

iscte

INSTITUTO
UNIVERSITÁRIO
DE LISBOA

GeoRep, plataforma de gestão de terrenos com recurso a *Vue.js*

Filipe Ferreira da Silva

Mestrado em Software de Código Aberto

Orientador:
Doutor Sancho Moura Oliveira,
Professor associado,
ISCTE-IUL

outubro, 2020



TECNOLOGIAS
E ARQUITETURA

***GeoRep*, plataforma de gestão de terrenos com recurso a
*Vue.js***

Filipe Ferreira da Silva

Mestrado em Software de Código Aberto

Orientador:
Doutor Sancho Moura Oliveira,
Professor associado,
ISCTE-IUL

outubro, 2020

Direitos de cópia ou Copyright

©Copyright: Nome Completo do(a) candidato(a).

O Iscte - Instituto Universitário de Lisboa tem o direito, perpétuo e sem limites geográficos, de arquivar e publicitar este trabalho através de exemplares impressos reproduzidos em papel ou de forma digital, ou por qualquer outro meio conhecido ou que venha a ser inventado, de o divulgar através de repositórios científicos e de admitir a sua cópia e distribuição com objetivos educacionais ou de investigação, não comerciais, desde que seja dado crédito ao autor e editor.

Resumo

Neste documento é representado e detalhado o processo de desenvolvimento da plataforma de gestão de terrenos “*GeoRep*”. A plataforma foi projetada como uma prova de conceito que disponibilize funcionalidades de georreferenciação, com o intuito de entender as implicações técnicas necessárias para conceber uma aplicação de grande escala que possa vir a ser usada por órgãos executivos menores.

Nos primeiros capítulos é feita uma introdução ao projeto, seguida de uma descrição do estado da arte no que toca a *frameworks* de *front-end* de *JavaScript*, convergindo posteriormente para uma comparação mais técnica, entre as duas opções que se revelaram mais consentâneas com o desenvolvimento deste projeto. Nos capítulos seguintes são descritas as metodologias e tecnologias utilizadas para a criação da plataforma, bem como as etapas de planeamento e concepção, os componentes que foram elaborados e o processo de implementação e desenvolvimento. Trata-se de uma aplicação de fácil consulta cujas principais funções são a possibilidade de adicionar terrenos à plataforma com recurso a uma ferramenta de desenho no mapa nacional, a consulta dos respetivos terrenos, e o armazenamento de imagens e documentação relativa aos mesmos.

Palavras-Chave: *Front-end frameworks*, *GeoRep*, Mapeamento, Terrenos, *Vue.js*.

Abstract

This document describes the development process of the land management platform “GeoRep”. The platform was designed as a proof of concept that provides georeferencing features, in order to understand the technical implications necessary to design a large-scale application that may be used by smaller executive bodies.

In the first chapters an introduction to the project is made, followed by a description of the state of the art with regard to JavaScript front-end frameworks, converging later on for a more technical comparison between the two options that proved to be more in line with the development of this project. The next chapters describe the methodologies and technologies used to create the platform, as well as the planning and design stages, the components that were developed and the implementation process. It is an easy-to-use application whose main functions are the possibility of adding lands to the platform using a drawing tool on the national map, consulting the respective land, and storing images and documentation related to them.

Keywords: Front-end frameworks, GeoRep, Lands, Mapping, Vue.js.

Índice Geral

Resumo	i
Abstract	ii
Índice Geral	iii
Índice de Tabelas	vi
Índice de Figuras	vii
Glossário de Abreviaturas e Siglas	ix
Capítulo 1 – Introdução	1
1.1. Enquadramento do tema	1
1.2. Motivação e relevância do tema	3
1.3. Questões e objetivos de investigação.....	3
1.4. Abordagem metodológica.....	4
1.5. Estrutura e organização da dissertação	4
Capítulo 2 – Estado da arte	5
2.1. Angular	7
2.2. React	8
2.3. Vue.js	8
2.4. Popularidade	9
2.5. Migrações.....	11
2.6. Performance	11
2.7. Curva de aprendizagem	12
Capítulo 3 – React vs. Vue.js	13
3.1. Estrutura.....	15
3.2. Mutação de dados	17
3.3. Event listeners.....	19
3.4. Computed properties.....	20
3.5. Watchers	21
3.6. Passagem de dados entre componentes	22
3.7. Renderização condicional	23
3.8. Ilação das comparações	24
Capítulo 4 – Metodologia e tecnologias utilizadas	25
4.1. Metodologias aplicadas.....	25
4.2. Tecnologias e ferramentas utilizadas	25
4.2.1. Vuetify.....	25
4.2.2. Node.js.....	25
4.2.3. Express	26

4.2.4.	Axios.....	26
4.2.5.	Mongoose	26
4.2.6.	JWT	26
4.2.7.	bcryptjs	27
4.2.8.	Google APIs	27
4.2.9.	Leaflet.....	27
4.2.10.	Vue I18n	28
4.2.11.	Git/GitHub	28
4.2.12.	Heroku	28
Capítulo 5 – Planeamento e concepção.....		30
5.1.	Funcionalidades e objetivos.....	30
5.2.	Desenho e modelação	31
5.2.1	Diagrama de contexto.....	31
5.2.2	Diagrama de casos de uso.....	32
5.2.3	Descrição de casos de uso e diagramas de sequência.....	33
5.2.3.1	<i>Adicionar terreno</i>	34
5.2.3.2	<i>Fazer upload de imagens de terrenos (Adicionar terreno)</i>	36
5.2.3.3	<i>Requisitar validação de terreno</i>	38
5.2.3.4	<i>Eliminar conta</i>	40
5.2.3.5	<i>Eliminar terreno</i>	41
5.2.3.6	<i>Eliminar outras contas</i>	42
5.2.3.7	<i>Editar terreno</i>	43
5.2.3.8	<i>Filtrar terrenos por distrito ou município</i>	45
5.2.3.9	<i>Consultar terreno no mapa nacional</i>	46
5.2.3.10	<i>Alterar a categoria de outro utilizador</i>	48
5.2.3.11	<i>Filtrar terrenos por utilizador</i>	50
5.2.3.12	<i>Validar terreno</i>	51
5.2.4	Diagrama e semântica de classes.....	52
5.3.	Componentes e instalação.....	57
5.3.1	Diagrama de componentes.....	57
5.3.2	Diagrama de instalação.....	59
Capítulo 6 – Implementação e desenvolvimento.....		60
6.1.	Back-end	60
6.2.	Front-end.....	61
6.2.1	App	61
6.2.1	NavBar.....	61
6.2.2	LanguageSwitcher	62

6.2.3	Home	63
6.2.4	Register.....	64
6.2.5	Profile	65
6.2.6	ChoroplethDistrictsMap	66
6.2.7	Explore	67
6.2.8	SetUserCategory	69
6.2.9	AddLand	70
6.2.10	PickAreaMap	72
6.2.11	Validation	73
6.2.12	DistrictsMap	74
6.2.13	FileInput	76
Capítulo 7 – Conclusões		77
Referências Bibliográficas		79
Anexos.....		82
	Anexo A - Glossário	83

Índice de Tabelas

Tabela 1 - Dados dos repositórios do GitHub em março de 2020.....	9
Tabela 2 - Ofertas de emprego no LinkedIn a nível mundial em março de 2020	10
Tabela 3 – Permissões dos utilizadores	30
Tabela 4 - Caso de uso: Adicionar terreno	34
Tabela 5 - Caso de uso: Fazer upload de imagens de terrenos	36
Tabela 6 - Caso de uso: Requisitar validação de terreno.....	38
Tabela 7 - Caso de uso: Eliminar conta.....	40
Tabela 8 - Caso de uso: Eliminar terreno	41
Tabela 9 - Caso de uso: Eliminar outras contas.....	42
Tabela 10 - Caso de uso: Editar terreno	43
Tabela 11 - Caso de uso: Filtrar terrenos por distrito ou município.....	45
Tabela 12 - Caso de uso: Consultar terreno no mapa nacional	46
Tabela 13 - Caso de uso: Alterar a categoria de outro utilizador	48
Tabela 14 - Caso de uso: Filtrar terrenos por utilizador	50
Tabela 15 - Caso de uso: Validar terreno	51
Tabela 16 - Classe: User - Utilizadores do sistema.....	53
Tabela 17 - Classe: Land - Terrenos adicionados ao sistema.....	54
Tabela 18 - Classe: File - Ficheiros carregados pelos utilizadores.....	55
Tabela 19 - Classe: Validation - Registos relativos às validações dos terrenos.....	56
Tabela 20 - Classe: ValidationState - Estados possíveis que uma validação pode ter ...	56
Tabela 21 - Classe: LandLike -Gostos de um terreno	56
Tabela 22 - Classe: Category - Categorias que podem ser atribuídas aos utilizadores ..	56

Índice de Figuras

Figura 1 - Logótipos do Angular, React e Vue.js.....	7
Figura 2 - Downloads no npm, de março de 2019 a abril de 2020.....	10
Figura 3 - Especificação do root component no Vue.js.....	14
Figura 4 - Especificação do root component no React.....	14
Figura 5 - Estrutura de um ficheiro .vue.....	15
Figura 6 - Estrutura de um ficheiro usado no React.....	15
Figura 7 - Importação de CSS externo no Vue.js.....	16
Figura 8 - Classes CSS num componente de React.....	16
Figura 9 - Inicialização de variáveis.....	17
Figura 10 - Alteração de valor de variável no Vue.js.....	18
Figura 11 - Event listener em input no React.....	19
Figura 12 - Filtro de teclas de event listener no React.....	19
Figura 13 - Event listener em input no Vue.js.....	19
Figura 14 - Event listener de cliques no React.....	19
Figura 15 – Caso de uso de computed property.....	20
Figura 16 - Uso de computed property.....	20
Figura 17 - Listener de alterações no React.....	21
Figura 18 - Alteração de valor de variável e verificação de preenchimento no React...	21
Figura 19 - Uso de watchers.....	21
Figura 20 - Passagem de dados do parent para o child component.....	22
Figura 21 - Passagem de função de parent para child component no React.....	22
Figura 22 - Definição de função no parent component no Vue.js.....	22
Figura 23 - Renderização condicional no React.....	23
Figura 24 - Renderização condicional no Vue.js.....	23
Figura 25 - Estruturação das tecnologias utilizadas.....	29
Figura 26 – Diagrama de contexto.....	31
Figura 27 - Diagrama de casos de uso.....	32
Figura 28 - Diagrama de sequência: Adicionar terreno.....	35
Figura 29 – Diagrama de sequência: Fazer upload de imagens de terrenos.....	37
Figura 30 – Diagrama de sequência: Requisitar validação de terreno.....	39
Figura 31 - Diagrama de sequência: Eliminar conta.....	40
Figura 32 - Diagrama de sequência: Eliminar terreno.....	41
Figura 33 - Diagrama de sequência: Eliminar outras contas.....	42
Figura 34 - Diagrama de sequência: Editar terreno.....	44
Figura 35 - Diagrama de sequência: Editar terreno (Carregar imagens).....	44
Figura 36 - Diagrama de sequência: Filtrar terrenos por distrito ou município.....	45
Figura 37 - Diagrama de sequência: Consultar terreno no mapa nacional.....	47
Figura 38 - Diagrama de sequência: Alterar a categoria de outro utilizador.....	49
Figura 39 - Diagrama de sequência: Filtrar terrenos por utilizador.....	50
Figura 40 - Diagrama de sequência: Validar terreno.....	51
Figura 41 - Diagrama de classes.....	52
Figura 42 - Diagrama de componentes.....	58
Figura 43 - Diagrama de instalação.....	59
Figura 44 - Exemplos de mensagens da "snack-bar".....	61
Figura 45 – NavBar.....	61
Figura 46 – LanguageSwitcher.....	62
Figura 47 - Home.....	63
Figura 48 - Register.....	64

Figura 49 - Profile	65
Figura 50 - Dialog de eliminar conta.....	65
Figura 51 – ChoroplethDistrictsMap.....	66
Figura 52 - Explore.....	67
Figura 53 - Dialog de detalhes de terreno.....	67
Figura 54 - Explore (Separador de Utilizadores).....	68
Figura 55 – SetUserCategory	69
Figura 56 - AddLand (Informação)	70
Figura 57 - AddLand (Mapa).....	71
Figura 58 - AddLand (Submeter)	71
Figura 59 – PickAreaMap	72
Figura 60 - Dialog de validação	73
Figura 61 - Terreno verificado	73
Figura 62 – DistrictsMap.....	74
Figura 63 - Exemplo de objeto GeoJSON.....	75
Figura 64 - FileInput.....	76

Glossário de Abreviaturas e Siglas

API – Interface de Programação de Aplicações, conjunto de códigos e padrões de programação para acesso a uma aplicação de software ou plataforma web.

CSS – Cascading Style Sheets, mecanismo para adicionar estilo (cores, fontes, espaçamento, etc.) a um documento web.

DOM – Document Object Model, interface para páginas web.

HTML – HyperText Markup Language, linguagem de marcação utilizada na construção de páginas na Web.

JS – JavaScript, linguagem de programação.

JSON – JavaScript Object Notation, formato compacto de troca de dados simples e rápida entre sistemas.

JSX – JavaScript XML, mistura de HTML com JavaScript.

JWT – JSON Web Token, padrão para a criação de tokens de acesso baseados em JSON.

MIT – Massachusetts Institute of Technology, licença de software.

NoSQL – termo genérico que representa bases de dados não relacionais.

NPM – Node Packet Manager, gestor de pacotes JavaScript.

ODM – Object Document Mapping, mapeador de objetos para armazenar em bases de dados.

PHP – Hypertext Preprocessor, linguagem interpretada livre, usada originalmente apenas para o desenvolvimento de aplicações.

SQL – Structured Query Language, linguagem de pesquisa declarativa padrão para bases de dados relacionais.

UI – User Interface, espaço onde a interação entre humanos e máquinas ocorre.

UML – Unified Modeling Language, linguagem padrão para a elaboração da estrutura de projetos de software.

URI – Universal Resource Identifier, cadeia de caracteres compacta usada para identificar ou denominar um recurso.

Capítulo 1 – Introdução

O relatório aqui apresentado é fruto do projeto ”*GeoRep*”, cujo desenvolvimento foi motivado pelo interesse em desenvolver uma aplicação de georreferenciação com o intuito de entender as implicações técnicas necessárias para criar uma aplicação de grande escala que permita a adição e visualização de terrenos numa interface gráfica no mapa nacional. A aplicação foi criada utilizando a framework Vue.js, através da metodologia Scrum, um subconjunto do Agile. As tarefas foram divididas em ciclos de trabalho de 2 a 4 semanas, com pequenas reuniões com o orientador no fim de cada, visando dar uma perspetiva geral do estado do trabalho e apontar/corrigir pequenas falhas e possíveis adições.

1.1. Enquadramento do tema

Em outubro de 2018 foi aprovado pelo governo um projeto de diploma de forma a regulamentar a forma como os prédios (propriedades e terrenos) sem dono conhecido passarão para as mãos do Estado, podendo ser reclamados pelos seus proprietários durante um período de 15 anos (Lança, Jornal de Negócios 2018). Este projeto foi criado principalmente por não existir em vigor um processo regulamentado que permitisse determinar com segurança quais são os prédios sem dono conhecido e de que forma podem passar a ser considerados como sendo de propriedade pública. Tratou-se de um passo muito importante para tornar mais eficazes a gestão e aproveitamento dos recursos florestais, uma vez que este projeto incide maioritariamente em meios rurais. O objetivo é ter um regime de identificação, reconhecimento e registo de prédios rústicos ou mistos sem dono conhecido, dotados de aptidão agrícola, florestal ou silvo pastoril (Confagri, 2019).

Este projeto fez com que órgãos executivos menores, como as juntas de freguesia, se comesçassem a mobilizar de forma a registar toda a informação relativa aos terrenos privados e seus proprietários, uma vez que uma grande percentagem da documentação existente acaba por ser desatualizada e, em muitos casos, inexistente. Apesar de vivermos na era da comunicação, e de haverem cada vez mais esforços para concretizar uma transição digital eficiente, os órgãos executivos locais ainda não contam com uma ferramenta digital que permita a fácil localização de terrenos, e respetivo armazenamento, o que motivou a ideia de projetar e desenvolver como prova de conceito uma plataforma que disponibilizasse funcionalidades de georreferenciação, com o intuito de entender as

implicações técnicas necessárias para conceber uma aplicação de grande escala que tenha capacidade de resolver este problema. Para isso foi necessário compreender quais as propriedades características de uma plataforma deste género, designadamente as tecnologias a utilizar pelo cliente que, contando com um sistema de suporte a coordenadas geográficas, pudessem ser utilizadas para o desenho de polígonos georreferenciados, na respetiva representação no mapa nacional, e mesmo na pesquisa de endereços com *feedback* imediato para o utilizador. Ainda na perspetiva de desenvolvimento teria de ser avaliada a melhor maneira de armazenar e lidar com estes dados criados pelo cliente, investigando quais os formatos e convenções utilizados pelos programadores, sempre tendo em conta a combinação com tecnologias já generalizadas de desenvolvimento, aliando a toda a lógica de funcionamento da plataforma, de forma a garantir o melhor equilíbrio entre usabilidade, performance e qualidade do sistema.

À criação desta aplicação está subjacente a compreensão do funcionamento das *frameworks front-end*, que se tornaram um dos campos mais dinâmicos do desenvolvimento de software, dinamismo esse que faz com que por vezes seja impossível acompanhar todas as tendências e ferramentas emergentes. Foi então feito um amplo estudo destes recursos, tanto a um nível mais global, onde se escalpelizaram diferentes parâmetros como a popularidade e a *performance*, como também a um nível mais técnico, de modo a concluir qual seria a *framework* mais apropriada para a aplicação.

1.2. Motivação e relevância do tema

Como motivação para a realização da aplicação, está não só o interesse em desenvolver como prova de conceito uma plataforma que disponibilize funcionalidades de georreferenciação e toda a investigação que lhe está adjacente, como também a vontade de aprender de forma aprofundada o funcionamento de várias ferramentas de programação. Não apenas das *frameworks front-end*, mas também das diferentes *libraries*, *APIs* e utilitários que são necessários para o desenvolvimento da aplicação.

Os serviços para programadores da *Google* oferecem um vasto leque de soluções para os mais variados fins, e como os seus recursos de mapeamento e localização estão tão massificados, geram um grande interesse de aprendizagem. O mesmo acontece com outros pacotes, como é o caso do *Leaflet* ou do *bcryptjs*, que contam com uma grande popularidade dentro das suas comunidades.

Levando em consideração estas intenções de aprendizagem, e a necessidade de uma ferramenta de registo de terrenos mais intuitiva, pareceu-me um projeto bastante extenso, atual e abrangente de várias tecnologias com as quais viria a solidificar e ganhar conhecimentos, para além da criação de uma ferramenta de grande utilidade.

1.3. Questões e objetivos de investigação

A questão de investigação proposta foi a seguinte:

Com que ferramentas e de que modo deverá ser elaborada uma plataforma de armazenamento de polígonos georreferenciados, e moldar esta funcionalidade de forma a que sirva como ferramenta de auxílio ao dispor de qualquer utilizador?

Depois de alguma pesquisa sobre o assunto foram levantados os seguintes objetivos de investigação:

- Quais as ferramentas de programação a serem usadas;
- Quais serão os padrões e formatos a utilizar;
- Como serão armazenados os dados;
- Que restrições o *software* deve ter de forma a manter a privacidade dos dados;
- Como validar os dados inseridos de modo a evitar possíveis casos de fraude.

1.4. Abordagem metodológica

Houve três fases metodológicas. A primeira fase consistiu na pesquisa do estado da arte no que toca às *frameworks front-end* disponíveis atualmente. Para abordar e recolher mais informação acerca deste tema, a opção tomada foi maioritariamente a leitura de artigos e matérias *online* acerca do uso e funcionamento destas tecnologias, tanto de fontes oficiais, como de outros programadores. Na segunda fase foram projetadas as principais funcionalidades que a aplicação deveria conter e o modo como estas se conjugariam com a *framework* escolhida. Foram também elaborados vários diagramas com base nas funcionalidades pretendidas, de forma a apontar possíveis adições/erros e acelerar o processo de desenvolvimento. Por fim, procedeu-se ao desenvolvimento e implementação da aplicação, seguindo processo de metodologia *Scrum*.

1.5. Estrutura e organização da dissertação

Este relatório divide-se em sete capítulos, sendo que no primeiro definiu-se a questão e os objetivos de investigação. No Capítulo 2 é apresentado o estado da arte no que toca a *frameworks front-end*. No Capítulo 3 é feita uma comparação técnica entre as duas *frameworks* que suscitaram mais interesse a partir do estado da arte. No Capítulo 4 são descritas as tecnologias e metodologias que foram adotadas no desenvolvimento da aplicação. No Capítulo 5 são descritos os objetivos e funcionalidades da aplicação. No Capítulo 6 é descrito o processo de programação e implementação da aplicação. No sétimo e último capítulo são apresentadas as conclusões do trabalho realizado e tarefas a realizar no futuro.

Capítulo 2 – Estado da arte

Uma aplicação *web* pode-se dividir em duas partes: o *back-end*, também chamado de *server-side*, e o *front-end* ou *client-side*. O *back-end* contém os serviços que processam a lógica de negócio e outros recursos, como bases de dados, servidores de ficheiros, servidores na nuvem, etc, formando a “espinha dorsal” de uma aplicação. O *front-end* de uma aplicação *web* é aquilo que utilizador vê e com que interage no seu navegador, albergando toda a aparência e apresentação da aplicação, bem como as funcionalidades que permitem tornar a aplicação interativa (Pastorino, M., Pluralsight).

Tomando como exemplo o *YouTube*, o *back-end* é onde correm os algoritmos de recomendação de vídeos, autenticação de contas, processamento de vídeos e, claro, o armazenamento desses mesmos vídeos na base de dados. O *front-end* é a página com que o utilizador interage no navegador, permitindo-lhe escolher o vídeo que quer ver no reprodutor. Estas unidades trabalham em conjunto para renderizar a aplicação.

Neste processo de renderização é gerado o output *HTML* (*HyperText Markup Language*) que vai ser lido pelo navegador para mostrar a aplicação ao utilizador. Este processo pode ocorrer tanto no *back-end* como no *front-end*. A renderização no *back-end* dá-se através de uma solicitação de conteúdo por parte do cliente ao servidor, que corre todos os processos necessários para a criação de uma resposta que é enviada de volta ao navegador. Esta resposta contém a página *HTML* que vai ser mostrada ao utilizador. Enquanto o servidor gera esta resposta, o navegador fica ocioso, apenas aguardando que o servidor conclua o processamento e lhe envie uma resposta, para que a interprete e exiba o *output*. No caso da renderização no *front-end*, a renderização do conteúdo acontece no lado do cliente, em vez de ser no servidor. Neste caso o servidor envia a aplicação *web* em bruto, e é o navegador que se encarrega de gerar o *HTML* final da aplicação. Esta possibilidade só surgiu nos últimos anos devido ao aparecimento das *frameworks front-end*.

As tecnologias base no desenvolvimento do *front-end* são o *JavaScript*, *HTML* e *CSS* (*Cascading Style Sheets*), e por alguns anos foram as únicas usadas. Durante os primeiros anos da *web*, o desenvolvimento de um *site* que mantivesse uma boa aparência em vários navegadores era um processo extremamente moroso e complicado. Então, no início da década de 2000, começaram a surgir *libraries* e *frameworks* de *CSS*. Estas *frameworks* eram compatíveis com a maioria dos navegadores, e ao adotarem um sistema em grelha, facilitaram a vida dos programadores no que toca à organização do conteúdo na página.

Algumas das frameworks CSS mais antigas são o *Blueprint*, *960*, *YUI Grids* e *YAML* (Wanyoike, LogRocket 2018).

Quando foram lançados os primeiros *smartphones*, não houve um grande esforço por parte dos programadores para redesenhar *sites* para ecrãs de telemóvel. Os próprios navegadores móveis foram capazes de reformatar as páginas de modo a serem apresentadas num ecrã mais pequeno. Porém, com a manifestação rápida de popularidade destes dispositivos, gerou-se a necessidade de projetar os *sites* para ecrãs menores. Então em 2011, o *Twitter* lançou o *Bootstrap*, uma framework CSS de código aberto, que com atualizações regulares e com um *design* mais orientado para dispositivos móveis, se tornou a framework CSS mais utilizada.

Tal como o CSS, o *JavaScript* também causava imensos problemas de compatibilidade. Uma das primeiras *libraries* que permitiu simplificar o trabalho com *JavaScript* foi o *jQuery*. Lançado em agosto de 2006, facilitava imenso o processo de desenvolvimento, ao disponibilizar inúmeras funções úteis aos programadores, que garantiam compatibilidade entre os diferentes navegadores. O *jQuery* ao longo dos anos ganhou imensa popularidade e tornou-se na *library JavaScript* mais popular para *interfaces web*, porém não possuía recursos para lidar com dados de maneira consistente em *views* compartilhadas. Foram então desenvolvidas várias *frameworks* para resolver esse problema, como o *Backbone*, *Knockout*, *Ember* e o *Angular*, que rapidamente se tornou a *framework JavaScript* de *front-end* mais usada.

Com o passar do tempo, foram sendo lançadas outras soluções, sendo que atualmente o *Angular* partilha a sua popularidade com outras duas soluções, o *React* e o *Vue.js*.

