INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

INTERNET BANKING APPLICATION USING REACT JS

Andreia Filipa Lopes de Sousa - a37525

Trabalho realizado sob a orientação de:

Orientador da Empresa: Fábio Maia

Gestor de Projeto: Bruno Pinto

Professor Supervisor do IPB: Paulo Matos

Mestrado em Informática

2021-2022

INTERNET BANKING APPLICATION USING REACT JS

Relatório da UC de Estágio Profissional

Mestrado em Informática

Escola Superior de Tecnologia e Gestão

Andreia Filipa Lopes de Sousa – a37525

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.

Andreia Filipa Lopes de Sousa - a37525

Dedicatória

Dedico este trabalho à minha mãe, aos meus familiares e aos meus professores de curso.

Também, quero dedicar, aos meus colegas de trabalho da empresa ITSector.

"*O sucesso não é a chave para a felicidade. A felicidade é a chave para o sucesso.* "

(Albert Schweitzer[1])

[1] Um teólogo, organista, filósofo e médico alemão, nascido na Alsácia. Descendente de uma linhagem de importantes políticos locais [1].

Agradecimentos

Agradeço, em primeiro lugar, à minha mãe pelo apoio incondicional que me foi transmitindo, seja em aspetos financeiros ou emocionais, bem como irradiar-me todas as suas forças que acabaram por chegar até mim de modo que eu recarregasse todas as minhas energias para continuar e terminar este meu bonito percurso universitário.

Gostaria de deixar o meu apreço, em especial, ao Supervisor Mestre Fábio Maia, sem ele não seria possível conseguir este presente relatório de estágio profissional, pois com toda a sua serenidade, agilidade, competência profissional, juízos, avaliações e recomendações fez com que eu fosse no caminho certo para conquistar o meu objetivo.

Também, quero deixar um agradecimento com muito carinho a todos os Professores do Instituto Politécnico de Bragança que me acompanharam nesta etapa universitária e estiveram sempre disponíveis e, forneceram recursos fundamentais para que cada um de nós tivesse ferramentas necessárias de ir mais além e assim, percorrer um trajeto com grande sucesso.

Aos meus avós e padrinhos, um muito obrigado por todo o seu apoio nesta minha etapa!

Ao meu namorado que foi incansável no apoio emocional e esteve sempre a apoiar-me em tudo aquilo que eu precisei.

Aos meus colegas de curso, um muito obrigado pela amizade e pelos bons momentos que passamos juntos e, pelas memórias que ficarão para sempre, da terra dos amigos para sempre.

Resumo

O *Internet Banking* é uma plataforma que atua na satisfação das necessidades de todos os utilizadores que tenham por hábito utilizar este tipo de serviço.

De um modo muito geral, todo este serviço bancário nasceu á volta de funcionalidades básicas como o caso de fazer uma transferência, efetuar um depósito, consultar a conta, fazer um levantamento, alterar os dados pessoais, o processamento de salários aos colaboradores por parte da entidade patronal, entre outras atividades.

Contudo, para a realização deste trabalho necessita-se de fazer uma análise das *User Stories* que foram propostas para a implementação da *app web*; de conhecer as tecnologias que vão integrar no *site* como o caso do React JS e do JavaScript.

Ainda assim, para este desenvolvimento também se necessita de conhecer metodologias ágeis, como o caso do *Scrum,* que foi uma das metodologias utilizadas para a organização da equipa.

Assim, no desenvolver deste relatório e projeto pretende-se demonstrar as tecnologias, ferramentas e metodologias que satisfazem toda a implementação deste projeto, sendo possível ainda, ajudar o cliente a ultrapassar barreiras como exemplo de filas de espera e, oferecer uma plataforma online sobretudo ágil, fácil de manusear, segura e, que suprima as necessidades de quem utiliza a mesma.

Palavras-chave: *Internet Banking*, Desenvolvimento Ágil, React JS, Micro-serviços.

Abstract

Internet Banking is a platform that works to satisfy the needs of all users who are used to using this type of service.

In a very general way, this entire banking service was born around basic functionalities such as the case of making a transfer, making a deposit, checking the account, making a withdrawal, changing personal data, processing salaries to employees by the entity employer, among other activities.

However, to carry out this work, it is necessary to analyse the User Stories that were proposed for the implementation of the web app; to know the technologies that will be integrated into the site, such as React JS and JavaScript.

Even so, for this development, it is also necessary to know agile methodologies such as Scrum, which was one of the methodologies used to organize the team.

Thus, in developing this report and project, it is intended to demonstrate the technologies, tools and methodologies that satisfy the entire implementation of this project, being also possible to help the client to overcome barriers such as service queues and offer an online platform above all agile, easy to handle, safe and that meets the needs of those who use it.

Keywords: Internet Banking, Agile Development, React JS, Microservices.

Índice

Lista de Figuras

Índice de Tabelas

Siglas

API	Application Programming Interface.
BE	Backend.
CSS	Cascading Style Sheets.
DOM	Document Object Model.
ESTiG	Escola Superior de Tecnologia e Gestão.
FE	Frontend.
HTML	Hypertext Markup Language.
IPB	Instituto Politécnico de Bragança.
JS	JavaScript.
ott	One Time Token.
PC	Personal Computer.
PHP	Hypertext Preprocessor.
PR	Pull Request.
SVG	Scalable Vector Graphics.
UI/UX	User Interface/User Experience.
UML	Unified Modelling Language.
URL	Uniform Resource Locator.
US	User Stories.
XHTML	eXtensible Hypertext Markup Language.
XML	eXtensible Markup Language.

Capítulo 1

Introdução

Nos dias de hoje, cada vez mais, a tecnologia não deixa de acelerar e, é impactante este seu crescimento seja ao nível dedicado à educação, ou financeiro ou até mesmo à saúde.

De facto, aceder a serviços bancários pelo computador, ou seja, a adesão ao *Internet Banking*, é muito mais para além daquilo que há uns anos se pudesse imaginar. E veio reduzir as idas às agências de milhares de pessoas e, que traduz num serviço mais benéfico e com maior agilidade para todos, não perdendo horas sem fim em filas longas nos bancos.

De certa forma, não nos podemos esquecer que a faixa etária mais idosa não consegue aceder tão facilmente a estes *websites* como uma pessoa que esteja compreendida entre a idade dos 18-65 anos, mas, há que contornar este obstáculo pensando em soluções para estas pessoas e, continuar com a crescente evolução da tecnologia que tem trazido muitas vantagens a toda a sociedade.

1.1 Enquadramento

Nas últimas décadas, as empresas têm enfrentado uma grande tensão competitiva, resultante dos avanços rápidos da tecnologia e da abundância dos conhecimentos e exigências que cada utilizador tem atualmente.

Não deixa dúvidas que de facto, a *Internet*, impulsionou vários setores sejam económicos, industriais ou financeiros, criando meios de serviços e assim, causando uma reviravolta na adaptação que cada empresa tem ou terá para ser ainda a melhor do mercado.

O *Internet Banking* surge sem qualquer dúvida como sendo uma plataforma extremamente inovadora e, para além disso, de grande proximidade de interação entre o banco e o próprio cliente.

Este serviço, tem evoluído progressivamente, e a sua melhoria pode-se observar em serviços e/ou funcionalidades que fornece ao cliente, como por exemplo, o acesso a tarefas que antes necessitavam de várias deslocações às agências bancárias.

Atualmente, essa necessidade já é quase inexistente. Existem outras atividades como transferências que podem ser realizadas sem a necessidade de sair de casa.

Assim, a contextualização do *Internet Banking* com o trabalho que se pretende referenciar neste relatório prevê que se consiga ajudar cada vez mais o lado do utilizador e/ou cliente para que a falta de funcionalidades e a melhoria das mesmas possam suprimir necessidades e contribuir para o sucesso, tanto do cliente, como da própria equipa do serviço bancário.

1.2 Objetivos

Os objetivos que definem o projeto proposto pela empresa ITSector durante a realização deste estágio profissional são os seguintes:

1) Desenvolver as várias funcionalidades que estão já identificadas e disponibilizadas na respetiva plataforma *web*, como por exemplo:
    a) A visualização dos pagamentos por parte do cliente;
    b) A possibilidade de fazer transferências;
    c) O cliente poder fazer simulações de créditos sem ter de dirigir-se a uma agência;
    d) O cliente poder modificar os seus dados da conta;
    e) Consultar o património financeiro;
    f) Entre outras funcionalidades que estão descritas no Capítulo 4 e 5 .
2) Análise de US (*User Stories*);
3) Análise de contratos de API’s com o BE (*Backend*);

4) Subir código aos ambientes, via PR (*Pull Request*);
5) Correções de *bugs*;
6) Integração de serviços internos ou já existentes de outras aplicações.

1.3 Estrutura do documento

O capítulo um apresenta uma breve introdução do trabalho proposto, o enquadramento deste mesmo trabalho, bem como, os principais objetivos a efetivar.

Durante o capítulo dois, abordo o conceito de serviço bancário, bem como, *workflow* da equipa de trabalho, e ainda, retrato algumas das vantagens e desvantagens do *Internet Banking*.

No capítulo três faz-se a apresentação das tecnologias/ferramentas, bem como as vantagens e desvantagens das mesmas.

No capítulo quatro aborda-se a modelação e a análise deste projeto.

No capítulo cinco tem-se toda a abordagem do desenvolvimento/implementação do projeto proposto e ainda descrevo os *bugs* resultantes da implementação.

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

Capítulo 2

Contexto

2.1 Internet Banking

O *Internet Banking*, por definição, é um serviço que praticamente todos os bancos possuem, ou seja, um *website* onde o utilizador consegue ter, aproveito de várias funcionalidades sem depender de uma agência.

Face à tecnologia avançada, muitos clientes nunca foram a uma agência fazendo assim usufruto desta via *online* e, conseguindo realizar diversas tarefas como pagamentos, transferências ou até mesmo fazerem simulações para créditos para compra de um imóvel ou de um automóvel.

A utilização deste serviço e/ou plataforma *online*, de um modo geral, não requer a instalação de qualquer tipo de *software* ou *plugin*, sendo que há a necessidade de que o indivíduo tenha um computador com ligação à internet.

Assim, uma vez mais, a facilidade de acesso, a flexibilidade de horários e a comodidade associada, fazem com que o *Internet Banking* tenha mais adesão.

Verifica-se então, que este serviço *online* veio revolucionar a vida do cliente, que atualmente não precisa de sair do seu lar para efetuar pagamentos, fazer consultas do seu extrato bancário, realizar a consulta do seu cartão e gerir o mesmo, como por exemplo, executar o cancelamento do cartão que esta a ser consultado.

Portanto, segue-se um role de vantagens ao *Internet Banking* que permite qualquer um de nós interagir de forma fácil e ágil sem ter grandes esperas e dificuldades para solucionar problemas que se tenha com este sector. E, para além disso, podem ser tomadas decisões mais cautelosas sem que haja um gestor do banco a pressionar o cliente para determinada decisão que de certa forma pode comprometer o cliente de forma indesejável.

Figura 1 - Exemplos de serviços bancários [43] que disponibilizam um *website* para facilitar o acesso e oferecer comodidade aos seus utilizadores.

2.2 Metodologias Utilizadas

Neste campo, passo a apresentar teoricamente as duas metodologias que foram utilizadas ao longo do projeto e são as seguintes:

- Metodologia Waterfall;
- Metodologia Scrum.

2.2.1 Metodologia Waterfall

O modelo Waterfall [1] é um processo de desenvolvimento sequencial que flui como uma cascata por todas as fases de um projeto (análise, *design*, desenvolvimento e teste) com cada fase terminando completamente antes da próxima.

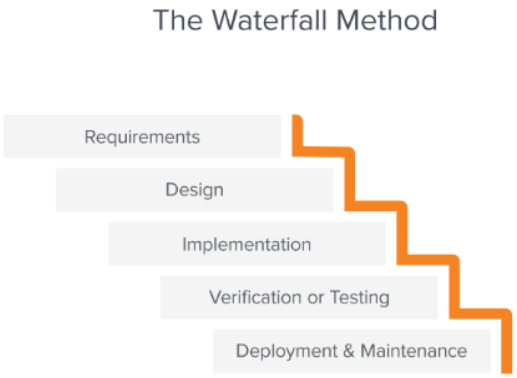

Figura 2 - Modelo "The Waterfall Method" [44].

No entanto, este tipo de metodologia tem vantagens e desvantagens. Abordando as vantagens, o Waterfall faculta estimar com alguma precisão o custo total do projeto. Existe uma fácil atualização sobre o projeto para quem entra depois no mesmo; e, ainda, como

segue uma abordagem bem estruturada, consegue-se rapidamente verificar o progresso das *tasks* que estão já definidos.

Por outro lado, as desvantagens que este modelo tem são:

- Se alguma das fases, seja de requisitos ou implementação, se desalinhar, todas as restantes fases descarrilam;
- Normalmente, neste tipo de metodologia, o cliente não está envolvido em fases de implementação;
- Em comparação com as metodologias ágeis o que se verifica nos projetos é o seguinte: o que está definido acaba por demorar mais face às tarefas que têm de ser entregues. Contudo, este modelo é desvantajoso em comparação com o uso do Ágil ou *Scrum*, são tipos de metodologias ágeis.

2.2.2 Metodologia Scrum

Analisando a figura 3, percebe-se que a metodologia *Scrum* [3] é baseada num conjunto de práticas e princípios bem estabelecidos e, como é óbvio, deve estar presente nos projetos de desenvolvimento de *software*.

Uma outra característica que este modelo apresenta é que a equipa trabalha sobre sprints, isto é, semanas de trabalho devidamente organizadas e planeadas para que a equipa de um modo individual e em grupo, consiga trabalhar dentro do mesmo projeto e tenha uma maior produtividade, eficiência e consistência no projeto em causa, que está a desenvolver para no final ter-se um produto com qualidade.

Como se pode ver, existe ali uma diversidade de roles que o *scrum* tem, desde o *product owner* (que é a pessoa que pode representar as necessidades de muitas partes interessadas no *Product Backlog*), o *scrum master* (é aquela pessoa que está a liderar uma equipa e tem a responsabilidade de manter o *scrum* atualizado), team *members* (membros integrantes da equipa), *users* e ainda *stakeholders*. Porém, esta metodologia traz a quem a utiliza diversos benefícios e analisando um conjunto de projetos existentes no mundo, é a metodologia ágil com maior referência de confiança e utilidade.

Algumas vantagens que esta vem mostrar a quem a manuseia são:

- Flexibilidade em mudanças de requisitos, escalabilidade e qualidade de *software*;
- Compromisso com as expectativas face ao que foi pedido nos requisitos;
- Através de *daily scrum* é muito mais ágil perceber o progresso que a sprint está a levar;
- Auto-organização.

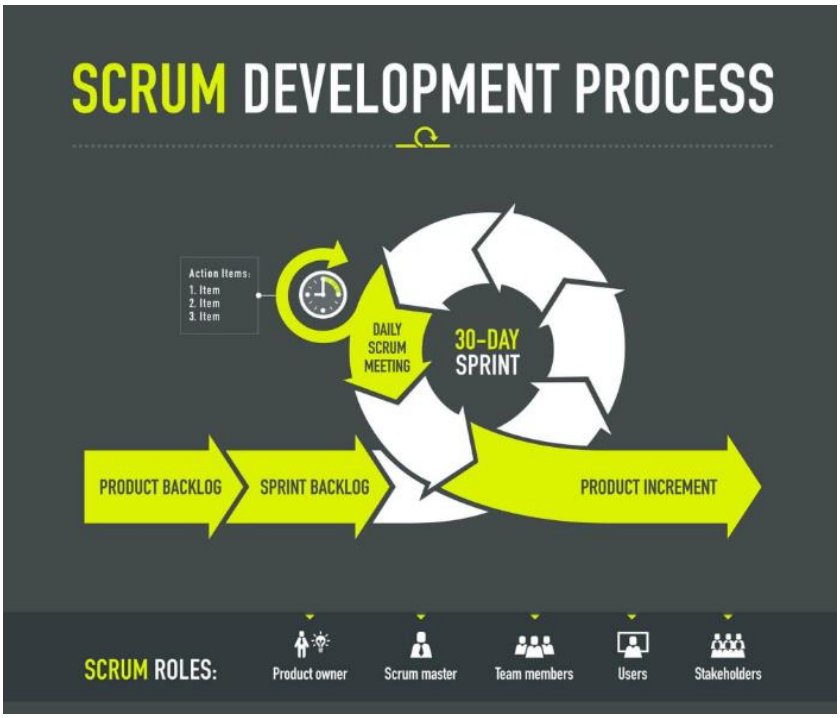

Figura 3 - *Scrum Development Process [3].*

2.3 Processamento da organização de trabalho entre a equipa

O desafio de realizar uma aplicação *web* tem normalmente uma boa carga de trabalho e isso faz com que de facto, para que o trabalho seja agilizado existam equipas de trabalho que tenham interajuda entre todos os membros e, proporcionem entre os mesmos, um bom ambiente de trabalho e sobretudo, a capacidade de organização de tarefas que os espera futuramente.

Posto isto, este projeto foi realizado em distintas metodologias visto que sofreu duas fases, uma inicial para a construção da *app web* e, uma outra de reestrutura ao *site* existente.

No entanto, como primeira fase de trabalho a equipa fez uso das metodologias Waterfall e Ágil – Kanban para o processo de desenvolvimento já explicadas na secção 2.2.

Assim, com o uso desta metodologia a mesmo levou-me a que no meu dia a dia tivesse *meets* de acompanhamento para indicar qual os entraves e, definir datas de término para as US (*User Stories*) ou tarefas que tinha em mãos. Portanto, tive no início a apresentação do modelo do projeto a desenvolver, bem como, os seus requisitos e *design*, e, depois passei para a implementação de algumas funcionalidades e análise das mesmas que teve uma duração de aproximadamente trinta e cinco semanas de trabalho e, incluindo aqui a fase de verificação e testes daquilo que desenvolvi.

Porém, na segunda fase, ficou estabelecido a equipa usar a metodologia *Scrum* de forma a todos trabalhem de forma organizada, com compromisso para o cliente, e sobretudo, com produtividade.

Deste modo, a abordagem que ficou decidida foi dividir a equipa por *squads*, figura 4, de forma a ninguém ficar atropelado pelas tarefas que não lhe competia. De seguida, em cada *squad* estavam as US para determinado elemento da equipa e assim levava a equipa num *board* estável, organizado e sucinto que permitiu a entrega das tarefas que cada um tinha em mãos.

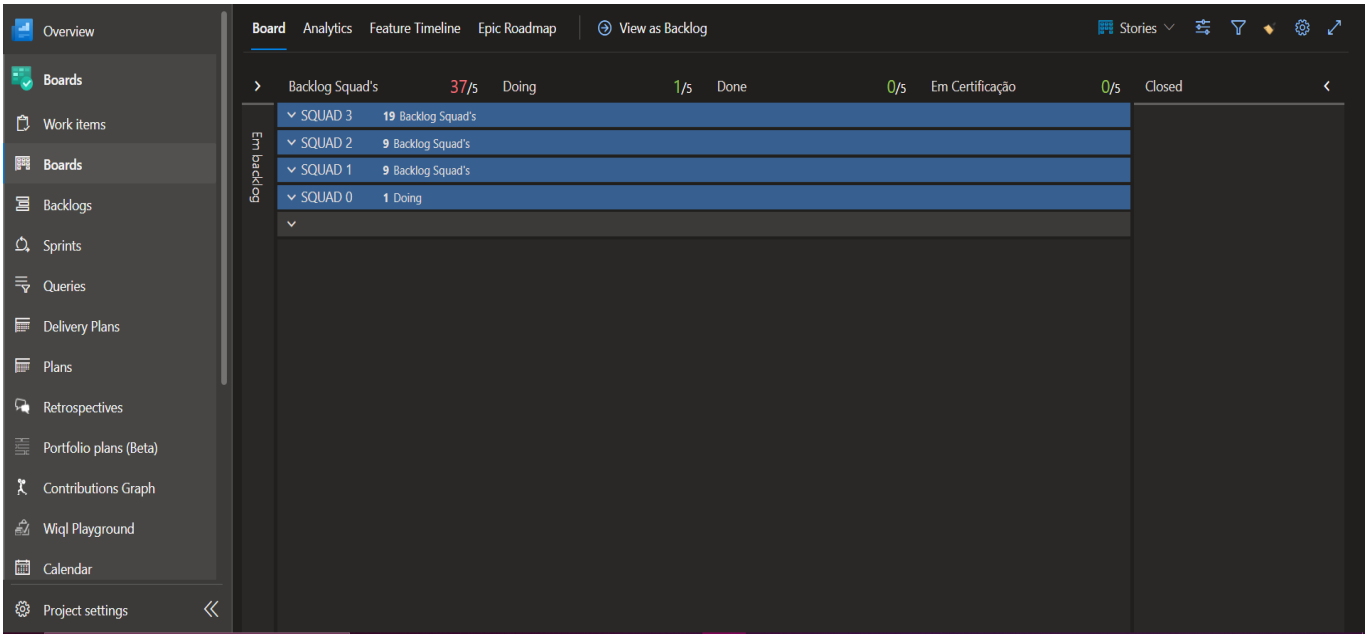

Figura 4 - Ilustra a forma de enquadramento da equipa bem como, a divisão por *squads*.

2.4 Workflow

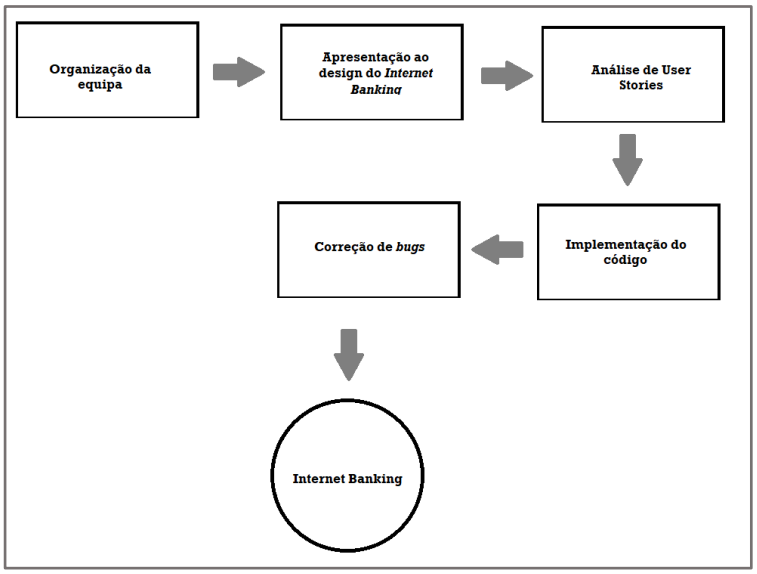

Figura 5 – Representa as etapas para chegar à *app web*.

Atendendo à figura 5, esta mostra-nos as diferentes etapas que tive de ter para fazer parte da concretização do *site*.

Como já indiquei anteriormente e contextualizando o meu trabalho, numa primeira fase, foi escolhida a organização da equipa. E após isso, foi-me apresentando o *site* de forma a perceber como iria ser o seu *design*, comportamento e também, para me contextualizar sobre os conceitos de serviço bancário.

Ainda, esta apresentação foi feita com base na ferramenta Figma que por sua vez, esta veio a ser usada numa fase de implementação para eu conhecer os componentes a que estava associada e, ajudar-me a estruturar o componente na parte do código.

No entanto, foi apresentado no Azure uma secção designada de *Backlog*. Aqui, estavam descritas as *User Stories* que o projeto ia ter e onde eu já estava associada a algumas delas para vir a desenvolver durante o meu período de estágio e pelo qual, estão expostas no Capítulo 4.

Contudo, após analisar estas *User Stories* que estive envolvida foi juntamente com a equipa de BE (Backend) que negociei os contratos para os serviços que precisava na fase de desenvolvimento.

Assim, a equipa de Backend analisou também a proposta sugerida por mim e acabaram depois por formalizar comigo os contratos finais para integrar já devidamente tipificado e com o retorno que era esperado para aquele ecrã ou componente.

Para o desenvolvimento desta aplicação *web*, uma etapa inicial, é a seleção das linguagens que se vai utilizar para implementar este *website* e inclusive, introduzir uma primeira estrutura de como ficará organizado o código.

Deste modo, comecei por dedicar-me à implementação de código que foi desenvolvida com o recurso do uso do VSC (*Visual Studio Code*) e face às tecnologias, tive de usar o React JS e também, TypeScript, React Hooks, Redux Saga, CSS, JavaScript entre outras ferramentas que estão descritas no Capítulo 3. Também, tive de integrar micro-serviços que foram implementados pela equipa de *Backend* de forma a ter retorno de dados para um certo componente ou página que estava a desenvolver.

Na segunda parte do trabalho deste *website* pode-se averiguar que a arquitetura face a este novo desafio é ligeiramente diferente ao anterior. Aqui, a equipa está a trabalhar sobre a nova versão React JS V18, *framework* interna do banco e ainda se conta com o apoio de componentes migratórias internas para esta aplicação *web* reestruturada.

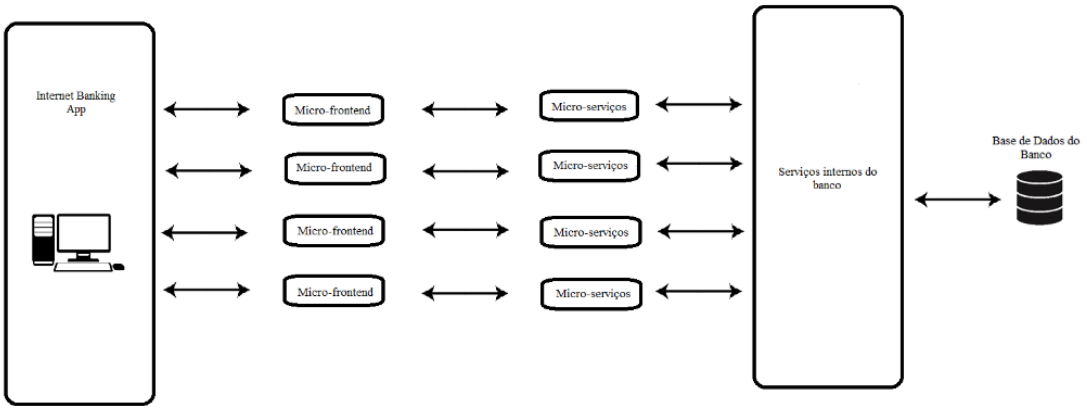

Figura 6 – Representa a arquitetura reestruturada para a aplicação *web*.

Para além desta nova abordagem, a equipa começou a trabalhar em *squads*, ou seja, como a equipa nesta segunda parte de trabalho é bastante composta a mesma vai ser

dividida em grupos mais pequenos para que a interligação entre todos os membros seja possível e organizada.

Assim, cada *squad* está definida para cada área envolvente deste projeto e permitiu que a entrega do produto fosse feita sem grande desalinhamento. E ainda, permite que cada um possa ser mais autónomo face ao que está a desenvolver e eficiente no trabalho que está a concretizar.

