UNIVERSIDADE BEIRA INTERIOR
Engenharia

Fullstack em web-development numa aplicação da área das telecomunicações

Daniel Nascimento Fernandes

Relatório do projeto de estágio
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor Nuno Gonçalo Coelho Costa Pombo
Supervisor da Empresa: Fábio Rodrigues Nunes Pinheiro

Covilhã, outubro de 2021

Resumo

Este relatório tem como objectivo descrever as actividades realizadas durante o estágio profissional na *Altran*. Esta empresa de consultoria em inovação e engenharia de alta tecnologia foi fundada em 1982 por Alexis Kniazeffe Hubert Martigny, que mais tarde foi fundida com a *Capgemini* em 2019.
O problema proposto a resolver era a implementação de uma aplicação *web* para os agentes de vendas de uma empresa de telecomunicações Belga denominada de *Proximus Group*. Esta aplicação teria que permitir aos colaboradores irem de porta-a-porta vender serviços, como por exemplo fibra.
Foram apresentadas diversas aplicações semelhantes à aplicação aqui desenvolvida, entre elas estão aplicações de *geomarketing* ou que usam a geolocalização no seu funcionamento.
A metodologia *scrum* foi usada para o desenvolvimento deste projecto, usando ferramentas como o *Jira* para a gestão das tarefas do projecto e o *GitLab* para a gestão do código.
Todo o desenvolvimento de novas funcionalidades foi feito num prazo de seis *sprints*, com mais uma Quality Assurance *(QA) Phase* onde foram corrigidos defeitos ou *bugs* encontrados pelos *testers* e o código foi também reformulado.
A arquitectura deste projecto é constituída por três componentes: o *frontend*, o Backend For Frontend *(BFF)* e as Application Programming Interface *(API)*s fornecidas pelo cliente. O *frontend* é um Single-page application *(SPA)* desenvolvido em *AngularJs* que comunica com o *BFF* por pedidos Representational State Transfer *(REST)*. O *BFF* desenvolvido em *NodeTs*, recebe os pedidos do *frontend*, que depois processa, comunica com a base de dados ou com as *API*s e de seguida devolve para o *frontend* a resposta do processamento ou erro em caso de alguma falha.
Nos *sprints* todas as páginas foram desenvolvidas usando ao máximo as componentes Hypertext Markup Language *(HTML)* e as classes Cascading Style Sheets *(CSS)* fornecidas pelo cliente. Só quando não houvesse algo disponível para a construção da página da maneira como estava definida nos *designs* fornecidos é que eram criados novas componentes ou classes. Foi usado o *NxJs* para a criação e gestão de um estado global, criando-se *actions*, *reducers*, *effects* e *selectors*. Cada serviço criado não era chamado directamente pelas componentes da aplicação, mas pelos *effects* que eram desencadeados pelos envios de *actions* das componentes. Todas as componentes e serviços tiveram que ter uma cobertura de 80% pelos testes unitários desenvolvidos com *Jest*.
Em conclusão, foram aplicados alguns dos conhecimentos obtidos em ambiente escolar e adquiridas novas competências. Esta experiência fez despertar um senso de responsabilidade e cautela ao escrever código em ambiente profissional.

Palavras-chave

Geolocalização
geomarketing
Serviços
Web

Abstract

This report aims to describe the activities carried out during the internship at Altran. This high-tech engineering and innovation consulting firm was founded in 1982 by Alexis Kniazeffe Hubert Martigny, which was later merged with Capgemini in 2019.
The proposed problem to be solved is the implementation of a web application for the sales agents of a Belgian telecommunications company called Proximus Group. This application would have to allow employees to go from door to door and sell services, such as fiber.
Several applications similar to the application developed here were presented, among them are geomarketing applications or those that use geolocation in their operation.
The scrum methodology was used in the development of this project, using tools like Jira for managing the project's tasks and GitLab for managing the code.
All the development of new features was done within six sprints, with another QA Phase where defects or bugs found by testers were fixed and the code was also refactored.
The architecture of this project consists of three components: the frontend, the BFF and the APIs provided by the client. The frontend is a SPA built on AngularJs that communicates with the BFF by REST requests.
The BFF was developed in NodeTs, it receives requests from the frontend, which then processes, communicates with the database or APIs and then the frontend receives the processing response or error in case of any failure.
Throughout the sprints all web pages were developed using as frequently as possible the HTML components and CSS classes provided by the client.
Only when there wasn't something available to build the pages defined in the designs provided, new components or classes were created.
NxJs was used to create and manage a global state, creating actions, reducers, effects and selectors.
Each service created was not called directly by the application's components, but by the effects that were triggered by the actions sent by the components.
All components and services had to have 80% coverage in the unit tests developed with Jest.
In conclusion, some of the knowledge obtained in school environment was applied to the project and new skills were acquired. This also awakened a sense of responsability and caution while writting code in a professional environment.

Keywords

Geolocalition
geomarketing

Índice

Lista de Figuras

Lista de Acrónimos

API *Application Programming Interface*

BFF *Backend For Frontend*

CSS *Cascading Style Sheets*

GPS *Global Positioning System*

HTML *Hypertext Markup Language*

HTTP *Hypertext Transfer Protocol*

JSON *JavaScript Object Notation*

QA *Quality Assurance*

REST *Representational State Transfer*

SPA *Single-page application*

UI *User Interface*

Capítulo 1

Introdução

1.1 Caracterização da empresa

A Altran é uma empresa multinacional francesa de consultoria em inovação e engenharia de alta tecnologia fundada em 1982 por Alexis Kniazeff e Hubert Martigny. Esta têm experiência em diversos setores chave como *Automotive, Aeronautics, Space, Defence & Naval, Rail, Infrastructure & Transport, Energy, Industry & Consumer, Life Sciences, Communications, Semiconductor & Electronics, Software & Internet, Finance & Public Sector*. Atualmente, têm mais de 50 mil colaboradores espalhados por mais de 30 países. Em 2019 Capgemini e a Altran anunciaram um plano de fusão baseado em uma oferta de aquisição amigável com o objetivo de criar um líder mundial em *"Intelligent Industry"*. Capgemini no segundo trimestre de 2020 já possuía mais de 98% do capital social da Altran.

1.2 Descrição do Problema

O objetivo deste estágio é implementar uma aplicação web para os colaboradores de uma empresa de telecomunicações Belga. Esta aplicação deve permitir aos colaboradores da empresa irem porta-a-porta vender serviços, como por exemplo fibra ou telemóvel fixo. Para dar resposta a este problema, o *frontend* da aplicação irá ser desenvolvido usando Angular e o *backend* usando NodeJS. Assim, no final pretende-se entregar uma aplicação *fullstack* que comunica em tempo real com os colaboradores e os informa de quais residências contêm possíveis clientes e se ainda não foram contactados.