Estas ferramentas tornaram-se populares porque para além de disponibilizarem ferramentas para um desenvolvimento mais rápido e simplificado, permitem também ao programador manter o controlo do estado da página de forma transparente (*state management*) (Schwarz Müller, M, Academind 2017).

Os componentes são parte integrante destas três ferramentas. Um componente geralmente recebe um *input* e altera o comportamento com base nele, o que facilita bastante a reutilização de código, visto que o mesmo componente pode ser usado para várias situações em páginas diferentes (Lobera, A., Medium 2018).

As três usam a licença *MIT* (*Massachusetts Institute of Technology*), que permite ao utilizador fazer o que quiser, desde que inclua o aviso de direitos autorais e licença original em qualquer cópia do *software*.

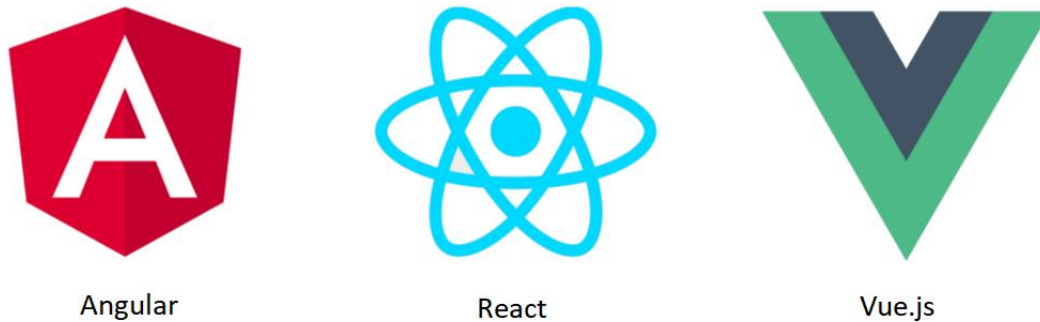


Figura 1 - Logótipos do Angular, React e Vue.js

2.1. Angular

O *Angular* é uma *framework* baseada em *TypeScript*, uma derivação de *JavaScript*, e foi lançado em 2010 pela *Google*. O *Angular 2* (agora simplesmente denominado de "*Angular*") sucedeu ao *Angular 1* (agora *AngularJS*) no final de 2016. Foi uma reescrita completa da *framework* que teve como objetivo corrigir muitos problemas de desempenho e comportamento que o *AngularJS* tinha. Esta mudança, que tornou o *AngularJS* completamente incompatível com o *Angular*, levou a que muitos programadores procurassem outras opções com receio que futuras atualizações viessem causar os mesmos problemas de incompatibilidade (Shaked, U., Medium 2017). A versão estável mais recente é o *Angular 8*, lançado em maio de 2019.

O *Angular* possui ligação de dados bidirecional, o que assegura que as alterações de dados num componente sejam refletidas no *parent component* (Marx, L., Medium 2017). Isto pode causar alguns problemas de escalabilidade e manutenção, uma vez que um componente pode alterar os dados do seu *parent component* sem que a fonte dessa alteração seja óbvia.

2.2. React

O *React* é uma *library*, ao contrário do *Angular* e do *Vue.js*, que são *frameworks*. A diferença técnica é explicada com o termo “inversão de controlo”. Enquanto que, quando está a usar uma *library*, o programador é o responsável pelo fluxo da aplicação, especificando quando e onde pretende chamar a *library*, com uma *framework* é esta a responsável pelo fluxo. Esta define onde deve ser inserido o código e usa-o conforme seja necessário.

Foi lançado em 2013, e é desenvolvido pelo *Facebook*, sendo utilizado em todos os seus produtos. A versão estável mais recente é a 16.13.1, lançada em março de 2020.

Em *React* todo o código é *JavaScript*, mais precisamente *JSX (JavaScript XML)*, que é uma mistura de *HTML* com *JavaScript* (Mohan, M., freeCodeCamp 2019). Caso se pretenda utilizar em aplicações mais complexas, com encaminhamentos de rotas (*routing*), é necessário recorrer a ferramentas *third-party*, uma vez que não existe nenhuma solução oficial para estes casos (Rogojan, B., ButterCMS 2019).

Ao contrário do *Angular*, no *React* a ligação de dados é unidirecional, o que para além de tornar o código mais estável, confere também um maior suporte ao desenvolvimento futuro.

2.3. Vue.js

Há uns anos, quando estava em causa a escolha de uma *framework front-end* para um projeto, os programadores deliberavam maioritariamente a sua escolha entre o *React* e o *Angular*. Mas, ao longo de 2018 e 2019, deu-se um crescendo no interesse de uma terceira opção chamada *Vue.js*.

Vue.js, também conhecido apenas como *Vue*, começou a ser desenvolvido em 2014 e, ao contrário das *frameworks* anteriores, não contou com uma grande empresa como o *Facebook* ou a *Google* na sua génese. Foi criado pelo ex-funcionário da *Google*, Evan You, que lidera atualmente a equipa de desenvolvimento.

Usa *Javascript*, *HTML* e *CSS*. Sendo a mais recentes das três *frameworks* apresentadas, e como forma de atrair programadores de outras *frameworks*, conta também com a possibilidade de trabalhar com *TypeScript* e *JSX*. As ferramentas de *routing* e *state management* são desenvolvidas e disponibilizadas pela equipa de desenvolvimento oficial.

A versão estável atual é a 3.0.4, lançada em dezembro de 2020. A equipa de desenvolvimento do *Vue* é mantida financeiramente através do *Patreon*, uma plataforma de financiamento coletivo, e de doações no *site* oficial.

2.4. Popularidade

A popularidade de uma *framework* influencia diretamente o seu ecossistema na quantidade e qualidade de pacotes *third-party* a seu serviço, no nível de suporte disponível e nos recursos de aprendizagem que se podem encontrar.

As grandes empresas acabam por atuar como uma grande ferramenta de *marketing*, o que faz com que mais empresas adotem as *frameworks* delas, e se por um lado isto pode trazer alguma desconfiança em relação a *frameworks* menores, por outro, acaba por trazer mais independência aos desenvolvedores.

Por não ser desenvolvido por uma grande empresa, seria de esperar que o *Vue* tivesse uma baixa popularidade, mas esta *framework* acabou por se tornar muito conhecida dentro da comunidade *open source*. Porém, a nível de oportunidades profissionais, o *Vue* ainda não se encontra ao mesmo nível do *Angular* ou do *React*.

Uma boa métrica de comparação da popularidade destas *frameworks* é a quantidade de estrelas e *forks* que os seus repositórios do *GitHub* têm.

Tabela 1 - Dados dos repositórios do *GitHub* em março de 2020

	<i>Angular</i>	<i>React</i>	<i>Vue</i>
Seguidores	3.2k	6.6k	6k
Estrelas	58.4k	145k	158k
<i>Forks</i>	16k	27.8k	23.9k
Contribuidores	1093	1364	289

Analisando os dados dos repositórios do *GitHub* podemos constatar que o *Vue* tem um grande número de seguidores, estrelas e *forks*, o que demonstra a sua popularidade e valor entre os programadores (Daityari, S, CodeinWP 2020). Uma das causas para ter menos contribuidores é o facto de o *Vue* ser desenvolvido inteiramente pela sua comunidade *open source*, enquanto que o *Angular* e o *React* têm uma parcela significativa de funcionários da *Google* e do *Facebook* a contribuir para os repositórios.

Através do *site npxmtrends.com*, podemos gerar gráficos comparativos da quantidade de *downloads* no *npm* (*Node Packet Manager*), um gestor de pacotes de *JavaScript*. Verifica-se na Figura 2 um claro domínio de *downloads* relacionados com o *React*, visto ser muito usado a nível profissional e também pelas diversas ferramentas suplementares disponíveis, no entanto o *Vue* também se sobrepõe ao *Angular* nesta métrica.

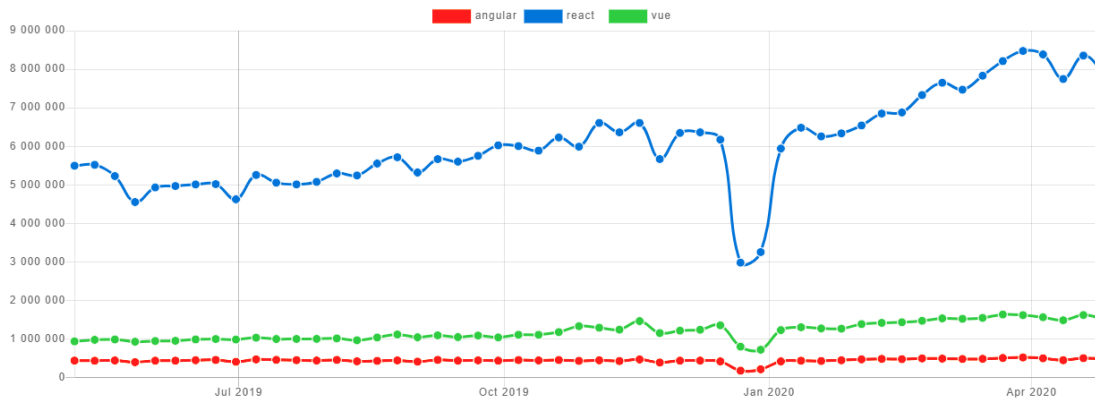
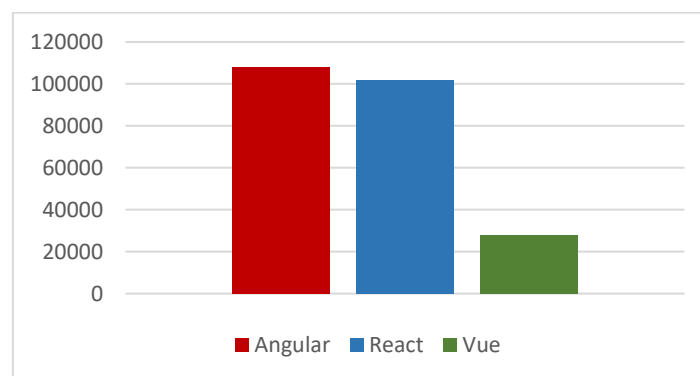


Figura 2 - Downloads no *npm*, de março de 2019 a abril de 2020

No que toca a ofertas profissionais, o *Vue* está em clara desvantagem. Comparando a quantidade de ofertas de emprego presentes no *LinkedIn*, a maior rede social profissional, com o *Angular* e *React*, verificamos que há muito menos propostas de emprego. Apesar de ser a *framework* com mais estrelas no *GitHub*, revelando que a comunidade de programadores tem uma grande apreciação pelo *Vue*, a indústria ainda está a começar a adotá-lo.

Tabela 2 - Ofertas de emprego no *LinkedIn* a nível mundial em março de 2020



Como o *Vue* ganhou popularidade apenas nos últimos três anos, o número de trabalhos relacionados com *Angular* ou *React* sobrepõe-se largamente aos trabalhos relacionados com *Vue*, que ainda são apenas uma fração. Pode levar algum tempo para projetos estabelecidos começarem a usar o *Vue*, ou que novos projetos que tenham adotado *Vue*

atingam um nível de maturidade que os faça ganhar a necessidade de contratar um número maior de programadores, mas todos os projetos mostram uma atividade significativa de desenvolvimento, e certamente continuará no futuro (Neagoie, Medium 2018).

2.5. Migrações

A partir do momento que tem uma *framework* a funcionar, nenhum programador se quer preocupar com atualizações da mesma que causem problemas no código. Embora na maioria dos casos não se encontrem muitos problemas de uma versão para outra, é importante manter o controlo, pois algumas atualizações podem ser mais significativas e requerem ajustes para manter tudo compatível.

O *Angular* tem grandes atualizações a cada seis meses. Também há um período de outros seis meses antes das principais *APIs* (Interface de Programação de Aplicações) serem descontinuadas, o que dá dois ciclos de lançamento (um ano) para fazer as alterações necessárias.

Quanto ao *React*, as atualizações de versões geralmente não causam grandes problemas, contando com *scripts* criados para ajudar no processo de migração.

Na sua documentação, o *Vue* menciona que a *API* se manteve 90% igual da versão 1 para a 2, e vai sofrer alterações mínimas da versão 2 para a 3. Terá também uma ferramenta auxiliar de migração que funciona na consola para avaliar o estado da aplicação.

Portanto, mesmo introduzindo novas funcionalidades com diferentes regularidades, as três opções mantêm uma boa taxa de compatibilidade com aplicações desenvolvidas com recurso a *APIs* de versões anteriores, e disponibilizam também mecanismos de auxílio a possíveis migrações ou um prazo prolongado de transição.

2.6. Performance

A *Document Object Model (DOM)*, é uma *interface* para páginas *web* que permite que os programas leiam e manipulem o conteúdo, estrutura e estilos da página. A *DOM* muda sempre que a *interface* do utilizador é atualizada, representando as alterações feitas na aplicação.

Pode ser usado de duas maneiras, como *DOM* real ou *DOM* virtual, e exerce um grande papel sobre o desempenho das *frameworks*. O *Angular* utiliza *DOM* real, o que afeta (ligeiramente) o seu desempenho tanto na fluidez da aplicação como no carregamento inicial do pacote, para além de também diminuir a capacidade de criar aplicações dinâmicas. Por outro lado, o *React* e o *Vue* usam *DOM* virtual. Tal como a *DOM* real, a

DOM virtual lista os elementos da página, e sempre que há uma mutação no estado da aplicação essa lista é atualizada, criando uma nova representação da *DOM*. A atualização da *DOM* do navegador é então atualizada com a diferença entre a representação anterior da *DOM* virtual e a nova que foi calculada. Desta forma apenas é atualizado na *DOM* do navegador aquilo que realmente foi alterado (Minnick, C., Accelebrate 2016).

2.7. Curva de aprendizagem

Um dos fatores mais importantes na escolha de uma *framework* é a sua facilidade de aprendizagem. Pode-se achar que a qualidade de uma *framework* é definida através da qualidade das aplicações que se criam com ela, a *performance* dessas aplicações e a sua escalabilidade, mas se uma *framework* é extremamente difícil de aprender, haverá menos programadores a utilizá-la e ainda menos a dominá-la, o que prejudicará imenso a sua popularidade.

O *Angular* possui uma curva de aprendizagem acentuada, considerando que é uma solução completa. O seu domínio exige que se aprendam outros conceitos associados, como o *TypeScript* e uma variedade de estruturas diferentes, ao contrário do *React* e do *Vue* quem têm apenas o componente em mente.

Para trabalhar com *React* é necessária a aprendizagem de *JSX*, que pode ser difícil de entender no começo. O *state management* é outro tópico que se pode tornar problemático visto ser necessário recorrer a soluções não oficiais.

O *Vue* oferece maior capacidade de personalização, sendo mais fácil de aprender do que *Angular* ou *React*. A transição para o *Vue* de qualquer um dos dois é uma opção fácil, visto que também suporta *Typescript* e *JSX*, contém muitas das mesmas funcionalidades, e conta também com a melhor documentação das três *frameworks*, o que ajuda a acelerar a curva de aprendizagem.

Num caso de transição do *Vue* para uma das outras alternativas, o processo não apresenta muitas complicações, uma vez que o *Vue* conta com vários *features* e funcionalidades presentes nas outras *frameworks*, o que garantirá ao programador uma boa base para esta transição.

O *Vue* junta numa só *framework* os melhores atributos das suas “rivais”, fazendo com que uma transição que parta de ou para o *Vue* acabe sempre por ser um processo menos complexo, do que uma transição entre o *Angular* e o *React*, por exemplo (Ahuvia, Medium 2019).

Capítulo 3 – React vs. Vue.js

A *framework/library* que iria ser usada no projeto não foi clara desde o início. Foi então necessário compará-las para saber o que as distingue, servindo para isso o estado da arte. O *Angular* foi descartado principalmente porque já não contava, com o mesmo favoritismo que o *React* ou o *Vue*, mas também devido à sua complexidade. A queixa mais frequente por parte da comunidade de *Angular* é a sua gestão confusa de ficheiros. Tanto que para um único componente, chegam a ser necessários cinco ficheiros distintos, para além de serem precisas outras dependências e para casos de uso mais complicados, declarar mais *interfaces*. O facto de *TypeScript* ser essencial também é um ponto negativo. Apesar de ser relativamente fácil para um programador com competências de JavaScript fazer a transição para *TypeScript*, seria necessário algum investimento em tempo e dedicação para aprender uma nova linguagem, investimento esse que acabaria por ser amplificado devido à documentação pouco detalhada, que levaria a uma produtividade baixa na fase inicial. Para complementar, como explicado anteriormente, a sua *performance* também revela ser a pior das três apresentadas.

Na criação de *UIs* (*interfaces* de utilizador), o *React* e o *Vue* possuem uma arquitetura baseada em componentes. Essencialmente, um componente é um pedaço de código que representa parte de uma página e trata-se de uma das principais características de uma *framework front-end*. Estes componentes podem ser reutilizados as vezes que forem necessárias, sendo normalmente programados com a configuração mais genérica possível de modo a que se possam reutilizar entre diversas páginas da aplicação, dependendo do seu fluxo de trabalho, garantindo uma maior capacidade de produção. O fluxo de trabalho de um componente dita o modo de como este recebe os dados de entrada e lida com esses dados.

O primeiro passo em qualquer uma destas *frameworks* é especificar o *root component*, ou seja, o componente a partir do qual todos os restantes componentes e *views* serão chamados.

No *Vue*, num ficheiro *JavaScript* que serve de ponto de entrada, é montada uma *div* com um *id* normalmente definido por “*app*”, que está inserida no único ficheiro *HTML*, o *index.html*.

<pre>new Vue({ render: h => h(App) }).\$mount("#app");</pre>		<pre><div id="app"></div></pre>
JS		HTML

Figura 3 - Especificação do *root component* no *Vue.js*

Dentro dessa *div* vai estar contido o *root component*, normalmente designado de *App.vue*. No *React* monta-se no ponto de entrada o elemento “*root*”, uma *div* que está inserida no ficheiro *HTML*. É passado como parâmetro o componente *App*, que está no ficheiro *App.js*, servindo o mesmo propósito do ficheiro *App.vue*.

<pre>const rootElement = document.getElementById("root"); ReactDOM.render(<App />, rootElement);</pre>		<pre><div id="root"></div></pre>
JS		HTML

Figura 4 - Especificação do *root component* no *React*

Como se pode perceber, logo num dos primeiros passos do desenvolvimento, a mesma declaração requer procedimentos diferentes nas duas opções. Foi, portanto, necessário fazer uma escolha ponderada, de modo a evitar percalços no caminho.

Se por um lado a flexibilidade, a popularidade crescente, e a estima dos programadores pelo *Vue* eram fatores atrativos, por outro, a maturidade e variedade de ferramentas disponíveis para trabalhar com *React* também não faziam a escolha fácil. Foi então necessário compará-los a um nível mais técnico, escrutinando alguns casos recorrentes durante o processo de programação.

Neste capítulo vão comparar-se estas duas alternativas e apresentar exemplos concretos de código da aplicação que foi desenvolvida.

3.1. Estrutura

O *Vue* adotou uma abordagem mais contida para a organização da sua estrutura, reunindo a parte da *interface*, do comportamento do componente e até da estilização, toda no mesmo ficheiro, mas em códigos separados, de maneira bastante organizada, sendo uma das mais notáveis qualidades do *Vue*. Esta estrutura está organizada em três secções: *template*, *script* e *style*, como é demonstrado na figura seguinte.

```
<template>
<div>Olá Mundo</div>
</template>

<script>
export default {

}
</script>

<style>
</style>
```

Figura 5 - Estrutura de um ficheiro *.vue*

Na secção *template* é colocado o código *HTML* da *interface* do componente e todos os elementos *HTML* válidos são também modelos válidos do *Vue*, podendo usufruir dos seus recursos. No *script* é colocada toda a lógica que faz o componente funcionar, e no *style* são colocadas classes *CSS* que determinarão os diferentes estilos que podem ser usados na *template*.

No *React*, a estrutura de um componente é bem diferente. A classe *Component* tem de ser importada, de forma a poder ser usada para estender a nova classe, que dá nome ao componente. No exemplo abaixo o componente chama-se “*OlaMundo*” e gera exatamente o mesmo conteúdo do componente *Vue* apresentado anteriormente.

```
import React, {Component} from 'react';

class OlaMundo extends Component{
  render(){
    return (
      <div>Olá Mundo </div>
    );
  }
}

export default OlaMundo;
```

Figura 6 - Estrutura de um ficheiro usado no *React*

Então para cada componente do *React*, tem de ser criada uma classe e exportá-la para poder ser usado noutros componentes.

No fundo, esta *classe* pode ser comparada às secções da *template* e do *script* nos componentes *Vue*. No entanto, o código *CSS* para a estilização está em locais diferentes. No caso do *Vue*, as classes *CSS* são inseridas na secção *style*. Estas classes são acessíveis a partir de qualquer componente, independentemente de em qual estas tenham sido adicionadas, mas caso se pretenda que as classes funcionem apenas no componente onde foram escritas, pode-se usar o atributo *scoped* na *tag* da secção. Opcionalmente, também pode ser importado *CSS* externo, como é demonstrado na Figura 7.

```
<style scoped>
@import "../assets/exemplo.css";

.verde{
  color: green;
}
</style>
```

Figura 7 - Importação de *CSS* externo no *Vue.js*

No *React* a prática comum é importar as classes *CSS* de um ficheiro *JS* (*JavaScript*) externo, onde estas são guardadas como objetos normais de *JS*, mas também é possível ter classes *CSS* dentro do componente. Este recurso não é muito utilizado por não favorecer a simplicidade do código, sendo necessário criar uma variável para os *styles*, como é demonstrado na Figura 8.

```
class OlaMundo extends Component {
  render() {
    const vermelho = {
      background: "red"
    };

    return <div style={vermelho}>Olá Mundo </div>;
  }
}
```

Figura 8 - Classes *CSS* num componente de *React*

3.2. Mutação de dados

A declaração de variáveis dá-se de forma muito semelhante nas duas *frameworks*, como é demonstrado na Figura 9, em que é declarada a variável “*selectedLand*” com o valor *null*. Assim que o utilizador clicar num terreno, vai ser corrido um método (que não é visível na figura) que vai alterar o valor da variável para o objeto referente ao terreno que esteja selecionado. A variável está inserida dentro de um elemento *HTML* para mostrar o nome do terreno.



```

<template>
  <div>
    <p>{{selectedLand.name}}</p>
  </div>
</template>

<script>
export default {
  data() {
    return {
      selectedLand: null
    };
  }
};
</script>

```

```

class Home extends Component {
  constructor(props) {
    super(props);
    this.state = {
      selectedLand: null
    };
  }

  render() {
    return (
      <div>
        <p>{this.state.selectedLand.name}</p>
      </div>
    );
  }
}

```

Figura 9 - Inicialização de variáveis

De forma a que a aplicação tenha uma maior interatividade, a *interface* deve reagir constantemente à mutação de dados. Mutação de dados é o processo de alteração de um dado armazenado, ou simplesmente, de mudança de valor de uma variável. Estes dados compõem o “estado” da aplicação, e existe aqui um grande contraste entre o *React* e o *Vue* (Rusev, A., Mentormate 2019). Enquanto que o *Vue* cria objetos que podem ser atualizados livremente, o *React* cria um objeto que exige algum trabalho quando precisa de ser atualizado. Isto porque o *React* promove a filosofia de que o estado deve ser imutável. Quando o objeto do estado é alterado não ocorre nenhuma renderização. Para que uma nova renderização ocorra é necessário chamar o método “*setState*” e passar o novo valor da variável como parâmetro. Este passo extra na mutação de dados executa certos eventos, de forma a saber se deve atualizar ou renderizar não apenas o *root component*, como também toda a subárvore do componente, o que acaba por oferecer ao programador um maior controlo sobre os processos de renderizações, em detrimento de uma maior flexibilidade de código, que se encontra no *Vue*. Na Figura 10 é apresentado

um exemplo do *Vue* em que, depois de declarada a variável “*selectedLand*” com o valor *null*, a mesma é referenciada através da expressão “*this.selectedLand*”, e o seu valor é alterado para o objeto correspondente ao terreno que foi selecionado pelo utilizador.

```
<script>
export default {
  data() {
    return {
      selectedLand: null
    };
  },
  methods: {
    selectLand(){
      this.selectedLand = this.$router.landToOpen;
    }
  }
};
</script>
```

Figura 10 - Alteração de valor de variável no *Vue.js*

No entanto, a mutação de propriedades de objetos ou de matrizes não acionam uma nova renderização. Neste caso pode usar-se o método *Vue.set* (semelhante ao método *setState* do *React*). O *Vue* oferece também o atributo “*v-model*”, que permite fazer ligações bidirecionais dentro do componente, da secção de *template* para o *script*. Devido a este atributo, existe uma ligação direta ao elemento, e a alteração de dados da variável aciona diretamente a renderização do mesmo. São também realizadas otimizações automaticamente para atualizar partes específicas da árvore de elementos, não existindo uma maneira manual de as impedir. Este comportamento pode levar à necessidade de criar uma variável auxiliar, caso o programador pretenda renderizar um valor que não está atualizado. Um exemplo deste caso é encontrado no *YouTube*. Quando o utilizador está a visualizar um vídeo, é-lhe mostrado o valor de uma variável auxiliar que contém um valor fixo relativamente à quantidade de visualizações que o vídeo já conseguiu, em vez deste valor ser atualizado constantemente, o que diminuiria a performance e poderia também tirar o foco do vídeo.

