INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Internet Banking Application for Companies

Diogo André Machado Durães - 34501

Trabalho realizado sob a orientação de

Fábio Maia, ITSector

Professor Paulo Matos, Instituto Politécnico de Bragança

Mestrado em Informática

2021-2022

Internet Banking Application for Companies

Relatório da UC de Projeto

Mestrado em Informática

Escola Superior de Tecnologia e Gestão

Diogo André Machado Durães – 34501

2021-2022

A Escola Superior de Tecnologia e Gestão não se responsabiliza pelas opiniões expressas neste relatório.

Declaro que o trabalho descrito neste relatório é da minha autoria e é da minha vontade que o mesmo seja submetido a avaliação.

Diogo André Machado Durães - 34501

Dedicatória

Dedico este trabalho à minha família, namorada, amigos, professores de curso, colegas de trabalho da empresa ITSector e à própria empresa.

"A vida é uma jornada, ela não leva a um destino exatamente, mas a uma transformação."

Natan R. Silva

Agradecimentos

Agradeço aos meus familiares e namorada por toda a motivação e apoio do qual estou deveras grato, seja a nível psicológico como a nível financeiro. Não existe maneira de agradecer a todo o carinho, amor e educação que me foi transmitido.

Quero deixar aqui um grande agradecimento a todos os meus amigos pela motivação, apoio, ajuda e conselhos. Na vida não se tem muitos amigos, mas aqueles eu que tenho são suficientemente bons para compensar os que não tenho.

Gostaria também de deixar o meu obrigado a todos os colegas de trabalho da ITSector, que me ajudaram a crescer como profissional e transmitiram-me conhecimentos importantes para o futuro.

Contudo, os colegas de trabalho da empresa ITSector seguem valores e esses mesmos valores são representados profissionalmente por uma empresa de excelência chamada ITSector, do qual me orgulho bastante de fazer parte.

Não me podia esquecer de duas das pessoas mais importantes na realização desta tese, o orientador e professor Paulo Matos e o supervisor e mestre Fábio Maia. Vocês foram incansáveis durante toda a realização deste projeto, ajudaram-me bastante com dicas imprescindíveis de melhorias que podia fazer neste mesmo relatório.

Queria também agradecer a todos os Professores do Instituto Politécnico de Bragança que sempre me acompanharam, ajudaram e elevaram o meu conhecimento. Foram deveras importantes nesta etapa e foi graças a vocês que me consigo ver como um profissional.

Aos meus colegas de curso, um muito obrigado por tudo. Ficam muitas memórias, muitos conselhos e acima de tudo uma amizade.

Resumo

O projeto surgiu de uma necessidade do cliente (banco), onde era essencial agilizar e atender o máximo de pedidos sem necessitar muitas das vezes que uma agência formalize o pedido. De realçar que muito do tempo desse pedido é perdido em esperas.

Este projeto foi-nos proposto pela empresa ITSector e cujo foco é a oferta de soluções de transformação digital para a Indústria Financeira. O intuito é desenvolver um website em ReactJS para que as empresas possam fazer transações, pagamentos, requisição de documentos, entre muitas outras funcionalidades que serão detalhadas posteriormente.

O objetivo deste projeto é a evolução do site antigo do banco, seguindo sempre a abordagem pedida. Para isso, desenvolvem-se módulos (micro-frontend) e esses módulos serão introduzidos no website, de maneira que, o site atualmente em produção, não deixe de estar em funcionamento para o utilizador final.

Nesses módulos serão introduzidos novos estilos e funcionalidades, reutilizando-se sempre que possível, os serviços do website antigo, priorizando o funcionamento do mesmo e, incorporando toda a segurança na aplicação.

Assim, qualquer empresa registada e com conta no banco do cliente, pode efetuar o seu login e gerir as necessidades da empresa sem necessitar de consultar e/ou deslocar-se a uma agência, bem como desfrutar de funcionalidades inovadoras.

Palavras-chave: Internet Banking, Website, Banco e Utilizador Final.

Abstract

The project arose from the customer’s (bank) need, where it was essential to speed up and meet as many orders as possible without often requiring a branch to formalize the order. It is noteworthy that much of the time for this request is lost in waiting.

The company ITSector, whose focus is to offer digital transformation solutions for the Financial Industry, proposed this project to us. The aim is to develop a website in ReactJS so that companies can make transactions, payments, document requests, among many other features that will be detailed later.

The goal of this project is the upgrade of the bank's old website, always following the requested approach. For this, modules are developed (micro-frontend), and these modules will be introduced on the website, so that the website currently in production does not cease to work for the end user.

In these modules, new styles and functionalities will be introduced, reusing, whenever possible, the services of the old website, prioritizing its operation and incorporating all the security in the application. Thus, any company registered and with an account at the customer's bank, can log in and manage the company's needs without having to consult and/or go to a branch, as well as enjoy innovative features.

Keywords: Internet Banking, Website, Bank and End User.

Conteúdo

Índice

Lista de Tabelas

Lista de Figuras

Siglas

API	Application Programming Interface.
CA	Crédito Agrícola.
CPU	Central Process Unit.
CSS	Cascading Style Sheets.
DOM	Document Object Model.
ESTiG	Escola Superior de Tecnologia e Gestão.
HTML	Hyper Text Markup Language.
IDE	Integrated Development Environment.
IPB	Instituto Politécnico de Bragança.
JS	Java Script.
JSON	Java Script Object Notation.
PDF	Portable Document Format.
PR	Pull Request.
SGML	Standard Generalized Markup Language.
TS	Type Script.
UI	User Interface.
US	User Storie.
UX	User Experience.
VSC	Visual Studio Code.

Capítulo 1

Introdução

Ao longo das últimas décadas, as transações sociais e económicas têm-se tornado cada vez maiores e mais evidentes, este fenómeno deu-se muito à custa da evolução da mentalidade das pessoas e do progresso tecnológico.

Por estas razões, e visto que cada vez mais o conceito de globalização (fenómeno que nada mais é do que o processo de aprofundamento mundial de integração económica, social, cultural e política) está presente na atualidade e está a influenciar a sociedade atual, existe a necessidade dos bancos incorporarem nos websites funcionalidades inovadoras que só seriam possível até então, presencialmente.

Aliado a esta forte evolução daquilo que é a Internet, a população atual cada vez mais informada e com mais meios de acesso à mesma, tende a procurar, portanto acessos mais rápidos e eficientes para tratar de problemas que por vezes demorariam imenso tempo.

Segundo estudos do Instituto Nacional de Estatísticas (INE), em 2020, cerca de 84,5% dos agregados familiares em Portugal detém acesso à internet na sua residência, e que cerca de 81,7% da população utiliza ligação através de banda larga, sendo que a tendência é em aumentar [1].

Ainda neste estudo [1], tem-se que cerca de 44,5% das pessoas entre os 16 e os 74 anos efetuaram comércio eletrónico pela Internet.

Percebendo agora o crescimento da Internet como um comércio e com a vivência atual do estado pandémico, onde não é possível atender pessoalmente um número muito elevado de pessoas devido a limitação de espaço e tempo, percebe-se o porquê de maior parte dos comerciantes escolherem elevar o seu comércio para o comércio eletrónico.

1.1 Enquadramento

O setor bancário tal como outros setores a nível mundial viram-se obrigados a enquadrarem-se com esta nova abordagem, tornando assim o seu foco principal na evolução/transformação das suas ferramentas digitais.

A aplicação desenvolvida, vem culminar a falta desta nova abordagem, sendo que com esta, o banco vai usufruir daquilo que é o internet banking para empresas, um banco online aberto 24 horas por dia.

Para além das mais valias obvias, tais como verificar o histórico de movimentos, é possível fazer pagamentos, editar os dados da conta e agendar transações, as empresas podem ainda fazer créditos sem necessitar de contactar a agência bancária, ter uma página inicial personalizável, entre outras funcionalidades. Todas estas funcionalidades são pedidas e disponibilizadas pelo banco.

1.2 Objetivos

O proposto pela empresa ITSector é a criação de uma aplicação web em ReactJS de internet banking para empresas. O desenvolvimento desta plataforma web terá uma arquitetura micro-frontend, sendo estes de forma modular e incremental.

O objetivo que se revela no projeto apresenta-se por:

1. A partir das especificações acordadas com o cliente proceder à conceção, desenvolvimento e validação de soluções.

Considerando o objetivo principal deste estágio, foram detalhados os objetivos em:

1. Integração na equipa, nos métodos de trabalho e organização da empresa;
2. Saber fazer uso das ferramentas e tecnologias utilizadas pela empresa no desenvolvimento de projetos;
3. Aprender a analisar uma *user storie* e a negociar API’s com o Back-End;
4. Desenvolver componentes e funcionalidades para determinadas áreas operacionais do projeto em desenvolvimento (área um, dois, três e cinco).

5. Validar os componentes e funcionalidades desenvolvidas internamente (com equipa de backend) e com o cliente.
6. Testar os componentes e/ou páginas desenvolvidas.

1.3 Estrutura do documento

No capítulo um, apresenta-se uma breve introdução do tema proposto bem como os seus principais objetivos a efetivar.

Durante o capítulo dois fala-se sobre o conceito de internet banking, bem como contextualiza-se a metodologia de trabalho, a organização da equipa e o esquema da aplicação.

O terceiro capítulo contém a apresentação das tecnologias e ferramentas, bem como as vantagens e desvantagens.

O quarto capítulo descreve a abordagem, análise e modelação.

No capítulo quatro aborda-se o desenvolvimento/implementação e a correção de *bugs* deste projeto.

Por último, no capítulo da conclusão deste relatório, exprime-se a opinião relativamente ao projeto proposto, dá-se a conhecer as principais dificuldades que surgiram, bem como enumeram-se as propostas de melhoria e o trabalho futuro.

Capítulo 2

Contexto

2.1 Conceito Internet Banking

Para perceber o que o cliente (banco) pretende e tendo em conta que já existem algumas aplicações do género no mercado, inclusive a do próprio banco, foi necessário perceber, numa primeira instância, o conceito de internet banking para que assim consigamos expor algumas ideias ao cliente, possíveis melhorias, alternativas mais eficientes, entre outras opiniões construtivas.

O conceito Internet Banking, Online Banking ou e-banking são termos utilizados para caraterizar transações, pagamentos, verificações de histórico, agendamento de transações, entre outras funcionalidades. Isto durante 24 horas por dia e sendo acessível em qualquer lugar do mundo, sendo só necessário um equipamento informático (tablet, telemóvel ou computador) e acesso à Internet.

Neste caso em específico o banco pediu este tipo de abordagem, mas com um único foco, nas empresas.

Com esta plataforma intuitiva, e considerando que maior parte das empresas (principalmente empresas médias e pequenas) não têm tempo nem dinheiro para investir, esta aplicação mostra desde logo todos os passos necessários para um melhor investimento, ajuda as empresas facultando créditos, mostra os movimentos e investimentos anteriores e facilita o acesso rápido para tirar dúvidas com agentes especializados sem necessitar de marcação previa e deslocações.

2.2 Equipa de trabalho

A equipa de trabalho é composta por gestores, arquitetos, programadores (Back-end e Front-end), analistas de negócio (do banco) e *testers*.

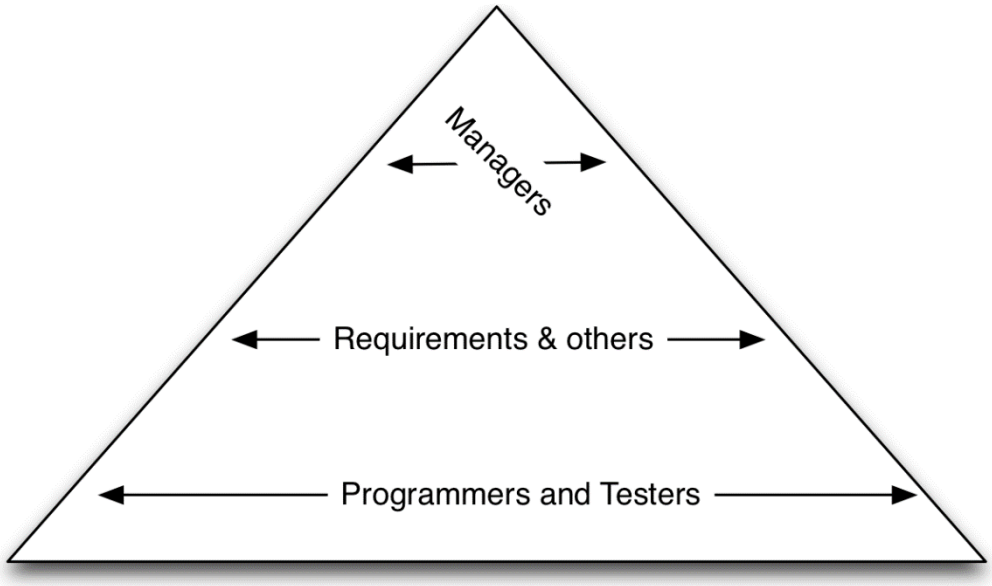

Figura 1 – Ilustração do esquema de pirâmide da esquipa de trabalho [2].

2.3 Metodologia de trabalho

A metodologia utilizada internamente pela ITSector, na maior parte dos projetos é o *Scrum*. O *Scrum* [3] é uma *framework* ágil, versátil e complexo, que tem como objetivo otimizar a Gestão de Projetos.

Para se perceber melhor o *Scrum* e o seu funcionamento é necessário entender primeiramente alguns termos, são eles:

- *Product Owner* (Dono do Produto) – é o responsável por coordenar a equipa de desenvolvimento e gerir o *Backlog;*
- Equipa de desenvolvimento – é a equipa que vai desenvolver o projeto;
- *Backlog* – é o conjunto de funcionalidades do produto, que vão sendo alteradas à medida que o projeto vai avançando;

- *Sprint* – é o ciclo de trabalho que gera uma entrega parcial. Normalmente é um período de semanas ou meses. Os projetos são constituídos por vários sprints;
- *Scrum Master* – é a pessoa que garante o funcionamento da *framework Scrum*. É responsável por assegurar que todos entendem e aplicam o *Scrum* corretamente;

Depois de percebido todos os termos básicos da *framework* e os seus significados, é importante ter em mente que o *Scrum* ainda contém termos importantíssimos no fluxo de trabalho, tais como: *sprint planning* (planeamento dos *sprints*), *daily scrum* (reunião diária), *sprint review* (revisão do sprint após a sua conclusão), *sprint retrospective* (reflexão de possíveis melhorias para impor nos próximos sprints), entre outros.

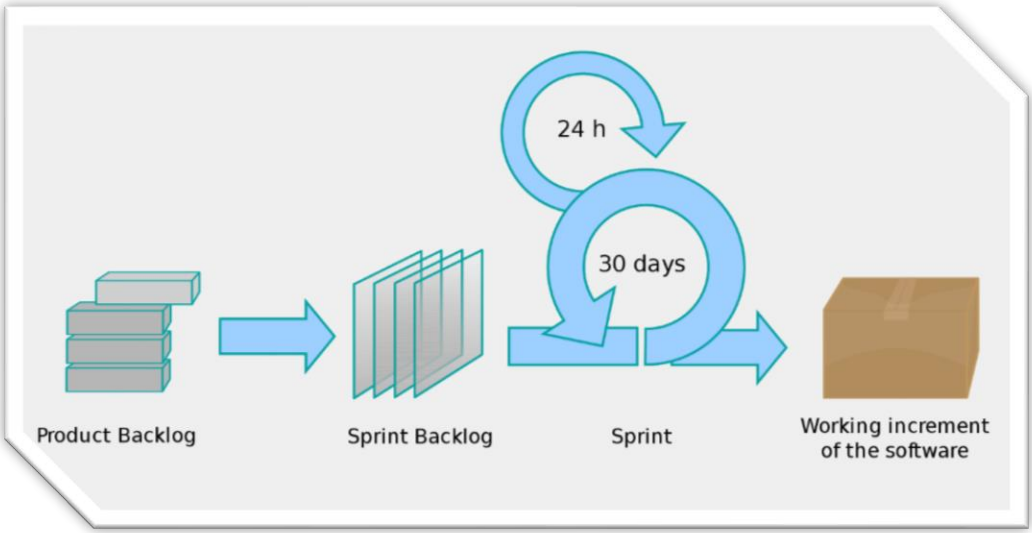

Figura 2 - Ilustração o funcionamento da metodologia *Scrum* [3].

As limitações do *Scrum* são as dependências externas, equipas geograficamente dispersas e produtos com necessidade de uma grande quantidade de testes.

Devido à falta de tempo imposta pelo banco para realizar todos estes eventos, o método utilizado, numa primeira instância, foi a metodologia *Waterfall* ou Cascata.

O processo desta metodologia é sequencial e uma vez que a etapa é considerada como concluída, não deve voltar para uma etapa anterior. Com esta, não há espaço para mudanças ou erros pois podem comprometer parte ou todo o projeto.

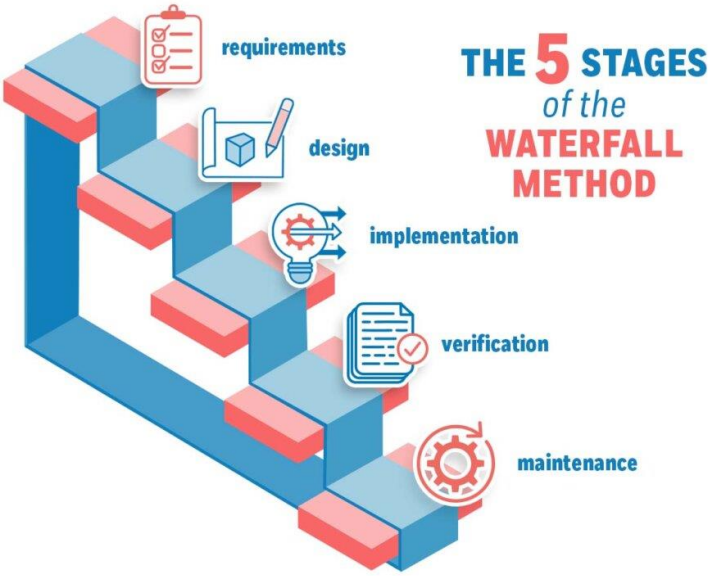

Figura 3 - Figura representante da metodologia *Waterfall* [4].

