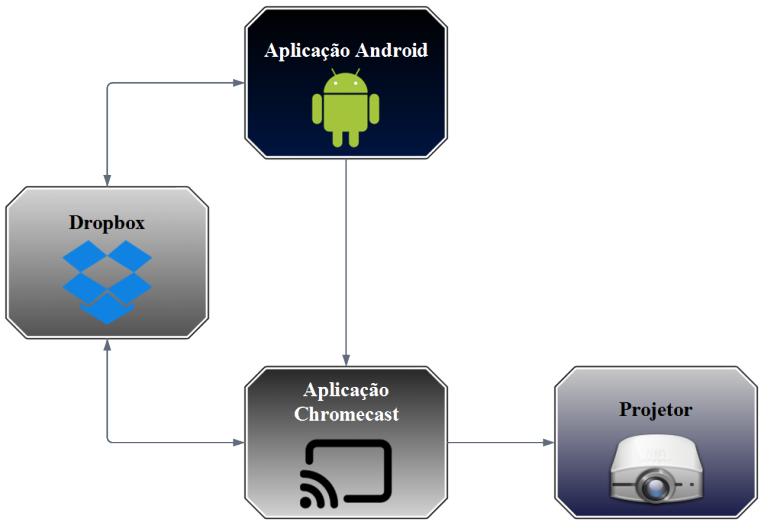


FIGURA 3.1: *Overview* do sistema na sua totalidade

Isto origina um sistema com quatro componentes, como se pode pode verificar na figura 3.1, que irão comunicar e interagir entre si. A aplicação Android comunica com o Dropbox, onde o utilizador necessita de fazer *login* com a sua conta, dando à aplicação Android acesso à conta Dropbox do utilizador. A aplicação Android também comunica com a aplicação Chromecast. Esta aplicação é denominada *web app*, sendo escrita em HTML5 e Javascript. Para comunicarem uma com a outra, utiliza-se comunicação *wireless* (Wi-Fi [4]), onde é criado um canal específico para a comunicação. Com a conexão estabelecida, a aplicação Android envia mensagens, que servirão como "comandos", que a *web app* a correr no Chromecast terá de executar. Quando o utilizador escolhe um ficheiro para visualizar, é enviado um "comando" da aplicação Android para a aplicação Chromecast. O Chromecast ao receber o comando, faz um pedido à conta Dropbox do utilizador para obter o ficheiro. Por fim , o projetor apresenta no ecrã o ficheiro e as interações do utilizador, sendo essa projeção feita através do cabo HDMI [24] que conecta o Chromecast ao projetor, permitindo assim ao utilizador fazer a apresentação desejada.

3.4.1 Ferrramentas de Desenvolvimento

Para o desenvolvimento das aplicações, foram analisados vários IDEs existentes no mercado, sendo decidido utilizar dois deles: um para o desenvolvimento da aplicação Android para o *Smartphone* e outro para o desenvolvimento da aplicação *Web App* para o Chromecast.

Aplicação Android

Para desenvolver a aplicação móvel para o Android foi escolhido o Android Studio.

O Android Studio é o IDE que a própria Google aconselha para o desenvolvimente de aplicações móveis, uma vez que este oferece novas funcionalidades comparativamente com outros IDEs, tais como: um editor de *layout* com suporte ao tema de edição com *drag and drop*, suporte para a Google *Cloud Platform*, *code Templates* para criação de funcionalidades normais em aplicações, entre outras [60].

O Android Studio permite não só criar aplicações para um Android *Smartphone*, como aplicações para comunicar com o Chromecast, sendo só necessário fazer o *download* do SDK do Google *Play Services*.

Com o Android *build system* integrado no IDE, já não é utilizado o sistema Apache Ant [61] usado no Eclipse ADT, permitindo ao Android Studio personalizar, configurar e aumentar o processo de *build*, e outras funcionalidades como reutilização de código [60].

O Android Studio possuí a capacidade de criar dispositivos virtuais, permitindo assim emular diversos dispositivos com o SO Android, desde *Smartphones* a televisões, dando ao programador a opção de testar as suas aplicações sem ter de adquirir o respetivo dispositivo [62].

Possuindo um SDK *Manager*, é possível ao Android Studio adicionar as APIs facilmente ao IDE e, juntamente com updates a APIs de versões mais antigas, permite a um programador criar aplicações para dispositivos antigos e novos de forma igual.

Por fim, como o Android Studio foi desenvolvido especificamente para o SO Android, não corre o risco de ter funcionalidades a correr com problemas, com as APIs a serem regularmente atualizadas e também possuindo uma comunidade enorme.

Aplicação Chromecast

O IDE escolhido para desenvolver a aplicação que correrá no Chromecast será o jetbrains Webstorm.

O Webstorm é um IDE capaz de criar código tanto para programas/aplicações *client-side* como para *server-side*, detetando erros de escrita de código no momento. Este IDE suporta muitas *frameworks*, linguagens e bibliotecas para programação *web*, como Javascript, HTML5, CSS, AngularJS, entre outros [63].

O Webstorm possuí a capacidade de fazer depuração a código *client-side*, estando imbutido diretamente no IDE, assim como a capacidade de *tracing* (usando o spy-js), podendo assim identificar pontos de estrangulamento [64]. O webstorm também é capaz de guardar o historial de mudanças nos ficheiros,

sendo capaz de voltar a versões antigas caso algum erro ou modificação acidental aconteça. Adicionalmente, o Webstorm é capaz de incorporar novas *Frameworks* e ferramentas, aumentando assim o número de funcionalidades. Por fim, o IDE também contém o seu próprio terminal, podendo assim correr comandos sem ser necessário sair do IDE [65].

Sendo assim um IDE robusto, rico em funcionalidades, ferramentas e versátil a nível de linguagens de programação e *frameworks*, o Webstorm é a escolha indicada.

3.4.2 Aplicação Android

Dentro da aplicação Android, existem dois casos de funcionalidades em que a aplicação Android não comunica com outros "componentes" exteriores. O primeiro é a opção de o utilizador poder modificar o brilho do ecrã. O segundo caso é a opção de modificar o idioma de acordo com a escolha do utilizador. Por fim existe outro caso onde o utilizador precisa de ir de encontro a certos requisitos antes de poder ir para o "menu" principal da aplicação.

Modificação do Idioma

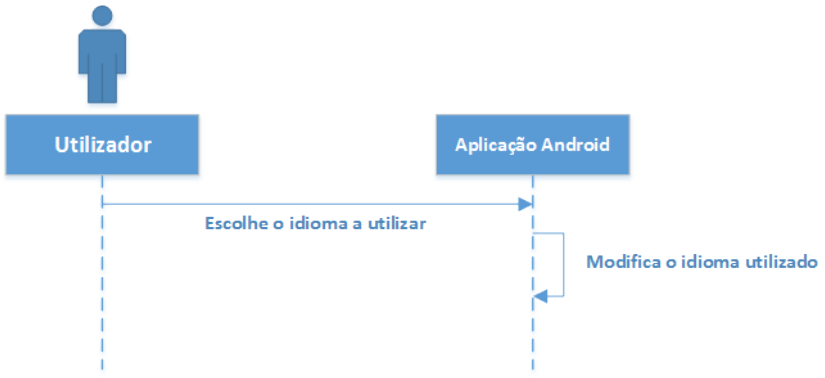


FIGURA 3.2: Diagrama sequencial da escolha do idioma da aplicação

Tal como ilustra a figura 3.2, para ser possível modificar o idioma é apresentado ao utilizador uma lista de idiomas para escolher. O utilizador escolhe o idioma desejado, ordenando assim à aplicação Android para modificar o idioma utilizado na aplicação.

Modificação do Brilho

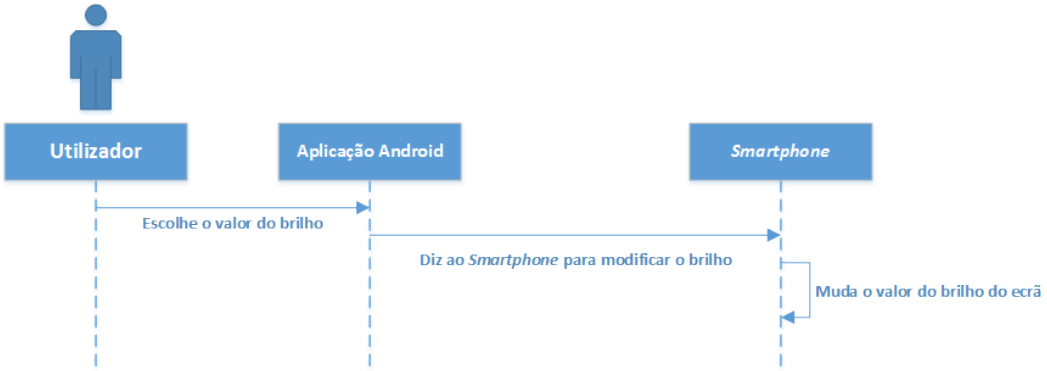


FIGURA 3.3: Diagrama sequencial da escolha do brilho da aplicação

A figura 3.3 apresenta o processo de modificação do brilho do ecrã no *Smartphone*. O utilizador escolhe o valor do brilho que deseja para o ecrã. A aplicação então acede aos dados do Smartphone, modificando o brilho do ecrã para corresponder ao desejado.

Menu Principal

Para o utilizador poder entrar no menu principal da aplicação Android, precisa primeiro de satisfazer certos requisitos.

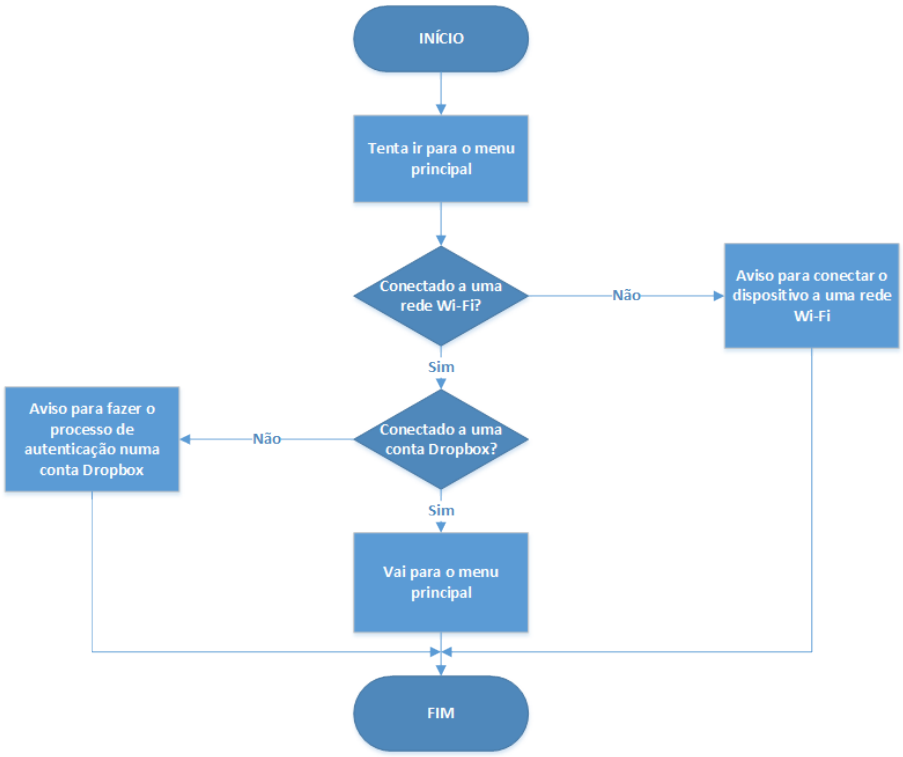


FIGURA 3.4: Fluxograma de requerimentos para entrar no menu principal

Como se pode visualizar na figura 3.4, as condições são verificadas por ordem, com a conexão Wi-Fi [4] a ser verificada em primeiro lugar. Caso o *Smartphone* não esteja conectado a uma rede Wi-Fi [4], é enviado para o ecrã uma notificação para informar o utilizador para se conectar a uma rede Wi-Fi [4] primeiro. Caso a primeira condição seja verificada, a aplicação verifica de seguida se o utilizador se conectou a uma conta Dropbox. Assim como a conexão a uma rede Wi-Fi [4], se o utilizador não se conectou é enviado uma notificação para o ecrã a pedir para se conectar primeiro a uma conta Dropbox. Quandos ambas as condições são ambas validadas, é então possível o utilizador entrar no menu principal.

3.4.3 Interação Android-Dropbox

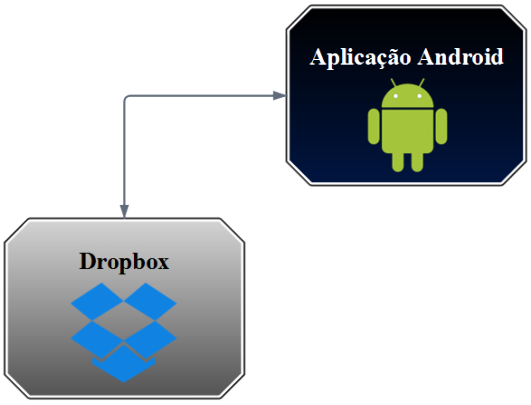


FIGURA 3.5: *Overview* da interação entre o Android e o Dropbox

A figura 3.5 apresenta um *overview* da interação entre a aplicação Android e o repositório *online*. A interação entre a aplicação Android e o Dropbox resume-se a dois casos: o primeiro é o caso em que o utilizador se conecta inicialmente à sua conta, com o segundo e último caso a ser o momento em que o utilizador se encontra a navegar entre as pastas da sua conta para encontrar o ficheiro pretendido, obtendo o *link* para o Chromecast poder fazer *download* o ficheiro numa fase posterior.

Login no Dropbox

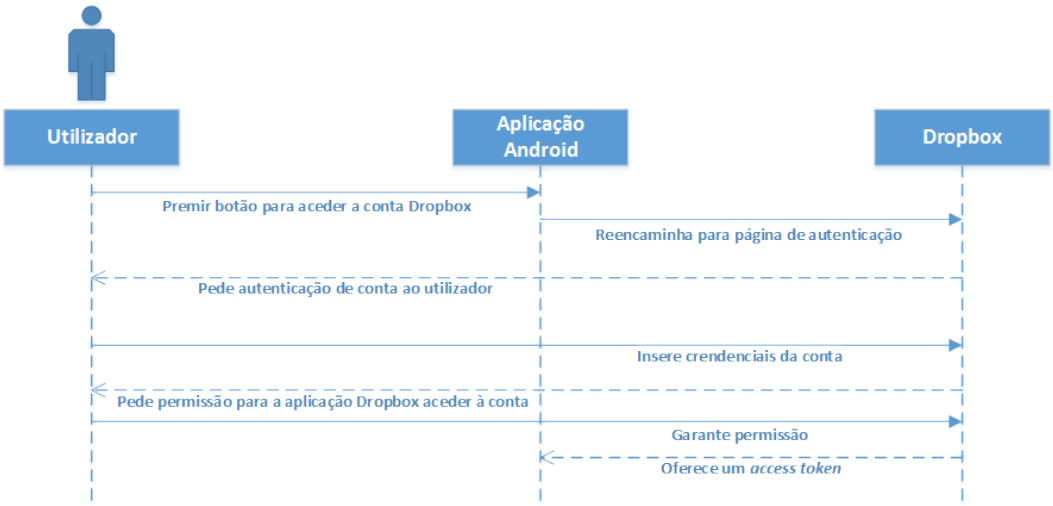


FIGURA 3.6: Diagrama sequencial para conectar conta Dropbox à aplicação

O diagrama sequencial da figura 3.6 mostra o processo normal para a aplicação se conectar a uma conta Dropbox. O utilizador começa por pressionar o botão para *login*. De seguida é reencaminhado para uma página de autenticação, onde lhe é pedido para inserir as credenciais da sua conta. Depois das credenciais serem introduzidas, é pedido ao utilizador permissão para a aplicação do Dropbox aceder à conta do utilizador. Depois de o utilizador permitir, a aplicação envia um *token* de acesso para a aplicação Android. Assim, a aplicação Android pode interagir com a conta Dropbox do utilizador.

Escolha do Ficheiro no Dropbox

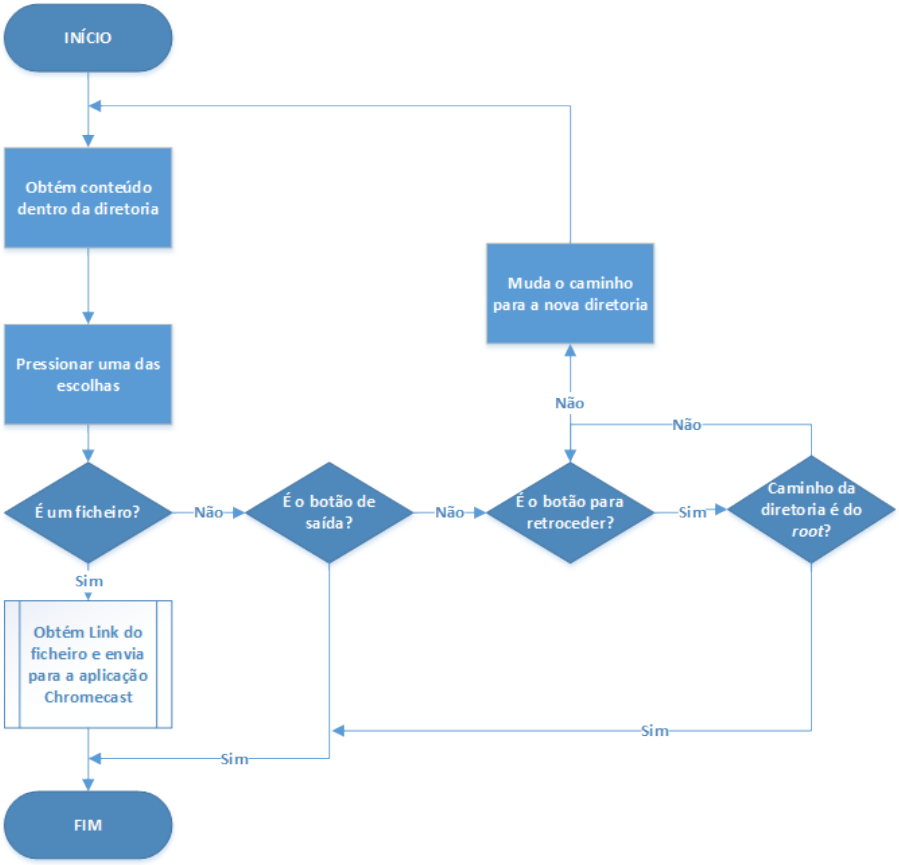


FIGURA 3.7: Fluxograma para navegar pelas diretorias para escolher o ficheiro desejado

Quando o utilizador pretende escolher um ficheiro para projetar no ecrã, primeiro a aplicação Android precisa de aceder ao conteúdo da sua conta Dropbox. Como a figura 3.7 demonstra, quando o utilizador quer escolher um ficheiro, é primeiro mostrado no ecrã o conteúdo da diretoria atual (no início

a diretoria é a *root* da conta). O utilizador depois pressiona uma das opções. Caso seja um ficheiro, a aplicação irá criar um *link* para enviar para a aplicação a correr no Chromecast. Caso não seja, verifica se o utilizador quer sair do modo "escolha de ficheiro", e caso o utilizador não deseje, a aplicação verifica se pressionou o botão para retroceder. Caso o botão para retroceder tenha sido pressionado, é verificado se o utilizador está na diretoria *root*, e caso esteja, sai do modo "escolha de ficheiro", se não, regressa para a diretoria anterior. Se nenhum dos casos anteriores for verificado, então é porque o utilizador pressionou uma subpasta, onde tem acesso ao conteúdo que se encontra dentro da diretoria.