3.3. Event listeners

Um *event listener* é um método que contém código que deve ser executado como resposta a um evento. Um exemplo destes eventos é o caso em que o utilizador pressiona a tecla “Enter” do teclado, depois de preencher as suas credenciais, para fazer *login* na aplicação. O *React* lida com estes eventos de maneira simples. Abaixo é apresentado um exemplo em que é configurado um evento “*onKeyPress*” que será lido pelo elemento “*input*”, uma caixa de texto.

```
<input type='password' onKeyPress={this.onPasswordKeyPress}/>
```

Figura 11 - Event listener em input no React

Isto fará com que o método “*onPasswordKeyPress*” seja chamado sempre que o utilizador pressionar qualquer tecla dentro da caixa de texto. Neste caso tem-se de filtrar, para apenas a tecla “Enter” poder chamar o método de *login*.

```
onPasswordKeyPress = e => {
  if (e.key === "Enter") {
    this.login();
  }
};
```

Figura 12 - Filtro de teclas de event listener no React

O *Vue* tem uma abordagem ainda mais simples, usando o atributo “*v-on*” seguido do evento que se pretende seguir. Para replicar o comportamento do *React*, a tecla que deve ser filtrada pode ser especificada diretamente na definição do evento, como é demonstrado na Figura 13.

```
<input type="password" v-on:keyup.enter='login' />
```

Figura 13 - Event listener em input no Vue.js

Para além disso o *Vue* também disponibiliza outros atributos que podem ser encadeados, como o “*.once*”, que previne o evento de ser disparado mais de uma vez.

Para o caso em que se pretende chamar um método ao clicar num botão, no *Vue* deve ser usada expressão “*v-on:click*”, enquanto que o seu paralelo no *React* é o “*onClick*”.

```
<button onClick={ this.login }>Login</button>
```

Figura 14 - Event listener de cliques no React

3.4. Computed properties

As *computed properties* são um recurso interessante do *Vue*, que não existe no *React*, e têm como principal objetivo deixar o código mais limpo, bem como melhorar a *performance*. Na Figura 15 é mostrado o caso em que se verifica no atributo “*disabled*” se o *email* e a *password* foram preenchidos, para ativar o botão de *login*.

```
<template>
  <div>
    <v-btn v-on:click="loginUser"
      :disabled="!email || !password"
    >Login</v-btn>
  </div>
</template>
```

Figura 15 – Caso de uso de *computed property*

A expressão presente no atributo “*disabled*” é simples, mas é necessário observar por alguns segundos para perceber de que se trata. Este problema agrava-se caso seja necessário fazer a verificação em mais sítios, o que irá aumentar a desorganização no código. É nestes casos que se recomenda o uso de uma *computed property*, declarando-a dentro da secção do *script*, e de seguida adicionando-a à *template*.

```
<template>
  <div>
    <v-btn v-on:click="loginUser"
      :disabled="isLoginDisabled"
    >Login</v-btn>
  </div>
</template>

computed: {
  isLoginDisabled() {
    return !this.email || !this.password;
  }
}
```

Figura 16 - Uso de *computed property*

No *React*, este recurso pode ser replicado, criando um método que determina o mesmo resultado, porém quando se usa uma *computed property*, o *Vue* sabe que só precisa de atualizar o elemento, quando foi alterado um dos valores dos quais a *property* depende. Os métodos não sabem se os valores foram alterados, portanto atualizam sempre o elemento, o que é escusado pois pode prejudicar a *performance*.

3.5. Watchers

No *React*, caso se queira verificar se uma variável foi alterada é necessário declarar um *listener*. Na Figura 17 é demonstrado um caso onde é configurado um *listener* que chama um método que atualiza o valor da variável sempre que o texto do *input* for alterado. Esta variável corresponde ao nome do terreno, quando se está na página “Adicionar terreno”.

```
<input value={this.state.nomeTerreno} onChange={this.guardarNome}/>
```

Figura 17 - Listener de alterações no React

A função “*guardarNome*” é executada sempre que o valor do *input* muda, de modo a guardá-lo numa variável. Neste momento também se faz a verificação se o nome está preenchido visto que este é obrigatório, e o separador “Submeter” precisa que esta verificação seja feita de modo a ficar ativado.

```
guardarNome = e => {
  this.setState({
    nomeTerreno: e.target.value
  });
  this.setState({
    nomeTerrenoValido: e.target.value.length > 0
  });
};
```

Figura 18 - Alteração de valor de variável e verificação de preenchimento no React

No *Vue* este processo é simplificado através do uso de *watchers*. Basta chamar a variável dentro do objeto “*watch*” da secção *script* que está ligada ao *input* através do “*v-model*”. Isto fará com que o código que está presente no *watcher* seja executado automaticamente sempre que o valor da variável mudar. Neste caso não é necessário atribuir de novo o valor à variável porque o “*v-model*” encarrega-se disso.

```
<input v-model="nomeTerreno">

watch: {
  nomeTerreno(valor) {
    this.nomeTerrenoValido = valor.length > 0;
  }
}
```

Figura 19 - Uso de watchers

De notar que os *watchers* funcionam também caso a variável não esteja ligada a nenhum elemento *HTML* e seja usada apenas na secção *script*.

3.6. Passagem de dados entre componentes

A passagem de dados do *parent* para o *child component* é praticamente igual nas duas *frameworks*, porém no caso contrário já não se verifica esta semelhança.

```
<PickAreaMap :pickedCounty="pickedCounty"/> | <PickAreaMap pickedCounty={this.state.pickedCounty}/>
```

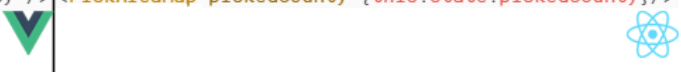


Figura 20 - Passagem de dados do parent para o child component

No *React*, caso se pretenda passar dados do *child* para o *parent component*, deve-se passar primeiro para o *child* uma função que permita alterar o valor de uma variável no *parent*. Assim, quando esta função for chamada a partir do *child*, estará a alterar valores no *parent*. Na Figura 21 é apresentado o caso em que o *child component* é o “*PickAreaMap*”, que necessita de informar o *parent* quando o desenho do terreno for concluído.

```
<PickAreaMap pickedCounty={this.state.pickedCounty}
concluirDesenho={this.concluirDesenho.bind(this, valor)}/>
```

Figura 21 - Passagem de função de parent para child component no React

Para replicar este comportamento no *Vue*, tem de se emitir o valor do *child* para uma função do *parent component*, função essa que é especificada na declaração do *child component*.

```
this.$emit("desenhoConcluido", valor);
```

```
<PickAreaMap :pickedCounty="pickedCounty"
@desenhoConcluido="concluirDesenho"/>
```

Figura 22 - Definição de função no parent component no Vue.js

Ou seja, no *React* os *child components* podem ter acesso a funções do *parent* se as mesmas forem passadas como *props*, enquanto que no *Vue* basta emitir o valor do *child* para o *parent* e este lidará com o valor através da função que for especificada.

3.7. Renderização condicional

No *React*, caso se pretenda renderizar um elemento condicionalmente, pode-se optar por várias maneiras. Abaixo estão representadas as duas maneiras mais comuns, demonstrando o caso em que no componente “*AddLand*”, caso o utilizador ainda não tenha escolhido um município, deve ser renderizado o componente “*DistrictsMap*”, caso contrário, deve ser renderizado o componente “*PickAreaMap*”.

```
{this.state.pickedCounty == null && <DistrictsMap />} | {this.state.pickedCounty == null ? (
{this.state.pickedCounty != null && <PickAreaMap />} | <DistrictsMap />
): (
<PickAreaMap />
)}
```

Figura 23 - Renderização condicional no *React*

Ou seja, em *React* caso se pretenda renderizar elementos opostos condicionalmente, tem de se recorrer a condições ternárias ou então, duplicar o código da condição e invertê-lo. No *Vue* isto torna-se muito mais simples, uma vez que este disponibiliza o atributo “*v-if*”, onde é colocada a condição, e de seguida se pode usar o atributo “*v-else*”, que se liga automaticamente à anterior declaração do “*v-if*”. Também se pode usar o atributo “*v-else-if*”, caso existam outras condições a verificar.

```
<DistrictsMap v-if="pickedCounty == null"/>
<PickAreaMap v-else/>
```

Figura 24 - Renderização condicional no *Vue.js*

3.8. Ilacão das comparações

Ambas as alternativas são excelentes escolhas para o desenvolvimento de uma aplicação *web* moderna. O *React* ainda é o líder, contando com o suporte de diversas grandes empresas, bem como da sua comunidade. Apresenta um bom ecossistema, que permite facilmente encontrar soluções para quase todos os problemas que se possam encontrar num projeto. Porém, como se pode concluir através dos exemplos anteriores, o *Vue* apresenta uma estrutura muito intuitiva, bem como uma simplicidade que nenhuma outra *framework* de *front-end* para aplicações de grande escala disponibiliza de momento. É fácil entender o porquê da sua popularidade ter ascendido tão rapidamente. Tem uma sintaxe mais tradicional, o que permite também que outros projetos e *libraries* sejam migrados facilmente e incorporados num projeto *Vue*, para além de o seu ecossistema ser providenciado em grande parte pela equipa de desenvolvimento oficial. Estes fatores, aliados à boa performance que se pretende numa aplicação *web*, à “limpeza” que o código permite através do formato adotado nos ficheiros, e à facilidade de aprendizagem, fazem do mesmo uma ferramenta muito interessante e capaz, com imensos recursos úteis, que justificam a preferência nesta *framework* por parte dos programadores.

Ambas as alternativas seriam apropriadas e capacitadas para este projeto, apresentando muitos dos mesmos recursos, porém, através de caminhos diferentes.

Sou defensor de que atualmente, qualquer escolha de uma *framework* deve sempre ser feita em favor da equipa de desenvolvimento. Se uma *framework* é mais fácil de estudar e aprender, possibilitar uma migração fácil de projetos ou *libraries* existentes, e estimular a criação de código limpo, de fácil e rápida leitura, a sua escolha vai compensar, porque o tempo gasto no estudo das especificidades da *framework* pode acabar por adiar o desenvolvimento da aplicação. Justifica-se então a escolha do *Vue* para o desenvolvimento deste projeto.

Capítulo 4 – Metodologia e tecnologias utilizadas

Neste capítulo serão descritas as metodologias aplicadas e as tecnologias que foram utilizadas para a realização da aplicação.

4.1. Metodologias aplicadas

Para a concepção da aplicação foi utilizada a metodologia de Desenvolvimento Ágil, que tem como objetivo acelerar o desenvolvimento de *software*, através de iterações e melhorias contínuas. O processo da metodologia utilizado foi o *Scrum*, onde as tarefas são divididas em ciclos, tipicamente de 2 a 4 semanas. Foram realizadas pequenas reuniões, acompanhadas de resumos do progresso, no fim de cada ciclo com o orientador, com o objetivo de dar uma visão geral do estado do trabalho e apontar/corrigir pequenas falhas e possíveis adições.

4.2. Tecnologias e ferramentas utilizadas

Neste subcapítulo irão ser enumeradas e abordadas as tecnologias e ferramentas que foram utilizadas em conjunto com o *Vue.js* no desenvolvimento da aplicação.

4.2.1. Vuetify

Vuetify é a *library* de componentes mais usada no *Vue.js*, com desenvolvimento constante desde 2016. Tem como objetivo disponibilizar aos programadores tudo o que é necessário para criar aplicações, oferecendo elementos fáceis de utilizar e com *design* moderno, uma vez que usa a especificação *Material Design*, a linguagem de *design* desenvolvida pela *Google*. Sofre atualizações frequentemente, tem suporte a longo prazo para versões anteriores, um grande envolvimento com a comunidade, e claro, um vasto catálogo de recursos e componentes de qualidade.

4.2.2. Node.js

O *Node.js* é um *runtime environment* em código aberto para programação em *JavaScript* do lado do servidor, graças ao *engine Chrome V8* (Patel, P., freeCodeCamp 2018). É um sistema que usa programação baseada em eventos para criar aplicações escaláveis e programas de rede sendo especialmente útil para a construção de servidores *web*. Oferece

funcionalidades que anteriormente apenas eram possíveis com *PHP* ou *Ruby*, como acessar aos ficheiros armazenados no computador, comunicar com *HTTP requests* na máquina, acessar diretamente a bases de dados, etc. Também permite a incorporação de módulos para expansão de funcionalidades e acelerar o desenvolvimento de aplicações através de *frameworks*.

4.2.3. Express

Express é a *framework* mais popular do *Node.js*, fornecendo mecanismos para a escrita de manipuladores de pedidos para diferentes rotas, integração com motores de renderização de *views* de forma a gerar respostas ao inserir dados em *templates*, definição de propriedades recorrentes como a porta que é usada na ligação, entre outros recursos.

4.2.4. Axios

Axios é uma *library Javascript* usada para fazer pedidos *HTTP* a partir do *Node.js*. Esta *library* é muito popular devido à sua sintaxe simples, ao *setup* rápido, e também graças aos seus recursos para lidar com erros.

4.2.5. Mongoose

Mongoose é uma *library ODM* (Modelagem de Documentos de Objetos) para *MongoDB* e *Node.js*. Serve para gerir as relações que os dados têm entre si, fornecendo recursos para validação de modelos de objetos e conversão entre objetos no código e a representação desses objetos no *MongoDB*. O *MongoDB* é uma base de dados *NoSQL* sem modelos, o que significa que é possível armazenar documentos *JSON* (*JavaScript Object Notation*) sem estes terem uma estrutura fixa, ao contrário das bases de dados *SQL*, o que acelera o desenvolvimento das aplicações e reduz a complexidade das implementações. O *MongoDB* não foi instalado localmente, devido ao uso do *MongoDB Atlas*, que permite o armazenamento de bases de dados na *cloud*.

4.2.6. JWT

O *JSON Web Token* (*JWT*) é um padrão aberto que define uma maneira compacta e independente de transmitir com segurança informações entre partes através de um objeto *JSON* (Nascimento, W., Medium 2018). Essas informações podem ser verificadas e

confiáveis porque são assinadas digitalmente. O *JWT* foi então usado para garantir que a sessão do utilizador atual é legítima. Este padrão é utilizado em conjunto com o “*passport-jwt*”, que para além de guardar o *token* do utilizador em cada sessão, oferece várias outras estratégias de autenticação de implementação fácil.

4.2.7. *bcryptjs*

O *bcryptjs* é uma *library* com recursos de encriptação. Funciona através de *hashing*, gerando bytes aleatórios que são combinados com a *password* do utilizador, criando *hashes* exclusivos das passwords. Se dois utilizadores tiverem a mesma *password*, não vão ter o mesmo *hash*, o que garante uma maior segurança. O *bcryptjs* também fornece recursos de comparação de *hashes* e *passwords*, que são usados no *login*, para verificar se a *password* inserida pelo utilizador pode ser validada com a *hash* guardada na base de dados referente a esse utilizador.

4.2.8. Google APIs

As *Google APIs* são um conjunto de *interfaces* de *APIs* desenvolvidas pela *Google* que permitem a comunicação com os Serviços da *Google* e integração dos mesmos noutros serviços. Destes conjuntos foram utilizados na aplicação a *Google Maps API* e a *Geocode API*. Os serviços do *Google Maps* foram usados na página inicial, onde é apresentado o mapa nacional com os terrenos que foram adicionados ao sistema, e na página de adicionar terreno, onde é apresentada uma ferramenta de desenho de polígonos no mapa do município escolhido. Os serviços de *Geocoding* foram utilizados para converter moradas em coordenadas geográficas, de modo a disponibilizar ao utilizador outros meios de navegação no mapa.

4.2.9. Leaflet

O *Leaflet* é uma *library JS* de código aberto muito usada em aplicações que contenham elementos de mapeamento. Lançado em 2011, permite o processamento de objetos *GeoJSON* de modo a renderizar mapas com diferentes camadas interativas sobrepostas sobre os mesmos. Na aplicação esta *library* foi usada para criar componentes onde é

apresentado o mapa nacional, com possibilidade de interação através da escolha de distritos/municípios e consulta de estatísticas de terrenos adicionados.

4.2.10. Vue I18n

O *Vue I18n* é um *plugin* de internacionalização para o *Vue*. Permite adaptar o *software* a várias línguas, através de um *setup* simples e fácil manutenção. Na aplicação foi usado para permitir ao utilizador a escolha entre o idioma português e inglês.

4.2.11. Git/GitHub

O *Git* é uma ferramenta de controlo de versões de código, criando um histórico de modificações. O *GitHub* é uma plataforma que permite o armazenamento desse código na *cloud*, oferecendo um sistema de controlo de versões, acessível em qualquer lado. Cada projeto é guardado num repositório, onde são armazenados todos os ficheiros. Por padrão o repositório cria automaticamente um ramo (*branch*) com o nome “*master*”, que é considerado o ramo definitivo. Outros *branches* são usados para fazer testes e edições antes de fazer *commit* para o *branch* *master*.

4.2.12. Heroku

O *Heroku* é um *Platform-as-a-Service* que oferece ferramentas de *deploy*, gestão e execução de aplicações desenvolvidas em *Ruby*, *Node.js*, *Java*, *Python*, *Scala*, *Go* e *PHP*. A sua integração com o *GitHub* possibilita uma fácil implementação de aplicações online, criando várias versões, o que acaba também por funcionar como *backup* redundante, visto o *GitHub* ser a alternativa predileta para *backups* e consultas de versões. No *Heroku* existem duas possibilidades de *deploy* através do *Github*:

- **Deploy manual** - opção adotada para este projeto, através de ordem manual, faz *deploy* imediato de qualquer *branch* do repositório *GitHub* ligado à aplicação. Também pode ser usado para implementar temporariamente um *branch* que não o configurado para *deploy* automático (para fins de teste);

- **Deploy automático** - qualquer novo *commit* para um *branch* do *GitHub*, é automaticamente implementado pelo *Heroku*. Existem medidas de precaução que podem ser ativadas, para limitar estes *deploys*, com base no contexto do *commit* que foi feito.

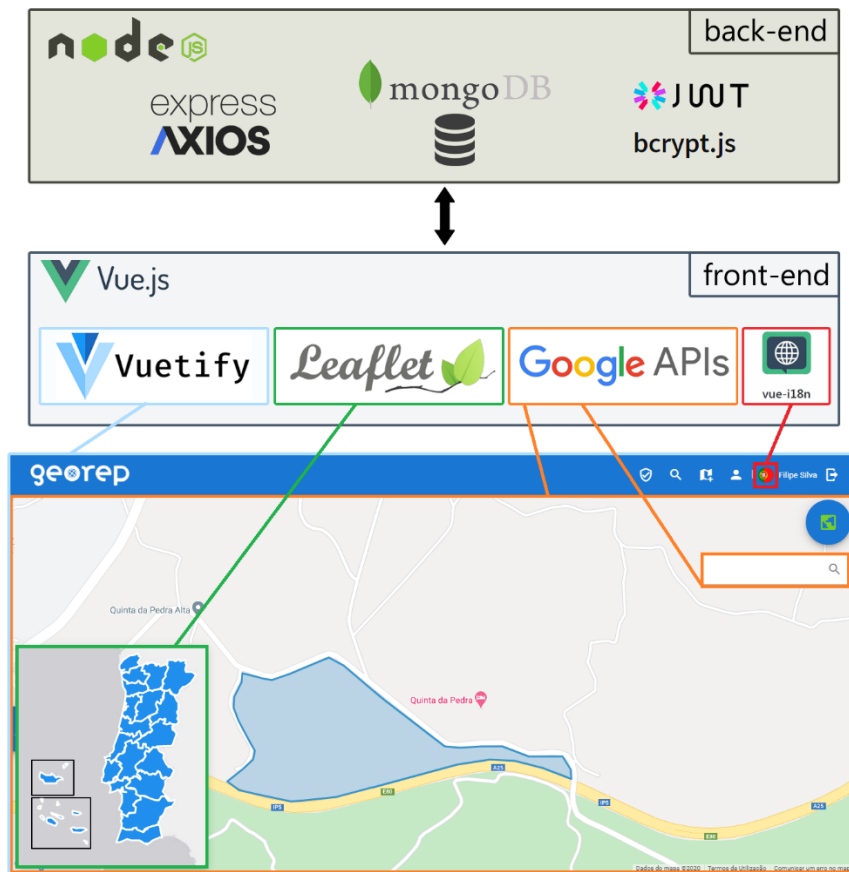


Figura 25 - Estruturação das tecnologias utilizadas

No capítulo seguinte serão abordados o planeamento e a concepção do projeto, descrevendo as suas funcionalidades, casos de uso e seu contexto, e diagramas referentes à instalação.

Capítulo 5 – Planeamento e concepção

Na fase de planeamento da aplicação foi importante ter em conta que usos é que os futuros utilizadores lhe vão dar, para isso foram feitos vários diagramas, de forma a que pudessem ser tiradas conclusões acerca do seu funcionamento.

5.1. Funcionalidades e objetivos

A aplicação “*GeoRep*” deverá poder interagir com três tipos de utilizadores: o Utilizador normal, que terá a capacidade de adicionar terrenos ao sistema (bem como editar a informação dos mesmos ou eliminá-los), requisitar validação, definir a visibilidade e fazer *upload* de imagens dos mesmos; o Manager, que também poderá fazer validações de terrenos e consultar dados de utilizadores que estejam localizados numa zona que lhe seja atribuída; e o Administrador, que para além de ter as mesmas permissões de um Manager, mas sem restrição de zona, poderá também alterar a categoria de utilizadores e eliminar contas. De salientar que as permissões no sistema funcionam de forma hierárquica: os utilizadores superiores, para além das permissões específicas do seu cargo, contam também com as dos utilizadores abaixo de si. Um Administrador ou um Manager também podem adicionar terrenos, apesar deste ser um caso de uso pensado para um Utilizador normal. As permissões de cada tipo de utilizador estão descritas na Tabela 3.

Tabela 3 – Permissões dos utilizadores

Caso de Uso \ Tipo de utilizador	Utilizador normal	Manager	Administrador
Adicionar/Editar/Eliminar terrenos	Verde	Verde	Verde
Requisitar validação dos seus terrenos	Verde	Verde	Verde
Definir visibilidade dos seus terrenos	Verde	Verde	Verde
Fazer upload de imagens dos seus terrenos	Verde	Verde	Verde
Pesquisar terrenos	Amarelo	Azul	Verde
Consultar/interagir com terrenos no mapa nacional	Amarelo	Azul	Verde
Validação de terrenos	Vermelho	Púrpura	Verde
Consultar dados de utilizadores	Vermelho	Púrpura	Verde
Eliminar outras contas	Vermelho	Vermelho	Verde
Alterar categoria de utilizadores	Vermelho	Vermelho	Verde

■ Não tem restrições
■ Apenas terrenos adicionados pelo utilizador ou com visibilidade pública
■ Apenas terrenos adicionados pelo utilizador, com visibilidade pública ou dentro da zona atribuída
■ Apenas na zona atribuída
■ Sem permissão

5.2. Desenho e modelação

De modo a estruturar e clarificar os atributos descritos anteriormente, foram usadas práticas, diagramas e símbolos gráficos de modelação descritos na linguagem *UML*, de forma a uniformizar o projeto através do emprego de componentes genéricos que contribuem para a melhor compreensão e detalhamento dos objetos e comunicação estipulada entre eles.

5.2.1 Diagrama de contexto

O diagrama seguinte demonstra o intuito e objetivo do projeto através de um esquema. No que toca ao desenvolvimento de *software*, é considerado o diagrama de fluxo de dados de maior nível, ou seja, o diagrama que representa todo o sistema. Demonstra como todas as entidades interagem com o sistema indicando as suas possíveis ações. No diagrama de contexto não é considerada a estrutura interna do sistema, sendo o seu principal objetivo a deteção de limitações/requisitos. Através dos requisitos enunciados anteriormente é então praticável a esquematização do diagrama da Figura 26 abaixo, onde se descreve a premissa da aplicação, com a indicação da direção dos movimentos de comunicação entre o sistema e os atores.