Atendendo às diferenças entre ambas as metodologias é percetível que o *Scrum* para desenvolvimento de produtos (*software*), é mais sustentável e interessante.

Decidido a metodologia, foi efetuada a primeira reunião de arranque (*kick-off*) do projeto onde foram apresentados os requisitos funcionais já expostos em forma de *user stories*, algumas delas careciam de uma confirmação da equipa de Front-end e do arquiteto de projeto, quanto a requisitos não funcionais.

Numa fase posterior, existiu uma necessidade de alterar a metodologia de trabalho, devido à sustentabilidade do projeto, passando a ser o *Scrum*.

2.4 Organização da equipa de trabalho

Os analistas de negócio do banco fornecem as *user stories* ao gestor e ao arquiteto. Enquanto o gestor trata de gerir todo o fluxo de trabalho, por sua vez, o arquiteto define a arquitetura do projeto, desde as tecnologias necessárias, aos procedimentos que devem ser seguidos, isto em casos muito específicos.

Posteriormente às decisões tomadas de arranque de projeto (*kick-off*), as *user stories* são fornecidas aos programadores.

Cabe a cada programador, numa primeira instância, fazer a análise das *user stories* em que está inserido. Os programadores Front-end começam a implementação e durante a implementação, são negociados os contratos das API's com os programadores Back-end.

Durante todo o projeto, existiram reuniões diárias entre os analistas de negócio do banco e toda a equipa de trabalho, onde eram expostas dúvidas de negócio, bem como, algumas propostas de melhoria por parte da equipa de desenvolvimento.

Da reunião de arranque de projeto, teve-se que algumas das tecnologias e ferramentas necessárias ao projeto são: o React, TypeScript, Redux, Redux-Saga, Jest, Azure DevOps, entre outras abordadas no capítulo três do presente relatório.

Depois de desenvolvida toda a aplicação, faz-se vários testes (testes de unidade e teste aos componentes) ao fluxo antes mesmo de os *testers* fazerem essa validação. O objetivo desta etapa é encontrar maior parte dos bugs e corrigi-los.

Posteriormente, os *testers* testão pormenorizadamente a aplicação no ambiente de qualidade e reportam os bugs, se existissem. Caso existam, são corrigidos pela equipa responsável (frontend, backend ou equipa do banco).

No final, os próprios analistas do negócio percorrem todo o fluxo para confirmar que está tudo de acordo e pedem aos programadores para subirem o código ao ambiente de produção.

2.5 Esquema da aplicação

Esta aplicação, inicialmente, vai ter como foco a web, necessitando, no entanto, que todos os componentes desenvolvidos sejam responsivos e possam ser utilizados, numa fase à posteriori pelo banco, para a criação de uma aplicação móvel.

O trabalho proposto prossupõe que existam dois tipos de utilizadores, são eles:

- Consultor;
- Autorizador (Empresa).

A aplicação é dividida em cinco áreas, são elas, a área um (área de *landing*), área dois (gestão de consultores), área três (gestão de projetos), área quatro (área de movimentos) e por último, a área cinco (área de investimentos).

Capítulo 3

Tecnologias/Ferramentas

3.1 Ferramentas

As ferramentas utilizadas vão ser apresentadas nos subcapítulos, tal como o motivo do uso das mesmas face ao projeto elaborado.

3.1.1 Azure DevOps

O Azure DevOps [5] é um produto da Microsoft que providencia de um conjunto de processos que reúnem programadores, gestores de projetos e colaboradores. Este permite que as organizações criem e melhorem produtos mais rapidamente comparativamente às abordagens tradicionais de desenvolvimento de software.

Para este projeto, este foi utilizado como repositório para o código implementado, é neste que constam as *user stories* e onde ocorre toda a gestão do projeto, desde planeamento das *sprints* às *queries* (com estas conseguíamos de forma mais rápida acompanhar o estado dos bugs), entre outras funcionalidades.

O Azure DevOps é uma ferramenta muito extensa e foi necessário uma formação interna para conhecermos as funcionalidades da mesma, bem como, melhorar a produtividade face ao poder que o mesmo tem.

No Azure DevOps [5], as vantagens são:

- Possibilidade de customização;
- Fácil uso;
- Muito intuitivo;

- Contém funcionalidades que se destacam (funcionalidades essas mencionadas anteriormente);
- Segurança e fiabilidade.

As desvantagens do Azure DevOps [6] são:

- A configuração avançada é de difícil perceção inicialmente;
- A documentação do mesmo é fraca e/ou desatualizada.

3.1.2 Figma

O Figma [7] é um editor gráfico de vetor, prototipagem de interface gráfica e desenvolvimento UI/UX (experiência da interface de utilizador) permitindo o desenvolvimento colaborativo baseado num navegador web, no entanto tem também ferramentas offline para aplicações desktop de sistemas operativos tais como Linux, MacOS e Windows.

Este foi utilizado para visualizarmos os ecrãs, as dimensões e o comportamento de cada componente.

Quanto as suas vantagens [8] são:

- Fácil de partilhar o desenvolvimento com o resto da equipa;
- É gratuito;
- Componentes interativos;
- De fácil uso;
- Contém uma enorme lista de plugins.

No entanto, as suas desvantagens [8] são:

- Não suporta animações e interações avançadas.
- É lento em alguns sistemas operativos.

3.1.3 Swagger

O Swagger [9] é uma linguagem de descrição de interface que descreve API's RESTful usando JSON. É usado em conjunto com ferramentas de código aberto para projetar, construir, documentar e usar serviços da web RESTful.

Para este projeto, o Swagger contém toda a documentação dos serviços fornecidos pelo Back-end.

As vantagens do Swagger [10] são:

- Boa Documentação;
- É gratuito;
- Facilmente ajustável;
- É ótimo para compartilhar com a equipa;
- Legível por humanos e pela máquina.

Quanto as desvantagens [10] são:

- Não dá para escrever testes;
- Não está definido para criar eventos e consumir API's.

3.1.4 Visual Studio Code

O Visual Studio Code (VSC) [11] é um ambiente de desenvolvimento integrado feito pela Microsoft para Windows, Linux e macOS. Os recursos incluem depuração, destaque de sintaxe, *snippets*, entre outros recursos que nos acompanhou para o resto do projeto, e nos ajudou bastante.

O Visual Studio Code é a ferramenta onde foi produzido todo o código que posteriormente foi colocado no repositório (Azure DevOps) com auxílio do Git.

Os prós [12] que se averiguou para este foram as seguintes:

- Ambiente com um conjunto vasto de funcionalidades;
- Gratuito, rápido e intuitivo;
- Contém um amplo ecossistema de plugins e com grande suporte;
- Tem o Git integrado;
- Suporta várias linguagens de programação.

No entanto, tem como desvantagens a limitação da sua capacidade de *debug* e ao instalar muitos plugins tona-se instável.

3.1.5 Git

O Git [13] é um sistema de controle de versões usado principalmente no desenvolvimento de aplicações. Em vez de ter um único local para o histórico de versão, este também é um repositório que pode conter o histórico completo de todas as alterações.

No Git, as principais vantagens [14] são:

- Contém operações rápidas;
- Capacidade de adicionar/alterar/remover ficheiros;
- Confiabilidade;
- Redução de custos devido ao recurso de um servidor.

Quanto as desvantagens [14] é a complexidade, a curva de aprendizagem, não tem interface gráfica e, por vezes, é lento.

3.2 Tecnologias

3.2.1 ReactJS

O ReactJS [15] é uma biblioteca JavaScript (JS) declarativa, eficiente, flexível e de código aberto com o intuito de fornecer ao utilizador a possibilidade de criar interfaces em páginas web.

Esta biblioteca foi criada pela equipa do Facebook e é utilizada em empresas gigantes tais como a Netflix, Walmart, Instagram, pelo próprio Facebook, entre outros.

As vantagens [16] deste são:

- Não tem padrão de arquitetura, tornando-se flexível;
- Facilidade de migrar entre versões;
- Reaproveitamento do código;
- A enorme comunidade que usufrui da mesma, que disponibiliza bibliotecas;
- Permite utilizar funções do JavaScript e a tipificação do TypeScript.

Quanto as suas desvantagens [16], tem-se que:

- Linha de aprendizagem (requer alguns conhecimentos à posteriori);
- O tamanho da biblioteca;
- Não utiliza uma abordagem isomorfa.

A linguagem escolhida para todo o projeto foi o ReactJS, isto implica o uso das seguintes linguagens:

- **HyperText Markup Language (HTML) [17]** - é uma linguagem utilizada para a construção de páginas web e a inserção de novos conteúdos, nomeadamente, imagens e vídeos com recurso a hipertextos.

- **Cascading Style Sheets (CSS) [18]** - é uma linguagem de estilos que serve para definir a parte visual de um website. Esta separa a parte do conteúdo da parte gráfica da mesma, manipulando o visual de documentos HTML (páginas).

- **JavaScript (JS) [19]** - é uma linguagem de programação de alto nível, leve, compilada simples e dinâmica. É também uma linguagem multiparadigma, baseada em protótipo, que suporta a orientação a objetos, imperativos e declarativos. A grande maioria dos sites utiliza esta mesma linguagem porque esta permite criar conteúdo que atualiza dinamicamente, controlar multimédias, imagens animadas, entre outras funcionalidades.

- **TypeScript (TS) [20]** - é uma linguagem de programação desenvolvida e é atualizada pela Microsoft. É uma linguagem de programação fortemente tipificada que se baseia em JavaScript. Esta linguagem é útil para grandes aplicações. Deve ser usada para aplicações em JavaScript para a execução do lado do cliente e do servidor. Devido à tipificação desta, as aplicações tornam-se melhores e mais coesas.

Não obstante o ReactJS ser naturalmente abordado em JavaScript, na empresa (ITSector) por questões de garantias, qualidade de código e verificação, temos como requisito interno não funcional a utilização do TypeScript.

3.2.2 Redux

O Redux [21] é uma biblioteca JavaScript de código aberto com o intuito de manusear e centralizar o estado da aplicação. É mais utilizado por bibliotecas como o React ou o Angular para desenvolver as suas interfaces.

Os seus prós [22] são:

- Benefícios de desempenho;

- Fácil de depuração;

- Persistência do estado.

Os seus contras [22] são:

- A complexidade para o uso do mesmo;
- A sobrecarga em projetos pequenos.

3.2.3 Redux-Saga

O Redux-Saga [23] é uma biblioteca de *middleware* usada para permitir que o Redux interaja com recursos externos de forma assíncrona. Esta faz pedidos HTTP para serviços externos, acede ao armazenamento do navegador e controla operações de entrada e saída.

Os prós [24] do seu uso são:

- Simplicidade de testes;
- Estilo declarativo;
- Agilidade no tratamento de falhas/erros.

Os contras [24] de utilizar Redux-Saga são:

- A necessidade de aprendizagem de alguns conceitos;
- Sintaxe gerada;
- Estabilidade da API.

3.2.4 Jest

O Jest [25] é uma *framework* de testes JavaScript que é atualizada pelo grupo Facebook, atualmente denominado de Meta. O seu foco principal é a simplicidade e suporte para testes em grandes aplicações web. Esta *framework* suporta projetos que usam Babel, TypeScript, Node.js, React, entre outros.

No Jest, as vantagens [26] são:

- *Open source* (código aberto);

- Não necessita de grandes configurações;
- É possível tirar *snapshots* com a mesma;
- Os testes são paralelizados (maximiza a performance);
- Rápido e seguro.

As desvantagens são:

- Não dá suporte a todas as bibliotecas;
- Para projetos grandes demora muito tempo.

3.2.5 Enzyme

O Enzyme [27] é uma *framework* de testes JavaScript para o React em que o seu objetivo é ajudar a melhorar os testes dos componentes do React. Com esta, pode-se afirmar, manipular e analisar saída dos componentes React.

Este foi criado pela Airbnb e adiciona alguns métodos excelentes para "renderizar" um componente (ou vários), encontrar elementos e interagir com os elementos.

As vantagens do Enzyme [28] são:

- Testes com "renderização" superficial são executados mais rapidamente;
- Os testes unitários têm influência no código escrito;
- Mostra onde existe a falha e o que provocou a mesma.

Por outro lado, as desvantagens [28] são:

- É construído com base no react-dom personalizado, não existindo suporte imediato aos novos componentes;
- Requer uma maior configuração e manutenção conforme a evolução das versões;

- Os detalhes dos testes focam mais na implementação (estado e propriedades);
- Caso exista uma reformulação da aplicação os testes são corridos novamente.

3.2.6 React Testing Library

O React Testing Library [29] é uma *framework*, muito leve, de testes para React DOM simples e complexo, adicionando assim boas práticas de teste. Este, fornece funções cuja utilidade é testar todos os componentes do React, no entanto, é construído em cima do react-dom e react-dom/test-utils, forçando que os testes aos componentes sejam na perspetiva do utilizador.

Quanto às suas vantagens [30] são:

- Ótima experiência do utilizador;
- É mais fiável porque testa os componentes como se fosse o utilizador;
- Limita a interação com os elementos, forçando a que todos os testes sejam com código totalmente compatível;
- É recomendado pela própria equipa do Facebook (equipa que criou o React);
- Com esta é mais fácil e mais rápido fazer testes;
- É bem documentada a *framework*;
- Tem um grande suporte da comunidade.

O contra de utilizar esta *framework* é que os testes são mais profundos, pelo que pode ser mais lento dos que os outros acima mencionados, daí a escolha passa por utilizar todos os referidos, dependendo da necessidade do teste no componente.

3.2.7 NodeJS

O NodeJS [31] é um programa que executa código JavaScript em *single-thread* do lado do servidor. É usado para criar aplicações não existindo a necessidade de um browser para a execução.

Apesar de recente, o NodeJS já é utilizado por grandes empresas no mercado de tecnologia, como Netflix, Uber, NASA e LinkedIn.

As vantagens do NodeJS [32] são:

- Tem um ecossistema enorme, ativo e de código aberto;
- Flexibilidade, recursos necessários e a produtividade;
- Tempo de resposta rápido.

Em contrapartida, as desvantagens associadas são:

- Perda de memória em processos de execução longa;
- É necessário pensar na escalabilidade do projeto antes de escolher esta ferramenta;
- O *loop* de eventos torna o tratamento de erros assíncrono mais difícil.

Capítulo 4

Abordagem/Análise/Modelação

Neste ponto do projeto, apresentou-se o levantamento e análise de requisitos. No entanto, foi necessário fazer a separação dos requisitos, dividindo os mesmos em requisitos funcionais e requisitos não funcionais.

4.1 Requisitos Funcionais vs Requisitos Não Funcionais

Os requisitos funcionais são funcionalidades de um produto (*software* ou componente). Este, representa o que é feito no mesmo, desde tarefas a serviços.

Já os requisitos não funcionais, são requisitos dos quais muitas das vezes a pessoa/empresa que pede um serviço/aplicação não reflete sobre e a equipa de desenvolvimento deve ter em consideração pois pode influenciar de forma negativa no funcionamento da aplicação.

Alguns requisitos não funcionais são o tempo de processamento, padrões, usabilidade, portabilidade, confiabilidade, qualidade, entre muitos outros.

Após a primeira reunião de arranque do projeto, sucederam-se mais algumas sessões de trabalho solicitadas pela equipa de desenvolvimento, tendo por objetivo debater melhorias dos requisitos funcionais, cujo foco principal seria resolver alguns problemas posteriores, nomeadamente de requisitos não funcionais.

Foi igualmente definido nas reuniões que, tal como foi indicado no quarto capítulo, existem dois atores, são eles: o consultor e o autorizador (empresa).

4.2 User Stories

As *User Stories* apresentadas pelos analistas de negócio do banco e sobre as quais incidiu o trabalho, desenvolvimento de componentes e/ou páginas da área um, dois, três e cinco, são:

- Autorizador (Empresa):

Identificador	Nome	Prioridade	Descrição
USC01	Gerir projeto	Muito alto	Como Autorizador quero ser capaz de gerir o(s) projeto(s) da empresa.
USC02	Registar projeto	Muito alto	Como Autorizador quero poder registar projeto(s) da empresa.
USC03	Ver projetos	Muito alto	Como Autorizador quero ser capaz de ver os projetos da empresa.
USC04	Gerir consultor	Muito alto	Como Autorizador quero conseguir gerir o(s) consultor(es).
USC05	Adicionar consultor	Muito alto	Como Autorizador quero ser capaz de adicionar consultor(es) se o(s) à rede.
USC06	Ver consultores	Muito alto	Como Autorizador quero ser capaz de ver os consultores que tenho associados.
USC07	Convidar consultor	Muito alto	Como Autorizador quero poder convidar consultor a registar-se na plataforma.
USC08	Conceber acesso ao consultor	Muito alto	Como Autorizador quero conseguir conceber acesso(s) ao(s) consultor(es).
USC09	Remover acesso ao consultor	Muito alto	Como Autorizador quero poder remover acesso ao(s) consultor(es).
USC10	Ver movimentos	Muito alto	Como Autorizador quero ser capaz de ver os movimentos das contas.
USC11	Ver investimentos	Muito alto	Como Autorizador quero ser capaz de ver os investimentos feitos.
USC12	Classificar os investimentos	Muito alto	Como Autorizador quero poder classificar os investimentos.
USC13	Gerir ficheiros dos investimentos	Muito alto	Como Autorizador quero poder gerir ficheiros dos investimentos.
USC14	Anexar ficheiros aos investimentos	Muito alto	Como Autorizador quero poder anexar ficheiros aos investimentos.
USC15	Remover ficheiros dos investimentos	Muito alto	Como Autorizador quero poder remover ficheiros aos investimentos.
USC16	Ver notificações	Alto	Como Empresa quero ser capaz ver as novas notificações.
USC17	Gerir notas dos ficheiros nos investimentos	Médio	Como Autorizador quero poder gerir notas dos ficheiros nos investimentos.
USC18	Adicionar notas aos ficheiros dos investimentos	Médio	Como Autorizador quero poder adicionar notas aos ficheiros dos investimentos.

