App Install & Go

Roberto Filipe Manso Barreto
(nrº 21123, regime diurno)

Orientação de
Luís Gonzaga Martins Ferreira

Licenciatura em Engenharia em Sistemas Informáticos
Escola Superior de Tecnologia
Instituto Politécnico do Cávado e do Ave

Identificação do aluno

Roberto Filipe Manso Barreto
Aluno número 21123, regime diurno
Licenciatura em Engenharia em Sistemas Informáticos

Orientação

Luís Gonzaga Martins Ferreira

Informação sobre o Estágio

Motorline Eletrocelos S.A
Travessa do Sobreiro, 29 Rio Côvo (Sta. Eugénia) 4755-474 Barcelos
Eng. Helder Remelhe

Resumo

Este documento trata o processo de análise, especificação e implementação da solução Install&Go. Esta vem resolver o problema de comunicação entre a empresa Motorline e os seus técnicos, dado que, na eventualidade de um problema estes deverão telefonar para a empresa, o que gera sobrecarga do sistema.

Para a resolução deste problema foi desenvolvido uma plataforma de fórum, onde as empresas podem registar os seus técnicos. Aqui, podem expor questões, onde técnicos externos de outras empresas, e/ou técnicos Motorline poderão prestar auxílio. Se um técnico possuir um problema já resolvido, este poderá pesquisar pela solução dada no fórum.

O desenvolvimento desta solução proveu a aquisição de novas capacidades tecnológicas como o desenvolvimento *cross-platform* e as suas *frameworks*, sendo que destas foi explorado o *flutter*. Através da elaboração desta aplicação foi possível, para além das competências mencionadas anteriormente, assimilar capacidades como análise, especificação de projetos e comunicação com clientes. Por fim, foi desenvolvida completamente a solução conforme as necessidades e expectativas do cliente.

Abstract

This document describes the analysis, specification and development process of the Install&Go solution. This solution solves the comunication problem between Motorline and their professionals, since that, in the event of a problem, they must call the company, which generates an overload.

To solve this problem a forum platforma was developed, where companies can register their professionals. Here they can submit questions, so that other professionals from other companies and/or Motorline can provide assistance. If a professional has a problem that has already neem solved, he can search for the solution in the forum

The development of this solution provided the acquisition of new technological skills such as cross-plataform development and its frameworks, of which flutter was explored.Through the elaboration of this application it was possible, besides the competences previously mentioned, to assimilate skills such as analysis, project specification and comunication with clients. At the end, the solution was completely developed according to the client's needs and expectations.

Agradecimentos

Em primeiro lugar, gostaria de agradecer à minha família, com destaque à minha mãe, ao meu padrinho, aos meus avós e à minha namorada, que com todo o carinho orientaram-me nesta caminhada.

Também, gostaria de salientar um caloroso agradecimento à empresa Motorline, uma vez que, fizeram-me sempre sentir um membro da equipa, com destaque ao supervisor Helder Remelhe que sempre esteve disponível para eventuais dúvidas que surgissem no desenvolvimento do projeto.

Por fim, mas não menos importante, gostaria de enfatizar toda a orientação, disponibilidade, conversa e ensinamentos proporcionados pelo professor doutor Luís Ferreira e também aos meus colegas de curso Henrique Cartucho e João Castro que mais que colegas, são amigos para a vida.

Conteúdo

Lista de Figuras

Lista de Tabelas

1. Estado da arte

Para a organização de todo o trabalho a desenvolver, visto que este é dividido com mais uma colega, foi utilizada a técnica de desenvolvimento ágil, através da qual foi possível organizar todas as tarefas entre os elementos de desenvolvimento do projeto.

1.1 Ferramentas de trabalho utilizadas

A organização de todas as tarefas, foi realizada na ferramenta *Github Projetcs*. Esta dispõe de funções que permitem ligar um projeto a um repositório de *Github*, onde se consegue personalizar completamente todo o projeto e parâmetros das tarefas que resulta numa organização minuciosa.

O Microsoft Excel foi utilizado para a engenharia de *software* onde foram descritos os requisitos do projeto, *user stories* e também a especificação de casos de uso. Esta ferramenta também foi utilizada para a organização de reuniões com o cliente e redação de tópicos a abordar.

Para o desenvolvimento do *design* do *software* foi utilizada a ferramenta *figma*, que permite o *design* de todas as componentes tendo em conta as reais dimensões de um dispositivo. Esta ferramenta dispõe de funções para criar apresentações interativas que conseguem demonstrar o comportamento da aplicação como resultado final, dando também suporte à implementação.