Todavia, nesta segunda fase de trabalho a equipa vai ter em conta aplicações federadas, ou seja, como a aplicação *web* é de grande dimensão necessitará de chamar outras aplicações de fora que não podem ser executadas sozinhas, ou então porque necessitam que executem todas juntas e desta forma passam a integrar um sistema compilado monolítico. Também, o que a equipa quer é evitar bibliotecas iguais, quer compartilhar facilmente o código e que sobretudo, seja flexível.

Sendo assim, e visualizando a aplicação a construir, figura 7, o que se tem são uma série de quadrados com vários tipos de funcionalidades desde o *header*, *sidebar*, perfil do cliente, componentes de *design* internos do banco, que podem, por ventura, estar em repositórios diferentes, e depois, a ideia é chegar a um ponto em que se arrasta cada um destes quadrados divididos desde a parte lógica, *design* ou os próprios ecrãs, desenvolvidos por equipas diferentes, até que se tem o *match* perfeito, isto é, através de módulos federados ambos possam ser consumidos em *real time* por qualquer aplicação que necessite destes mesmos módulos.

Também, após o código desenvolvido tive de subir o código aos ambientes, via PR (*Pull Request*). Esta fase está com mais detalhe no Capítulo 5.

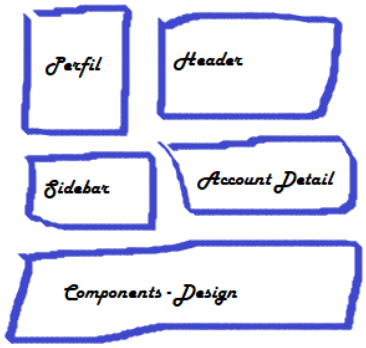

Figura 7 - Demonstra o contexto dos diferentes módulos diferentes que numa fase posterior podem ser consumidos ao mesmo tempo.

Ainda assim, fiz uso da ferramenta Postman que veio a ajudar-me a perceber se os serviços estavam disponíveis e ver o retorno de dados.

Porém, num fase terminal do projeto, existiram correções de *bugs*, mencionados na secção 5.4. Uma vez mais, passei pela análise novamente das US e perceber assim o erro que estava a tratar e ter uma solução para o mesmo.

Capítulo 3

Tecnologias/Ferramentas

3.1 Inicialização do projeto proposto

Primeiramente, apresentar-se-á as tecnologias e ferramentas utilizadas para este projeto e iniciar-se-á assim, numa fase posterior, toda a implementação necessária para o desenvolvimento desta plataforma designada de *Internet Banking*.

No entanto, face ao Estado da Arte este aqui não é abordado pois não tive qualquer envolvimento de decisão de escolha do produto para o cliente pois foi um trabalho realizado pela empresa.

3.2 Tecnologias de desenvolvimento

Para o desenvolvimento da *app web* como já referi anteriormente, as tecnologias/ ferramentas que vem a seguir são tecnologias usadas no *site* e que me ajudaram a concretizar a implementação das minhas tarefas.

3.2.1 React JS

O React JS [4] surgiu no ano de 2013 com o foco em criar interfaces de utilizador. No entanto, pelas diversas atualizações da versão, foi em 2018 que se começa a criar uma arquitetura muito mais consistente de forma a dar assistência aos projetos em que cada *developer* está inserido.

Relativamente a hábitos de programadores do *site* Stack Overflow, o React foi a terceira biblioteca mais citada pelos utilizadores e *developers* profissionais, ficando atrás do Node.js e Angular.

As vantagens do React JS [5] são:

- É flexível;
- Facilidade de migrar entre versões;
- Open Source;
- Boa documentação;
- Permite o reaproveitamento do código.

Uma das desvantagens do React JS é que não há uma entidade central responsável por produzir documentação.

3.2.1.1 React Hooks

Os Hooks [6] permitem que se use outros recursos do React sem escrever uma classe e desta forma, agiliza imenso o processo de desenvolvimento.

Os Hooks são funções que permitem a ligação de estado do React e os recursos de ciclo de vida dos componentes da função, mas, também, estes podem ser uma simples função para trazer dados sem interagir com o estado ou ciclo de vida dos componentes. Estes não funcionam dentro das classes.

O React fornece alguns Hooks integrados como *useState*, *useEffect*[2], *useContext*, *useReducer*, entre outros. Também, pode-se criar Hooks próprios para reutilizar o comportamento do estado entre diferentes componentes. Portanto, estes fornecem uma API mais direta para os conceitos do React como: *props*, *state*, *context*, *refs* e *lifecycle*.

```
1 const [titulo, setTitulo] = useState('');
2 useEffect(() => {
3     document.title = titulo;
4 }, [titulo]);
```

Figura 8 – Hook *useEffect* utilizando o estado [8].

[2] O *useEffect* simula vários *life cicles* do React.

Em suma, para utilizar os Hooks em componentes React é preciso seguir duas regras [7] específicas, caso contrário, não há como garantir a integridade do recurso na aplicação:

- Não se deve chamar os Hooks dentro de *loops*, funções *callback* ou estruturas condicionais;
- Os Hooks devem ser chamados dentro de um componente funcional React.

3.2.1.2 React Router

O React Router [8] é uma biblioteca padrão para o roteamento no React. Este permite a navegação entre visualizações de vários componentes numa aplicação React, deixa alterar a URL do navegador e mantém a UI *Design* (*User Interface*) sincronizada com a URL.

Os principais componentes [9] do React Router são:

- *BrowserRouter* – é uma implementação de roteador que usa a API (*pushState*, *replaceState* e o evento *popstate*) para manter sua UI sincronizada com a URL;
- Rota – é o componente mostrado condicionalmente que renderiza alguma UI quando seu caminho corresponde ao URL atual;
- *Link* – é usado para criar *links* para diferentes rotas e implementar a navegação em torno da aplicação;
- *Switch* – é usado para renderizar apenas a primeira rota que corresponda ao local, em vez de renderizar todas as rotas correspondentes.

O React Router apresenta algumas características [10] tais como:

- Validação de parâmetro de rota;
- Correspondência de rota no estado de localização;
- Gestão automática de foco nas transições de rota.

No entanto, esta biblioteca disponibiliza a utilização de quatro Hooks [11] que servem como ferramentas e/ou outras *features* de React sem precisar da utilização de classes como por exemplo do *useLocation* ou o *useHistory*.

```
import {
    BrowserRouter as Router,
    Route,
    Link,
    Switch
} from 'react-router-dom';
```

Figura 9 – Adicionar os componentes do React Router num certo ficheiro [11].

3.2.1.3 CSS

O CSS[3] [12] é uma linguagem de estilo usada para descrever a apresentação de um documento escrito em HTML ou em XML (incluindo várias linguagens em XML como SVG, MathML ou XHTML). Assim, é uma linguagem usada para controlar o estilo de desenvolvimento *web* e ainda lidar com a aparência das páginas *web*.

Esta linguagem, é uma das principais sendo open *web* e é padronizada em navegadores *web*. Desenvolvido em versões, temos:

- O CSS1 está atualmente obsoleto;
- O CSS2.1 é uma recomendação;
- O CSS3, agora dividido em pequenos módulos, está progredindo para a sua padronização.

3.2.1.4 JavaScript

O JavaScript (JS) é uma linguagem de programação interpretada simples e dinâmica, que, suporta ainda estilos de programação orientados a eventos, funcionais e imperativos, apresentado a recursos com *closures* e funções de alto nível. Também, tem como vantagem a facilidade de migrar entre versões.

A grande maioria dos *sites* utiliza esta mesma linguagem.

[3] CSS significa folha de estilo em cascata.

3.2.1.4.1 Emotion

O Emotion [13] é uma biblioteca que tem o intuito de escrever estilos CSS com JavaScript. Ela fornece composição de estilo poderosa e previsível, além de uma ótima experiência de *developer* com recursos como mapas de origem ou rótulos. Ambos os estilos de *string* e objeto são suportados.

3.2.1.5 TypeScript

O TypeScript [14] é uma linguagem de programação tipificada que se baseia em JavaScript. Ao utilizar o TypeScript, tem-se a possibilidade de aplicar tipos à programação, juntamente com interfaces em um sistema construído unicamente com JavaScript.

O TypeScript começou a ser desenvolvido pela Microsoft em 2012, com o objetivo de adicionar recursos e ferramentas que não estão presentes nativamente na linguagem como os tipos estáticos e a orientação a objetos [15].

No projeto usei o TypeScript para tipificar todas as variáveis, exemplifico usando a figura 10. Desta forma o JavaScript é potencializado.

```
interface Pessoa {
    idade: number;
    nome: string;
}

function imprime(pessoa: Pessoa) {
    console.log(pessoa);
}

let pessoa = {
    idade: 25,
    nome:'Teste'
};

imprime(pessoa);
```

Figura 10 – Ilustra um exemplo de TypeScript [45].

As vantagens [16] que o TypeScript[4] faz transparecer são:

- Tipo estáticos para o JavaScript e, é por meio destes tipos podemos construir aplicações muito mais seguras e melhora a produtividade;
- Potencializador da linguagem JavaScript. Ele permite que grandes sistemas complexos sejam construídos com essa linguagem sem nenhum problema e sem

[4] É uma linguagem de código-aberto, com intensa participação da comunidade.

que deixe a desejar diante de outras linguagens de back-end, como PHP ou Java.

3.2.1.6 Redux

O Redux [17] é uma biblioteca ou *package* que atua na administração de estados dos componentes de um certo programa. Este ajuda a escrever aplicações que se comportam de forma consistente e ainda, podem ser executados em diferentes tipos de ambientes como cliente, servidor e nativo.

Ter-se em consideração que o Redux tem como grande vantagem ter um estado global da aplicação centralizado.

Este tem no seu conjunto alguns recursos associados como:

- *Store* – esta é a parte mais importante do Redux pois é nela que se aloca os dados de cada um dos componentes da aplicação que se esteja a desenvolver. A *store* é consultada por cada um destes componentes quando se pretende realizar uma ação.

- *Reducers* – são funções puras e que fazem disparar eventos na aplicação a ser desenvolvida. Ou seja, deste modo, o *reducer* registará na *store* os dados das ações realizadas, modificando as condições mostradas. Por outras palavras, os *reducers* permitem tratar as ações retornando um novo estado.

- *Actions* – são conjuntos de informações que cada aplicação suporta e pelo qual, as *actions*, são alteradas na *store* a partir dos *reducers*. Isto para dizer que, as *actions* permitem disparar uma intenção de manipulação de dados na *store*.

Por fim, o Redux é bastante utilizado no desenvolvimento *web* ou aplicações *mobile* e, geralmente, são projetos extensos, que envolvem muitos componentes e funções que precisam ser atualizadas frequentemente. Também, o Redux, tem alguns *middlewares* [18] que complementa, como por exemplo:

- Redux Thunk;

- Redux Saga;

- Redux Logger;
- Redux Offline.

3.2.1.6.1 Redux Saga

O Redux Saga [19] é uma biblioteca de *middleware* usada para permitir que uma *store* Redux interaja com recursos externos a ela de forma assíncrona. Isso inclui fazer solicitações HTTP para serviços externos, aceder ao armazenamento do navegador e executar operações de E / S.

No Redux Saga [20], Sagas são implementadas usando funções de gerador. Para expressar a lógica do Saga, produz-se objetos JavaScript simples do gerador. Chama-se a esses objetos de efeitos. Um efeito é um objeto que contém algumas informações a serem interpretadas pelo *middleware*. Para criar efeitos, usa-se as funções fornecidas pela biblioteca do *redux-saga/effects package*.

Algumas vantagens que o Redux Saga apresenta são:

- Testabilidade (é muito fácil testar sagas, pois *call* () retorna um *pure object)*;
- Redux Saga vem com muitas funções auxiliares úteis sobre tarefas;
- As Sagas oferecem um local independente para lidar com todos os efeitos colaterais.

Já no que diz respeito a contras, o Redux Saga apresenta:

- Bastantes conceitos para aprender;
- Atualização de novas versões pode não ser compatível com as anteriores e os recursos já utilizados dos mesmos virem a ser desnecessários.

3.2.1.7 React Testing Library

A biblioteca React Testing [21] baseia-se na DOM Testing Library, isto é, a adição de API’s para trabalhar com componentes React. O seu principal objetivo é que quanto mais os seus testes se assemelham à forma como o software é usado, mais confiança eles podem dar ao realizar os mesmos.

O React Testing Library [22] é uma biblioteca de teste do React e tem um conjunto de auxiliares que permite testar os componentes do React sem depender dos seus detalhes de implementação. Embora não forneça uma maneira de renderizar superficialmente um componente sem os seus componentes filhos, um simulador de teste como Jest permite fazer isso por meio de demonstração.

Contudo, este [23], é uma alternativa ao Enzyme[5].

Enquanto o Enzyme permite testar os detalhes de implementação dos componentes React, o React Testing Library ajuda o programador a testar o comportamento dos componentes React da perspetiva dos utilizadores que usarão a aplicação.

3.2.1.8 Jest

O Jest [24] é uma biblioteca de teste de JavaScript com foco na simplicidade. No entanto, este interage com certas bibliotecas como Babel, TypeScript, Node, React, entre outras. Este tem como objetivo trabalhar sem configuração na maioria dos projetos JavaScript.

Esta *framework* de teste pode executar testes em paralelo de forma confiável. Primeiro executa os testes que falharam anteriormente e, só depois, reorganiza as execuções com base na duração dos arquivos de teste.

Como curiosidade, esta *framework* foi construída pelo Facebook e, também, esta é projetada principalmente para aplicações baseadas no React (que também é desenvolvido pelo Facebook), mas pode ser usada para qualquer base de código baseada em JavaScript.

As vantagens que o Jest manifesta são:

- Rápido e seguro;
- Cobertura do código;
- Suporte poderoso para simulação;
- Testes isolados e em área restrita;
- Boa documentação.

Em contrapartida, o Jest [25] apresenta:

- Poucas ferramentas ou bibliotecas são suportadas pelo Jest;
- Difícil de aprender;
- O teste instantâneo não se adequa à *framework* Jest.

3.2.1.9 Enzyme

O Enzyme é [26, 27] uma biblioteca de teste de JavaScript para React que torna mais fácil manipular e analisar a saída dos componentes do React.

Este foi criado pela Airbnb. O Enzyme contribui com alguns métodos de utilidade adicionais excelentes para renderizar um componente (ou vários componentes), encontrar elementos e interagir com os mesmos.

Tanto o Jest, quanto o Enzyme, são projetados especificamente para testar aplicações React.

O Enzyme pode ser usado sem a biblioteca Jest, mas o Enzyme deve ser combinado com um outro simulador de teste caso o Jest não seja utilizado.

3.2.1.10 Swagger

O Swagger [28] é uma *framework* composto por diversas ferramentas que, independente da linguagem, auxilia a descrição, consumo e visualização de serviços de uma API REST.

O Swagger permite criar a documentação da API de três formas:

- Automaticamente – simultaneamente ao desenvolvimento da API é gerada a documentação;
- Manualmente – permite ao programador escrever livremente as especificações da API e publicar as mesmas posteriormente em seu próprio servidor;

[5] Enzyme é um simulador de teste de JavaScript para React que torna mais fácil testar a saída dos componentes do React. Também, pode-se manipular, percorrer e, de algumas maneiras, simular o tempo de execução a partir da saída.

- *Codegen* – converte todas as anotações contidas no código fonte das API's REST em documentação.

Contudo, o Swagger apresenta também outras ferramentas associadas como, o Swagger Editor, o Swagger UI, o Swagger Codegen e o Swagger Inspector. Também, esta *framework* traz algumas utilidades [29] como por exemplo:

- Interface bastante intuitiva;
- Organização das rotas;
- Modelagem da API;
- Geração de código do Cliente e do Servidor, com suporte a várias linguagens de programação.

3.2.1.10.1 Swagger UI

O Swagger UI [30] a partir da especificação da API, permite ao programador criar documentações elegantes e acessíveis, abordando assim uma compreensão maior da API.

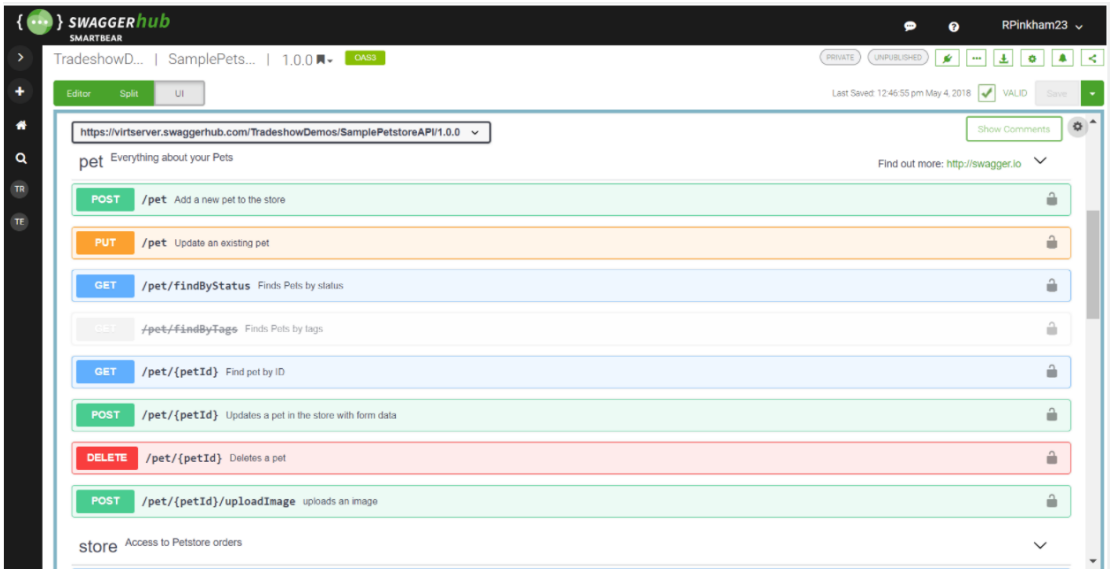

Figura 11 – Swagger UI exemplo que mostra a sua arquitetura.

Como se pode observar na figura 11, o Swagger UI dá a conhecer a forma como interage com os programadores. Então, pode-se observar que ela mostra quatro barras coloridas e elas representam quatro métodos: o POST, o GET, o PUT e o DELETE e, nestas mesmas barras ainda apresentam o caminho da rota que se pretende para os diferentes métodos. De seguida, rodeado a vermelho, figura 9, tem-se um botão "*Try it out*" e, ao clicar neste botão é possível testar a requisição que se pretende, ou seja, no campo "*Request body"* é possível fazer as alterações que se pretende, no exemplo da figura 12 tem-se o caso dos seguintes parâmetros, o id, o *name*, o *initialValue*, etc… e que podem ser alterados de forma a testar se estarão visíveis no GET/api/Accounts, referente já a uma conta criada. Para isso, modifica-se os valores e ao clicar em "*Execute"* pode-se observar que o método POST foi criado e está a ser representado no método GET, figura 13, respetivamente.

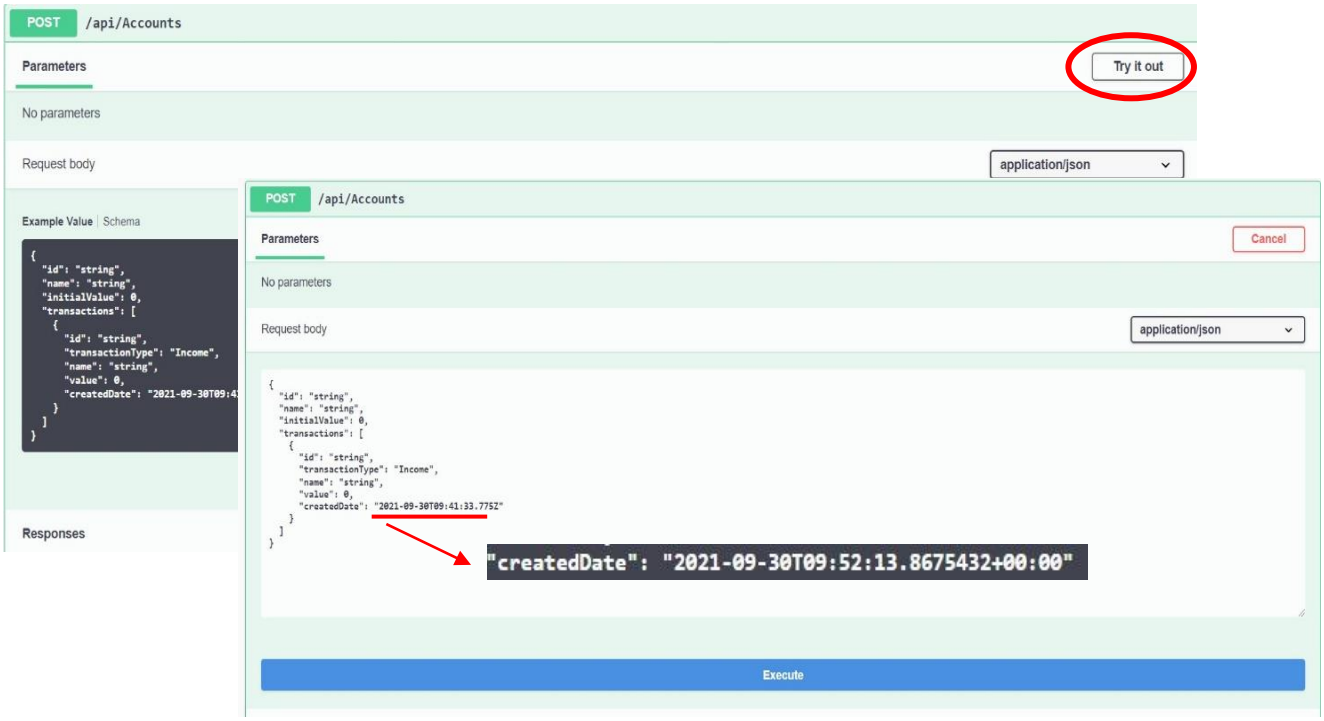

Figura 12 – Swagger UI de um exemplo Account a criar dados no método POST.

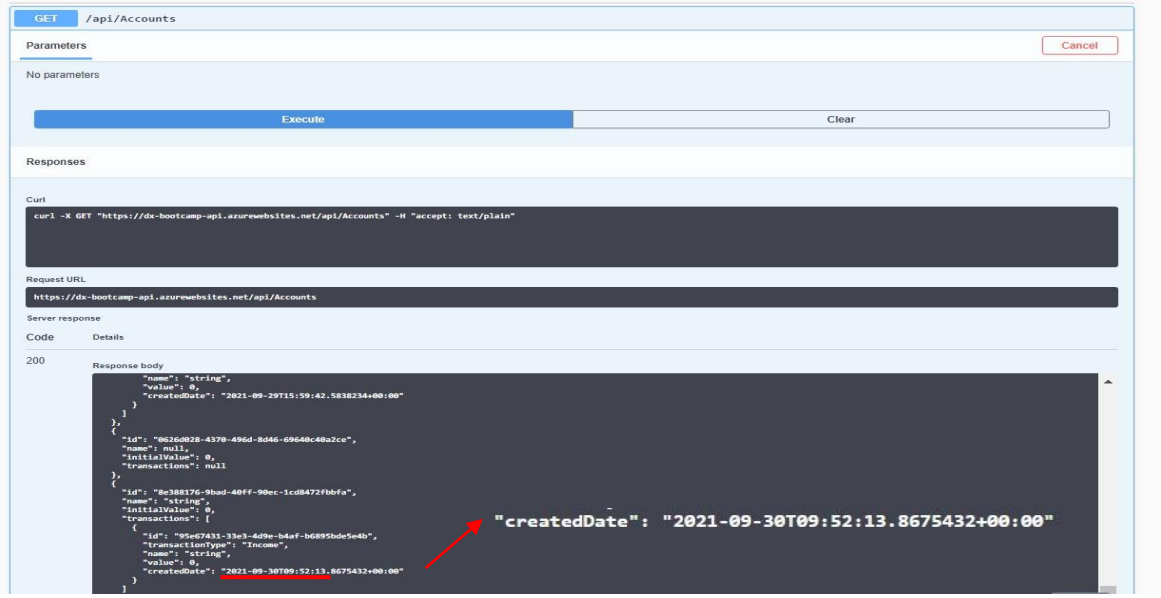

Figura 13 – Swagger UI exemplo Account com o método GET devidamente criado.

3.2.1.11 Babel

O Babel [31] é um conjunto de ferramentas usado principalmente para converter o código ECMAScript 2015+ em uma versão compatível com versões anteriores de JavaScript em navegadores ou ambientes atuais e mais antigos.

A abordagem que o Babel faz chegar até ao programador é que este é um transpilador, ou seja, ele busca o código escrito num padrão JavaScript e converte o mesmo num outro padrão diferente.

O Babel, neste projeto foi invocado automaticamente pelo transpilador da versão face ao uso de JavaScript.

As vantagens [32] que esta ferramenta possui são:

- Ajuda na escrita de código JS nas versões mais antigas de forma mais facilitada;
- Fornece compatibilidade com todos os recursos recém-adicionados ao JavaScript e pode ser usado em qualquer navegador;
- Pode ser usado junto com o Gulp, o Webpack, o Flow, o React ou o TypeScript tornando-o muito poderoso e assim, este ser usado em grandes projetos;
- Tem suporte para plugins, *polyfills* e *babel-cli*, facilitando o trabalho com grandes projetos;

- Fácil *debug*.

Os contras que se averigua no Babel são:

- O código BabelJS altera a sintaxe o que dificulta a compreensão do código quando está a ser analisado;
- Nem todos os recursos do ES6/7/8 ou alguns futuros recursos podem ser "transferidos" e ainda pode ser preciso usar o *Polyfill* para que funcione em navegadores mais antigos;
- O código geralmente que é alterado aumenta de tamanho em relação ao original.

3.3 Ferramentas Utilizadas

3.3.1.1 Visual Studio Code

Desde logo, que se decidiu a ferramenta de edição de código e pelo qual acompanhou o projeto, e ajudou bastante devido às suas funcionalidades. Assim sendo, a ferramenta escolhida foi o Visual Studio Code [33]. Esta ferramenta foi utilizada no projeto para o desenvolvimento dos componentes e funcionalidades que me foram atribuídas.

As vantagens [34] que se averiguou para este foram as seguintes:

- Plataforma com um conjunto vasto de funcionalidades;
- É gratuito e direcionado para quem está a aprender a programar;
- Tem facilidade em localizar referências, com o recurso Find All Reference (em português, Encontrar Todas As Referências), que auxilia no processo de localizar tipos ou funções específicas;
- Integração com o Azure e .NET;
- Multilinguagem;
- IntelliSense;
- Tem imensos atalhos e comandos que faz com que a produtividade aumente bastante no decorrer da codificação de um projeto;

- Suporta várias linguagens de programação.

3.3.1.2 Figma

O Figma [35, 36] é um editor gráfico de prototipagem de projetos de design baseado principalmente no navegador *web*, com ferramentas *offline* adicionais para aplicações *desktop* para GNU/Linux, MacOS e Windows. O Figma é um software focado no desenvolvimento de sistemas de design gráfico, prototipagem de interface gráfica do utilizador e desenvolvimento de UI/UX (experiência da interface com o utilizador), permitindo também o desenvolvimento colaborativo em tempo real com outros utilizadores remotamente.

Algumas das suas vantagens são:

- É gratuito para ferramentas gerais;
- Fácil de partilhar com uma equipa ou outro tipo de utilizador;
- Componentes interativos.