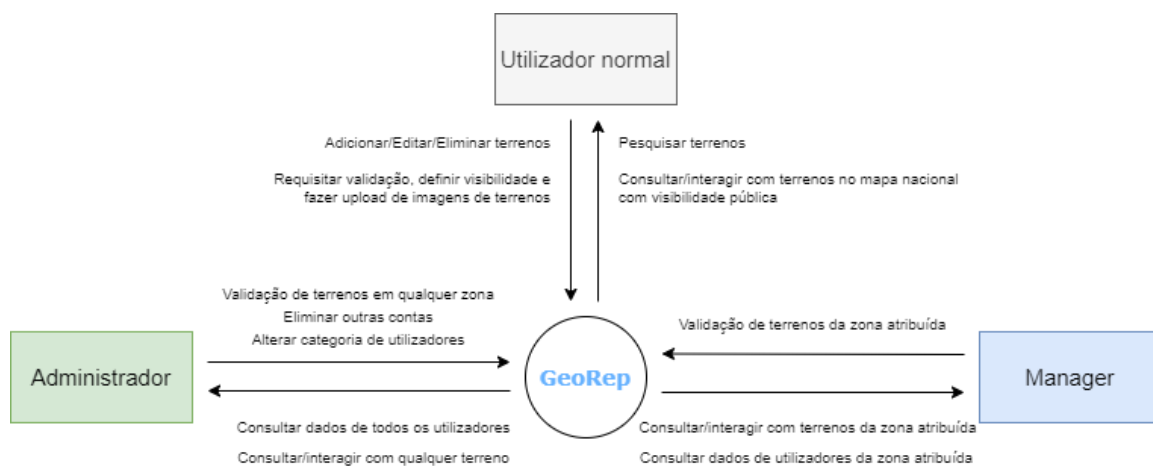


Figura 26 – Diagrama de contexto

5.2.2 Diagrama de casos de uso

Este diagrama documenta o que o sistema faz do ponto de vista do utilizador. Noutras palavras, descreve a relação das funcionalidades do sistema com os utilizadores. Tal como no diagrama anterior, neste também não são aprofundados detalhes técnicos do sistema. Os diagramas de casos de uso são compostos basicamente por quatro partes:

- **Cenário** - sequência de eventos que acontecem quando um utilizador interage com o sistema;
- **Ator** - tipo de utilizador do sistema;
- **Caso de uso** - tarefa ou uma funcionalidade realizada por um ator;
- **Comunicação** - é o que liga um ator a um caso de uso.

De referir que o *login* não é apresentado como um caso de uso porque não é uma funcionalidade de valor que o sistema em questão ofereça aos seus atores, visto que utilizadores sem login apenas podem fazer o registo, servindo apenas para autenticação. Também se destaca o conceito <<extend>>, que significa que quando o caso de uso estendido tem a sua funcionalidade executada, o caso de uso que estende poderá opcionalmente ser executado também.

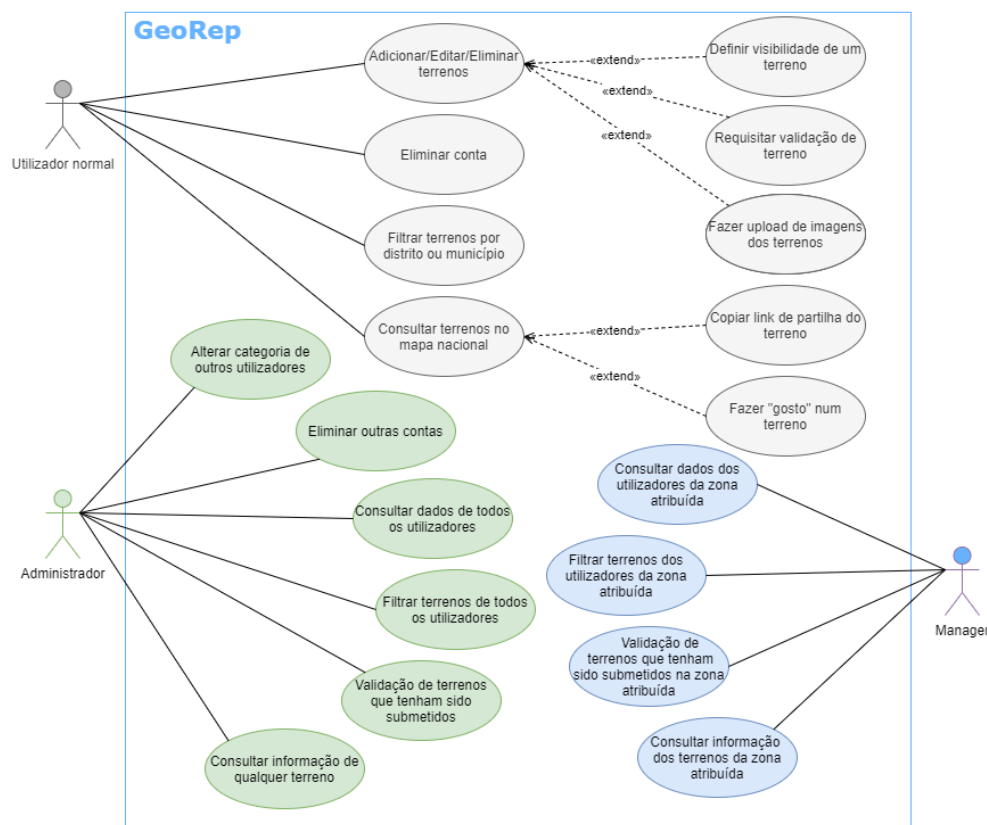


Figura 27 - Diagrama de casos de uso

5.2.3 Descrição de casos de uso e diagramas de sequência

A descrição de casos de uso, para além do já mencionado, permite achar caminhos, trajetos ou cenários alternativos e momentos de exceção. A descrição sequencial possibilita a descoberta de pré-condições ou requisitos anteriores à ocorrência, bem como a descoberta de sucessos que podem ocorrer após o término do caso de uso. Os casos de uso estão organizados em oito secções: nome, objetivo, atores envolvidos, pré-condição, fluxo principal, fluxos alternativos, pós-condição e casos de teste.

5.2.3.1 Adicionar terreno

A ideia original deste projeto teve como ponto orientador dois importantes casos de uso, sendo este um deles. A Tabela 4 descreve o caso de uso em que um utilizador adiciona um novo terreno no sistema. Pretende-se, portanto, dar ênfase a este caso, visto que será uma das funcionalidades mais frequentes da aplicação. É fundamental que este seja um processo simples, logo tentou-se diminuir ao máximo os passos que o utilizador tem de dar durante a introdução do terreno no sistema. Num cenário normal apenas terá de preencher a informação do terreno e desenhar o polígono no mapa. Opcionalmente o utilizador pode também fazer o *upload* de imagens do terreno, e mudar a sua visibilidade (predefinida como pública).

Tabela 4 - Caso de uso: Adicionar terreno

Nome	Adicionar terreno
Objetivo	Adicionar um novo terreno
Atores envolvidos	Utilizador normal, Manager ou Administrador
Pré-condição	Login válido
Fluxo Principal	<ol style="list-style-type: none"> 1. O utilizador acede à página de adicionar terreno. 2. O sistema abre a página de adicionar terreno no separador "Informação". 3. O utilizador preenche a informação do terreno. 4. O utilizador abre o separador "Mapa". 5. O sistema devolve o mapa de distritos/municípios. 6. O utilizador seleciona o município. 7. O sistema abre o mapa de desenho no município selecionado. 8. O utilizador desenha o terreno. 9. O utilizador conclui o desenho. 10. O sistema abre o separador "Submeter". 11. O utilizador submete o terreno. 12. O sistema devolve mensagem de sucesso.
Fluxos alternativos	<ol style="list-style-type: none"> 3a. O utilizador pode nesta altura mudar a visibilidade predefinida do terreno, de pública para privada, usando o botão presente para esse efeito no separador "Informação", como é descrito no caso de uso "Definir visibilidade de um terreno". 3b. O utilizador pode nesta altura optar por fazer upload de imagens do terreno, descrito no caso de uso "Fazer upload de imagens de terrenos". 3c. O utilizador pode nesta altura optar por requisitar a validação do terreno, abrindo o separador "Validação", descrito no caso de uso "Requisitar validação de terreno". 9a. O terreno tem menos de 3 pontos, o utilizador não pode concluir e deve continuar o desenho. 11a. O nome do terreno não está preenchido corretamente (min. 3 caracteres), o sistema volta ao separador "Informação".
Pós-condição	Após redirecionamento para a página de perfil, a tabela que contém os terrenos adicionados pelo utilizador contém o novo terreno.
Casos de teste	<ol style="list-style-type: none"> 1. Verificar se nome do terreno é válido (min. 3 caracteres). 2. Verificar se desenho do terreno tem no mínimo 3 pontos.

Dada a análise anterior torna-se possível chegar a um diagrama de sequência que resulta da descrição do caso de uso, onde figura a página "AddLand", o componente de escolha do município e o componente que contém o mapa e as ferramentas de desenho para o polígono que formará o terreno.

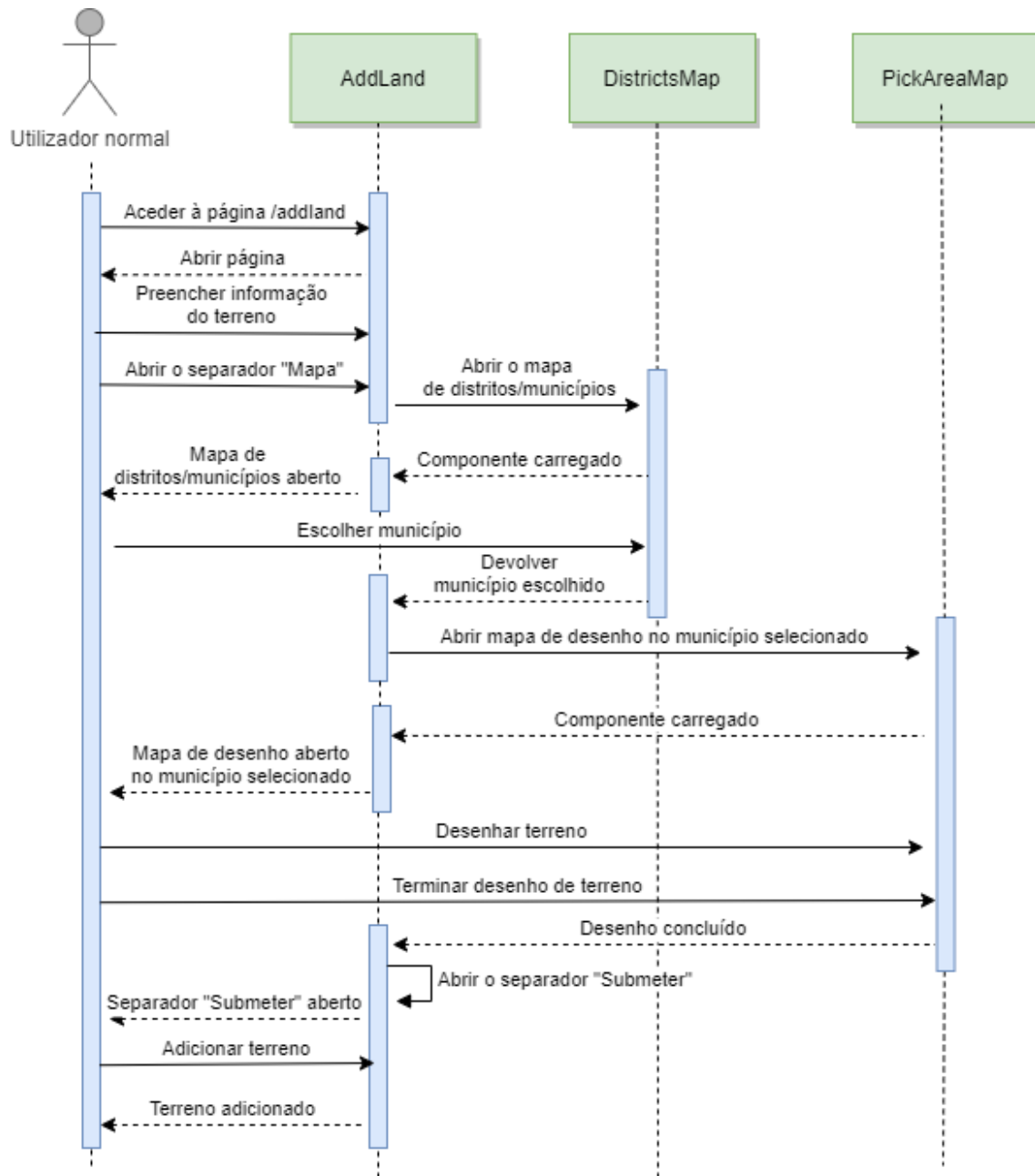


Figura 28 - Diagrama de sequência: Adicionar terreno

5.2.3.2 Fazer upload de imagens de terrenos (Adicionar terreno)

Como referido nos fluxos alternativos do caso de uso anterior, o utilizador pode, aquando do preenchimento da informação do terreno, fazer o *upload* de imagens do mesmo. A Tabela 5 descreve o caso de uso em que o utilizador faz o *upload* destas imagens. Para isto apenas precisa de seleccionar os ficheiros que pretende carregar, assim que é aberta a janela do gestor de ficheiros. O utilizador pode também fazer o *upload* de imagens quando está a editar o terreno, como está descrito nos fluxos alternativos do caso de uso "Editar terreno".

Tabela 5 - Caso de uso: Fazer upload de imagens de terrenos

Nome	Fazer upload de imagens de terrenos (Adicionar terreno)
Objetivo	Fazer upload de imagens descritivas de um terreno no momento em que este é adicionado
Atores envolvidos	Utilizador normal, Manager ou Administrador
Pré-condição	Login válido
Fluxo Principal	<ol style="list-style-type: none"> 1. O utilizador acede à página de adicionar terreno. 2. O sistema abre a página de adicionar terreno no separador "Informação". 3. O utilizador preenche a informação do terreno. 4. O utilizador clica no botão "Carregar imagens". 5. O sistema devolve uma janela para o utilizador escolher quais as imagens que pretende fazer upload. 6. O utilizador escolhe as imagens e faz o upload. 7. O utilizador abre o separador "Mapa". 8. O sistema devolve o mapa de distritos/municípios. 9. O utilizador selecciona o município. 10. O sistema abre o mapa de desenho no município seleccionado. 11. O utilizador desenha o terreno. 12. O utilizador conclui o desenho. 13. O sistema abre o separador "Submeter". 14. O utilizador submete o terreno. 15. O sistema devolve mensagem de sucesso.
Fluxos alternativos	<ol style="list-style-type: none"> 3a. O utilizador pode nesta altura mudar a visibilidade predefinida do terreno, de pública para privada, usando o botão presente para esse efeito no separador "Informação", como é descrito no caso de uso "Definir visibilidade de um terreno". 3b. O utilizador pode nesta altura optar por requisitar a validação do terreno, abrindo o separador "Validação", descrito no caso de uso "Requisitar validação de terreno". 12a. O terreno tem menos de 3 pontos, o utilizador não pode concluir e deve continuar o desenho. 14a. O nome do terreno não está preenchido corretamente (min. 3 caracteres), o sistema volta ao separador "Informação".
Pós-condição	Após redireccionamento para a página de perfil, a tabela que contém os terrenos adicionados pelo utilizador contém o novo terreno.
Casos de teste	<ol style="list-style-type: none"> 1. Verificar se nome do terreno é válido (min. 3 caracteres). 2. Verificar se os ficheiros escolhidos pelo utilizador são imagens, caso contrário não é feito o upload. 3. Verificar se desenho do terreno tem no mínimo 3 pontos.

No diagrama de sequência seguinte estão então explícitos os mesmos componentes usados no caso de uso "Adicionar terreno", mas desta vez com a inclusão do componente "FileInput", criado para dar ao utilizador uma visão simples dos ficheiros que pretende carregar.

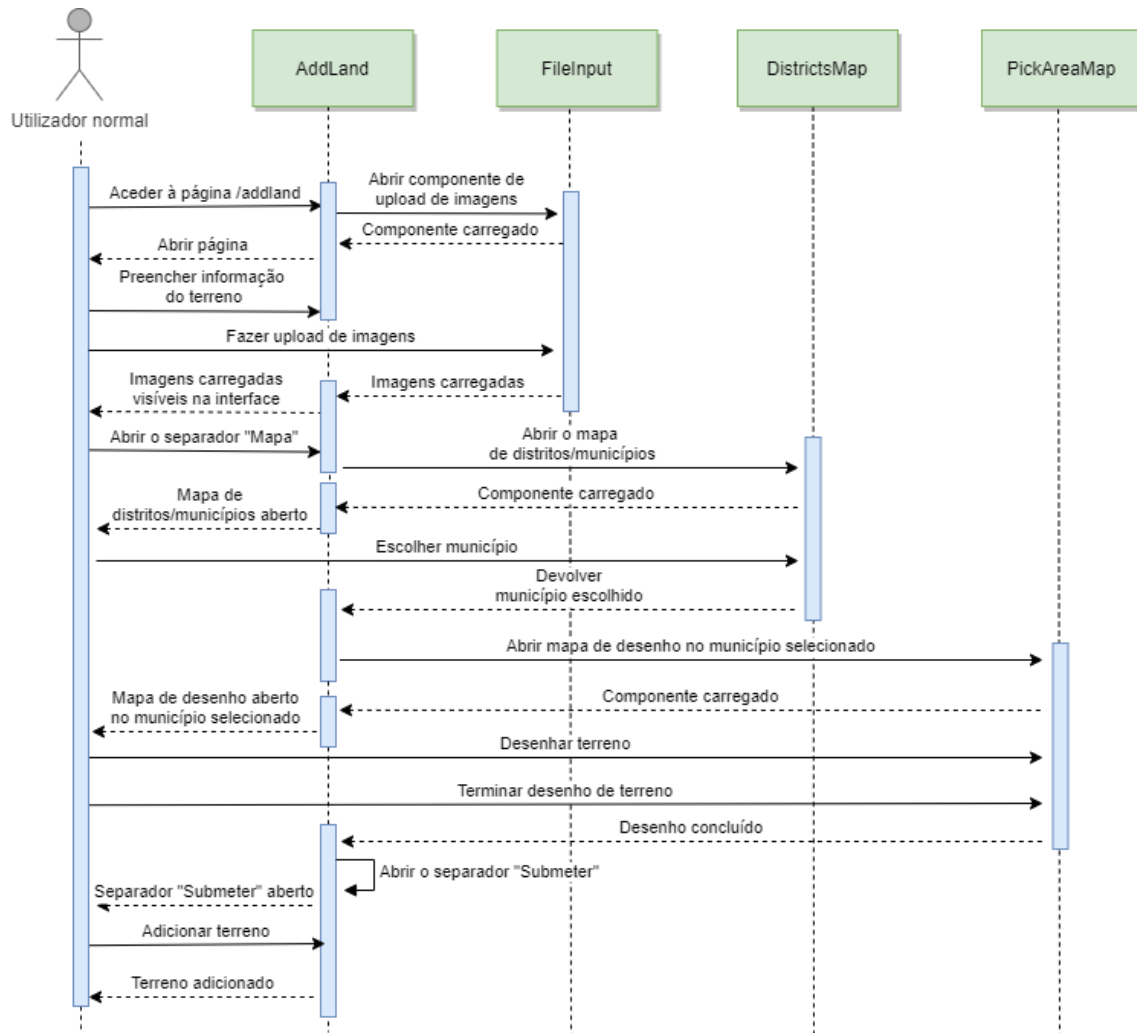


Figura 29 – Diagrama de sequência: Fazer upload de imagens de terrenos

5.2.3.3 Requisitar validação de terreno

No momento da adição do terreno o utilizador pode também requisitar que o terreno seja validado. Se for o caso, qualquer pessoa que aceda ao terreno terá a confirmação de que se trata de um registo legítimo. A Tabela 6 descreve o caso de uso em que o utilizador faz a requisição da validação do terreno. Para isto o utilizador precisa de aceder ao separador "Validação" e escolher que documentos pretende carregar, da mesma forma que faz no caso de uso descrito anteriormente para as imagens.

Tabela 6 - Caso de uso: Requisitar validação de terreno

Nome	Requisitar validação de terreno
Objetivo	Requisitar validação de um terreno
Atores envolvidos	Utilizador normal, Manager ou Administrador
Pré-condição	Login válido
Fluxo Principal	<ol style="list-style-type: none"> 1. O utilizador acede à página de adicionar terreno. 2. O sistema abre a página de adicionar terreno no separador "Informação". 3. O utilizador preenche a informação do terreno. 4. O utilizador abre o separador "Validação". 5. O sistema devolve o separador "Validação". 6. O utilizador clica no botão "Carregar ficheiros". 7. O sistema devolve uma janela para o utilizador escolher quais os ficheiros (documentos de validação) que pretende fazer upload. 8. O utilizador escolhe os ficheiros e faz o upload. 9. O utilizador abre o separador "Mapa". 10. O sistema devolve o mapa de distritos/municípios. 11. O utilizador seleciona o município. 12. O sistema abre o mapa de desenho no município selecionado. 13. O utilizador desenha o terreno. 14. O utilizador conclui o desenho. 15. O sistema abre o separador "Submeter". 16. O utilizador submete o terreno. 17. O sistema devolve mensagem de sucesso.
Fluxos alternativos	<ol style="list-style-type: none"> 3a. O utilizador pode nesta altura mudar a visibilidade predefinida do terreno, de pública para privada, usando o botão presente para esse efeito no separador "Informação", como é descrito no caso de uso "Definir visibilidade de um terreno". 3b. O utilizador pode nesta altura optar por fazer upload de imagens do terreno, descrito no caso de uso "Fazer upload de imagens de terrenos". 14a. O terreno tem menos de 3 pontos, o utilizador não pode concluir e deve continuar o desenho. 16a. O nome do terreno não está preenchido corretamente (min. 3 caracteres), o sistema volta ao separador "Informação".
Pós-condição	Após redirecionamento para a página de perfil, a tabela que contém os terrenos adicionados pelo utilizador contém o novo terreno.
Casos de teste	<ol style="list-style-type: none"> 1. Verificar se nome do terreno é válido (min. 3 caracteres). 2. Verificar se os ficheiros escolhidos pelo utilizador são documentos válidos, caso contrário não é feito o upload. 3. Verificar se desenho do terreno tem no mínimo 3 pontos.

Os componentes presentes no seguinte diagrama de sequência são os mesmos do caso de uso anterior, mas aplicados no caso da requisição de validação do terreno.

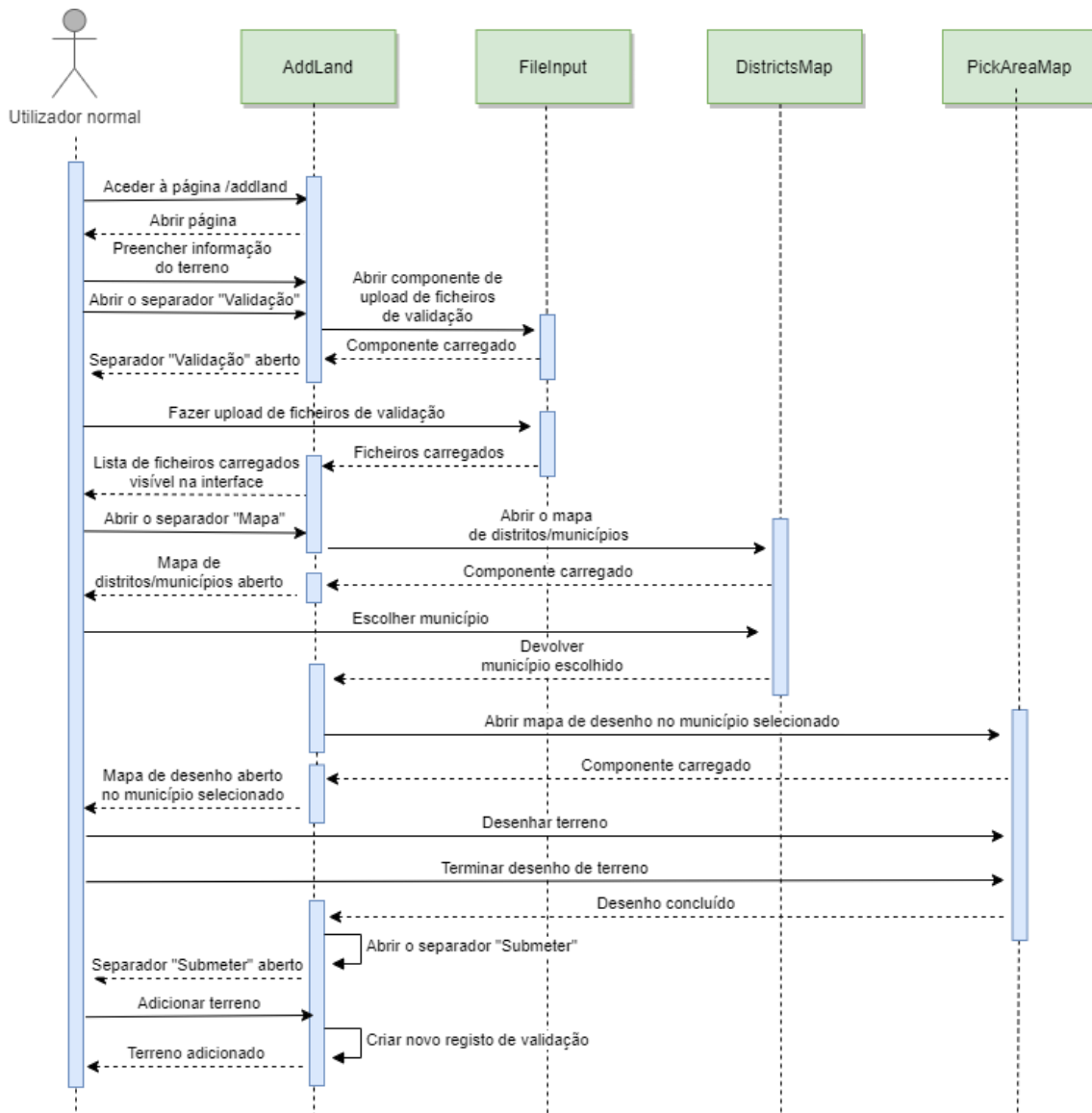


Figura 30 – Diagrama de sequência: Requisitar validação de terreno

5.2.3.4 Eliminar conta

O utilizador pode eliminar a sua conta, acedendo à página de perfil e clicando no botão "Eliminar conta". Sendo esta uma escolha que deve ser tomada com consciência, a *dialog* de confirmação obriga a que o utilizador insira o seu *email*, para evitar uma eliminação acidental. O utilizador, caso não pretenda que os terrenos adicionados por si até então fiquem visíveis no mapa, tem neste momento a opção de eliminar também os seus terrenos. Caso a eliminação da conta seja confirmada, o utilizador é redirecionado para a página inicial.