O *draw.io* foi utilizado para os desenhos das arquiteturas do projeto tendo revelado grande auxílio, uma vez que, permite uma grande liberdade ilustrativa. Esta ferramenta permite conectar com o *github* o que proporciona a facilidade de guardar projetos e ter acesso a partir de qualquer dispositivo.

A engenharia de *software* foi realizada através da utilização *Visual Studio Paradigm*, esta é uma ferramenta muito completa que contém modelos e regras para a engenharia de *software*. Esta tornou-se um grande recurso no desenvolvimento da base de dados, dado que é possível desenhar o modelo e exportar para um ficheiro de criação de base de dados.

1.2 Plataforma Tecnológica

1.2.1 Web Scraper

Web scraping é terminologia dada à "(...)*construction of an agent to download, parse, and organize data from the web in an automated manner*(...)"(vanden Broucke & Baesens, 2018). O grande problema com *web scraping* é que poderá ser considerado ilegal e é facilmente detetado. Tendo em conta este problema surgiram duas grandes formas de realizar *web scraping*. A forma mais comum de *web scraping* é realizar um pedido para obter uma página web e ler esta, sendo assim um processo rápido e simples.

A segunda forma de realizar *web scraping* é através da simulação da ação humana com da abertura do navegador programáticamente, pesquisa pela página desejada, descarregar e daí ler os dados. Este torna-se um processo lento e complexo.

A grande diferença entre estas duas formas é a velocidade, visto que a segunda forma tem de esperar que o navegador inicie, de seguida terá de esperar que a página carregue e apenas após este processo se poderá ler os dados.

1.2.1.1 Selenium

O *Selenium* é uma ferramenta "(...)*for a range of tools and libraries that enable and support the automation of web browsers*(...)"(Selenium, 2023), esta provém "(...)*extensions to emulate user interaction with browsers*(...)"(Selenium, 2023). Na sua base esta "(...)*is WebDriver, an interface to write instruction sets that can be run interchangeably in many browsers*(...)"(Selenium, 2023). Esta ferramenta provém também a possibilidade de escalar com *multi threading*, o que permite abrir diversas janelas do navegador simultaneamente e obter dados destas o que diminui drasticamente o tempo de execução, esta funcionalidade não foi explorada devido a limitações de *hardware*, mas seria uma importante implementação futura.

1.2.2 Serviços Backend

Para a realização da integração entre a aplicação *frontend* e os dados, foi necessário desenvolver uma API para dar suporte a todos os serviços necessários para a aplicação. API sigla para "(...)*Application Programming Interface*(...)" (Masse, 2011), "(...)*exposes a set of data and functions to facilitate interactions between computer programs and allow them to exchange information.*(...)" (Masse, 2011). Estas ferramentas apesar de serem "(...)*designed to work with other programs, they're mostly intended to be understood and used by humans writing those other programs*(...)"(Jin et al., 2018).

1.2.2.1 Serviços RestFull e SOAP

Os seviços *RestFull* "(...)*expose data as resources and use standard HTTP methods to represent Create, Read,Update, and Delete (CRUD) transactions against these resources*(...)"(Jin et al., 2018), sendo que a resposta é no formato *JSON* "(...)*due to its simplicity and ease of use with JavaScript, JSON has become the standard for modern APIs*(...)"(Jin et al., 2018).

Já *SOAP* é "(...)*XML-based envelope for the information being transferred, and a set of rules for translating application and platform-specific data types into XML representations*(...)"(Snell et al., 2002) sendo que a resposta é realizada em *XML* leva a que "(...)*It relies heavily on XML standards like XML Schema and XML Namespaces for its definition and function*(...)"(Snell et al., 2002). A utilização de *XML* permite que "(...)*two applications, regardless of operating system, programming language, or any other technical implementation detail, may openly share information using nothing more than a simple message encoded in a way that both applications understand*(...)"(Snell et al., 2002).

Por fim foi decidido utilizar *RestFull* devido a ser *much lighter compared to SOAP. It does not require formats like headers to be included in the message, like it is required in SOAP architecture*(...)"(Halili & Ramadani, 2018). Outra maior valia é que "(...)*it parses JSON, a human readable language designed to allow data exchange and making it easier to parse and use by the computer. It is estimated to be at around one hundred times faster than XML*(...)"(Halili & Ramadani, 2018).

1.2.2.2 NodeJS

Para o desenvolvimento do projeto *backend* foi escolhido *NodeJS*, este surgiu quando "(...)*the original developers took JavaScript, something you could usually only run inside the browser, and they let it run on your machine as a standalone process*(...)"(Mead, 2018) isto significa que é possível correr código "(...)*JavaScript outside the context of the browser*(...)"(Mead, 2018) .