USC19	Editar notas dos ficheiros nos investimentos	Médio	Como Autorizador quero poder editar notas dos ficheiros nos investimentos.
USC20	Eliminar notas dos ficheiros nos investimentos	Médio	Como Autorizador quero poder eliminar notas dos ficheiros nos investimentos.
USC21	Gerir notas dos investimentos	Médio	Como Autorizador quero poder gerir notas dos investimentos.
USC22	Adicionar notas aos investimentos	Médio	Como Autorizador quero poder adicionar notas aos dos investimentos.
USC23	Editar notas dos investimentos	Médio	Como Autorizador quero poder editar notas dos investimentos.
USC24	Eliminar notas dos investimentos	Médio	Como Autorizador quero poder eliminar notas dos investimentos.
USC25	Fazer download de um documento com todos os investimentos	Médio	Como Autorizador quero fazer download de um documento com todos os investimentos.
USC26	Filtrar os investimentos	Baixo	Como Autorizador quero poder filtrar os investimentos.
USC27	Verificar o estado da inserção do consultor	Baixo	Como Autorizador quero conseguir verificar o estado da inserção do consultor.

Tabela 1 - Ilustração das *user stories* do Autorizador.

- Consultor:

Identificador	Nome	Prioridade	Descrição
USC01	Escolher a empresa	Muito alto	Como Consultor quero poder escolher a Empresa para prestar serviço.
USC02	Alterar de conta	Muito alto	Como Consultor quero poder alterar de conta da Empresa ao qual está a prestar serviço.
USC03	Gerir projetos	Muito alto	Como Consultor quero ser capaz de gerir os projetos aos quais tenho acesso.
USC04	Ver projetos	Muito alto	Como Consultor quero ser capaz de ver os projetos aos quais tenho acesso.
USC05	Registar projeto	Muito alto	Como Consultor quero poder registar projetos da Empresa.
USC06	Ver movimentos	Muito alto	Como Consultor quero ser capaz de ver os movimentos das contas da Empresa se a mesma lhe conceber acesso.
USC07	Ver investimentos	Muito alto	Como Consultor quero ser capaz de ver os investimentos feitos pela Empresa se a mesma lhe conceber acesso.
USC08	Classificar os investimentos	Muito alto	Como Consultor quero poder classificar os investimentos se a Empresa lhe conceber acesso.
USC09	Gerir ficheiros dos investimentos	Muito alto	Como Consultor quero poder gerir ficheiros dos investimentos

USC10	Anexar ficheiros aos investimentos	Muito alto	Como Consultor quero poder anexar ficheiros aos investimentos se a Empresa lhe conceber acesso.
USC11	Remover ficheiros dos investimentos	Muito alto	Como Consultor quero poder remover ficheiros aos investimentos se a Empresa lhe conceber acesso.
USC12	Ver notificações	Alto	Como Consultor quero ser capaz ver as novas notificações.
USC13	Gerir notas dos ficheiros nos investimentos	Médio	Como Consultor quero poder gerir notas dos ficheiros nos investimentos se a Empresa lhe conceber acesso.
USC14	Adicionar notas aos ficheiros dos investimentos	Médio	Como Consultor quero poder adicionar notas aos ficheiros dos investimentos se a Empresa lhe conceber acesso.
USC15	Editar notas aos ficheiros dos investimentos	Médio	Como Consultor quero poder editar notas aos ficheiros dos investimentos se a Empresa lhe conceber acesso.
USC16	Eliminar notas aos ficheiros dos investimentos	Médio	Como Consultor quero poder eliminar notas aos ficheiros dos investimentos se a Empresa lhe conceber acesso.
USC17	Gerir notas aos investimentos	Médio	Como Consultor quero poder gerir notas aos investimentos se a Empresa lhe conceber acesso.
USC18	Adicionar notas aos investimentos	Médio	Como Consultor quero poder adicionar notas aos dos investimentos se a Empresa lhe conceber acesso.
USC19	Editar notas dos investimentos	Médio	Como Consultor quero poder editar notas dos investimentos se a Empresa lhe conceber acesso.
USC20	Eliminar notas dos investimentos	Médio	Como Consultor quero poder eliminar notas dos investimentos se a Empresa lhe conceber acesso.
USC21	Remover ficheiros dos investimentos	Médio	Como Consultor quero poder remover ficheiros dos investimentos se a Empresa lhe conceber acesso.
USC22	Fazer download de um documento com todos os investimentos	Médio	Como Consultor quero fazer download de um documento com todos os investimentos se a Empresa lhe conceber acesso.
USC23	Filtrar os investimentos	Baixo	Como Consultor quero poder filtrar os investimentos se a Empresa lhe conceber acesso.

Tabela 2 - Representação das *user stories* do Consultor.

4.3 Digrama de Casos de Uso

Em resumo, o diagrama de casos de uso com base nas *user stories* que os analistas de negócio do banco forneceram e sobre as quais incidiu o trabalho (desenvolvimento de componentes e/ou páginas da área um, dois, três e cinco) está representado na figura quatro e cinco.

Seguiu-se com as permissões de cada um desses atores, e assim obtém-se o seguinte:

- Consultor: gere projetos, movimentos e investimentos da empresa. Este consegue ter acesso multiempresa.
- Autorizador (Empresa): atribui acesso aos consultores, consegue acompanhar o processo da gestão de projetos, movimentos e investimentos, bem como, consegue adicionar ficheiros e notas a documentos e investimentos, de forma que os consultores consigam perceber de que se trata, tornando-se numa plataforma colaborativa.

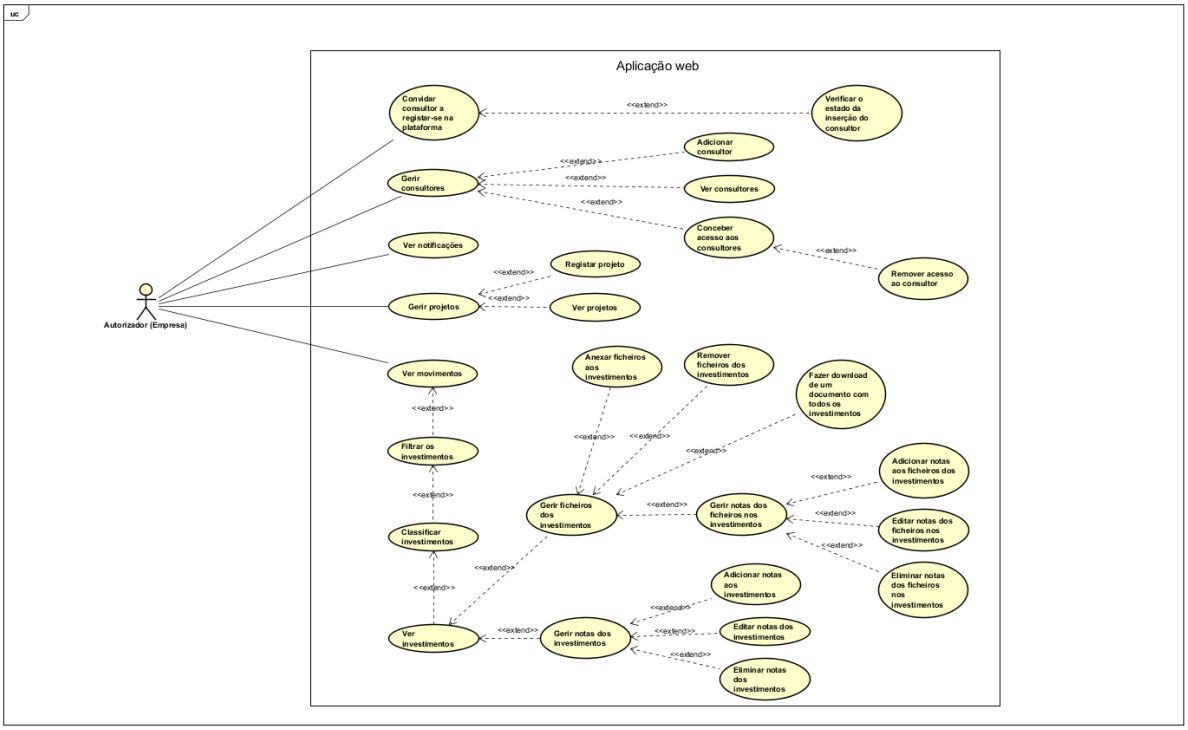

Figura 4 - Diagrama de Casos de Uso do Autorizador da Empresa.

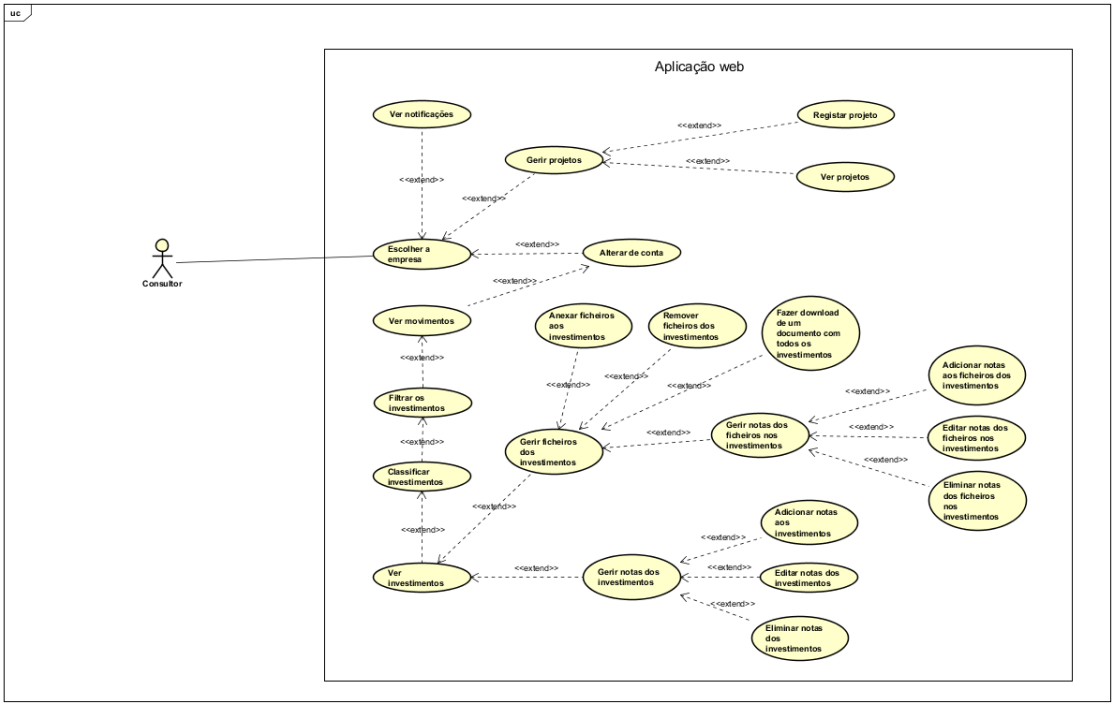

Figura 5 - Diagrama de Casos de Uso do Consultor.

4.4 Arquitetura do projeto

Quanto à arquitetura do projeto é composta por *micro-frontend* e *micro-serviços*, sendo estes de forma modular e incremental.

O funcionamento da arquitetura utilizada está representado na seguinte figura.

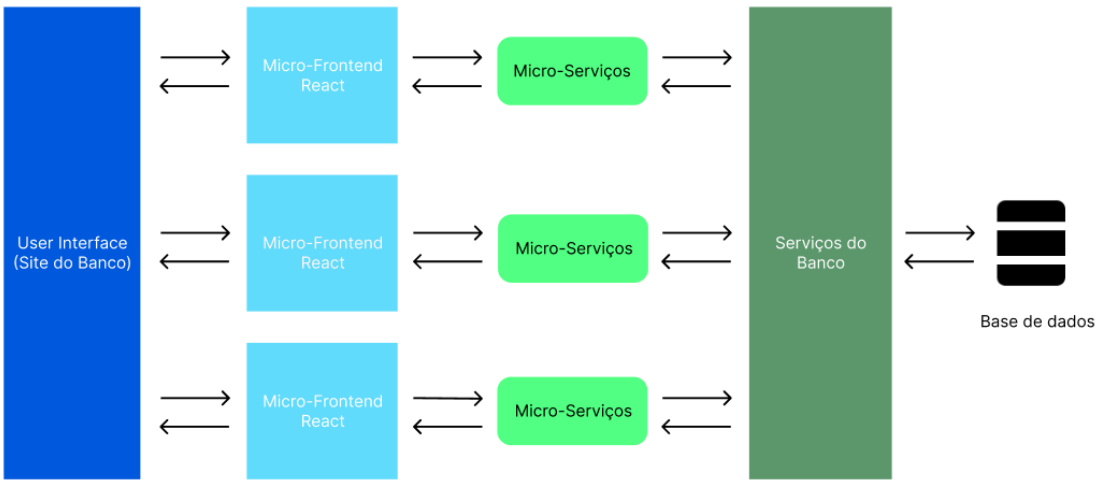

Figura 6 – Figura que ilustra a arquitetura do projeto.

A escolha desta arquitetura deve-se ao facto de que o projeto é bastante grande e, portanto, exigia que existissem várias equipas. Para que essas mesmas equipas conseguissem trabalhar de forma coordenada.

Para funcionar esta arquitetura, os *micro-frontends* desenvolvidos em ReactJS são integrados no website e não dependem uns dos outros. Estes são independentes podendo usar um ou mais serviços para a obtenção de dados e processar operações.

A intenção de seguir esta abordagem é para conseguir maior agilidade e flexibilidade com o objetivo de utilizar numa primeira fase a abordagem antiga e ser posteriormente substituída pela nova abordagem.

4.5 Wireframes

Neste subcapítulo estão apresentados alguns dos wireframes e a descrição de cada página da aplicação. Devido à necessidade de anonimato do banco, exposta pela ITSector, todos os ecrãs e componentes encontram-se com cores neutras e sem que seja possível identificar o mesmo.

Na figura sete está representado o ecrã de *landing*, ou seja, o ecrã que é apresentado após o utilizador proceder ao *login* no website. Ainda, na área um, contempla a figura oito que representa as configurações ao qual o utilizador tem acesso.

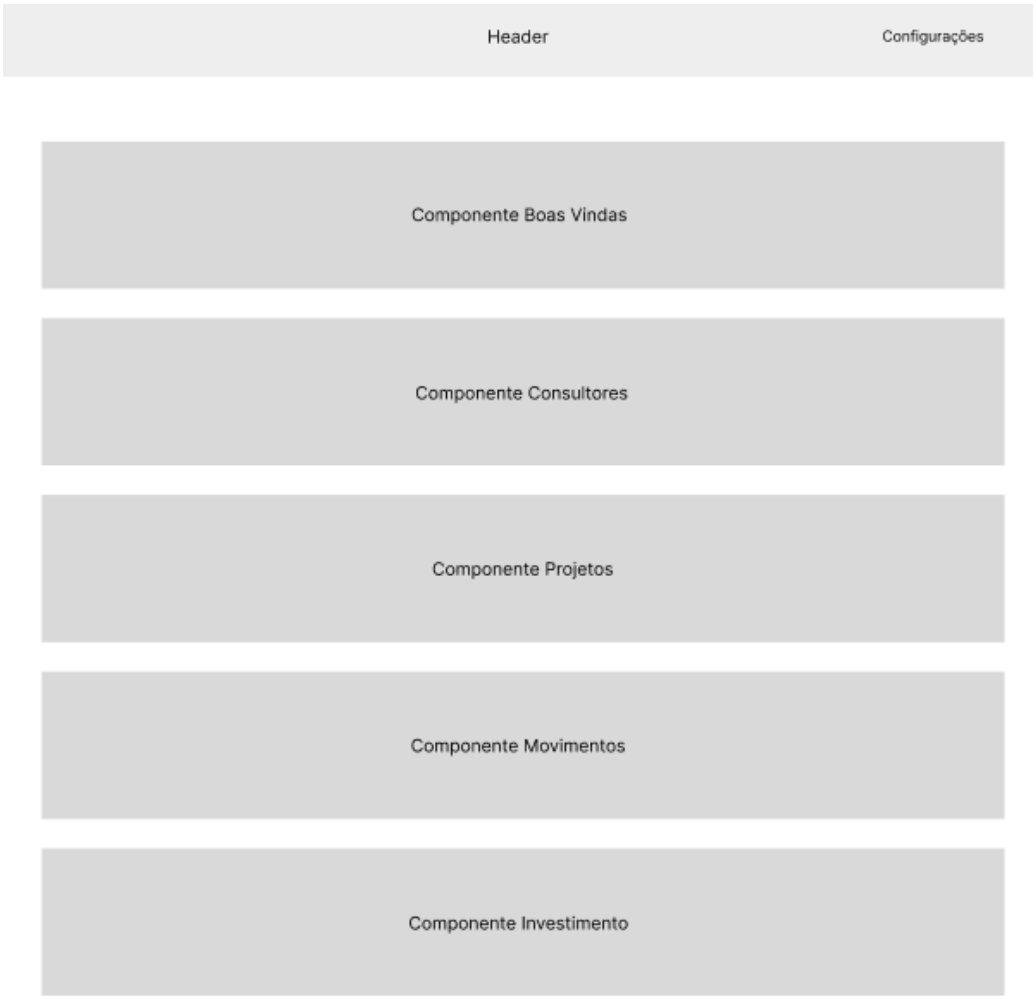

Figura 7 – Representação do ecrã de *landing*.

Figura 8 – Ilustração do ecrã de configurações.

Seguido a área dois, nesta onde é possível fazer toda a gestão dos consultores, desde adicionar um novo, consultar os consultores associados, editar acessos e até mesmo removê-lo da empresa, tal como é possível verificar na figura nove.

Figura 9 – Ilustração do ecrã de gestão de consultores.

Seguido da área dos projetos, onde é possível criar projetos novos, consultá-los, editá-los e removê-los (ver figura dez).

Figura 10 – Ilustração do ecrã de gestão de projetos.

Passando para a gestão dos movimentos efetuados, onde consta uma tabela ordenada com os movimentos, é possível verificar os movimentos das contas da empresa, associar documentos, associar a projetos e adicionar/editar/remover notas (ver figura onze).

Na área de gestão de movimentos contém um componente de pesquisa para ajudar o utilizador a conseguir encontrar o movimento de forma mais rápida e fácil. A pesquisa neste componente é feita de várias formas (por nome ou número dos movimentos, por montante e por data).