Tabela 7 - Caso de uso: Eliminar conta

Nome	Eliminar conta
Objetivo	Eliminar a própria conta e, como opção, apagar todos os terrenos adicionados pelo utilizador.
Atores envolvidos	Utilizador normal, Manager ou Administrador
Pré-condição	Login válido
Fluxo Principal	<ol style="list-style-type: none"> 1. O utilizador acede à página de perfil. 2. O sistema abre a página de perfil. 3. O utilizador clica no botão "Eliminar conta". 4. O sistema abre a dialog de eliminação de conta. 5. O utilizador insere o seu email para confirmar a eliminação. 6. O utilizador confirma a eliminação. 7. O sistema termina a sessão e elimina a conta.
Fluxos alternativos	5a. O utilizador pode nesta altura escolher a opção "Eliminar terrenos também" se pretender que os seus terrenos também sejam eliminados.
Pós-condição	Redirecionamento para a Home (página inicial).
Casos de teste	1. Verificar se o email inserido é igual ao email do utilizador.

No diagrama de sequência resultante deste caso de uso, são visíveis o componente da página de perfil e a *dialog* de eliminação da conta.

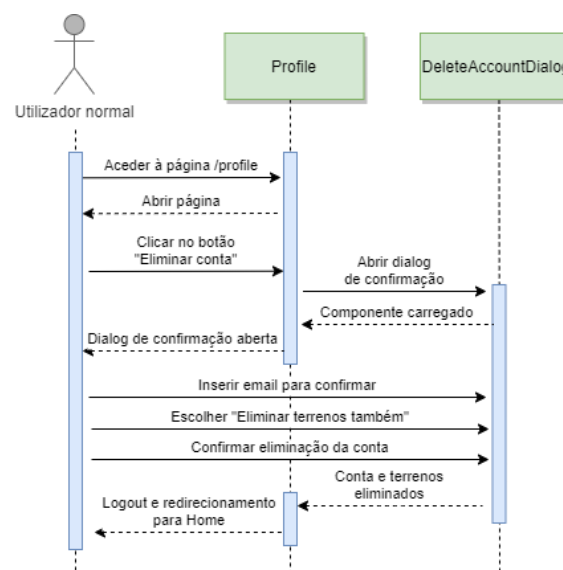


Figura 31 - Diagrama de sequência: Eliminar conta

5.2.3.5 Eliminar terreno

Caso o utilizador pretenda eliminar do sistema um terreno adicionado por si, deve aceder à página de perfil e clicar no botão "Eliminar" do terreno pretendido. Surgirá uma *dialog* de confirmação e, uma vez aceite, o terreno e todos os ficheiros referentes ao mesmo serão eliminados do sistema.

Tabela 8 - Caso de uso: Eliminar terreno

Nome	Eliminar terreno
Objetivo	Eliminar terreno do utilizador.
Atores envolvidos	Utilizador normal, Manager ou Administrador
Pré-condição	Login válido e ter algum terreno adicionado.
Fluxo Principal	<ol style="list-style-type: none"> 1. O utilizador acede à página de perfil. 2. O sistema abre a página de perfil. 3. O utilizador clica no botão "Eliminar" de um terreno. 4. O sistema abre a dialog de eliminação de terreno. 5. O utilizador confirma a eliminação. 6. O sistema devolve mensagem de sucesso.
Fluxos alternativos	Não existem.
Pós-condição	A tabela que contém os terrenos adicionados pelo utilizador já não contém o terreno.
Casos de teste	Não existem.

No diagrama de sequência resultante deste caso de uso, são visíveis o componente da página de perfil e a *dialog* de eliminação de terreno.

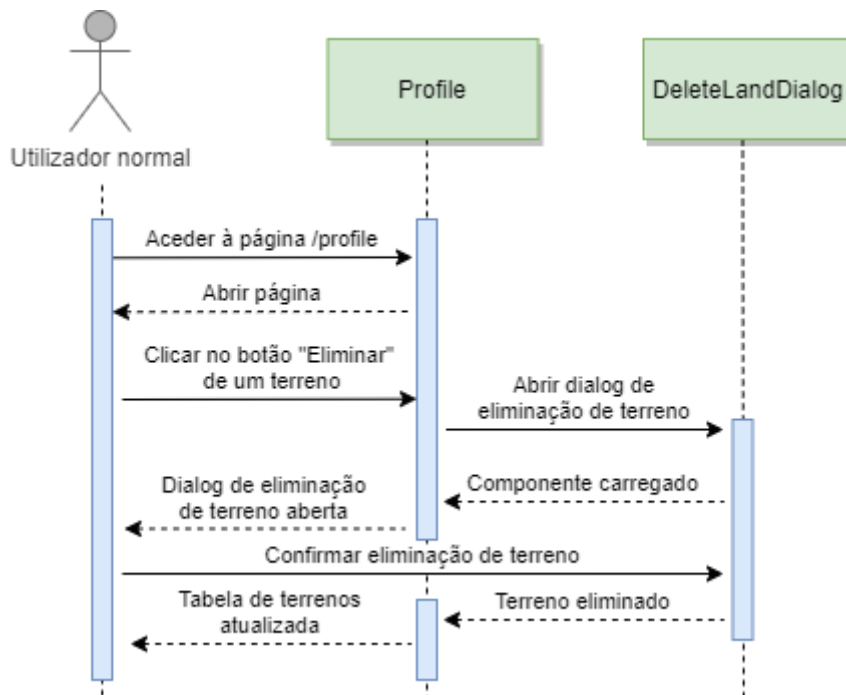


Figura 32 - Diagrama de sequência: Eliminar terreno

5.2.3.6 Eliminar outras contas

Os Administradores têm a possibilidade de eliminar contas do sistema. Para isso devem consultar a lista de utilizadores na página "Explorar" e escolher qual o utilizador que pretendem remover. À semelhança do que acontece quando se elimina a própria conta, o utilizador tem de inserir o *email* do utilizador (visível como *placeholder*) antes de confirmar a eliminação.

Tabela 9 - Caso de uso: Eliminar outras contas

Nome	Eliminar outras contas
Objetivo	Eliminar conta de outro utilizador que não o atual e, como opção, apagar todos os terrenos adicionados pelo utilizador.
Atores envolvidos	Administrador
Pré-condição	Login válido e permissões de administrador.
Fluxo Principal	<ol style="list-style-type: none"> 1. O utilizador acede à página "Explorar". 2. O sistema abre a página "Explorar". 3. O utilizador clica no botão "Utilizadores". 4. O sistema devolve a lista de utilizadores. 5. O utilizador clica no botão "Eliminar utilizador" do utilizador pretendido. 6. O sistema abre a dialog de eliminação de conta. 7. O utilizador insere o email para confirmar a eliminação. 8. O utilizador confirma a eliminação. 9. O sistema devolve mensagem de sucesso.
Fluxos alternativos	7a. O utilizador pode nesta altura escolher a opção "Eliminar terrenos também" se pretender que os terrenos do utilizador que vai ser eliminado também sejam eliminados.
Pós-condição	A tabela que contém os utilizadores do sistema já não contém o utilizador eliminado.
Casos de teste	1. Verificar se o email inserido é igual ao email do utilizador.

No diagrama de sequência resultante deste caso de uso, são visíveis o componente da página "Explorar" e a *dialog* de eliminação de conta.

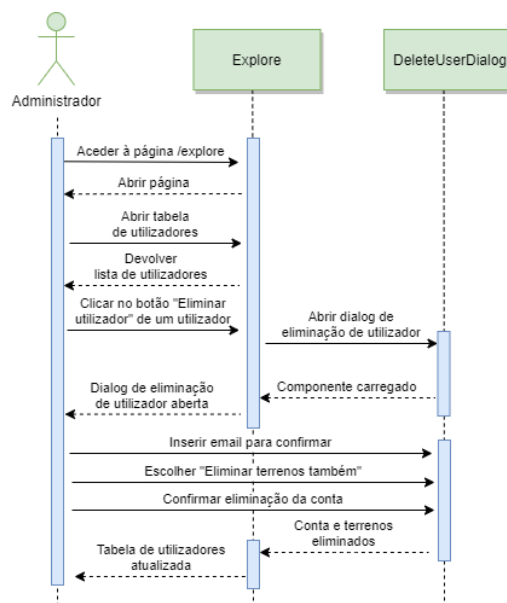


Figura 33 - Diagrama de sequência: Eliminar outras contas

5.2.3.7 Editar terreno

Acedendo à página de perfil, o utilizador pode escolher um dos seus terrenos e editar a sua informação. Para isso terá de clicar no botão "Editar" de um terreno e a *dialog* de edição tornar-se-á visível contendo os campos do terreno bem como uma pré-visualização no mapa. Para além disso, na mesma janela também é possível mudar a sua visibilidade e as imagens anexadas ao mesmo.

Tabela 10 - Caso de uso: Editar terreno

Nome	Editar terreno
Objetivo	Editar informação de um terreno, e opcionalmente alterar a visibilidade do terreno.
Atores envolvidos	Utilizador normal, Manager ou Administrador
Pré-condição	Login válido e ter algum terreno adicionado.
Fluxo Principal	1. O utilizador acede à página de perfil. 2. O sistema abre a página de perfil. 3. O utilizador clica no botão "Editar" de um terreno. 4. O sistema abre a dialog de edição de terreno. 5. O utilizador edita a informação do terreno. 8. O utilizador confirma a edição. 9. O sistema devolve mensagem de sucesso.
Fluxos alternativos	5a. O utilizador pode nesta altura mudar a visibilidade do terreno, usando o botão presente para esse efeito. 5b. O utilizador pode nesta altura optar por fazer upload de imagens do terreno, descrito no caso de uso "Fazer upload de imagens de terrenos". 8a. O nome do terreno não está preenchido corretamente (min. 3 caracteres), o sistema não permite a conclusão da edição.
Pós-condição	A tabela que contém os terrenos adicionados pelo utilizador é atualizada com os novos dados.
Casos de teste	1. Verificar se nome do terreno é válido (min. 3 caracteres). 2. Verificar se os ficheiros escolhidos pelo utilizador são imagens, caso contrário não é feito o upload.

Abaixo são mostrados dois diagramas de sequência do caso de uso apresentado. O primeiro para uma edição normal da informação do terreno, e o segundo contendo o fluxo alternativo referente ao carregamento de imagens.

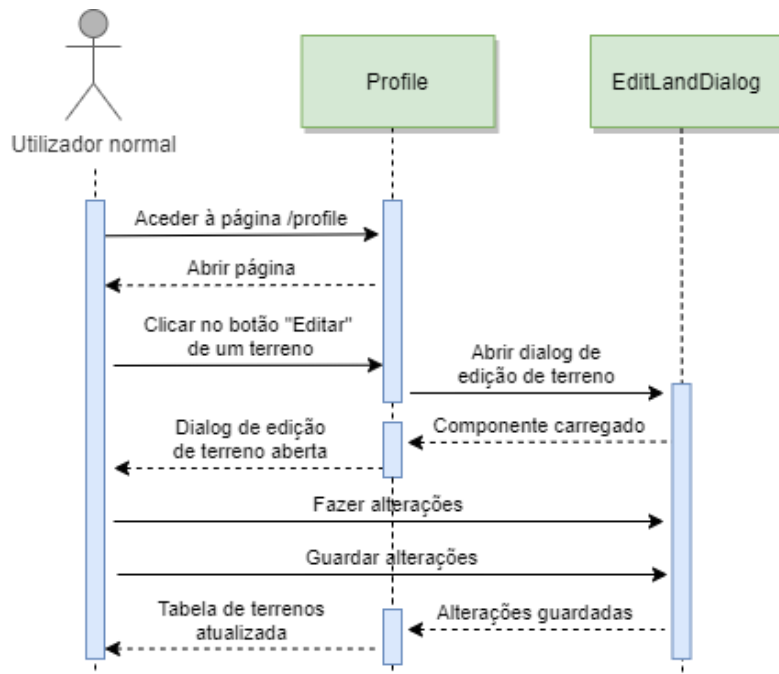


Figura 34 - Diagrama de sequência: Editar terreno

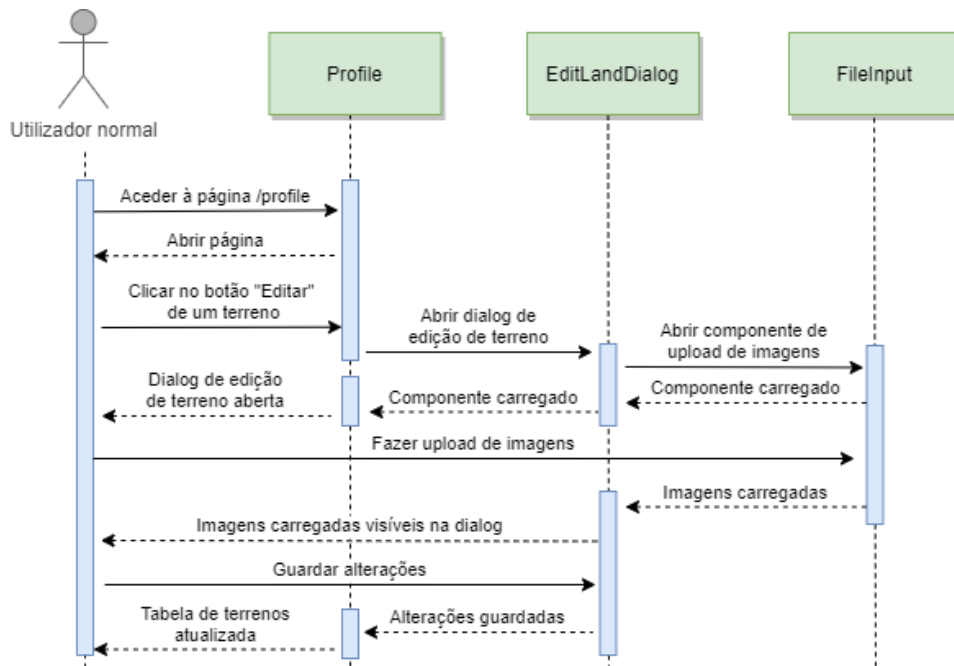


Figura 35 - Diagrama de sequência: Editar terreno (Carregar imagens)

5.2.3.8 Filtrar terrenos por distrito ou município

O componente "*DistrictsMap*" foi criado como uma maneira simples e apelativa de escolher distritos e municípios e é usado em vários pontos da aplicação. Excetuando no caso de uso "Adicionar terreno", este componente é usado para filtrar registos consoante o território escolhido. Para filtrar o utilizador apenas precisa de clicar no distrito/município que pretende usar como parâmetro de pesquisa, ou clicar no botão "Voltar" para anular o território escolhido.

Tabela 11 - Caso de uso: Filtrar terrenos por distrito ou município

Nome	Filtrar terrenos por distrito ou município
Objetivo	Filtrar terrenos de uma lista através do distrito ou município onde o terreno está localizado.
Atores envolvidos	Utilizador normal, Manager ou Administrador
Pré-condição	Login válido.
Fluxo Principal	<ol style="list-style-type: none"> 1. O utilizador acede à página "Explorar". 2. O sistema abre a página "Explorar". 3. O utilizador clica num distrito do mapa. 4. O sistema filtra os terrenos através do distrito selecionado. 5. O utilizador clica num município do mapa. 6. O sistema filtra os terrenos através do município selecionado.
Fluxos alternativos	Não existem.
Pós-condição	Não existe.
Casos de teste	Não existem.

No diagrama de sequência abaixo é usada a página "Explorar" como exemplo de demonstração de funcionamento do componente "*DistrictsMap*" no caso de ser usado para filtrar registos.

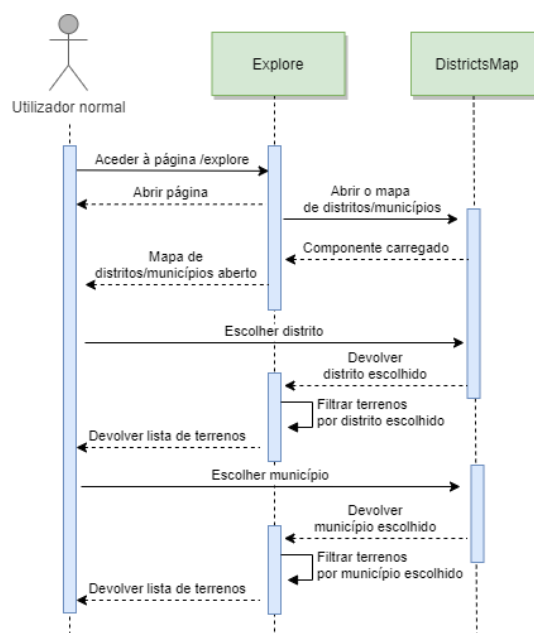


Figura 36 - Diagrama de sequência: Filtrar terrenos por distrito ou município

5.2.3.9 Consultar terreno no mapa nacional

Este trata-se do segundo caso de uso mais importante relativamente à orientação e objetivo principal deste projeto: a consulta dos terrenos adicionados no mapa nacional. Existem várias maneiras de consultar um terreno específico no mapa, sendo que a mais usual é a que é descrita como fluxo principal: através da página "Explorar", clicando no botão "Abrir" do terreno pretendido. Alternativamente o utilizador também pode consultar um terreno através da sua página de perfil (caso o terreno tenha sido adicionado por si), ou através de um *link* de partilha. Quando um utilizador está a consultar um terreno no mapa, tem também a possibilidade de fazer "Gosto" e de criar um *link* de partilha desse terreno.

Tabela 12 - Caso de uso: Consultar terreno no mapa nacional

Nome	Consultar terreno no mapa nacional
Objetivo	Consultar um terreno numa interface com mapa nacional
Atores envolvidos	Utilizador normal, Manager ou Administrador
Pré-condição	Login válido.
Fluxo Principal	1. O utilizador acede à página "Explorar". 2. O sistema abre a página "Explorar". 3. O utilizador clica no botão "Abrir" de um terreno. 4. O sistema abre a página do mapa nacional com o terreno selecionado.
Fluxos alternativos	1a. Em vez de especificar o terreno selecionado através da página "Explorar", o utilizador pode selecionar um terreno adicionado por si na página de perfil. 1b. Em vez de especificar o terreno selecionado através da página "Explorar", o utilizador pode selecionar um terreno através de um link de partilha. 4a. A qualquer altura o utilizador pode selecionar qualquer outro terreno visível no mapa. 4b. O utilizador pode fazer "gosto" no terreno selecionado, ou tirar o seu "gosto". 4c. O utilizador pode gerar e copiar para a área de transferência o link de partilha do terreno selecionado.
Pós-condição	Não existe.
Casos de teste	1. Caso o utilizador use um link de partilha para abrir o terreno, verificar se o terreno é válido, visto que o link pode ser alterado.

O diagrama de sequência abaixo apresenta as três maneiras que o utilizador tem de consultar um terreno no mapa. Com a letra "A" está apresentada a sequência principal onde o utilizador seleciona um terreno através da página "Explorar". Na sequência "B" é mostrado o caso em que o utilizador seleciona um dos seus terrenos na página de perfil, e o caso "C" mostra a situação em que o utilizador seleciona um terreno através de um *link* de partilha.

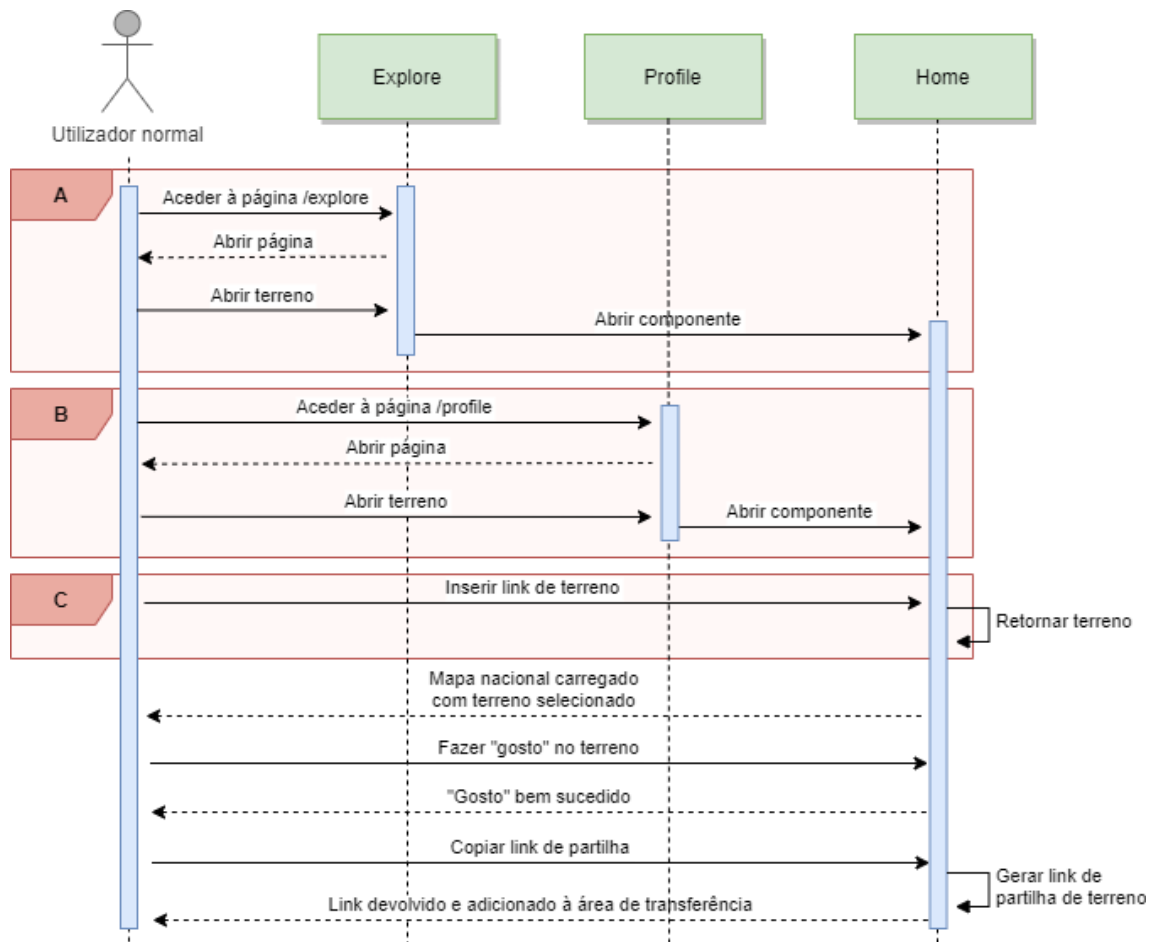


Figura 37 - Diagrama de sequência: Consultar terreno no mapa nacional

5.2.3.10 *Alterar a categoria de outro utilizador*

O Administrador pode alterar a categoria de qualquer utilizador, tornando-o num Utilizador normal, num Manager (de distrito ou município) ou também num Administrador. Para isso deve aceder à página "Explorar" e clicar no separador "Utilizadores". Ao clicar no ícone do utilizador que pretende, será aberta uma *dialog* que permite a mudança de categoria.

Tabela 13 - Caso de uso: Alterar a categoria de outro utilizador

Nome	Alterar a categoria de outro utilizador
Objetivo	Alterar a categoria de um utilizador, definindo um cargo de gestão (Administrador ou Manager), ou removendo o cargo de gestão (Utilizador normal).
Atores envolvidos	Administrador
Pré-condição	Login válido e permissões de administrador.
Fluxo Principal	<ol style="list-style-type: none"> 1. O utilizador acede à página "Explorar". 2. O sistema abre a página "Explorar". 3. O utilizador clica no botão "Utilizadores". 4. O sistema devolve a lista de utilizadores. 5. O utilizador clica no ícone de categoria do utilizador pretendido. 6. O sistema abre a dialog de definição de categoria de utilizador. 7. O utilizador remove o cargo de gestão, para definir o utilizador selecionado como Utilizador normal. 8. O utilizador confirma a definição. 9. O sistema devolve mensagem de sucesso.
Fluxos alternativos	<ol style="list-style-type: none"> 7a. O utilizador pode nesta altura optar por confirmar a categoria predefinida, o que dará ao utilizador selecionado a categoria de Administrador. 7b. O utilizador pode nesta altura escolher um distrito para definir a zona de gestão do utilizador que ficará com a categoria de Manager. 7b.a. O utilizador pode nesta altura escolher um município para definir a zona de gestão do utilizador que ficará com a categoria de Manager.
Pós-condição	O registo da tabela que contém os utilizadores do sistema é atualizado.
Casos de teste	Não existe.