3.4.4 Interação Android-Chromecast

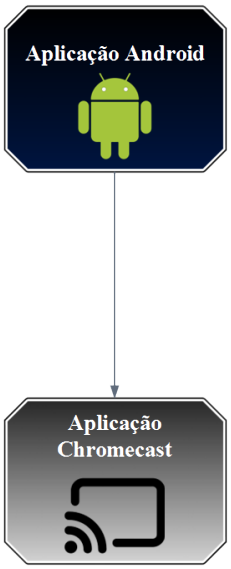


FIGURA 3.8: *Overview* do sistema entre o Android e o Chromecast

A figura 3.8 representa um *overview* da interação entre ambas aplicações, com a existência de três interações no total. O primeiro caso é na escolha do ficheiro, onde a aplicação Android envia o *link* do ficheiro. O segundo é no modo laser, onde é criado um ponto no ecrã (através da aplicação no Chromecast), e a aplicação Android envia coordenadas para mover o ponto laser, assim como mudar de página. E por fim, o terceiro caso é no modo de desenho, onde a aplicação Android não só envia comandos para mudar de página, também envia coordenadas para desenhar no ecrã, e um comando para modificar a cor a ser utilizada.

Conexão entre Aplicações

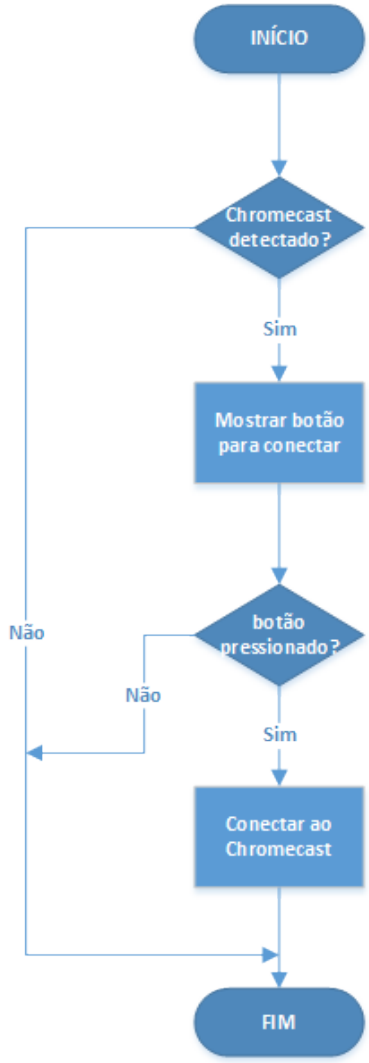


FIGURA 3.9: Fluxograma para ambas as aplicações serem conectadas

Para ambas as aplicações comunicarem uma com a outra, é preciso conectarem-se uma à outra. Para isto ser possível, é necessário o *Smartphone* e o Chromecast estarem ligados à mesma rede Wi-Fi [4]. Como é possível visualizar no fluxograma da figura 3.9, quando ambos estão conectados à mesma rede, é mostrado o botão de conexão ao dispositivo na aplicação Android. Quando o botão é pressionado, a aplicação Android conecta-se ao Chromecast, iniciando a aplicação Chromecast.

Envio do *Link* do Ficheiro entre Aplicações

FIGURA 3.10: Diagrama sequencial para o envio do *link* do ficheiro desejado entre aplicações

Para o envio do ficheiro, a aplicação Android precisa primeiro que o utilizador escolha o ficheiro. Assim que o ficheiro é escolhido, a aplicação Android cria um *link*, que envia para a aplicação Chromecast (como se pode verificar no diagrama sequencial da figura 3.10).

Modo Laser

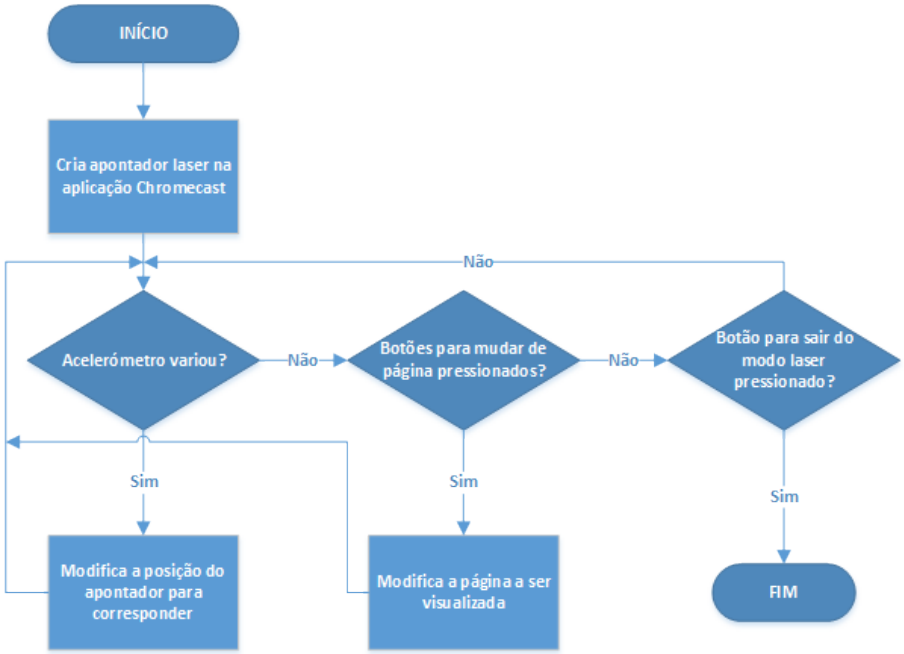


FIGURA 3.11: Fluxograma para o envio de comandos durante a utilização do apontador laser

A figura 3.11 apresenta o fluxograma de como o modo "apontador laser" funciona. Assim que o utilizador entra nesse modo, é criado um apontador laser por parte da aplicação Chromecast para projetar no ecrã. A aplicação Android irá então prestar atenção a variações no Acelerómetro do *Smartphone*, e caso detete variações, é enviado um comando à aplicação Chromecast para atualizar as coordenadas do apontador laser no ecrã. Caso não detete variações, tenta detetar se foi pressionado um botão para mudar a página, onde caso se verifique a página é modificada para a nova. Se nenhum dos casos anteriores for verificado, é verificado se o utilizador premiu o botão para sair do modo "apontador laser".

Modo Desenho

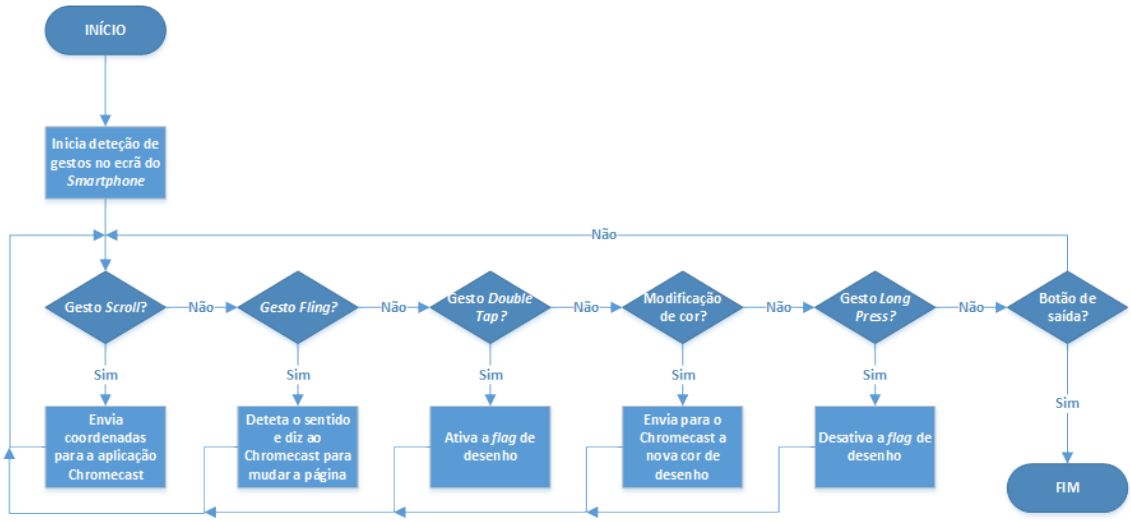


FIGURA 3.12: Fluxograma para o envio de comandos durante a utilização do modo de desenho

O outro modo que o utilizador pode utilizar durante uma apresentação é o modo "desenho" (representado pela figura 3.12). Este modo cria uma zona no ecrã onde é detetado movimentos feitos pelo utilizador (fora pressionar botões), onde cada movimento será comparado a um gesto, e dependendo do gesto os comandos enviados para a aplicação Chromecast serão diferentes. Os gestos detetados são quatro: *Fling*, *Scroll*, *Long Press* e *Double Tap*. Caso seja um gesto *Scroll*, é enviado para a aplicação Chromecast as coordenadas do gesto, para que a aplicação Chromecast saiba onde o utilizador está a premir em comparação com o ecrã *Smartphone*. Para o *Fling*, é enviado para o comando para mudar

de página depois de detetar o sentido do gesto (esquerda para retroceder e direita para avançar). Quando o gesto detetado é um *Double Tap*, a aplicação Android envia para a aplicação Chromecast o comando para ativar a *flag* de desenho, para que quando o gesto *Scroll* for detetado, a aplicação Chromecast começa a desenhar no ecrã (neste caso na página corrente). Para desativar a *flag* de desenho, através da deteção do gesto *Long Press*, que envia o comando da aplicação Android para a aplicação Chromecast.

Fora os gestos, é também detetado quando o utilizador pretende modificar a cor de desenho. Quando o utilizador escolhe uma cor, é enviado para a aplicação Chromecast um comando com o nome da cor, para que quando o utilizador comece a desenhar, os traços sejam da cor correspondente à escolhida pelo utilizador.

Por fim, verifica quando o botão para sair é pressionado, que em caso positivo sai do modo desenho, voltando para o menu principal.

3.4.5 Interação Chromecast-Dropbox

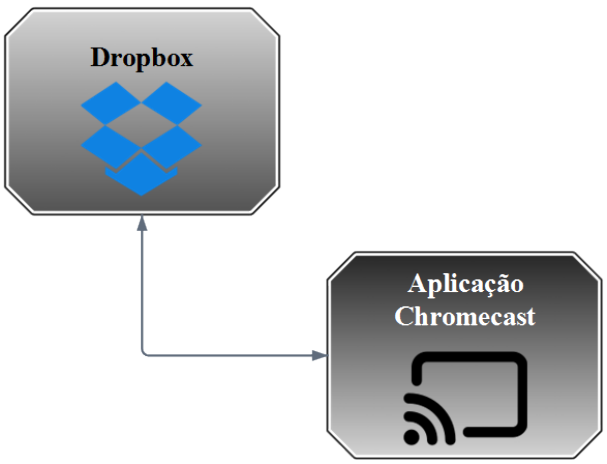


FIGURA 3.13: *Overview* da interação entre o Chromecast e a Dropbox

A figura 3.13 apresenta um *overview* da interação entre a aplicação Chromecast e o repositório *online*. O único caso existente é o envio do *link* do ficheiro da aplicação Android para a aplicação Chromecast. A aplicação Chromecast obtém o conteúdo do ficheiro, convertendo o conteúdo para numa fase posterior mostrar no ecrã.

Aquisição do Ficheiro para Projeção

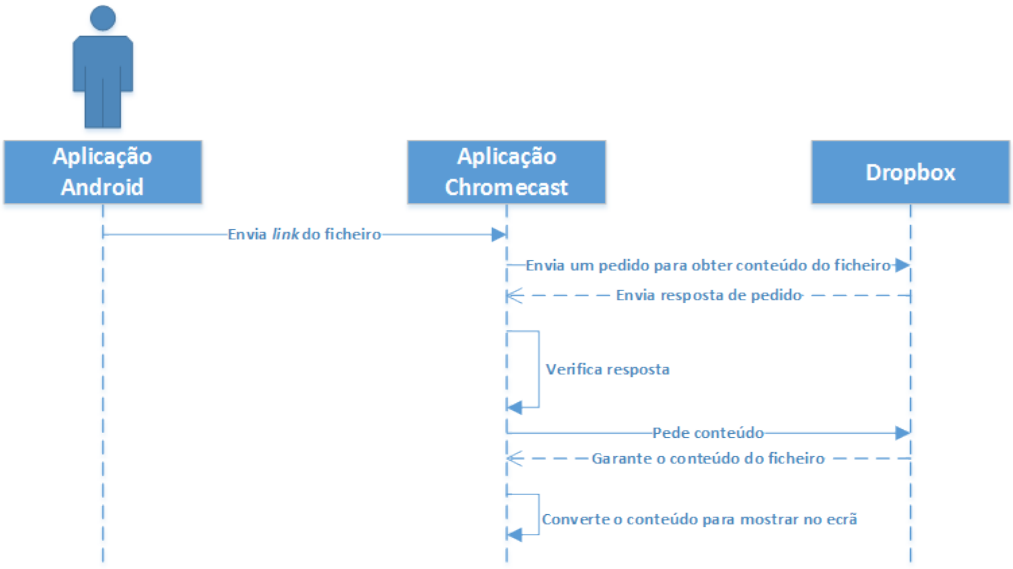


FIGURA 3.14: Diagrama sequencial para a aplicação adquirir o ficheiro e convertê-lo para projeção

A interação entre a aplicação Chromecast e o Dropbox é bastante simples. Como é possível verificar na figura 3.14, a aplicação envia o *link* para a aplicação Chromecast. Com esse *link* a aplicação Chromecast cria um pedido e envia-o para o Dropbox para obter o conteúdo do ficheiro. O Dropbox envia a resposta de volta, onde a aplicação Chromecast processa a resposta e caso seja positiva, obtém o conteúdo da resposta do Dropbox.

3.4.6 Interação Chromecast-Projetor

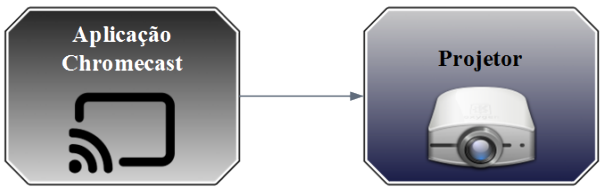


FIGURA 3.15: *Overview* do sistema entre o Chromecast e o Projetor

A interação entre o Chromecast e o projetor é retratada na figura 3.15. Só existe um caso, com a conexão entre os dois a ser através de um cabo HDMI [24] (imbutido no próprio Chromecast). O Chromecast fica então em *standby* até

que a aplicação Android se conecte. Assim que a aplicação Android se conecte ao Chromecast, inicia então a *web app*, sendo mostrada no ecrã a aplicação.

Projeção da Aplicação Chromecast

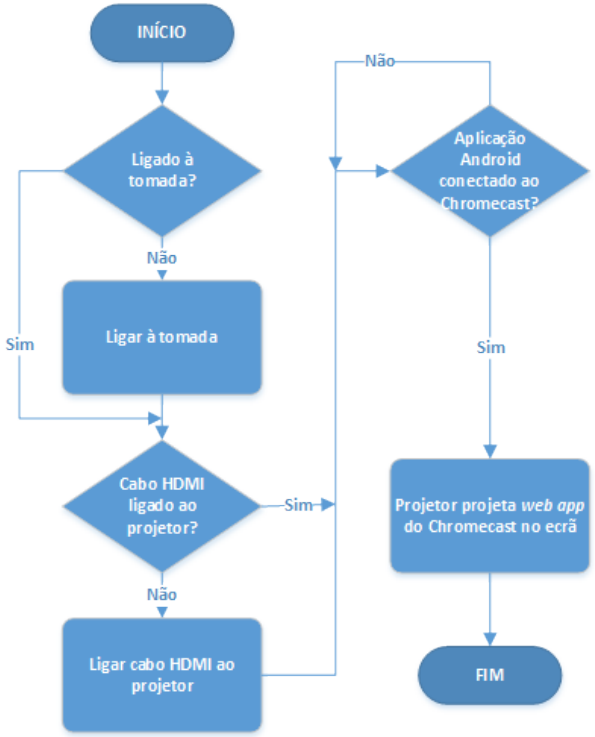


FIGURA 3.16: Fluxograma para a projeção da Aplicação Chromecast

O fluxograma representado na figura 3.16 explica de forma simples como o Chromecast interage com projetor para ser visualizada a aplicação Chromecast. Para esta ser projetada no ecrã, é necessário verificar três condições: i) Chromecast estar ligado a uma tomada, ii) o Chromecast estar ligado ao projetor, e iii) a aplicação Android conectar-se ao Chromecast, o que irá causar o início da execução da aplicação deste sistema embebido.

Capítulo 4

Implementação

Este capítulo aborda a implementação dos processos descritos no capítulo anterior. Como foi descrito no capítulo anterior, o sistema é composto por quatro componentes: Dropbox, aplicação Android, aplicação Chromecast e projetor. Foi mostrado o sistema na sua totalidade e como os componentes interagem entre eles de uma forma geral para dar a ideia de como é o fluxo de execução do sistema. De seguida foram descritas as comunicações/interações entres os componentes, desde a interação entre a aplicação Android com o Dropbox até à interação entre o Chromecast e o projetor, mostrando a sequência requirida para a aplicação Android se conectar ao Dropbox, assim como ir à procura do ficheiro desejado. Também foram apresentadas as funcionalidades internas à aplicação Android, nomeadamente o brilho do ecrã do *Smartphone*, os requisitos para entrar no menu principal da aplicação e, por fim, a modificação do idioma utilizado na aplicação. De seguida foram explicadas as interações entre ambas aplicações, mostrando como funciona a comunicação em ambos os modos (laser e desenho), como funciona o processo de conexão entre as aplicações e como a aplicação Android avisa a aplicação Chromecast de qual ficheiro que é suposto ser obtido do Dropbox. Por fim, foi mostrado como a aplicação Chromecast obtém o conteúdo do ficheiro desejado do Dropbox, bem como a aplicação Chromecast interage com o projetor para poder mostrar no ecrã a aplicação.

Assim, é apresentado o trabalho desenvolvido em relação ao sistema, ou seja, como os processos mencionados no parágrafo anterior foram implementados, dando uma vista com maior detalhe às interações entre os componentes. Também é dado maior ênfase à aplicação Chromecast, mostrando como são interpretados os comandos recebidos da aplicação Android.