No entanto, o Figma, apresenta desvantagens tais como:

- O facto de ser uma plataforma lenta principalmente em ambientes MacOS;
- Não estar disponível *offline*, isto é, necessita de conexão à *internet* aquando da sua utilização.

3.3.1.3 Azure DevOps

O Azure DevOps [37] fornece um conjunto de ferramentas de desenvolvimento de software que se integram ao seu IDE permitindo assim que qualquer equipa trabalhe efetivamente em projetos de software.

Contudo, o Azure DevOps possui uma série de ferramentas que são úteis ao utilizador como por exemplo:

- *Kanban boards* – são quadros flexíveis que facilmente podem adaptar-se ao processo e estilo que se está a construir. Também, tem atualizações que garantem um ótimo fluxo da plataforma;

- *Backlogs* – usa-se para manter itens na ordem correta e conectado às coisas certas e a rastreabilidade mantém os itens em cada lista de pendências ligado aos diferentes cenários que está a ser usada para conduzir o projeto em questão;
- *Dashboards* – ajudam a construir visualizações personalizadas dos dados que se precisa;
- *Scrum boards* – construído com base no *scrum* e ferramentas de planeamento para ajudar as equipas. Estes *boards* funcionam eficazmente na medida de levantamentos, planeamentos, reuniões e retrospetivas;
- *Connect to code* – Rastreabilidade e visibilidade na evolução do código.

O planeamento com o Azure DevOps é baseado em metodologias ágeis, o que significa que tudo é planeado em ciclos curtos e, assim sendo, cada ciclo é chamado de iteração, e cada um deles é definido por:

- Um nome de iteração;
- Um período de data.

Em cada iteração, o coordenador define:

- O *Backlog*, ou seja, as histórias do utilizador a serem desenvolvidas, as suas tarefas e o esforço correspondente a cada uma dessas tarefas;
- A equipa: colaboradores bem como, a disponibilidade.

O Azure DevOps segue ainda outro tipo de funcionalidades tais como:

- Alertas de *email*;
- *Queries*;
- Colocar pin *queries* na página inicial;
- Criar mais do que um *dashboard*.

No entanto, como vantagens [38], o Azure DevOps apresenta as seguintes:

- Otimização de *deployments* e velocidade;
- Segurança e fiabilidade.

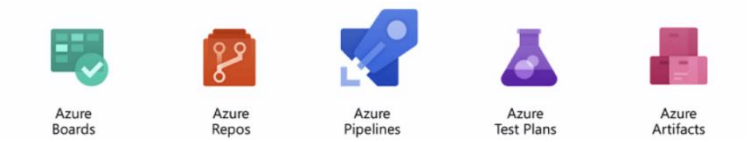

Figura 14 – *Azure DevOps* é dividido em cinco componentes principais [46].

3.3.1.4 Git

O *Git* [39, 40] é um sistema de controle de versão distribuído. Com este, existe uma facilidade de trabalho *offline* ou remotamente.

No *Git*, podemos modificar arquivos, prepará-los e posteriormente fazer *commit*[6] para um repositório como por exemplo, Azure DevOps.

No projeto que desenvolvi, a importância que teve o Git foi ajudar-me com a subida do código aos ambientes, utilizando os seguintes comandos como, *o "git pull"*, o *"git push"*, o "*git checkout*", o "*git merge"* ou o "*git commit"*.

Porém, ao utilizar esta ferramenta uma das vantagens que verifico é a facilidade de trabalhar em equipa, pois permite cada elemento da mesma ter a sua *branch* individual e após a sua conclusão da implementação é possível juntar a uma só *branch* mais geral e assim, se reunir todo o desenvolvimento; também, é de fácil aprendizagem, confiável e flexível.

3.3.1.5 Astah UML Diagrams

O Astah UML Diagrams [41] é um pograma de modelação UML, e este *software*, oferece suporte a todos os tipos de diagramas UML (*Unified Modelling Language*) que se pretenda construir.

Esta plataforma é uma comunidade que abrange uma série de perfis desde os estudantes, os professores, grandes equipas e, face ao seu acesso fácil e compreensível, de um modo geral, um certo utilizador consegue utilizar de forma intuitiva esta plataforma na construção e visualização dos seus próprios diagramas.

O Astah UML traz alguns benefícios tais como:

[6] Este comando move os arquivos da *state* área para um repositório local.

- *Open Source* em alguns recursos;
- Recursos para diagramas que são fáceis de manipular e organizar da forma que se pretende.

No entanto, este *software* como contras tem alguns recursos bloqueados face à versão *Open Source* como já referido.

3.3.1.6 MIRO | Website Wireframe Template

O Website Wireframe Template – MIRO [42] – é uma plataforma muito intuitiva e usada para criar *wireframes* para o desenvolvimento de determinado projeto que se queira construir.

Existe uma série de plataformas que traz esta vantagem de o utilizador desenhar a sua própria aplicação e representar a mesma para a sua equipa de forma eficiente e clara.

Assim sendo, a principal vantagem que se encontra é mesmo ter a possibilidade de criar o próprio *design* que cada um quer dar a uma determinada aplicação, seja *web* ou *mobile*, e de uma forma rápida mostrar a um conjunto de pessoas e, assim ser-se perspicaz no enquadramento do assunto que se quer abordar.

Além do mais é uma aplicação gratuita e qualquer pessoa pode facilmente aceder à mesma no seu portátil.

No caso do projeto que estive a desenvolver usei o MIRO para a representação da aplicação web que estou envolvida.

Capítulo 4

Abordagem/Análise/Modelação

4.1 Análise das User Stories

As *User Stories* em que eu estive envolvida no projeto são apresentados no Apêndice D, mas deixo abaixo alguns exemplos dessas US (*User Stories*).

Neste projeto em específico tem-se como atores: o agente bancário e o cliente. As tabelas estão identificadas pelo ator em questão e, dentro da mesma, podemos organizar pelo identificador, nome, prioridade e uma breve descrição que esse ator terá.

- Cliente

Identificador	Nome	Prioridade	Descrição
USC01	Editar dados pessoais	Muito alto	Como cliente quero editar as informações da conta.
USC04	Consultar as contas associadas	Muito alto	Como cliente posso consultar todas as contas associadas.
USC21	Consultar perguntas frequentes	Muito baixo	Como cliente posso consultar as perguntas frequentes.

Tabela 1 – Tabela que gere as US do Cliente.

- Agente Bancário

Identificador	Nome	Prioridade	Descrição
USA04	Consultar detalhe da conta dos clientes	Muito alto	Como agente bancário quero poder consultar em detalhe todas as informações das contas.
USA05	Consultar as contas dos clientes	Muito alto	Como agente bancário posso consultar todas as contas associadas ao banco.
USA06	Pesquisar contas dos clientes	Muito alto	Como agente bancário posso pesquisar contas dos clientes.

Tabela 2 – Tabela que gere as US do Agente Bancário.

4.1.1 Diagrama de Casos de Uso

Em resumo, segue-se com as permissões dos atores, e assim obtém-se o seguinte:

- Agente Bancário: gere todas as contas, consulta os seus dados pessoais, adiciona consultores a contas dos clientes, encontra e valida empréstimos e, pesquisa a conta dos clientes;
- Cliente: pode consultar as suas contas, pode adicionar os consultores, pode editar o seu perfil, pode descarregar comprovativos, pode consultar o seu capital financeiro e, claro, consultar os tipos de pagamento (pagamento ao estado ou transferência).

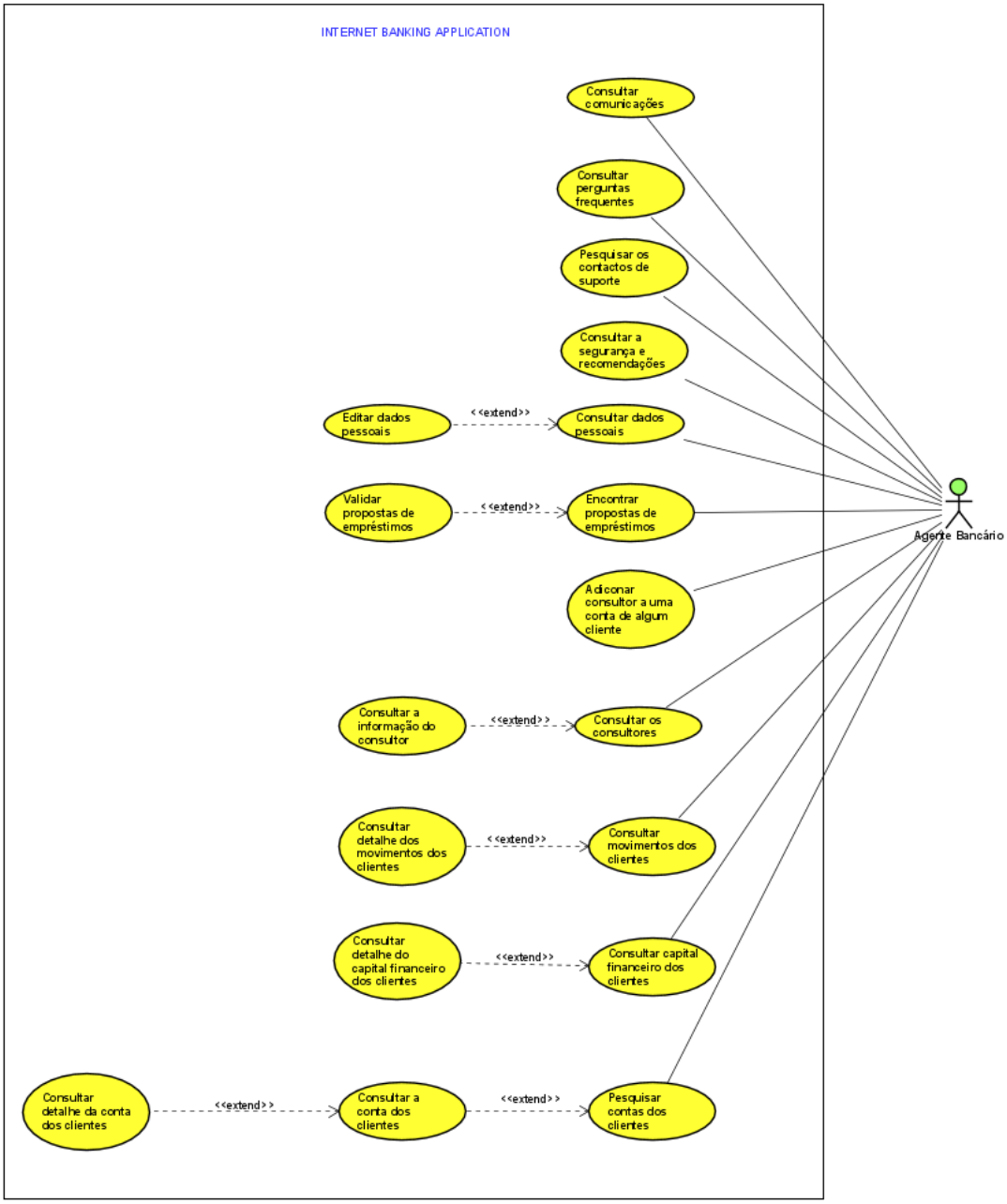

Figura 15 – Diagrama de Casos de Uso do ator Agente Bancário.

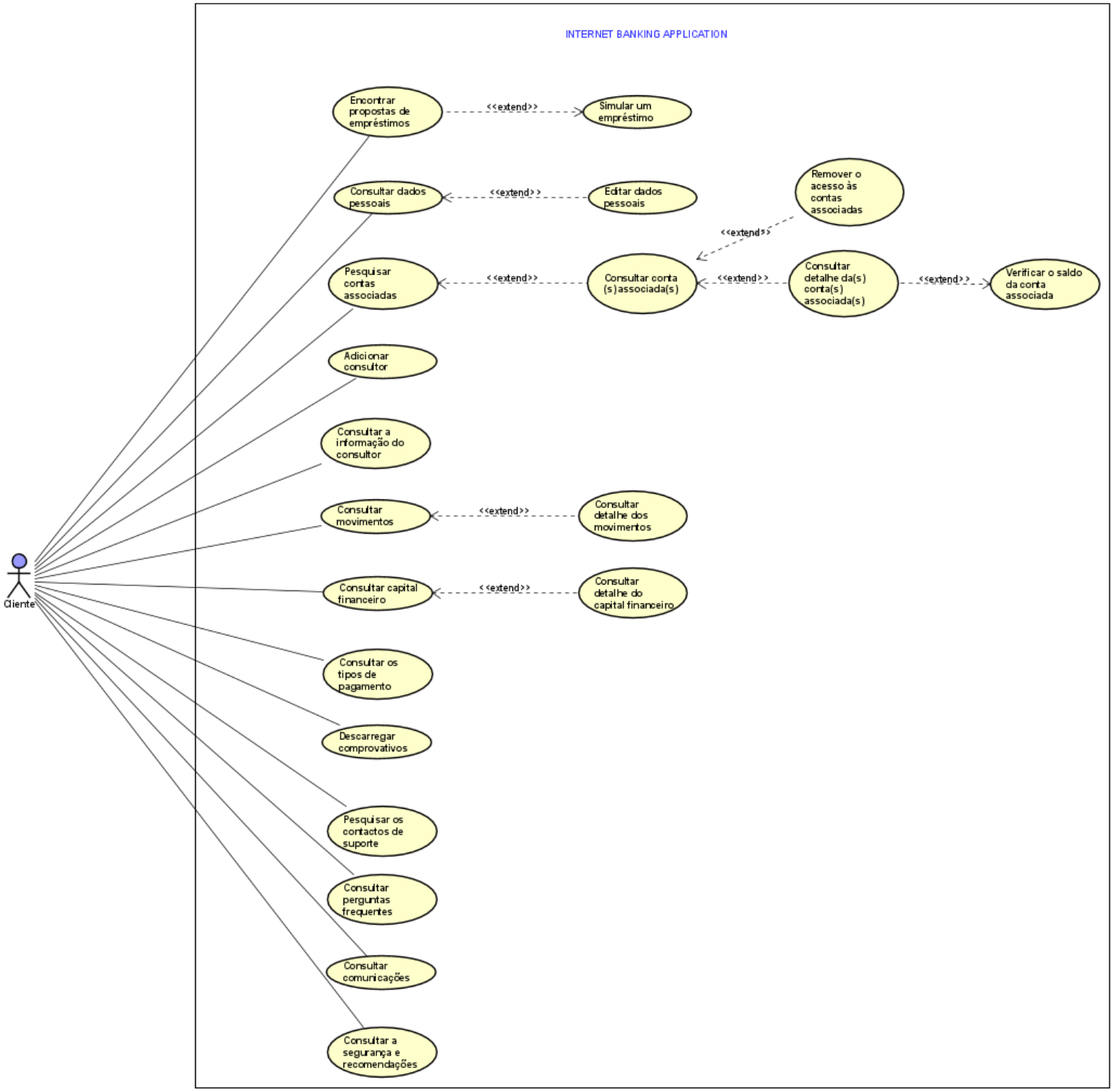

Figura 16 – Diagrama de Casos de Uso do ator Cliente.

4.1.2 Wireframes

Nesta subsecção, estão expostos exemplo de *wireframes*[7]. Todos os *wireframes* constam no Apêndice A, do respetivo projeto.

Na figura A.10, pode-se observar que o projeto vai apresentar uma série de funcionalidades essenciais para satisfazer a necessidade ao cliente e, seja de fácil acesso a toda a comunidade.

Assim, na figura A.1, é nos apresentado um *wireframe* que será visualizado pelo cliente, e, a interação que o cliente tem nesta página é que pode ver os destaques ou informações que o banco tem para anunciar e oferecer; pode ir para os seus movimentos; pode simular um crédito; pode encontrar propostas de crédito; associar e/ou ver os seus consultores; ver o saldo da sua conta; e, ainda, ir para o seu perfil caso pretenda editar o mesmo.

Depois, na figura A.2, encontra-se uma página relativa ao crédito, isto é, uma funcionalidade pelo qual o cliente pode colocar nos campos montante e prazo, os valores que pretende e depois simular o crédito, resultando numa prestação mensal.

Na figura A.3, apresenta-se um *wireframe* que retrata o saldo da conta, bem como, os movimentos efetuados pelo cliente.

Na figura 17, o cliente vê o ecrã com as contas associadas e pelo qual pode editar o nome dessas contas. Também, na figura A.5 representa um componente que desenvolvi e que permite ao cliente remover o acesso a uma certa conta.

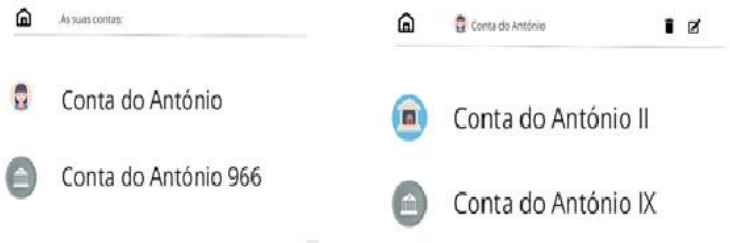

Figura 17 – *Wireframe* que representa o ecrã com as contas associadas.

[7] Todos os *wireframes* estão disponibilizados no seguinte link https://miro.com/app/board/o9J_lqzNE5w=/ . Apenas com acesso para orientadores e professores do estágio desenvolvido no Instituto Politécnico de Bragança.

Depois, na figura 18, é apresentado um *wireframe* que tem campos editáveis no momento de edição da conta, mas o cliente está com acesso negado para certos campos, tais como, IBAN, SWIFT e o próprio número de conta. Aqui, ainda tem a possibilidade de ver as contas e cartões associados para se selecionar um deles aparecerá um *modal*, figura A.7, para remover ou não uma das contas ou cartões.

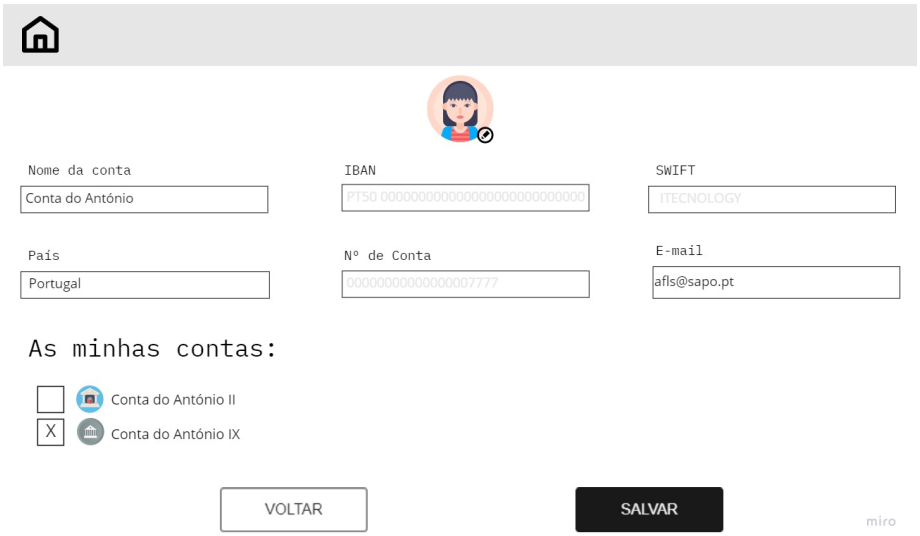

Figura 18 – *Wireframe* a representar a edição da conta.

Posteriormente, na figura A.8, tem-se a listagem dos últimos movimentos de certo cliente e, pode selecionar o Movimento que quer para ver detalhadamente.

Depois, na figura A.9, visualiza-se o movimento selecionado no *wireframe* anterior para ver em detalhe o que foi sucedendo. Neste caso, o cliente pode ver o valor total da conta, os movimentos efetuados neste mês e fazer os respetivos *downloads* dos comprovativos.

Já numa outra fase do desenvolvimento desta aplicação, é possível que o cliente adicione consultores, figura A.11 e figura A.12. Ora, nesta funcionalidade e antes de integrar o consultor diretamente ao seu perfil ele tem algumas etapas a cumprir e são representadas na figura A.13 e A.14, mas para chegar a esta área terá de entrar pelo *Home Page*.

Nas figuras referenciadas anteriormente, ele terá à sua disposição a leitura de um PDF. E, finalmente, é remetido para uma página de consultor adicionando com sucesso, figura A.15.

Pode ainda ver-se depois todos os consultores que tem já arrecadados na sua conta, figura A.11.

Na figura 19, após o cliente selecionar uma conta pode visualizar alguns *cards* que são compostos por contas ou cartões e, informações dos movimentos. Também, se tiver diversas contas ou cartões, pode fazer uma pesquisa e ainda pode clicar no ícone do *Home* para voltar á página inicial, com recurso ao uso do React Router.

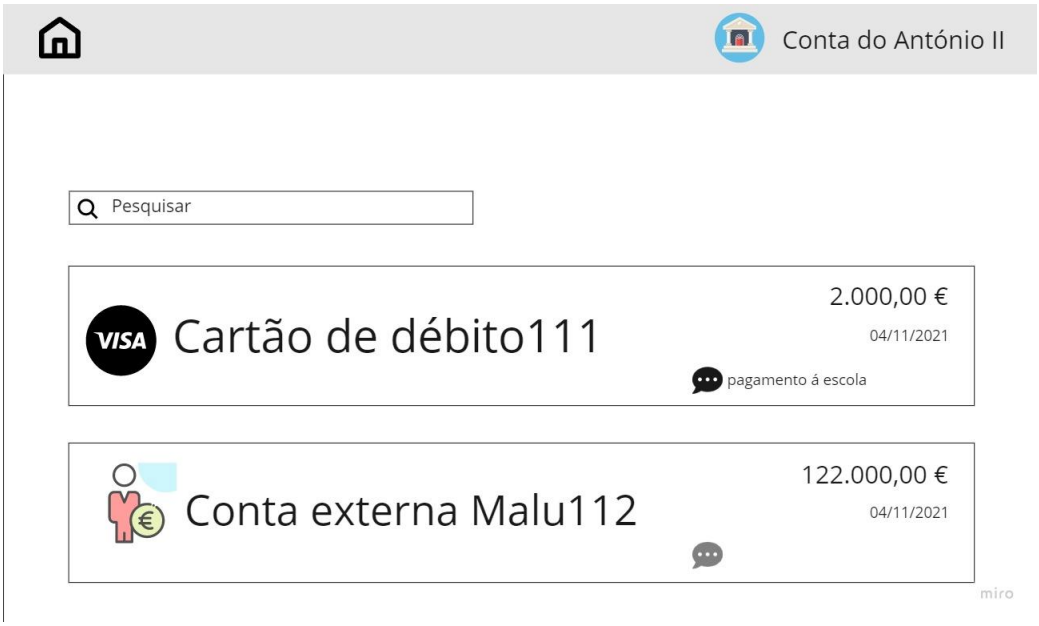

Figura 19 – *Wireframe* representativo de um certo cliente que contem duas contas associadas e pode ser conta ou cartão.

Capítulo 5

Desenvolvimento/Implementação

O capítulo cinco apresenta a implementação do trabalho proposto que foi elaborado por mim e, desta forma, aborda-se conceitos como estrutura de código, micro-serviços, entre outros temas pertinentes nesta área.

Assim sendo, a linguagem principal utilizada para o desenvolver deste *website* é o React JS e como estrutura de código, numa primeira instância, tem-se dentro da pasta "*src*" , figura B.1, Apêndice B, um conjunto de pastas que faz com que o código para esta implementação seja funcional e daí se traduza para que o cliente tenha interação com o *website*.

Salienta-se ainda que a construção deste *website* tem duas fases, uma inicial (construção do *site* principal) e uma outra fase que é de reestrutura site antigo do banco.

Contudo, estive a trabalhar sobre parte dessas mesmas funcionalidades, que são:

- Implementação e análise do *Header* e Menu Vertical;
- Implementação e análise de gráficos;
- Implementação e análise de componentes editar a conta, ver dados pessoais;
- Implementação e análise de página de privacidade e *cookies*, de segurança e recomendações, de perguntas frequentes, e de contactos de apoio;
- Implementação e análise de comprovativos de ficheiros;
- Implementação de transferências/ pagamentos;
- Implementação de modais de tempo de sessão expirado.

5.1 Fase 1 da aplicação web

Como ponto de partida comecei por analisar as *User Stories* e após isto contratualizar com o Backend os serviços.

Deste modo, passo a descrever cada pasta de forma a mostrar o trabalho que realizei para depois conseguir realizar as tarefas que fui submetendo ao longo do meu estágio.

Numa primeira fase, a pasta "*src*" está dividida em treze pastas e são as seguintes:

- *Assets* – nesta pasta constam todas as imagens necessárias ao projeto;
- *Components* – nesta pasta, tem-se componentes que serão reutilizáveis. Sendo assim, através destes, consegue-se em pequenas partes de código colocar a lógica de código e, reutilizar de página em página. Também, como se utiliza o React JS os componentes conseguem receber entradas, isto é, *props* que retornarão posteriormente novos valores;
- *Configs* – nesta parte, está todas as configurações essenciais à aplicação. Ora, através da *framework* em questão tem-se um conjunto de características que permitirão ajudar o *developer* para a comunicação, para a criptografia de dados, a gerir a conectividade externa, a fornecer recursos de segurança, a receber e/ou pedir dados de configuração remota, entre outros exemplos úteis para a aplicação *web*;
- *Constants* – nesta pasta, criou-se um *file*, "translation.ts", onde permitirá disponibilidade, à restante *app*, de traduções necessárias e ainda, a sua estruturação foi feita por áreas e/ou páginas de forma a simplificar a sua chamada;
- *Contexts* – nesta área, tem-se os contextos importantes para o funcionamento da aplicação, isto é, é apontado como uma abordagem mais fácil e leve para gerir o estado usando o Redux ou mover propriedades de pai para filho, por exemplo;
- *Helpers* – neste campo, este, ajuda o *developer* em assuntos gerais da *app* como traduções e até mesmo com os pixéis que se utiliza. Por exemplo nesta pasta foi criada uma função que converte pixéis em "*rems"*, de modo a não sofrer desfigurações em ecrãs maiores;
- *Hooks* – na pasta *Hooks*, estes, permitem a ligação de estado do React e os recursos de ciclo de vida dos componentes da função;

- *I18n* – nesta pasta, temos as *translations* locais;
- *Store* – nesta área, e, através do *Redux Saga*, tem-se dados *mock* que enquanto os serviços não estejam ligados, estes permitem a visualização de outros dados por meio de *props*. Aqui divide-se por áreas definidas pelo cliente e, desta forma, ajuda na estrutura do seu desenvolvimento e organização;
- *Themes* – nesta secção, ela apresenta os temas e as cores da *app*;
- *Types* – nos *Types* tem-se os tipos da aplicação;
- *Ui* – nesta pasta, *Ui*, nós temos mais duas subdivisões que são, uma espécie de grupos reservados (*groupChildren*) e uma outra espécie de grupo que agrega os grupos reservados (*groupParent*). Assim sendo, no de grupo que agrega os grupos reservados, tem-se quatro pastas (*area1*, *area2*, *area3*, *common*) e onde tem-se o conjunto dos componentes, e nos grupos reservados é representado as mesmas pastas com o intuito de criar componentes não reutilizáveis e, podem ser ou mudados de ordem, ou mostrados ou então, ocultados, mas, contudo, caso exista a necessidade de reutilizar certo componente também se pode fazê-lo chamando da pasta "*Components*".