1.3 Organização do Documento

Este relatório encontra-se estruturado em 5 capítulos, cujo o conteúdo encontra-se dividido da seguinte maneira:

1. O primeiro capítulo – **Introdução** – apresenta o relatório, a empresa, a descrição do problema e a respetiva organização do documento;

2. O segundo capítulo – **Estado da Arte** – apresenta três aplicações semelhantes ao do projeto de estágio;

3. O terceiro capítulo – **Metodologia e Ferramentas** – descreve o processo de desenvolvimento que irá decorrer ao longo do projeto e a ferramenta usada;

4. O quarto capítulo – **método Proposto** – apresenta as diferentes fases de desenvolvimento do projeto;
5. O quinto capítulo – **Arquitetura** – descreve as diferentes componentes do projeto;
6. O sexto capítulo – **Implementação e Resultados** – expõe o trabalho realizado em cada *sprint* e as ferramentas usadas;
7. O sétimo capítulo – **Conclusão** – apresenta as conclusões retiradas do desenvolvimento do projeto e trabalho fututo.

Capítulo 2

Estado da Arte

Este capítulo pretende apresentar aplicações semelhantes ao problema que irá ser tratado no projeto de estágio. As secções 2.1, 2.2, 2.3 2.4, 2.5, 2.6 e 2.7 apresentam sete aplicações que usam a geolocalização no seu funcionamento.

2.1 *Uber*

A *Uber* é uma aplicação móvel que liga os utilizadores a parceiros licenciados. Onde a *Uber* existe, os utilizadores podem requisitar uma viagem através de um simples clique. Quando o utilizador requisita, o pedido é enviado para o motorista mais próximo e uma vez aceite, a aplicação indicará aproximadamente quanto tempo o motorista levará até chegar. Quando se faz o pedido o utilizador tem acesso ao nome do motorista, marca, modelo e matricula do carro de forma a garantir que o utilizador consiga facilmente e de forma segura encontrar-se com o motorista. Esta aplicação tem também um sistema de avaliação, que funciona nos dois sentidos, tanto para o motorista como para o utilizador, assegurando a qualidade das viagens e a experiência para todos [Ube].

2.2 *MyGeoTab*

MyGeoTab é uma aplicação móvel para gestão de frotas, com análises avançadas e integração em tempo real. Esta aplicação é uma ferramenta conveniente para gerentes de frotas ou donos de empresas que trabalham em trânsito. Tem um *design* responsivo que se ajusta a qualquer monitor e seis diferentes línguas disponíveis. O mapa da aplicação permite ver a localização dos veículos em tempo real, assim como o histórico das viagens e o caminho que cada um fez [Geo].
As principais funcionalidades são: relatórios avançados de frota e de motoristas, criação de regras personalizadas como limites de velocidade ou tempo de pausa, rastrear veículos por *Global Positioning System* (GPS) em tempo real em um mapa, gestão do comportamento do motorista, monitorizar a integridade e manutenção do motor, optimização de rota, plataforma aberta permitindo integridade e personalização de dados adicionais [GPS].

2.3 *Strava*

O *website Strava* de partilha de actividades permite que os utilizadores rastreiem e carreguem os seus passeios e corridas usando dados de GPS, quer da aplicação *Strava* como

de qualquer outro computador de bicicleta GPS. A aplicação guarda os dados como distância, tempo, elevação, calorias e velocidade. O *Strava* também usa o GPS para acompanhar com precisão a rota dos utilizadores, seja nas estradas ou nas trilhas.
Esta aplicação é como se fosse uma rede social para atletas, pois os utilizadores podem registar a sua actividade que por sua vez vai para o *feed*, onde amigos e seguidores podem partilhar e comentar. Esta aplicação também permite que o utilizador ligue o *beacon* para partilhar a sua localização atual em tempo real para quem quiser, permitindo assim que esta apareça no mapa, permitindo assim que no caso de uma emergência o utilizador possa ser encontrado facilmente. A localização do utilizador está também acompanhada de horas de começo de atividade, tempo de atividade e percentagem de bateria do dispositivo.
Este *software* para gestão de frotas, ajuda a perceber onde estão as viaturas, qual o seu estado, quem está responsável por as conduzir e cuidar delas [Str].

2.4 *Google Maps*

O *Google maps* é um serviço *web* que providencia informação detalhada sobre regiões geográficas do mundo inteiro. Além dos mapas de estradas, o *Google Maps* oferece vistas aéreas e de satélites de muitos lugares e se pesquisarmos por ”café”, o *Google* irá usar a localização do utilizador dando uma lista dos cafés mais próximos e mostrando no mapa todos os cafés existentes [Tec]. O *Google Maps* também oferece diversos serviços como:

- Criador de rotas oferecendo direcções para condutores, ciclistas, peões e utilizadores de transportes públicos;
- Uma *API* para outras aplicações embutirem o *Google Maps* nas suas aplicações;
- *Google Street View* permite aos utilizadores visualizar e navegar por imagens panorâmicas ao nível das ruas de várias cidades do mundo;
- Imagens da Lua, Marte e do céu.

2.5 *Zubie*

Este *software* para gestão de frotas, ajuda a perceber onde estão as viaturas, qual o seu estado, quem está responsável por as conduzir e cuidar delas. O dispositivo *Zubie* é ligado à porta de diagnostico do veiculo, conseguindo assim rastrear a velocidade média, a localização, acelerações súbitas, travagens a fundo, tempo parado, entre outras informações. *Zubie* usa a API do *Google Maps* para o seu mapa a tempo real com os veículos. No mapa consegue-se ver a localização dos veículos e cada um pode ter cores distintas, dependendo do estado em que esse se encontra. A aplicação também fornece *dashboards* com dados e estatísticas dos veículos, sabendo assim quem anda em excesso de velocidade, a travar a fundo, em pausas constantes, entre outras informações [Zub].

2.6 *Mobalo*

Mobalo é um provedor de serviços para publicidade móvel baseada na localização. Conecta os dados de localização com os *mobile banners* de forma a segmentar as publicidades. As aplicações são mostradas em aplicações que os utilizadores estão já a usar. Com o *Mobalo* os anunciantes podem disseminar as suas mensagens para clientes relevantes, baseando-se na localização e na situação dos clientes. Para haver esta selecção do grupo-alvo são necessários dados de localização, pesquisa do mercado e dos dados do próprio cliente [Mob].