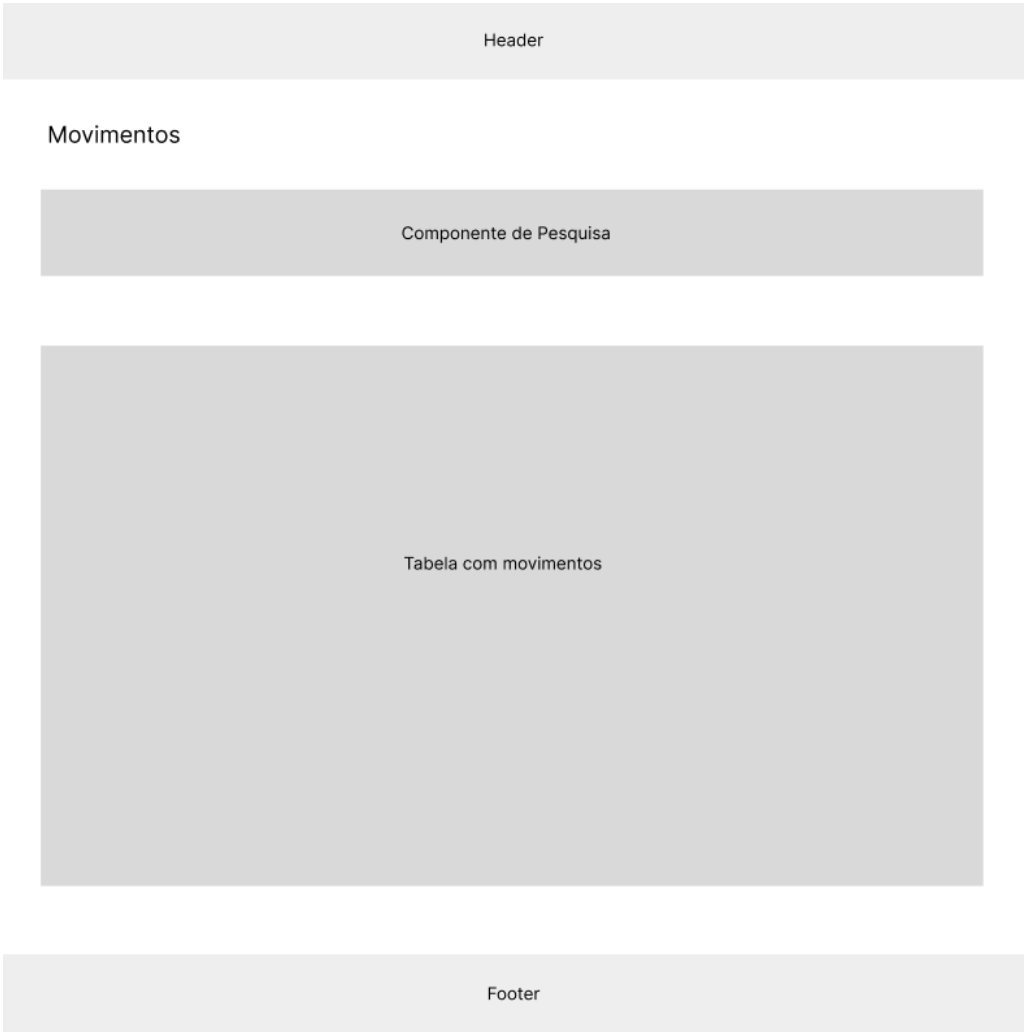

Figura 11 – Representação do ecrã de movimentos (área quatro).

Prosseguindo, chegamos à última área, a gestão de investimentos (figura doze), onde constam todos os investimentos da empresa em detalhe, gerir documentos, adicionar notas aos mesmos, associar os investimentos a projetos e efetuar *download* dos documentos.

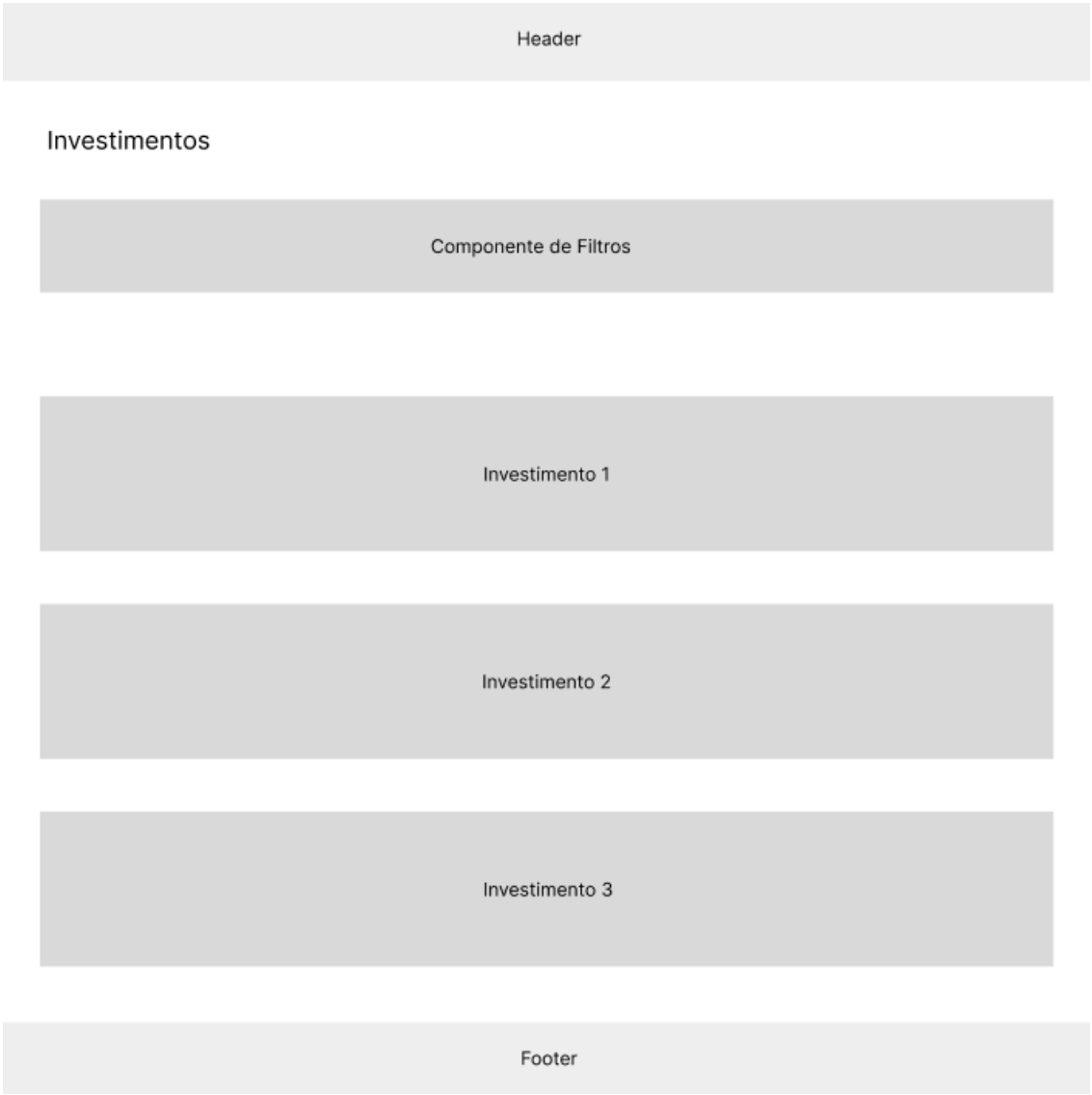

Figura 12 – Ilustração do ecrã de investimentos (área cinco).

Capítulo 5

Desenvolvimento/Implementação

Iniciando a parte de desenvolvimento, foi exigido pelo banco a utilização de duas *frameworks* internas.

Uma delas forçou a utilização de componentes partilhados do banco para todos os *sites* deles (Ícones, Cores, Temas, entre outros), de forma que todos os sites tenham o mesmo estilo, cores, temas, ícones e alguns componentes. A outra *framework* cria uma estrutura interna inovadora.

Sabendo da necessidade de utilização destas duas *frameworks* e tendo em conta que a equipa sénior tinha uma arquitetura definida, foi necessário a implementação de um pequeno site local para o uso das mesmas, ambicionando a prática e acima de tudo, os testes da arquitetura de Front-end com estas.

Após a criação do *site* local de teste e depois do teste de várias arquiteturas, chegou-se à conclusão de qual seria a melhor arquitetura de Front-end.

A par dos desenvolvimentos dos serviços, foram implementados alguns dos componentes necessários para a aplicação, esta decisão foi pretendida pelo banco devido ao pouco tempo que restava para desenvolver toda a aplicação, já que este projeto era prioritário e importantíssimo.

5.1.1 Padrão de desenvolvimento

Sendo assim, na primeira fase do projeto, antes mesmo de implementar qualquer componente, foi dividido o projeto em cinco áreas distintas e desenvolvido o padrão de desenvolvimento do Front-end, separando as áreas por pastas, de forma que seja mais percetível

para qualquer programador que necessite de integrar o projeto e para melhor organização dentro da equipa de programadores.

Foi necessário instalar todas as linguagens e ferramentas descritas acima, bem como, configurar as ferramentas, como por exemplo, atualizar o NodeJs para a versão recomendada pela equipa de desenvolvimento sénior, configurar o git, configurar a versão de ReactJS, entre outras.

Tendo a arquitetura de Front-end definida e exposta para toda a equipa no Azure, começou-se o desenvolvimento, para isso os programadores fizeram o clone da arquitetura do projeto.

No IDE, Visual Studio Code, foram instalados alguns *plugins* para formatação de código, de forma que toda a equipa ao enviar o código para o repositório, envia-se formatado nos estilos definidos.

5.1.2 Funcionamento do Git

De referir que no repositório contém três *branches* de três ambientes distintos, sendo eles, o ambiente de desenvolvimento, de qualidade e de produção.

Para iniciarmos a implementação, cada programador tinha de criar uma *branch* baseada no ambiente de desenvolvimento, os passos são os seguintes:

- Fazer *pull* após a criação da nova *branch;*
- Fazer *checkout* para a nova *branch*, usando o comando "git checkout feature/numero-descricao", onde o número correspondente ao número da *user storie* e a descrição é relativa ao que foi implementado;
- Produzir o código necessário;
- Subir o código produzido, utilizando o comando "git commit -a -m "mensagem"", a mensagem é uma descrição breve daquilo que foi produzido;
- Fazer *checkou*t para a *branch* de desenvolvimento, "git checkout dev/realese-1.0.0" onde o número da *release* alterava à medida que iam sendo criados artefactos (que se ia subindo o código ao ambiente de qualidade);
- Atualizar a *branch* (com o uso do comando "git pull");

- Fazer *merge* da *branch* com a *branch* de desenvolvimento mais atualizada, com recurso ao comando “git merge feature/numero-descricao”;
- Fazer “git push” para enviar o código para a *branch* de trabalho no repositório (feature/numero-descricao).

Após estes passos, era necessário criar um *Pull Request* (PR) com todas as informações, desde descrição, inserir o *link* da *user storie* e esperar que o mesmo fosse aprovado pela restante equipa de desenvolvimento.

Posteriormente, após ser validado o código, concluir uma área e testá-la, era criado um PR desta mesma área para o ambiente de qualidade, após analisar e verificar a inexistência de *bugs*, era criado um PR para o ambiente de produção.

Após uma passagem para qualidade e/ou produção os números das *releases* subiam e os programadores tinham de entregar o novo código nessa mesma *release*.

Para auxiliar a implementação e visto que não existiam todos os *micro-serviços* implementados antes de dar início ao projeto, utilizou-se serviços *mocks* (simulação de serviços reais) implementados no Front-end de maneira que fosse possível testar todos os casos e de maneira que posteriormente fosse só necessário a integração dos serviços.

5.2 Análise das *User Stories*

Tal como foi especificado no capítulo dois, após serem fornecidas as *user stories* aos programadores, os mesmos fazem uma análise pormenorizada das mesmas, sendo que identificam quais os serviços que necessitam de ser invocados, bem como, todas as questões de negócio que precisam de ser esclarecidas.

Após serem analisadas as *user stories* e esclarecidas as dúvidas de negócio, os programadores Front-end identificam quais são os campos de cada serviço para as *user stories*, enquanto os programadores Back-end verificam os serviços fornecidos pelo banco e os campos associados.

Ainda, após a análise das mesmas os programadores Front-end analisam o fluxo e identificam possíveis melhorias para propor ao arquiteto e, caso seja aceite por este, é proposta pelo próprio programador aos analistas do negócio.

Posteriormente, os programadores Back-end reúnem com os programadores Front-end para negociarem as API's e formalizarem os esqueletos dos serviços (os campos que o Front-end necessita e a tipificação dos mesmos).

De referir que toda a implementação descrita nas seguintes áreas foi efetuada por mim.

5.3 Area 1 – Landing Area

5.3.1 Página Inicial

Posto isto, começou-se a desenvolver os componentes da página inicial. Nesta página dependendo do tipo de *login* feito previamente, era apresentado uma gama de componentes com diferentes funcionalidades (ver figura sete do quarto capítulo).

O primeiro componente a ser desenvolvido foi um cartão de boas-vindas (figura treze). Neste contém uma frase emblemática referentes à receção do utilizador no *site*, contém um botão que faz a navegação para uma modal com um vídeo a demonstrar a usabilidade do mesmo, contém umas imagens apelativas com um texto associado que demonstram as funcionalidades inovadoras da aplicação, contém uma cruz que fecha o componente e contém um *link* para uma página externa.

Figura 13 - Representação do cartão de boas-vindas.

Uma das funcionalidades deste componente é que o mesmo só aparece nas primeiras três vezes que o utilizador entra na aplicação. O utilizador pode fechar o *banner*, fechando este, nunca mais será exibido.

Outra exclusividade do *banner* é que todo o texto e imagens são distintas tendo em conta o tipo de *login*. Se for um consultor, aparecem umas certas imagens com o respetivo texto, se for um empresário, aparecem outras imagens com as funcionalidades inovadoras para este.

Aliado a este componente e de forma a concluir o mesmo, foi feita a modal para onde o utilizador caminha caso queira reproduzir o vídeo.

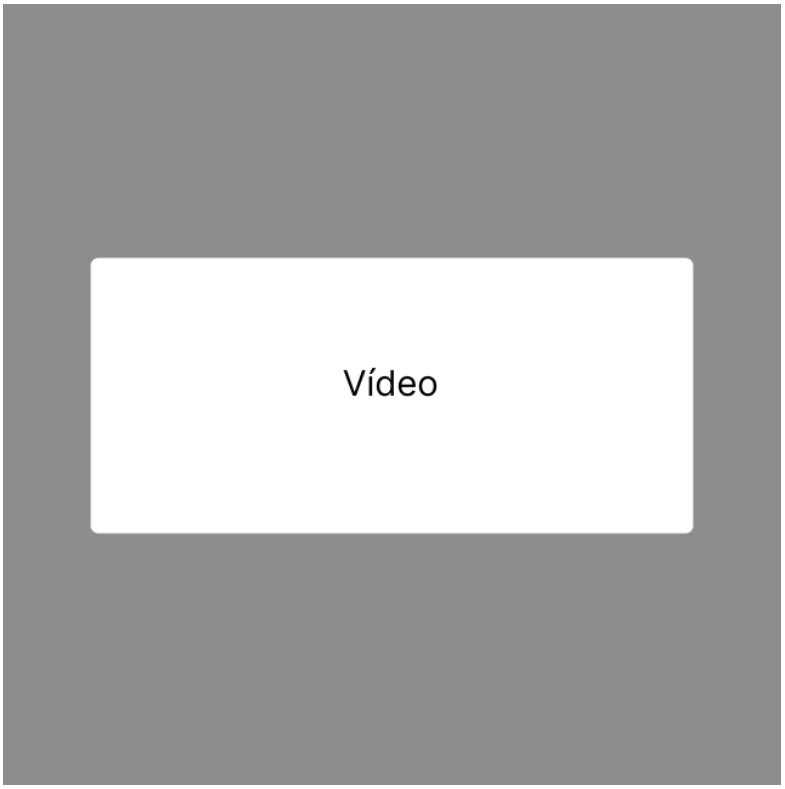

Figura 14 - Ilustração da modal do vídeo.

De realçar que foi pedido para que o vídeo reproduzisse automaticamente e fechar a modal após o utilizador acabar de ver o vídeo.

Estas funcionalidade de reproduzir o vídeo automaticamente e fechar a modal após o utilizado acabar de ver o vídeo foram propostas por mim aos analistas do banco como ponto de melhoria.

Depois destes analisarem, acharam boa ideia e procedi à implementação destas funcionalidades.

Na estrutura do código, foi implementado de maneira que fosse só necessário dar um nome de referência (id da modal em questão) às modais. Para isso, foi criado um *helper* com todos os ids das modais, sendo que quando chamasse o *helper* com esse id, a modal era exibida ou fechada (caso já tivesse sido aberta anteriormente).

Outra das funcionalidades da modal era que mal se clicasse fora do vídeo, a mesma ia fechar ficando na página que antecedera ao clique de exibir a modal.

Numa fase inicial foi desenvolvido com um *link* de um vídeo Youtube para teste visto que o *link* para o vídeo correto foi fornecido após a chamada aos serviços, serviços estes que como tinha referido, ainda estavam a acabar de ser desenvolvidos.

Posteriormente e após a utilização de serviços, o *link* do vídeo era o configurado na base de dados do banco.

De seguida, foi necessário implementar um *banner* notificativo com as novidades de atividades registadas por outros utilizadores da mesma empresa.

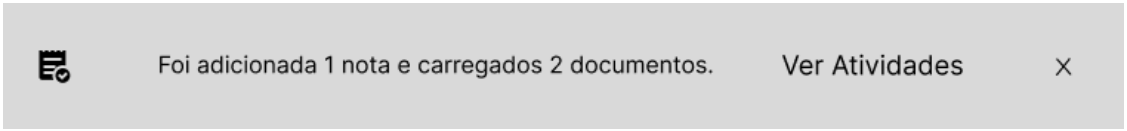

Figura 15 - Representação do *banner* de atividades.

Neste componente são apresentadas todas as alterações feitas pelos outros utilizadores da mesma empresa, é possível a navegação para a área de investimentos para ver essas mesmas atividades. O utilizador pode também fechar o *banner*.

Este *banner* tem características especificas como, só aparecer após o primeiro *login* de outra conta com acesso à mesma empresa e caso haja atividades novas provocadas pela atualização por parte de outros membros da mesma empresa.