No entanto, para o desenvolvimento da aplicação *web*, é feita a sincronização do Visual Studio Code com o repositório no Azure.

De seguida, efetua-se a instalação através do terminal do VSC com o comando "npm i", e assim, alguns *packages* necessários são configurados, desde o React, o TypeScript e o Prettier[8]. Também, todas estas instalações podem ser vistas no *package.json* bem como a versão em que se encontra e também modificar a versão atual se assim se pretender.

Este desenvolvimento, consiste em implementar as páginas dinâmicas e previamente, estas foram ilustradas nos *wireframes* que se destacam nos Anexos do presente relatório.

Primeiramente, o trabalho da equipa foi divido por ecrãs e componentes individuais para que depois numa fase terminal todo o trabalho fosse interligado e além disto, os mesmos encontram-se em áreas distintas com US associadas para facilitar a implementação de cada elemento da equipa.

[8] É um formatador de código que visa ajudar os programadores a escrever aplicações que são mais fáceis de entender e mais uniformizadas entre as diversas formas de programar que existem. Atualmente ele fornece formatação de código para as seguintes linguagens, JSX, Flow, TypeScript, CSS, HTML, JSON, Angular, etc.

Sem demora, abordo de seguida com detalhe os ecrãs e componentes realizados ao longo deste processo de trabalho. Assim sendo tem-se o seguinte:

1) *Home Page* – neste ecrã em concreto o cliente tem a possibilidade de selecionar um ou mais consultores para que integrem a sua conta se assim o desejar. Para isto, criei na pasta *groupParent*, um ficheiro do tipo ".tsx" designado por *homePage* onde começa-se por fazer os devidos *imports* e seguidamente criar diferentes constantes que invocam aquilo que o ecrã levará no seu conjunto, ou seja, ir-se-á buscar os grupos criados na pasta UI.

   Depois, na pasta *groupChildren*, crio outro *file* cujo nome é "homePage.tsx" para que nessa pasta contenha os diversos componentes que o ecrã vai ter para mostrar ao cliente. E, os componentes realizados para a página inicial foram assim o *header*, o *footer*, os movimentos, a seleção de consultor, entre outros.

   Deste modo, liga-se a serviços realizados pelo Backend já negociados anteriormente, para testar as diferentes funcionalidades e, com o recurso ao Postman inicia-se sessão com um *ott* (*one time token*), figura 20, para que o *localhost* da *web* nos responda com a interação que é pretendida como o caso de, ao clicar no botão selecionar consultor, o serviço traz os diferentes consultores, depois, o cliente escolhe, e, de seguida, o serviço responde trazendo as condições gerais, ou seja, um PDF que terá de ser lido até ao final e, após isso, ao clicar que concorda com os termos, um outro serviço do BE volta a invocar uma nova chamada que fará com que se receba uma SMS com o código de acesso e, assim, passando para a última fase que é, ter o consultor adicionado com sucesso à sua conta.

   No entanto, através de *mocks*[9], é possível fazer diversos testes pois eles permitem que sem serviços consiga manipular do lado FE, os dados pretendidos e desta forma o resultado em causa é mostrado.

[9] São objetos que simulam o comportamento de objetos reais de forma controlada, ou seja, são objetos falsos que simulam o comportamento dos ecrãs funcionais quando não se tem serviços com disponibilidade.

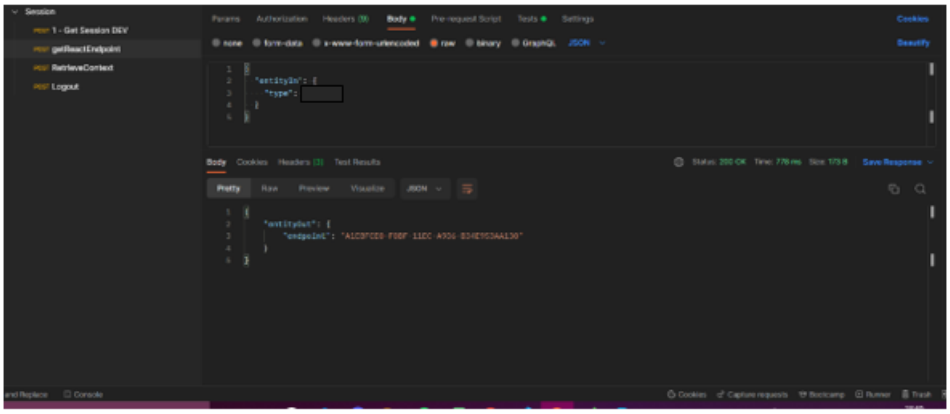

Figura 20 – Demonstra o *ott* a ser utilizado depois no VSC para que a aplicação seja chamada na *web*.

Movimentos – neste componente, começa-se por visualizar na US aquilo que é pretendido e, sendo assim, verifica-se que irei ter um conjunto de *label's* compostas por o nome do movimento, a data, bem como o valor desse mesmo movimento. E, ainda terá um título que indica que se refere só a movimentos, bem como um *button*, que se clicar no mesmo levará para um detalhe em específico desse mesmo movimento. Aqui, o desenvolvimento deste componente seguiu o mesmo caminho que o anterior já descrito até porque, como é componente inserido no *Home Page* só criei um novo *file* do tipo ".tsx" na pasta movimentos e, assim, o *groupParent* do *Home Page* foi buscar este componente ao *groupChildren* para ser agregado na parte do ecrã que se queria. Os movimentos provenientes do serviço são guardados na *store* através do Redux e Redux Saga.

2) Remover acesso às contas – na criação deste componente, comecei por construir um *modal* que permite se o cliente quer ou não remover o acesso à conta. Portanto este *modal*, composto por dois *button's*, um para cancelar, que ao clicar neste irá retornar para a página inicial, com recurso ao uso do React Router e, outro botão para remover e, também, é composto por uma *label* que indica "Quer remover a conta X?". No seu desenvolvimento, este foi criado na pasta componentes pois será reutilizável na sua maioria pela *app* tendo funções diferentes, mas com o mesmo corpo de código *open-source*. Uma vez mais através de *mocks* consegue-se fazer testes enquanto os serviços da parte do BE não estão totalmente integrados e pelo qual vieram numa fase mais tardia da implementação.

3) Perfil do Cliente – nesta página são apresentadas todas as informações do cliente. Para tal, foi implementado um *useDidMount (*é um *hook)* que recolhe as variáveis locais, como o caso do *token*.

O *token* é utilizado para garantir que o pedido à base de dados é efetuado por uma pessoa autenticada, e, de seguida é efetuado um pedido GET à base de dados sendo passado o *userId* (id do cliente, por exemplo), para recolher a informação relativa a esse mesmo id. Os campos são renderizados em formato de formulário e é apresentado um botão de salvar que redireciona para a página inicial da aplicação, com recurso ao uso do React Router, e um botão para cancelar e, ao ser clicado, é aberto um *modal* implementado para confirmar se pretende de facto cancelar as informações alteradas. Caso confirme, é então efetuado um pedido delete à base de dados sendo passado o *userId*.

4) Página de detalhe das contas – nesta página podemos ver os detalhes de uma determinada conta. Estes dados são recolhidos localmente após serem enviados por outras páginas. Para além de apresentar os dados da conta, foi implementada a funcionalidade de adicionar outras contas e, para tal, foi criado um método que efetua um pedido POST em que no corpo da função são enviados os dados da conta *main* e assim faz depois a adição da conta secundária.

Também aqui foi implementado mais duas funcionalidades que é o poder editar a conta que adicionou, estando limitado em certos campos como o IBAN, SWIFT e Nº de Conta, que não podem ser alterados pelo cliente.

5) Página Editar Perfil e Editar Conta Associada – nestas páginas o cliente pode editar e é possível ao cliente alterar as suas informações de ambas as contas.

Foram implementados *useDidMount's (*são *hooks)* que recolhem o valor das variáveis locais, o *token*, *type* e *userId*. O *token* é utilizado para garantir que os pedidos à base de dados são efetuados por utilizadores autenticados. De seguida é efetuado um pedido GET à base de dados. É passado o *userId e* a resposta é guardada na *store* através do Redux e Redux Saga e posteriormente é invocada para apresentar os campos pré-preenchidos do formulário.

É renderizado então o formulário, com os campos nome, morada, código postal, telemóvel, país, email e fax, cada um destes campos é preenchido pelo *default value* definido nas variáveis independentes guardadas após o pedido GET.

Foram implementados *handlers* independes que guardam a informação alterada em cada um dos campos do formulário. Após efetuar as alterações, e clicando no botão de salvar, é efetuado um pedido PUT à base de dados sendo passado o *id*

e no corpo da função as variáveis alteradas pelos *handlers*. No entanto, os valores definidos pelo banco como IBAN, SWIFT e Nº de Conta são campos truncados e inalteráveis pelo cliente.

5.2 Fase 2 da aplicação web

Começa-se então por mencionar em que sentido vai ser retrabalhado esta segunda parte do *site* bem como, mostrar algumas das novas funcionalidades e *design* que o mesmo passa a incorporar. Assim, segue-se com os novos ecrãs implementados e que desenvolvi:

- *Header* – neste componente, o mesmo sofreu alterações face ao anterior isto porque o mesmo obteve uma nova *label* que indica a hora e a data da sessão iniciada. Este novo campo vem acompanhado também de um serviço interno que traz uma hora e uma data para a conta que está a efetuar o *login*.
- Menu Vertical – no menu vertical observa-se um menu semelhante ao principal, mas que fica do lado esquerdo em modo vertical da página e que possui funcionalidades como ver os pagamentos, consultar os consultores, consultar ficheiros guardados de comprovativos realizados anteriormente, ver qual o gestor associado à nossa conta e ainda, ter um contacto de ajuda para determinado momento de impedimento num acesso a estas atividades.
- Linha especial formada por botões – neste componente, pode-se verificar que o mesmo efetua páginas transacionais naquilo que se pretende e é composto por quatro elementos funcionais que levarão a outras quatro páginas de intuito diferente. As quatro *label's* que esta linha especial com *button's* tem são:
    - Corrente – nesta área tem-se desde *cards* a mostrar com algum tipo de detalhe da conta, bem como, traz o saldo dessa mesma conta, sendo ele positivo ou negativo.
    - Atividade – pode-se verificar nesta área os últimas atividades efetuadas e, ainda, com algum detalhe, desde o nome, a hora e a possibilidade de transferir o comprovativo desse mesmo movimento.
    - Capital – nesta área o cliente pode ver o seu património, isto é, moedas de coleção que possa eventualmente ter, poupanças, entre outros investimentos que tenham sido realizados pelo mesmo. Contudo, é uma página transacional

que tem um gráfico que recebe este conjunto de variáveis e através dele mostra uma *tooltip* que dá a representação do total do capital que tem, bem como, a percentagem que cada parte do seu capital foi investido.

- Comunicações – uma página composta por comunicações que podem ser do tipo alerta como, por exemplo, de não se esquecer de atualizar os dados da conta ou então comunicações de informações gerais sobre o serviço bancário. Porém, pode não ter comunicações ou qualquer outro tipo de alerta.

- Pagamentos – tem-se uma área toda ela dedicada a pagamentos tal como maior parte dos serviços bancários nos permite fazer.

  Apresenta-se aqui um conjunto de *label's* a indicar ao cliente qual o passo deve seguir de forma a prosseguir com o respetivo pagamento e apenas com três tipos disponíveis. Ainda, é-lhe apresentado a seguir um *modal* com a confirmação dos dados que colocou para um posterior pagamento que queira realizar.

- *Dashboard* – neste componente, figura 21, o mesmo mostra um gráfico que representa a visão da atividade tida, para dar a conhecer ao cliente essa mesma visão da sua conta e, ainda, mostra as quantidades que o cliente fez essa operação nesse determinado mês.

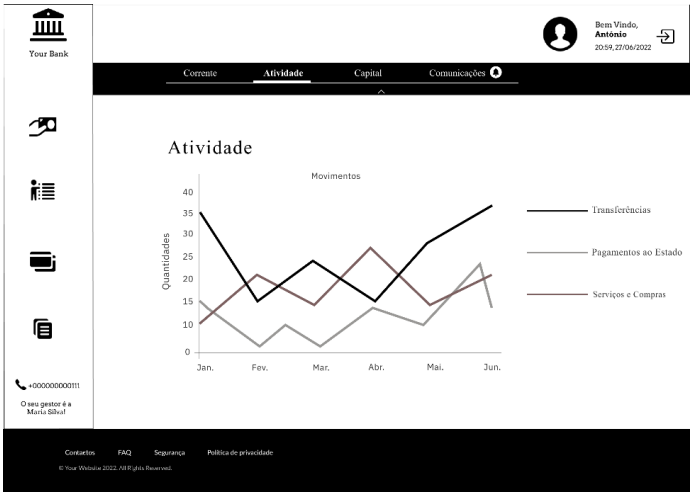

Figura 21 – *Mockup* que representa o gráfico relativamente aos movimentos.

- Página de Privacidade e *Cookies* – implementação estática desta página em que é possível ver a privacidade do *site* e *cookies*.

- Página de Contactos – implementação estática desta página em que é possível ver os contactos de suporte a qualquer dúvida que surja na aplicação ou em alguma funcionalidade específica.
- Página de Segurança e Recomendações – implementação estática desta página em que é possível ver a segurança e recomendações.
- Página de Perguntas Frequentes e Página de Detalhe de cada tema– implementação estática desta página em que é possível ver as perguntas frequentes e também, ao clicar no tema que deseja pesquisar, surge uma outra página com a explicação desse mesmo tema.
- *Modal* de eliminar ficheiros – Para ser eliminado um ficheiro, foi implementado primeiro um pedido GET, em que são recolhidos os ficheiros para um *array* e, de seguida, é retirado desse *array* o ficheiro e ao clicar no botão através da função *pop*, caso o *array* fique vazio, é então efetuado um DELETE, caso ainda possua mais ficheiros é efetuado um PUT e é efetuado o *reload* à página.
- Página de Serviços Inválidos – neste ecrã tem-se um *modal* que nos mostra uma *label* "Por favor, repita esta operação mais tarde. ", isto porque, os serviços internos podem ter entrado em indisponibilidade e só minutos ou horas depois o mesmo retornará ao ativo. Assim, quando isto acontece, é lançado um *modal* pelos serviços internos do sistema bancário, figura 22.

Figura 22 - *Mockup* que representa o *modal* de indisponibilidade de serviço.

- Página de Comprovativos – nesta página tem-se uma tabela que apresenta os comprovativos, figura 23, de todas as operações realizadas e onde é possível através

de cada um deles fazer o seu *download*, onde é mostrado o componente *toast* de sucesso do mesmo ou então, tem-se um ícone para eliminar esse comprovativo.

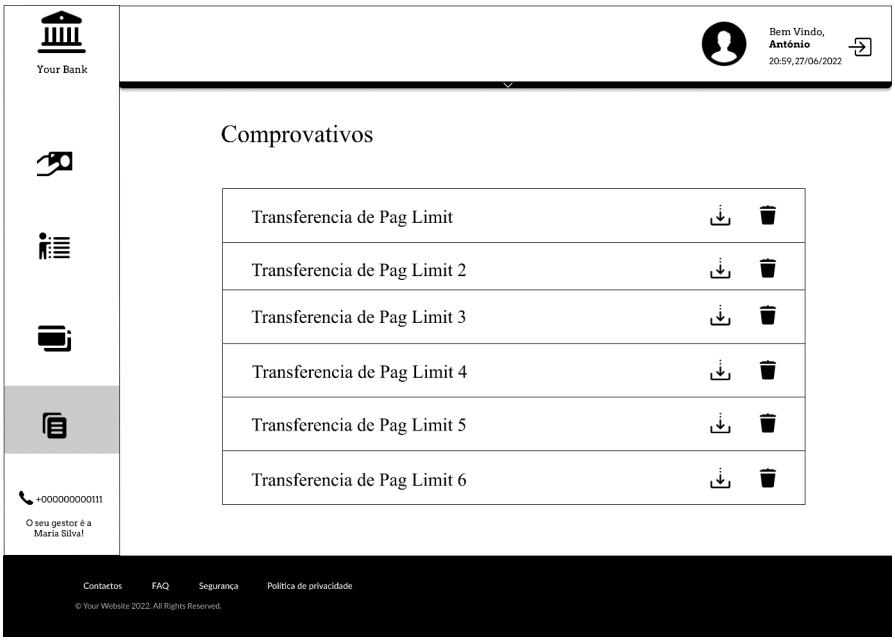

Figura 23 - *Mockup* que representa a funcionalidade de todos os comprovativos.

- Página de Detalhe do Capital – é apresentado uma visão em detalhe do capital que o cliente tem desde o seu saldo disponível, aos juros ou outros tipos de património que a ele esteja associado.

  Contudo, aqui foi construído, uma *box* pelo qual, esta leva dentro uma *label* com esse título de detalhe, figura C.40, Apêndice C e ainda, um conjunto de *cards* que têm o propósito de efetivamente mostrar esse detalhe.

  De notar que, estes *cards* agregam um ícone que se traduz numa informação que é facultada ao cliente de forma a representar que tipo de saldo é aquele. Depois, o cliente ao clicar no *button* "Voltar" retorna à página Capital.

5.3 Subida do Código para o ambiente

Após a concretização dos componentes que ia fazendo e de verificar que estava tudo alinhado com a análise feita à *User Storie*, tinha de subir o meu código.

Ora, para satisfazer esta condição a equipa tinha três *branches* de diferentes ambientes: *dev* (desenvolvimento), *qa* (qualidade) e produção.

Assim, como já tinha criado a minha *branch* baseada no ambiente de desenvolvimento, os passos foram:

- Executei *pull*[10] após criar a *branch*[11]*;*
- Fiz *checkout* para a nova *branch*, usando o comando "*git checkout feature/numeroDaUserStorie-nomeComponente*";
- Subi o código, utilizando o comando "*git commit -a -m "message""*, a *message* é uma referência do que desenvolvi no código de forma abreviada;
- Voltei a fazer *checkou*t para a *branch* de desenvolvimento, "*git checkout dev/realese-numeroRelease*" onde o número da *release* é diferente face aos artefactos[12] criados;
- Dei *update* a *branch* fazendo "*git pull*";
- Fiz *merge*[13] da *branch* com a *branch* de desenvolvimento atual, utilizando o comando "*git merge feature/ numeroDaUserStorie-nomeComponente*";
- Por último, fiz o comando "*git push*" para enviar o código para a *branch* de trabalho.

Posto isto, era necessário criar um *Pull Request* (PR), figura 24, onde eu colocava a descrição, inseria o número da *User Storie* e esperava que o mesmo fosse aprovado pela equipa ou então por um elemento mais sénior.

[10] *Pull* é o contrário do comando *Push*, isto é, envia o repositório local para um repositório remoto.
[11] *Branch* é o nome dado a uma ramificação do projeto.
[12] Sempre que se ia subindo o código ao ambiente de qualidade o número da *release* alterava.
[13] O *merge* une os *branchs* aos *commits*.

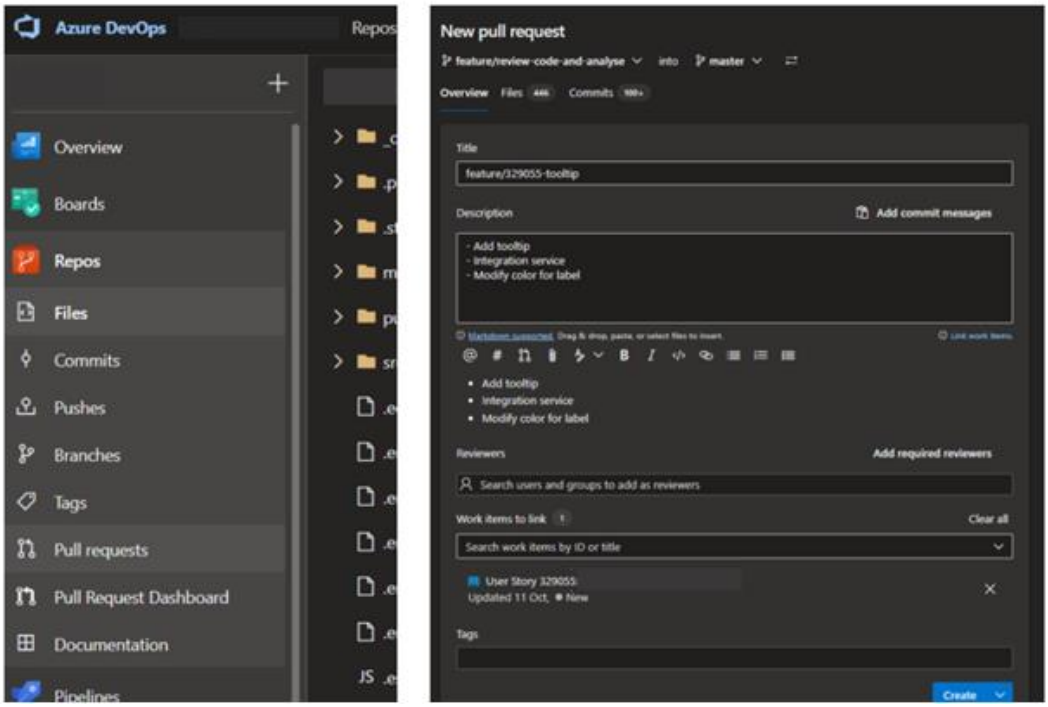

Figura 24 – Representação no *Azure DevOps* da criação do *Pull Request* e de seguida mostra a criação para o mesmo.

5.4 Bugs

Na reta final da implementação fiz alguns testes funcionais com recurso ao React Testing Library, Enzyme e Jest e que vieram a originar os *bugs*.

Portanto o *bug* é, nada mais nada menos, do que um comportamento estranho que certa funcionalidade ou componente está a ter e não é esperado como resultado. E isso acontece por várias razões, como por exemplo os erros cometidos no desenvolvimento do código. De facto, também acabam por surgir face a mudanças ou melhorias que são pedidas ao longo do desenvolvimento da *User Storie.*

Sendo assim, seguem alguns bugs que resolvi na aplicação *web*:

a. Componente "nota" – este *bug* apareceu na secção onde se podia adicionar notas aos movimentos e o problema que se teve foi uma opacidade que impedia de ver a *tooltip* à volta do ícone em forma de balão. Ora como era um componente interno do serviço bancário este já tinha um *opacity* que faria o que se queria e o ícone como estava a ser envolvido dentro de um <*div*/> para lhe dar uma opacidade ele

não estava a responder de forma correta. Deste modo, percebeu-se que não era a melhor implementação que se estava a seguir e facilmente se resolveu retirando esse <*div*/> do seu redor.

b. *Card* dos movimentos no *Home Page* – para a resolução deste erro foi perceber a razão dos últimos movimentos não estarem a aparecer para serem mostrados ao cliente. Assim, ao analisar o código e testando com os serviços ligados repara-se que estava a existir uma falha do lado interno do banco junto com o BE do serviço bancário e eles com um *update* ao serviço facilmente se deu por fechado o *bug*.

c. *Header* da aplicação *web* – neste *bug* não existiu mesmo qualquer tipo de erro de sistema, mas sim melhoria para que se encaixasse um ícone de *Home* e isso fez com que após esta melhoria existem reaberturas sucessivas de *bugs* face ao *design* visual e impacto que teria para o cliente e também no que diz respeito à qualidade do serviço prestado para com o serviço bancário. Todavia, se solucionou alterando no código fonte todos os estilos de todas as áreas envolventes.

d. Ícone dos bancos – a questão face a este *bug* era perceber qual a razão pelo qual o mesmo não estava a contribuir com os ícones dos bancos que cada conta possuía , ou seja, primeiro ficava mais difícil de entender que conta era aquela e também, não era o design esperado. Ora, uma vez mais isto deveu-se a uma atualização dos serviços internos do banco e também a alterações de componentes internas que sofreu uma alteração no *crossOrigin* e isso afetou a aplicação nas áreas que tinham os respetivos logos de cada banco. Desta forma, alterando e fazendo um *update* aos serviços, o *bug* ficou resolvido com sucesso.

e. Cancelamento da criação do PDF – no caso deste bug, quando se clicava no botão cancelar PDF, este não fazia o cancelamento. Ora, aqui o problema estava no código fonte que não estava com o valor correto na ação do *onPress*. Uma vez mais a resolução foi agilizada e solucionada.

f. Campo de pesquisa – no *bug* aberto face a esta pesquisa o que se tinha era que quando se fazia uma pesquisa e não se tinha sucesso da mesma, e logo a seguir se limpava o *backSpace*, as contas, por exemplo, não eram mostradas. Pois bem, ao analisar este *bug* a resolução que o mesmo teve foi que estava a faltar, da parte interna dos componentes do serviço bancário, um *update* ao código fonte deles e após esta nova versão do lado deles foi possível atualizar a pesquisa com o

backSpace e ainda ter a possibilidade de acionar essa mesma pesquisa através da tecla "*ENTER*".

g. Limite de caracteres no campo nota – face a este *bug* tem-se o não limite dos caracteres que estava descrito na US para um limite de 66 palavras e isto deveu-se á má análise de um colega, o que influenciou para a abertura de um erro. Desta forma, como foi algo fácil de detetar, bastou ir ao componente que tinha este ícone da nota e colocar um *maxLenght* cujo valor é 66, e isto permite que o cliente ao escrever a nota e/ou comentário esteja limitado a este número de caracteres.

h. Fontes diferentes em elementos similares – relativamente a este *bug* o que está devidamente incorreto é a incoerência que existia em componentes iguais na mesma área com fontes de tamanho ou estilo diferentes e de facto na análise a estes não era assim que estava definido. Para resolução deste erro, uma vez mais, foi-se ao código fonte criado pelo *developer* e alterando os valores do *fontWeight*, *fontFamily* e *fontSize* o bug ficou corrigido.

i. Linhas da tabela – face a este erro o problema que estava a acontecer era que uma vez mais sendo um componente interno do banco, a tabela, não estava atualizada face aquilo que o cliente pretendia e desta forma as linhas da mesma apareciam da cor que se pretendia. Como os colegas da equipa interna estavam à espera da validação da parte UX (*User Experience*) para alterar o estilo de cor foi aberto um *bug* para ficar documentado, e numa fase final, ficou resolvido apenas mudando a versão que se estava a utilizar no "*package.json*".

j. *Skeleton*[14] – no erro que se segue o problema era de não se ter o *skeleton* numa das áreas e estar incoerente com o resto da *app* e ainda, o carregamento da página ser superior a um segundo. Portanto, para se resolver o mesmo, no código implementado faltava um *width* que teria de ter para renderizar neste caso o componente da tabela exatamente igual, bem como as *rows* envolventes, mas, por despiste verifica-se que a falta de serviços internos do banco não permitia que este mesmo *skeleton* ficasse estável, o que prejudicava o efeito de a página estar a ser carregada e o utilizador ter de aguardar breves segundos.

[14] *Skeleton* é um conjunto de elementos de espaço reservado que o utilizador vê antes que o conteúdo real seja carregado numa página *web*. Pode ser apresentado de duas formas ou por marcadores de cores ou por caixas cinzas.