O diagrama de sequência abaixo apresenta os vários caminhos que podem ser percorridos para alterar a categoria de um utilizador. Uma vez que a categoria predefinida da *dialog* é a de Administrador, neste caso o Administrador apenas precisa de confirmar a mudança de categoria. Com a letra "A" está apresentado o caso em que o Administrador escolhe um distrito para que o utilizador pretendido seja Manager. Caso pretenda atribuir ao utilizador um município (precisando de escolher primeiro o distrito), o quadro correspondente é o "B", que engloba o "A". No caso "C" é descrita a remoção do cargo de gestão o que fará com que o utilizador passe a ser um Utilizador normal.

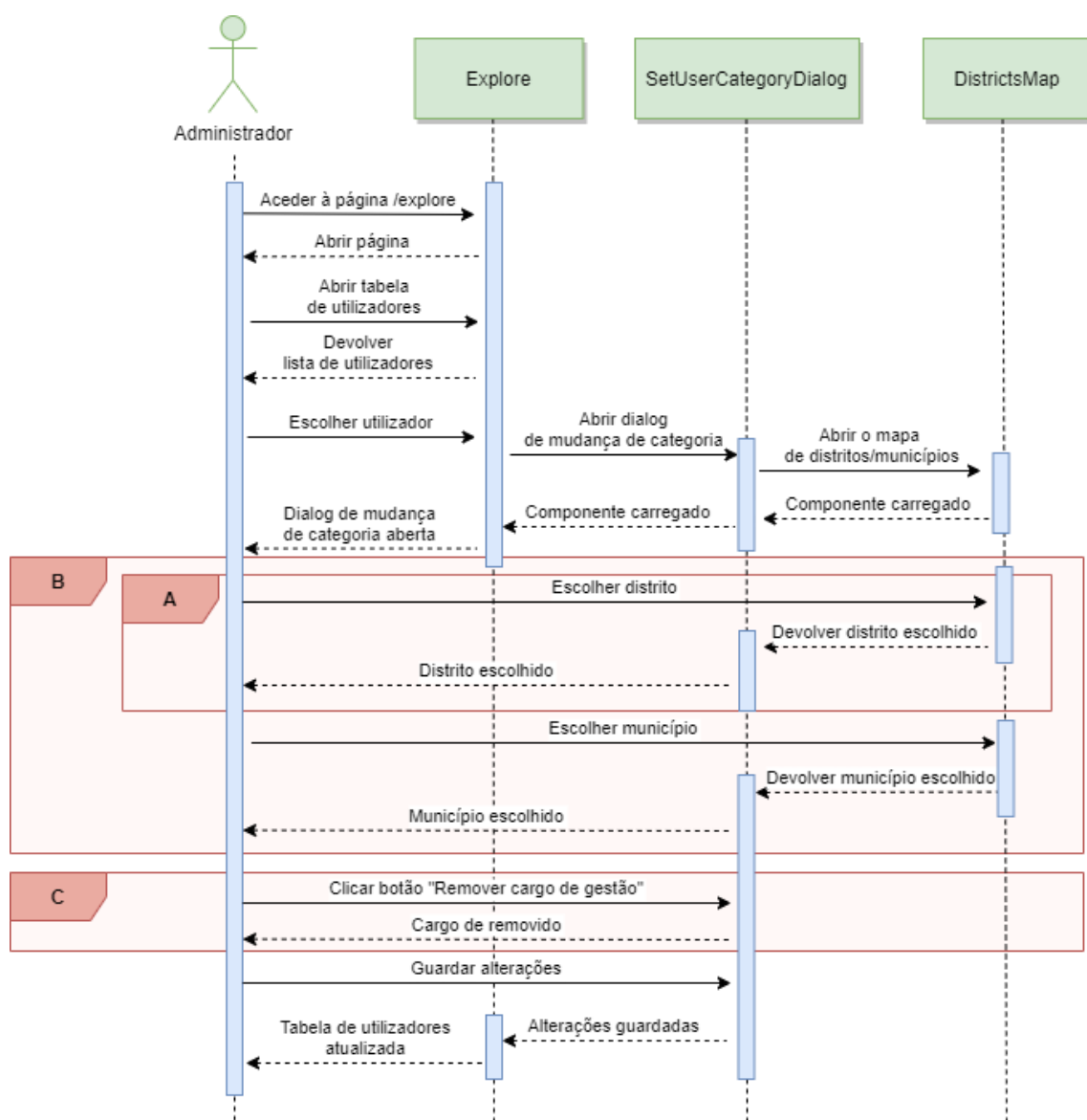


Figura 38 - Diagrama de sequência: Alterar a categoria de outro utilizador

5.2.3.11 Filtrar terrenos por utilizador

Os Administradores e Managers podem filtrar os registos de terrenos através do utilizador que os adicionou. Para isso devem aceder à página "Explorar" e clicar no separador "Utilizadores". Aqui verão a lista de utilizadores do sistema. No caso do Manager apenas são visíveis utilizadores que tenham terrenos na sua zona de gestão. De seguida devem clicar no botão "Listar terrenos do utilizador" e ser-lhes-á mostrada a lista de terrenos do utilizador selecionado.

Tabela 14 - Caso de uso: Filtrar terrenos por utilizador

Nome	Filtrar terrenos por utilizador
Objetivo	Mostrar na lista de terrenos da página "Explorar" apenas os terrenos do utilizador selecionado
Atores envolvidos	Manager ou Administrador
Pré-condição	Login válido e permissões de manager ou administrador.
Fluxo Principal	<ol style="list-style-type: none"> 1. O utilizador acede à página "Explorar". 2. O sistema abre a página "Explorar". 3. O utilizador clica no botão "Utilizadores". 4. O sistema devolve a lista de utilizadores. 5. O utilizador clica no botão "Listar terrenos do utilizador" do utilizador pretendido. 6. O sistema mostra a lista de terrenos, contendo apenas os terrenos adicionados pelo utilizador pretendido.
Fluxos alternativos	Não existem.
Pós-condição	Não existe.
Casos de teste	Não existe.

Abaixo é demonstrado o diagrama de sequência deste caso de uso, que usa apenas o componente da página "Explorar".

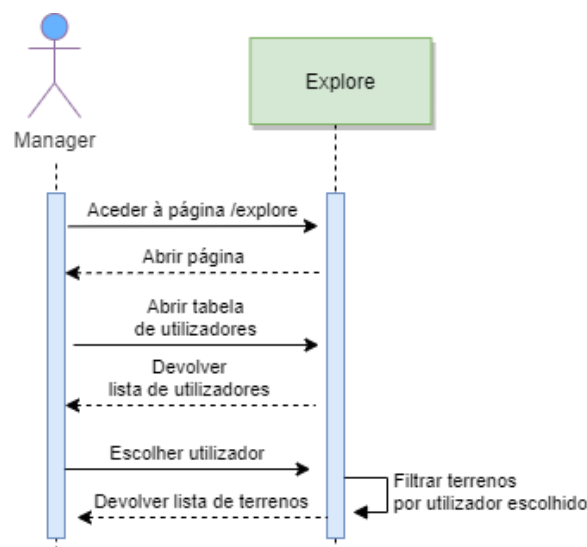


Figura 39 - Diagrama de sequência: Filtrar terrenos por utilizador

5.2.3.12 Validar terreno

Os Administradores e Managers podem confirmar a validação de terrenos que a tenham requisitado. Para isso devem aceder à página "Validação", onde verão uma lista de registos correspondentes aos pedidos de validação. Os Managers apenas têm acesso a pedidos de terrenos localizados dentro da zona que lhes foi atribuída. Ao clicar no botão "Abrir" de um registo de validação será aberta uma *dialog* onde é visível a informação do terreno e onde o utilizador pode descarregar os documentos disponibilizados pelo requisitante. De seguida o utilizador pode optar por confirmar ou cancelar a validação.

Tabela 15 - Caso de uso: Validar terreno

Nome	Validar terreno
Objetivo	Validar um terreno após a consulta dos documentos de validação.
Atores envolvidos	Manager ou Administrador
Pré-condição	Login válido e permissões de manager ou administrador.
Fluxo Principal	<ol style="list-style-type: none"> 1. O utilizador acede à página "Validação". 2. O sistema abre a página "Validação". 3. O utilizador clica no botão "Abrir" de um registo de validação. 4. O sistema abre a dialog de validação. 5. O utilizador clica no botão de descarregamento dos ficheiros de validação. 6. Uma vez analisados os documentos e a informação do terreno, o utilizador confirma a validação. 7. O sistema devolve mensagem de sucesso.
Fluxos alternativos	6a. Se os documentos não justificarem que a validação seja confirmada, o utilizador pode optar por cancelar a validação.
Pós-condição	A tabela que contém os registos de validação é atualizada com os novos dados.
Casos de teste	1. Verificar se os documentos justificam que o terreno seja validado.

Para este caso de uso os componentes necessários são a *view* referente à página de validação e a *dialog* de validação, como mostra o diagrama de sequência abaixo.

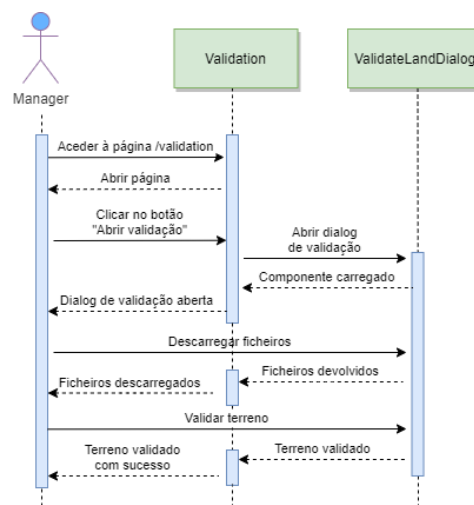


Figura 40 - Diagrama de sequência: Validar terreno

5.2.4 Diagrama e semântica de classes

Neste capítulo irão ser apresentados um diagrama de classes (adaptado para melhor perceção de como os registos da base de dados se relacionam entre si), e a semântica das respetivas classes, onde são descritos os seus atributos.

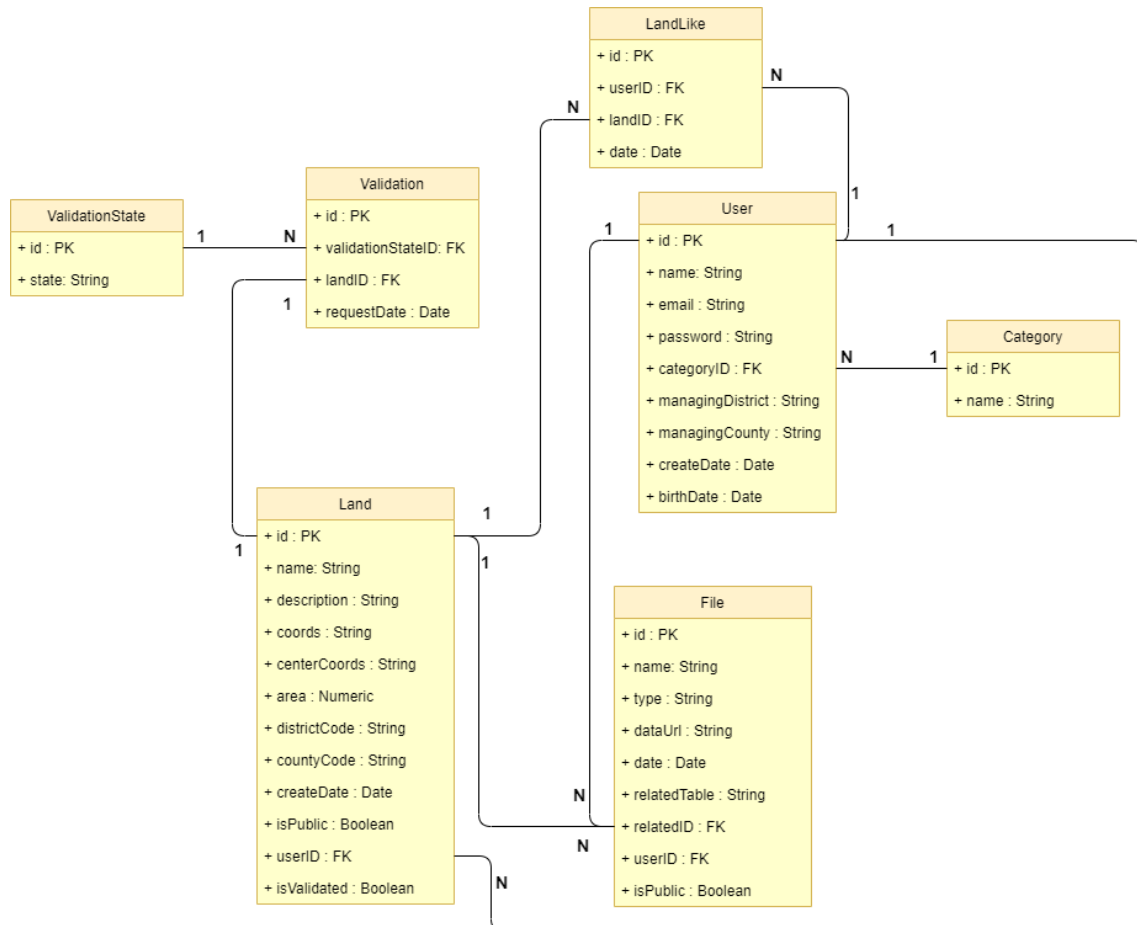


Figura 41 - Diagrama de classes

Tabela 16 - Classe: User - Utilizadores do sistema

Atributo	Tipo	Descrição	Valores válidos	Restrição
id	Chave primária, String hexadecimal	Identificação do terreno	24 caracteres	Obrigatório, gerado pelo sistema
name	Texto	Nome do utilizador	Min. 1 caracter	Obrigatório
email	Texto	Email do utilizador	Formato de email	Obrigatório
password	Texto	Password da conta do utilizador	Min. 6 caracteres	Obrigatório
categoryID	Chave estrangeira, String hexadecimal	Identificação da categoria do utilizador	24 caracteres	Obrigatório
managingDistrict	Texto	Código do distrito, caso o utilizador tenha a categoria de Manager de distrito	3 caracteres	Opcional
managingCounty	Texto	Código do município, caso o utilizador tenha a categoria de Manager de município	3 caracteres	Opcional
createDate	Data	Data de criação da conta	dd/mm/aaaa	Obrigatório, gerado pelo sistema, data atual no momento do registo
birthDate	Data	Data de nascimento do utilizador	dd/mm/aaaa	Obrigatório, no mínimo 18 anos mais antiga que a atual

Tabela 17 - Classe: Land - Terrenos adicionados ao sistema

Atributo	Tipo	Descrição	Valores válidos	Restrição
id	Chave primária, String hexadecimal	Identificação do terreno	24 caracteres	Obrigatório, gerado pelo sistema
name	Texto	Nome do terreno	Min. 3 caracteres	Obrigatório
description	Texto	Descrição do terreno	-	Opcional
coords	Texto	Coordenadas que formam o polígono do terreno	Matriz de coordenadas(lat,lng) convertido para String	Obrigatório, gerado pelo sistema
centerCoords	Texto	Coordenadas do centro do terreno	(lat,lng)	Obrigatório, gerado pelo sistema
area	Numérico	Área do terreno em metros quadrados	-	Obrigatório, gerado pelo sistema
districtCode	Texto	Código do distrito onde o terreno está localizado	3 caracteres	Obrigatório, gerado pelo sistema
countyDate	Texto	Código do município onde o terreno está localizado	3 caracteres	Obrigatório, gerado pelo sistema
createDate	Data	Data de criação do terreno	dd/mm/aaaa	Obrigatório, gerado pelo sistema, data atual no momento da adição do terreno
isPublic	Booleano	Define se a visibilidade do terreno é pública ou privada	True/False	Obrigatório
userID	Chave estrangeira, String hexadecimal	Identificação do utilizador que adicionou o terreno	24 caracteres	Obrigatório, gerado pelo sistema, data atual no momento da adição do terreno
isValidated	Booleano	Campo auxiliar para verificar se o terreno foi validado	True/False	Obrigatório, gerado pelo sistema

Tabela 18 - Classe: File - Ficheiros carregados pelos utilizadores

Atributo	Tipo	Descrição	Valores válidos	Restrição
id	Chave primária, String hexadecimal	Identificação do ficheiro	24 caracteres	Obrigatório, gerado pelo sistema
name	Texto	Nome do ficheiro	Min. 1 caracter + extensão	Obrigatório
type	Texto	Define se é um ficheiro usado para validar o terreno, ou se é uma imagem	-	Obrigatório, gerado pelo sistema
dataUrl	Texto	Ficheiro convertido para base 64	-	Obrigatório, gerado pelo sistema
date	Data	Data de carregamento do ficheiro	dd/mm/aaaa	Obrigatório, gerado pelo sistema
relatedTable	Texto	No estado atual da aplicação apenas é permitido o upload de ficheiros relacionados com registos da tabela de terrenos, mas este campo foi criado para prevenir a situação em que a aplicação escala para um contexto em que possa ser necessário o upload de ficheiros para outras tabelas	-	Obrigatório, gerado pelo sistema
relatedID	Chave estrangeira, String hexadecimal	Identificação do registo ao qual corresponde o ficheiro	24 caracteres	Obrigatório, gerado pelo sistema
userID	Chave estrangeira, String hexadecimal	Identificação do utilizador que carregou o ficheiro	24 caracteres	Obrigatório, gerado pelo sistema
isPublic	Booleano	Define se o ficheiro tem visibilidade pública ou privada	True/False	Obrigatório

Tabela 19 - Classe: Validation - Registos relativos às validações dos terrenos

Atributo	Tipo	Descrição	Valores válidos	Restrição
id	Chave primária, String hexadecimal	Identificação da validação	24 caracteres	Obrigatório, gerado pelo sistema
validationStateID	Chave estrangeira, String hexadecimal	Identificação do estado da validação	24 caracteres	Obrigatório, gerado pelo sistema
landID	Chave estrangeira, String hexadecimal	Identificação do terreno a que corresponde a validação	24 caracteres	Obrigatório, gerado pelo sistema
requestDate	Data	Data em que o utilizador requisitou validação	dd/mm/aaaa	Obrigatório, gerado pelo sistema

Tabela 20 - Classe: ValidationState - Estados possíveis que uma validação pode ter

Atributo	Tipo	Descrição	Valores válidos	Restrição
id	Chave primária, String hexadecimal	Identificação do estado de validação	24 caracteres	Obrigatório, gerado pelo sistema
state	Texto	Estado de validação	-	Obrigatório

Tabela 21 - Classe: LandLike -Gostos de um terreno

Atributo	Tipo	Descrição	Valores válidos	Restrição
id	Chave primária, String hexadecimal	Identificação do "Gosto"	24 caracteres	Obrigatório, gerado pelo sistema
userID	Chave estrangeira, String hexadecimal	Identificação do utilizador que fez o "Gosto"	24 caracteres	Obrigatório, gerado pelo sistema
landID	Chave estrangeira, String hexadecimal	Identificação do terreno a que corresponde o "Gosto"	24 caracteres	Obrigatório, gerado pelo sistema
date	Data	Data em que o utilizador fez o "Gosto"	dd/mm/aaaa	Obrigatório, gerado pelo sistema

Tabela 22 - Classe: Category - Categorias que podem ser atribuídas aos utilizadores

Atributo	Tipo	Descrição	Valores válidos	Restrição
id	Chave primária, String hexadecimal	Identificação do tipo de categoria	24 caracteres	Obrigatório, gerado pelo sistema
name	Texto	Categoria	-	Obrigatório

5.3. Componentes e instalação

Neste subcapítulo são demonstrados e descritos diagramas relativos aos componentes do projeto e sua instalação.

5.3.1 Diagrama de componentes

Em *UML*, os diagramas de componentes mostram a estrutura do sistema, descrevendo os componentes do *software*. Este tipo de diagrama suporta o desenvolvimento com base em componentes no qual um sistema de *software* é dividido em componentes e *interfaces* que são reutilizáveis e substituíveis, portanto enquadra-se perfeitamente quando utilizado em aplicações que usem *Vue.js*. Estes diagramas são úteis pelos seguintes motivos:

- Definir os aspetos executáveis e reutilizáveis de um sistema de *software*;
- Revelar problemas de configuração de *software* através de relacionamentos de dependência;
- Mostrar uma representação precisa de uma aplicação antes de fazer alterações ou aprimoramentos.

A aplicação contém um único ficheiro *HTML*, chamado “*index.html*”. O corpo deste ficheiro contém apenas um elemento que é representado no diagrama de componentes como “*main*”. Este elemento é controlado através do ficheiro “*App*”. Dentro deste ficheiro são adicionados os elementos que devem ser sempre visíveis, como é o caso do componente da barra de navegação e do “*router-view*”. Este elemento contém a *view* selecionada de momento, que irá diferir de página para página. A diferença entre uma *view* e um componente normal é que a *view* atual é definida através do *VueRouter*, que a incorpora dentro do elemento “*router-view*”, fazendo assim com que uma *view* possa ser encarada como uma página, com outros componentes a constituir o seu corpo. As ligações entre estes componentes são visíveis no diagrama de componentes abaixo.

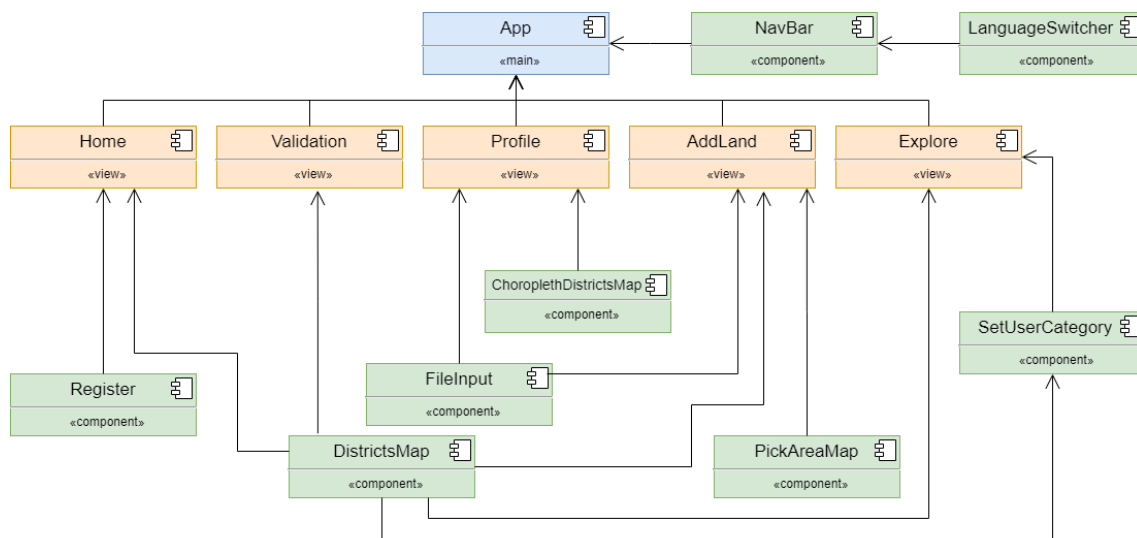


Figura 42 - Diagrama de componentes

5.3.2 Diagrama de instalação

Em *UML*, os diagramas de instalação modelam a arquitetura física de um sistema. Mostram os relacionamentos entre os componentes de *software* e *hardware* no sistema e a distribuição física do processamento. Os diagramas de instalação, que normalmente são preparados durante a fase de desenvolvimento da implementação, mostram a organização física dos nós num sistema distribuído, onde são feitos armazenamentos e outros elementos que sejam implementados.

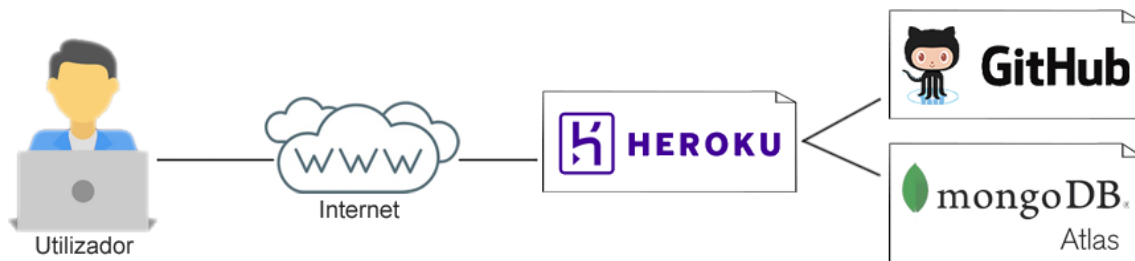


Figura 43 - Diagrama de instalação

Capítulo 6 – Implementação e desenvolvimento

Para a criação da aplicação “*GeoRep*” foi adotada a *stack MEVN: MongoDB, Express.js, Vue e Node.js*. O primeiro passo foi o desenvolvimento do *back-end*, o que inclui a *API*, o processo de autenticação, *state management*, etc, e só depois deste estar completo, se iniciou o desenvolvimento do *front-end*, que inclui a construção da *UI* através de componentes do *Vue*.

Serão então descritos neste capítulo o funcionamento e finalidade dos vários elementos que compõem tanto o *back*, como o *front-end*, com maior incidência sobre este último, uma vez que contém uma maior variedade de processos e técnicas que o *back-end*, que acaba por ser mais repetitivo.