Para correr *JavaScript* a nível de servidor *web* e a nível de computador pessoal é utilizado o mesmo motor, "(...)*it's called the V8 JavaScript runtime engine*(...)"(Mead, 2018), este é "(...)*an opensource engine that takes JavaScript code and compiles it into much faster machine code. And that's a big part of what makes Node.js so fast*(...)"(Mead, 2018) .

NodeJs permite a utilização de bibliotecas externas "(...)*by providing the Node Package Manager*(...)"(Mead, 2018), este provém ao programador "(...)*to easily install, manage, and even provide your own modules for a rapidly grown and well-maintained open source repository*(...)"(Mead, 2018).

NodeJS foi escolhido para o *backend*, uma vez que, permite a utilização de *Typescript* para o desenvolvimento e por ser vastamente utilizado, o que dá acesso a diversas fontes de informação para a resolução de problemas, assim como, para o auxílio ao desenvolvimento.

1.2.2.3 Typescript

O Typescript "(...)*is a bit unusual as a language in that it neither runs in an interpreter (as Python and Ruby do) nor compiles down to a lower-level language (as Java and C do).*(...)"(Vanderkam, 2019) isto porque este "(...)*compiles to another high-level language, JavaScript*(...)"(Vanderkam, 2019), isto faz com que o *Typescript* seja visto como "(...)*a superset of JavaScript in a syntactic sense*(...)"(Vanderkam, 2019).

Todos os programas *JavaScript* "(...)*are TypeScript programs, but the converse is not true*(...)"(Vanderkam, 2019), "(...)*This is because TypeScript adds additional syntax for specifying types*(...)"(Vanderkam, 2019). O sistema de tipagens do *TypeScript* tem como objetivo "(...)*to detect code that will throw an exception at runtime, without having to run your code*(...)"(Vanderkam, 2019). "(...)*The type checker cannot always spot code that will throw exceptions, but it will try*(...)"(Vanderkam, 2019).

Esta linguagem foi escolhida para o *backend*, visto que, assegura as tipagens, o que proporciona um maior nível de segurança quando se trabalha com dados recebidos, assim como, a agilização do processo de programação devido à capacidade de prever a maioria dos erros de código.

1.2.2.4 PostgreSQL

PostgreSQL "(...)*is an open source object relational database management system*(...)"(Juba et al., 2015). Esta "(...)*emphasizes extensibility*(...)"(Juba et al., 2015) o que permite a utilização de extensões desenvolvidas pela comunidade, mas "(...)*Also, there are several extensions to access, manage, and monitor PostgreSQL clusters, such as pgAdmin III*(...)"(Juba et al., 2015). Esta ferramenta é de aprendizagem simples devido ao facto de "(...)*it complies with ANSI SQL standards*(...)"(Juba et al., 2015), o que leva a que, qualquer indivíduo com conhecimentos prévios em *SQL* consiga facilmente aprender esta tecnologia.

PostgreSQL foi escolhido para a base de dados, dado que, a empresa já utiliza vastamente esta tecnologia, mas também porque é *open-source* e não existem custos associados para este tipo de utilização. A funcionalidade de extensões foi utilizada para implementar *id's*, que utilizam a estrutura *uuid*, o que dificulta o ataque aos dados, uma vez que, é difícil de prever os valores de *id's*.

1.2.2.5 Logs e Logging

Logs "(...)*are a very useful source of information for computer system resource management (printers, disk systems, battery backup systems, operating systems, etc.), user and application management (login and logout, application access, etc.), and security*(...)"(Chuvakin et al., 2012). Um Log é "(...)*what a computer system, device, software, etc. generates in response to some sort of stimuli. What exactly the stimuli are greatly depends on the source of the log message*(...)"(Chuvakin et al., 2012), ou seja, perante um estímulo desejado, um *log* poderá ser gerado. Os dados dos *logs* são "(...)*the intrinsic meaning that a log message has*(...)"(Chuvakin et al., 2012), o que significa que estes contêm apenas dados relevantes ao objetivo do *log*. *Logging* é o nome que se dá ao processo de geração de *logs*.

Neste contexto *logging* poderá ser utilizado para realizar a monitorização de pedidos e erros. Estas informações poderão até auxiliar na toma de decisões sobre o *software* e em quais funcionalidades colocar mais atenção.

1.2.2.6 Morgan

Morgan é uma biblioteca que permite extrair dados de um pedido, assim como a criação de *logs*. Este atua como um *middleware* do servidor, que recebe qual o tipo de *log* a ser escrito, estes estão definidos na biblioteca. Os principais dados obtidos por este são, a data e hora do pedido, o tipo, o serviço, os dados recebidos, a resposta devolvida e também a descrição do sistema utilizado. Com estes dados é possível saber que plataforma é mais utilizada no *software*, quais as horas de maior utilização e quais os serviços mais executados. Estes dados permitem direcionar mais recursos para uma indicada plataforma e/ou serviço, assim como escolher os melhores horários de manutenção dos servidores.