Capítulo 6

Conclusão

A realização deste trabalho permitiu que o conhecimento ao nível da linguagem React JS, JavaScript, Redux Saga, bem como outras ferramentas/linguagens ficasse mais vincado e à altura de eu conseguir ser mais eficiente ao lidar com *bugs* e também, conseguir ultrapassar barreiras pessoais como sendo mais autodidata e, ter a capacidade de ajudar elementos da equipa que pudessem ter dúvidas e/ou problemas em determinada situação e, assim, eu conseguir corresponder nesse sentido.

No entanto, na vanguarda deste trabalho existiu uma série de complexidades que foram sendo ultrapassadas com o apoio da equipa e, sobretudo, com muita pesquisa e, de facto, o que mais veio dificultar este contratempo e tornar ainda maior a sua exigência foi lidar com elementos nunca trabalhados, como o caso de serviços criados por outrem.

Todavia, como a equipa em si estava bem organizada e todos cooperaram com a mesma essência, este problema tornou-se um objetivo cumprido e, pelo qual, uma vez mais com a recolha de dados para perceber melhor todos estes conceitos, veio facilitar o caminho para atingir o percurso pretendido que consistia em ter uma aplicação *web* funcional e cujo público alvo são todos os utilizadores e/ou clientes de um certo serviço bancário que através desta, facilitava o seu dia-a-dia e resolveria dezenas de contrariedades.

Eventualmente, todo este percurso ficou com melhorias a serem realizadas num futuro próximo, tais como um novo *refactor*[15] ao *site* e assim ter-se a melhoria do mesmo, seja visualmente, como funcionalmente. Mas, é um trabalho que se tem gradualmente e com vários pontos de situação feitos diariamente com a equipa e o cliente, de modo que tudo flua no caminho certo.

[15] A ideia do "*refactor*" é a reestrutura do código ao longo do tempo e a mudança de certas funcionalidades que as antigas US tinham e no momento atual deixaram de ter o impacto esperado e assim, sofreram melhoria.

De notar que, o esforço foi contínuo, mas o trabalho árduo e a capacidade de querer conquistar, como o caso, de mais um componente para a aplicação, ou mudar a organização estrutural do código, ou opinar sobre determinado tema que envolvesse a melhoria da *app web*, esteve sempre realçado e colocado em primeiro lugar para que a ideia de negócio fosse efetivamente concretizada e desenvolvida com qualidade e excelência.

De salientar ainda, que todos os objetivos traçados inicialmente foram conseguidos.

Em suma, todo este tratamento que o *site* teve deste serviço bancário foi sempre realizado em prol daquilo que o cliente se sentia mais confortável e também, foram cumpridos os requisitos definidos pelo *Product Owner*, e o que se tinha idealizado para a sua exposição no mercado e assegurando uma oferta diferenciadora.

Bibliografia

[1] Adobe Communications Team, 2022, Waterfall Methodology. Acedido a 15 de Abril 2022, em: https://business.adobe.com/blog/basics/waterfall

[2] Jasmin, 2022, Agile. Acedido a 15 de Abril 2022, em: https://www.jasminsoftware.pt/blog/metodologia-agile/

[3] Digité, 2022, What is Scrum? Acedido a 15 de Abril 2022, em: https://www.digite.com/agile/scrum-methodology/

[4] Conta Corrente, Maio de 2017, Internet Banking. Acedido a 30 de Setembro 2021, em: https://www.conta-corrente.com/duvidas/internet-banking/

[5] Wikipédia, Agosto de 2021, React. Acedido a 21 de Novembro 2021, em: https://pt.wikipedia.org/wiki/React_(JavaScript)

[6] Henrique Freire, Agosto de 2019, React – Vantagens e Desvantagens. Acedido a 21 de Novembro 2021, em: https://henrique-freire.medium.com/react-vue-angular-conhe%C3%A7a-suas-vantagens-e-desvantagens-e-qual-%C3%A9-melhor-para-seus-projetos-53734bb3d37f

[7] React, 2021, React Hooks. Acedido a 21 de Novembro 2021, em: https://reactjs.org/docs/hooks-intro.html

[8] DevMedia, 2019, React Hooks. Acedido a 21 de Novembro 2021, em: https://www.devmedia.com.br/react-hoje-e-amanha-o-que-muda-com-os-hooks/40314

[9] React Training, 2021, React Router. Acedido a 1 de Dezembro 2021, em: https://reactrouter.com/

[10] GeeksforGeeks, Junho de 2020, React Router. Acedido a 1 de Dezembro 2021, em: https://www.geeksforgeeks.org/reactjs-router/

[11] Platzi, 2020, React Router. Acedido a 1 de Dezembro 2021, em: https://platzi.com.br/blog/3-dicas-importantes-sobre-react-router/

[12] React-Redux, 2021. Acedido a 14 de Dezembro 2021, em: https://react-redux.js.org/api/hooks

[13] MDN Web Docs, 2021, CSS. Acedido a 14 de Dezembro 2021, em: https://developer.mozilla.org/pt-BR/docs/Web/CSS

[14] Emotion, 2021, Emotion. Acedido a 14 de Dezembro 2021, em: https://emotion.sh/docs/introduction

[15] TypeScript, 2021, TypeScript. Acedido a 22 de Dezembro 2021, em: https://www.typescriptlang.org/

[16] Desenvolvimento web, 2020. Acedido a 22 de Dezembro 2021, em: https://blog.betrybe.com/desenvolvimento-web/typescript/

[17] Tecnoblog, Março de 2021. Acedido a 22 de Dezembro 2021, em: https://tecnoblog.net/426754/o-que-e-typescript-guia-para-iniciantes/

[18] Redux, 2021. Acedido a 27 de Dezembro 2021, em: https://redux.js.org/

[19] Remessa Online, Setembro de 2021, Redux: para que serve a biblioteca e como usar seus recursos ? Acedido a 27 de Dezembro 2021, em: https://www.remessaonline.com.br/blog/redux-para-que-serve-a-biblioteca-e-como-usar-seus-recursos/

[20] LoginRadius, Novembro de 2020, Introdução ao Redux Saga. Acedido a 27 de Dezembro 2021, em: https://www.loginradius.com/blog/async/introduction-to-redux-saga/

[21] Redux-Saga, 2021. Acedido a 27 de Dezembro 2021, em: https://redux-saga.js.org/

[22] Testing Library, 2021. Acedido a 27 de Janeiro 2022, em: https://testing-library.com/docs/react-testing-library/intro

[23] React, 2021. Acedido a 21 de Novembro 2021, em: https://reactjs.org/docs/testing.html

[24] Academind, Yousaf Khan, Agosto de 2020. Acedido a 21 de Novembro 2021, em: https://academind.com/tutorials/testing-react-apps

[25] JEST, 2021, Jest. Acedido a 8 de Fevereiro 2022, em: https://jestjs.io/

[26] Morioh, Fevereiro de 2021, Top 3 JavaScript Testing Frameworks. Acedido a 8 de Fevereiro 2022, em:

https://www.rlogical.com/blog/top-3-javascript-testing-frameworks-with-their-pros-and-cons/?ref=morioh.com&utm_source=morioh.com

[27] Code, Dominic Fraser, 2018, Jest and Enzyme. Acedido a 8 de Fevereiro 2022, em: https://medium.com/codeclan/testing-react-with-jest-and-enzyme-20505fec4675

[28] Enzyme, 2021. Acedido a 8 de Fevereiro 2022, em: https://enzymejs.github.io/enzyme/

[29] SmartBear Software, 2021, Swagger. Acedido a 28 de Março 2022, em: https://swagger.io/

[30] Aristoteles Lopes, Setembro de 2019, Porque usar Swagger na sua Api? Acedido a 28 de Março 2022, em: https://medium.com/@arikardnoir/porque-usar-swagger-na-sua-api-e80c4ed15190

[31] Terralab, 2020, Api REST com Swagger. Acedido a 28 de Março 2022, em: http://www2.decom.ufop.br/terralab/documentando-sua-api-rest-com-swagger/

[32] Babel, 2021. Acedido a 28 de Dezembro 2021, em: https://babeljs.io/docs/en/

[33] BlogImpacta, 2018, Vantagens do Visual Studio Code. Acedido a 6 de Outubro 2021, em: https://www.impacta.com.br/blog/2018/02/16/quais-vantagens-usar-visual-studio-desenvolver-aplicativos/

[34] ByLearn, Abril de 2019, Visual Studio Code. Acedido a 6 de Outubro 2021, em: https://medium.com/@bylearn/11-motivos-para-migrar-para-o-vs-code-5b9574a057f5

[35] Capterra, Paulo S., Maio de 2020, Figma. Acedido a 4 de Abril 2022, em: https://www.capterra.pt/reviews/175027/figma

[36] Wikipédia, Julho de 2021, Figma. Acedido a 4 de Abril 2022, em: https://pt.wikipedia.org/wiki/Figma

[37] Azure DevOps, 2021. Acedido a 17 de Abril 2022, em: https://azure.microsoft.com/pt-pt/services/devops/

[38] Microsoft, 2021, DevOps. Acedido a 17 de Abril 2022, em: https://docs.microsoft.com/pt-br/dotnet/architecture/cloud-native/devops

[39] Git, 2022. Acedido a 6 de Julho 2022, em: https://git-scm.com/book/pt-br/v2/Come%C3%A7ando-O-B%C3%A1sico-do-Git

[40] Bruno Kriger, 18 de Outubro de 2022, O que é o Git: conceitos, principais comandos e quais as vantagens? Acedido a 20 de Outubro 2022, em: https://kenzie.com.br/blog/o-que-e-git/

[41] Astah UML, 2021. Acedido a 8 de Abril 2022, em: https://astah.net/products/astah-uml/uml-diagrams/

[42] Miro, 2022. Acedido a 17 de Fevereiro 2022, em: https://miro.com/pt/

[43] Google, 2022. Acedido a 15 de Outubro de 2022, em: http://pplware.sapo.pt/wp-content/uploads/2005/04/cgd-720x291.jpg , https://www.jornalcontabil.com.br/wp-content/uploads/2019/04/novo-logo-santander-fundo-vermelho-696x365.jpg , https://pt.teamlyzer.com/static/uploads/companies/552x300/banco-bpi.png , http://s3-eu-central-1.amazonaws.com/cja-blogassets/wp-content/uploads/sites/3/2016/09/27170503/ca.png

[44] LindekIn, 2022, Waterfall. Acedido a 15 de Outubro de 2022, em: https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.linkedin.com%2Fpulse%2Fpopular-project-management-methodologies-waterfall-purohit%3Ftrk%3Dpulse-article&psig=AOvVaw3_yfxo5sGZtk2vl5IOhtty&ust=1667339083061000&source=images&cd=vfe&ved=0CA0QjRxqFwoTCJjB2KC4i_sCFQAAAAAdAAAAABAD

[45] Tecsinapse, 2017. Typescript. Acedido a 15 de Outubro de 2022, em: https://miro.medium.com/max/445/1*zZdruqvx5FsjnxbquOgwRQ.png

[46] Latine Group, 2022. Azure DevOps. Acedido a 15 de Outubro de 2022, em: https://lattinegroup.com/wp-content/uploads/2020/09/azure-devops-1.png

Apêndice A.

Wireframes

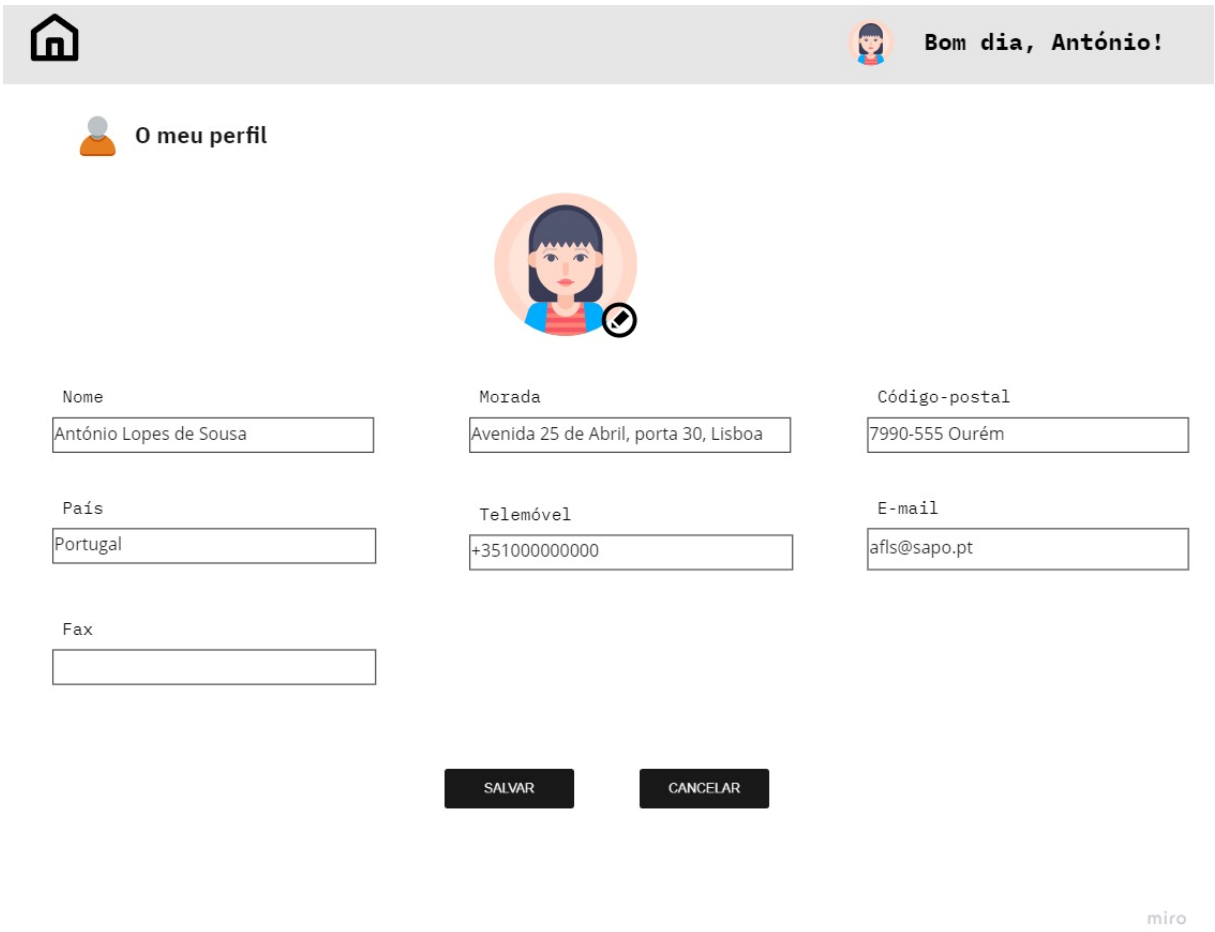

Figura A. 1 – *Wireframe* a representar a edição da conta do utilizador.

Figura A. 2 – *Wireframe* simulador de crédito pessoal de saúde.

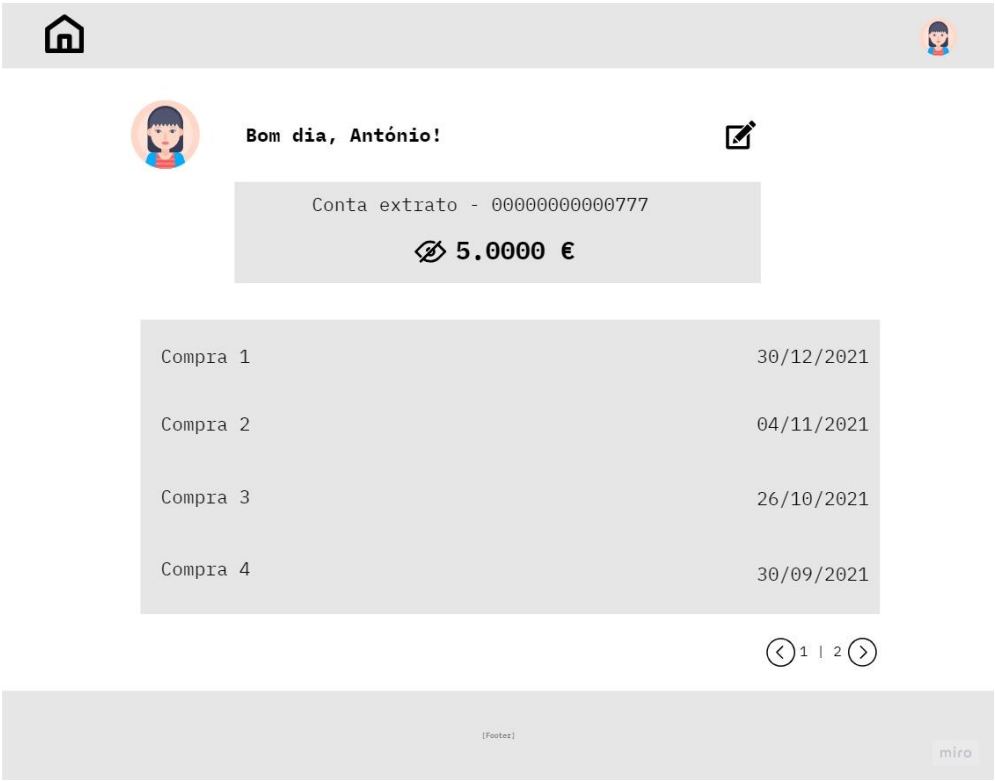

Figura A. 3 – *Wireframe* do dia-a-dia de um cliente com *login* efetuado.

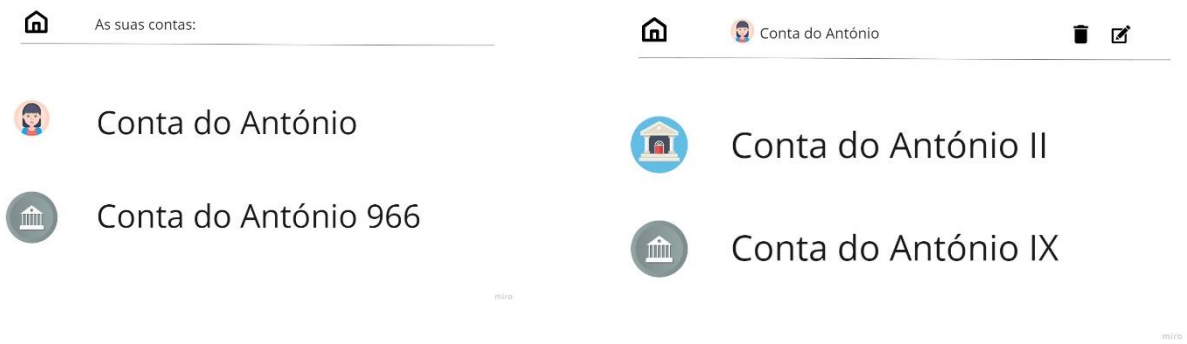

Figura A. 4 – *Wireframe* que representa o ecrã com as contas e a conta que está selecionada, mas com o modo de editar e remover, respetivamente.

Figura A. 5 – *Wireframe* que representa o componente *modal* para eliminar uma certa conta.

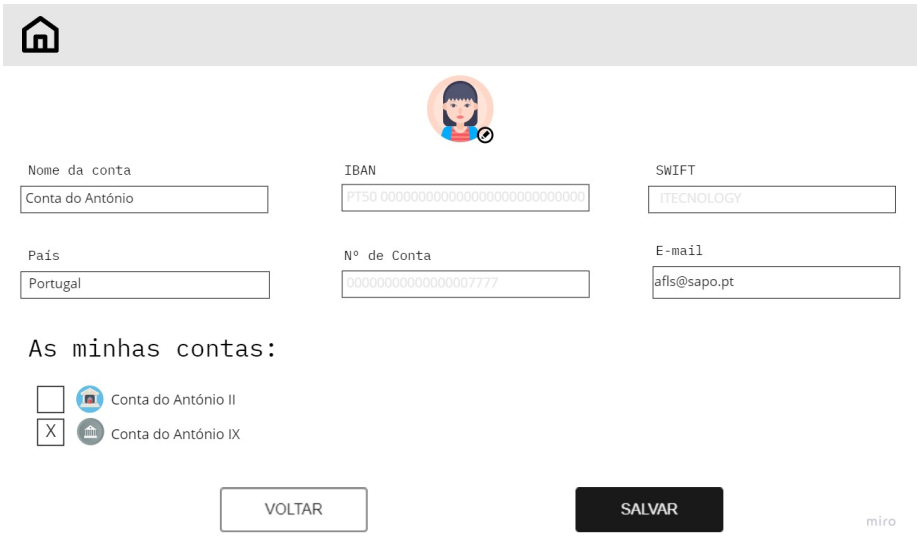

Figura A. 6 – *Wireframe* a representar a edição da conta e ainda pode eliminar contas ou cartões que tenha associados.

Figura A. 7 – *Wireframe* a representar a remoção da conta.

Figura A. 8 – *Wireframe* que representa o componente com os últimos movimentos, o componente com as condições que o utilizador pode ler e o componente com a estrutura para o resumo de uma certa matéria, respetivamente.

Figura A. 9 – *Wireframe* a representar os movimentos em detalhe da "Conta do António".

Figura A. 10 – *Wireframe* do projeto com as diferentes funcionalidades.

Figura A. 11 – *Wireframe* a representar os consultores existentes e a possibilidade de poder adicionar ainda mais.

Figura A. 12 – *Wireframe* a representar a seleção de um novo consultor.

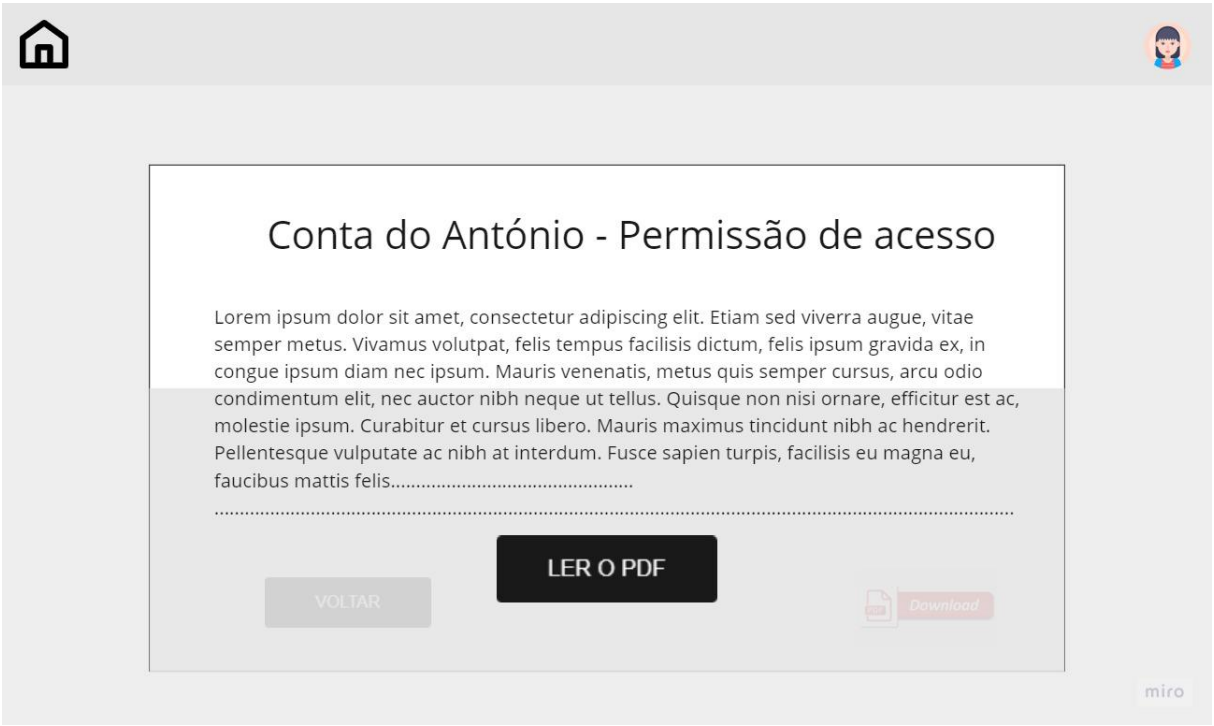

Figura A. 13 – *Wireframe* a mostrar o PDF aquando de um cliente pretende adicionar um consultor e de seguida terá de adicionar o código de acesso para a próxima etapa.

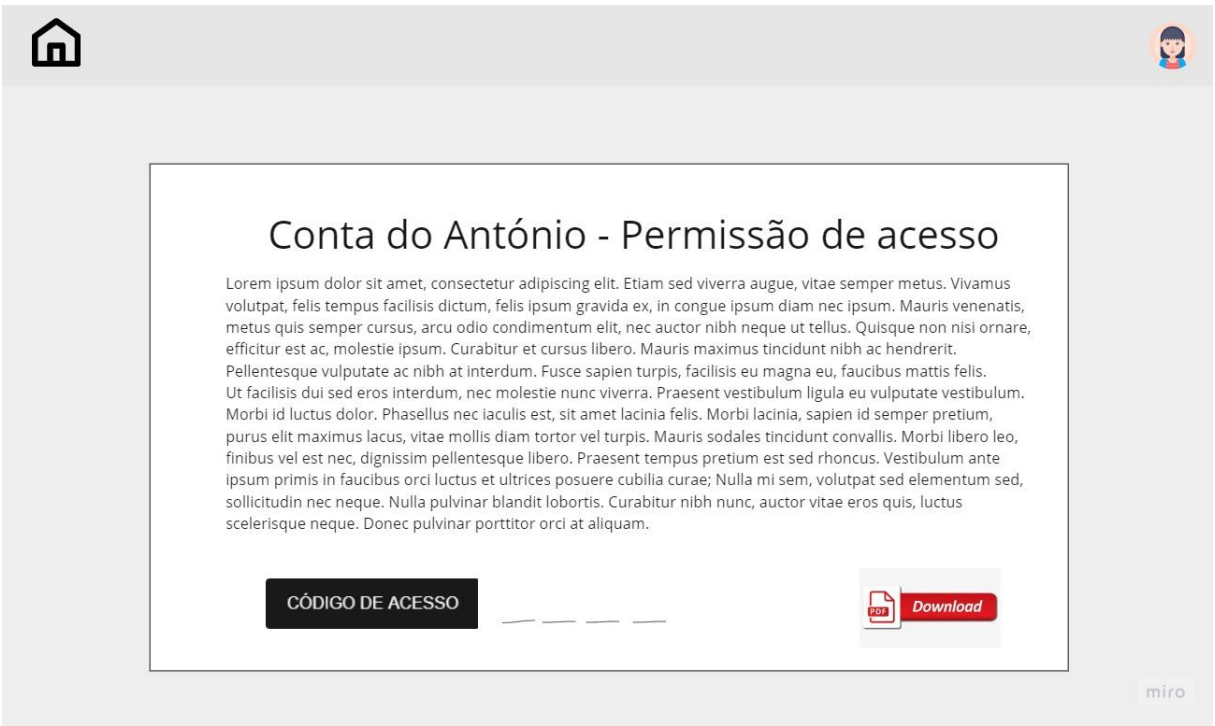

Figura A. 14 – *Wireframe* a representar o PDF aquando de um cliente pretende adicionar um consultor e tem aqui de inserir o código de acesso para avançar.