6.1. Back-end

Para a base de dados foi usado *MongoDB*. Trata-se de um sistema *NoSQL*, ou seja, os documentos não têm qualquer relação entre si. Estes documentos, em vez de tabelas e colunas, são objetos simples, semelhantes a objetos *JS*, com sintaxe *JSON*. As bases de dados *NoSQL* tendem a ser uma opção melhor para aplicações modernas que têm conjuntos de dados mais complexos e que podem mudar constantemente, acabando por exigir um modelo de dados flexível que não precise ser definido imediatamente. Não houve necessidade de instalar o *MongoDB* localmente porque foi usado o *MongoDB Atlas*, que permite o armazenamento de bases de dados na *cloud*. Através de um *URI (Universal Resource Identifier)* que é disponibilizado pelo *MongoDB Atlas*, é feita uma conexão ao *Mongoose*, estabelecendo ligação com a aplicação. Para que o *Mongoose* pudesse interagir corretamente com a base de dados, foi necessário criar modelos em *JavaScript* das várias classes da base de dados (utilizadores, terrenos, etc.). O facto de a base de dados estar na *cloud* permite também um *deploy* muito mais prático, visto que para configurar a base de dados no *Heroku* basta adicionar o *URI* como variável de configuração.

Na aplicação foi também construída a *API* dividida em vários ficheiros com as diferentes rotas. Para testar os pedidos às rotas foi usado o *Postman*.

Foi utilizado *JWT* para autenticar utilizadores e permitir o seu acesso a rotas protegidas e o *bcryptjs* para encriptar a password no momento de registo e verificar se a password inserida quando se tenta iniciar sessão é válida.

6.2. Front-end

Neste capítulo irão ser descritos os componentes que compõem o front-end, de modo a transmitir como estes funcionam, e as funcionalidades da aplicação que deles advêm.

6.2.1 App

O *root component* foi denominado de “App”, contendo o componente “NavBar”, uma “snack-bar” e o “router-view”. Todos estes foram inseridos dentro do *root component* porque devem ser vistos a qualquer altura, independentemente da página que o utilizador tiver aberto. O “router-view” contém a *view* seleccionada de momento, que irá diferir de página para página e a “snack-bar” não é nada mais que uma pequena janela de aviso que se torna visível por alguns segundos no canto superior direito da aplicação para dar informações ao utilizador, sejam de erro ou de sucesso.

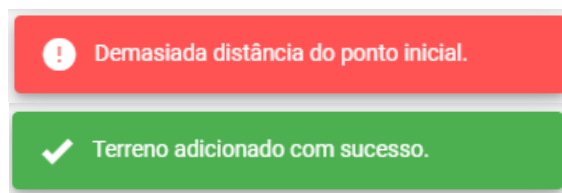


Figura 44 - Exemplos de mensagens da "snack-bar"

6.2.1 NavBar

A *NavBar* é o componente referente à barra de navegação que é sempre visível no topo da aplicação. Dependendo do estado da sessão do utilizador tem comportamentos diferentes. Caso o utilizador não tenha iniciado sessão, é visível o componente “*LanguageSwitcher*” e duas caixas de texto para autenticação com o respetivo botão de *login*. Se o utilizador estiver com sessão iniciada, são visíveis, para além do “*LanguageSwitcher*”, vários botões de navegação que permitem aceder a diferentes páginas da aplicação, para além do botão de *logout*.



Figura 45 – NavBar

6.2.2 LanguageSwitcher

Este componente permite que o utilizador mude o idioma da aplicação, sem necessidade de atualizar a página. Funciona em conjunto com *Vue I18n*, alterando a variável global que define o idioma atual do sistema. É composto por um menu colapsável que contém uma lista das línguas disponíveis.

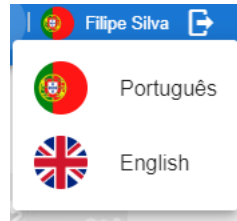


Figura 46 – LanguageSwitcher

6.2.3 Home

Este componente é também uma *view* definida para ser visível através do “*router-view*” quando se acede à página inicial, em que o *path* é “/”. Sendo esta a página inicial, tem comportamentos diferentes dependendo do estado da sessão do utilizador. Caso o utilizador não tenha iniciado sessão é visível o componente “*Register*”, caso contrário, é apresentado o mapa nacional com os diferentes polígonos que constituem os terrenos adicionados no sistema, com recurso à *Google Maps API*. Ao seleccionar um terreno no mapa torna-se visível um painel de informação, onde o utilizador pode ver imagens do terreno, consultar a sua informação e gerar um *link* de partilha, que pode ser enviado para outras pessoas acederem facilmente ao mesmo terreno.

São também disponibilizados dois mecanismos que permitem ao utilizador abrir o mapa numa localização à sua escolha. Um deles é um mapa interativo, com recurso ao componente “*DistrictsMap*”, em que o utilizador escolhe o distrito ou município que pretender abrir, o outro trata-se de uma caixa de texto que, através da *Geocode API*, abre o mapa numa morada inserida pelo utilizador.

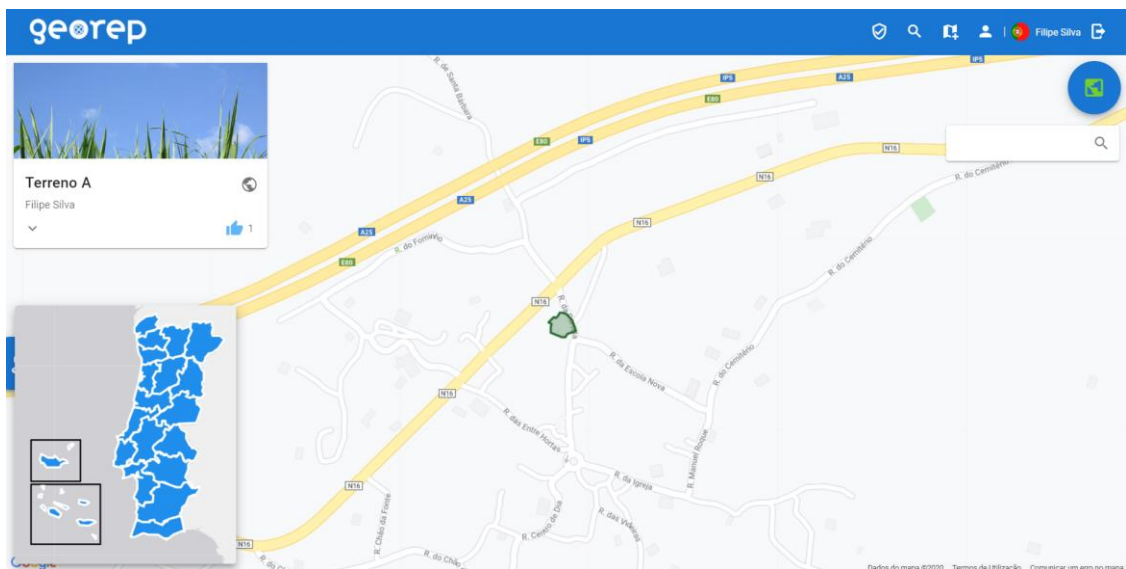
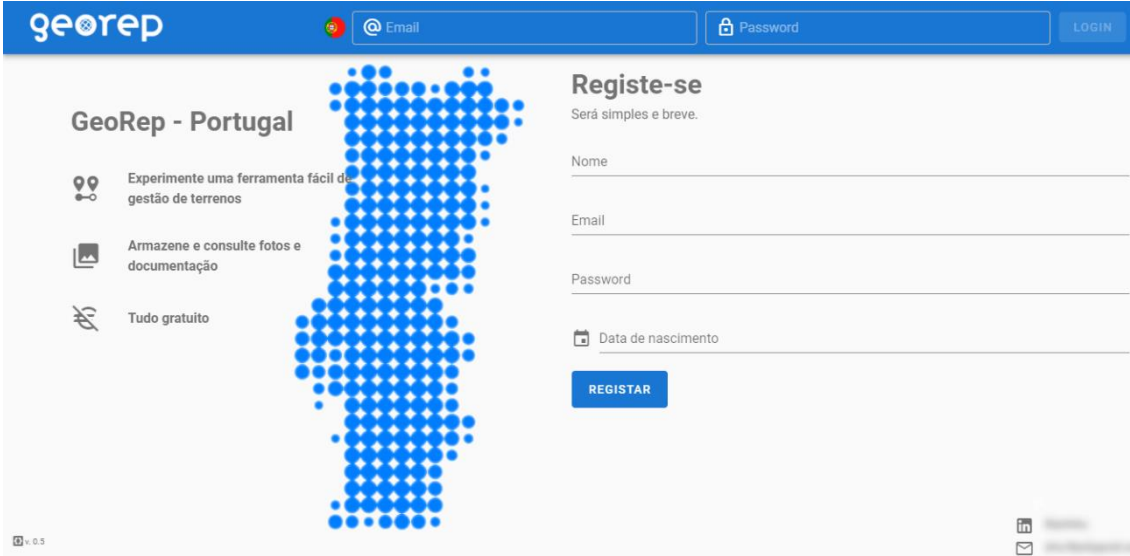


Figura 47 - Home

6.2.4 Register

Este componente é visível na *view* “*Home*” quando o utilizador não tem sessão iniciada. Para além de alguma informação simples relativa às funcionalidades da aplicação, contém um formulário que permite a criação de contas.



The screenshot shows the registration page for GeoRep - Portugal. The page has a blue header with the 'georep' logo on the left, an email input field, a password input field, and a 'LOGIN' button. Below the header, the main content area is divided into two columns. The left column features the text 'GeoRep - Portugal' and three bullet points: 'Experimente uma ferramenta fácil de gestão de terrenos', 'Armazene e consulte fotos e documentação', and 'Tudo gratuito'. A large graphic of a map of Portugal, composed of blue dots, is positioned between the two columns. The right column is titled 'Registe-se' and contains the subtext 'Será simples e breve.' followed by four input fields: 'Nome', 'Email', 'Password', and 'Data de nascimento'. A blue 'REGISTAR' button is located below the 'Password' field. In the bottom right corner, there are social media icons for LinkedIn and Facebook.

Figura 48 - Register

6.2.5 Profile

Este componente é também uma *view* cujo *path* é “/profile”. Nesta página é visível uma lista dos terrenos que foram adicionados pelo utilizador atual, um botão que permite eliminar a conta e um mapa coroplético que é gerado a partir do componente “*ChoroplethDistrictsMap*”. Na lista de terrenos, é apresentada a informação relativa aos vários terrenos como o nome, descrição, distrito, município, visibilidade, validação, mas também são disponibilizados botões que permitem tomar ações relativamente a um certo terreno: editar a informação do terreno, abri-lo no mapa da página inicial ou eliminá-lo.

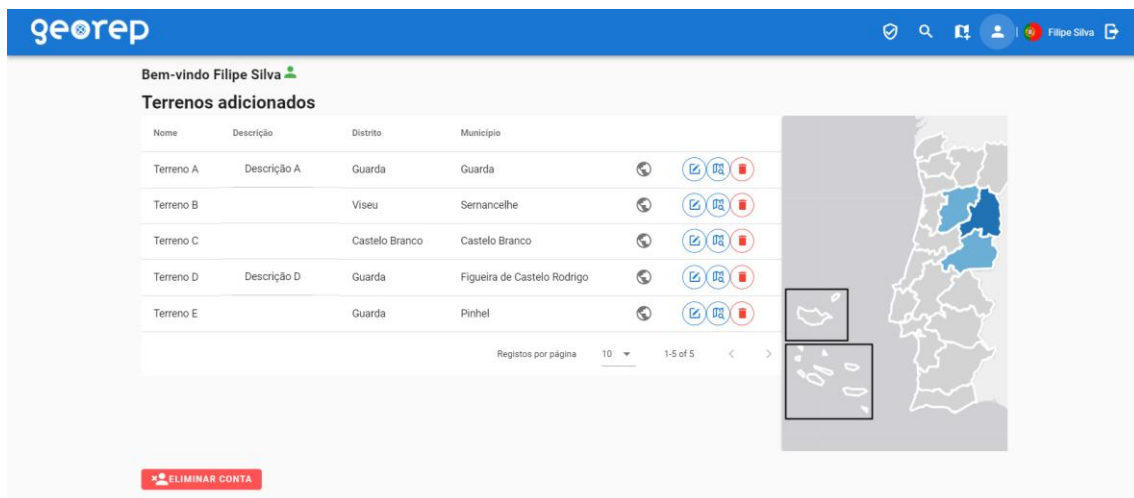


Figura 49 - Profile

Caso o utilizador pretenda eliminar a sua conta, a *dialog* de confirmação obriga a que o utilizador insira o seu *email*, para evitar uma eliminação acidental. Opcionalmente, pode também eliminar os seus terrenos, caso não pretenda que estes fiquem visíveis no mapa.

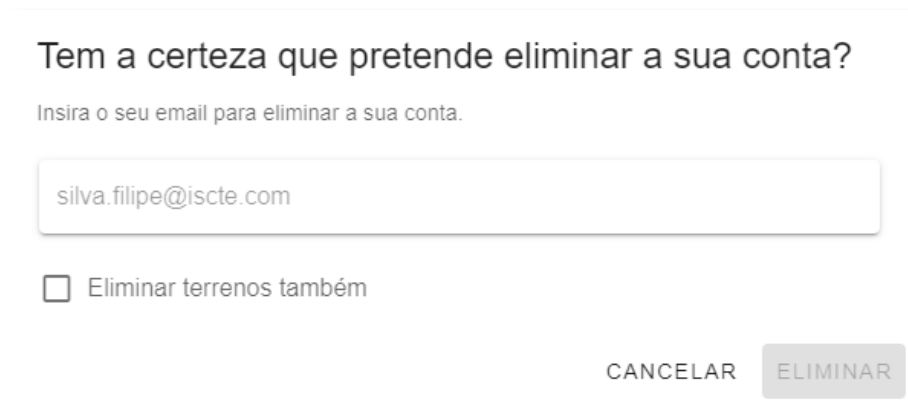


Figura 50 - Dialog de eliminar conta

6.2.6 ChoroplethDistrictsMap

Este componente contém apenas uma *div*, na qual através do *Leaflet* é incorporado um mapa coroplético interativo. É utilizado no “*Profile*” para filtrar os terrenos visíveis na lista, mas também para representar uma estatística, simbolizando com cores os distritos e municípios onde o utilizador adicionou mais terrenos. A intensidade da cor corresponde a uma maior concentração de terrenos, relativamente a outras zonas.

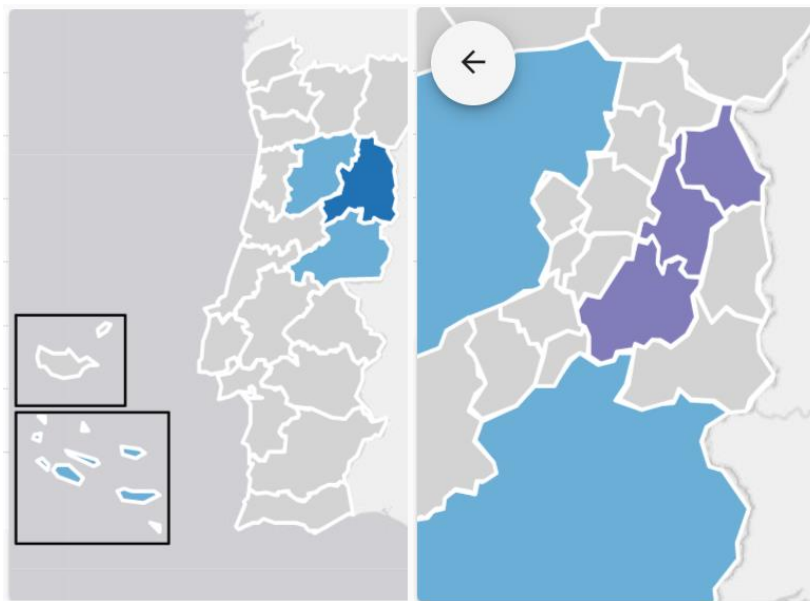


Figura 51 – *ChoroplethDistrictsMap*

6.2.7 Explore

Este componente é também uma *view* cujo *path* é “/explore”. Nesta página é visível uma lista, onde o utilizador pode consultar detalhes de todos os terrenos que tenham visibilidade pública (para além dos que tenham sido adicionados por si), nomeadamente o nome, distrito, município, data em que foi inserido, estado da validação e visibilidade. Juntamente são disponibilizados vários recursos de filtragem: através de texto, por estado de validação dos terrenos, ocultar terrenos do utilizador atual e filtrar por distrito ou município com recurso ao componente “*DistrictsMap*”. O utilizador pode clicar no botão “Abrir terreno”, de forma a selecionar o terreno pretendido e abri-lo no mapa da página inicial.



Figura 52 - Explore

É também possível abrir uma *dialog* para cada terreno que apresentará mais detalhes, como a área e um mapa onde é visível o polígono que constitui o terreno.

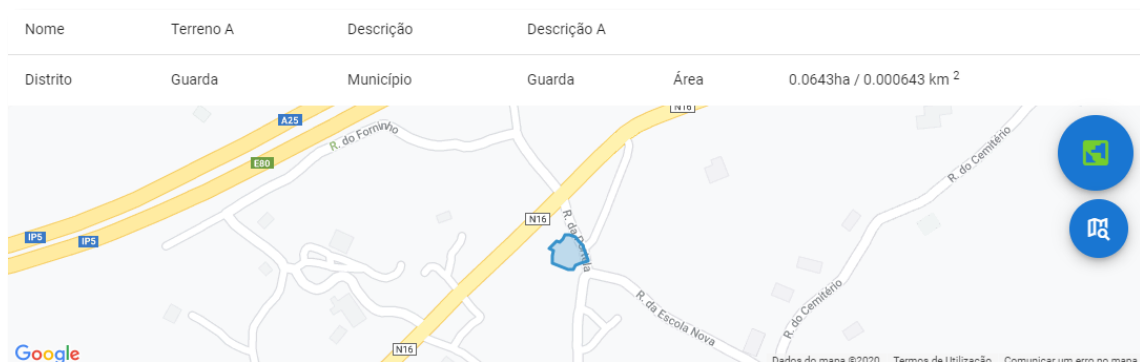
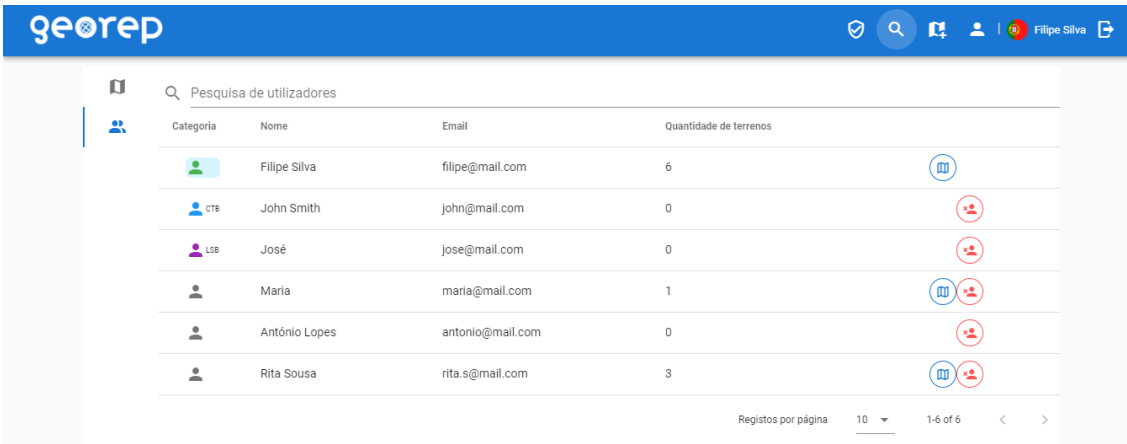










Figura 53 - Dialog de detalhes de terreno

Caso o utilizador seja Administrador ou Manager, terá acesso a um separador dedicado aos utilizadores do sistema, onde é visível uma lista dos utilizadores e respetiva informação. É também disponibilizado um botão que permite filtrar pelo utilizador pretendido a lista de terrenos descrita anteriormente. Se o utilizador for Manager, apenas são visíveis na lista utilizadores que tenham adicionado terrenos na zona que lhe foi atribuída. Se o utilizador for Administrador, para além de serem visíveis todos os utilizadores do sistema, tem também a possibilidade de eliminar as contas dos utilizadores, ou alterar a sua categoria, com recurso ao componente “*SetUserCategory*”.



Categoria	Nome	Email	Quantidade de terrenos	
	Filipe Silva	filipe@mail.com	6	
CTE	John Smith	john@mail.com	0	
LSE	José	jose@mail.com	0	
	Maria	maria@mail.com	1	 
	António Lopes	antonio@mail.com	0	
	Rita Sousa	rita.s@mail.com	3	 

Registos por página 10 1-6 of 6 < >

Figura 54 - Explore (Separador de Utilizadores)

6.2.8 SetUserCategory

Este componente é utilizado dentro do “*Explore*” para mudar a categoria dos utilizadores. Este é aberto dentro de uma *dialog* quando o Administrador clica sobre o ícone do utilizador que pretende. Este componente também contém o “*DistrictsMap*”, que neste caso é usado para definir o distrito ou município que vai passar a ser a zona atribuída ao utilizador, no caso de ser Manager. Caso o Administrador opte por remover o cargo de gestão, o utilizador em questão passa a ser um Utilizador normal.

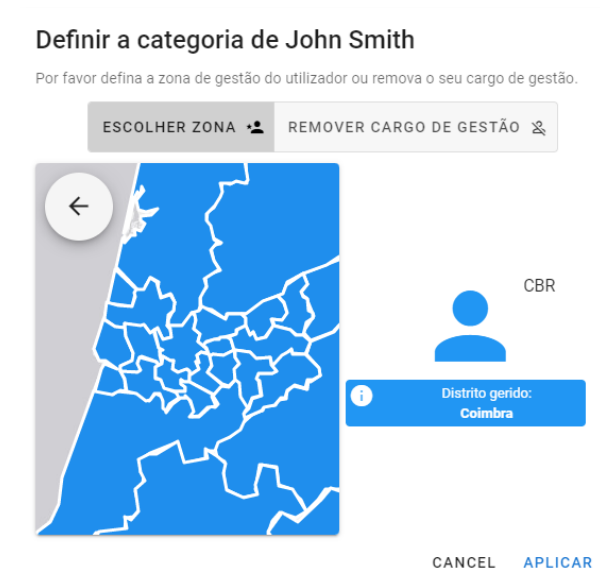


Figura 55 – SetUserCategory

6.2.9 AddLand

Este componente é também uma *view* cujo *path* é *“/addland”*. Esta página é usada para adicionar terrenos ao sistema. São visíveis vários separadores, sendo o primeiro chamado de “Informação”, que contém caixas de texto para o nome e descrição do terreno. A visibilidade do terreno está por defeito definida como pública, mas neste separador o utilizador pode alterá-la se assim preferir. Também é possível, com recurso ao componente *“FileInput”*, carregar imagens do terreno e definir individualmente a visibilidade de cada imagem.

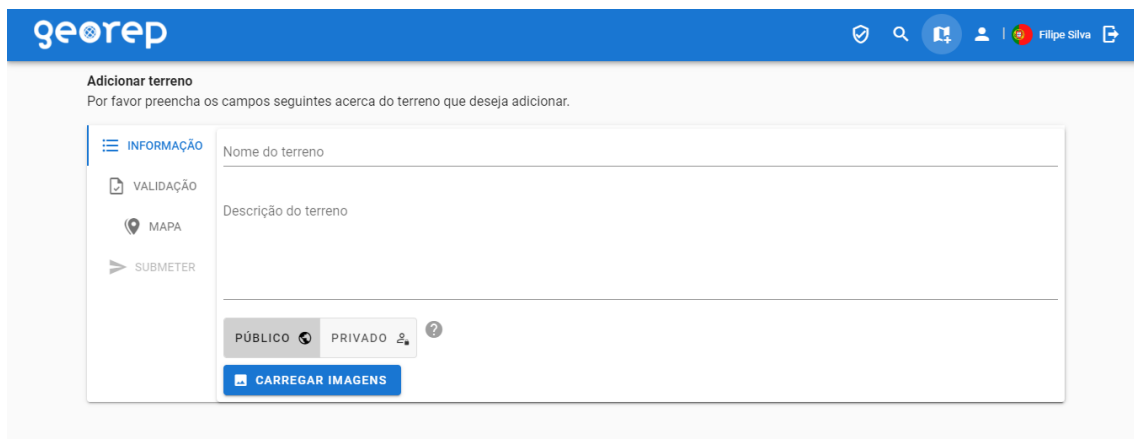


Figura 56 - AddLand (Informação)

O separador “Validação” contém de novo o componente *“FileInput”* de modo a que o utilizador possa fazer *upload* de documentos para requisitar a validação do terreno. Estes documentos serão analisados posteriormente por um Administrador ou Manager.

O separador “Mapa” contém dois componentes. Primeiro é usado o componente *“DistrictsMap”* para que o utilizador defina o município onde pretende desenhar o terreno. Assim que o município é escolhido, torna-se visível o componente *“PickAreaMap”*, que se trata de uma ferramenta de desenho de polígonos feita de raiz usando a *Google Maps API*. Aqui o utilizador, para além de poder desenhar polígonos através de pontos, pode também arrastar os mesmos, retroceder nos seus passos ou refazê-los.