4.1 Arquitetura do Sistema

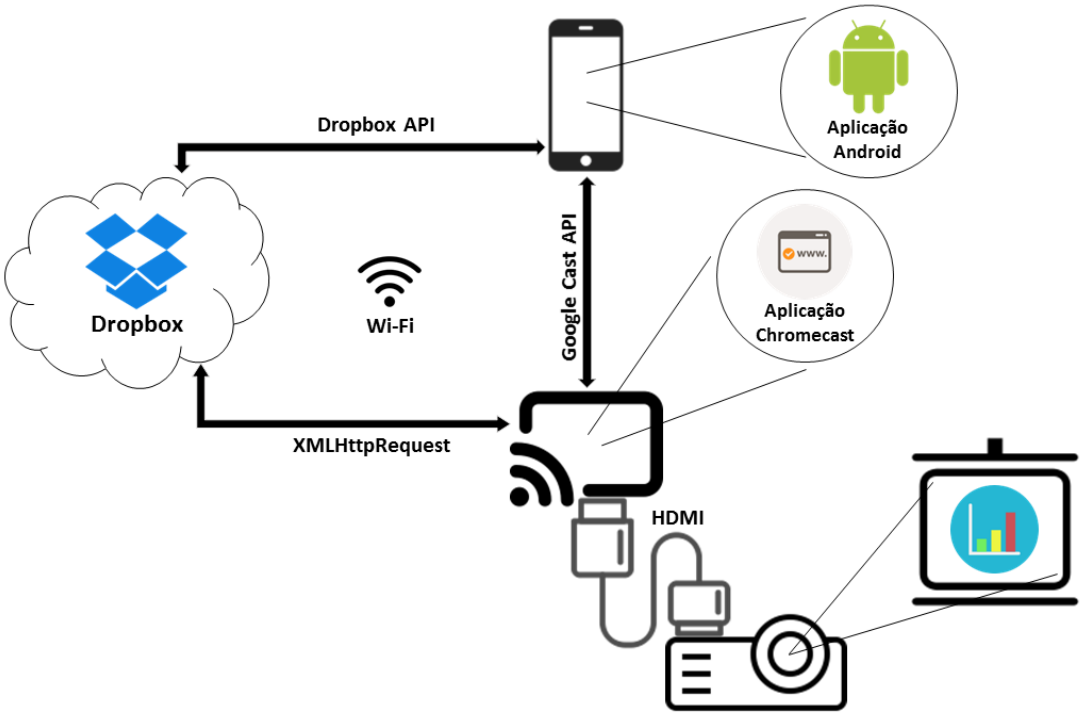


FIGURA 4.1: Arquitetura do sistema

No capítulo anterior já tinha sido apresentado um *overview* do sistema, que é composto por quatro componentes: o repositório *online* (Dropbox), uma aplicação a correr num *Smartphone*, uma aplicação *web* a correr no Chromecast e um projetor (necessário para a projeção).

Na figura 4.1, é apresentada a arquitetura do sistema com maior detalhe. A maior importância neste sistema é a conexão Wi-Fi, visto que as comunicações entre a aplicação Android, aplicação Chromecast e Dropbox necessitam que ambas as aplicações estejam conectadas a uma rede. Para ser possível à aplicação Android interagir com o Dropbox, é necessário utilizar o Dropbox API para o Android, que oferece as funções e classes necessárias para implementar o processo de comunição desejado. Assim como para o Dropbox, a comunicação entre ambas as aplicações serão feitas através da Google *Cast* API, onde cada aplicação utilizará a sua versão para ser possível comunicarem uma com a outra (biblioteca para a aplicação Android e um ficheiro javascript para aplicação Chromecast). Entre o Dropbox e a aplicação Chromecast, a comunicação é mais simples, onde a aplicação Chromecast faz um *XMLHttprequest* para o

Dropbox utilizando o *link*, recebendo a resposta do pedido, e em caso positivo obter o ficheiro para processar mostrar o seu conteúdo no ecrã. Por fim, a comunicação entre o Chromecast e o projetor é básica, onde só é necessário um cabo HDMI para o Chromecast comunicar com o projetor (Chromecast já vem com o cabo).

4.2 Aplicação Android

A aplicação Android é executada a partir de um *Smartphone*. Quando o utilizador inicia a aplicação é apresentada a atividade inicial, denominada de *MainActivity*. Esta atividade é reponsável pelo processo de *Login* no Dropbox, assim como serve de validador de que o utilizador só pode ir para o menu principal depois de se conectar a uma rede Wi-Fi e depois de se conectar a uma conta Dropbox. Quando o utilizador cumpre os dois requisitos, a atividade corrente inicia a atividade do menu principal através de um *Intent*.

```
private void startMainMenu(){
        ...
        Intent myIntent = new Intent(this, Menu.class);
        startActivity(myIntent);
    }
```

ALGORITMO 4.1: Arranque da atividade do menu principal

É criado um novo *Intent* com a classe do menu principal (*Menu*), e inicia o *Intent* como atividade. A figura 4.2 apresenta a transição entre as atividades.

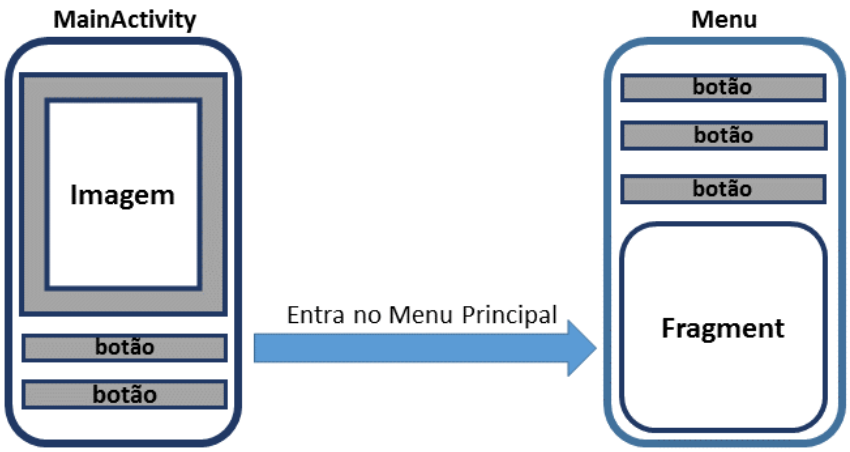


FIGURA 4.2: Transição de atividades

No menu principal, fora os botões também é colocado um *Fragment*. Este *Fragment* é inserido dentro de um *FrameLayout* (chamado *fragment_place*), e sempre que haja transição de um *Fragment* para outro, o novo será inserido no *FrameLayout* em vez do antigo.

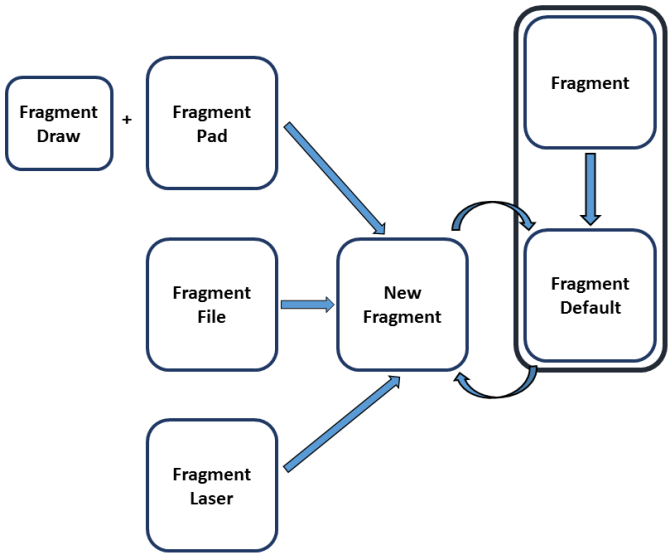


FIGURA 4.3: Transição entre os *Fragments*

Tal como é possível visualizar na figura 4.3, o *FrameLayout* é constituído por uma rotação entre quatro *Fragments*. O primeiro *Fragment* é o *Fragment Default*, que é posto no *FrameLayout* quando se entra no menu principal ou quando é fechado o *Fragment* que está atualmente no *FrameLayout*. Este *Fragment* é responsável por sair do menu principal, modificar o brilho do ecrã, e modificar o idioma utilizado.

O segundo *Fragment* é o *Fragment File*, que é responsável por mostrar os ficheiros e diretorias dentro da conta Dropbox, criação e envio do *link* do ficheiro, e por enviar o comando para começar a visualização do ficheiro na aplicação Chromecast.

O terceiro *Fragment* é uma combinação de dois *Fragments*, definidos como *Fragment Pad* e *Fragment Draw*. Estes *Fragments* possuem duas funções distintas, com o *Fragment Pad* a funcionar como o *Fragment* "pai" e o *Fragment Draw* como "filho". O *Fragment Pad* inicia o modo de desenho na aplicação Chromecast, dando ao utilizador também a opção de sair, assim como de escolher a cor. O *Fragment Draw* implementa um detetor de gestos, que dependendo dos gestos,

envia comandos diferentes para aplicação Chromecast, como movimentos do cursor, modificação de página e desenho.

O último *Fragment* é o *Fragment Laser*, que é responsável por ler dados do acelerómetro do *Smartphone* e enviar os dados para aplicação Chromecast, que permite manusear o apontador laser, com botões para avançar e retroceder entre as páginas do documento quando pressionados.

Para alternar entre os *Fragments*, é utilizado o método *changeFragment*, que recebe o nome do novo *Fragment*, cria uma instância, e inicia a transição do *Fragment* para o *fragment_place*.

```
public void changeFragment(String fragmentClass){
        Fragment frags = Fragment.instantiate(this, stringPackage + "."
                                + fragmentClass);
        FragmentManager fm = getFragmentManager();
        FragmentTransaction ft = fm.beginTransaction();
        ft.replace(R.id.fragment_place, frags);
        ft.commit();
    }
```

ALGORITMO 4.2: Processo de transição entre *Fragments*

4.2.1 Processos Internos

Os processos internos são descritos os mesmos que os referidos no capítulo anterior, ou seja: o *login* no menu principal, onde o utilizador deve cumprir dois requisitos para se poder autenticar na aplicação, a modificação do brilho do ecrã do *Smartphone*, onde o utilizador escolhe o valor desejado e, por fim, a modificação do idioma utilizado na aplicação, onde o utilizador escolhe qual prefere, modificando o nome dos botões da aplicação para o idioma correspondente.

Login no Menu Principal

Para o utilizador poder entrar no menu principal da aplicação, necessita de satisfazer dois requisitos: necessita de estar conectado a um rede Wi-Fi e de estar conectado a uma conta Dropbox.

Para o primeiro caso, é pedido ao *Smartphone* acesso à informação da conexão a uma rede Wi-Fi, obtendo de seguida a informação da rede. A verificação é feita pelo método *isNetworkAvailable()*, que pede ao sistema do *Smartphone* informações sobre a rede ativa, retornando um valor booleano para verificar se o dispositivo está ou não conectado a uma rede. Em caso negativo é apresentada uma mensagem de aviso no ecrã para informar o utilizador para conectar o *Smartphone* a uma rede.

```
private boolean isNetworkAvailable() {
        ConnectivityManager connectivityManager =
                (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo activeNetworkInfo = connectivityManager.getActiveNetworkInfo();

        return activeNetworkInfo != null && activeNetworkInfo.isConnected();
    }
```

ALGORITMO 4.3: Verificação da conexão a uma rede

Para o Dropbox é utilizado uma variável do tipo booleano chamada "*isLogged*". Esta variável, quando a aplicação inicia, é colocada com o valor "*false*", e só é colocada com o valor "*true*" caso o processo de autenticação do Dropbox tiver sucesso, indicando que o utilizador se conectou uma conta Dropbox. Se o utilizador tentar entrar quando o valor é "*false*", é colocado um *Dialog* no ecrã para avisar o utilizador a se conectar a uma conta Dropbox, igual ao processo de verificação de rede.

Por fim, caso os dois requisitos sejam cumpridos, o utilizador entra no menu principal quando pressiona o botão de entrada.

```
public void onClick(View view) {
        ...
        if(!isNetworkAvailable()){
                noWiFi();
                }
        else{
                if(!isLogged){
                        notLogged();
                        }
                else {
                        startMainMenu();
                        }
                }
        ...
        }
```

ALGORITMO 4.4: Verificação de requisitos

Modificação do Brilho

Para mudar o brilho do ecrã, o utilizador precisa de estar com o *Fragment Default* colocado na atividade do menu principal. No *Fragment* existe uma barra implementada por uma *seekBar*, que serve para o propósito de modificar o valor do brilho dependendo do valor que o utilizador colocar (deslize da barra).

Também é utilizada uma variável que é necessária para interagir com os valores de brilho da janela da atividade, para que seja possível realmente modificar o brilho no ecrã, denominada de *layoutParams*. Esta variável permite não só obter dados relativos à janela da aplicação, mas também modificar os valores.

A seguir é implementado um *Listener* na *seekbar*, para que sempre que o utilizador modifique o valor, o *Listener* detete e modifique o valor do brilho de acordo com o valor da *seekbar*. A variável que contém o valor da *seekbar* é a variável *progress*. Convém referir que a *seekbar* pode ir até ao valor 0, que significaria ecrã completamente escuro, sendo assim necessário colocar um limite inferior a nível de valores que serão aceites da variável *progress*. Caso o valor não seja menor que o limite inferior, o *layoutParams* obtém os atributos da janela da atividade, modifica o valor do brilho (*brightness*) e passa os novos atributos para a janela. Como ao modificar o *Fragment* causa a destruição do anterior, o valor da *seekBar* é guardado na variável *brightnessLevel* da atividade, para que quando o *Fragment Default* for novamente inserido na atividade, a *seekBar* apresente o valor correto.

```
private void setupSeekBar(){
        ...
        seekBar.setOnSeekBarChangeListener(
                new SeekBar.OnSeekBarChangeListener() {
                    @Override
                    public void onProgressChanged(SeekBar seekBar,
                                                  int progress, boolean fromUser) {
                        float brightness = progress / (float) 100;
                        if (brightness > 0.05f) {
                            layoutParams = getActivity().getWindow().getAttributes();
                            layoutParams.screenBrightness = brightness;
                            getActivity().getWindow().setAttributes(layoutParams);
                        }
                        ((Menu) getActivity()).brightnessLevel = progress;
                    }

                    @Override
                    public void onStartTrackingTouch(SeekBar seekBar) {}

                    @Override
                    public void onStopTrackingTouch(SeekBar seekBar) {}
                }
        );
    }
```

ALGORITMO 4.5: Modificação do brilho do ecrã

Modificação do Idioma

Para modificar o idioma utilizada, utiliza-se uma caixa de diálogo. Quando o utilizador prime o botão para modificar o idioma, aparece no ecrã uma caixa de diálogo a pedir para escolher o idioma desejado. Quando o utilizador prime a seta é mostrado as opções todas, onde o utilizador escolhe uma e prime o botão "OK".

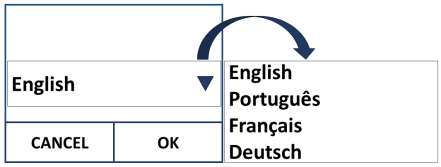


FIGURA 4.4: *Sketch* da caixa de diálogo

Tal como se pode ver na figura 4.4, cada idioma na caixa de diálogo é a uma *string*, contendo os valores "en", "pt", "fr" e "de", respetivamente. Estes valores indicam qual dos ficheiros de *strings* será invocado, ficheiros que possuem os nomes/texto dos botões (entre outros) num determinado idioma.

```
public void setLocale(String language) {
        ...

        Locale locale = new Locale(language);

        Resources resources = getResources();
        DisplayMetrics displayMetrics = resources.getDisplayMetrics();
        Configuration configuration = resources.getConfiguration();

        configuration.locale = locale;
        resources.updateConfiguration(configuration, displayMetrics);
        onConfigurationChanged(configuration);
    }
```

ALGORITMO 4.6: Modificação do idioma utilizado na aplicação

Para modificar o idioma utiliza-se um método chamado *setLocale*. Dentro deste método cria-se um objeto *Locale* com a string do idioma escolhida ("en", "pt", "fr" e "de"). De seguida é pedido os recursos da aplicação, retirando de lá os dados métricos de *display* (interessa a escala dos nomes/textos que irão ser modificados), assim como os dados de configuração da aplicação. Modifica-se o *locale* para condizer com o novo (indicando que será modificado o idioma utilizado na aplicação). Por fim, faz-se *update* ao recursos da aplicação, invocando de seguida o método *onConfigurationChanged* para executar as novas modificações, modificando assim o idioma nos nomes/textos dos botões (entre outros).

4.2.2 Interação com o Dropbox

Para interagir com o Dropbox, é necessário utilizar a API do Dropbox. Antes de começar a implementar métodos para as interações desejadas, é necessário primeiro preparar a aplicação para poder implementar os métodos. Primeiro é necessário adicionar a biblioteca da API do Dropbox às dependências no processo de compilação da aplicação.

```
dependencies {
        ...
        compile files('libs/dropbox-android-sdk-1.6.3.jar')
        ...
        }
```

ALGORITMO 4.7: Comando utilizado para compilar a biblioteca do Dropbox API

Para ser possível a API do Dropbox conectar-se a uma conta de utilizador, é necessário criar uma aplicação Dropbox no *website* para programadores (`https://www.dropbox.com/developers`). A aplicação Dropbox serve de "intermediário" entre a conta do utilizador e a aplicação Android. Ao criar a aplicação, é fornecido ao programador uma *App Key* e um *App Secret*, ambos necessários para implementar a conexão à aplicação Dropbox por parte da aplicação Android.

A seguir, é adicionado ao ficheiro *AndroidManifest* uma atividade do tipo *BROWSABLE*. Como já foi referenciado anteriormente, a API do Dropbox utiliza o protocolo 2.0 oauth, o que requer que o utilizador seja reencaminhado para uma página *web* para se poder conectar à sua conta Dropbox. Esta atividade é a responsável pela página *web*.

```
<activity
            android:name="com.dropbox.client2.android.AuthActivity"
            android:launchMode="singleTask"
            android:theme="@android:style/Theme.Translucent.NoTitleBar"
            android:configChanges="orientation|keyboard">
            <intent-filter>
                <!-- Change next line to be "db- followed by your app key" -->
                <data android:scheme="db-xxxxxxxxxxxxxxx" />
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.BROWSABLE"/>
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
```

ALGORITMO 4.8: Atividade para página de *login* Dropbox

Acesso à Conta Dropbox

Para ser possível a aplicação Android aceder a uma conta Dropbox é preciso criar uma sessão na aplicação. Dentro do método *initDropbox*, a sessão é criada

recorrendo à *App Key* e à *App Secret* da aplicação Dropbox, sendo guardada na variável *dbAPI*.

```
private void initDropBox() {
        AppKeyPair appKeys = new AppKeyPair(APP_KEY, APP_SECRET);
        AndroidAuthSession session = new AndroidAuthSession(appKeys);
        dbAPI = new DropboxAPI<>(session);
    }
```

ALGORITMO 4.9: Inicialização da sessão Dropbox

Quando o utilizador pretende conectar-se a uma conta Dropbox, precisa de carregar no botão de Dropbox *login*. Quando isso acontece, o utilizador inicia o processo de autenticação, onde é reencaminhado para a página de autenticação da conta Dropbox através da atividade *BROWSABLE*, pondo a atividade corrente em pausa.

Na página é pedido ao utilizador para inserir as credenciais de uma conta Dropbox. Quando as credenciais da conta forem validadas, é pedido ao utilizador se permite que a aplicação Dropbox se conecte à conta Dropbox. O utilizador carrega no botão de permissão, onde é então reencaminhado de volta para a aplicação. Assim que o utilizador regressa à aplicação, a atividade em pausa volta à normalidade, onde tenta acabar o processo de autenticação (através do método *onResume*). Caso o processo de autenticação tenha sido um sucesso, finaliza a criação da sessão, conectando a sessão à conta Dropbox.

```
@Override
public void onResume(){
        super.onResume();

        if (dbAPI.getSession().authenticationSuccessful()) {
                try {
                        dbAPI.getSession().finishAuthentication();
                        dbAPI.getSession().getOAuth2AccessToken();
                        isLogged = true;
                        }
                catch (IllegalStateException e) {
                        Log.e("error", "Authentication failed");
                        }
                }
        ...
        }
```

ALGORITMO 4.10: Finalização do processo de conexão entre aplicação e conta

Escolha do Ficheiro

Quando o utilizador prime o botão para escolher o ficheiro que deseja visualizar, é invocado o *Fragment File*. Dentro deste *Fragment*, existe uma lista que contém os ficheiros e diretorias da conta Dropbox a ser utilizada, sendo implementada por uma *ListView*.