Figura A. 15 – *Wireframe* que representa a conclusão de adição de um consultor.

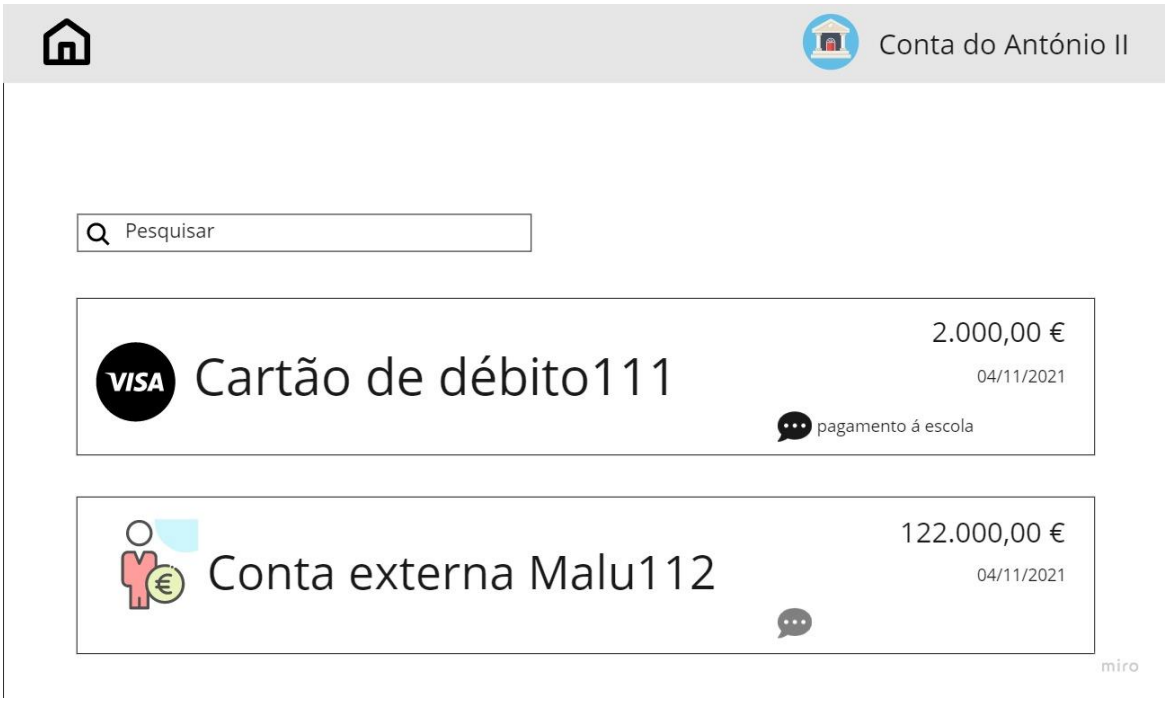

Figura A. 16 – *Wireframe* representativo de um certo cliente que contem uma certa conta e essa mesma conta tem mais duas associadas de diferentes bancos ou não e neste caso tem uma conta e um cartão.

Apêndice B.

Figuras do desenvolvimento e da implementação do projeto

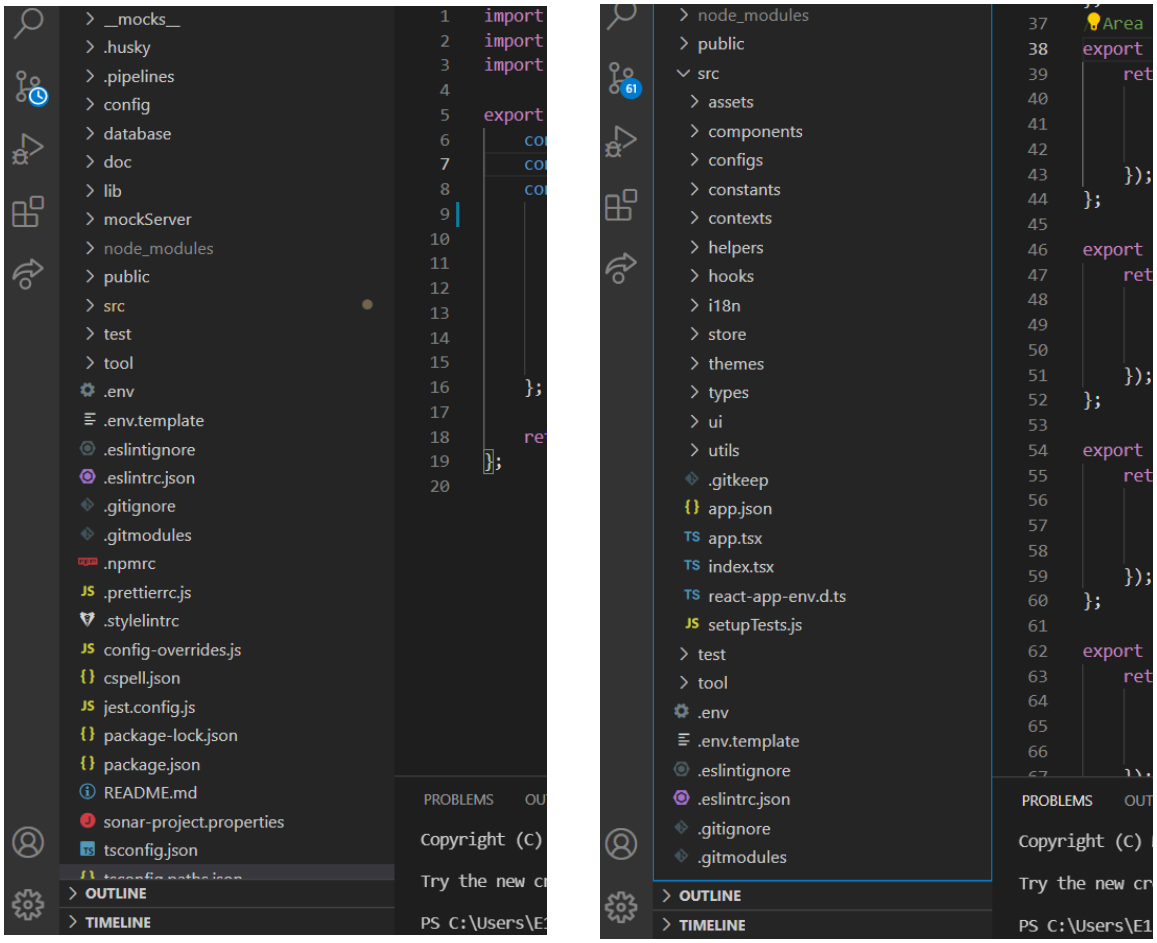

Figura B. 1 – Representa a estrutura inicial para esta implementação e ainda, a representação de como se encontra subdividida a pasta "*src*", respetivamente.

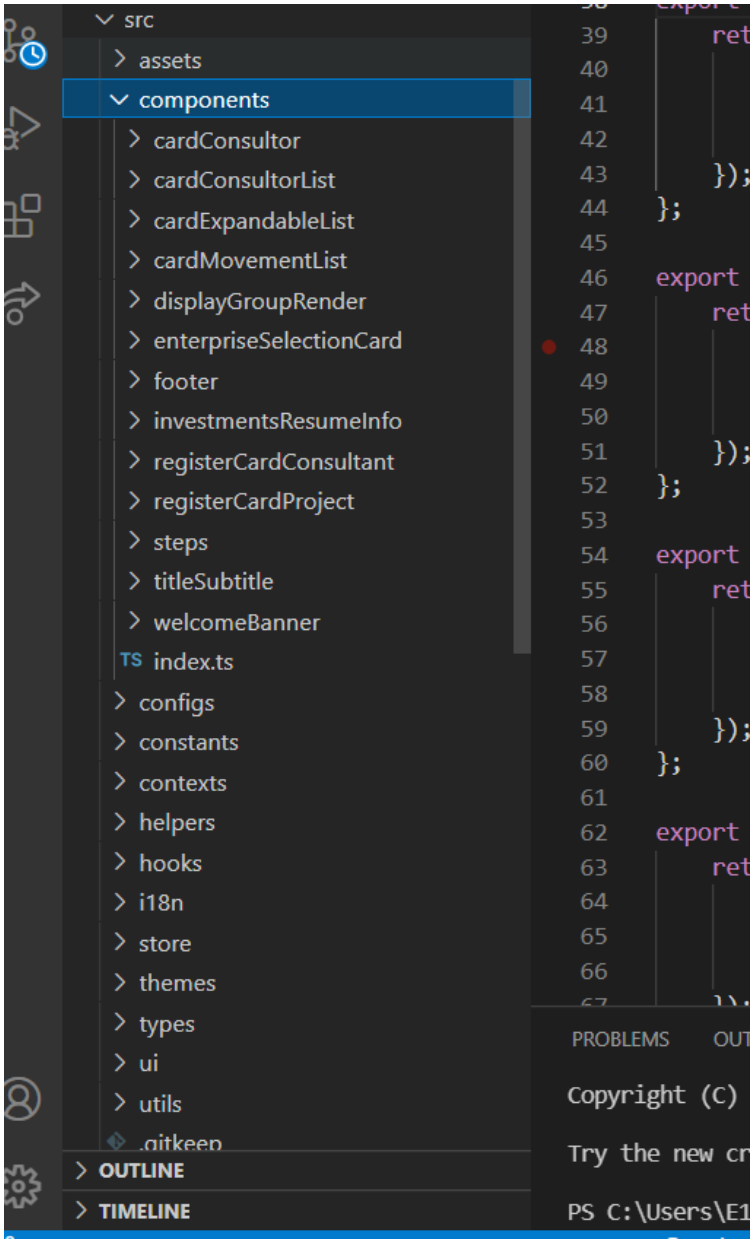

Figura B. 2 – Ilustra a pasta *components* a serem usados em toda a *app*.

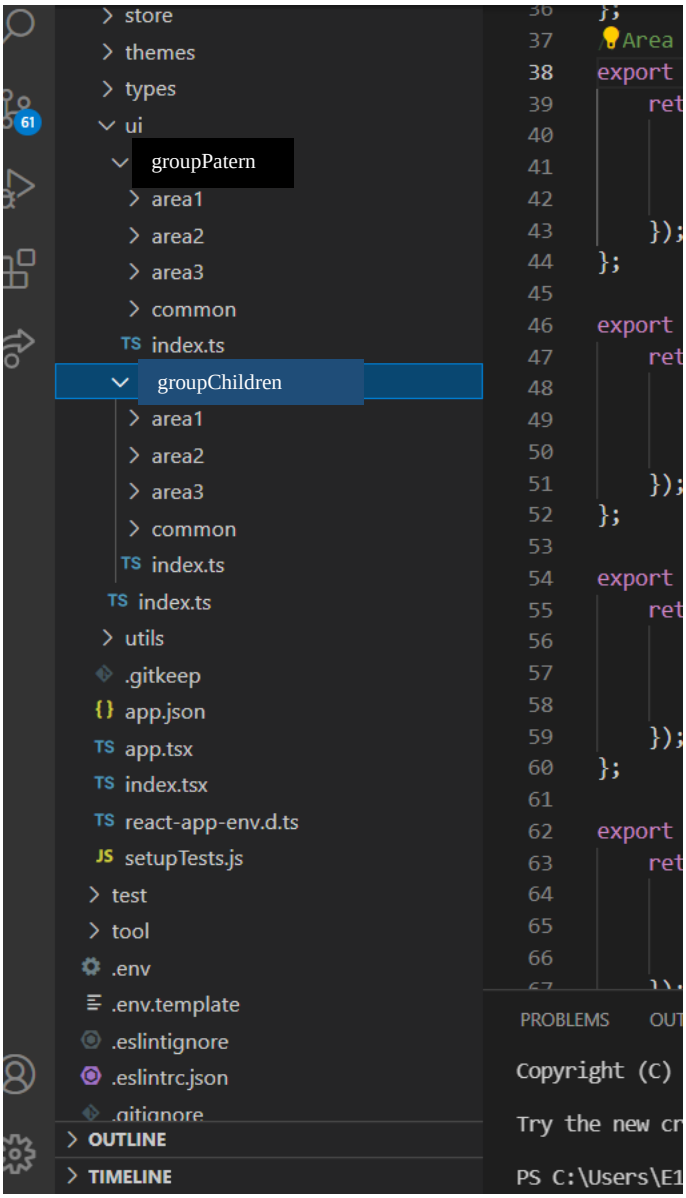

Figura B. 3 – Representa o groupPatern e o groupChildren e com as respetivas áreas que a aplicação *web* terá.

Apêndice C.

Mockups dos testes da aplicação implementada

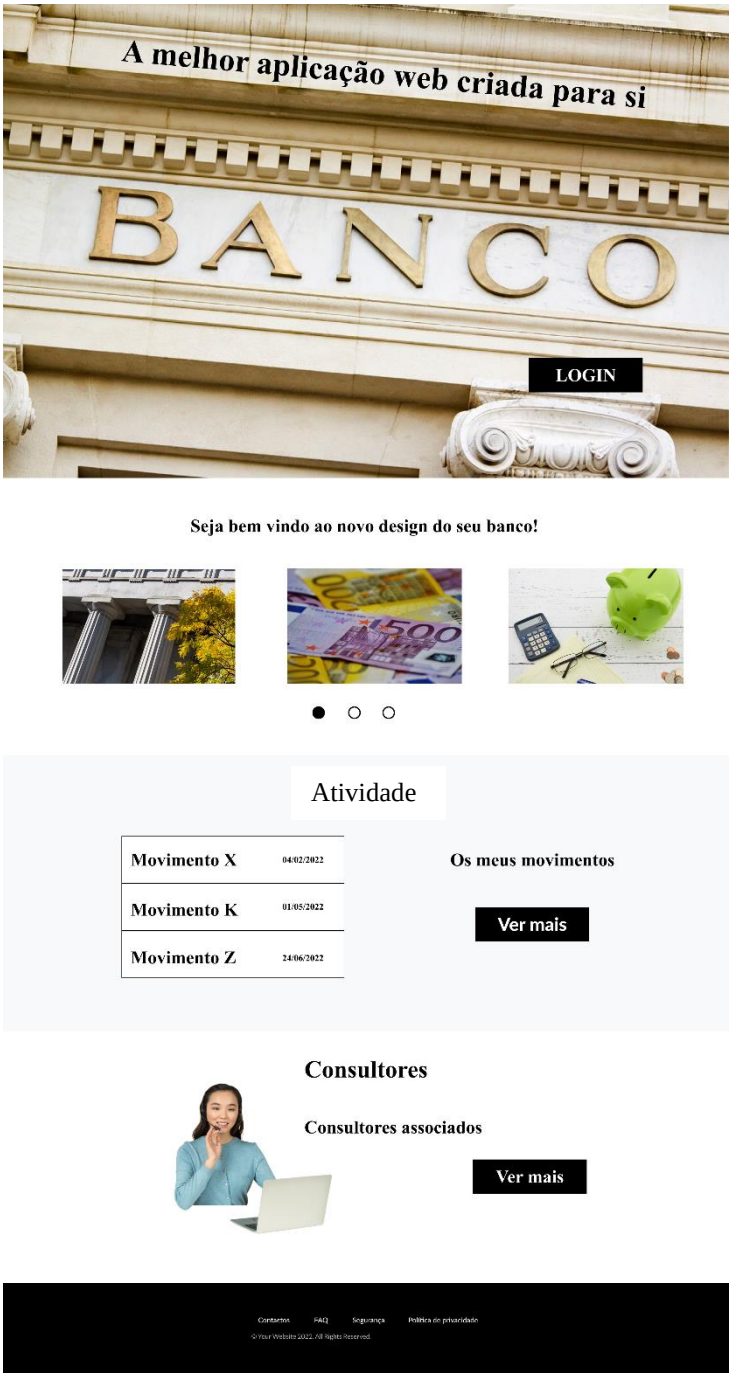

Figura C.1 – *Mockup* que representa a aplicação *web* face á sua entrada inicial no *site*.

Figura C. 2 – *Mockup* que representa a página de entrada do *login* quando bem sucedido.

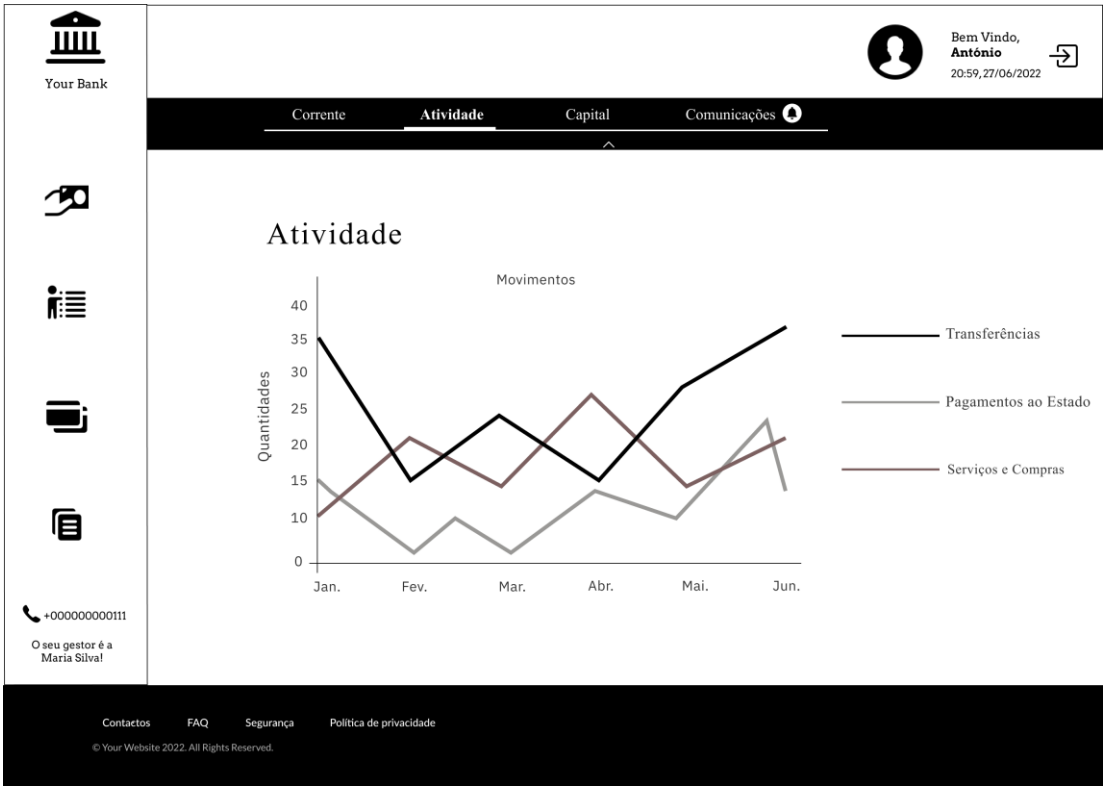

Figura C. 3 – *Mockup* que representa o gráfico relativamente aos movimentos.

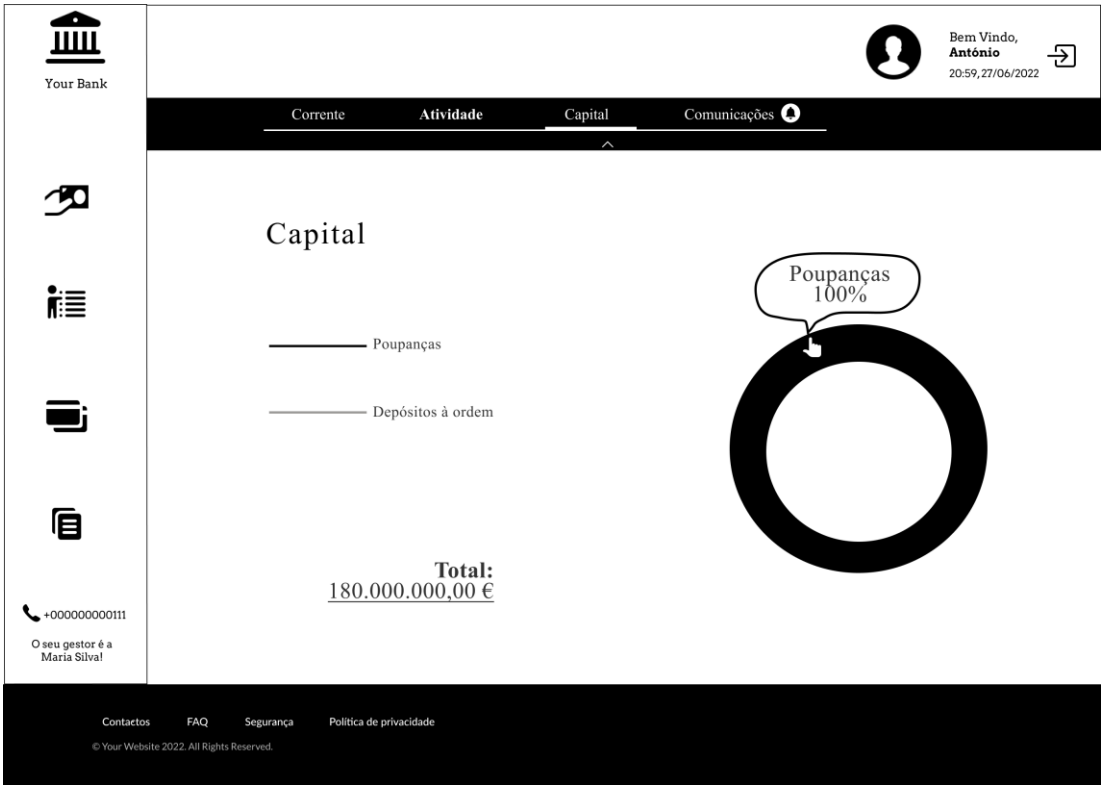

Figura C. 4 – *Mockup* que representa a capital que o cliente tem agregado.

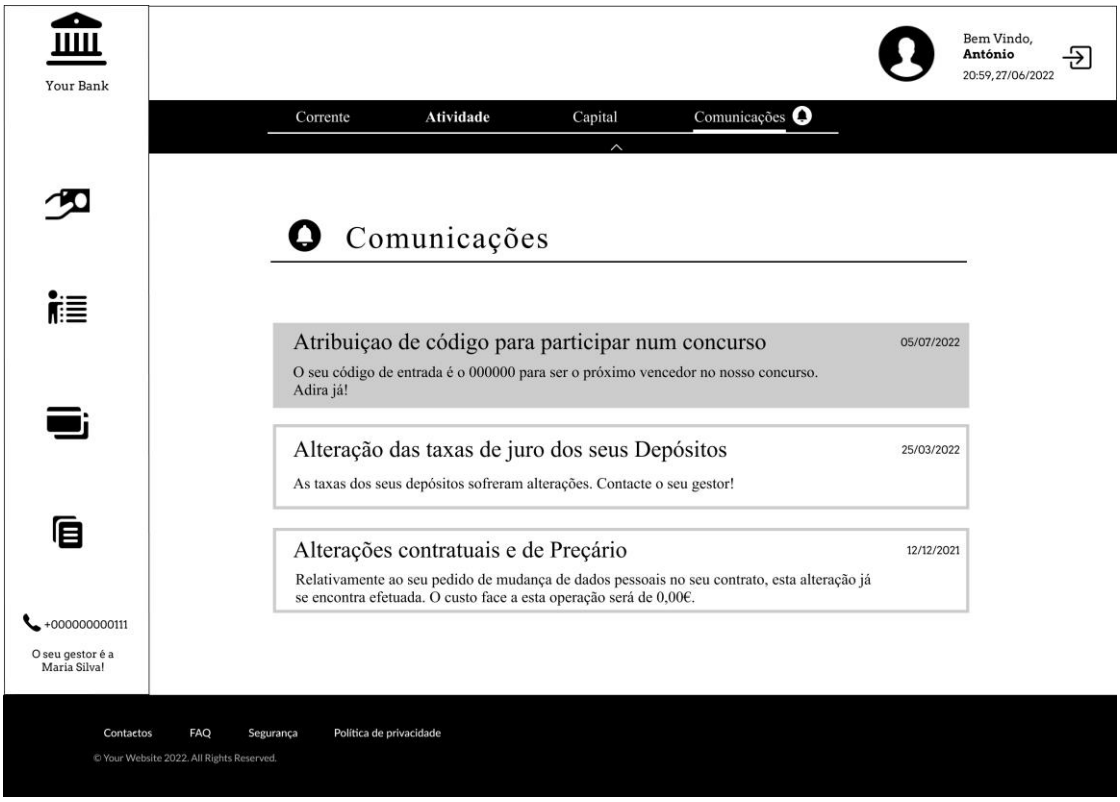

Figura C. 5 – *Mockup* que representa a páginas das comunicações de cada cliente.

Figura C. 6 – *Mockup* que representa a simulação de um certo crédito.

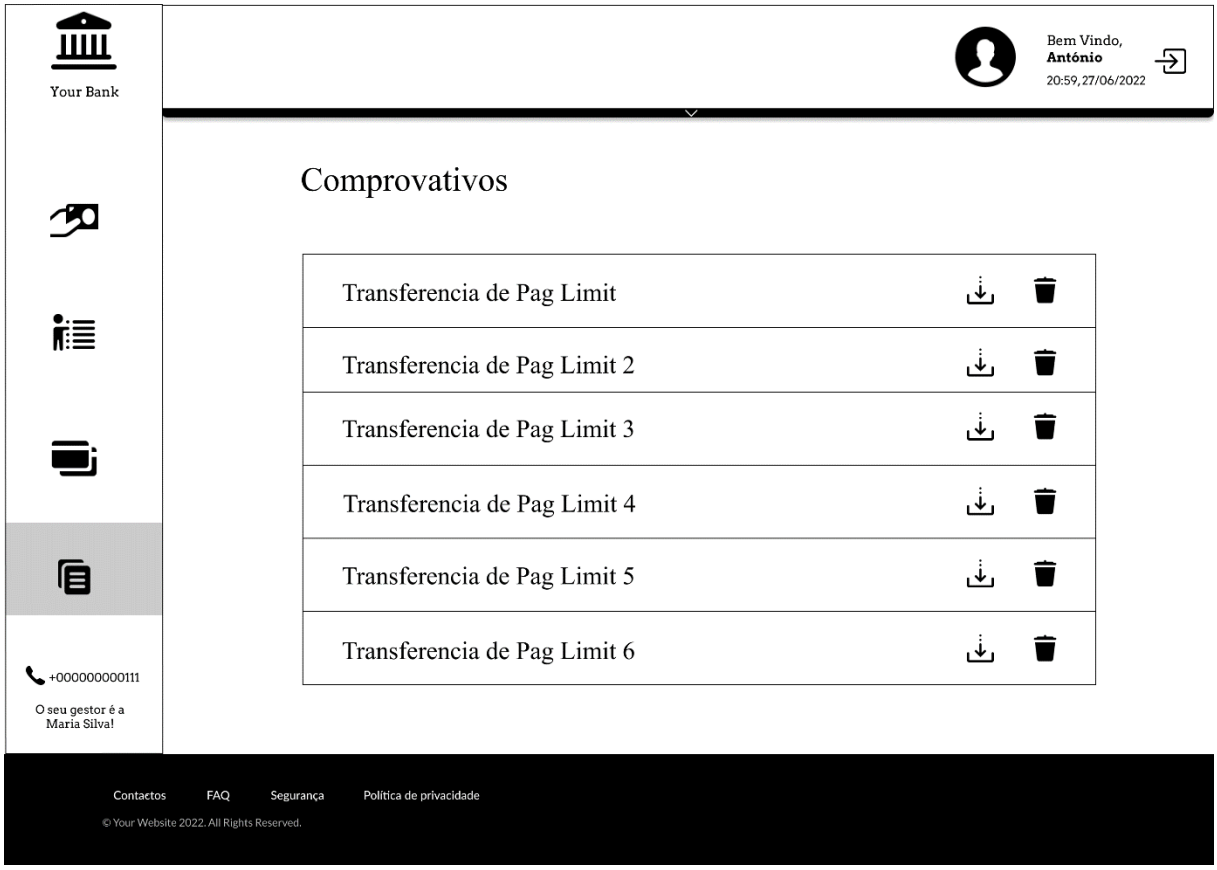

Figura C. 7 – *Mockup* que representa a funcionalidade de todos os comprovativos.