1.2.2.7 Gestão de *emails*

O envio de *emails* para os utilizadores, foi desenvolvido através da biblioteca *nodemailer*, que permite a utilização de um servidor de *SMTP*. Esta ferramenta foi escolhida devido a ser uma das mais utilizadas para este tipo de necessidade, o que permite que exista mais informação sobre a mesma que auxilia a resolução e identificação de erros.

Para desenvolver o conteúdo dos *emails* foi utilizada a ferramenta *Tabular Email*, esta permite realizar o *design* do conteúdo de um *email*, sendo possível exportar para *html*. A maior dificuldade desta ferramenta é que não permite a utilização de acentuação, e visto que o *html* é gerado por uma máquina este torna-se complicado de navegar e traduzir.

1.2.2.8 Agendamento de tarefas

Um requisito deste projeto foi o envio diário de *emails* com o relatório de notificações ao final do dia. Primeiramente, para realizar o agendamento de tarefas, foi feita uma análise das ferramentas existentes para a realização deste tipo de ações. Deste modo, foram encontradas o *cronetab* e o *node-cron*. A grande diferença entre estas duas ferramentas é que o *cronetab* funciona a nível de servidor, sendo que, o funcionamento tem por base "(...)*run this command at this time on this date*(...)"(Linux, 2023), este comando poderá por exemplo executar um código para enviar *emails*. Já o *node-cron* trata-se de uma biblioteca de *NodeJs* "(...)*in pure JavaScript for node.js based on GNU crontab*(...)"(merencia, 2023), este permite o fácil agendamento de tarefas de forma programática, assim como a indicação do código a ser executado sem necessidade de criar comandos.

A hora de execução do código de envio de *emails* poderá variar e necessitar de reprogramação, pelo que, foi optada a utilização do *node-cron*, uma vez que, facilita a utilização e agiliza o processo de reprogramação de horas de envio do relatório de notificações.

1.2.2.9 Encriptação de *passwords*

Para garantir a segurança das *passwords* dos utilizadores é necessário encriptar estas, a encriptação poderá ser realizada manualmente ou com o auxílio de ferramentas, a grande diferença é que manualmente poderá não se obter uma encriptação tão segura como com o auxílio de uma ferramenta. Sendo assim, foi decidido utilizar uma ferramenta para encriptação de *passwords*, a ferramenta escolhida foi *bcrypt*. Esta foi escolhida devido a ser vastamente utilizada e tem por base a *hash bcrypt*, esta "(...)*uses a variant of the Blowfish encryption algorithm's keying schedule, and introduces a work factor, which allows you to determine how expensive the hash function will be*(...)"(Hale, 2023) isto permite que esta acompanhe a lei de *Moore*, pois "(...)*As computers get faster you can increase the work factor and the hash will get slower*(...)"(Hale, 2023). Esta ferramenta oferece um conjunto de métodos dos quais foram utilizados os de *hash* e de *compare*. O método de *hash* permite através de um valor, chamado *salt*, que não indica a complexidade a aplicar sobre a encriptação, sendo de seguida devolvida a *password* encriptada. O método *compare* permite comparar uma *password* encriptada com uma *password* não encriptada, e devolve verdadeiro ou falso conforme as *passwords* sejam iguais ou não.

1.2.2.10 Encriptação de configurações do servidor

Com o objetivo de garantir um nível de segurança maior foram realizadas pesquisas sobre as principais falhas de segurança no *NodeJs*. Nestas, foi descoberto que as principais formas de ataque são as bibliotecas de *malware* e o ataque direto com o objetivo de obter dados de acesso a servidores que se encontram nos ficheiros de configuração.

Por norma, nas metodologias mais recentes de desenvolvimento de *software*, é sugerido que se coloque todas as configurações de servidores num ficheiro à parte, devido à praticidade de gerir estes dados, mas esta leva a um nível de segurança mais baixo, uma vez que, se alguém conseguir acesso a este ficheiro, consegue obter todos os dados de configuração de servidores. Neste projeto foi utilizado o ficheiro *env* para este fim, este no momento de iniciar o servidor é utilizado para carregar todas as variáveis para o ambiente do mesmo. Sendo assim qualquer um com acesso ao ficheiro ou às variáveis de ambiente poderá ver todas as configurações do servidor.