A *ListView* cria campos, onde cada campo será ou um ficheiro ou uma diretoria. A diretoria de onde os ficheiros e subdiretorias são carragados é a diretoria *home*, que em implementação é representada por "/". Quando o utilizador pretende entrar numa subdiretoria, é só premir o campo dessa subdiretoria. O contéudo da subdiretoria é então inserido na *ListView* (fazendo *reset* à *ListView*). Quando o utilizador pretende retroceder para a antiga diretoria, é só pressionar o botão "Retroceder". Se o utilizador já estiver na diretoria base, é avizado que não pode retroceder mais. Caso contrário é procurado o "/" anterior, obtendo o conteúdo da diretoria anterior, sendo inserido na *ListView*, como se ilustra na figura 4.5.

FIGURA 4.5: Retrocesso para diretoria anterior

Em relação ao processo de obter o conteúdo de uma diretoria e colocar na *ListView*, é utilizada uma atividade asíncrona, denominada *ShowFiles*. Esta atividade (também referida como *AsyncTask*) é executada em *background*, não suspendendo a atividade à espera que acabe de obter o conteúdo total de uma diretoria. Dentro desta atividade, é obtido o adaptador da *ListView*, para que seja possível manipular a *ListView* dentro da *AsyncTask*. Para obter o conteúdo é primeiro obtido os metadados da diretoria. Para isso utiliza-se o método *metadata*, que utiliza o caminho para a diretoria, obtém até 1000 diretorias/ficheiros da diretoria e obtém os dados dos ficheiros (parâmetro *true*). A seguir são criados dois *arrays* auxiliares (*files* e *directory*), que são utilizados dentro de um ciclo *for*. O *array files* guarda o conteúdo de cada diretoria/ficheiro como uma *DropboxAPI.Entry*, para que seja possível extrair o caminho (*path*), que é guardado como *String* no *array directory*, que é utilizado para preencher a *ListView* através do método *onProgressUpdate*.

```
@Override
protected Void doInBackground(Void...params){
        try{
                DropboxAPI.Entry list = dbAPI.metadata(path, 1000, null, true, null);
                ArrayList<DropboxAPI.Entry> files = new ArrayList<>();
                ArrayList<String> directory=new ArrayList<>();
                for (DropboxAPI.Entry entry: list.contents){
                        files.add(entry);
                        directory.add(new String(files.get(count++).path));
                        }
                count = 0;
                fileNames = directory.toArray(new String[directory.size()]);
                for(String item: fileNames){
                publishProgress(item);
                }
        }
catch(DropboxException e){}
return null;
}
```

ALGORITMO 4.11: Inicialização dos *arrays* e aquisição dos metadados de uma diretoria

```
protected void onProgressUpdate(String... values){
        adapter.add(values[0]);
        super.onProgressUpdate(values);
    }
```

ALGORITMO 4.12: *Update* do adaptador da *ListView*

Quando o utilizador prime um ficheiro (neste caso PDF) na *ListView*, é enviado um *link* para a aplicação Chromecast. No entanto, é primeiro necessário não só fazer a verificação que realmente é um ficheiro (PDF), mas também criar o *link* que será enviado. Para isto foi utilizada outra atividade asíncrona, denominada *isDir*. Esta atividade é similar à atividade *ShowFiles*, obtendo também os metadados de um *path*, da mesma forma que a atividade *ShowFiles*. No entanto em vez de um ciclo *for*, utiliza um *if* para verificar se o *path* aponta para um ficheiro. Caso não aponte (neste caso é uma diretoria), a atividade envia um avizo para o *Fragment* para iniciar a atividade *ShowFiles* com o *path*, visto que é um caminho para uma diretoria.

Se for um ficheiro, é obtido o seu *sharelink*, que é utilizado para permitir a pessoas que não conseguem aceder à conta Dropbox visualizar o ficheiro. Com o *sharelink* obtido, é necessário modificar a sua estrutura para a aplicação Chromecast poder obter o conteúdo do ficheiro (explicado mais adiante). Assim, o *sharelink* é convertido para *String*, sendo depois modificada a primeira parte ("https://www") para "https://dl", e retirado a última parte ("?dl=0").

```
@Override
protected Void doInBackground(Void...params){
        try {
                DropboxAPI.Entry list = dbAPI.metadata(path, 1000, null, true, null);
                if(!list.isDir){
                        value = false;
                        DropboxAPI.DropboxLink shareLink = dbAPI.share(list.path);
                        shareAddress = getShareURL(shareLink.url)
                                .replaceFirst("https://www", "https://dl");
                        int index = shareAddress.indexOf("?dl=");
                        shareAddress = shareAddress.substring(0, index);
                        }
                else{
                        value = true;
                        }
                }
        catch(DropboxException e){
                Log.e(TAG, "Error has occurred");
                }
        return null;
}
```

ALGORITMO 4.13: Aquisição do *sharelink* do ficheiro apontado pelo *path*

Assim é criado o *link* como demonstrado na figura 4.6 que será utilizado para a aplicação Chromecast fazer o pedido para obter o conteúdo do ficheiro.

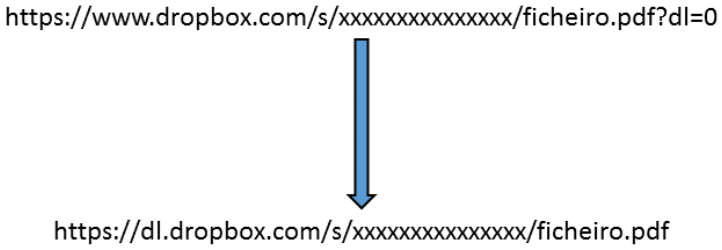


FIGURA 4.6: Modificação do *sharelink*

4.2.3 Interação com a Aplicação Chromecast

Assim como para integrar a API do Dropbox, também é necessário preparar a aplicação para poder implementar os métodos desejados. De forma similar ao método para o Dropbox, é necessário também criar uma aplicação *Cast* no *website* para programadores que utilizam a Google *Cast* API, denominado Google *Cast SDK Developer Console* (`https://cast.google.com/publish`). Aqui é criada a aplicação, dizendo o tipo de aplicação que se pretende criar (neste

caso uma aplicação o tipo *Custom Receiver*), colocando o nome da aplicação *Cast* (para identificar a aplicação para o utilizador). Também é necessário incluir uma URL para a aplicação, ou seja, a URL que o Chromecast abre quando a aplicação inicia. Desta forma é recebida a *App* ID, que é utilizado para invocar a aplicação no Chromecast quando a aplicação Android se conectar ao Chromecast.

De seguida é necessário verificar qual a versão mínima do SDK suportado pela aplicação. A Google *Cast* API suporta a versão mínima SDK 9, e como esta aplicação possui a versão mínima SDK 21, com a versão alvo SDK 23, então a aplicação pode incluir a Google *Cast* API.

```
defaultConfig {
        ...
        minSdkVersion 21
        targetSdkVersion 23
        ...
        }
```

ALGORITMO 4.14: Definição das versões SDK no ficheiro Gradle

Para utilizar a Google *Cast* API, é necessário obter o Google *Play Services*, que incluí a Google *Cast* API. No Android Studio usa-se o SDK *Manager* para obter o Google *Play Services*. Com a API obtida, é so adicionar as bibliotecas necessárias.

```
dependencies {
        ...
        compile 'com.android.support:appcompat-v7:23.2.1'
        compile 'com.google.android.gms:play-services:8.4.0'
        compile 'com.android.support:support-v4:23.2.1'
        }
```

ALGORITMO 4.15: Compilação das bibliotecas necessárias

No ficheiro *AndroidManifest* será adicionado um campo para metadados, contendo a versão do *Google Play Services*.

```
<application
        ...>

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

ALGORITMO 4.16: Metadados da versão do Google *Play Services* no ficheiro *AndroidManifest*

Com os passos realizados, fica então possível para a aplicação implementar os métodos necessários para a aplicação.

Deteção do Chromecast

Para a aplicação detetar o Chromecast, é implementado um *mediaRouter*. A *framework* do *mediaRouter* permite a implementação de métodos de deteção e comunicação com aparelhos de aúdio/projeção visual que possibilita fazer *stream* de conteúdo *wireless*, com o Chromecast a encaixar na lista de aparelhos. Com isto é criado um objeto *mediaRouter*, sendo responsável por encontrar dispositivos Chromecast e mais tarde, quando a conexão tiver sido estabelecida, gerir a comunicação. No entanto, só o objeto *mediaRouter* por si só não chega. Como existem diversos dispositivos capazes de fazer *stream* via *wireless*, a lista é reduzida, implementando um *mediaRouterSelector*, dizendo assim ao objeto *mediaRouter* para só encontrar dispositivos Chromecast e que o dispositivo é suposto executar uma aplicação do seu lado (utiliza-se a *App* ID da aplicação *Cast*).

```
private void initMediaRouter(){
        mediaRouter = MediaRouter.getInstance(getApplicationContext());
        mediaRouteSelector = new MediaRouteSelector.Builder().
                addControlCategory(CastMediaControlIntent.
                        categoryForCast(CHROME_APP_ID)).build();
        }
```

ALGORITMO 4.17: Inicialização das variáveis *mediaRouter* e *mediaRouterSelector*

Para completar o processo é necessário adicionar mais dois objectos: um *mediaRouter.Callback*, que é anexado ao objecto *mediaRouter*, responsável por detetar eventos nas conexões de *media* disponíveis e por iniciar o processo de conexão ao dispositivo Chromecast, e um objecto *mediaRouterProvider*, que é responsável por canalizar o conteúdo quando a conexão entre as aplicações estiver concluída.

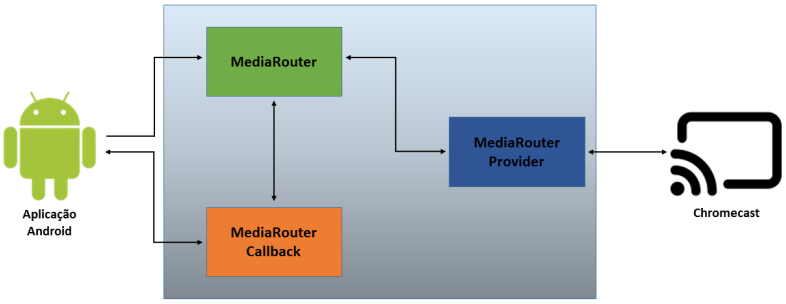


FIGURA 4.7: *Overview* da *framework mediaRouter*

Para a aplicação Android, é criado um botão do tipo *MenuItem*, que é responsável por mostrar os dispositivos Chromecast conectados à mesma rede que o *Smartphone*, e quando selecionado o dispositivo Chromecast, o *mediaRouterProvider* gere o fluxo de conteúdo entre a aplicação Android e neste caso, a aplicação executada no Chromecast. O *mediaRouterSelector* é incluído no *mediaRouterProvider*, notificando ao *mediaRouterProvider* que dispositivos ele pode encontrar.

```
public void setMediaRouterSelectorButton(MenuItem menuItem) {
        MediaRouteActionProvider mediaRouteActionProvider =
                (MediaRouteActionProvider) MenuItemCompat.getActionProvider(menuItem);
        mediaRouteActionProvider.setRouteSelector(mediaRouteSelector);
}
```

ALGORITMO 4.18: Inicialização do botão e anexação do *mediaRouterProvider*

A seguir inicializa-se o *mediaRouter.Callback*, que no caso da aplicação Android é responsável por notificar o *mediaRouter* de quando o utilizador se conectou/desconectou de um dispositivo, e que é reponsável por iniciar o processo de conexão com o dispositivo Chromecast (*onRouteSelected*), ou de desconectar (*onRouteUnselected*).

```
private final MediaRouter.Callback mediaRouterCallback = new MediaRouter.Callback() {
        @Override
        public void onRouteSelected(MediaRouter router,
                                            MediaRouter.RouteInfo route) {...}

        @Override
        public void onRouteUnselected(MediaRouter router,
                                            MediaRouter.RouteInfo route) {...}
    };
```

ALGORITMO 4.19: Inicialização do *mediaRouter.Callback*

Por fim, quando a atividade do menu principal inicia, é adicionado ao *mediaRouter* tanto o *mediaRouter.Callback* como o *mediaRouterSelector* através do método *mediaRouterStart*. A *flag* notifica a *mediaRouter* para procurar os dispositivos pretendidos em intervalos, fazendo *refresh*.

```
public void mediaRouterStart() {
        mediaRouter.addCallback(mediaRouteSelector, mediaRouterCallback,
                MediaRouter.CALLBACK_FLAG_PERFORM_ACTIVE_SCAN);
        }
```

ALGORITMO 4.20: Começo do *mediaRouter* na atividade

Conexão ao Chromecast e Início da Aplicação no Chromecast

Com a deteção de dispositivos Chromecast implementada, é agora necessário finalizar a conexão ao dispositivo, assim como o arranque da aplicação *web* no Chromecast. Como é possível verificar no *mediaRouter.Callback*, existe um método chamado *onRouteSelected* que é invocado quando o utilizador escolhe um dispositivo Chromecast, obtendo informações sobre o dispositivo para poder inicializar a *Cast* API para finalizar o processo.

```
private final MediaRouter.Callback mediaRouterCallback = new MediaRouter.Callback() {
        @Override
        public void onRouteSelected(MediaRouter router, MediaRouter.RouteInfo route) {
                CastDevice device = CastDevice.getFromBundle(route.getExtras());
                setChosenDevice(device);
                }
        ...
        };
```

ALGORITMO 4.21: Aquisição de informação para finalização do processo de conexão

Dentro do método *setChosenDevice*, é verificado se foi obtido a informação do dispositivo, e em caso afirmativo tenta conectar a *Cast* API ao dispositivo. Antes de conectar a *Cast* API é criado um *Cast.castOptions*, que é necessário para configurar a instância da *GoogleApiClient*, que corresponde a uma instância da textitCast API. A variável que corresponde ao *Cast.castOptions* é definida como *castOptions*, e é inicializada com as informações sobre o dispositivo, assim como com um *Cast.Listener*, que é reponsável por escutar por eventos entre as aplicações (como quando a aplicação Android se desconecta do dispositivo).

Também é implementado um *connectionFailedListener*, para o caso de a conexão com o dispositivo falhar. Neste caso é parado o processo da conexão da *Cast* API ao dispositivo e é avisado ao *mediaRouter* para simplesmente iniciar uma conexão para visualizar conteúdo *Media* em vez de executar a aplicação pretendida.

Além do *connectionFailedListener*, também é implementado um *GoogleApiClient.ConnectionCallback*. Este *connectionCallback* é então anexado à instância da *GoogleApiClient* (definida como *clientAPI*), instância responsável por iniciar a aplicação no Chromecast quando *clientAPI* se conecta ao dispositivo.

```
private final GoogleApiClient.ConnectionCallbacks connectionCallback =
        new GoogleApiClient.ConnectionCallbacks() {
                @Override
                public void onConnected(Bundle bundle) {
                        try {
                                Cast.CastApi.launchApplication(clientAPI,
                                        CHROME_APP_ID).
                                        setResultCallback(new ResultCallback<Cast.
                                                ApplicationConnectionResult>(){
                                                ...
                                                });
                                }
                        ...
                }
        };
```

ALGORITMO 4.22: Início da aplicação no Chromecast

Com o *connectionFailedListener*,*connectionCallback* e *castOptions* implementados, é então iniciado a conexão entre ambas as aplicações.

```
private void connectClientAPI() {
        Cast.CastOptions castOptions = Cast.CastOptions
                .builder(chosenDevice, castClientListener).build();

                clientAPI = new GoogleApiClient.Builder(this)
                        .addApi(Cast.API, castOptions)
                        .addConnectionCallbacks(connectionCallback)
                        .addOnConnectionFailedListener(connectionFailedListener)
                        .build();
                clientAPI.connect();
                }
```

ALGORITMO 4.23: Início da conexão entre ambas as aplicações

Envio de Mensagens para a Aplicação Chromecast

Quando é necessário enviar uma mensagem para a aplicação Chromecast, utiliza-se o método *sendMessage*. Dentro do método, é verificado se existe conexão com o aparelho Chromecast, e em caso positivo envia a mensagem pelo canal de comunicação (*CHANNEL*). É colocado uma *callback* no envio da mensagem, para saber se a mensagem foi enviada com sucesso.

```
public void sendMessage(String message) {
        Log.i(TAG, message);
        if (clientAPI != null) {
                try {
                        Cast.CastApi.sendMessage(clientAPI, CHANNEL, message)
                                .setResultCallback(new ResultCallback<Status>() {
                                        @Override
                                        public void onResult(@NonNull Status result) {
                                                if (result.isSuccess()) {
                                                        Log.i(TAG, "Message was sent");
                                                        }
                                                else{
                                                        Log.i(TAG, "Message not sent");
                                                        }
                                                }
                        });
                }
                catch (Exception e) {
                        Log.e(TAG, "Error while sending message", e);
                        }
                }
        }
```

ALGORITMO 4.24: Envio de uma mensagem para a aplicação Chromecast

Modo Laser

O modo laser é iniciado através do *Fragment Laser*. Este modo é bastante simples, visto que a sua função principal é ler os valores do acelerómetro do *Smartphone*. Este permite enviar quatro tipos de mensagens para a aplicação Chromecast: a mensagem para avançar para a página seguinte, a mensagem para voltar à página anterior, a mensagem com as percentagens do eixo X e Y (explicado mais adiante e, por fim, a mensagem para aplicação Chromecast também sair do modo laser quando a aplicação Android sair.

Para ler os valores do acelerómetro implementou-se um *SensorManager* e um objeto do tipo *Sensor*. O *SensorManager* pede ao serviço para ler um sensor do sistema, atribuindo ao objeto *Sensor* o tipo *TYPE_ACCELEROMETER*, indicando que o sensor para ler valores é o acelerómetro. Por fim, regista um *Listener* ao sensor para ler os valores.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        ...
        sensorManager=(SensorManager)getActivity()
                .getSystemService(Context.SENSOR_SERVICE);
        acceleration=sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        sensorManager.registerListener(this, acceleration,
                SensorManager.SENSOR_DELAY_NORMAL);
        ...
        }
```

ALGORITMO 4.25: Inicialização do método de leitura de dados do acelerómetro

Sempre que os valores do acelerómetro forem modificados, é chamado o método *onSensorChanged*. Este método lê o evento do sensor, de onde são obtidos os valores do eixo X e eixo Y. Os valores de ambos os eixos modificam de acordo com a rotação do *Smartphone* medida pelo acelerómetro incorporado nos eixos respetivos como representado na 4.8, geralmente variando entre -10 e 10 (intervalo modifica ligeiramente dependendo do *Smartphone*), com precisão de até 5 casas decimais.