2.7 *Foursquare City Guide*

O *Fourquare* é uma aplicação móvel usada para encontrar novos locais perto do utilizador, com base nas pesquisas ou recomendações personalizadas dadas pela aplicação. Esta aplicação usa a localização do dispositivo e encontra locais para recomendar ao utilizador. Quanto mais a aplicação é usada, mais esta aprende o tipo de locais o utilizador gosta, recomendando cada vez mais locais do género. As recomendações são armazenadas através dos utilizadores, que já estiveram no local e deixaram avaliações positivas e comentários. De maneira semelhante a outras aplicações de descoberta de locais através da localização, como o *Yelp*, as recomendações do *Foursquare City Guide* são trazidas a partir da sua comunidade [Lif].

Capítulo 3

Metodologia e Ferramentas

Neste capitulo irá ser apresentada a metodologia usada no desenvolvimento do projeto na sub-seccção 3.1, bem como as ferramentas necessárias para gerir as tarefas e o código do *frontend* como o *backend* na sub-secção 3.2.

3.1 Metodologia

O projeto foi desenvolvido usando a metodologia *agile Scrum*. O desenvolvimento foi feito em *sprints* de duração de duas semanas, com reuniões recorrentes como as *daily meetings*, *groomings* e *demos*. A figura 3.1 apresenta os procedimentos que são realizados em cada *sprint*.

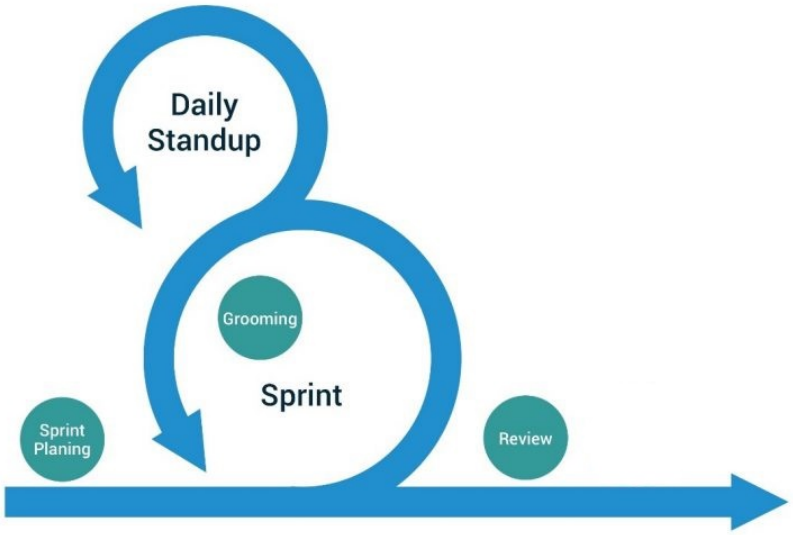

Fig. 3.1: Diagrama de *Scrum*

No inicio de todas as *sprints* houve uma *sprint planning* onde se planeou a *sprint*, isto é, que tarefas iriam ser feitas durante as duas semanas seguintes. Nesta reunião foram escolhidas diversas tarefas das disponíveis, tendo em conta o grau de prioridade de cada uma e os pontos associados a cada uma destas. Cada tarefa tem uma pontuação associada de acordo com o grau de dificuldade e tempo necessário para a realização dessa. Assim,

foram escolhidas tarefas até o somatório dos pontos chegar à velocidade da equipa, sendo a velocidade da equipa a quantidade de pontos que esta consegue desenvolver por *sprint*.

Todos os dias no inicio da manhã houve uma *daily meeting* entre os *developers* e o *scrum master*, onde todos disseram o que fizeram no dia anterior, o que farão durante esse dia e se tiveram algum problema. Esta reunião não deve demorar mais de 15 minutos, caso alguém tenha algum problema e haja algum colega que possa ajudar, estes devem continuar a discussão sozinhos depois da reunião para não prender os restantes. Se o problema não for apenas dificuldades no desenvolvimento, como por exemplo falta de ferramentas, quem deve dar resposta é o *scrum master*.

As *groomings* ocorram uma vez por semana entre a equipa de desenvolvimento e alguns elementos da empresa do cliente. Nesta reunião foram discutidas e avaliadas as tarefas que poderiam vir a ser entregues no *sprint* seguinte. Para cada tarefa foram retiradas todas as dúvidas para o desenvolvimento da mesma e no final foram pontuadas as tarefas de acordo com o grau de dificuldade e tempo necessário para as desenvolver.

No final de cada *sprint* ocorreu uma *demo*, em que foi apresentado todo o trabalho feito nas últimas duas semanas a alguns elementos do cliente. Em cada reunião foi apresentado um *powerpoint* com as informações do estado atual do projeto e quais as tarefas que a equipa realizou. No final, um dos membros da equipa mostrou visualmente as tarefas concluídas, correndo a aplicação e mostrando que cada ponto associado às mesmas.

3.2 Ferramentas

3.2.1 *Jira*

Durante o desenvolvimento do projeto foi usado o *Jira*, sendo esta uma ferramenta de apoio à gestão de projetos, mantendo o estado das tarefas [Atl]. Em cada *sprint* foram puxadas diversas tarefas para o desenvolvimento tendo em conta o grau de prioridade das tarefas e a velocidade da equipa. Cada tarefa pode ser classificada como um *bug*, uma *task* ou uma *user story*. Ao iniciar um *sprint* cada *developer* pegou numa tarefa para desenvolver, associando no *Jira* essa tarefa a ele mesmo e colocou-a no estado *in progress*. Neste projeto os únicos estados que podiam ser atribuídos a uma tarefa eram *in progress*, *code review* e *closed*. Para manter o estado das tarefas de forma mais clara, foram adicionadas sub-tarefas com os outros estados, sendo estas associadas a quem as iria fazer. Cada uma destas tarefas tinha, no máximo, 6 sub-tarefas: *DeskCheck*, *Analysis*, *Code*, *Unit Testing*, *Code Review* e *Final Review*. As primeiras duas sub-tarefas destinam-se à análise da tarefa e estudo dos seus requisitos. A sub-tarefa *code* consiste no desenvolvimento de código enquanto que a de *Unit Testing* visa a implementação de *unit tests* para o código desenvolvido. O *Code Review* e *Final Review* foi feito por uma pessoa diferente de quem realizou o *Code* e *Unit Testing*, onde o código foi avaliado e a aplicação testada manualmente depois do *deployment*, respetivamente.