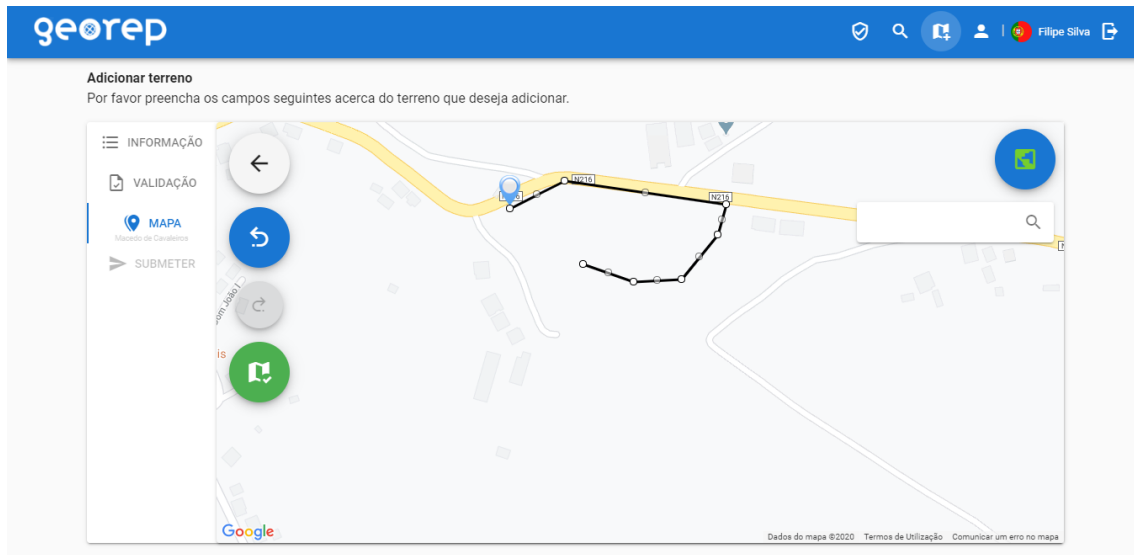


Figura 57 - AddLand (Mapa)

Assim que o polígono está finalizado o utilizador é automaticamente redirecionado para o separador “Submeter” onde, para confirmação, lhe são apresentados os detalhes inseridos no terreno, bem como uma pré-visualização no mapa, mais uma vez com recurso à *Google Maps API*.

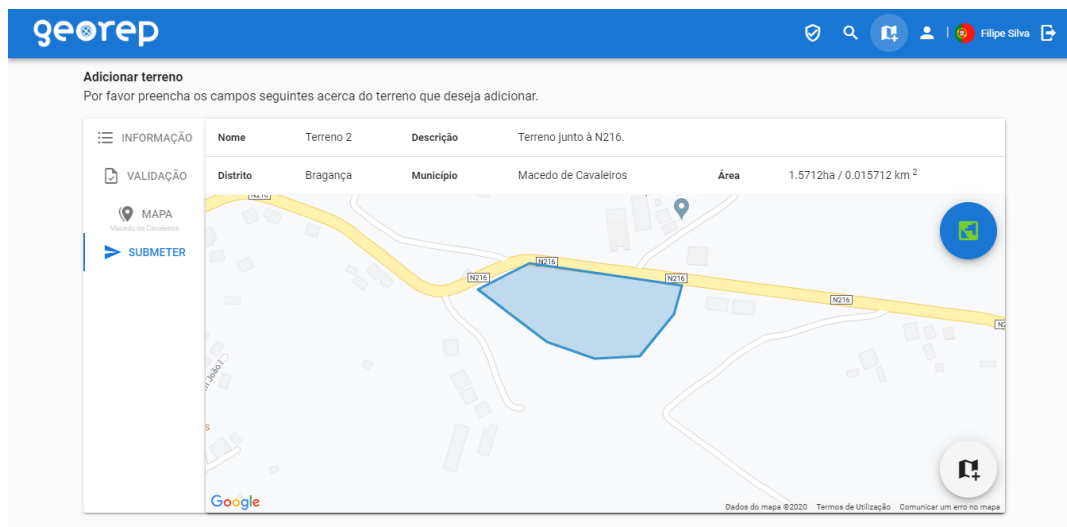


Figura 58 - AddLand (Submeter)

6.2.10 PickAreaMap

Este componente contém a ferramenta de desenho que é utilizada no separador “Mapa” do componente “AddLand”. Depois do utilizador escolher o município onde pretende desenhar o terreno, este componente constrói um mapa com recurso à *Google Maps API* nas coordenadas do respetivo município, bem como polígonos relativos a terrenos que estejam localizados nessa zona. É depois gerada uma *polyline* vazia, que através dos cliques do utilizador, vai ganhando pontos de coordenadas *LatLng* (latitude, longitude). Sempre que é feito um clique no mapa é guardado o estado da *polyline*, de forma a ser possível reverter as ações do utilizador, caso seja pretendido. O utilizador pode optar por ver um mapa normal ou com fotos de satélite. Assim que o desenho é concluído, a *polyline* é convertida para um polígono de modo a guardar as suas coordenadas.

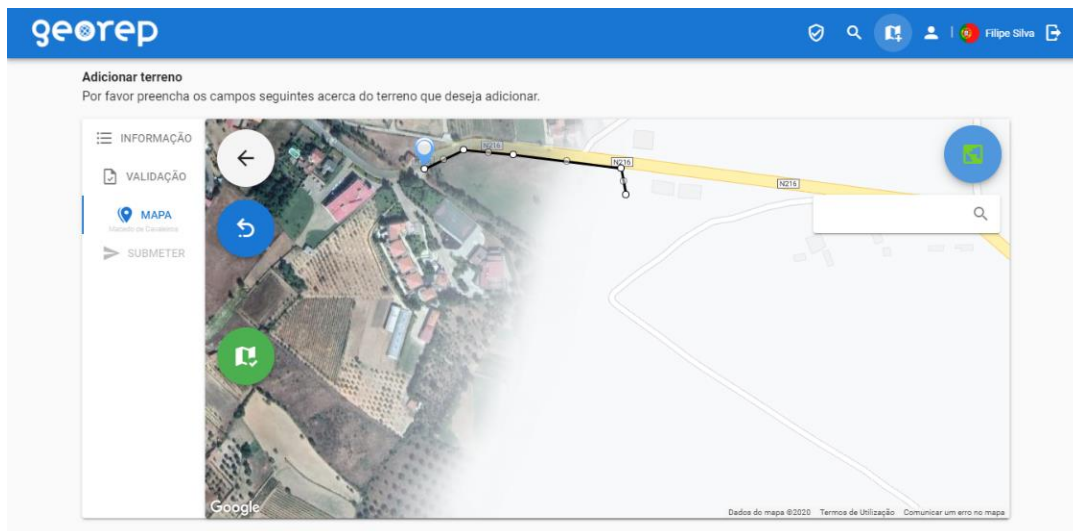


Figura 59 – PickAreaMap

6.2.11 Validation

Este componente é também uma *view* cujo *path* é “/validation”. Esta página é acessível apenas por Administradores e Managers, sendo apresentada a lista de validações que foram requisitadas. No caso dos Managers só são visíveis validações cujo terreno esteja inserido dentro da zona que lhes foi atribuída. O componente “*DistrictsMap*” também é aqui usado para filtrar as validações por distrito ou município. Ao clicar no botão “Abrir validação” da validação pretendida, é aberta uma dialog com os detalhes do terreno, o mapa com o polígono referente ao terreno e os documentos de validação para descarregar.



Figura 60 - Dialog de validação

Caso a validação seja aceite, o terreno passará a contar com o símbolo de verificado no seu painel de informação, bem como em pesquisas na página “Explorar”.



Figura 61 - Terreno verificado

6.2.12 DistrictsMap

Este componente é o mais usado em toda a aplicação, estando presente dentro de outros cinco componentes. Trata-se de um mapa interativo construído através do *Leaflet*, que permite ao utilizador escolher um distrito ou um município. Nos casos das páginas “Explorar” e “Validação” é usado como ferramenta de filtragem dos registos apresentados. No componente “*SetUserCategory*”, caso se pretenda que o utilizador em questão seja Manager, é usado para definir o distrito ou município que vai passar a ser a zona atribuída. Na página “Adicionar Terreno” é utilizado para abrir a ferramenta de desenho no município pretendido e, de forma semelhante, na página inicial é usado para mudar a vista atual do mapa para a zona pretendida.

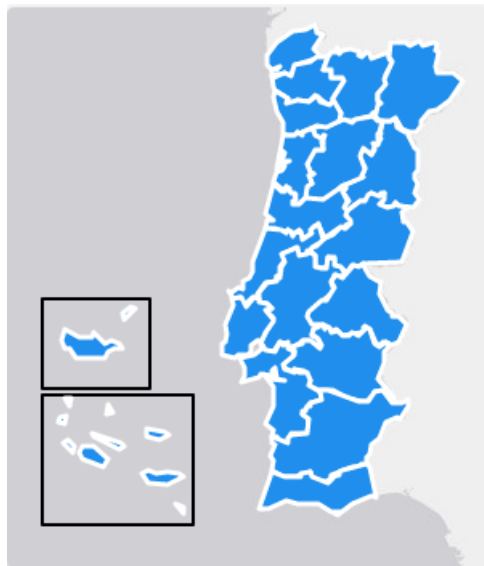


Figura 62 – DistrictsMap

Os diferentes objetos que constituem os vários polígonos representativos de distritos e municípios são gerados a partir do padrão *GeoJSON*. Este formato é utilizado para codificar várias estruturas geográficas, contendo as coordenadas, tipo de objeto, entre outras propriedades personalizáveis.

```
{
  type: "Feature",
  properties: {
    id: "GRD",
    name: "Guarda",
    type: "district",
    centerCoords: [40.53733, -7.26575],
    zoom: 9
  },
  geometry: {
    type: "Polygon",
    coordinates: [
      [
        [-6.92962646484375, 41.04621681452063],
        [-6.8115234375, 40.33503005827232],
        (...),
        [-6.941986083984375, 41.006848111213586],
        [-6.92962646484375, 41.04621681452063]
      ]
    ]
  }
}
```

Figura 63 - Exemplo de objeto *GeoJSON*

6.2.13 FileInput

Este componente foi criado de forma a disponibilizar uma ferramenta simples e fácil de usar para o *upload* de ficheiros. É usado no componente “*AddLand*” para carregar imagens do terreno e documentos de validação, e também no “*Profile*”, quando se edita um terreno, no caso de o utilizador precisar de mudar os ficheiros que carregou.

É composto por um botão, que quando clicado abre uma janela do gestor de ficheiros para o utilizador poder escolher os ficheiros que pretende carregar, e por uma tabela onde é visível o nome dos ficheiros carregados, bem como botões para alterar a sua visibilidade ou eliminá-los.

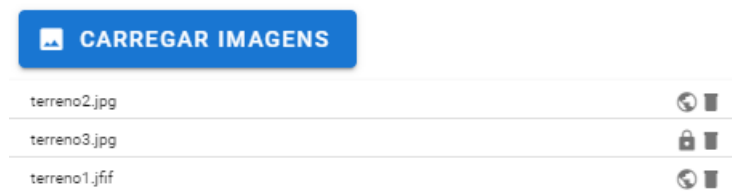


Figura 64 - FileInput

Capítulo 7 – Conclusões

Uma vez que a *framework/library* que iria ser usada no projeto não foi clara desde o início, a pesquisa do estado da arte foi fulcral para entender as diferenças que existem entre as principais *frameworks* de *front-end* e perceber qual a mais apropriada para cada caso. O *Angular* foi descartado por não contar com o mesmo favoritismo que o *React* ou o *Vue*, pela sua complexidade no que toca à gestão de ficheiros, pelo facto de ser necessário trabalhar com *TypeScript*, e pela sua performance ser a pior das três opções apresentadas. Entre o *React* e o *Vue*, foi escolhida a alternativa que apresentou a estrutura mais intuitiva, a sintaxe mais tradicional, permitindo que outros projetos e *libraries* fossem migrados facilmente, a maior qualidade de suporte para os utilitários, oferecido em grande parte pela equipa de desenvolvimento oficial, bem como uma boa performance, um código “limpo” conseguido através do formato adotado nos ficheiros, e uma boa facilidade de aprendizagem.

No que toca ao desenvolvimento, foram exploradas profundamente funcionalidades do *Vue*, e de outros utilitários como o *Vuetify* na disponibilização de componentes, o padrão *JWT* para a autenticação com recurso a *tokens*, o *bcryptjs* na encriptação e validação de passwords, e diferentes *APIs* da *Google* para visualização de mapas e pesquisa de endereços, *softwares* com os quais nunca tinha trabalhado anteriormente.

A metodologia de desenvolvimento *Scrum* foi favorável ao desenvolvimento do projeto, e graças ao contacto constante com o orientador durante o desenvolvimento do projeto, e ao planeamento da aplicação com recurso a desenho e modelação *UML*, foi possível alcançar os objetivos com um bom grau de qualidade.

Foi criada uma aplicação com várias funcionalidades, através da qual se podem adicionar terrenos a uma base de dados através de uma ferramenta de desenho fácil de utilizar, com uma *interface* intuitiva criada com recurso à *framework* de *front-end* mais admirada pela comunidade de código aberto, com vários tipos de utilizadores, utilizadores esses que possuem diferentes permissões e funções.

Como prova de conceito não ficou nada por fazer. Para continuar e concretizar o verdadeiro objetivo de ter uma plataforma pronta a usar por qualquer utilizador ou órgão executivo será agora necessário recolher informações acerca da existência e funcionamento de registos oficiais, da motivação que cada utilizador precisa para participar numa plataforma deste cariz, e dos regulamentos existentes.

Para finalizar, considero que este projeto foi um ponto marcante na minha aprendizagem, não só devido à consolidação de conhecimentos já adquiridos, mas também devido à descoberta de novas tecnologias e soluções, que de outra forma não teria tido oportunidade conhecer e explorar.

O código da aplicação está disponível no seguinte link:

https://github.com/SilvaFilipe/GeoRep_PT

Referências Bibliográficas

SCHWARZMÜLLER, M. 2017. The World of JavaScript. *Academind.com*. [Consulta: 14 março 2020]. Disponível em: <https://academind.com/learn/javascript/the-world-of-javascript/>.

DAITYARI, S. 2020. Angular vs React vs Vue: Which Framework to Choose in 2020. *CodeinWP*. [Consulta: 14 março 2020]. Disponível em: <https://www.codeinwp.com/blog/angular-vs-vue-vs-react/>.

VUE.JS DEV TEAM 2020. Comparison with Other Frameworks — Vue.js. *Vuejs.org*. [Consulta: 14 março 2020]. Disponível em: <https://vuejs.org/v2/guide/comparison.html>.

KRAJKA, B. 2015. The difference between Virtual DOM and DOM. *React Kung Fu*. [Consulta: 14 março 2020]. Disponível em: <https://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/>.

VUE.JS DEV TEAM 2020. Computed Properties and Watchers — Vue.js. *Vuejs.org*. [Consulta: 14 março 2020]. Disponível em: <https://vuejs.org/v2/guide/computed.html>.

ALTEXSOFT 2019. The Good and the Bad of Angular Development. *AltexSoft*. [Consulta: 14 março 2020]. Disponível em: <https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/>.

BITSOFCODE 2018. What, exactly, is the DOM?. *bitsofcode*. [Consulta: 14 março 2020]. Disponível em: <https://bitsofco.de/what-exactly-is-the-dom/>.

SHAKED, U. 2017. A Deep, Deep, Deep, Deep, Deep Dive into the Angular Compiler. *Medium*. [Consulta: 14 março 2020]. Disponível em: <https://medium.com/angular-in-depth/a-deep-deep-deep-deep-deep-dive-into-the-angular-compiler-5379171ffb7a>.

MARX, L. 2017. Angular: Everything you need to get started. *Malcoded.com*. [Consulta: 14 março 2020]. Disponível em: <https://malcoded.com/posts/angular-beginners-guide/>.

MOHAN, M. 2019. How React works under the hood. *freeCodeCamp.org*. [Consulta: 14 março 2020]. Disponível em: <https://www.freecodecamp.org/news/react-under-the-hood/>.

LOBERA, A. 2018. A Beginner's Guide to React: How does React work?. *Medium*. [Consulta: 14 março 2020]. Disponível em: <https://medium.com/leanjs/introduction-to-react-3000e9cbcd26>.

CHRIS ON CODE 2019. Getting Started with React (2019 Edition). *Scotch*. [Consulta: 14 março 2020]. Disponível em: <https://scotch.io/starters/react/getting-started-with-react-2019-edition>.

ROGOJAN, B. 2019. Vue vs React: Which is the better framework?. *Buttercms*. [Consulta: 14 março 2020]. Disponível em: <https://buttercms.com/blog/vue-vs-react-which-is-the-better-framework>.

FRANKLIN, J. 2019. Getting up and Running with the Vue.js 2.0 Framework — SitePoint. *SitePoint*. [Consulta: 14 março 2020]. Disponível em: <https://www.sitepoint.com/up-and-running-vue-js-2-0/>.

DRASNER, S. 2018. Intro to Vue.js: Components, Props, and Slots | CSS-Tricks. *CSS-Trick*. [Consulta: 14 março 2020]. Disponível em: <https://css-tricks.com/intro-to-vue-2-components-props-slots/>.

PATEL, P. 2018. What exactly is Node.js?. *freeCodeCamp.org*. [Consulta: 14 março 2020]. Disponível em: <https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/>.

SCHWARZMÜLLER, M. 2019. MEAN, MERN & MEVN - Stack: My Thoughts. *Academind.com*. [Consulta: 14 março 2020]. Disponível em: <https://academind.com/learn/vue-js/mean-mern-mevn/>.

NASCIMENTO, W. 2018. Entendendo tokens JWT (Json Web Token). *Medium*. [Consulta: 14 março 2020]. Disponível em: <https://medium.com/tableless/entendendo-tokens-jwt-json-web-token-413c6d1397f6>.

FRANCIS, N. [sem data]. Collection of the Coolest Uses of the Google Maps API | The JotForm Blog. *Jotform Blog*. [Consulta: 14 março 2020]. Disponível em: <https://www.jotform.com/blog/collection-of-the-coolest-uses-of-the-google-maps-api/>.

EQUIPE DEVMEDIA [sem data]. Modelagem de sistemas com UML: Principais tipos de diagramas. *DevMedia*. [Consulta: 14 março 2020]. Disponível em: <https://www.devmedia.com.br/modelagem-de-sistemas-atraves-de-uml-uma-visao-geral/27913>.

LYNCH, A. 2019. UML Component Diagram Symbols. *Edrawsoft.com*. [Consulta: 14 março 2020]. Disponível em: <https://www.edrawsoft.com/uml-component-symbols.html>.

GITHUB. 2020. angular / angular. GitHub. [Consulta: 14 março 2020]. Disponível em: <https://github.com/angular/angular>.

GITHUB. 2020. vuejs / vue. GitHub. [Consulta: 14 março 2020]. Disponível em: <https://github.com/vuejs/vue>.

GITHUB. 2020. Search for your next job. LinkedIn. [Consulta: 14 março 2020]. Disponível em: <https://www.linkedin.com/jobs/>.

LINKEDIN. 2020. facebook / react. GitHub. [Consulta: 14 março 2020]. Disponível em: <https://github.com/facebook/react>.

NPM TRENDS. 2020. angular vs react vs vue. [Consulta: 18 abril 2020]. Disponível em: <https://www.npmtrends.com/angular-vs-react-vs-vue>.

FERGUSON, N. 2020. What's The Difference Between Frontend And Backend Web Development?. *CareerFoundry*. [Consulta: 11 outubro 2020]. Disponível em: <https://careerfoundry.com/en/blog/web-development/whats-the-difference-between-frontend-and-backend/>.

PASTORINO, M. [sem data]. Frontend Vs. Backend: What's The Difference?. Pluralsight. [Consulta: 11 outubro 2020]. Disponível em: <https://www.pluralsight.com/blog/software-development/front-end-vs-back-end/>.

WANYOIKE, M. 2018. History of front-end frameworks. LogRocket. [Consulta: 11 outubro 2020]. Disponível em: <https://blog.logrocket.com/history-of-frontend-frameworks/>.

BARKER, A. 2018. The Super-Brief History of JavaScript Frameworks For Those Somewhat Interested. Dev.to. [Consulta: 11 outubro 2020]. Disponível em: https://dev.to/_adam_barker/the-super-brief-history-of-javascript-frameworks-for-those-somewhat-interested-3m82/.

MINNICK, C. 2016. The Real Benefits of the Virtual DOM in React.js. Accelebrate. [Consulta: 11 outubro 2020]. Disponível em: <https://www.accelebrate.com/blog/the-real-benefits-of-the-virtual-dom-in-react-js/>.

RUSEV, A. 2019. React vs Vue: The Core Differences. Mentormate. [Consulta: 11 outubro 2020]. Disponível em: <https://mentormate.com/blog/react-vs-vue-the-core-differences/>.

LANÇA, F. 2018. Terrenos sem dono conhecido vão passar para propriedade pública. Jornal de Negócios. [Consulta: 11 outubro 2020]. Disponível em: <https://www.jornaldenegocios.pt/economia/detalhe/terrenos-sem-dono-conhecido-vaopassar-para-propriedade-publica/>.

CONFAGRI 2019. Regime de identificação e administração de terrenos sem dono publicado em Diário da República. [Consulta: 11 outubro 2020]. Disponível em: <https://www.confagri.pt/regime-identificacao-administracao-terrenos-sem-dono-publicado-diario-da-republica/>.

CLOUDFLARE. [sem data]. What Do Client-Side and Server-Side Mean? | Client Side vs. Server Side. Cloudflare. [Consulta: 11 outubro 2020]. Disponível em: <https://www.cloudflare.com/learning/serverless/glossary/client-side-vs-server-side/>.

NEAGOIE, A. 2018. Tech Trends Showdown: React vs Angular vs Vue. Medium. [Consulta: 12 outubro 2020]. Disponível em: <https://medium.com/zerotomastery/tech-trends-showdown-react-vs-angular-vs-vue-61ffaf1d8706/>.

AHUVIA, Y. 2019. Migrating from Angular to Vue. Medium. [Consulta: 12 outubro 2020]. Disponível em: <https://medium.com/fundbox-engineering/migrating-a-front-end-framework-and-still-committing-to-a-product-roadmap-angular-to-vue-1d5c00dd5709/>.

Anexos

Anexo A - Glossário

ARROW FUNCTION – forma de se escreverem funções com uma sintaxe mais compacta.

BACKUP – cópia de segurança.

CLASSE – estrutura que abstrai um conjunto de objetos com características semelhantes.

CLIENT-SIDE – sinónimo de front-end.

CLOUD – rede global de servidores remotos que operam como um ecossistema único.

COMMIT – encerramento de uma transação de modo a salvar alterações no código.

COMPONENT – pedaços de código reutilizáveis (parent = pai; child = filho; root = raiz).

DESIGN – projeção.

DEPLOY – passagem do software para outro ambiente, ex.: produção.

DIALOG – janela que se abre dentro do navegador.

DIV – container genérico.

DOWNLOAD – transferência.

ELDER-FRIENDLY – fácil de utilizar por parte de cidadãos sénior; interface simples.

ENGINE – programa de computador e/ou conjunto de bibliotecas, para simplificar e abstrair desenvolvimentos.

FORK – bifurcação; ramificação; acontece quando um programador inicia um projeto independente com base no código de um projeto já existente.

FRAMEWORK – conjunto de técnicas, ferramentas ou conceitos pré-definidos usados para resolver um problema de um projeto ou domínio específico.

FRONT-END – parte do software responsável por lidar com a entrada de dados do utilizador e adequá-la a uma especificação que o back-end possa utilizar.

FULLSTACK – junção de front-end e back-end.

GITHUB – plataforma de armazenamento de código com controlo de versões.

HARDWARE – componentes eletrónicos físicos de um sistema.

INPUT – entrada de dados.

INTERFACE – ambiente de interação entre o utilizador e o software.

LIBRARY – coleção de recursos usados por um software.

LINK – ligação; referência a outro documento.

LISTENER – recurso que “escuta” por um evento de modo a acionar uma função.

NULL – valor indeterminado/vazio.

ON-THE-FLY – durante o progresso.

OPEN SOURCE – código aberto.

PATH – caminho na ligação.

PLACEHOLDER – indicação dentro de uma caixa de texto.

PLATFORM-AS-A-SERVICE – serviço de hospedagem e implementação de hardware e software.

PLUGIN – programa usado para adicionar funções a outros programas

POLYLINE – objetos compostos por vários segmentos de linhas.

POSTMAN – programa que permite gerar API requests.

RENDERIZAÇÃO – processo pelo qual se obtém o produto final de um processamento digital.

REQUEST – requisição; solicitação.

ROUTING – encaminhamento de rotas.

RUNTIME ENVIRONMENT – ambiente de execução.

SCRIPT – texto com instruções de programação.

SCRUM – metodologia de desenvolvimento Ágil.

SETUP – instalação; configuração.

SOFTWARE – programa informático.

STACK – estrutura.

STATE MANAGEMENT – gestão de estados.

TAG – elemento HTML.

THIRD-PARTY – desenvolvimento não diretamente ligado a um produto primário.

TOKEN – chave eletrónica.

TYPESCRIPT – linguagem derivada de JavaScript.

UPLOAD – carregamento de dados.

VIEW – component que pode ser acedido através do VueRouter.