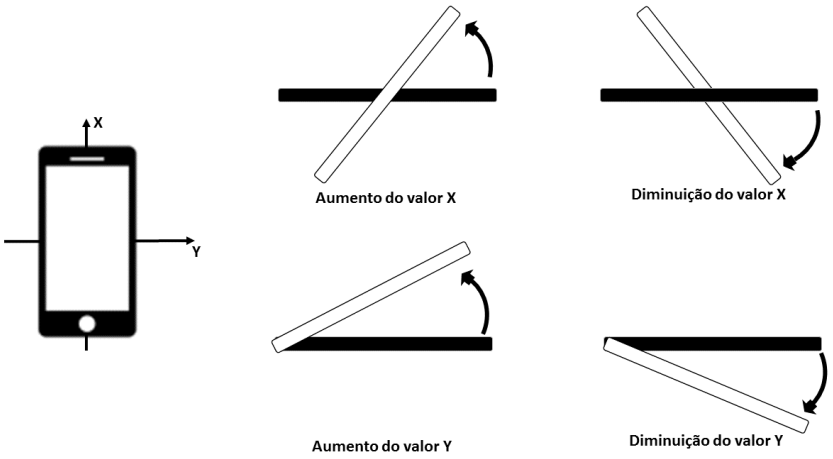


FIGURA 4.8: Eixos de rotação do acelerómetro

Em programação visual (*canvas*, *layouts*, imagens , *etc*), geralmente são utilizados píxeis , com o ponto inicial (ponto no canto superior esquerdo) com o valor de 0 em tanto o eixo X como no Y. À medida que se vai "descendo" pelo eixo Y, maior é o valor Y (eixo invertido), com o eixo X a manter a sua lógica normal (valor aumenta da esquerda para a direita), como se poder visualizar na figura 4.9.

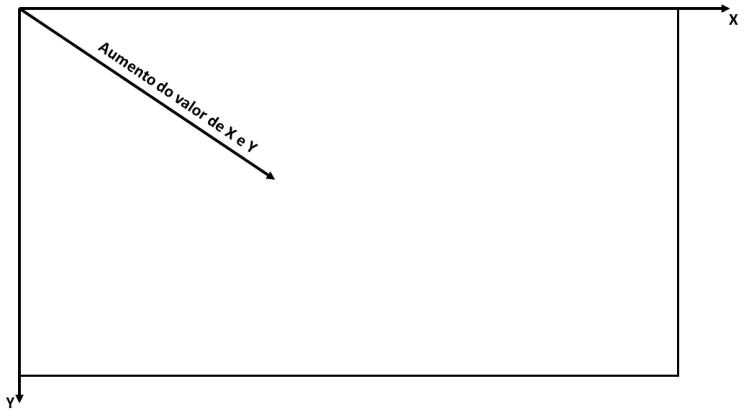


FIGURA 4.9: Eixos de uma imagem

Olhando para a rotação e como os eixos funcionam , são invertidos os valores lidos do acelerómetro. Assim quando se roda o *Smartphone* para a direita no eixo X, o valor de X é maior, e quando se roda o *Smartphone* no eixo Y

para cima, menor é o valor de Y. A seguir converte-se o intervalo para percentagem. Assim um valor negativo de -10 é convertido para 0%, com o valor 10 a ser 100%. Este valores de percentagem são utilizados na aplicação Chromecast para saber a posição onde se deve posicionar o apontador laser, sendo que se os valores percentagem X e Y forem 100% para ambos, o apontador laser deve mover-se para o canto inferior direito, assim como se os valores percentagem X e Y forem 0%, o apontador laser deve mover-se para o canto superior esquerdo.

```
@Override
public void onSensorChanged(SensorEvent event){
        int x = (int)(event.values[0]*(-10));
        int y = (int)(event.values[1]*(-10));

        int xPercentage = 100 * (x-(-100)) / (100-(-100));
        int yPercentage = 100 * (y-(-100)) / (100-(-100));
        ...
}
```

ALGORITMO 4.26: Inversão dos valores X e Y do acelerómetro com conversão dos valores para percentagens

Modo Desenho

O modo desenho é implementado por dois *Fragments*: o *Fragment Pad* e o *Fragment Draw*. O primeiro é o *Fragment* invocado quando se pretende ir para o modo desenho, com o *Fragment Draw* a ficar dentro do *Fragment Pad*. O *Fragment Pad* desempenha duas funções: modificar a cor para desenhar, bem como de sair do modo desenho (botão de saída). O *Fragment Draw* é responsável por implementar um detetor de gestos, enviando comandos diferentes para a aplicação Chromecast dependendo do gesto. A figura 4.10 apresenta um *sketch* de como é visualizado o modo desenho.

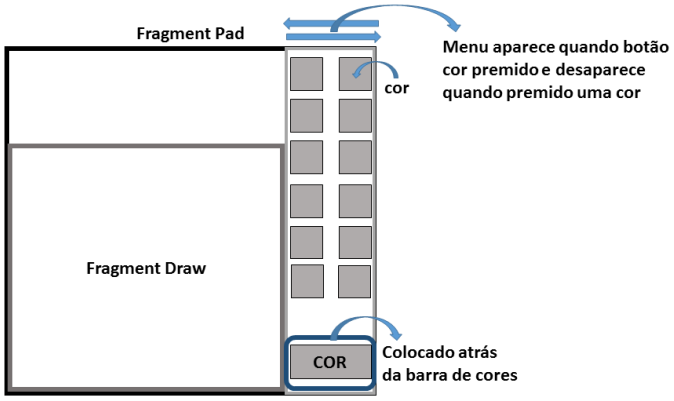


FIGURA 4.10: *Overview* do modo desenho

Em relação ao *Fragment Pad*, é criado um *layout* "filho" que contém objetos do tipo *ImageButton* para as cores, implementando um menu de escolha para as cores, que fica dentro do *layout* do *Fragment Pad*. Este *layout* "filho" contém os botões para as cores (*ImageButtons*), sendo botões *custom* para poderem mostrar a que cores se referem. Uma variável denominada *chosenColor* é utilizada para saber qual foi o botão de cor pressionado, colocando uma *border* especial à volta do botão escolhido, indicando a cor que se está a utilizar para desenhar, sendo colocada por defeito no primeiro botão do menu de cores (botão com a cor *black*).

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
        ...
        relativeLayout = (RelativeLayout)view.findViewById(R.id.colorLayout);
        chosenColor = (ImageButton)relativeLayout.getChildAt(0);
        chosenColor.setImageDrawable(getResources()
                .getDrawable(R.drawable.color_chosen, null));
        ...
        }
```

ALGORITMO 4.27: Inicialização dos botões das cores

Quando o menu das cores está visível, o utilizador pode escolher uma nova cor. Quando é escolhida a nova cor, antes de o menu desaparecer, é verificado se foi escolhida uma nova cor. Em caso positivo é identificado qual dos botões foi premido, sendo atribuído a esse botão à variável *chosenColor* (ficando esse botão com a *border* especial.

Como foi escolhida uma nova cor, obtém-se o nome da cor ("*red*", "*green*", "*blue*", *etc*), sendo convertido para uma *String* para depois enviar para a aplicação Chromecast.

```
public void onClick(View view){
        ...
        if(view != chosenColor){
                ImageButton imageView = (ImageButton)view;
                imageView.setImageDrawable(getResources()
                        .getDrawable(R.drawable.color_chosen, null));
                chosenColor.setImageDrawable(getResources()
                        .getDrawable(R.drawable.color_button, null));
                chosenColor = (ImageButton)view;
                }
        String color = view.getResources().getResourceEntryName(view.getId());
        ...
        }
```

ALGORITMO 4.28: Transição de cor utilizada

Por fim, para invocar o *Fragment Draw*, é utilizado outro *layout* para inserir o novo *Fragment*.

```
getChildFragmentManager()
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                            Bundle savedInstanceState) {
        ...
        Fragment fragmentDraw = new FragmentDraw();
        getChildFragmentManager()
                .beginTransaction()
                .add(R.id.fragment_holder, fragmentDraw, "TAG")
                .commit();
        ...
        }
```

ALGORITMO 4.29: Inicialização do *Fragment Draw*

Em relação ao *Fragment Draw*, como já foi referido o *Fragment* implementa um detetor de gestos.

```
final GestureDetector gesture = new GestureDetector(getActivity(),
        new GestureDetector.SimpleOnGestureListener() {
                @Override
                public boolean onDown(MotionEvent e) {
                        return true;
                        }
                ...
        });
```

ALGORITMO 4.30: Inicialização do *Listener* de gestos

Este *Gesture Listener* é responsável por saber que gesto foi feito. Para o sistema pretendido, são reconhecidos quatro gestos no total: *double tap*, *scroll*, *long press* e *fling*. Quando é reconhecido o gesto *double tap*, é enviado uma mensagem para ativar uma *flag* na aplicação Chromecast para que sempre que o cursor se mova, é desenhado um traço para emular desenho. Para o gesto *long press*, é enviado uma mensagem para desativar a mesma *flag* na aplicação Chromecast. O gesto *fling* serve para mudar a página na aplicação Chromecast. Quando o gesto é detetado, é visto para que lado foi feito (esquerda ou direita), para saber se é para ir para a página seguinte ou a anterior (enviando a mensagem correspondente). Por fim o gesto *scroll* mede a distância (para ambos os eixos) entre a posição onde foi detetado o toque no *Fragment* até o utilizador levantar o dedo. Isto levanta um problema, visto que o utilizador pode movimentar o seu dedo numa curva para chegar a uma posição, mas a distância retornada para ambos os eixos forma uma reta (caminho mais curto). Para resolver o problema foi implementado uma *Thread Runnable*, que sendo controlada por um *Handler*, funciona como um *timer*, enviando assim as distâncias medidas em intervalos pequenos. Assim quando o utilizador forma uma curva, são enviadas as distâncias múltiplas vezes para aplicação Chromecast, podendo assim emular no ecrã o mesmo movimento que o utilizador.

```
private Handler handler = new Handler();
private Runnable runnable = new Runnable(){
        public void run() {
                makeMessage();

                distX = 0;
                distY = 0;
                handler.postDelayed(runnable, interval);
                }
        };
```

ALGORITMO 4.31: Implementação do *Runnable* para envio das distâncias

Por fim com a detação de gestos implementada, é colocada na *view* do *Fragment* um detetor de toque, que retorna o gesto para o *Gesture Listener* poder verificar qual dos gestos foi efetuado.

```
view.setOnTouchListener(new View.OnTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
                return gesture.onTouchEvent(event);
                }
        });
```

ALGORITMO 4.32: Implementação da deteção de toque na *view* do *Fragment*

4.3 Aplicação Chromecast

A aplicação Chromecast, como já foi referido anteriormente, só é inicializada quando a aplicação Android se conecta ao Chromecast. A aplicação executada no Chromecast é uma aplicação *web*, ou seja, possuí o seu próprio domínio, o que implica que, de uma forma simples, o Chromecast simplesmente abre a página *web* onde a aplicação se encontra. Isso implica que uma aplicação para o Chromecast seja primeiro registada, para ser possível abrir a página *web* onde a aplicação se encontra (explicação com maior detalhe na sub-secção de interação com a aplicação Android).

A aplicação é composta por um ficheiro HTML, que serve de *index* para a aplicação, onde está definido a estrutura da página *web*, assim como o seu aspecto, com o processamento todo a ser colocado em ficheiros javascript.

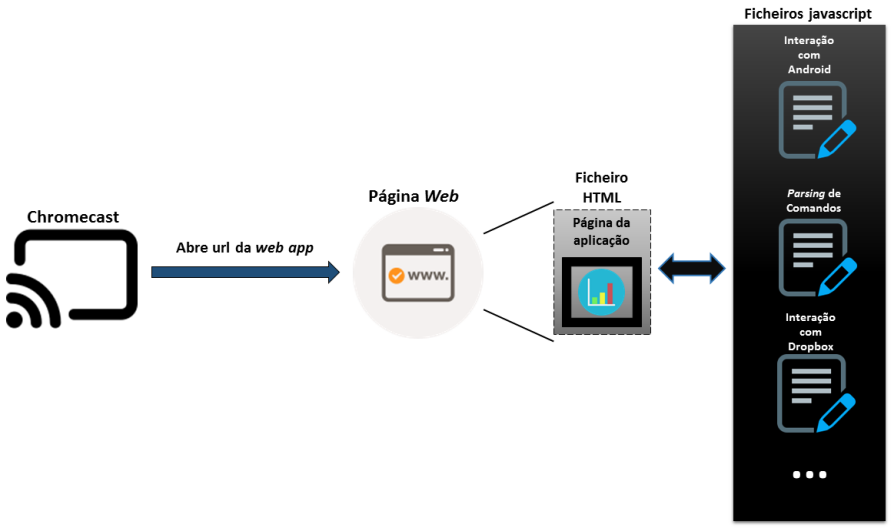


FIGURA 4.11: Estrutura da aplicação Chromecast

No que diz respeito ao aspecto da página da aplicação, a aplicação apresenta um "*look*" simples, com a visualização do ficheiro na página (como se pode ver na figura 4.11). Para ser possível visualizar as páginas do ficheiro utilizou-se um *canvas* onde, depois de processar o ficheiro, as páginas serão então inseridas no *canvas* (à medida que o utilizador navega entre elas). No entanto, como a aplicação suporta a funcionalidade de desenho, assim como um apontador laser, um único *canvas* é pouco para três objetos serem lá inseridos. Por isso decidiu-se colocar três *canvas*, onde é inserido em cada um o objeto designado. Os três *canvas* são posicionados em forma de pilha, funcionando como se fossem "*layers*", com o *canvas* do ficheiro a ser o mais distante e o dos apontadores/cursores a ser o mais próximo. Isto pode ser implementado ao colocar os três na mesma posição absoluta, mas com o indíce z (que controla a profundidade do *canvas*) a possuir valores diferentes para cada um dos *canvas*. Desta forma fica a página do ficheiro a ser vista no primeiro *canvas*, e quando o utilizador começa a desenhar, os traços do desenho são desenhados no segundo *canvas*, que fica à frente do *canvas* da página do ficheiro, dando uma ideia de que o utilizador está a desenhar sobre a página, quando na realidade está a desenhar numa *layer* diferente. Por fim os apontadores (laser e de desenho) estão na *layer* acima, para seja possível visualizar os cursores mesmo depois de desenhar no *canvas* de desenho. A visualização das camadas por parte do utilizador pode ser visto na figura 4.12.

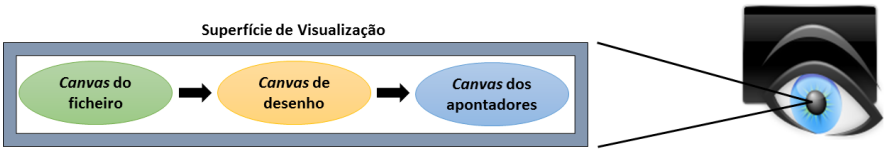


FIGURA 4.12: Visualização dos *canvas* do ponto de vista do utilizador

```
<div style="position: relative; width:800px; margin:0 auto;">
        <canvas id="canvasPDF"  style="position: absolute; left: 0;
                                       top: 0; z-index: 0; border:1px  solid white">
                                       </canvas>
        <canvas id="canvasDraw" style="position: absolute; left: 0;
                                       top: 0; z-index: 1; border:1px  solid white">
                                       </canvas>
        <canvas id="canvasPointer" style="position: absolute; left: 0;
                                       top: 0; z-index: 2; border:1px  solid white">
                                       </canvas>
</div>
```

ALGORITMO 4.33: Implementação dos *canvas* em HTML

4.3.1 Processos Internos

A aplicação Chromecast possui dois processos internos: o processamento do conteúdo do ficheiro e visualização do documento. Quando o conteúdo do ficheiro é obtido (explicado mais abaixo), é necessário que o conteúdo seja processado, antes de ser possível interagir com ele. Depois de ser processado, é então possível mostrar o conteúdo do ficheiro no ecrã, como será explicado mais adiante.

Processamento do Conteúdo do Ficheiro

Para processar o conteúdo de um ficheiro, é necessária a uma API específica. Esta API é composta por dois ficheiros: pdf.js e pdf.worker.js. Para iniciar então o processo, é invocado um objeto do pdf.js chamado PDFJS, que é responsável pelo processamento do conteúdo do ficheiro em *background* quando inicializado com o pdf.worker.js.

Quando o conteúdo do ficheiro é obtido, é enviado para outra função para processamento. Esta função é denominada *callGetDocument*. Nesta função o conteúdo é enviado para o *PDFJS*, onde é processado e retornado como um objeto PDF, de onde é retirado o número de páginas. Conluído este processo, são retiradas do objeto as páginas para guardar num *array*.

```
function callGetDocument (response) {
    PDFJS.getDocument(response).then(function getPDF(pdfObject) {
        pdfFile = pdfObject;
        numPages = pdfFile.numPages;
        getPages();
    });
}
```

ALGORITMO 4.34: Processamento do conteúdo do ficheiro

Este *array*, denominado *pdfArray*, guarda as páginas do ficheiro, simplificando assim o processo de navegação entre páginas. Com o número de páginas obtido, um simples ciclo *for* serve para o propósito, onde se obtém o conteúdo específico a uma página e de seguida faz-se *push* do conteúdo para o *array*.

```
function getPages(){
    var i;
    var page;

    for(i=0;i<numPages;i++){
        page = i+1;
        pdfFile.getPage(page).then(function getCertainPage(page) {
            pdfArray.push(page);
        });
    }
}
```

ALGORITMO 4.35: Função para obter as páginas do ficheiro PDF

Visualização do Documento

Depois do processamento do conteúdo do ficheiro, é então possível passar para a etapa de visualização do documento. Quando a aplicação recebe o comando para iniciar a apresentação (designado por "*Start*"), é iniciado o processo de visualização.

```
function startPDF(){
    changePage(0);
    pageFlag = 1;
}
```

ALGORITMO 4.36: Função que inicia a visualização do ficheiro

Quando é iniciado a apresentação, é inserida a primeira página do ficheiro no *canvasPDF*, que se encontra armazenada na posição 0 no *pdfArray*, iniciando o processo onde o conteúdo da página é convertido para ser colocado no *canvasPDF*

```
function changePage(page){
        var data = pdfArray[page];

        var scale = 1.0;
        var viewport = data.getViewport(scale);

        var canvas = document.getElementById('canvasPDF');
        var context = canvas.getContext('2d');

        canvas.height = viewport.height;
        canvas.width = viewport.width;

        var contextObject = {
                canvasContext: context,
                viewport: viewport
                };
        data.render(contextObject);
        ...
        }
```

ALGORITMO 4.37: Processo para colocar o conteúdo de uma página no *canvasPDF*

Dentro da função *changePage* é obtido o conteúdo da página desejada, assim como é obtido o tamanho (largura e comprimento) da página, utilizando o método *getViewport*, que retorna o tamanho da página em pixéis e a sua rotação inicial. De seguida é criado uma referência ao *canvas* desejado, assim como o seu contexto. Com a referência modificam-se as medidas do *canvas* para corresponder às da página do ficheiro. De seguida é criado um objeto *context* através do contexto do *canvas* e do tamanho da página em pixéis (*viewport*), passando a seguir o objeto para o método *render*, que preenche o *canvas* com o conteúdo da página.

Por fim, quando se pretende mudar de página, existe um variável denominada *currentPage*, que corresponde à página do ficheiro em relação ao *array* (que começa em 0). Sempre que o utilizador quer mudar de página é verificado se essa variável se encontra dentro dos limites de páginas (entre o 0 e *numPages*). Em caso positivo, a variável é incrementada/decrementada, invocando a função *changePage* para mudar para a nova página.

4.3.2 Interação com a Aplicação Android

Como já foi referido anteriormente, quando a aplicação Android se conecta ao Chromecast, este é reencaminho para a página *web*, onde se encontra a *web app* que irá executar. Uma vez lá, é necessário executar alguns passos para ser possível receber mensagens da aplicação Android.

Em primeiro lugar, é necessário adicionar um ficheiro javascript próprio da Google, que é responsável pelas tarefas em *background*.