3.2.2 *GitLab*

Ao longo do projeto, o *GitLab* foi usado para controlar o progresso do código. Nesta ferramenta existiam dois repositórios, um para o *frontend* e outro para o *backend*. Sempre que uma tarefa do *Jira* iniciou o desenvolvimento foi criada uma nova *branch* para o código que iria ser desenvolvido. Ao concluir uma tarefa o *developer* teria que ir ao *GitLab* e abrir um *merge request* que iria ser avaliado por um outro *developer*. Só depois da aprovação, por parte do *developer* responsável, cada *branch* foi *merged* com a *branch* principal. Caso o avaliador encontrasse algum problema, este poderia fazer um comentário no *GitLab* para que o responsável o corrigisse.

Capítulo 4

Método Proposto

Este capítulo apresenta as diferentes fases de desenvolvimento do projeto, estando estas divididas em *sprints* e *QA Phase*. Cada uma destas fases está separada em subsecções em que para cada uma destas é apresentado o que irá ser feito.

4.1 Planificação do Trabalho

O projeto irá ter uma duração de seis *sprints* seguidos de uma *QA Phase* como demonstra a figura 4.1.

4.1.1 *sprint* 1

No primeiro *sprint* a equipa de desenvolvimento irá se focar mais na implementação da *User Interface* (UI) e no serviço de *toaster*. A parte da UI não necessitará de ter lógica nenhuma, apenas fazer aparecer o mapa, menu, página de *landing* e o *toaster*.

4.1.2 *sprint* 2

No segundo *sprint* será implementado alguma lógica à UI criada no *sprint* anterior. Esta lógica irá incluir pedidos à API do cliente, mudanças no UI de acordo com os dados recebidos da API e o uso do serviço do *toaster*. Nesta *sprint* também haverá implementação de mais UI para apresentar os dados que virão do *backend*.

4.1.3 *sprint* 3

No terceiro *sprint* serão feitos os testes unitários ao código implementado até ao momento. Neste *sprint* será implementado mais lógica aos menus implementados nos *sprints* anteriores e também irá ser feito um *Sign-On*.

4.1.4 *sprints* 4, 5 e 6

Nestes últimos *sprints* será para terminar o *flow* da aplicação e lógica, desenvolver mais o *backend* e integrar o *Couch base*.

4.1.5 *QA Phase*

Esta última fase será usada para corrigir *bugs* que sejam encontrados pela equipa de testes, que deverá ter uma duração de aproximadamente um mês.

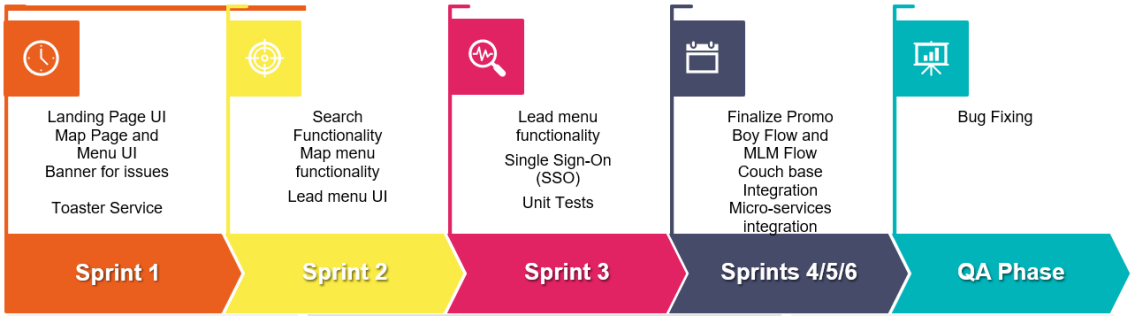

Fig. 4.1: Diagrama de *sprints*

Capítulo 5

Arquitectura

Este capítulo apresenta a arquitectura do projeto, detalhando a composição de cada componente e como estás se relacionam entre si.

5.1 Arquitectura das Aplicações

A figura 5.1 representa a arquitectura das aplicações que foram desenvolvidas. A arquitectura é composta por 3 elementos, o *frontend* do lado esquerdo, o *BFF* no meio e as *API*s do cliente que já estão desenvolvidas do lado direito.

O *frontend* é uma *SPA* desenvolvida em *AngularJs* em que as componentes têm acesso a um estado criado e mantido pelo *RxJs*. Os dados que se encontram no estado são sincronizados com os que estão na base de dados do *CouchDb* através de um *listener*, que deteta quando há qualquer modificação. Sendo esta uma *SPA*, a página é sempre a mesma ao longo de todo o fluxo, apenas mudam as componentes a serem renderizadas. O *frontend* é composto por componentes, serviços, *models* e todos os elementos necessários para alterar e gerir o estado global. Cada componente está acompanhada de um ficheiro *HTML*, um *CSS* e um de testes unitários. Os serviços também estão acompanhados do seu próprio ficheiro de testes. O *frontend* comunica com o *BFF* usando os serviços que realizam pedidos *REST* para pedir ou modificar dados. Em alguns casos, o *frontend* usa os serviços para comunicar diretamente com as *API*s do cliente, quando não há necessidade de processamento dos dados retornados por estas ou enviados para estas.

O *BFF* é uma *API REST* que serve de intermediário entre o *frontend* e as *API*s fornecidas pelo *cliente*. O *BFF* recebe os pedidos do *frontend* e dependendo do que seja necessário fazer este comunica com as *API*s ou com a base de dados. Os dados modificados na base de dados pelo *BFF* são depois sincronizados com os do *frontend*. Desta forma, quando dois ou mais utilizadores acedem aos mesmos dados no *frontend*, se um fizer uma alteração esta irá-se refletir no *frontend* dos outros devido à constante sincronização. O *BFF* é composto por *controllers* que recebem os pedidos, por serviços que processam dados e comunicam com as *API*s do cliente e por *models* que definem as interfaces dos diferentes objetos usados. As alterações ou pesquisas feitas ao *couchDb* são realizadas através de serviços que usam pedidos *REST*, enquanto que os pedidos às *API*s do cliente são feitas através de pedidos *REST* ou usando o *GraphQL*. Cada ficheiro de serviço é sempre acompanhado do seu ficheiro de testes como no *frontend*.