A solução mais indicada para este problema é a encriptação do ficheiro *env* e das variáveis de ambiente. A biblioteca mais utilizada para este objetivo é a *secur-env*, visto que, esta permite realizar a encriptação de um ficheiro com a indicação de uma *password*. A *password* deverá ser indicada no processo de inicialização do servidor de forma a ser possível ao mesmo desencriptar o ficheiro, sendo que a gestão das variáveis cifradas passa então a estar encarregue desta biblioteca.

Embora exista esta solução, continuam a haver possibilidades de ataque, uma vez que, é possível ver o histórico do terminal do servidor, o que permite obter a *password* escrita. Para resolver este problema é indicada a biblioteca *readline*, pois esta possui o modo de *password* que apaga o histórico do terminal sempre que utilizado. Esta contém a vertente assíncrona, *readline* e a vertente síncrona, *readline-sync*. Neste projeto, foi utilizada a versão síncrona da biblioteca visto que o objetivo é o servidor apenas inicie após a indicação da *password* sem nenhum serviço a correr em simultâneo.

1.2.2.11 Firebase

Firebase é uma solução que foi comprada pela *Google* em 2014. O seu objetivo é "(...)*to provide the tools and infrastructure that you need to build great apps*(...)"(Moroney, 2017), esta alcança este objetivo através da oferta de serviços pré configurados, sendo que "(...)*many of the technologies are available at no cost*(...)"(Moroney, 2017).

Firestorage também conhecida como *cloud storage*, é um serviço que dispõe "(...)*a simple API that is backed up by Google Cloud Storage*(...)"(Moroney, 2017), que permite guardar e transmitir até um gigabyte de ficheiros de forma gratuita.

Cloud messaging é também um serviço do *Firebase* que permite "(...)*to reliably deliver messages at no cost*(...)"(Moroney, 2017). Este garante que"*Over 98% of connected devices receive these messages in less than 500ms*(...)"(Moroney, 2017). *Cloud messaging* permite a utilização de diversas formas de envio de notificações como "(...)*driven by analytics to pick audiences, or using topics or other methods*(...)"(Moroney, 2017).

Dynamic links é um serviço do *Firebase* que permite a criação de "(...)*links to an app that contain context about what you want the end user to see in the app*(...)"(Moroney, 2017).

Esta ferramenta foi escolhida devido à sua capacidade de fornecer serviços pré configurados de forma gratuita, o que evita o gasto monetário e a alocação de tempo para a configuração de servidores durante o desenvolvimento.

1.2.3 Axios

Para ser possível realizar pedidos a outros serviços externos como *Firebase*, é necessário utilizar uma biblioteca capaz do mesmo. Para isso foi optado por utilizar *Axios*. Esta é "(...)*a promise-based HTTP Client for node.js*(...)"(Axios, 2023) que "(...)*uses the native node.js http module*(...)"(Axios, 2023). Esta disponibiliza um conjunto de métodos para a realização de pedidos a serviços externos, assim como a configuração total dos mesmos.

1.3 *Frontend*

Um dos requisitos do projeto é o desenvolvimento do *frontend* com a utilização de *Flutter*, visto que a empresa no seu trabalho diário já utiliza esta ferramenta. Deste modo, foi fulcral a aprendizagem desta ferramenta e da sua linguagem de programação o dart.

1.3.1 Desenvolvimento cross-platform

O desenvolvimento de aplicações *cross-platform* ou multi-plataforma, consiste no desenvolvimento de uma aplicação para diversas plataformas e este pode ser realizado de diversas formas, mas as principais formas conhecidas são WebView, nativo e outras abordagens.

As *frameworks* nativas são "(...)*the most stable choice for mobile application development*(...)"(Mainkar & Giordano, 2019) e dispõem de um grande comunidade e leque de aplicações desenvolvidas. O que torna estas *frameworks* estáveis é o facto de "(...)*the app in this framework talks directly to the system*(...)"(Mainkar & Giordano, 2019). Todo o desenho no ecrã é realizado através de o que é chamado de *OEM components* que são disponibilizados pela *framework* mas não permitem customização total. A grande desvantagem desta abordagem é o facto de se o objetivo do projeto é o desenvolvimento para *iOS* e *Android*, então "(...)*you need to learn two different languages*(...)"(Mainkar & Giordano, 2019), porque estas são utilizadas para "(...)*write two different apps with the same functionalities*(...)"(Mainkar & Giordano, 2019) o que significa que "(...)*every modification must be duplicated on both platforms*(...)"(Mainkar & Giordano, 2019).