```
<script src="https://www.gstatic.com/cast/sdk/libs/receiver/2.0.0/cast_receiver.js">
</script>
```

ALGORITMO 4.38: Ficheiro Javascript da Google

Em segundo, é necessário adicionar o canal utilizado para a comunicação entre as duas aplicações.

```
//Insert the channel below
var channel = "urn:x-cast:xxxxxxxxxxxxxxx";
```

ALGORITMO 4.39: Variável contendo o canal para a comunicação entre aplicações

De seguida cria-se uma função onde se implementa a comunicação. Utilizando a *Cast* API, é criado um *castReceiverManager*, que é conectado à pagina da aplicação. Depois são criadas duas funções para quando a aplicação Android conecta e disconecta (o *onSenderConnected* e *onSenderDisconnected*), e é implementado um *Bus* com o auxílio do canal de comunicação, que é utilizado para recepção de mensagens enviadas da aplicação Android, e criada a função que é sempre invocada quando o evento de recepção de mensagem for ativado (*onMessage*). Com o essencial implementado, é dito ao *castReceiverManager* para iniciar.

```
function onLoadCast(){
                window.castReceiverManager = cast.receiver
                        .CastReceiverManager.getInstance();
                window.castReceiverManager.onSenderConnected = onSenderConnected;
                window.castReceiverManager.onSenderDisconnected = onSenderDisconnected;

                window.customMessageBus = window.castReceiverManager
                        .getCastMessageBus(channel);
                window.customMessageBus.onMessage = onMessage;

                window.castReceiverManager.start();
        }
```

ALGORITMO 4.40: Implementação da comunicação entre aplicações

Esta função é adicionada a um *listener* na janela da aplicação, utilizando a função para iniciar o processo conexão entre as aplicações.

```
window.addEventListener("load", onLoadCast);
```

ALGORITMO 4.41: Adição do *listener* à janela

Aquisição de Comandos

A aquisição de comandos passa pela função registada para atender os eventos de receção de mensagem, nomeadamente a função *onMessage*. Esta função é ativada sempre que seja detetada uma mensagem enviada da aplicação Android, onde é lido o "comando" recebido (guardado no evento como dados) e é passado como parâmetro para a função responsável por saber que comando foi enviado.

```
function onMessage(event){
                var message = event.data;
                whatCommand(message);
        }
```

ALGORITMO 4.42: Implementação da função *onMessage*

Parsing de Comandos

Para saber que comando foi recebido pela aplicação Chromecast, é feito um processo de separação para identificar o tipo de comando. Na função *onMessage* é invocada a função *whatCommand*, para onde é enviada a mensagem recebida pela aplicação Chromecast. Dentro desta função é determinado o tipo de comandos que podem ser executados e, no caso do modo laser e modo de desenho, ativar/desativar as funcionalidades. Sempre que o utilizador entra numa das três opções na aplicação Android (escolha de ficheiro, modo laser ou modo desenho), é enviado um comando para modificar o modo na aplicação Chromecast. A variável *whatMode* é modificada para um valor que depende do modo que se escolheu. Quando se sai de um modo, é recebido o comando "*exit*" , que atribui um valor específico para impedir que a aplicação entre nas funções dos outros modos e desative o modo laser ou o modo de desenho, caso esteja um ativo.

```
function whatCommand(message){
        if(message == "file"){
                whatMode = 1;
        }
        else if(message == "laser"){
                whatMode = 2;
                toggleLaser();
        }
        else if(message == "draw"){
                whatMode = 3;
                toggleDrawMouse();
        }
        else if(message == "exit"){
                if(whatMode == 2){
                        toggleLaser();
                }
                else if(whatMode == 3){
                        toggleDrawMouse();
                }
                whatMode = 0;
        }
        else{
                whatParse(message, whatMode);
        }
}
```

ALGORITMO 4.43: Implementação da função *whatCommand*

Caso nenhum dos comandos dos modos tenha sido enviado (ou o comando comando "*exit*"), então é porque é um dos comandos para execução da tarefas na aplicação Chromecast. Nesse caso é invocada a função *whatParse*, que

recebe a mensagem e o valor da variável *whatMode*, para saber que função é necessário invocar. Dentro desta função é verificado o valor da variável *whatMode*, invocando a função para determinar que comando foi recebido para o modo particular.

```
function whatParse(message, value){
        switch(value){
                case 1:
                        fileParse(message);
                        break;
                case 2:
                        laserParse(message);
                        break;
                case 3:
                        drawParse(message);
                        break;
                 default:
                        break;
        }
}
```

ALGORITMO 4.44: Implementação da função *whatParse*

Se o valor da variável *whatMode* for 0, então nenhuma das funções é executada, visto que o utilizador não está a usufruir de nenhum dos modos, impedindo assim verificações desnecessárias. Caso algum dos modos esteja ativo, entra na respetiva função para saber que comando foi enviado.

Modo Laser

O modo laser utiliza duas *flags* para poder ser inicializado. A primeira é a *flag pageFlag*, que só é ativada quando o utilizador inicia a apresentação, causando uma reestruturação do tamanho do *CanvasPDF*. A segunda *flag* (*laserFlag*) só é ativada depois da primeira ser ativada, e serve para fazer *toggle* ao modo laser depois da apresentação começar. Quando as duas *flags* são ativadas, é obtido o tamanho do *canvasPDF*, e enviado para a função *reloadLaser*. Dentro desta função é reestruturado o *canvasPointer* para corresponder ao tamanho do *canvasPDF*, assim como colocar o apontador/cursor laser no *canvas*.

```
function reloadLaser(width, height){
        if(laserFlag == 1) {
                var canvasLaser = document.getElementById('canvasPointer');
                var contextLaser = canvasLaser.getContext('2d');

                canvasLaser.height = height;
                canvasLaser.width = width;
                ...
                laserImage.onload = function () {
                        var x = width / 2;
                        var y = height / 2;

                        pointerWidth = this.width/2;
                        pointerHeight = this.height/2;

                        var xLaser = x - pointerWidth;
                        var yLaser = y - pointerHeight;

                        contextLaser.drawImage(laserImage, xLaser, yLaser);
                        }
                laserImage.src = 'ledOrange.png';
                }
        }
```

ALGORITMO 4.45: Inicialização do cursor reestruturação do *canvasPointer*

O modo laser recebe três tipos de comandos fora o comando habitual para sair ("*exit*"), que são: o comando para avançar de página ("*forwards*"), o comando para retroceder ("*backwards*") e o comando com as percentagens do eixo X e Y.

```
function laserParse(message){
        if(message == "forwards"){
                openNextPage();
        }
        else if(message == "backwards"){
                openPreviousPage();
        }
        else{
                var index = message.indexOf("|");
                if(index > 0){
                        moveLaser(message, index);
                }
        }
}
```

ALGORITMO 4.46: Implementação da função *laserParse*

Os dois primeiros simplesmente invocam as funções para avançar e retrocer de página, enquanto que o terceiro é responsável por mover o cursor laser pelo *canvas* de acordo com as percentagens. A função *moveLaser* obtém as duas percentagens da mensagem enviada da aplicação Android, separando-as em duas

Strings (*xString* e *yString*), e depois convertendo-as para tipo *float*, com um valor decimal entre 0.0 e 1.0.

De seguida multiplica o comprimento (*width*) e a largura (*height*) por esse valor, indicando assim onde é suposto o cursor estar (tendo em conta o tamanho do próprio cursor), limpando o *canvasPointer* e desenhando de novo o cursor na sua nova posição.

```
function moveLaser(message, index){
        ...
        var xPercentage = parseFloat(xString, 10);
        var yPercentage = parseFloat(yString, 10);

        xPercentage = xPercentage / 100;
        yPercentage = yPercentage / 100;

        xLaser = Math.floor( xLaser ) - pointerWidth;
        yLaser = Math.floor( yLaser ) - pointerHeight;

        contextLaser.clearRect(0, 0, laserWidth, laserHeight);
        contextLaser.drawImage(laserImage, xLaser, yLaser);
        }
```

ALGORITMO 4.47: Movimentação do cursor laser no *canvasPointer*

Modo Desenho

O modo desenho funciona de forma mais complexa que o modo laser, principalmente pelo facto de funcionar com dois *canvas* ao mesmo tempo (*canvasDraw* e *canvasPointer*). O modo desenho utiliza quatro *flags* na totalidade:

1. A mesma *flag* utilizada para o modo laser, nomeadamente a *pageFlag*, para avizar que o *CanvasPDF* foi reestruturado;
2. A equivalente à *laserFlag*, chamada de *flagDrawMouse*, que fará o *toggle* do modo desenho;
3. A *flagDraw*, que é ativada para desenhar no *canvasDraw*;
4. A *flagDrawn*, que é ativada quando se desenha e serve para guardar o *canvasDraw* num *array*, quando é desenhado algo no *canvas*.

Assim como no modo laser, quando é ativado o modo desenho é obtido o tamanho do *canvasPDF*, que é enviado para a função *reloadDraw*, executando o mesmo que a função *reloadLaser*, mas para o modo desenho.

O modo desenho recebe seis comandos fora o comando para sair do modo desenho. O comandos *swipeRight* e *swipeLeft* servem para avançar e retroceder de página durante a apresentação, respetivamente. O comando *pressLong* irá desativar a *flagDraw*, para impedir que o desenho no *canvasDraw*. O comando *doubleTap* ativa a *flagDraw*, que permite desenhar traços no *canvasDraw* e a *flagDrawn*, que avisa a aplicação Chromecast para guardar o corrente *canvasDraw* antes que o utilizador mude de página. O comando com as distâncias serve para movimentar o cursor do modo desenho. Por fim, o comando da cor modifica a cor dos traços desenhados.

```
function drawParse(message){
        if(message == "swipeLeft"){
                openPreviousPage();
        }
        else if(message == "swipeRight"){
                openNextPage();
        }
        else if(message == "pressLong"){
                flagDraw = false;
        }
        else if(message == "doubleTap"){
                flagDraw = true;
                flagDrawn = true;
        }
        else if(message.indexOf("move") != -1){
                movePoint(message);
        }
        else if(message.indexOf("color") != -1){
                changeColor(message);
        }
}
```

ALGORITMO 4.48: Implementação da função *drawParse*

A função *changeColor* obtém da mensagem a cor respetiva e guarda-a numa variável que é utilizada quando for para desenhar os traços.

```
function changeColor(message){
  var index = message.indexOf("-");
  index = index + 1;
  color = message.substring(index);
}
```

ALGORITMO 4.49: Implementação da função *changeColor*

A função *movePoint* obtém as distâncias X e Y de forma similar às percentagens no modo laser, guardando-as em duas *Strings* (*xString* e *yString*), que são convertidas para valores inteiros e adicionados à posição corrente do cursor

(*xDraw* e *yDraw*), guardando a posição anterior para caso seja preciso desenhar no *canvasDraw* (*xDrawPrevious* e *yDrawPrevious*).

```
function movePoint(message){
        ...
        var xDistance = parseInt(xString, 10);
        var yDistance = parseInt(yString, 10);

        xDrawPrevious = xDraw;
        yDrawPrevious = yDraw;

        xDraw = xDraw + xDistance;
        yDraw = yDraw + yDistance;

        contextDraw.clearRect(0, 0, drawWidth, drawHeight);
        contextDraw.drawImage(drawImage, xDraw, yDraw);
        ...
}
```

ALGORITMO 4.50: Movimento do cursor no *canvasPointer* no modo desenho

Antes de sair da função, é verificado se *flagDraw* está ativa, e em caso positivo, invoca a função *draw*. Esta função desenha o traço correspondente ao movimento do utilizador, desenhando um traço entre a posição de início e a nova posição, com a cor escolhida pelo utilizador.

```
function draw(){
    var canvasDrawn = document.getElementById('canvasDraw');
    var contextDrawn = canvasDrawn.getContext("2d");

    contextDrawn.beginPath();
    contextDrawn.moveTo(xDrawPrevious, yDrawPrevious);
    contextDrawn.lineTo(xDraw, yDraw);
    contextDrawn.strokeStyle = color;
    contextDrawn.lineWidth = lineWidth;
    contextDrawn.stroke();
    contextDrawn.closePath();
}
```

ALGORITMO 4.51: Implementação do desenho no *canvasDraw*

Por fim, quando o utilizador desenhar algo no *canvasDraw*, e pretende mudar de página, o *canvasDraw* é limpo. No entanto, como a *flagDrawn* foi ativa, é guardado o corrente *canvasDraw* num *array* (*saveCanvas*). Para isso utiliza a função *saveChanges*, que é invocada quando o utilizador enviou um comando para mudar de página.

```
function saveChanges(currentPageNumber){
    if(flagDrawn == true){
        flagDrawn = false;

        var canvasDraw = document.getElementById('canvasDraw');
        var drawnSlide = canvasDraw.toDataURL();

        saveCanvas.splice(currentPageNumber, 1);
        saveCanvas.splice(currentPageNumber, 0, drawnSlide);
    }
}
```

ALGORITMO 4.52: Implementação da função *saveChanges*

Por fim é utilizado uma variável chamada *savedCanvas*, que é incrementada sempre que o utilizador muda para uma página nova que ainda não tenha sido carregada para o *canvas*. Esta variável serve para carregar no *canvasDraw* as versões que foram guardadas sempre que o utilizador retrocede para uma página anterior. Assim se o utilizador desenhou numa página e regressa à mesma, é carregado para o *canvasDraw* a versão desenhada e não uma versão limpa.

```
function loadDrawnCanvas(page, contextDraw){
        if(savedCanvas > page){
        var imageDraw = new Image();
        imageDraw.onload = function(){
            contextDraw.drawImage(imageDraw,0,0);
        };
        imageDraw.src = saveCanvas[page];
    }
}
```

ALGORITMO 4.53: Aquisição das versões anteriores do *canvas-Draw*

4.3.3 Interação com o Dropbox

A interação com o Dropbox, ao contrário da aplicação Android, utiliza uma API própria, nomeadamente a API *XMLHttpRequest*, que permite transferir dados entre o cliente e o servidor. Neste caso, é criado um pedido, que é utilizado para requisitar o ficheiro ao servidor do Dropbox.

Depois de criada a variável *XMLHttpRequest*, o pedido é inicializado. Para isso utiliza-se o método *open()*, inserir o parâmetro "*GET*" para avisar que é pretendido obter a informação do ficheiro, inserir o *link* onde o ficheiro se encontra, e utilizando a opção para tornar o pedido asíncrono (opção "*true*").

Para o ficheiro PDF, no entanto, o conteúdo do ficheiro obtido está em formato binário, sendo assim necessário colocar a resposta recebida (onde é guardado o conteúdo do ficheiro) como um "*arraybuffer*", que suporta dados em formato binário. Por fim é enviado o pedido para obter o conteúdo do ficheiro.

```
function getURLData (url) {
        var request = new XMLHttpRequest();

        request.open('GET', url, true);
        request.responseType = 'arraybuffer';
        ...
        request.send();
        }
```

ALGORITMO 4.54: Pedido do conteúdo do ficheiro ao Dropbox

Aquisição do Ficheiro

Com o pedido enviado, é necessário esperar pela resposta. Para isso é utilizado o método *onreadystatechange* para monitorizar o pedido. Para o caso de receber o conteúdo do ficheiro, é necessário averiguar quando a propriedade *readyState* indica que a operação do pedido acabou (representada pelo valor 4). Quando é averiguado que a operação acabou, é preciso de ver o estado ("*status*") do pedido. Caso o estado seja 200, então é porque o pedido foi executado como pretendido, ou seja, foi capaz de obter o conteúdo do ficheiro. Neste caso a resposta do pedido é enviado para outra função para processar o conteúdo.