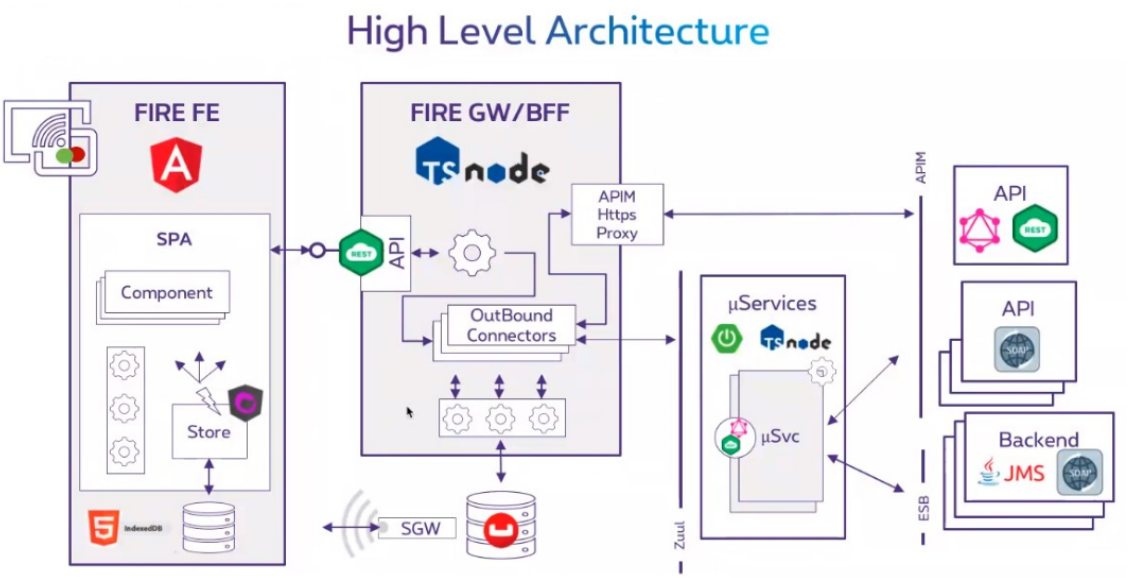

Fig. 5.1: Arquitectura das Aplicações, disponibilizada pela *Proximus Group*

Capítulo 6

Implementação e Resultados

Neste capitulo são apresentadas todas as tecnologias necessárias para desenvolver o *frontend* e o *backend* na sub-secção 6.1. Da sub-secção 6.2 à 6.7 são descritas as *sprints* de desenvolvimento, explicando o que foi realizado em cada uma destas. por fim, na sub-secção 6.8 são descritos os defeitos corrigidos nas últimas semanas do projeto.

6.1 Tecnologias

O desenvolvimento do *Frontend* foi realizado usando diversas tecnologias como: *Angular*, *NgRx*, *Jest* e *Leaflet*. O *angular* sendo uma *framework* baseada em *TypeScript* foi usada para construir a aplicação *web*, usando o *HTML* e *CSS* para conceber o *UI* e o *Typescript* para criar as componentes e toda a lógica de negócio. O *NgRx* possibilitou a criação e gestão de um estado global, onde todas as componentes podem aceder caso necessitem algum dado que não possa ser passado pela componente ”pai”. O mapa e as suas funcionalidades foram adicionadas a partir do *Leaflet*, uma biblioteca *Javascript Open-source* para mapas interactivos. Todas as componente e serviços criados tiveram que ser testados, para tal foi usado o *Jest*, que é uma *framework* de testes de *Javacript*.
O *Backend* foi desenvolvido usando *Nodejs*, *Jest*, *GraphQL* e *Axios Express*. A construção deste foi feita a partir do *NodeTs*, que é um ambiente de execução de *Javascript* fora de um navegador *web*. As rotas e pedidos *REST* foram criados através de um cliente Hypertext Transfer Protocol *(HTTP)* para navegador e *NodeJs* denominado *Axios Express*, enquanto que os pedidos às *API*s foram criados através do *GraphQL*, que é uma linguagem de *consulta* de *API*s, que permite aos clientes pedirem exactamente o que precisam. Os testes unitários feitos ao código de lógica de negócio e serviços foram também feitos usando o *Jest*.

6.2 Resultados da Primeira *Sprint*

Na primeira *sprint* foram criadas as *UI*s de várias páginas, um *Banner* para qualquer tipo de avisos e um serviço de *toaster*. Todo o *design* necessário para construir as páginas, foi feito usando componentes de *HTML* e *CSS* já existentes fornecidos pelo cliente *Proximus Group*, só em casos em que não houvesse *CSS* que fizesse o que era necessário é que se criava novas classes. Todos os *UI*s foram seguidos à risca de acordo com os *designs* fornecidos pelo cliente no *Adobe XD*.
Uma das páginas criadas foi a primeira página do fluxo, uma página constituída por 4 *input fields* e um botão. Esta página é destinada à introdução do nome da cidade, bloco,

zona de fibra, nome da rua e o botão que confirma a pesquisa e redireciona para a próxima página do fluxo. Neste momento foi só adicionado uma expressão regular a cada *input field*, para que se pudesse mostrar e testar as bordas dos *input fields* a mudar de cor de acordo com as regras, sendo preta para quando ainda não se introduziu nada, vermelho para demonstrar erro e verde para mostrar sucesso. Cada *input field* é também acompanhado de uma *label* que aparece no caso de erro, dando a informação de como este deve ser preenchido.

A outra página criada nesta *sprint* é constituída por um menu em lista do lado esquerdo e um mapa do lado direito. O mapa neste momento não tem qualquer funcionalidade, apenas as fornecidas por omissão pela biblioteca *Leaflet*, como *zoom in*, *zoom out* e movimentação. O menu é constituído por um botão de filtro do tipo *accordion*, que ao ser clicado expande verticalmente mostrando diversas *checkboxes* com *labels*, que servirá para filtrar a lista que se encontra por baixo. Esta lista serve para mostrar os edifícios e clientes resultantes da pesquisa da primeira página do fluxo. Esta lista é constituída por *accordions*, onde cada um destes representa um edifício, e mostra o nome da rua, quantos clientes ou possíveis clientes da *Proximus Group* residem neste e se existe fibra ou não disponível. Ao se clicar num *accordion* este expande também verticalmente, mostrando agora o nome, o número do piso, o número do apartamento e o número de quarteirão de cada *Lead* caso estes dados pertençam a um cliente ou organização, pois se for um possível cliente estes dados nao existem e então no ecrã aparece a palavra "*unknown*" a substituir. Uma *Lead* pode ser um cliente, um possível cliente ou uma organização. Para se mostrar e testar esta lista foi criado um *mock* dos dados seguindo a estrutura fornecida. A figura 6.1 mostra a interface do objecto *Lead*, em que este é composto por 8 chaves. Neste objecto existe "*Person*" ou "*organisation*", nunca ambos ao mesmo tempo e caso um exista também existirá "*customer*", pois este contém os serviços que o *Lead* possui. Os dados usados para mostrar informação do edifício e morada estão contidos no "*geographicAddress*" e no "*fulladdress*", respectivamente. O objecto "*lead*" contém a informação em que o *Lead* se encontra, por exemplo "não voltar a contactar" ou "não se encontra em casa".

```
export interface Lead {
    person: Person;
    organisation: Organisation;
    customer?: Customer;
    Qos: Qos;
    lead: LeadDetails;
    geographicAddress: GeographicAddress;
    fulladdress?: Fulladdress | null;
    id?: number | string | null;
  }
```