Uma outra abordagem para o desenvolvimento para diversas plataformas através de uma única base de código é o *WebView*. "(...)*Cordova-, Ionic-, PhoneGap-, and WebView-based frameworks in general are good examples of cross-platform frameworks*(...)"(Mainkar & Giordano, 2019), mas o grande problema desta abordagem é a "(...)*lack in performance*(...)"(Mainkar & Giordano, 2019) pois esta é composta por um processo intermédio chamado *WebView* que renderiza código *HTML*, isto significa que "(...)*the app is basically a website*(...)"(Mainkar & Giordano, 2019). Esta abordagem acrescenta também o componente de ponte que realiza o "(...)*switch between JavaScript to the native system*(...)"(Mainkar & Giordano, 2019) para obter acesso aos serviços nativos.

Um concorrente à tecnologia mencionada na secção seguinte(1.3.2) é o *React Native*, este assim como as *frameworks* nativas "(...)*heavily relies on OEM components*(...)"(Mainkar & Giordano, 2019) e "(...)*expands the bridge concept in the WebView systems, and uses it not only for services, but also to build widgets*(...)"(Mainkar & Giordano, 2019), isto leva a grandes problemas em termos de performance devido a que "(...)*a component may be built hundreds of times during an animation, but due to the expanded concept of the bridge, this component may slow down to a great extent*(...)"(Mainkar & Giordano, 2019).

1.3.2 Flutter

Flutter é uma *framework* desenvolvida pela *Google*, de inicio "(...)*was an experiment, as the developers at Google were trying to remove a few compatibility supports from Chrome, to try to make it run smoother*(...)"(Mainkar & Giordano, 2019), por fim, acabaram por descobrir que "(...)*they had something that rendered 20 times faster than Chrome did and saw that it had the potential to be something great*(...)"(Mainkar & Giordano, 2019). Em suma, *Google* desenvolveu "(...)*a layered framework that communicated directly with the CPU and the GPU in order to allow the developer to customize the applications as much as possible*(...)"(Mainkar & Giordano, 2019).

Para o *Flutter* tudo é um *widget*, "(...)*Orientation, layout, opacity,animation... everything is just a widget*(...)"(Mainkar & Giordano, 2019), isto permite que os utilizadores "(...)*choose composition over inheritance, making the construction of an app as simple as building a Lego tower*(...)"(Mainkar & Giordano, 2019). Todos estes *widgets* oficiais estão identificados no catálogo de widgets do *Flutter*. Como tudo no *Flutter* é composto por *widgets* "(...)*the more you learn how to use, create, and compose them,the better and faster you become at using Flutter*(...)"(Mainkar & Giordano, 2019).

A abordagem ao *cross-platform* realizada pelo *Flutter* é baseada em "(...)*AOT (Ahead Of Time) instead of JIT (Just In Time) like the JavaScript solutions*(...)"(Mainkar & Giordano, 2019) mostradas anteriormente. Esta também permite a conversação direta com o cpu sem necessidade de ponte e "(...)*does not rely on the OEM platform*(...)"(Mainkar & Giordano, 2019). Esta faculta que "(...)*custom components to use all the pixels in the screen*(...)"(Mainkar & Giordano, 2019), o que significa que "(...)*the app displays the same on every version of Android and iOS*(...)"(Mainkar & Giordano, 2019). Esta também utiliza "(...)*Platform Channels to use the services*(...)"(Mainkar & Giordano, 2019), o que leva a que "(...)*if you need to use a specific Android or iOS feature, you can do it easily*(...)"(Mainkar & Giordano, 2019).

1.3.3 Dart

Dart é a linguagem de programação utilizada pela *framework Flutter*, esta é "(...)*a general purpose programming language*(...)"(Bracha, 2015) que foi desenhada para ser "(...)*familiar to the vast majority of programmers*(...)"(Bracha, 2015). Esta linguagem é "(...)*purely object-oriented*(...)"o que significa que "(...)*all values a Dart program manipulates at run time are objects*(...)"(Bracha, 2015), até tipos básicos como números e booleanos, esta é também "(...)*class-based, optionally typed*(...)"(Bracha, 2015). Esta é opcionalmente tipada, o que significa que a decisão de utilizar tipagens cai sobre o programador, mas no caso de *Flutter*, na sua versão mais recente é recomendado a utilização de tipagens de variáveis. Por fim, esta "(...)*supports mixin-based inheritance and actor-style concurrency*(...)"(Bracha, 2015).

1.3.4 Links de aplicações

Existem diferentes tipos de *links* sendo que para *mobile* é utilizado os *app links*, *deep links* e os *dynamic links*.

Os *app links* são "(...)*web links that use the HTTP and HTTPS schemes*(...)"(Developers, 2023), estes possuem também um atributo extra chamado *autoVerify*. Este atributo permite a uma aplicação "(...)*to designate itself as the default handler of a given type of link*(...)"(Developers, 2023), isto permite que "(...)*app opens immediately if it's installed*(...)"(Developers, 2023). O grande problema é que estes *links* não permitem o redirecionamento do utilizador para uma parte específica da aplicação e é necessário dispor de um domínio próprio.