Ainda nesta área consta de um modal para convidar consultores a registarem projetos e de uma outra modal de atribuição de acessos.

Tendo este modal concluído foi necessário produzir o *banner* de convite de empresa para associar projeto. Este *banner* só aparece após uma empresa pedir ao consultor para o mesmo fazer essa associação, feito através da modal da figura dezasseis.

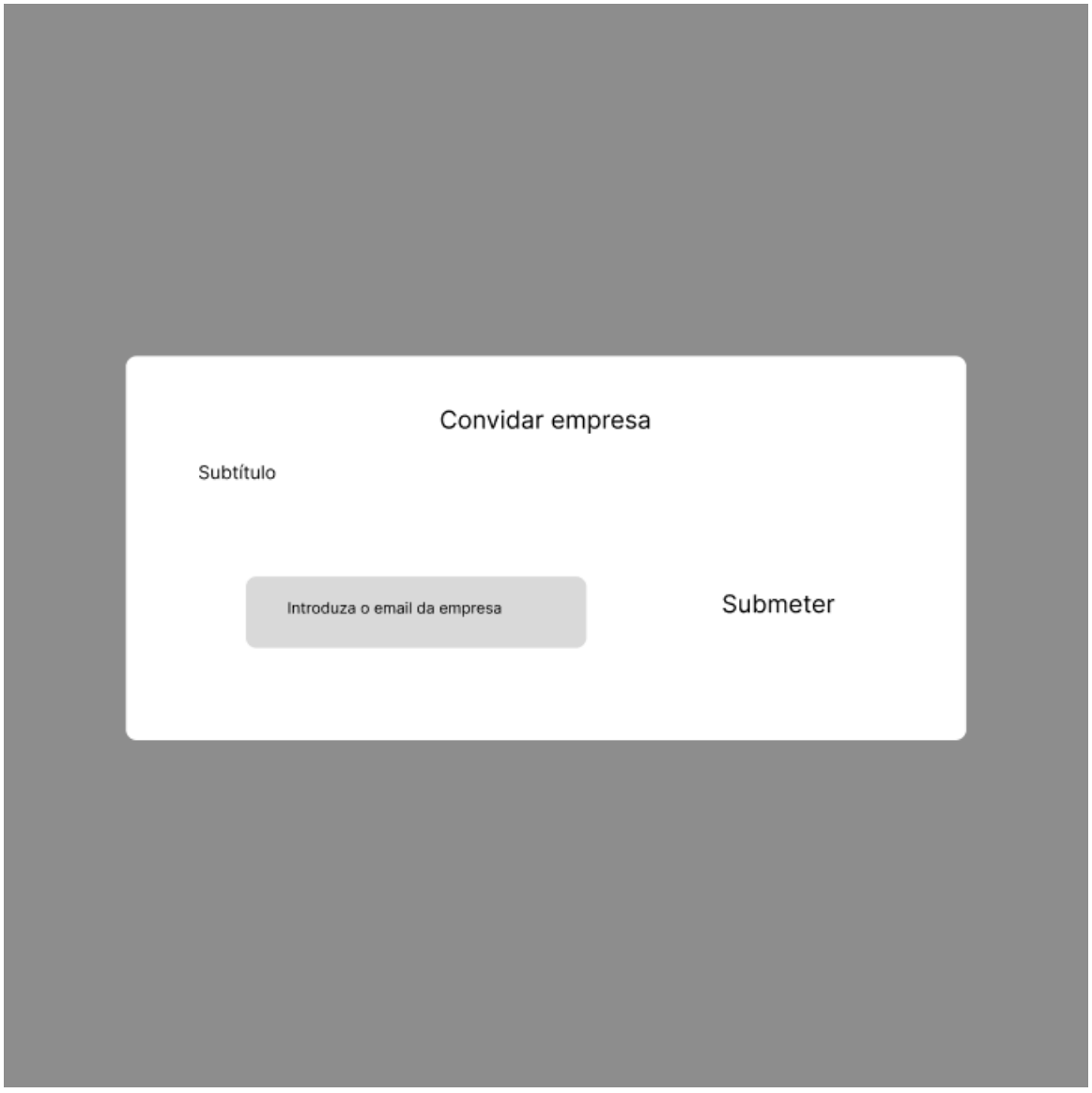

Figura 16 - Modal de convite à empresa.

Figura 17 - Modal de convite do consultor para o projeto.

Após a inserção de um email, é feita uma validação a constatar se o email é válido e, caso não seja, aparece a mensagem correspondente e não deixa submeter, isto é válido para ambas as modais.

Caso seja válido, ao clicar no submeter, é verificado se o email já existe na base de dados do banco, se existir ele não envia o email.

Para a modal de convidar empresa (ver figura dezasseis), caso não exista é enviado para a empresa um convite do banco com as vantagens da aplicação de forma a convencer a empresa a registar-se.

Para a modal de convidar consultor (ver figura dezassete), caso não exista o consultor é enviado para o consultor um convite do banco para o mesmo registar-se.

5.3.2 Configurações

Na segunda página desta primeira área, temos acesso à página de configurações. Nesta, conseguimos definir a preferência de notificações que queremos receber, bem como, o contacto (email) onde queremos receber essas mesmas notificações.

Ao clicar no menu *dropdown* aqui das configurações (três pontos na vertical), contém navegação para poder abrir a modal do vídeo, ver as políticas internas do banco e contactar a equipa de suporte caso tenha alguma dúvida (figura dezoito).

Passando ao componente do *footer*, o mesmo dependo da página, sendo que para esta página de configuração o mesmo contém um botão para guardar alterações e um botão para voltar para a página inicial.

No componente de preferências de notificação consta uns *radio buttons* cujo utilizador seleciona ou desseleciona dependendo das notificações que prefere receber na conta de *email*. Já no componente de contacto, contém um pequeno formulário com um *input*, no qual deve ser introduzido um *email* válido para o qual transitam as notificações previamente selecionadas.

Figura 18 – Ilustração do ecrã de configurações.

Para o caso em que o utilizador tente entrar à força numa das áreas do projeto, foi ainda desenvolvida nesta área (*Landing Area*), uma modal que o impossibilita de aceder a essa mesma área, barrando o mesmo na página inicial com a modal a surgir em primeiro plano (figura dezanove).

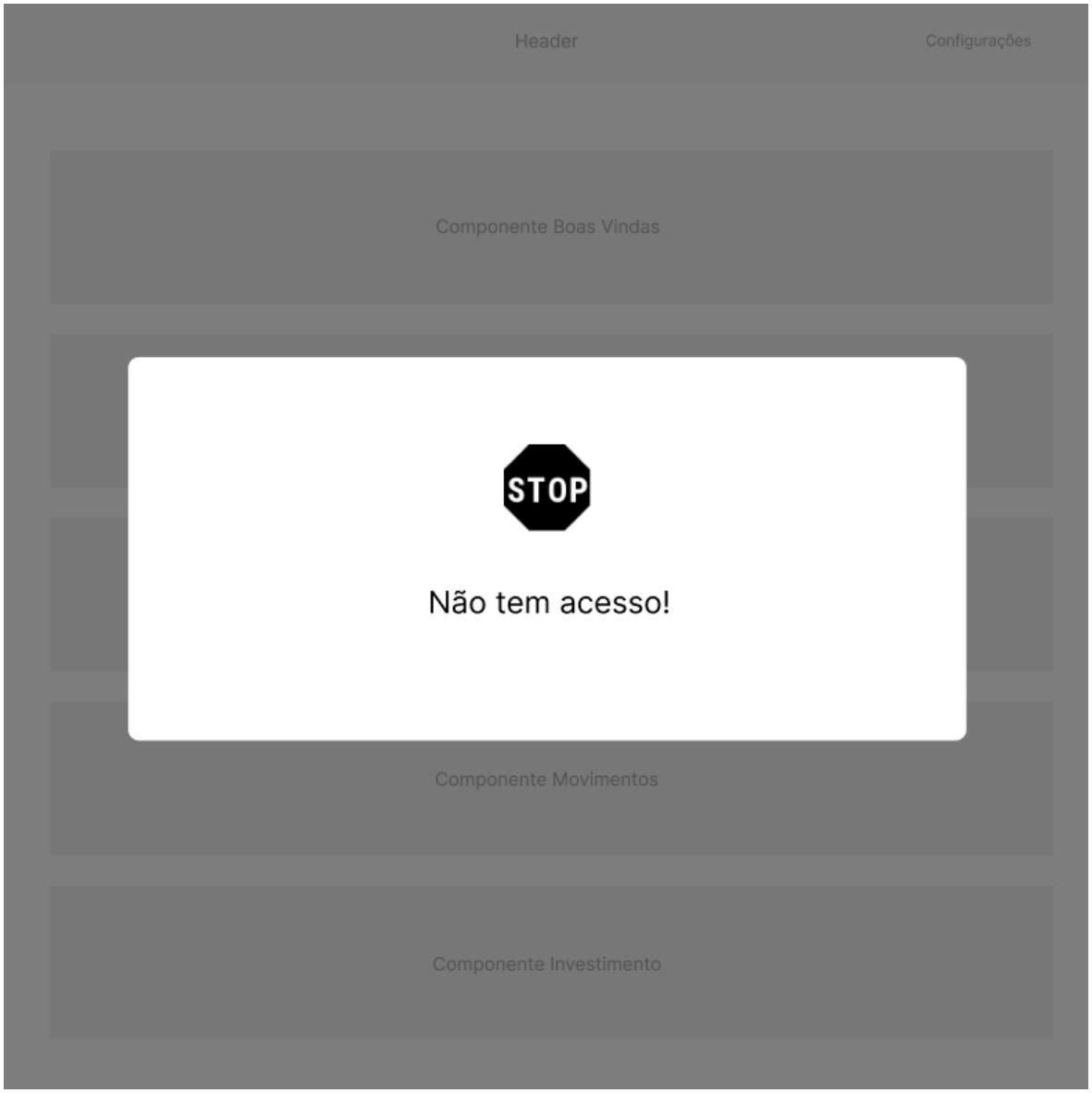

Figura 19 - Modal que restringe o acesso ao consultor.

5.4 Área 2 - Consultores

5.4.1 Adicionar consultor

Figura 20 – Página de inserção de um novo consultor.

Na página de registar consultor, é devolvido através de um serviço as bandeiras existentes bem como o país relativo a essas bandeiras, de forma a acrescentar atrás do número de contacto o prefixo do país.

Todos os campos deste formulário de registo são validados, como por exemplo, o número fiscal só pode ser constituído por nove dígitos, se o utilizador tentar introduzir um número fiscal com mais ou menos de nove dígitos será impedido de registar o consultor.

Para o caso do número fiscal, após o utilizador introduzir todos os campos válidos, o serviço compara o número fiscal com os existentes e caso o mesmo exista retorna um alerta de erro a indicar que o consultor já se encontra registado no site.

Tal como as modais, para os alertas, foi criado um *helper* para facilitar a invocação do alerta em questão, para tal, basta passar o "id do alerta" e o mesmo aparecerá. O desaparecimento deste ocorre após dez segundos.

Caso não se encontre registado, foi desenvolvido uma modal com uma mensagem de que o pedido será analisado pelos técnicos internos do banco e que receberá resposta no prazo de setenta e duas horas após o registo.

5.4.2 Associação de um consultor

Depois de ser introduzidos e validados os consultores, a empresa pode consultar os consultores existentes, bem como, associá-los à sua rede.

Para isso existe na página de consultores disponíveis no site, detém uma pesquisa pelos mesmos e após a empresa achar o consultor, é só atribuir-lhe o devido acesso. Para adicioná-lo a rede necessita de, numa primeira fase, aceitar o documento interno do banco.

Posteriormente à seleção, é redirecionado para a página onde é confirmado a introdução do consultor na rede da empresa.

Nesta página a empresa pode-lhe conceder ou restringir o acesso aos movimentos e investimentos da empresa (ver figura vinte e um).

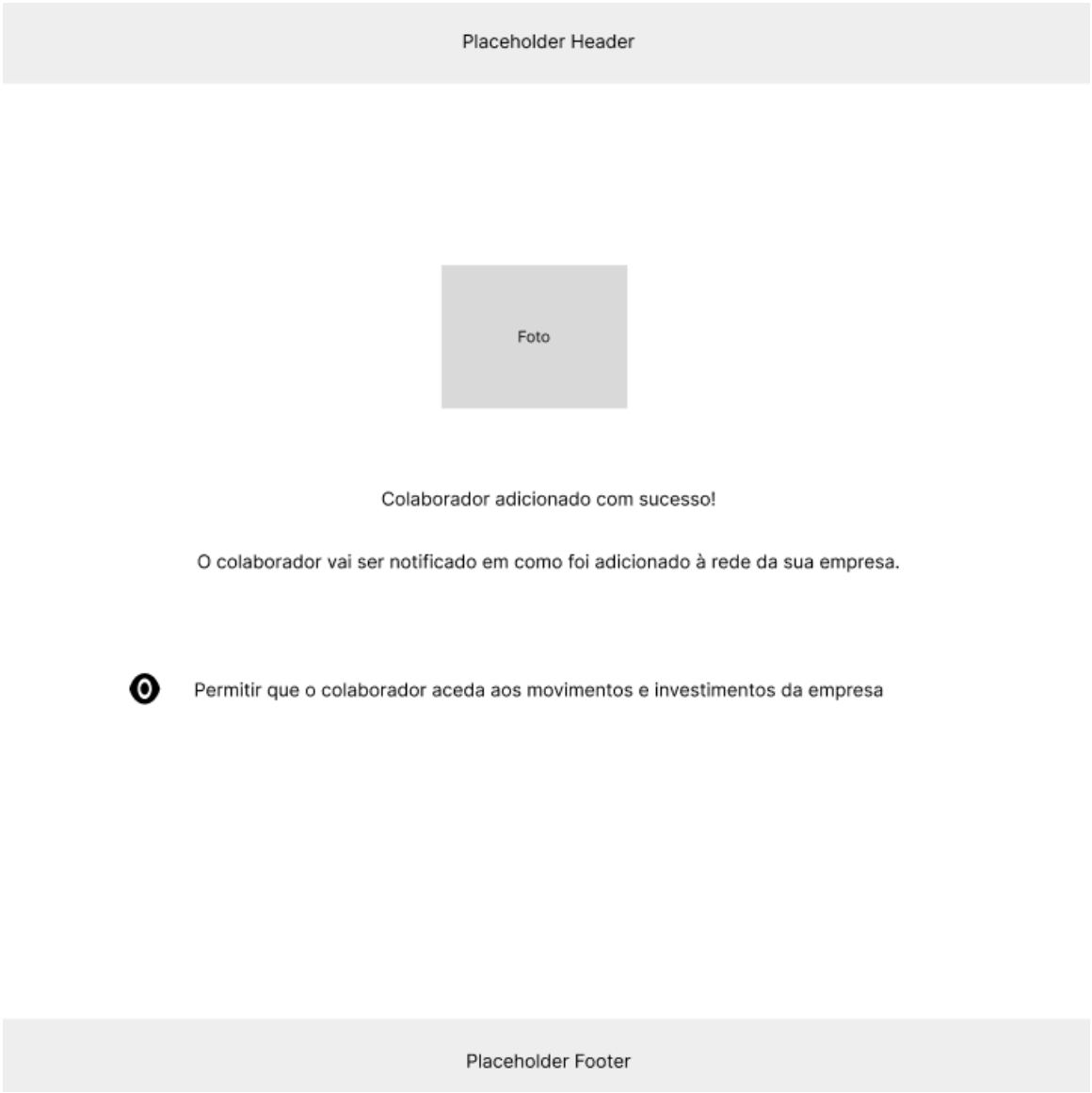

Figura 21 - Página de conclusão da inserção de um novo consultor.

Depois de conceder ou restringir o acesso, o *icon* e o texto mudam e ao passar o rato por cima do *icon* é exibida, em forma de *tooltip*, uma mensagem a indicar ao utilizador que o clique modifica o acesso do consultor aos movimentos e investimentos da empresa.

Após o desenvolvimento desta área foi necessário começar a apontar as chamadas dos serviços *mock* para os serviços implementados, de seguida, certificou-se cada área desenvolvida com o uso dos mesmos.

Dado por concluído e certificadas as duas primeiras áreas do projeto em desenvolvimento, enquanto se avançou para as áreas seguintes, foi necessário subir a aplicação para o ambiente de testes.

Como a aplicação necessitava de ser integrada no site do banco (os *micro-frontends*), esta teve como consequência a criação de vários *bugs* de *layout*, provenientes dos estilos definidos no site onde a aplicação ia ser integrada. Para isso foi necessário rever todas as áreas desenvolvidas e certificá-las no site de forma que pudéssemos passar novamente para os *testers*.

Como consequência, algumas das implementações careceram de modificações e/ou melhorias e foi necessário maior esforço para subir ao ambiente de qualidade.

Avançando esta grande barreira, foi-nos cedido os micro-serviços para implementar a próxima área enquanto os *backends* continuavam a implementação dos serviços da área quatro (classificação de movimentos).

5.5 Área 3 – Gestão de Projetos

Na terceira área é onde ocorre a gestão de projetos, para isso, é necessário primeiro recorrer à criação dos mesmos.

5.5.1 Registo de projetos

Na área de registo de projetos tanto a empresa como o consultor podem fazer esta mesma criação, sendo apenas necessário preencher um formulário com o número do projeto, o nome do projeto e os dados do projeto.