Fig. 6.1: Interface do Objecto *Cliente*

O ultimo elemento de *UI* criado foi um *banner*, que serve para mostrar avisos de erro, sucesso ou informação. Este *banner* é constituído por um título, uma mensagem e/ou por um botão que redireciona para a página inicial. Para que esta componente ficasse genérica para aceitar os 3 tipos, foi feita de maneira a que recebesse 4 valores, sendo 3 dos valores *strings*, um para preencher o título, outro a mensagem e o ultimo para identificar qual o tipo do *banner* que se pretende utilizar. O outro valor é um *boolean* em que se for *true* o botão de redirecionamento é mostrado.
Para se mostrar o trabalho feito nesta *sprint* foi necessário definir as rotas para cada uma destas páginas, pois só desta maneira se conseguia aceder a estas e no caso do *banner* foi criado uma página muito simples com os 3 tipos de *banner*.

6.3 Resultados da Segunda *Sprint*

Nesta *sprint* foram criados dois serviços, em que cada um destes realiza um pedido *GET* à *API* fornecida pela *Proximus Group* e que no final retorna os valores recebidos da *API*. O primeiro serviço envia o que utilizador introduziu no campo da cidade, recebendo todas as cidades em que o nome seja parecido ou igual ao que utilizador introduziu. O segundo recebe o nome da cidade que o utilizador escolheu do retorno do primeiro pedido e o valor que este introduziu no campo da rua, retornando também as ruas que tenham o nome parecido ou igual ao introduzido e que estejam na cidade escolhida.
Foi utilizado o *NgRx* para a criação de *actions*, *effects*, *reducers*, *selectors* e um estado global na aplicação. Para melhor compreensão foi adicionado a figura 6.2, em que esta mostra o esquema destes cinco elementos. A componente envia uma *action* em que pode ou não desencadear um *effect* ou fazer com que o *reducer* para essa *action* faça algo, como por exemplo guardar dados no estado global. O *effect* pode fazer qualquer coisa, como por

exemplo chamar serviços ou até mesmo enviar outras *actions*. Neste sentido inicialmente foi criado um estado que guarda um *array* com as cidades, um com as ruas e outro com os *Leads*. Foram também criadas duas *actions* que desencadeiam *effects*, sendo uma delas para a busca das cidades e outra para as das ruas. Cada uma destas *actions* tem o seu próprio *effect* em que estes chamam o serviços falados anteriormente, que ao receberem os dados chamam outras *actions*. Estas outras *actions* também tiveram que ser criadas, duas para cada uma das *actions* faladas anteriormente. Estas estão definidas no *reducer*, em que recebem os dados do serviço e os guardam no estado global. Caso ocorra um erro no pedido do serviço, este é guardado também no estado global usando uma das *actions* com *reducer*.

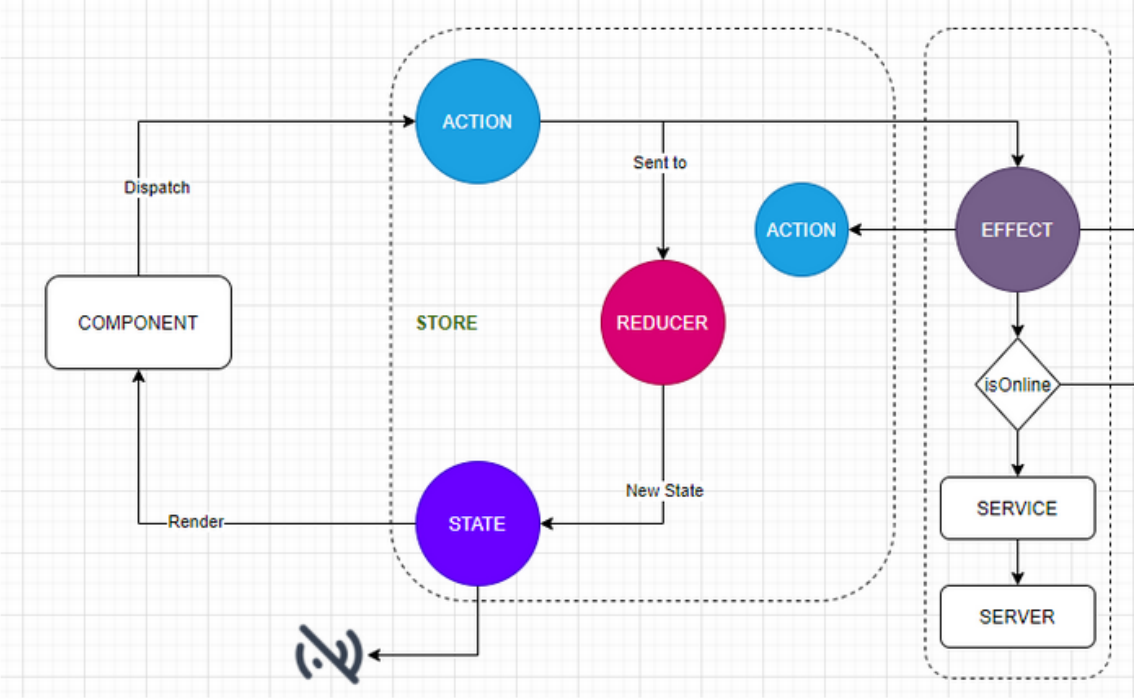

Fig. 6.2: Diagrama de *NgRx*

Os elementos da figura 6.2 que pertencem ao *NgRx* são:

- ***State*** – representa um objecto imutável que contém o estado da aplicação;
- ***Action*** – representa eventos na aplicação. Podem desencadear efeitos colaterais nos *effects*. A *interface* contém uma propriedade denominada de *type* que identifica a *action*. *Actions* pode conter *metadata* opcional;
- ***Reducer*** – são funcções que realizam transições de um estado para outro com base na última *action* lançada;
- ***Selector*** – retornam um parte do estado da aplicação;
- ***Effect*** – são efeitos colaterais. Os *effects* permitem ”ouvir” por *actions* em particular e fazer algo sempre que isso acontece.