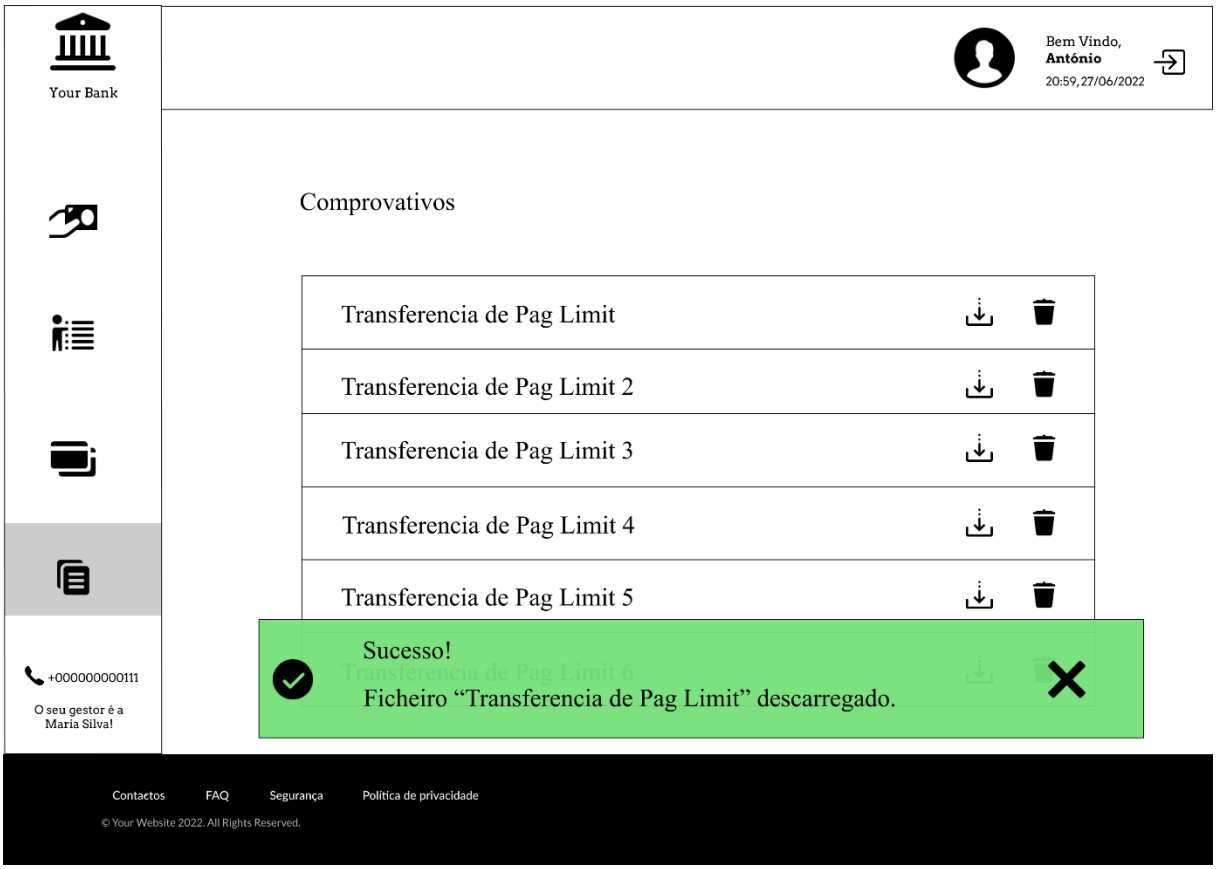

Figura C. 8 – *Mockup* que representa o *download* de um dos comprovativos.

Figura C. 9 – *Mockup* que representa a eliminação de um dos comprovativos.

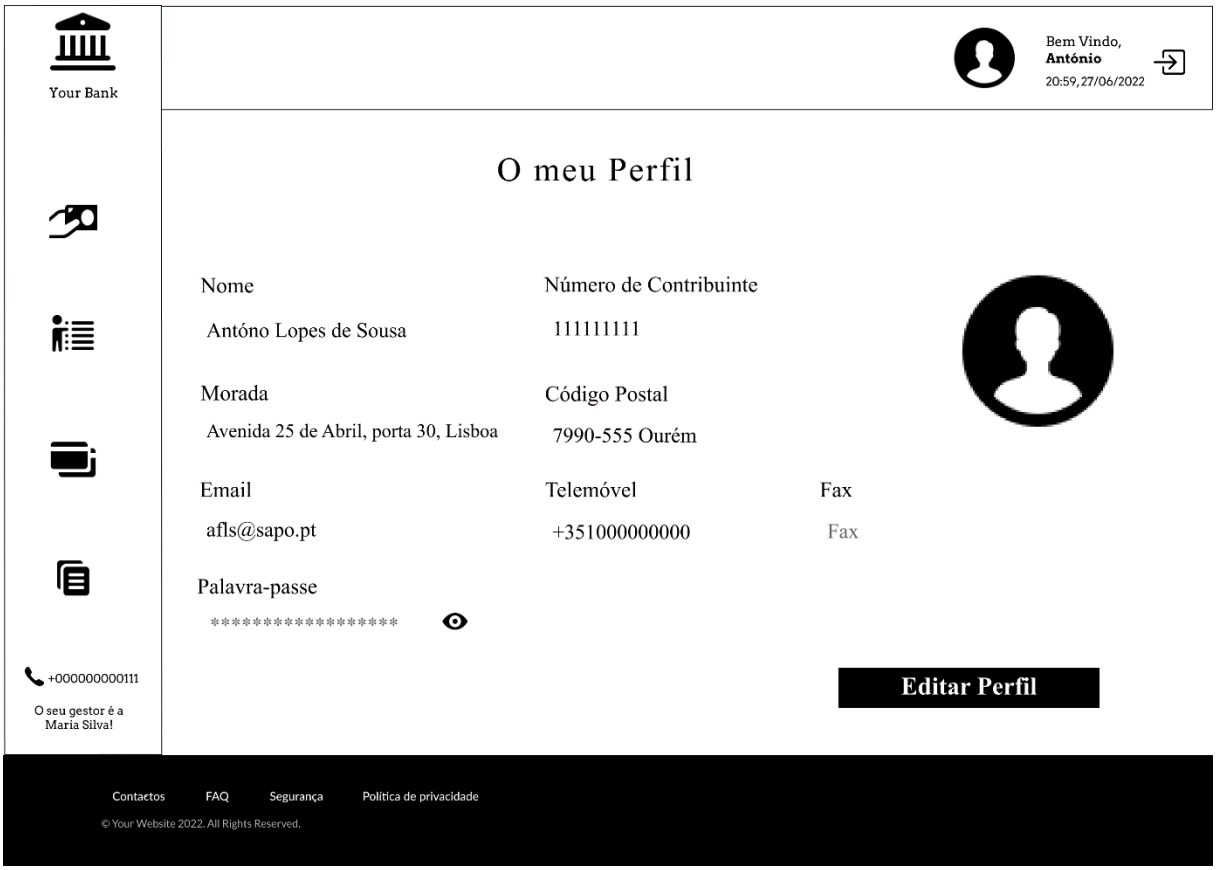

Figura C. 10 – *Mockup* que representa a área de perfil do cliente.

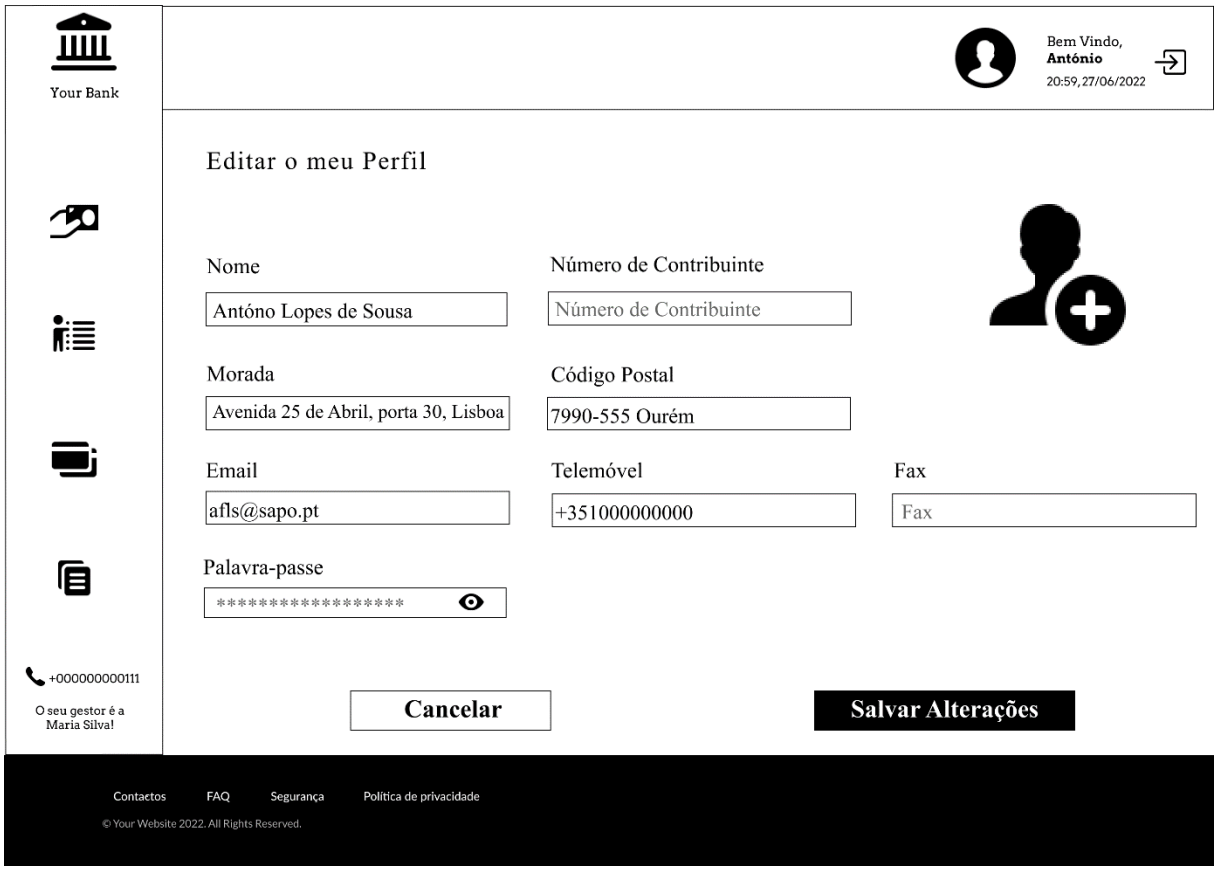

Figura C. 11 – *Mockup* que representa a área de edição de perfil do cliente.

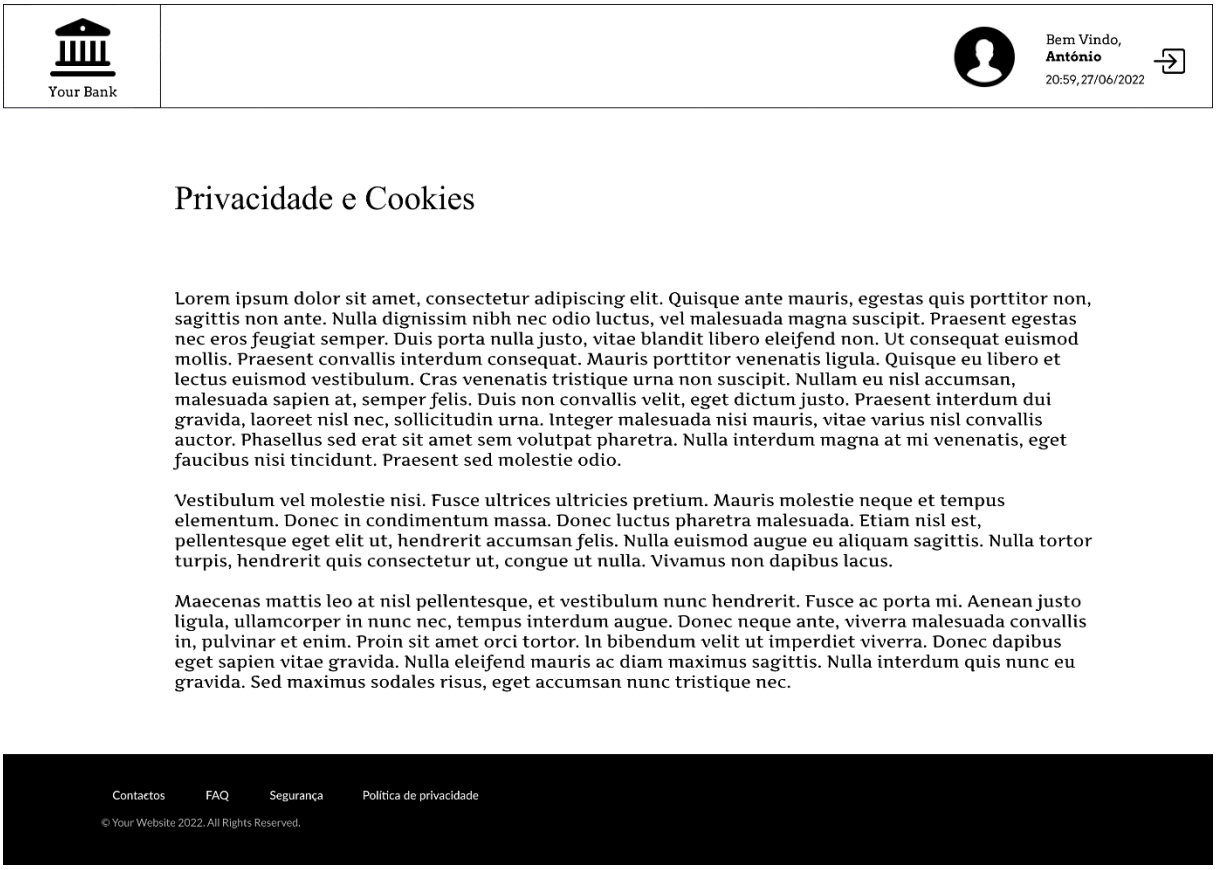

Figura C. 12 – *Mockup* que representa a privacidade e *cookies*.

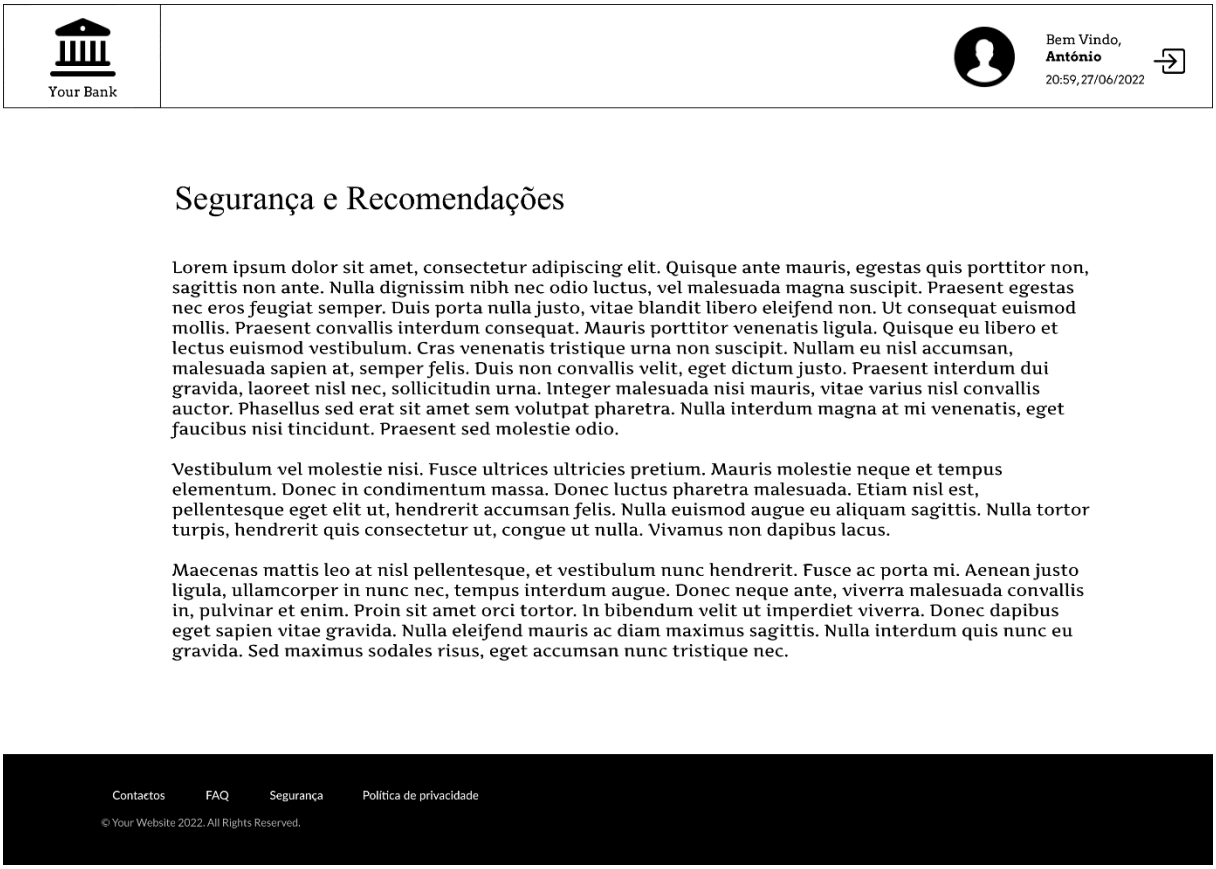

Figura C. 13 – *Mockup* que representa a segurança e recomendações.

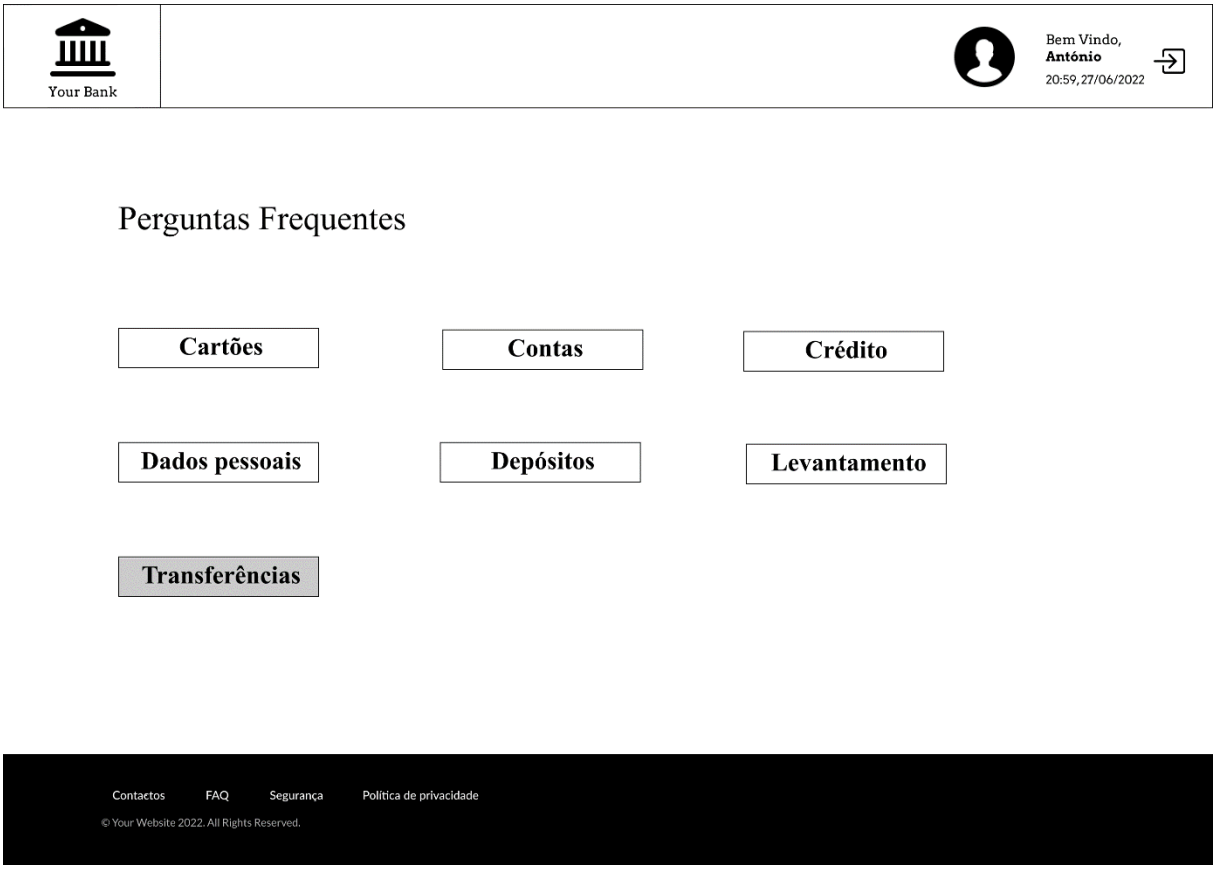

Figura C. 14 – *Mockup* que representa as perguntas frequentes.

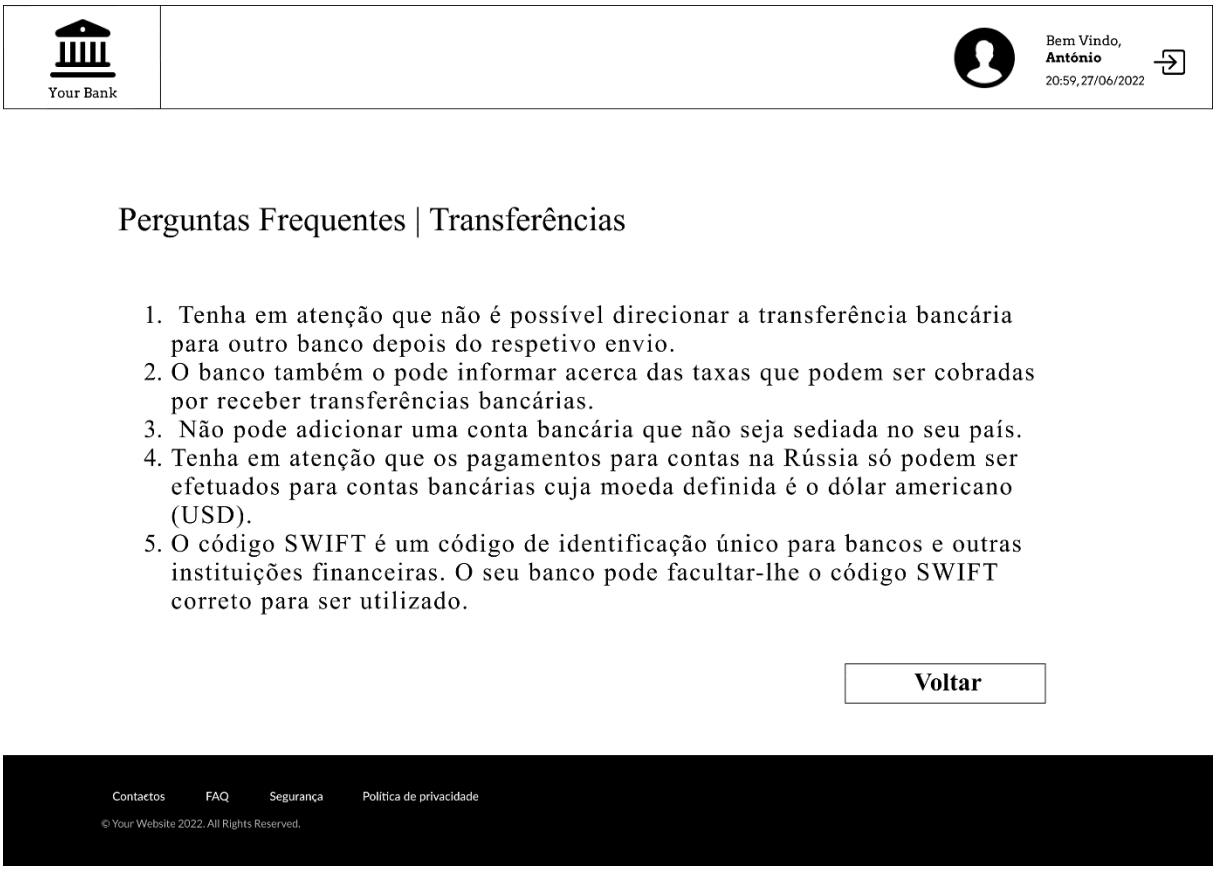

Figura C. 15 – *Mockup* que representa o tema de cada área das perguntas frequentes.

Figura C. 16 – *Mockup* que representa os contactos de apoio a qualquer dúvida.

Figura C. 17 – *Mockup* que representa a atividade do cliente.

Figura C. 18 – *Mockup* que representa em detalhe os movimentos.

Figura C. 19 – *Mockup* que representa o sucesso do descarregamento de um ficheiro.