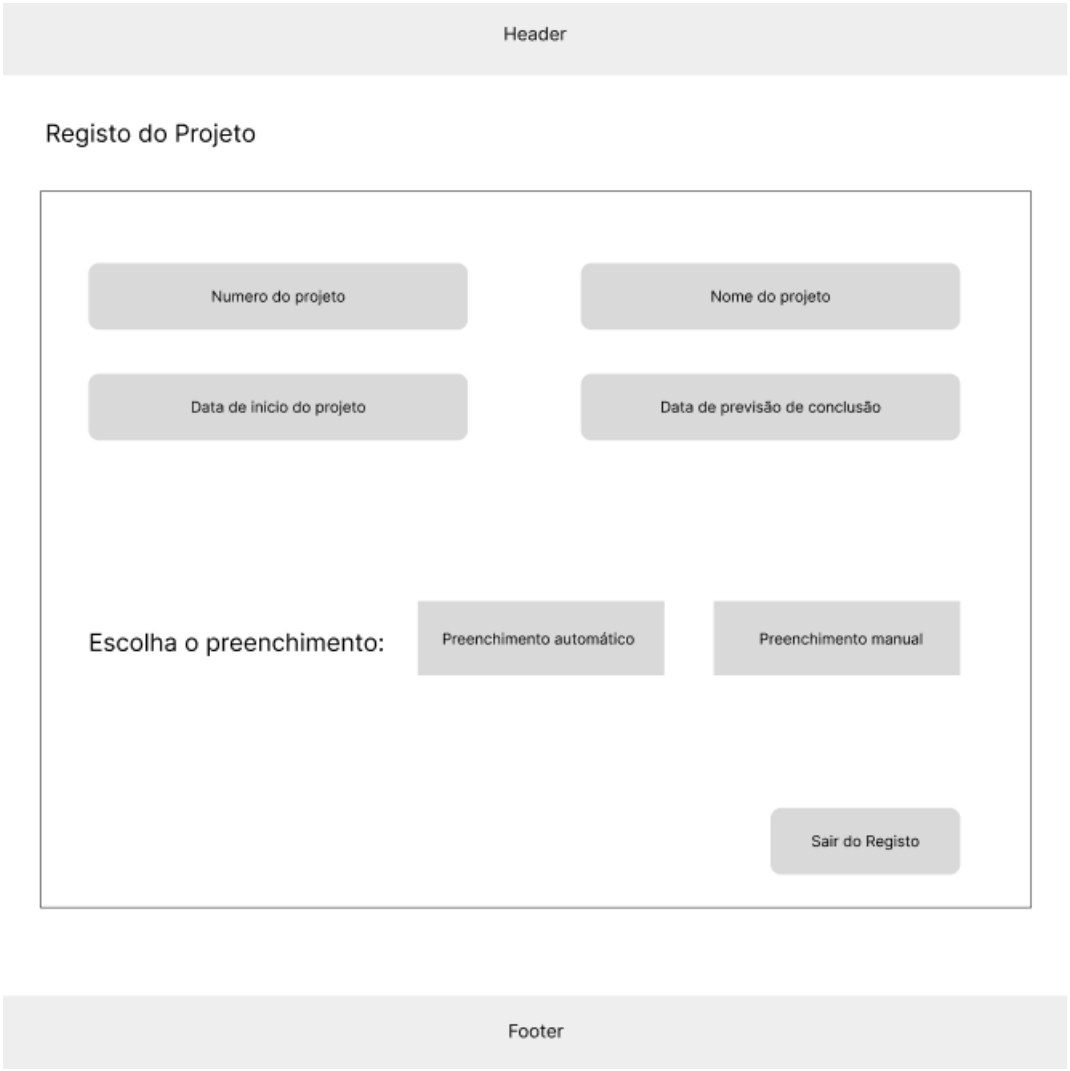

Figura 22 - Ecrã de registo de projeto.

Nesta página o utilizador pode escolher o preenchimento, sendo que será redirecionado para uma nova página para preencher o resto dos dados de forma manual ou automática.

Se a opção escolhida for preenchimento automático, ele é redirecionado para a página de registo de projeto – preenchimento automático, onde tem de introduzir um ficheiro e completa automaticamente os dados de investimento global, investimento inicial, elegível e o incentivo comunitário, tal como é possível verificar na figura vinte e dois.

Nesta parte de introdução do ficheiro deu origem a vários problemas a nível de implementação, desde o desenvolvimento do componente, visto que o mesmo podia ser por seleção e/ou por arrastamento do ficheiro e a implementação dos serviços.

Para este caso em concreto foram necessários dois serviços, um que executava o upload do ficheiro e que outro que devolvia, em tempo real, do ficheiro os dados para serem apresentados na tabela da figura vinte e três.

O componente tinha de estar preparado também para aceitar única e exclusivamente ficheiros com um tamanho máximo de MB indicado pelo negócio e o formato estava limitado a Excel (extensão .xls) ou Livro do Excel (extensão .xlsx).

O utilizador podia introduzir mais que um ficheiro, sendo que devia estar limitado ao tamanho máximo em MB imposto pelo negócio.

Para verificar a extensão do ficheiro que estava a ser arrastado ou selecionado, fez-se uma função que conseguia obter a extensão do ficheiro, função essa que foi revista várias vezes devido ao nível de complexidade da mesma.

Já para guardar o limite máximo dos ficheiros, foi utilizada a *store* do Redux com recurso ao Redux-Saga. Primeiro, fez-se a criação de um estado na *store* do Redux e com recurso as sagas, estas podem ser invocadas interminavelmente, sendo que só seria acordada para despoletar uma ação especifica, neste caso, modificar o valor da quantidade de MB, somando o valor existente ao valor do novo ficheiro.

De realçar que para o caso em que os ficheiros tinham extensões diferentes das pretendidas e, que o tamanho fosse maior do que o estipulado, o utilizador era informado com alertas específicos para cada caso.

Estes alertas necessitaram de um comportamento especial já que este *micro-frontend* seria incorporado na aplicação e, portanto, não havia forma de calcular a janela de visualização do utilizador com o objetivo de que o alerta estivesse presente na janela durante aquele tempo estipulado.

Para conseguir ultrapassar o problema, foi necessário criar um container que agregava todo o código desenvolvido no *micro-frontend* para que este estivesse à escuta da *viewport* em que o mesmo estava inserido para que assim fosse possível lançar o alerta e ser exibido na janela de visualização do utilizador sem que este fosse alguma vez sobreposto.

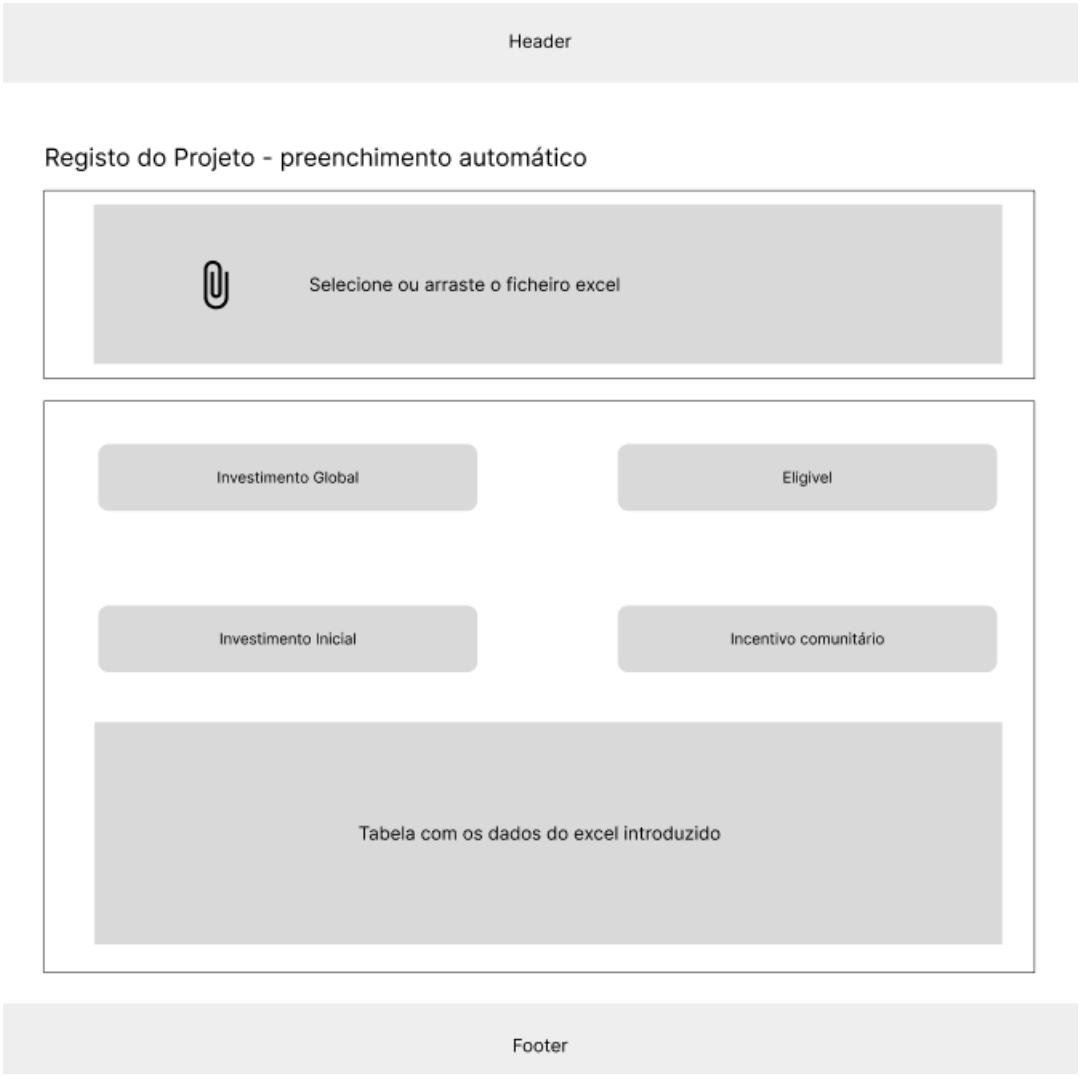

Figura 23 – Página de inserção automática de um projeto.

Caso escolha preenchimento manual, os campos passam a ser preenchidos pelo utilizador, onde existe validação para o caso dos investimentos (os valores dos investimentos são números e o investimento global não pode ser menor do que o investimento inicial), tal como é visível na figura vinte e quatro.

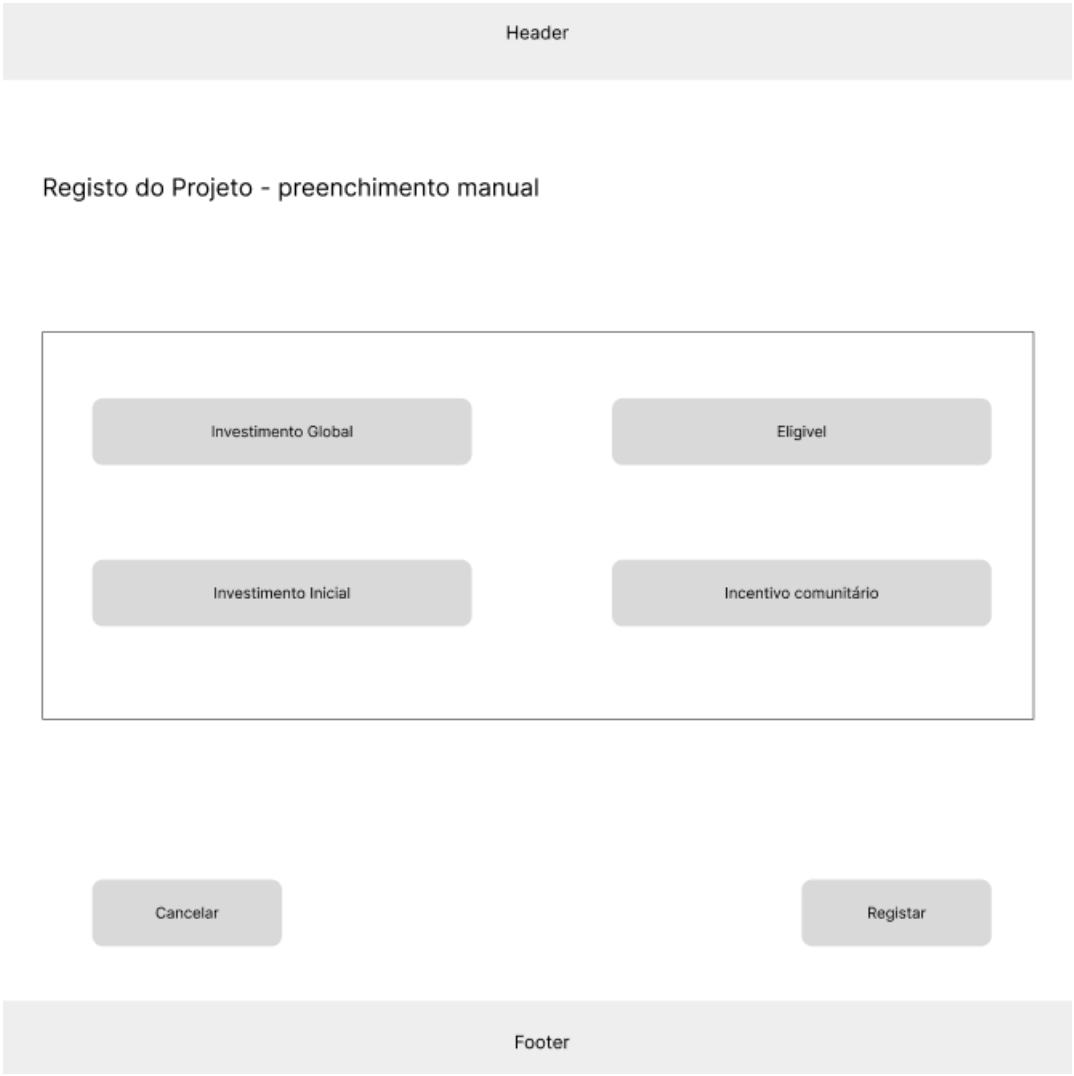

Figura 24 - Página de inserção manual de um projeto.

Após isto, é registado o projeto sendo que pode ser alterado o mesmo a qualquer momento, basta navegar para a página projetos, consultar o projeto e clicar em editar projeto, ainda nesta página, é possível remover projetos.

5.5.2 Registo com sucesso

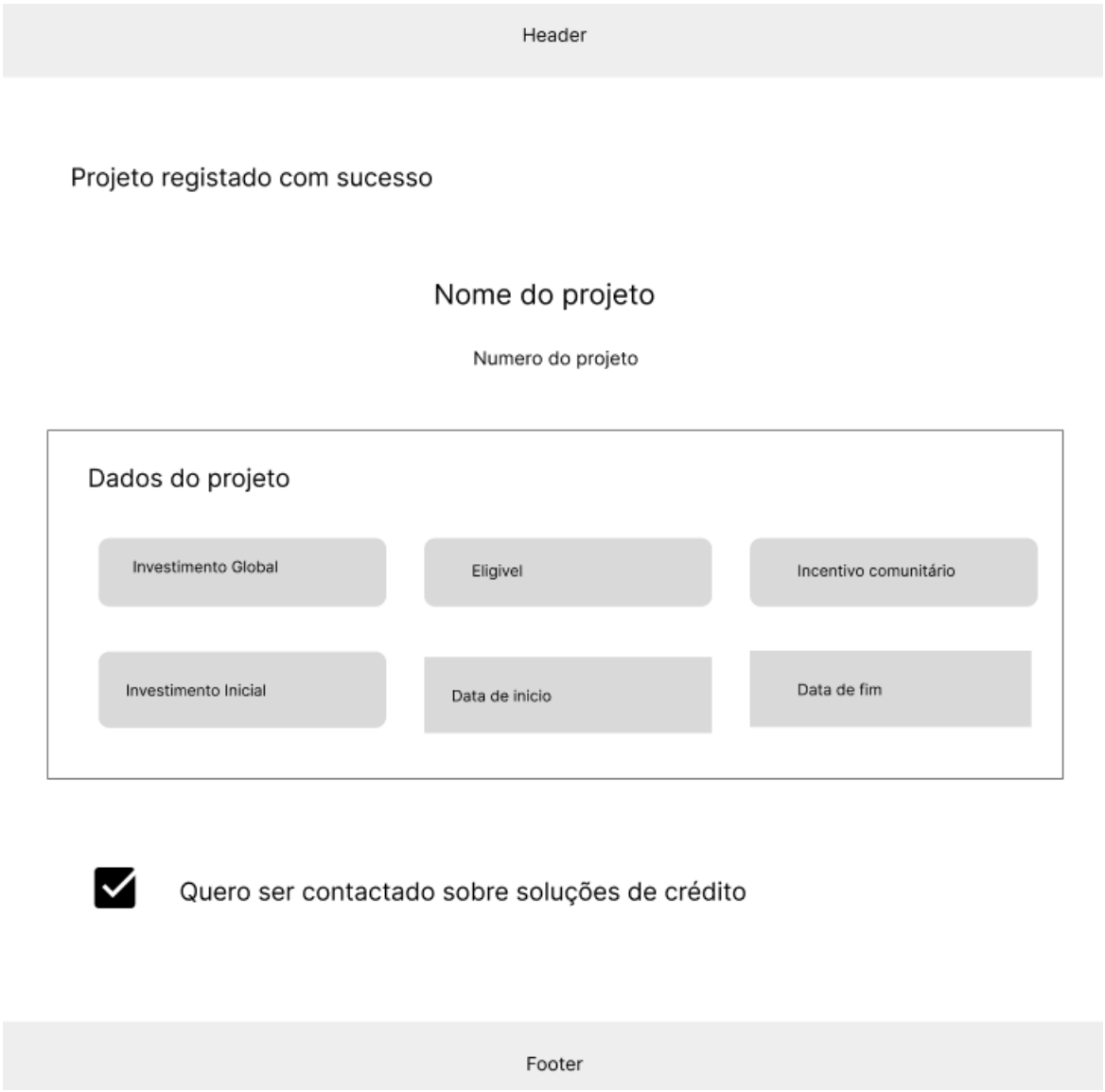

Figura 25 - Página de registo de um projeto com sucesso.

Depois de ser registado o novo projeto ou editado os dados do projeto, o utilizador é remetido para esta página (ver figura vinte e cinco), onde para além dos dados do projeto, o mesmo pode pedir ou não para ser contactado com soluções de crédito.

Aquando do final do desenvolvimento da área 3 (Gestão de projetos), a equipa foi dividida para as duas restantes áreas, área quatro e cinco.

5.6 Área 5 – Comprovantes de investimento

Nesta área, igualmente ao que é exibido na área 4, caso não tenha projetos e não tenha movimentos, aparece um *banner* a indicar ao utilizador que deve registar projetos (ver figura vinte e seis e vinte e sete).

Figura 26 - *Banner* representativo de registo de projetos.

Figura 27 - Ecrã de investimentos sem nenhum projeto.

Caso não tenha projetos registados e tenha movimentos classificados, aparece um *banner* para registar projetos, no entanto, consegue classificar os movimentos existentes, exceto associar a um projeto pois não existe nenhum projeto associado (ver figura vinte e oito).