Os *deep links* são "(...)*URIs of any scheme that take users directly to a specific part of your app*(...)"(Developers, 2023), o grande problema deste tipo de *links* é que se os utilizadores não dispuserem da aplicação instalada no dispositivo, este irá falhar e não permite a customização de comportamento.

Já os *dynamic links*, desenvolvidos pela *Firebase*, assim como os *deep links* "(...)*if a user opens a Dynamic Link on iOS or Android, they can be taken directly to the linked content in your native app*(...)"(Firebase, 2023), mas para além disto, este permite que "(...)*if a user opens the same Dynamic Link in a desktop browser, they can be taken to the equivalent content on your website*(...)"(Firebase, 2023), ou seja, este permite a customização de comportamento de *links* para diversas situações e em caso do utilizador não dispor da aplicação instalada, este permite que "(...)*the user can be prompted to install it; then, after installation, your app starts and can access the link*(...)"(Firebase, 2023). Visto que este é o comportamento desejado pelo cliente da aplicação, então foi decidido utilizar esta abordagem.

1.3.5 Qualidade de código

A qualidade do código desenvolvido é de extrema importância para possibilitar e facilitar a manutenção da solução desenvolvida, assim como a melhoria da segurança da mesma. Esta permite a perceção de objetivo de código e a estruturação do mesmo de acordo com normas estabelecidas perante a comunidade. A qualidade de código tem como objetivo diminuir a complexidade, pois nem sempre documentação e estruturação é o suficiente para o código ser de qualidade, sendo que, este deverá ser simplificado para ser interpretável. Todos estes pontos poderão ser implementados através do uso de *design patterns*, documentação e testes de código.

1.3.5.1 Design patterns

Os *design patterns* "(...)*names, abstracts, and identifies the key aspects of a common design structure that make it useful for creating a reusable object-oriented design*(...)"(Gamma et al., 2009). Estes identificam "(...)*the participating classes and instances, their roles and collaborations, and the distribution of responsibilities*(...)"(Gamma et al., 2009).

O *MVC* consiste em "(...)*three kinds of objects*(...)"(Gamma et al., 2009), o *Model* que é "(...)*the application object*(...)"(Gamma et al., 2009), a *View* que é a "(...)*screen presentation*(...)"(Gamma et al., 2009) do *Model* e por fim o *Controller* que "(...)*defines the way the user interface reacts to user input*(...)"(Gamma et al., 2009). Antes do *MVC* "(...)*user interface designs tended to lump these objects together*(...)"(Gamma et al., 2009), o *MVC* "(...)*decouples them to increase flexibility and reuse*(...)"(Gamma et al., 2009), através de "(...)*establishing a subscribe/notify protocol between them*(...)"(Gamma et al., 2009).

1.3.5.2 Documentação

Para ser possível manter todo o projeto desenvolvido, foi criada documentação a diferentes níveis, sendo esta desenvolvida a nível de serviços com a explicação do objetivo do serviço e que dados este recebe, como também a nível de código sendo explicado o código desenvolvido em cada *script* existente. Para estes níveis de documentação foram utilizadas diferentes ferramentas, para documentação de serviços, foi utilizada a ferramenta *swagger* e para a documentação de código foi utilizado o *typedoc*.

1.3.5.3 Typedoc

Typedoc é um "(...)*documentation generator for TypeScript*(...)"(TypeDoc, 2023), uma ferramenta "(...)*which reads your TypeScript source files, parses comments contained within them, and generates a static site containing documentation*(...)"(TypeDoc, 2023). Esta utiliza chaves específicas para detetar as informações e criar categorias para melhor organizar toda a documentação. Esta documentação possibilita a interligação entre si mesma, o que permite ao visualizador seguir todo o processo.

1.3.5.4 Swagger

Swagger é uma ferramenta "(...)*built around the OpenAPI Specification*(...)"(SmartBear, 2023) que ajudam com "(...)*design, build, document and consume REST APIs*(...)"(SmartBear, 2023), o *OpenAPI* "(...)*is an API description format for REST APIs*(...)"(SmartBear, 2023) que poderá ser "(...)*written in YAML or JSON*(...)"(SmartBear, 2023). Esta permite gerar documentação a nível de serviços que é acessível a partir do mesmo servidor com indicação de uma rota, o que evita outro servidor para hospedar a documentação. Esta documentação poderá ser gerada a partir de comentários a nível de código ou a partir de um ficheiro em formato *json* ou *yaml*, como mencionado anteriormente. Este ficheiro poderá ser mantido manualmente ou automaticamente. Neste projeto foi decido manter este manualmente em *json*.