Na página inicia do fluxo da aplicação foram adicionados toda a lógica de guardar os dados introduzidos nos *input fields* em variáveis da componente. Foi adicionado também a funcionalidade de quando o utilizador introduz no campo da cidade três caracteres a componente envia a *action* falada anteriormente, que desencadeia o *effect* que usa serviço com o valor introduzido para preencher o *dropdown* do *input field* para o utilizador escolher. Quando o campo cidade está preenchido, o campo da rua fica disponível para ser também preenchido, tendo também a mesma lógica do campo da cidade, usando apenas as suas próprias *actions*, *effects* e *reducers*. O botão da página está bloqueado até que o campo cidade e rua estejam preenchidos ou então o campo da cidade, bloco e zona de fibra. Este botão quando disponível apenas redireciona para a página do mapa por enquanto.
Na página do mapa foi criado uma componente que mostra os dados do *Lead* escolhido da lista, mostrando esta informação em blocos. No primeiro bloco o nome e a localização, no segundo os serviços que possui e no terceiro os serviços disponíveis para ele. Foi também adicionados diversos botões, como um botão para voltar atrás, um botão para quando o *Lead* está interessado em algum dos serviços, um para informar que ele não se encontra em casa e outro para quando não está interessado em nenhum. Quando o utilizador clica num *Lead* o menu desaparece, o mapa passa a ocupar 50% da área de cima do ecrã e esta componente aparece ocupando os 50% de baixo.

6.4 Resultados da Terceira *Sprint*

Nesta *sprint* removeu-se o *mock* dos dados, criando-se a mesma lógica quando foi com a pesquisa das cidades ou das ruas. O botão da página inicial quando é clicado agora envia uma *action* e depois é que redireciona. A *action* desencadeia o *effect* e este constrói o *payload* com os dados introduzidos e manda para o serviço. O serviço criado faz o pedido à rota do *BFF* que foi implementada pelo resto da equipa, que devolve os *Lead*'s num *array* pertencentes à localização introduzida pelo utilizador nos campos da página inicial. Quando os dados ou erro chegam ao *effect*, este envia uma *action* de acordo com o que recebeu para que estes sejam guardados. Caso tenha sido um erro agora o serviço do *toaster* implementados pelo resto da equipa faz com que o *banner* de erro apareça.
Foi criado no estado global um chave para guardar o *Lead* seleccionado da lista, podendo assim ser acedido de qualquer das páginas da aplicação. Para isso foi necessário a criação de uma *action* e de um *reducer* para guardar este no estado e mais uma *action* e um *reducer* para alterar os valores dentro deste *Lead*. Para que se pudesse aceder ao *Lead* em cada componente da aplicação foi criado um *selector* que apenas devolve a chave que contém o *Lead* no estado.
O *Jest* foi também configurado e foram feitos todos os testes unitários a todas as componentes e serviços feitos pela equipa. Cada componente ou serviço tem que ter uma cobertura pelos testes de pelo menos de 80% do código. Para cada função, era testado se ela fazia o que realmente era suposto fazer e caso chama-se qualquer outra função era necessário fazer um *mock* da função ou da resposta desta, para que não fosse executada.
A partir deste *sprint*, sempre que houvesse uma nova componente ou serviço a ser desen-

volvido os testes unitários eram logo feitos, não sendo aceites *merge requests* sem estes.
Foi adicionado também a funcionalidade de tradução na aplicação. Para isto foi preciso usar uma biblioteca do *AngularJs* e serem criados 3 JavaScript Object Notation *(JSON)*s, um para a língua inglesa, outro para a francesa e o outro para a holandesa. Para a tradução funcionar todos os *JSON*s têm as mesmas chaves apenas o que muda são os valores. A língua é seleccionada num *dropdown* no *topbar* que já estava criado no inicio do projecto. Para cada *string* estática nos ficheiros *HTML*s for colocado um *pipe* em que do lado esquerdo leva a chave do ficheiro das traduções e do lado direito a palavra ”translate” para que o serviço das traduções substitua o *pipe* pelo valor da chave da língua seleccionada.

6.5 Resultados da Quarta *Sprint*

Foi adicionado funcionalidade ao botão de interesse criado na segunda *sprint*. Este botão redireciona para uma página de pesquisa de *Leads* caso a pessoa seleccionada seja um possível cliente ou então se for já um cliente ou organização redireciona para uma página onde se altera ou adiciona informações sobre o este.
A página que altera os dados do *Lead* é composta por um botão para voltar atrás, botão de confirmação, duas *checkboxes*, uma para escolher a língua e outra para escolher o género e 8 *input fields*, em que todos podem ter os seus valores alterados excepto o que contém o nome da rua. Os outros campos são para o primeiro nome, último nome, email, número de telefone, piso, apartamento e o número do quarteirão. Quando os dados estão correctamente preenchidos o botão de confirmação fica disponível e redireciona para a próxima página do fluxo, que neste momento é apenas uma página em branco. De acordo com o *Lead* escolhido os dados são colocados nos *input fields* caso existam. Estes dados estão guardados no estado global, então foi usado o *selector* que vai apenas devolver o objecto do *Lead* que foi seleccionado numa das páginas anteriores para esta componente para que estes dados possam ser colocados nos *input fields*.
Outra página desenvolvida foi uma página de pesquisa de *Lead*s, que aparece quando o utilizador clica em ”interessado” quando seleccionou um possível cliente. Esta página é composta por dois *input fields* para colocar o último nome e o primeiro nome de um cliente que já possa existir para que se possa ligar à mesma ”party”. Isto acontece em casos em que há mais que uma pessoa na mesma casa e querem estar todos no mesmo pacote. Caso não se queira ou não haja ninguém para se fazer a ligação, foi adiciona uma *checkbox*. No fim da página foi colocado um botão de confirmação que dependendo se a *checkbox* foi seleccionada ou não, redireciona para duas páginas distintas. Se a *checkbox* foi seleccionada o botão redireciona para a página onde se altera a informação do cliente, mas como não houve nenhum cliente existente seleccionado, a página irá ter todos os campos editáveis vazios para que o utilizador possa preencher e criar assim um novo objeto ”*person*” e seja adicionado ao *Lead*. Caso o cliente tenho preenchido os *input fields* o botão redireciona para outra página que neste momento se encontrava em branco. Para a pesquisa funcionar, foi feito um *selector* que trás todos os *Lead*’s guardados no estado global e a componente trata de filtrar, ficando só disponíveis para a pesquisa os clientes

que pertencem ao mesmo edifício.

6.6 Resultados da Quinta *Sprint*

Nesta *Sprint* foi desenvolvido a página para onde redireciona o botão da página de pesquisa de *Leads* quando os *input fields* são preenchidos. Esta página é constituída por *radio buttons*, em que cada elemento mostra o nome, a rua, o apartamento, o piso e o quarteirão dos *Leads*s do edifício em que o nomes inseridos são iguais. A página tem também um botão de confirmar que quando é clicado redireciona para a página de edição do *Lead* e para este possível cliente o objecto "*person*" do *Lead* seleccionado no *radio button* é copiado para este.