Figura 28 - Ecrã de classificação de investimentos

Caso tenha projetos registados e não tenha movimentos classificados, aparece um *banner* para classificar movimentos (ver figura vinte e nove e trinta).

Figura 29 - *Banner* ilustrativo de classificação de movimentos.

Figura 30 - Ecrã de investimentos sem movimentos classificados.

Na hipótese de ter projetos registados e movimentos classificados o utilizador pode usufruir da área cinco na totalidade, ao ponto de, para cada movimento, poder associar projeto, anexos e notas (ver figura trinta e um).

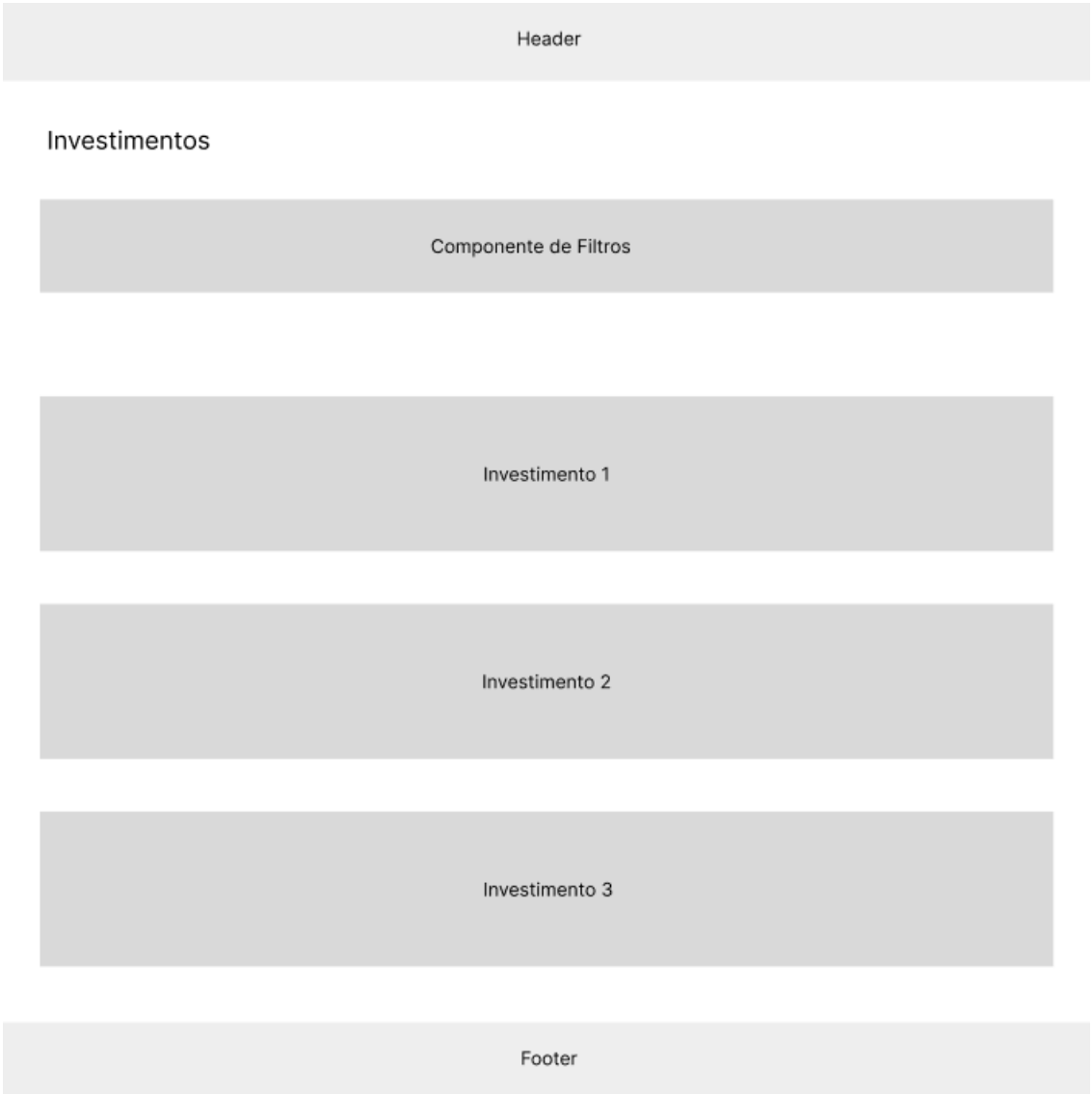

Figura 31 - Página de classificação de investimentos.

Pode ainda selecionar movimentos e fazer o *download* dos mesmos, quer individualmente ou em comprovantes (ver figura trinta e dois).

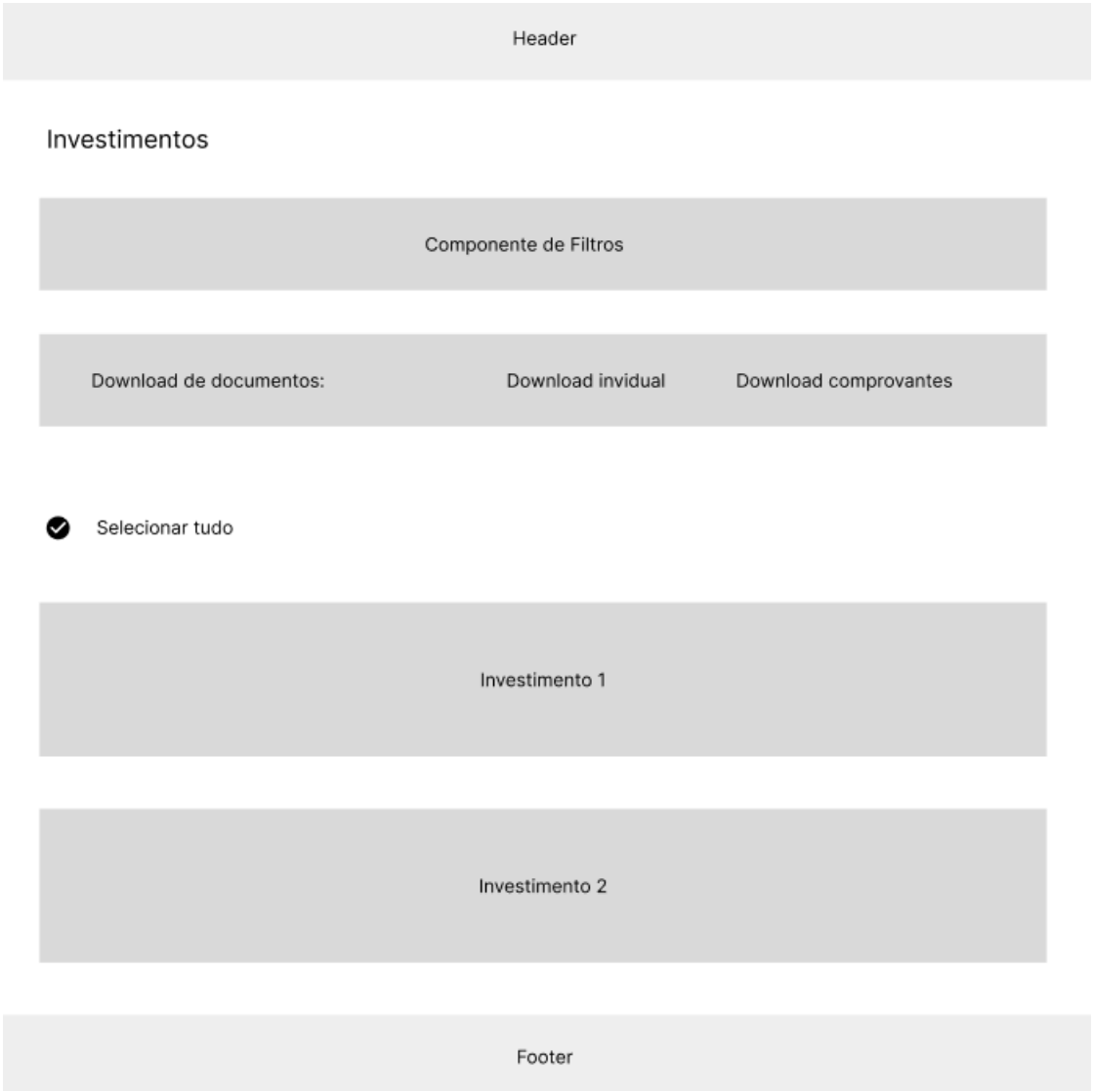

Figura 32 - Ecrã de classificação de investimentos com a possibilidade de download de documentos.

Para poder classificar os comprovantes da melhor forma, existe um conjunto de filtros que o utilizador pode escolher, desde selecionar um projeto existente, selecionar por descrição ou pela conta, filtrar selecionando números de ordem definidos nos dados do projeto (área dois), filtrar por datas e/ou montantes e filtrar por novidades ou ver tudo.

Quer na área quatro quer na área cinco, a lista de cartões de investimento e a lista de elementos na tabela de movimentos usavam um método chamado "*scroll* infinito" (em inglês, *Infinite Scroll*) sobre o qual eram chamados alguns elementos no primeiro serviço e o serviço era paginado.

Para que se conseguisse fazer este "*scroll* infinito" primeiramente, foram guardados todos os elementos provenientes da página zero (página com os primeiros elementos) na *store* e criada uma *div* (elemento HTML) com um tamanho definido (através de recurso ao CSS), em que caso o utilizador chegasse ao final da mesma, era invocado o serviço adicionando mais um número de elementos à página e assim sucessivamente, até que o serviço no último pedido retornava que não existiam mais elementos para apresentar e então não era invocado mais nenhum serviço.

Outra das grandes propostas do cliente para esta página seria conjugar o *download* de todos os comprovantes de investimentos mostrados na página. Para que conseguíssemos concretizar este pedido, foi necessário implementar alguma lógica que dependia da interação do utilizador com a própria lista de comprovantes.

Então, para se desenvolver essa mesma lógica foi necessário que quando o utilizador final recorresse à opção de selecionar tudo, na *store* antes mesmo de correr à invocação do serviço, era produzida uma lista com todos os ids dos documentos e enviado para a invocação do serviço, através do Redux e Redux-Saga.

Nesta página para além dos *downloads* de documentos que eram descarregados após o clique no botão de *download*, quer do individual quer os de comprovantes, dentro de cada cartão de investimento.

Tal como foi dito anteriormente, nos cartões, era possível fazer a gestão dos documentos, desde fazer *upload* a eliminar documentos, ordenar os documentos pela ordem pretendida (sendo que os dois primeiros nunca podiam ser ordenados), criar e/ou remover notas associadas ao documento (estas com recurso a uma *tooltip* de maneira que fosse possível a visualização de todo o conteúdo desta), adicionar e/ou eliminar uma nota ao investimento para uma melhor gestão por parte dos consultores e adicionar e/ou remover um movimento ao investimento.

A parte mais complexa de todas estas funcionalidades do cartão, para além de toda a "renderização" e desempenho da página, foi a ordenação dos documentos, visto que esta para além de ser personalizável, era guardada nos serviços para todas as pessoas associadas à empresa e a consultora tivesse acesso à ordenação.

Para isso foi criada na *store* uma lista de ficheiros da qual provinha com uma ordenação predefinida pelos serviços e, após o utilizador fazer a ordenação de um ficheiro, era guardado nessa lista de *arrays* quer o id do investimento quer a ordenação dos documentos, após isso era

invocado o serviço de alteração de ordenação até que a mesma fosse efetuada de acordo com o que utilizador pretendeu.

Caso ocorresse algum erro, era lançada uma mensagem para o ecrã referente à mesma ordenação, de modo que o utilizador percebesse que a ordenação pedida não foi efetuada com sucesso.

A parte dos documentos não foi das mais simples, também porque necessitou de algumas implementações do lado do *frontend*, visto que os documentos podiam ser carregados por seleção na pasta ou por arrastamento para uma *div* definida. Existia ainda, um máximo de uploads que o utilizador podia executar, um máximo de MB (*Megabytes*) por ficheiro, um máximo de MB também por investimento e, acima de tudo, os documentos só eram aceites se fossem do tipo PDF.

Para executar isto, foi necessário recorrer novamente à *store*, criar variáveis para contar a quantidade de MB já existentes por investimento e mostrar para o utilizador caso tenha excedido o limite por investimento.

Quanto ao limite máximo por documento e à validação de extensão (PDF), eram feitas antes mesmo de ser possível introduzir o documento, visto que essa mesma *div* estava preparada só para receber ficheiros com a extensão PDF (“.pdf”) e com um limite de *Megabytes*.

A área cinco foi de longe a área mais complexa de ser produzida visto que nesta os componentes tinham bastantes “renderizações”, usufruíam bastante de estados locais, comunicavam muito com a *store*, com os serviços e acima de tudo, era inovadora nas funcionalidades.

De referir que em todo o projeto fez-se uso do *TypeScript* para tipificar todas as variáveis utilizadas no mesmo de forma que se qualquer utilizador tentasse ultrapassar a segurança do *website*, tinha de ter acesso a essas mesmas tipografias para descobrir de que tipo era a variável que estava a tentar manipular.

5.7 Tecnologias Aplicadas

Nesta subsecção vou mostrar a aplicação de algumas tecnologias utilizadas durante o projeto, como é o caso das tecnologias de teste (Jest, Enzyme e React Testing Library).

Tal como foi dito no segundo capítulo, os testes realizados são testes unitários e teste aos componentes.

Os **testes unitários** são testes a funções, métodos e/ou procedimentos. Já o **teste aos componentes** tem o objetivo de testar os componentes na sua individualidade. Nos testes aos componentes estes podem ser testes a componentes criados e/ou a componentes de React (estes são feitos com recurso ao React Testing Library).

O cliente exige que os testes de cobertura de toda a aplicação sejam superiores a 70%.

A ponto de exemplificar os testes produzidos durante a implementação da aplicação, produzi o seguinte exemplo, figura trinta e três, para demonstrar exemplos de testes executados.

```
import React, { useState } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { changeValue, deleteValue } from '../../store';
import { StoreState } from '../../store/store';

export const Example: React.FC = () => {
    // read values from store
    const storeValue = useSelector((storeState: StoreState) => storeState.field.value);
    // dispatch actions to store
    const dispatch = useDispatch();
    // internal component state
    const [internalValue, setInternalValue] = useState(storeValue);

    /**
     * Method to save value in store
     */
    const saveChanges = (): void => {
        dispatch(changeValue(internalValue));
    };

    /**
     * Method delete the value in store
     */
    const deleteValueInStore = (): void => {
        dispatch(deleteValue());
    };

    return (
        <>
            <p>Value in redux-store:</p>
            <p className="displayedValue">{storeValue}</p>
            <input onChange={(event): void => setInternalValue(event.target.value)} value={internalValue} />
            <button onClick={deleteValueInStore}>Delete</button>
            <button onClick={saveChanges}>Save</button>
        </>
    );
};
```

Figura 33 - Exemplo de código para a aplicação de testes.

Um exemplo de teste aos valores recebidos na *store* do Redux é o apresentado na figura trinta e três. Nessa figura podemos verificar um teste ao valor inicial, à alteração do valor na

store e à possibilidade de eliminar os dados desse mesmo valor. Este teste é um exemplo do que foi feito durante todos os testes à aplicação.

```
import { changeValue } from '..';
import { deleteValue } from '../action';
import { exampleInitialState, exampleReducer } from '../reducer';
import { ExampleState } from '../types';

describe('exampleReducer', (): void => {
    it('should return the initial state', (): void => {
        expect(exampleReducer(undefined, {} as any)).toEqual(exampleInitialState);
        return;
    });

    it('should change value property', (): void => {
        const changeValueAction = changeValue('123');
        const expectedState: ExampleState = {
            value: '123',
        };
        expect(exampleReducer(exampleInitialState, changeValueAction)).toEqual(expectedState);
        return;
    });

    it('should change value property to empty', (): void => {
        const changeValueAction = deleteValue();
        const expectedState: ExampleState = {
            value: '',
        };
        expect(exampleReducer(exampleInitialState, changeValueAction)).toEqual(expectedState);
        return;
    });
});
```

Figura 34 - Exemplo de testes na *store* do Redux e Redux-Saga.

No entanto, um exemplo de um teste a um ecrã/componente é o executado na seguinte figura, onde podemos verificar que o primeiro teste verifica se o ecrã foi "renderizado" corretamente e o segundo produz uma alteração do valor a ponto de verificar se o mesmo é guardado na *store* do Redux.

```
import React from 'react';
import { mount } from 'enzyme';
import { Example as LandingScreen } from '..';
import { Provider } from 'react-redux';
import configureMockStore from 'redux-mock-store';
import { mockState } from '../../../store/initialMockState';

const buildMockStore = (state: object) => configureMockStore()(state);

const buildTestableComponent = (component: JSX.Element, state = mockState) =>
    mount(<Provider store={buildMockStore(state)}>{component}</Provider>);

describe('<Example Screen />', () => {
    it('should renders screen', (): void => {
        const wrapper = buildTestableComponent(<LandingScreen />);
        expect(wrapper).toBeDefined();
    });

    it("should have field value equal 'Other Value In Store' ", (): void => {
        const alternativeMockState = {
            field: { value: 'Other Value In Store' },
        };
        const wrapper = buildTestableComponent(<LandingScreen />, alternativeMockState);
        const value = wrapper.find('.displayedValue').text();
        expect(value).toEqual('Other Value In Store');
    });
});
```

Figura 35 - Exemplo de testes ao componente e/ou página.

Para este exemplo, depois de correr os testes com recurso ao comando “npm run test -- --coverage --watchAll=false” conseguimos perceber quais foram os testes produzidos e o resultado (ver figura trinta e seis).