1.3.5.5 Testes de código

Aquando o fim do desenvolvimento de cada serviço é necessário testar este para verificar se a funcionalidade encontra-se de acordo com o desejado e/ou existem erros de código. Para realizar estes testes poderão ser utilizadas ferramentas de auxílio ou então poderão ser realizados manualmente. O grande problema dos testes manuais é a exaustividade, uma vez que, são longos e propensos a erros, pelo que, são realizados em menor escala. Para os testes deste projeto foram utilizadas ferramentas de auxílio, sendo as ferramentas escolhidas *mocha* e *chai*.

Mocha é uma *framework* de testes *javascript* que permite teste assíncrono."(...)*Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases*(...)"(Foundation, 2023). Esta ferramenta foi escolhida devido à sua simplicidade e à capacidade de testar código *typescript* para além de *javascript*.

Chai é "(...)*a BDD / TDD assertion library for node and the browser that can be delightfully paired with any javascript testing framework*(...)"(Library, 2023), neste caso esta foi utilizada em conjunto com *Mocha*. A utilização de *BDD* permite a utilização de uma "(...)*expressive language & readable style*(...)"(Library, 2023), já *TDD* é "(...)*more classical feel*(...)"(Library, 2023). Para tornar os testes de código mais interpretáveis foi explorado o estilo de testes *BDD*.

A realização de testes de código foi muito importante, visto que, com esta ferramenta foi possível encontrar erros de lógica de negócio, bem como, erros de código tanto a nível da formulação de respostas como a nível de código.

Bibliografia

Axios. 2023. Getting started | axios docs. https://axios-http.com/docs/intro. [Abril-2023].

Bracha, Gilad. 2015. The dart programming language.

vanden Broucke, Seppe & Bart Baesens. 2018. *Practical web scraping for data science*. Apress. doi:10.1007/978-1-4842-3582-9.

Chuvakin, Dr. Anton A., Kevin J. Schmidt & Christopher Philips. 2012. Logging and log management.

Developers, Android. 2023. Handling android app links. https://developer.android.com/training/app-links#android-app-links. [Abril-2023].

Firebase. 2023. Firebase dynamic links | firebase documentation. https://firebase.google.com/docs/dynamic-links. [Abril-2023].

Foundation, OpenJS. 2023. Mocha - the fun, simple, flexible javascript test framework. https://mochajs.org/. [Abril-2023].

Gamma, Erich, Richard Helm, Ralph Johnson & John Vlissides. 2009. *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.

Hale, Coda. 2023. How to safely store a password. https://codahale.com/how-to-safely-store-a-password/. [Abril-2023].

Halili, Festim & Erenis Ramadani. 2018. Web services: A comparison of soap and rest services. *Modern Applied Science* 12. 175. doi:10.5539/mas.v12n3p175.

Jin, Brenda, Saurabh Sahni & Amir Shevat. 2018. *Designing web apis*. O'Reilly Media, Inc.

Juba, Salahaldin, Andrey Volkov & Achim Vannahme. 2015. *Learning postgresql : create, develop, and manage relational databases in real-world applications using postgresql*. Packt Publishing Ltd.

Library, Chai Assertion. 2023. Chai. https://www.chaijs.com/. [Abril-2023].

Linux. 2023. crontab(5) - linux manual page. https://man7.org/linux/man-pages/man5/crontab.5.html. [Abril-2023].

Mainkar, Prajyot & Salvatore Giordano. 2019. *Google flutter mobile development quick start guide : Get up and running with ios and android mobile app development.* Packt Publishing Ltd.

Masse, Mark. 2011. *Rest api design rulebook.* O'Reilly Media, Inc.

Mead, Andrew. 2018. *Learning node.js development : learn the fundamentals of node.js, and deploy and test node.js applications on the web.* Packt Publishing Ltd.

merencia. 2023. node-cron - npm. https://www.npmjs.com/package/node-cron. [Abril-2023].

Moroney, Laurence. 2017. *The definitive guide to firebase.* Apress. doi:10.1007/978-1-4842-2943-9.

Selenium. 2023. The selenium browser automation project | selenium. https://www.selenium.dev/documentation/. [Abril-2023].

SmartBear. 2023. Swagger specification | documentation | swagger. https://swagger.io/docs/specification/. [Abril-2023].

Snell, James., Doug. Tidwell & Pavel. Kulchenko. 2002. *Programming web services with soap.* O'Reilly & Associates.

TypeDoc. 2023. Overview | typedoc. https://typedoc.org/guides/overview/. [Abril-2023].

Vanderkam, Dan. 2019. Effective Typescript 62 specific ways to improve your typescript.