Foi também desenvolvido a última página do fluxo que consiste em quatro botões, um para voltar atrás e três que abrem uma nova *tab* do navegador com uma página externas pertencentes ao *Proximus Group*, passando os dados do cliente seleccionado em *query string*.

Foi criado um novo serviço em que recebe o objecto cliente e que de seguida é feito um *POST* a uma *rota* do *BFF* criada por outros elementos da equipa, que tem como objectivo alterar o estado do *Lead* para "interessado", "não em casa", "não interessado" ou "não contactar outra vez". Foi também criado uma *action* e um *effect*, usando a mesma lógica inicial, em que se envia a *action* para desencadear o *effect* que chama o serviço.

Na página que mostra a informação do cliente, agora cada botão altera o estado do *Lead*, de seguida chamada a *action* passando o objecto deste e de seguida redireciona para a página que contém o mapa e a lista de *Lead*s. Caso ocorra algum erro no pedido, o *banner* de erro é mostrado com o botão de redirecionamento à página inicial disponível.

6.7 Resultados da Sexta *Sprint*

Foi criado um serviço para actualização dos dados do cliente em que recebe o objecto do cliente e faz um pedido *POST* à rota do *BFF* também feita por outros elementos da equipa. Juntamente com este serviço foi também implementado a *action* e o *effect* usando a mesma lógica destes elementos anteriormente falados.

Na página de edição dos dados do cliente, agora os dados são alterados no objecto usando a *action* e *reducer* criados na *sprint* 3, e quando o botão de confirmar é clicado, a *action* falada em cima é lançada com o objecto do *Lead* como parâmetro.

Na página que continha três botões de redirecionamento criada na *sprint* anterior, foi adicionado a chamada ao *selector* que devolve o *Lead* seleccionado e uma função que constrói a *query string* com os dados requeridos pelas páginas externas. Os dados passados irão ser usados para preencher um formulário de outra aplicação do *Proximus Group*. Cada botão depois de abrir uma nova *tab* também redireciona para a página que contém o mapa e a lista de *Leads*.

6.8 Resultados da *QA Phase*

Nesta fase foram corrigidos defeitos encontrados pelos *testers* na aplicação, desde funcionalidades com algum tipo de falha a alguma imperfeição no *UI*.
Ao longo desta fase foram alteradas várias traduções de várias *strings*. Para isto, foi apenas alterado o valor da chave usada no ficheiro da língua que se queria alterar e a tradução ficava imediatamente corrigida.
Ao nível de *UI* foi também alterado o tamanho de certas partes de componentes para melhor visualização, para isto foi só necessário alterar ou criar classes *CSS* simples que modificassem o tamanho a propriedade que queremos que aumente.

Capítulo 7

Conclusão

Este capítulo descreve as conclusões tiradas da aplicação dos conhecimentos previamente obtidos em ambiente escolar, dos novos conhecimentos retirados ao longo do desenvolvimento do projecto e do cuidado que se deve ter quando se desenvolve em ambiente profissional.

7.1 Conclusões Principais

Durante o período de estágio foi possível aplicar conhecimentos aprendidos nas cadeiras de Programação Orientada a Objectos, Programação *Web* e Interação Humana com o Computador que muito auxiliaram ao longo de todo o desenvolvimento.
Com este estágio conseguiu-se aprender e experienciar como é o desenvolvimento de *software* em ambiente de empresa e as suas formalidades, bem como usar as ferramentas de gestão de código e de tarefas de projecto.
Permitiu também melhorar o conhecimento de algumas tecnologias as quais já tinham sido usadas em âmbito escolar, como por exemplo, o *HTML* e *CSS*, mas também aprender e aprofundar novas tecnologias como o *AngularJs*, *Jest* e o *RxJs*.
Dado que esta aplicação foi feita fora do ambiente académico, permitiu despertar atenção para a maneira como o código é escrito e estruturado, para evitar que no futuro fossem abertos defeitos ou *bugs*. Pois isto aumenta a quantidade de horas despendidas a trabalhar no projecto, podendo levar atrasos indesejáveis na implantação do projecto em ambiente de produção.

7.2 Trabalho Futuro

Como trabalho futuro propõe-se fazer um *refactor* de todos os ficheiros *HTML* e *CSS*, removendo estilos que não são necessários ou que se sobrepõem uns aos outros. A remoção dos mesmos irá facilitar alterações futuras às páginas, pois não existirá código perdido.
Outra melhoria seria adaptar a aplicação dinamicamente para ecrãs pequenos, permitindo assim o uso desta em telemóveis. Para tal seria necessário o desenvolvimento de novos *designs* de *UI* para cada página da mesma.
Há ainda uma proposta por parte do cliente, para o desenvolvimento de uma nova página, composta por um campo de assinatura para os *Leads* assinarem quando tencionam adquirir algum serviço disponibilizado pela empresa.

Bibliografia

[Atl] Atlassian. Jira [online]. Available from: https://www.atlassian.com/software/jira [cited 30 fevereiro 2021]. 8

[Geo] GeoTab. Mygeotab [online]. Available from: https://www.geotab.com/blog/mygeotab-gps-fleet-management-app/ [cited 30 fevereiro 2021]. 3

[GPS] GPS Tracking America. Mygeotab fleet management app by geotab [online]. Available from: https://www.gpstrackingamerica.com/mygeotab-app/ [cited 30 fevereiro 2021]. 3

[Lif] Lifewire. Foursquare city guide [online]. Available from: https://www.lifewire.com/foursquare-app-4590138 [cited 30 fevereiro 2021]. 5

[Mob] Mobalo. Mobalo [online]. Available from: https://www.mobalo.com/en/technology/ [cited 30 fevereiro 2021]. 5

[Str] Strava. Mygeotab features [online]. Available from: https://www.strava.com/features [cited 30 fevereiro 2021]. 4

[Tec] TechTarget. Google maps [online]. Available from: https://whatis.techtarget.com/definition/Google-Maps [cited 30 fevereiro 2021]. 4

[Ube] Uber. Uber [online]. Available from: https://help.uber.com/riders/article/how-does-uber-work?nodeId=738d1ff7-5fe0-4383-b34c-4a2480efd71e [cited 7 outubro 2021]. 3

[Zub] Zubie. Zubie review [online]. Available from: https://www.pcmag.com/reviews/zubie [cited 30 fevereiro 2021]. 4