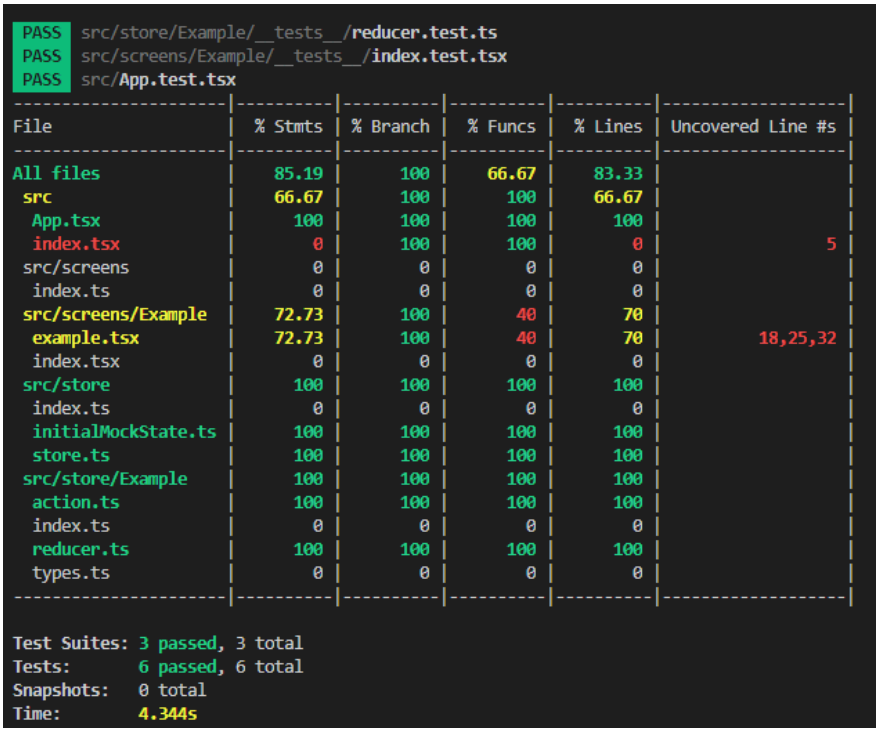

Figura 36 - Resultado de todos os testes executados.

Ainda, no final destes é criado automaticamente uma pasta com o resultado dos testes onde é incluído um ficheiro HTML que contém informações mais pormenorizadas como é o caso da figura trinta e sete, onde consta que o exemplo criado tem uma cobertura de testes total de 66.7%.

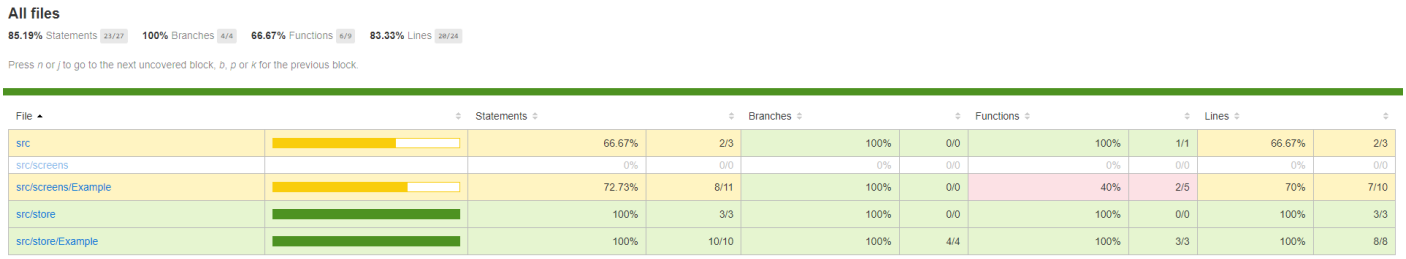

Figura 37 - Página HTML criada após a execução dos testes.

Ainda, nesta página web criada automaticamente, é possível ver os testes feitos, a quantidade de passagem, e o que necessita de testes para cobertura (o que é apresentado a vermelho na figura trinta e oito).

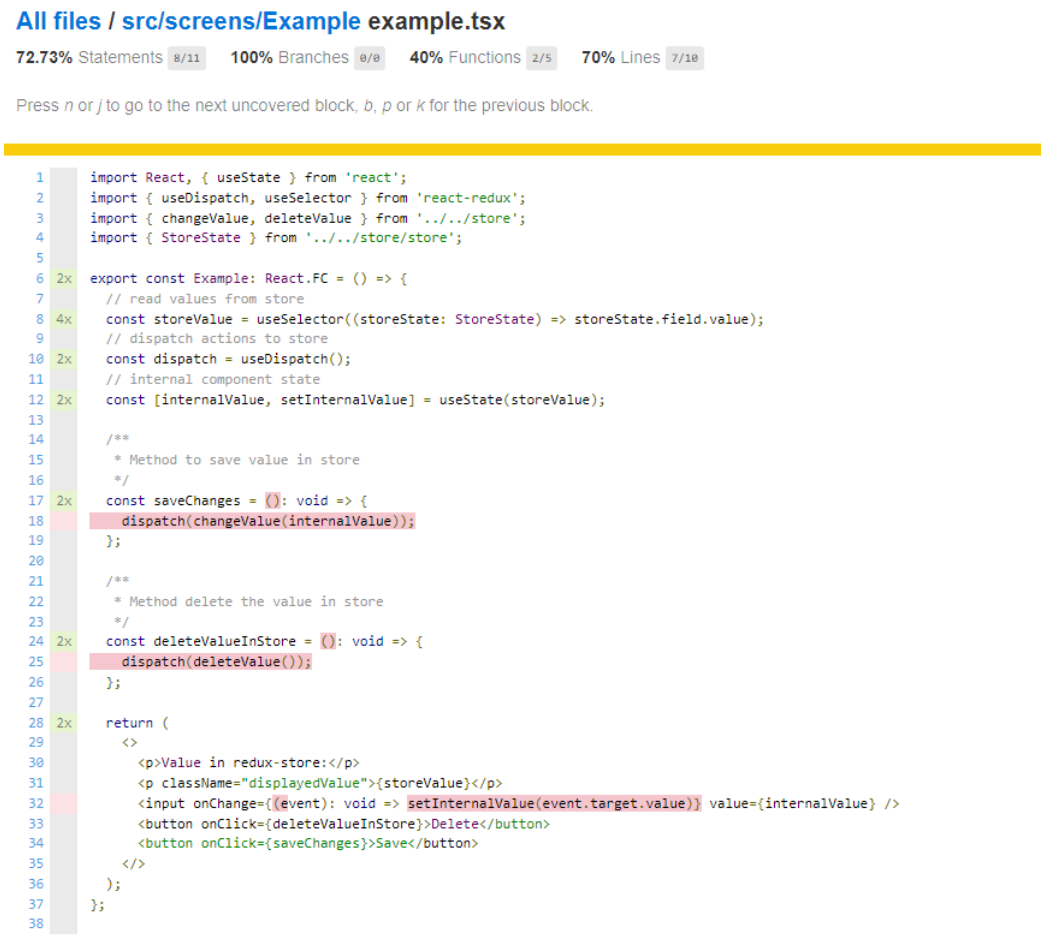

Figura 38 - Exemplo de teste executado, descrição de cobertura e possíveis melhorias.

5.8 Testes/*Bugs*

Depois de efetuados os testes a toda à aplicação pela equipa de React, com recurso ao Jest, Enzyme e React Testing Library tal como foi apresentado no subcapítulo anterior e, posteriormente, por parte dos *testers*, foram descobertos bastantes *bugs* na aplicação. *Bugs* esses que por vezes levaram alguns dias de desenvolvimento a corrigir devido à sua complexidade.

Para o desenvolvimento de cada *bug* era necessário criar uma *branch*, tal como acontecia para o desenvolvimento das *features* (funcionalidades). Porém, a nomeação da *branch* passou de "feature/numero-nome", em que o número era o número da *user story* e o nome a descrição da mesma, a "fix/numero-nome", em que o número seria o número da *user story* onde o *bug*

estava a ocorrer e o nome seria o nome representativo do que seria necessário produzir para colmatar esse mesmo *bug*.

Alguns dos *bugs* encontrados na aplicação passavam por:

- A pesquisa por projeto no *card* de investimento, muitas das vezes não estava a fazer o comportamento pedido, porque necessitava de, ao se pesquisar, aparecer uma descrição do projeto por de baixo do mesmo, através de um *Accordion*;
    - Foi necessário mais uma vez, guardar o estado na *store*, sendo que cada vez que se eliminava a pesquisa, este estado voltava à forma inicial.
- Na área um ao entrar como um consultor que não tem acesso aos movimentos e investimentos de uma empresa, esse acesso não era bloqueado;
    - Para barrar este acesso foi necessário criar nos micro-serviços um campo em que vinha preenchido caso o consultor não tivesse acesso à área em questão (área quatro e cinco).
- O excesso de "renderizações" estava a comprometer o comportamento de alguns dos elementos da página na área cinco;
    - Foi necessário remover grande parte da complexidade do código (r*efactoring* aos componentes).
- Não existia uma validação na pesquisa por datas na área cinco, sendo que o utilizador podia pesquisar qualquer data e introduzir que a data final podia ser maior que a data inicial (o que em nenhuma circunstância é possível);
    - Foi criada essa mesma validação, exigindo ao utilizador final datas verídicas, visto que a data inicial nunca poderia ser maior que a data final.
- O "selecionar tudo" funcionalmente não estava completamente bem, não estava a incorporar documentos cujo *upload* tinha sido feito anteriormente;
    - Isto estava a acontecer por conseguinte das validações da camada dos micro-serviços e da camada interna do banco, foi adicionado um *icon* rotativo (através de uma *div* com recurso ao CSS, que rotacionava a imagem) com uma *tooltip* no *icon* a indicar que o ficheiro se encontrava em validação. Esta implementação foi uma sugestão minha aos analistas do banco.
- Algumas das preferências de notificações na área um não estavam a ser guardadas;

    - O problema que estava a decorrer era que após ser guardado um novo *email* na base de dados, os serviços de adicionar preferências, não estavam a funcionar como o pretendido.
    - Foi necessário guardar as preferências na *store* de maneira a não se perderem em "renderizações" por atualização dos estados internos dos componentes da página;
- Alguns *bugs* encontrados provinham de *bugs* de serviços, estes por vezes eram mais complexos, pois existiam duas camadas de *backend*, uma de micro-serviços e uma interna do banco;
    - Para a resolução destes *bugs* foram necessárias várias reuniões com as equipas de *backend* e a equipa interna do banco).

Validado pelos *testers*, o projeto foi evoluído para o ambiente de qualidade e apresentado pela equipa de desenvolvimento à equipa do banco. Posteriormente, pela equipa do banco aos engenheiros de negócio internos do banco.

Dessa reunião foi decidido que parte dos estilos dos rótulos (ou *labels* como se costuma chamar na programação) teriam de ser revistas pela equipa de *design* e foram pedidas novas funcionalidades em pontos específicos do projeto.

A pedido do banco, para além das funcionalidades novas, foi necessário reformular todas as transições de páginas. Estas estavam a ser feitas com recurso a "*loaders*" e passaram a ser feitas com recurso a "*skeletons*".

Posto isto, evolui-se o projeto a produção e começou-se os testes intensivos. Foram resolvidos muitos *bugs* e finalmente apresentado aos consultores e empresas da confiança do banco (*Family&Friends*).

Depois destes fazerem testes de utilização normal, foi necessário só a alteração de algumas imagens e conteúdo das *labels*, visto que o cliente pretendia dar mais enfase a todo o trabalho desenvolvido.

Capítulo 6

Conclusões

Com esta aplicação enfrentei o desafio de lidar com um projeto real, com dificuldades do dia-a-dia de programação, com grandes quantidades de dados, com a necessidade de aprendizagem de termos bancários e acima de tudo com a pressão de concluir o projeto na data estipulada.

No entanto, agregado a este projeto existiu uma grande aprendizagem quer no uso das ferramentas e linguagens conhecidas (como por exemplo React, CSS, Javascript e HTML), quer na aprendizagem das novas ferramentas e linguagens, como o caso das *frameworks* internas do banco e na elevação do conhecimento como um todo.

A equipa de trabalho para além de conter elementos experientes na área, muitas das vezes foi necessário um trabalho redobrado para estudar aprofundadamente as novas metodologias e conceitos inovadores para entregar ao cliente o melhor projeto possível sempre tendo em conta os requisitos funcionais e não funcionais estabelecidos.

Um dos pontos que falhou neste projeto foi a escolha do método de trabalho *Waterfall*, mas serviu de aprendizagem para todos e acima de tudo uma melhoria para projetos futuros.

A definição inicial da arquitetura de pastas e a organização do trabalho foi bastante importante e crucial para que se tenha conseguido concretizar um trabalho deste tamanho no tempo estipulado.

Este projeto foi uma prova importante da capacidade de autonomia individual e da equipa de trabalho, a todo o empenho e dedicação, a muitas horas de pesquisa e cooperação como uma verdadeira equipa.

Como em todos os projetos, existiram fases em que a equipa tinha mais liberdade de trabalho, estas fases foram aproveitadas para melhorar o código produzido, e outras fases em que as funcionalidades e resolução dos *bugs* tinham de ser rápidas e eficazes.

Em suma, este projeto foi acima de tudo uma prova para todos os elementos e uma vitória com base em todo o esforço e empenho para concluir esta obra-prima.

Bibliografia

[1] ANACOM, 2020, “INE - Inquérito à utilização de TIC nas famílias em 2020”. Acedido a 15 de abril de 2022, em https://anacom.pt/render.jsp?contentId=1585063.

[2] Allan Kelly, “The staffing pyramid”. Acedido a 15 de abril de 2022, em https://www.allankelly.net/archives/461/the-staffing-pyramid/

[3] Wikipédia, 2006, “Scrum”. Acedido a 15 de Abril de 2022, em https://pt.wikipedia.org/wiki/Scrum

[4] Ahad Waseem, 2022, “Waterfall Methodology: History, Principles, Stages & More”. Acedido a 10 de Junho de 2022, em https://managementhelp.org/waterfall-methodology

[5] Microsoft, 2022, “O que é o Azure DevOps?”. Acedido a 25 de Outubro de 2022, em https://learn.microsoft.com/pt-br/azure/devops/user-guide/what-is-azure-devops?toc=%2Fazure%2Fdevops%2Fget-started%2Ftoc.json&bc=%2Fazure%2Fdevops%2Fget-started%2Fbreadcrumb%2Ftoc.json&view=azure-devops

[6] Sandhya Mungarwadi, 2020, “The Pros and Cons of Jenkins vs Azure DevOps”. Acedido a 10 de Junho de 2022, em https://yml.co/pros-and-cons-of-jenkins-vs-azure-devops

[7] Wikipédia, 2020, Figma. Acedido a 10 de Junho de 2022, em https://pt.wikipedia.org/wiki/Figma

[8] Bayu Ferdian, 2019, “Pros & Cons of Using Figma (Updated)”. Acedido a 10 de Junho de 2022, em https://medium.com/gizalab/pros-cons-of-using-figma-317115f762be

[9] Wikipédia, 2016, “Swagger (software)”. Acedido a 11 de Junho de 2022, em https://en.wikipedia.org/wiki/Swagger_(software)

[10] Swagger. Acedido a 11 de Junho de 2022, em https://swagger.io/

[11] Wikipédia, 2015, “Visual Studio Code”. Acedido a 11 de Junho de 2022, em https://en.wikipedia.org/wiki/Visual_Studio_Code

[12] Visual Studio Code. Acedido a 11 de Junho de 2022, em https://code.visualstudio.com/

[13] Git. Acedido a 11 de Junho de 2022, em https://git-scm.com/

[14] Susana Valente, 2016, “Git: manual de utilização e vantagens e desvantagens”. Acedido a 11 de Junho de 2022, em https://www.growunder.com/pt/blog/dicas/65-git-como-funciona-e-quais-as-vantagens-e-desvantagens

[16] React. Acedido a 12 de Junho de 2022, em https://reactjs.org/

[17] JavaTPoint, “Pros and Cons of ReactJS”. Acedido a 12 de Junho de 2022, em https://www.javatpoint.com/pros-and-cons-of-react

[18] Wikipédia, 2003, “HTML”. Acedido a 12 de Junho de 2022, em https://pt.wikipedia.org/wiki/HTML

[19] Portal Web Designer, “Programação Web – O que é CSS”. Acedido a 12 de Junho de 2022, em https://portalwebdesigner.com/programacao/css/

[20] MDN contributors, 2022, “JavaScript”. Acedido a 15 de Agosto de 2022, em https://developer.mozilla.org/en-US/docs/Web/JavaScript

[21] TypeScript. Acedido a 15 de Agosto de 2022, em https://www.typescriptlang.org/

[22] Redux. Acedido a 15 de Agosto de 2022, em https://redux.js.org/

[23] ACERVO LIMA, “QUAIS SÃO AS VANTAGENS DE USAR REDUX COM REACTJS?”. Acedido a 15 de Agosto de 2022, em https://acervolima.com/quais-sao-as-vantagens-de-usar-redux-com-reactjs/

[24] Redux-Saga. Acedido a 15 de Agosto de 2022, em https://redux-saga.js.org/

[25] Newbedev, “Pros/cons of using redux-saga with ES6 generators vs redux-thunk with ES2017 async/await”. Acedido a 15 de Agosto de 2022, em https://newbedev.com/pros-cons-of-using-redux-saga-with-es6-generators-vs-redux-thunk-with-es2017-async-await

[26] Jest. Acedido a 20 de Agosto de 2022, em https://jestjs.io/

[27] Rahul Panchal, 2021, “Top 3 JavaScript Testing Frameworks with their Pros and Cons”. Acedido a 20 de Agosto de 2022, em https://www.rlogical.com/blog/top-3-javascript-testing-frameworks-with-their-pros-and-cons/

[28] Enzyme. Acedido a 20 de Agosto de 2022, em https://enzymejs.github.io/enzyme/

[29] Thinley Norbu, 2020, “React Testing Library Vs. Enzyme”. Acedido a 20 de Agosto de 2022, em https://articles.wesionary.team/react-testing-library-vs-enzyme-afd29db380ac

[30] Testing Library. Acedido a 20 de Agosto de 2022, em https://testing-library.com/

[31] Himanshu Kanojiya, 2022, “What is the better testing library for React, Enzyme (from Airbnb) VS React testing library (from Kent C. Dodds)?”. Acedido a 20 de Agosto de 2022, em https://dev.to/himanshukanojiya/what-is-the-better-testing-library-for-react-enzyme-from-airbnb-vs-react-testing-library-from-kent-c-dodds-c5e

[32] Wikipédia, 2009, “Node.js”. Acedido a 22 de Agosto de 2022, em https://en.wikipedia.org/wiki/Node.js

[33] Altexsoft, 2019, “The Good and the Bad of Node.js Web App Development”. Acedido a 23 de Agosto de 2022, em https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-node-js-web-app-development/