```
function getURLData (url) {
        ...
        request.onreadystatechange = function(){
                if (request.readyState === 4){
                        if (request.status === 200){
                                callGetDocument(request.response);
                                }
        ...
        }
```

ALGORITMO 4.55: Envio da resposta do pedido para processamento

Capítulo 5

Testes e Resultados

No capítulo anterior foi descrito como foram implementadas ambas as aplicações. Primeiro foi descrita a implementação da aplicação Android , explicando os processos por detrás das funcionalidades, nomeadamente as funcionalidades internas, interação com o Dropbox e com a aplicação Chromecast (do lado da Aplicação Android). De seguida foram explicados os processos por detrás da aplicação Chromecast, explicando como se obtêm os comandos da aplicação Android, como estes são processados, como se obtém o ficheiro desejado e como se emulam os comandos do utilizador para poder fazer uma apresentação.

Neste capítulo são apresentados os resultados da implementação de ambas as aplicações, focando-se nas funcionalidades internas, assim como na interação entre as duas e o repositório. Por fim também é mostrado um teste sobre o consumo de bateria por parte da aplicação Android (parte dos requisitos).

5.1 *Login* no Menu Principal

Como já foi referido anteriormente, quando se inicia a aplicação vai-se para um menu inicial representado na figura 5.1, onde é necessário cumprir dois requisitos para poder entrar no menu seguinte (onde se encontram as funcionalidades da aplicação Android): estar conectado a uma rede e estar conectado a uma conta Dropbox.

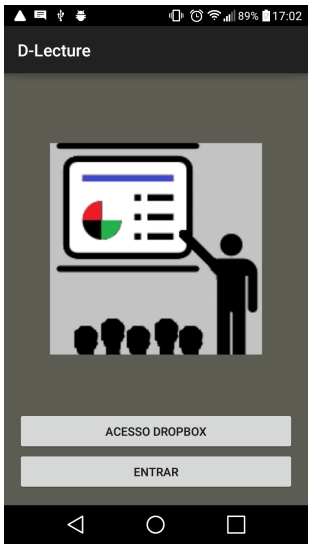


FIGURA 5.1: Menu inicial após arranque da aplicação Android

A verificação é feita em escada, ou seja, primeiro é verificado se o utilizador está conectado a uma rede e, caso não esteja, mostra no ecrã o aviso para se conectar primeiro a uma rede. Caso esteja conectado a uma rede, é verificado se está conectado a uma conta Dropbox, e caso não esteja, é mostrado outro aviso para se conectar a uma conta Dropbox. Os avisos podem ser visualizados nas figuras 5.2 e 5.3.

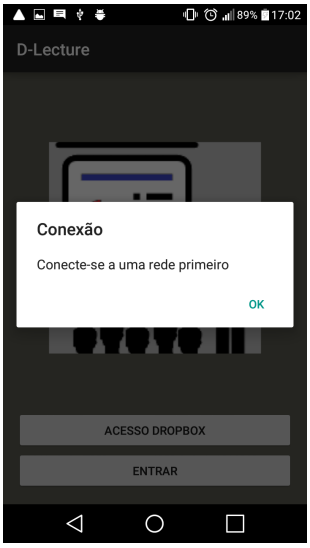


FIGURA 5.2: Aviso sobre conexão à rede

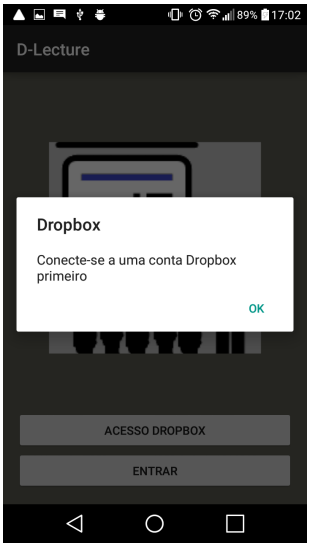


FIGURA 5.3: Aviso sobre conexão ao Dropbox

Caso o utilizador cumpra os dois requisitos, ao premir o botão "Entrar" irá então para o menu principal, como se pode ver na figura 5.4.

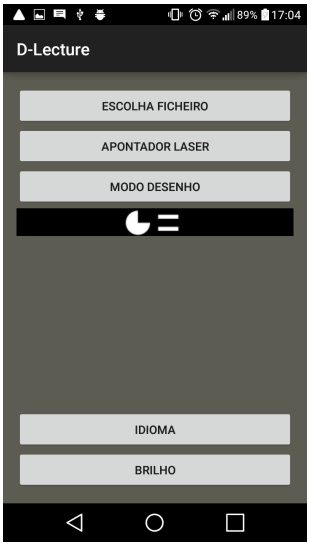


FIGURA 5.4: Menu principal da aplicação Android

5.2 Modificação do Brilho

Dentro do menu principal, o *Fragment* colocado inicialmente é o *Fragment Default*. Existe uma opção chamada "Brilho", que ao premir mostra uma barra com um ponto que permite deslizar (exemplificado nas figuras 5.5 e 5.6).

FIGURA 5.5: Menu principal normal

FIGURA 5.6: Opção de brilho premido

Ao deslizar para cima, o brilho do ecrã aumenta, e ao deslizar para baixo diminuí o brilho, como é possível verificar nas figuras 5.7 e 5.8.

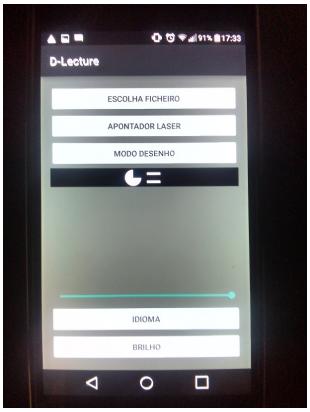


FIGURA 5.7: Brilho no máximo

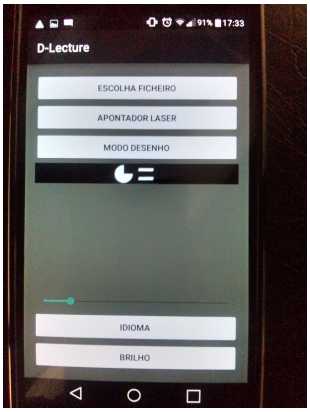


FIGURA 5.8: Brilho a cerca de 20%

5.3 Modificação do Idioma

Como para o brilho, existe também a opção para modificar o idioma na aplicação. Ao premir a opção é mostrado uma lista com as quatro opções ("English", "Português", "Français", "Deutsch"), onde o utilizador escolhe qual deseja, e depois prime o botão "OK". Isto origina a modificação do idioma na aplicação. A figura 5.9 mostra o diálogo no ecrã com as opções dos idiomas.

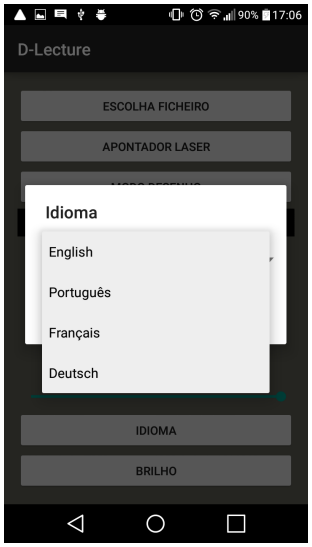


FIGURA 5.9: Escolha do idioma a utilizar na aplicação

As figuras 5.10 e 5.11 apresentam o menu principal em dois idiomas diferentes: português e inglês.

FIGURA 5.10: Aplicação em português

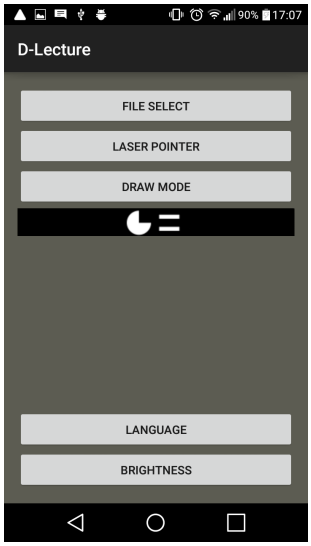


FIGURA 5.11: Aplicação em inglês

5.4 Conexão à conta Dropbox

Para conectar a uma conta Dropbox, o utilizador tem de premir a opção "Acesso Dropbox" no menu inicial. Uma vez premido a opção, o utilizador é reencaminhado para uma página *web* para fazer o *login* na sua conta Dropbox.

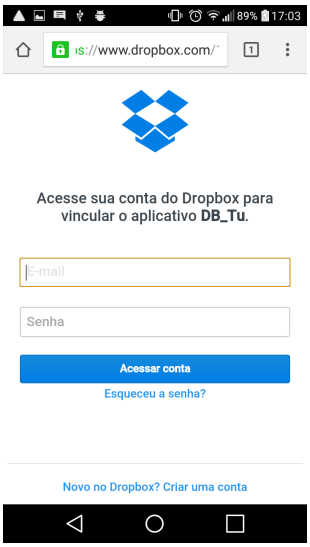


FIGURA 5.12: Página de autenticação da conta Dropbox

A figura 5.12 apresenta a página *web* onde o utilizador pode autenticar a sua conta Dropbox. Depois de preencher os campos e premir no botão para autenticar, caso a conta seja validada, é levado para a página de permissão para vincular a conta Dropbox à aplicação Dropbox, como se pode verificar na figura 5.13. O utilizador prime a opção para permitir, regressando para o menu inicial.

FIGURA 5.13: Página de vinculação da conta Dropbox à aplicação Dropbox

5.5 Escolha do Ficheiro

Dentro do menu principal, quando o utilizador prime a opção de "Escolha Ficheiro", é modificado o *Fragment* para o *Fragment File*. Aí o utilizador possui uma lista com os ficheiros e diretorias da conta Dropbox a ser utilizada, podendo entrar em subdiretorias ou escolher o ficheiro. A figura 5.14 apresenta o menu principal com a lista de campos contendo os ficheiros/diretorias na conta Dropbox.

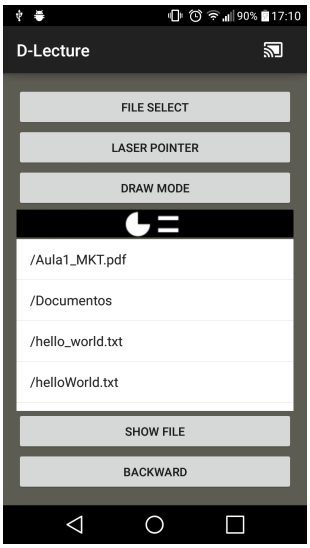


FIGURA 5.14: Menu principal após premida a opção "Escolha Ficheiro"

Quando o utilizador seleciona o campo de um ficheiro, é obtido o *share link* do ficheiro, que será convertido para um novo *link* para poder ser obtido o contéudo do ficheiro da parte da aplicação Chromecast. Na figura 5.15 podem ser visualizados ambos os *links*.

FIGURA 5.15: Visualização do *Sharelink* e *link* no *Debugger* do Android Studio

5.6 Deteção e conexão ao Chromecast

Quando o *Smartphone* e o Chromecast estão conectados à mesma rede, aparece o botão no canto superior direito. A figura 5.16 mostra o botão de conexão, indicando que existe pelo menos um Chromecast conectado à mesma rede que o *Smartphone*.

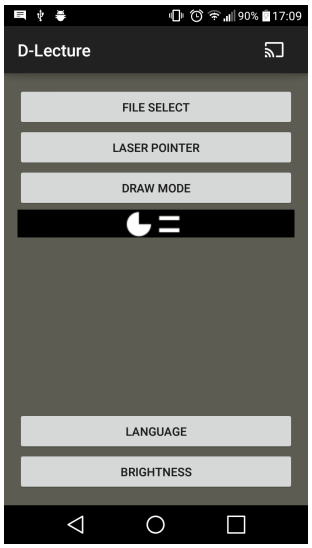


FIGURA 5.16: Botão após deteção de dispositivos compatíveis na rede

Quando pressionado, é apresentada a lista de Chromecasts que se encontram conectados à mesma, pedindo ao utilizador para escolher um dos dispositivos para se conectar, como pode ser verificado na figura 5.17.

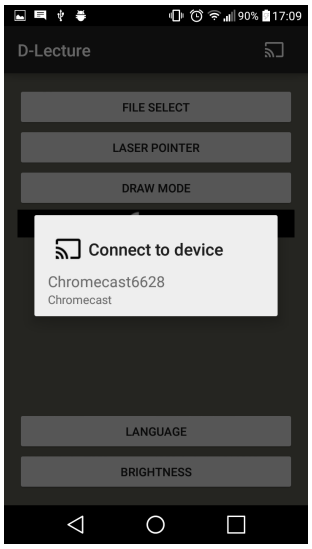


FIGURA 5.17: Dispositivos encontrados

Quando o utilizador seleciona uma das opções, começa o processo de conexão (explicado no capítulo anterior), e em caso de sucesso muda a imagem do botão (parte interior do botão muda de cor), como demonstrado na figura 5.18.

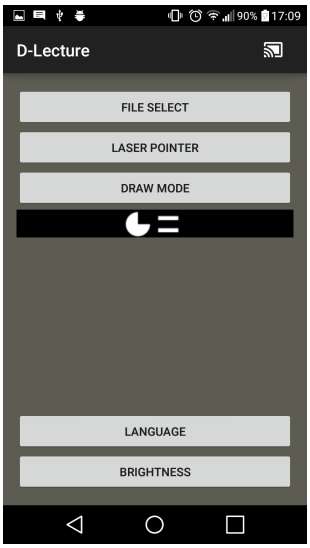


FIGURA 5.18: Indicação que a aplicação Android se conectou a um dispositivo Chromecast

5.7 Interação entre as Aplicações

Com o *Smartphone* e o Chromecast conectados (o mesmo para ambas as aplicações), o utilizador pode interagir com a aplicação Chromecast de três formas diferentes: escolha e visualização do ficheiro, modo laser e modo desenho.

5.7.1 Visualização do Ficheiro

Para visualizar o ficheiro, é necessário escolher qual o ficheiro a utilizar. Como se pode ver na figura 5.14, o utilizador precisa de escolher o ficheiro. Ao escolher um ficheiro (neste caso Aula1_MKT), é criado um *link*, é enviado para a aplicação Chromecast através do método *sendMessage*, como é possível verificar na figura 5.19.

FIGURA 5.19: Envio do *link* convertido no *Debugger* do Android Studio

Quando a aplicação Chromecast recebe o *link*, inicia-se o processo de obter o conteúdo do ficheiro e de processar o ficheiro para guardar no *array*.

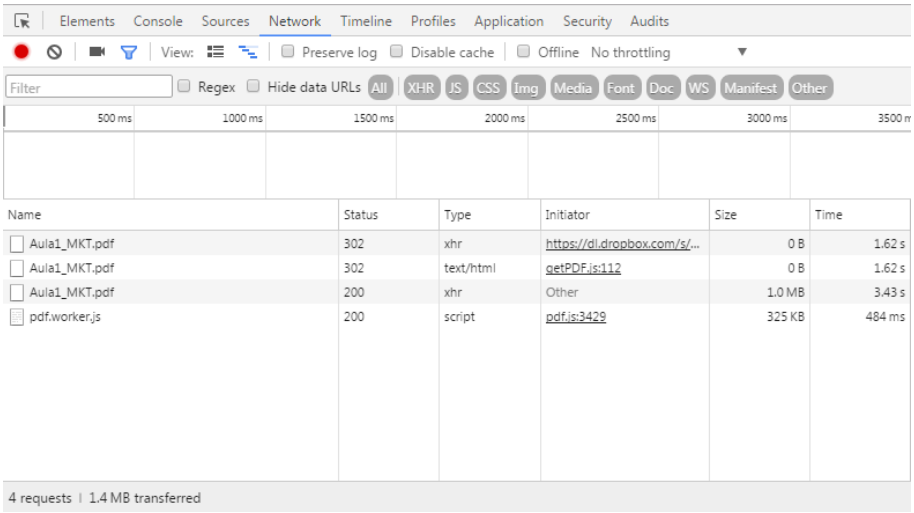


FIGURA 5.20: Aquisição do ficheiro por parte da aplicação Chromecast

A figura 5.20 demonstra no *debugger* do Chrome a aquisição do conteúdo do ficheiro por parte da aplicação Chromecast. Com o processo acabado, o utilizador só precisa de premir a opção de mostrar o ficheiro, onde é carregado a primeira página para ser visualizada, como a figura 5.21 demonstra.

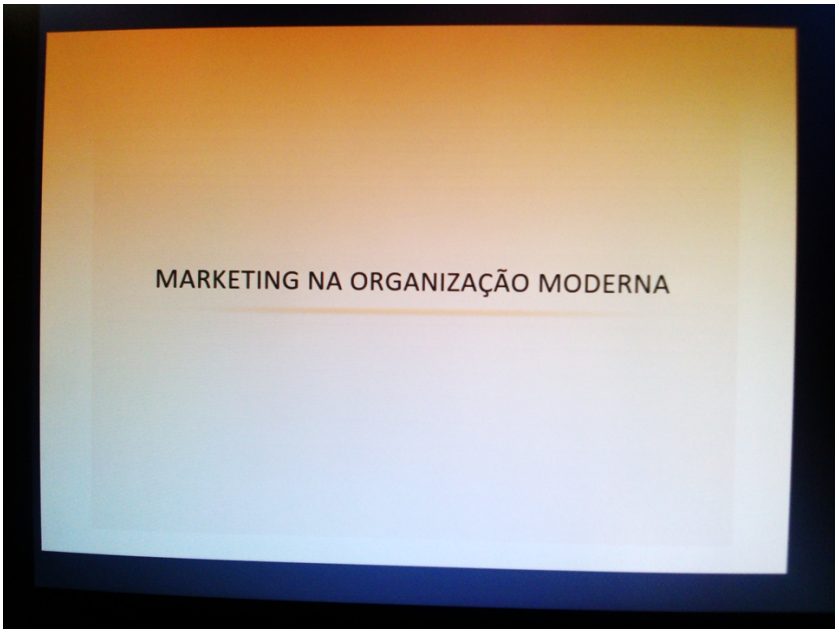


FIGURA 5.21: Visualização do ficheiro após aquisição e processamento

5.7.2 Modo Laser

Quando o utilizador entra no modo laser, é modificado o *Fragment* para o *Fragment Laser*, mostrando os dois botões: avançar e retroceder de página, como a figura 5.22 exibe.

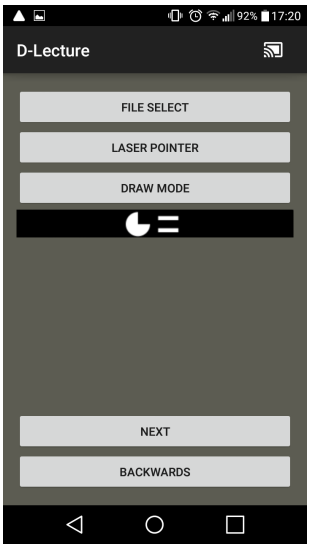


FIGURA 5.22: Modo laser na aplicação Android

A aplicação Android envia para a aplicação Chromecast o comando para entrar no modo laser também, fazendo com que o cursor apareça no ecrã como na figura 5.23.

FIGURA 5.23: Modo laser na aplicação Chromecast

À medida que o utilizador vai movimentando o *Smartphone*, os valores do acelerómetro são lidos e transformados em percentagens, sendo depois enviados para a aplicação Chromecast, onde o cursor se movimenta para a posição pretendida. A figura 5.24 demonstra a movimentação do cursor laser no ecrã durante a apresentação.

FIGURA 5.24: Movimentação do cursor laser na apresentação

5.7.3 Modo Desenho

Quando o utilizador entra no modo desenho, a aplicação Android inicializa ambos os *Fragments* (*Pad* e *Draw*), com a interface da aplicação a ficar como demonstrado na figura 5.25.

FIGURA 5.25: Modo desenho na aplicação Android

Para modificar a cor o utilizador pressiona a opção cor, causando a aparição da barra de cores para o utilizador escolher que cor prefere utilizar, como visto nas figuras 5.26 e 5.27.

FIGURA 5.26: Escolha da cor "castanho"

FIGURA 5.27: Escolha da cor "cinzento"

Quando o utilizador entra no modo desenho, é enviado para a aplicação Chromecast o comando para entrar no modo desenho, criando um cursor na apresentação para o utilizador saber onde irá começar a desenhar, como se pode verificar na figura 5.28.

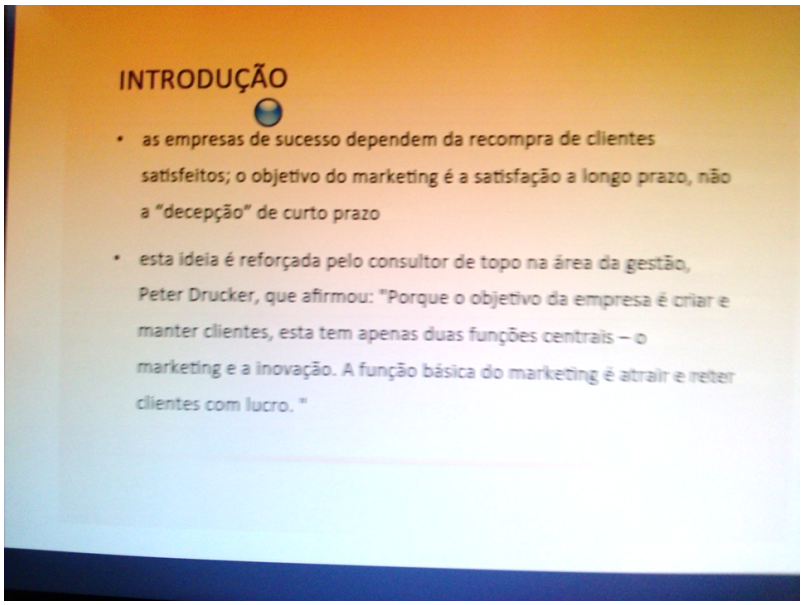


FIGURA 5.28: Modo desenho durante uma apresentação

Para desenhar, o utilizador precisa de dar dois toques na área do *Fragment Draw* (zona branca na aplicação Android), ativando a *flag* de desenho. Assim, quando o utilizador começa a movimentar o cursor, também começa a desenhar durante a apresentação. A figura 5.29 demonstra o processo de desenho durante a apresentação de um documento.

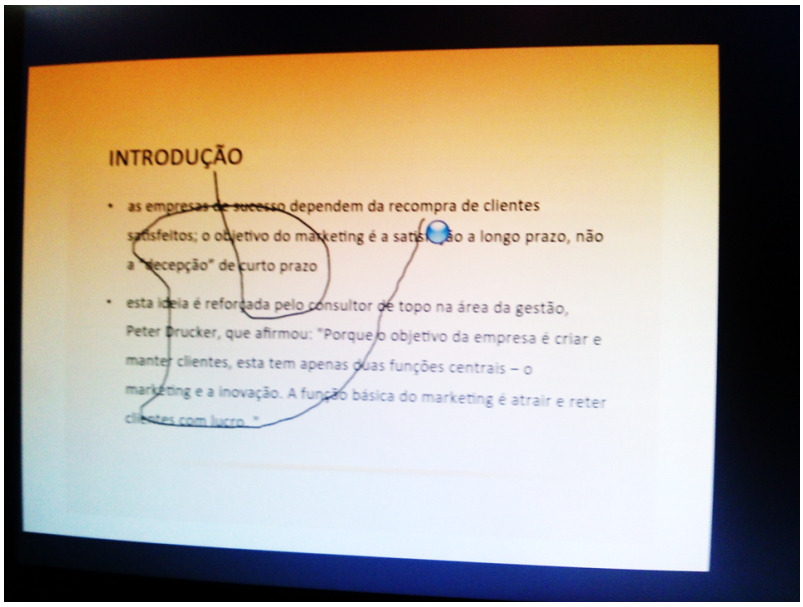


FIGURA 5.29: Desenho de traços durante uma apresentação

Caso o utilizador decidir modificar a cor, necessita de ir à barra de cores e

escolher uma nova, enviando para a aplicação Chromecast a nova cor. Assim os traços serão desenhados com a nova cor enquanto nenhuma cor nova for escolhida. A figura 5.30 apresenta a transição de uma cor para a outra, com a modificação da cor de preto para cinzento durante o processo de desenho.

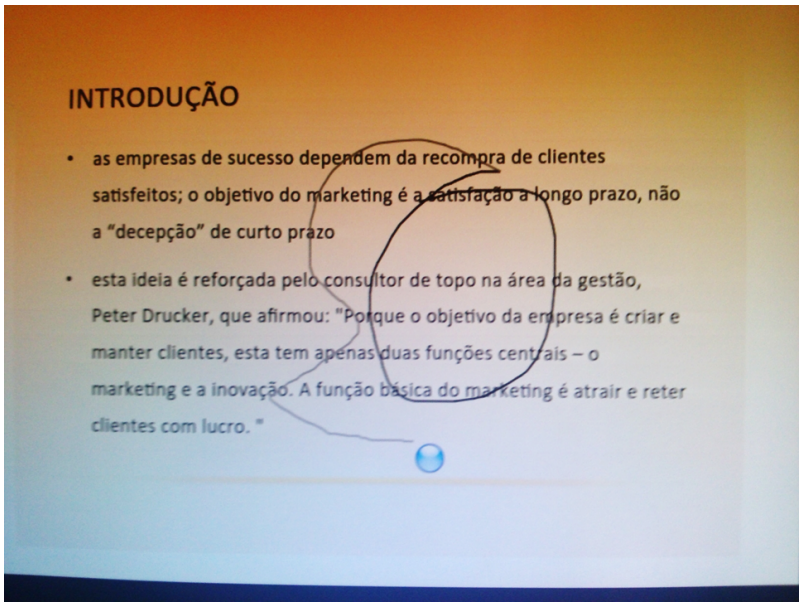


FIGURA 5.30: Modificação de cor durante o processo de desenho (para cinzento)

Quando o utilizador deseja modificar a página, basta recorrer o gesto "Fling". Quando o "Fling" é feito na direção esquerda, a apresentação retorna à página anterior. Quando é feito na direção direita, a apresentação avança para a página seguinte.

5.8 Consumo de Bateria

O teste da bateria aqui representado, foi realizado num *Smartphone* LG G4c com cerca de ano e meio, possuindo já um certo consumo na bateria. O teste em questão apresenta o maior consumo de bateria entre 9 *Smartphones* utilizados, dando assim a noção de quanto irá consumir um *Smartphone* mais antigo em relação à bateria.

O teste teve uma duração de de 1h e 16 minutos, onde a bateria começou nos 64%, e com a constante utilização das funcionalidades da aplicação móvel notou-se uma queda de 15% na bateria.

FIGURA 5.31: Utilização da aplicação em modo desenho no início do teste

FIGURA 5.32: Utilização contínua da aplicação em modo desenho

FIGURA 5.33: Utilização da aplicação em modo laser

FIGURA 5.34: Utilização contínua da aplicação em modo laser

A utilização da aplicação foi dividida pela utilização dos dois modos de interação, com o modo desenho a ser utilizado cerca de 40 minutos, e com o tempo restante a ser utilizado pelo modo laser.

Como é possível verificar entre o primeiro par de figuras (5.31 e 5.32) e o segundo par (5.33 e 5.34), com um tempo similar de uso entre os dois modos,

o modo laser acabou por consumir menos bateria que o modo desenho, o que é justificável pela comparação da complexidade entre os dois modos, onde o modo laser só precisa de ler os valores do acelerómetro enquanto que o modo desenho precisa de não só detetar o gesto no ecrã, mas também que gesto é, além do facto que o modo desenho é composto por dois *Fragments*, cada um com a sua função.

No entanto, para um telemóvel recente, regista-se um consumo satisfatório de bateria.

Capítulo 6

Conclusão e Trabalho Futuro

Neste capítulo são apresentadas as conclusões em relação ao desenvolvimento do projeto retratado nesta dissertação, assim como o trabalho futuro a ser implementado para melhorar o sistema, aumentando o número de funcionalidades, assim como expandir o sistema, com vista a aumentar a sua viabilidade.

6.1 Conclusões

O sistema implementado nesta dissertação consiste no desenvolvimento de duas aplicações que serão executadas em plataformas diferentes (*Smartphone* e Chromecast), assim como um repositório *online* (Dropbox), onde os ficheiros que serão utilizados para as apresentações se encontram guardados.

Com o sistema em questão, fica possível para o utilizador apresentar ficheiros durante uma reunião/aula, sem necessitar de recorrer a um computador portátil, sendo capaz de fazer uma apresentação simplesmente com um *Smartphone* e um Chromecast. Desta forma é possível reduzir o peso que precisa de suportar, bem como o espaço necessário para o uso do computador. A nova solução é de montagem simples, sendo apenas necessário conectar o Chromecast a um projetor por entrada HDMI, e a uma tomada para alimentar o Chromecast.

O único requisito que este sistema impõem é que o projetor possua entrada HDMI para conectar o Chromecast, permitindo assim a utilização de diversos modelos de projetos, desde os modelos mais antigos até aos mais recentes, permitindo utilizar funcionalidades de desenho que muitos projetores novos (e as

aplicações específicas que os controlam) disponibilizam.

A aplicação Android estabelece conexão a um repositório *online*, podendo navegar pelas diversas diretorias e procurar o ficheiro necessário para fazer a apresentação. A aplicação também permite ao utilizador controlar a apresentação, executando ações na aplicação que serão depois emuladas na aplicação Chromecast.

Na aplicação Chromecast, os comandos recebidos da aplicação Android são processados internamente, indo buscar o ficheiro escolhido e apresentando o ficheiro através do projetor. Também irá reponder aos comandos da aplicação Android nos dois modos apresentados, emulando as ações do utilizador, permitindo assim novas funcionalidades ao projetores.

Por fim, os requisitos inicias foram na sua maioria cumpridos, com as aplicações a responderem ao proposto, dotando os projetores mais antigos de novas funcionalidades, oferecendo a possibilidade de acrescentar mais opções/-funcionalidades. O sistema é de montagem simples e baixo custo, com um consumo de bateria durante o período de utilização muito satisfatório.

6.2 Trabalho Futuro

Um sistema como o implementado nesta dissertação ainda carece de algumas melhorias, desde novas funcionalidades, bem como, expandir o mesmo para novas plataformas.

Como primeira sugestão, é aconselhado dar suporte a mais repositórios *online*, visto que nem todos os utilizadores possuem uma conta Dropbox, e alguns utilizadores têm contas em múltiplos repositórios.

Para segunda sugestão, é colocado a opção de fazer *upload* de ficheiros no *Smartphone* para a conta Dropbox, sendo assim possível utilizar o ficheiro durante uma apresentação.

Como terceira sugestão, existe a expansão do modo desenho, colocando a possibilidade de modificar o tamanho do traço e do seu estilo, assim como tentar colocar a opção de adicionar formas.

Como quarta sugestão, a adição de colocar comentários/anotações na aplicação, expandindo assim às funcionalidades possíveis durante uma apresentação.

Por fim, a última sugestão é disponibilizar a aplicação para mais plataformas móveis, nomeadamente o iOS e o Windows Phone.

Referências

[1] T. Mtonga, "A situational analysis on the availability and access to computers for educational purposes by learners with visual impairements in Zambi: A case of three basic and three high schools for the blinds", in *Information and Communication Technology and Accessibility (ICTA), 2013 Fourth International Conference on*, Hammamet: IEEE, 2013, pp. 1–5.

[2] R. Pinter and S. M. Cisar, ""ONE QUESTION PER DAY" A MOBILE LEARNING PROJECT", in *Intelligent Systems and Informatics (SISY), 2014 IEEE 12th International Symposium on*, Subotica: IEEE, 2014, pp. 105–109.

[3] K. Qian, D. Lo, Y. Pan, Y. Zhang, and L. Hong, "Real-World Relevant Learning with Android Smartphones", in *Advanced Learning Technologies (ICALT), 2012 IEEE 12th International Conference on*, Rome: IEEE, 2012, pp. 476–477.

[4] L. Pei, J. Liu, R. Guinness, Y. Chen, T. Kröger, R. Chen, and L. Chen, "The Evaluation of WiFi Positioning in a Bluetooth", in *Ubiquitous Positioning, Indoor Navigation, and Location Based Service (UPINLBS), 2012*, Helsinki: IEEE, 2012, pp. 1–6.

[5] AAXA TECHNOLOGIES INC, *AAXA LED Android Micro Projector - DLP Portable LED Micro Projector - Android Operating System*. [Online]. Available: `http://aaxatech.com/products/led{\_}showtime{\_}android{\_}projector.html`.

[6] BenQ Corporation, *BenQ Europe | Products - Projectors - QPresenter Pro-Android*. [Online]. Available: `http://benq.eu/product/projector/qpresenterpro-android/`.

[7] Seiko Epson Corporation, *Epson iProjection - Epson*. [Online]. Available: `https://www.epson.pt/pt/pt/viewcon/corporatesite/cms/index/10815/`.

[8] Panasonic Corporation, *Wireless Projector for Android | Projector | Panasonic Global*. [Online]. Available: `http://panasonic.net/avc/projector/android/`.

[9] Gartner, *Gartner Says Worldwide Smartphone Sales Grew 3.9 Percent in First Quarter of 2016*, 2016. [Online]. Available: http://www.gartner.com/newsroom/id/3323017.

[10] T.-M. Grønli, J. Hansen, G. Ghinea, and M. Younas, "Mobile Application Platform Heterogeneity: Android vs Windows Phone vs iOS vs Firefox OS", in *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*, Victoria, BC: IEEE, 2014, pp. 635–641.

[11] D. Morrill, *Announcing the Android 1.0 SDk, release 1 | Android Developers blog*, 2008. [Online]. Available: http://android-developers.blogspot.in/2008/09/announcing-android-10-sdk-release-1.html.

[12] F. Maker and Y.-H. Chan, "A Survey on Android vs. Linux", in *A Survey on Android vs. Linux*, California: University of California, 2009, pp. 1–10.

[13] *No Title*. [Online]. Available: http://cdn.edureka.co/blog/wp-content/uploads/2013/01/Android-Stack1.jpg.

[14] Apple Inc., *About the iOS Technologies*. [Online]. Available: https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html.

[15] ——, *Core OS Layer*. [Online]. Available: https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/CoreOSLayer/CoreOSLayer.html{\#}//apple{_}ref/doc/uid/TP40007898-CH11-SW1.

[16] ——, *Core Services Layer*. [Online]. Available: https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/CoreServicesLayer/CoreServicesLayer.html{\#}//apple{_}ref/doc/uid/TP40007898-CH10-SW5.

[17] ——, *Media Layer*. [Online]. Available: https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/MediaLayer/MediaLayer.html{\#}//apple{_}ref/doc/uid/TP40007898-CH9-SW4.

[18] ——, *Cocoa Touch Layer*. [Online]. Available: https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSTechnologies/iPhoneOSTechnologies.html{\#}//apple{_}ref/doc/uid/TP40007898-CH3-SW1.

[19] Microsoft, *Guide to Universal Windows Platform (UWP) apps - Windows app development*. [Online]. Available: https://msdn.microsoft.com/library/windows/apps/dn894631.aspx.

[20] *No Title*. [Online]. Available: http://img.directindustry.com/images{_}di/photo-mg/101395-3181735.jpg.

[21] J. Turley, "Atmel AVR Brings RISC to 8-Bit World", *Microprocessor Report*, vol. 11, no. 9, p. 4, 1997. [Online]. Available: ftp://pti.kpi.ua/pub/electric/Atmel/acrobat/atmelavr.pdf.

[22] Raspberry Pi Foundation, *Raspberry Pi Downloads - Software for the Raspberry Pi*. [Online]. Available: https://www.raspberrypi.org/downloads/.

[23] Google Inc., *Para TV - Chromecast - Google*. [Online]. Available: https://www.google.com/intl/pt{_}pt/chromecast/tv/explore/?gclid=COPj076d2ckCFYoBwwodClYE6w.

[24] HDMI Licensing, LLC, *HDMI :: Resources :: FAQ*. [Online]. Available: http://www.hdmi.org/learningcenter/faq.aspx{\#}1.

[25] A. Martonik, *Chromecast is now the No. 1 streaming device in the U.S., users have casted 1 billion times | Android Central*. [Online]. Available: http://www.androidcentral.com/chromecast-now-number-1-streaming-stick-us-users-have-casted-1-billion-times.

[26] Google Inc., *Specs and supported operating systems - Chromecast help*. [Online]. Available: https://support.google.com/chromecast/answer/3046409?hl=en.

[27] ——, *Get Started | Cast | Google Developers*. [Online]. Available: https://developers.google.com/cast/docs/developers.

[28] *No Title*. [Online]. Available: https://www.google.com/intl/pt-BR{_}br/chromecast/static/images/tv/chromecast.jpg.

[29] Raspberry Pi Foundation, *Raspberry Pi FAQs - Frequently Asked Questions*. [Online]. Available: https://www.raspberrypi.org/help/faqs/{\#}introWhatIs.

[30] *No Title*. [Online]. Available: https://www.raspberrypi.org/wp-content/uploads/2014/07/rsz{_}b-.jpg.

[31] Raspbrry Pi Foundation, *Raspberry Pi Model Specifications - Raspberry Pi Documentation*. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/models/specs.md.

[32] D. Chheda, D. Darde, and S. Chitalia, "Smart Projectors using Remote Controlled Raspberry Pi", *International Journal of Computer Applications*, vol. 82, no. 16, pp. 6–11, 2013.

[33] *Identify your Apple TV model*. [Online]. Available: https://support.apple.com/en-us/HT200008.

[34] *Apple TV (4.ª geração) - Especificações técnicas*. [Online]. Available: https://support.apple.com/kb/SP724?locale=pt{_}PT.

[35] *Apple TV and tvOS*. [Online]. Available: https://developer.apple.com/library/content/documentation/General/Conceptual/AppleTV{_}PG/.

[36] C. S. Wang and S. L. Lin, "Why are people willing to pay for cloud storage service?", in *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, 2016, pp. 1–6. DOI: 10.1109/ICIS.2016.7550806.

[37] Z Li, Z. L. Zhang, and Y Dai, "Coarse-grained cloud synchronization mechanism design may lead to severe traffic overuse", *Tsinghua Science and Technology*, vol. 18, no. 3, pp. 286–297, 2013. DOI: 10.1109/TST.2013.6522587.

[38] Dropbox, Inc., *Dropbox release notes*. [Online]. Available: https://www.dropbox.com/release{_}notes.

[39] *Core API*. [Online]. Available: https://www.dropbox.com/developers-v1/core.

[40] *Build your app on the Dropbox platform*. [Online]. Available: https://www.dropbox.com/developers.

[41] *OAuth 2.0*. [Online]. Available: https://oauth.net/2/.

[42] *OAuth guide*. [Online]. Available: https://www.dropbox.com/developers/reference/oauth-guide.

[43] Microsoft, *Use OneDrive with Office*. [Online]. Available: https://support.office.com/en-us/article/Use-OneDrive-with-Office-b1c976de-ef52-4d53-950f-d48f2c6427df.

[44] ——, *SDKs for OneDrive integration*. [Online]. Available: https://dev.onedrive.com/sdks.htm.

[45] ——, *Getting started with OneDrive API*. [Online]. Available: https://dev.onedrive.com/getting-started.htm.

[46] ——, *Registering your app for OneDrive API*. [Online]. Available: https://dev.onedrive.com/app-registration.htm.

[47] S. Pichai, *Introducing Google Drive... yes, really*, 2012. [Online]. Available: https://googleblog.blogspot.pt/2012/04/introducing-google-drive-yes-really.html.

[48] Google Inc., *Google Drive*. [Online]. Available: https://www.google.com/intl/pt-PT/drive/.

[49] Dropbox, Inc., *Quanto custa o Dropbox?* [Online]. Available: https://www.dropbox.com/help/73.

[50] Microsoft, *OneDrive storage plans*. [Online]. Available: https://onedrive.live.com/about/en-US/plans/.

[51] Google Inc., *Explore as funcionalidades de armazenamento do Drive*. [Online]. Available: https://www.google.com/intl/pt-PT/drive/using-drive/.

[52] ——, *Google Drive APIs*. [Online]. Available: https://developers.google.com/drive/.

[53] ——, *Getting Started*. [Online]. Available: https://developers.google.com/drive/android/get-started.

[54] Dr. M. Elkstein, *Learn REST: A Tutorial*. [Online]. Available: http://rest.elkstein.org/.

[55] Google Inc., *Introduction to the Google Drive Platform for iOS*. [Online]. Available: https://developers.google.com/drive/ios/intro.

[56] ——, *Google Drive REST API Overview*. [Online]. Available: https://developers.google.com/drive/v3/web/about-sdk.

[57] ——, *About Authorization*. [Online]. Available: https://developers.google.com/drive/v3/web/about-auth.

[58] Apple Inc., *Compre a Apple TV*. [Online]. Available: http://www.apple.com/pt/shop/buy-tv/apple-tv.

[59] I. Dropbox, *How much free referral space can I earn?* [Online]. Available: https://www.dropbox.com/help/200.

[60] Google Inc., *Android Studio Overview | Android Developers*. [Online]. Available: http://developer.android.com/tools/studio/index.html.

[61] The Apache Software Foundation, *The Apache Ant Project*. [Online]. Available: http://ant.apache.org/.

[62] Google Inc., *Run Apps on the Android Emulator*. [Online]. Available: https://developer.android.com/studio/run/emulator.html.

[63] J. s.r.o, *The smartest JavaScript IDE*. [Online]. Available: https://www.jetbrains.com/webstorm/?fromMenu.

[64] ——, "Debugging, tracing and testing", [Online]. Available: https://www.jetbrains.com/webstorm/features/debugging-and-testing.html.

[65] ——, *IDE features*. [Online]. Available: https://www.jetbrains.com/webstorm/features/ide-features.html.