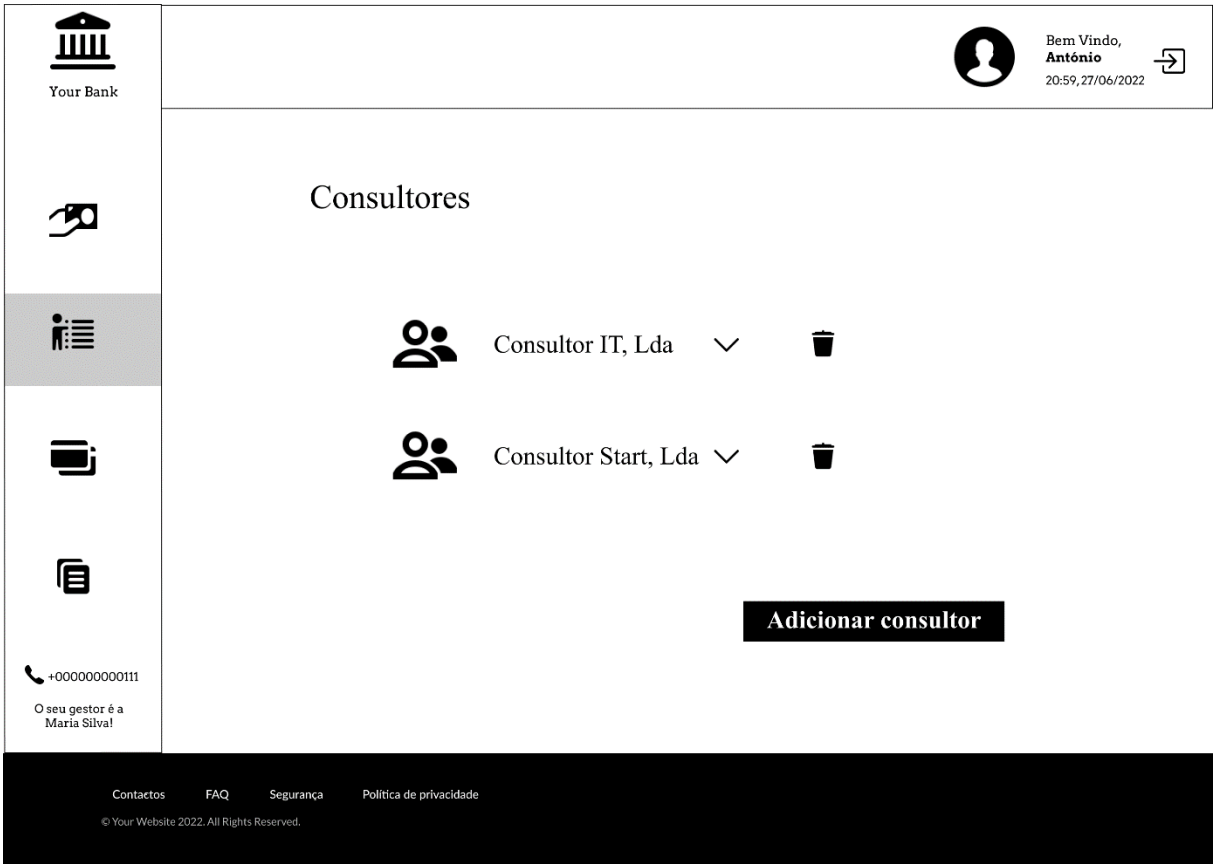

Figura C. 20 – *Mockup* que representa a página dos consultores.

Figura C. 21 – *Mockup* que representa o *modal* para se eliminar um consultor.

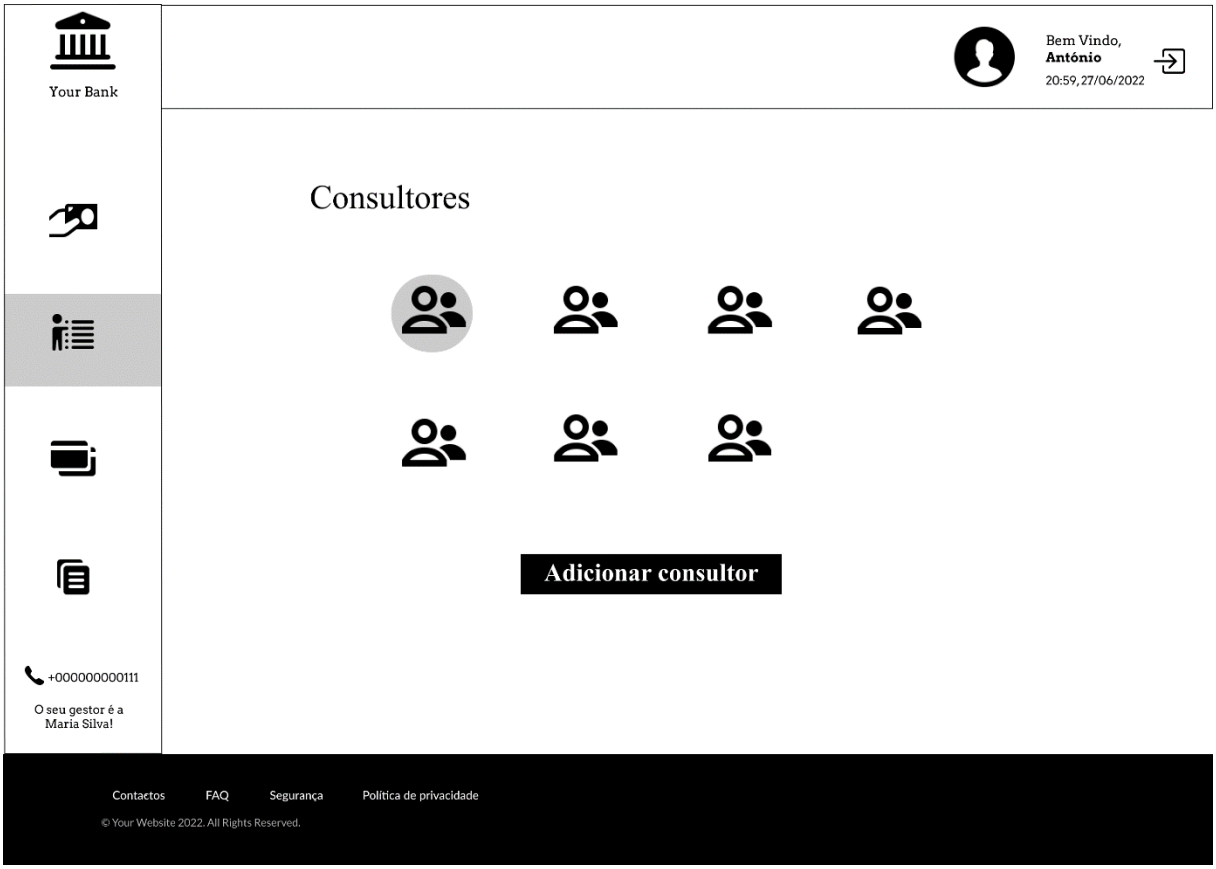

Figura C. 22 – *Mockup* que representa a possível adição de consultores.

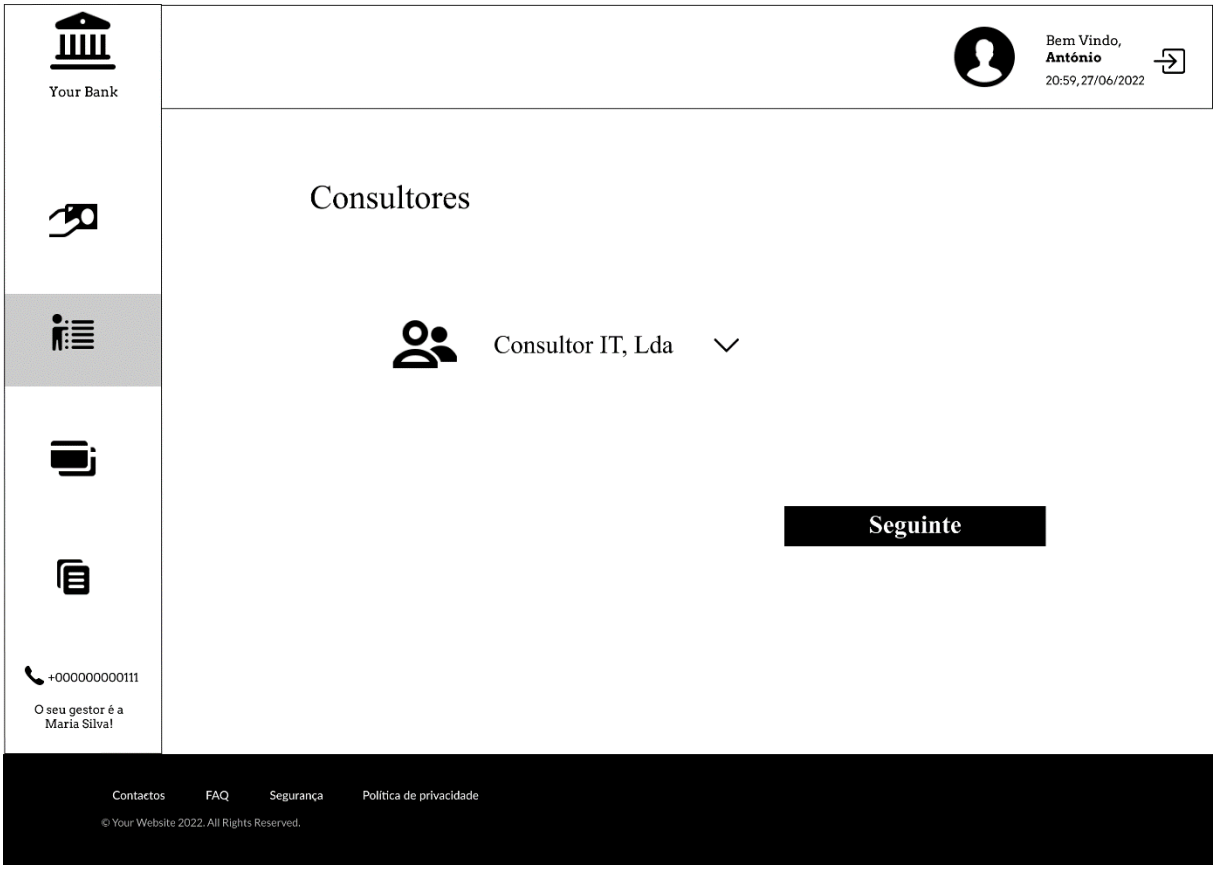

Figura C. 23 – *Mockup* que representa a página com o consultor selecionado para a adição.

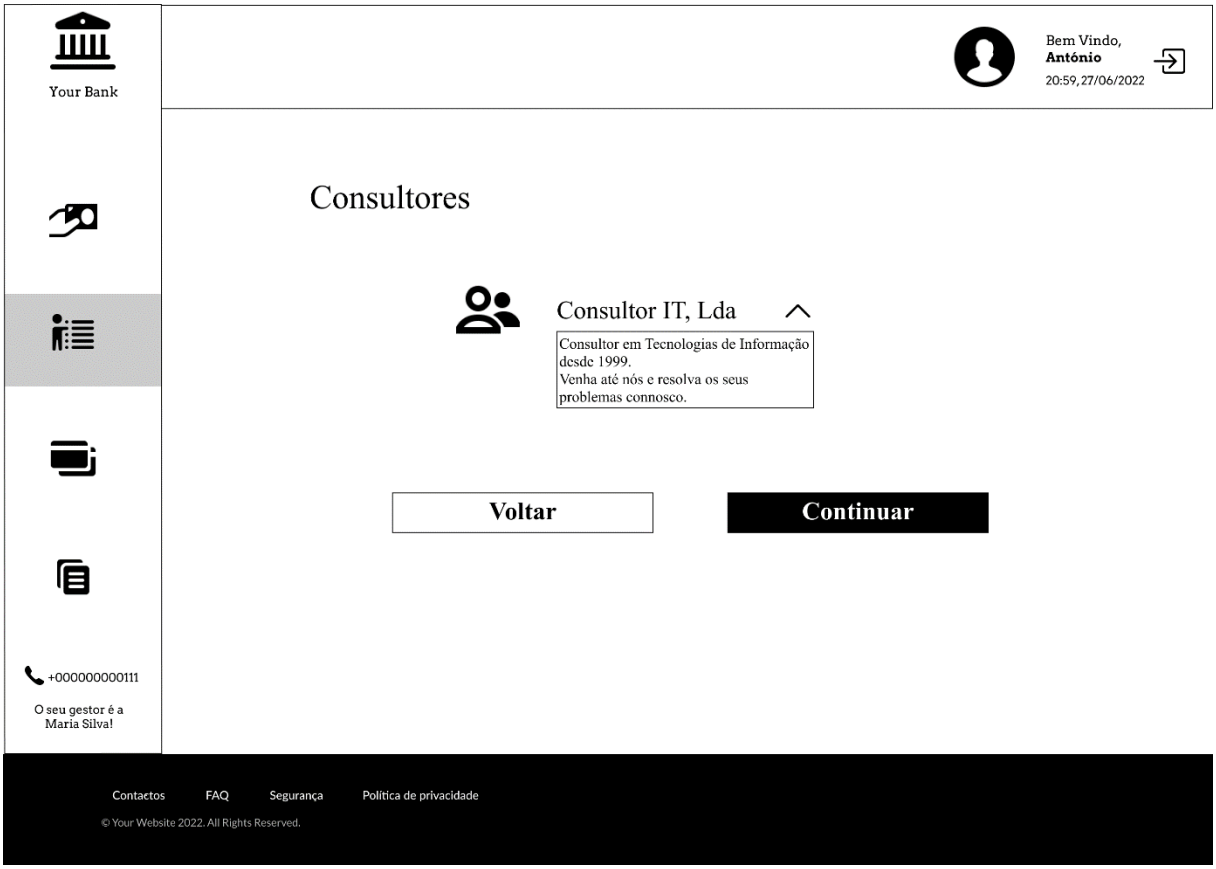

Figura C. 24 – *Mockup* que representa um breve detalhe do consultor escolhido.

Figura C. 25 – *Mockup* que representa a leitura do PDF.

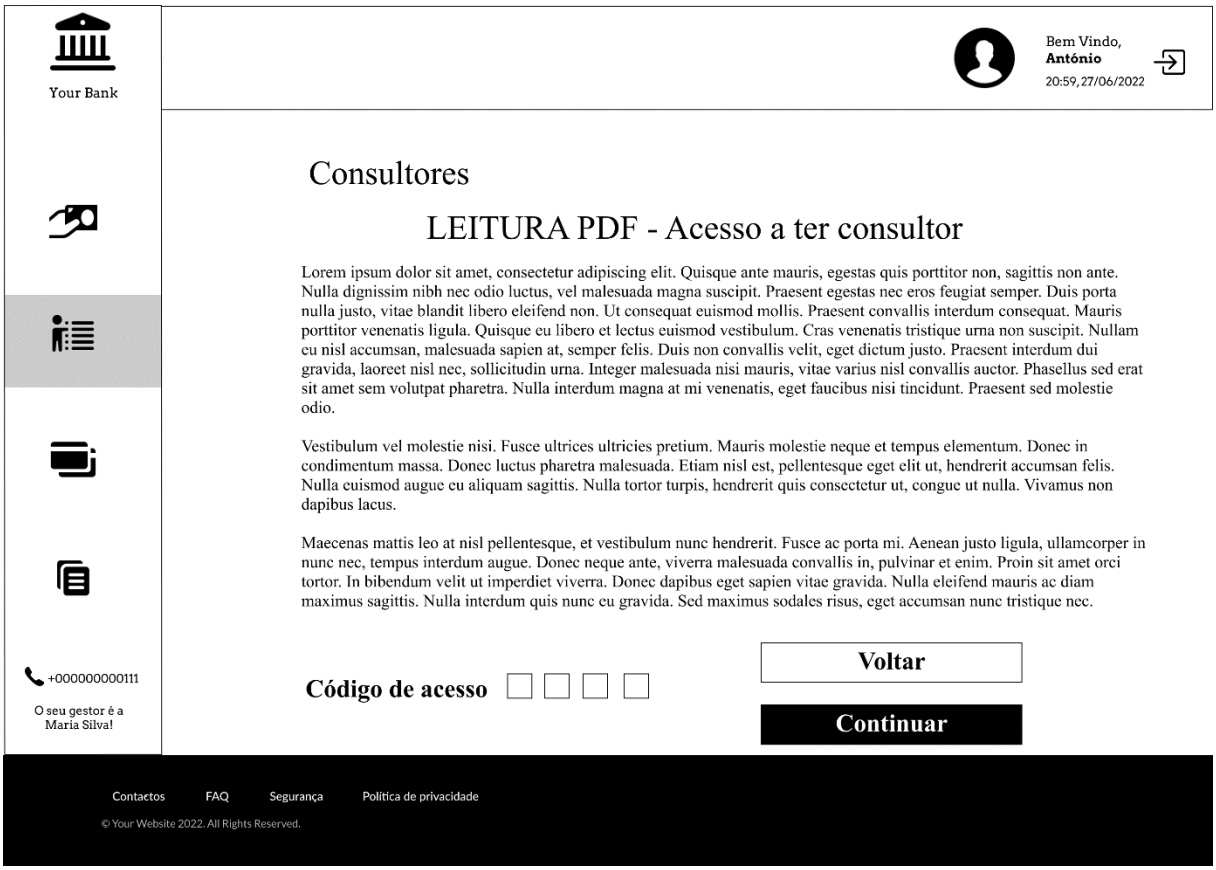

Figura C. 26 – *Mockup* que representa a validação com um código para a adição do consultor.

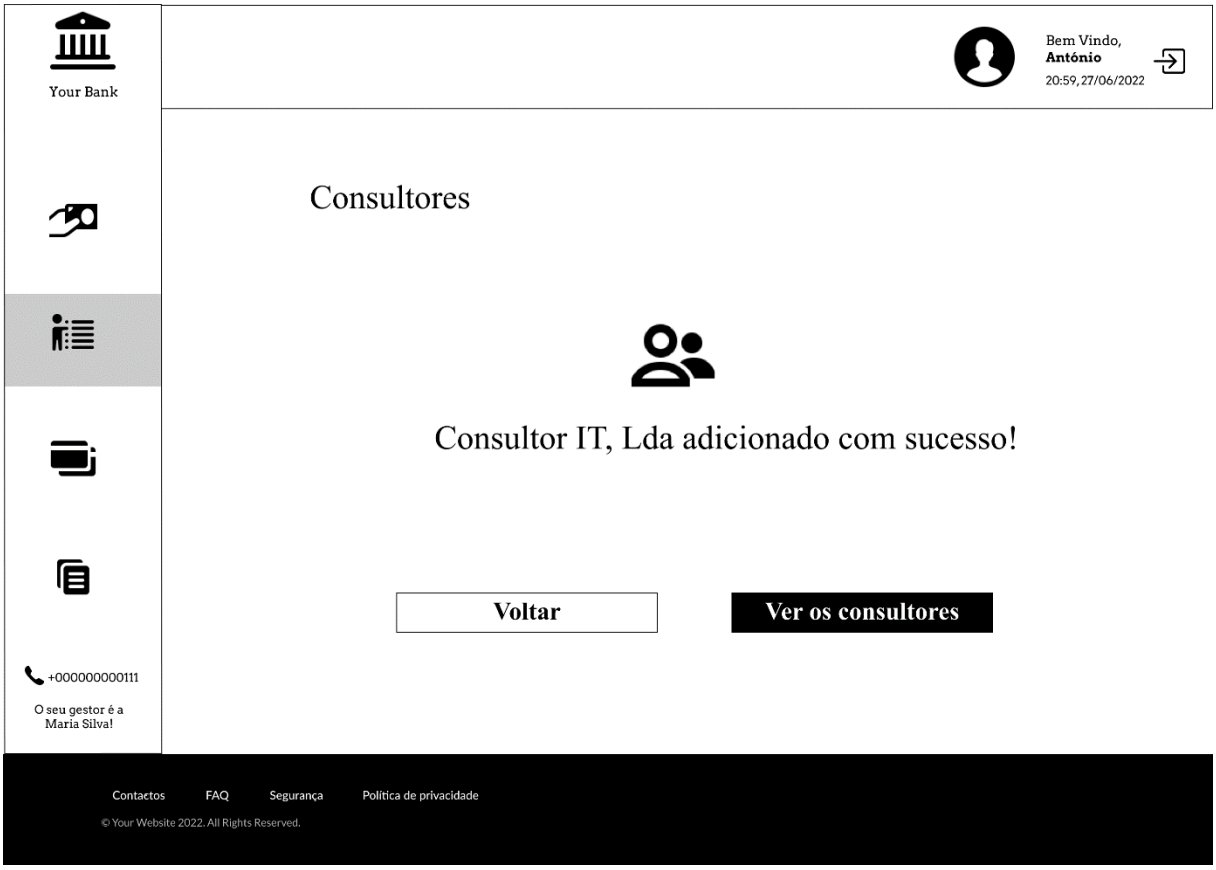

Figura C. 27 – *Mockup* que representa a sucesso da inserção do consultor.

Figura C. 28 – *Mockup* que representa os tipos de pagamentos.

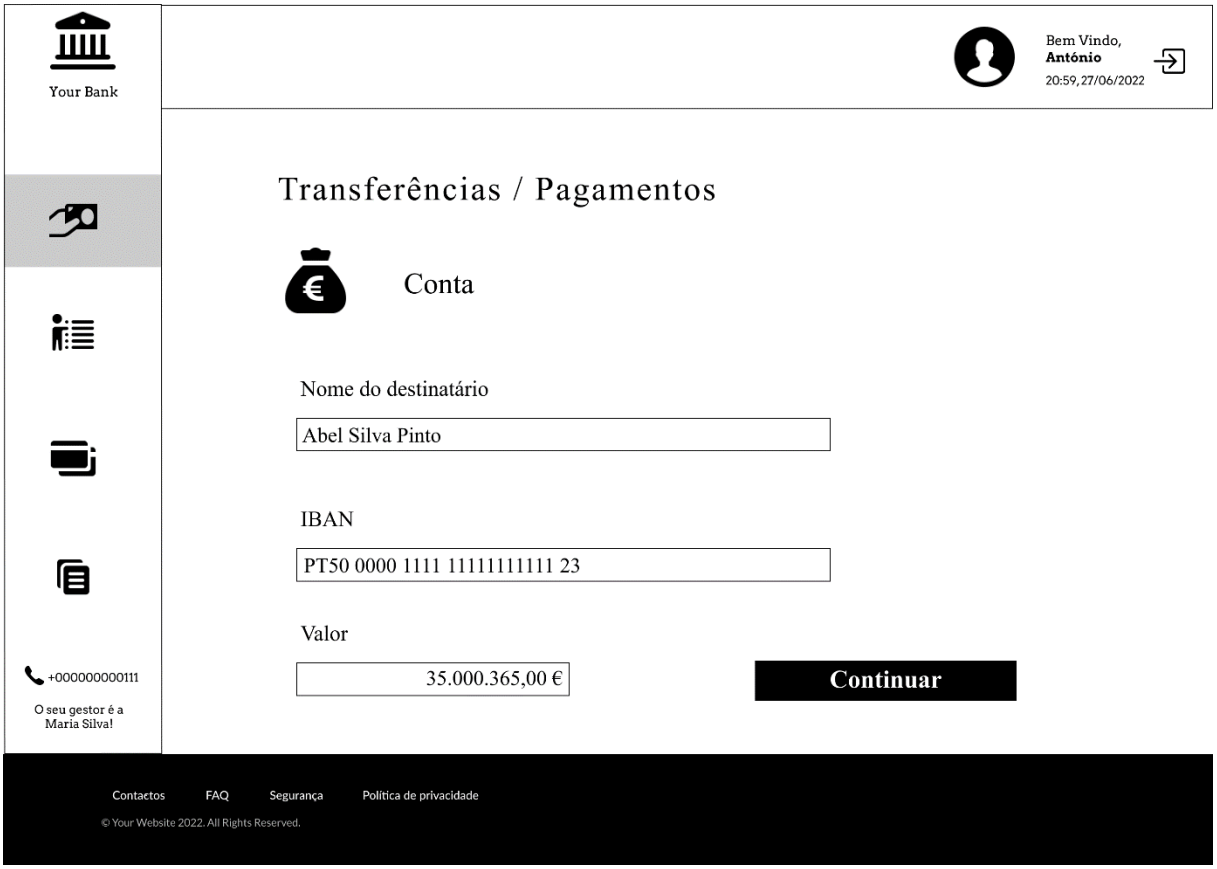

Figura C. 29 – *Mockup* que representa a página com os dados para o pagamento.

Figura C. 30 – *Mockup* que representa o *modal* para confirmar os dados de pagamento.

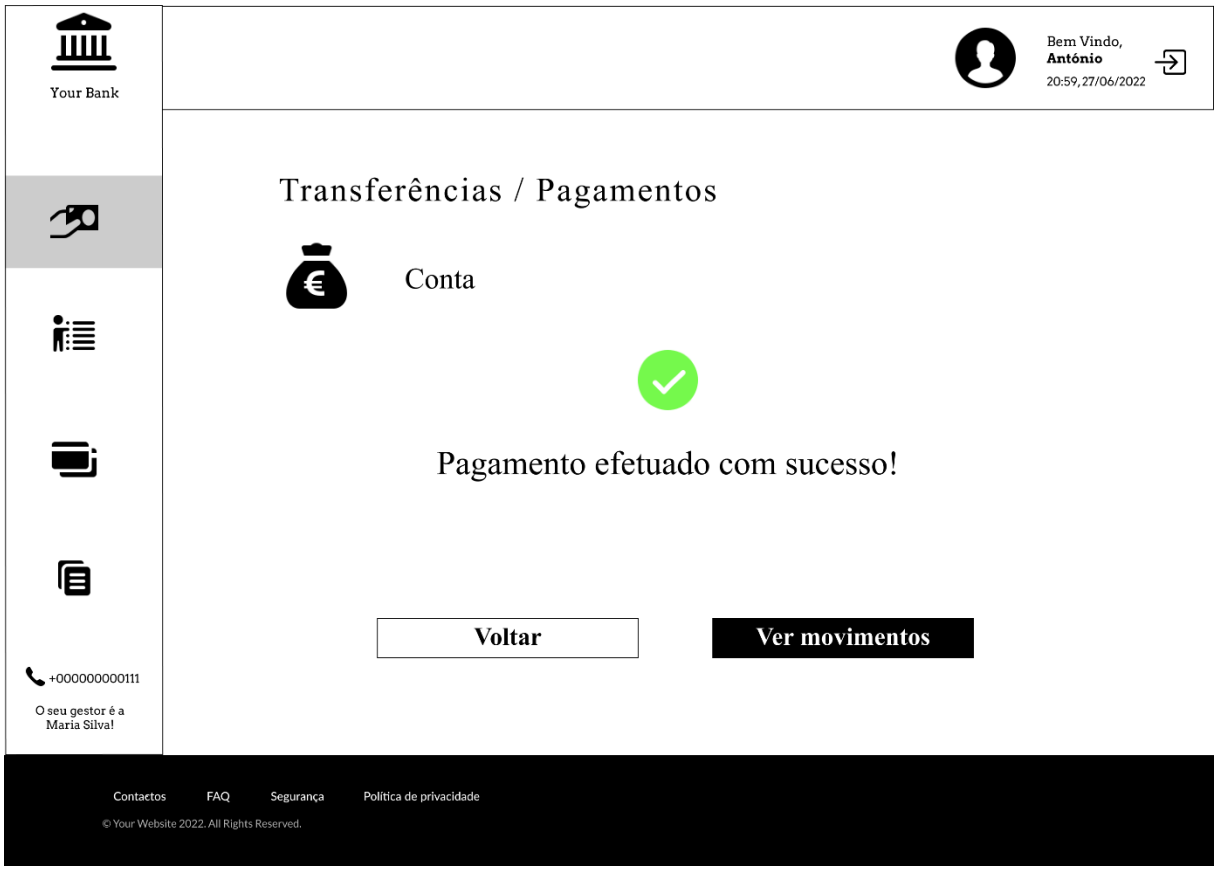

Figura C. 31 – *Mockup* que representa o sucesso do pagamento por conta para uma outra conta.

Figura C. 32 – *Mockup* que representa o *modal* de indisponibilidade de serviço.

Figura C. 33 – *Mockup* que representa o *modal* de sessão expirada.

Figura C. 34 – *Mockup* que representa as contas associadas a um cliente.

Figura C. 35 – *Mockup* que representa a edição de uma certa conta.

Figura C. 36 – *Mockup* que representa o *modal* de eliminação de conta.

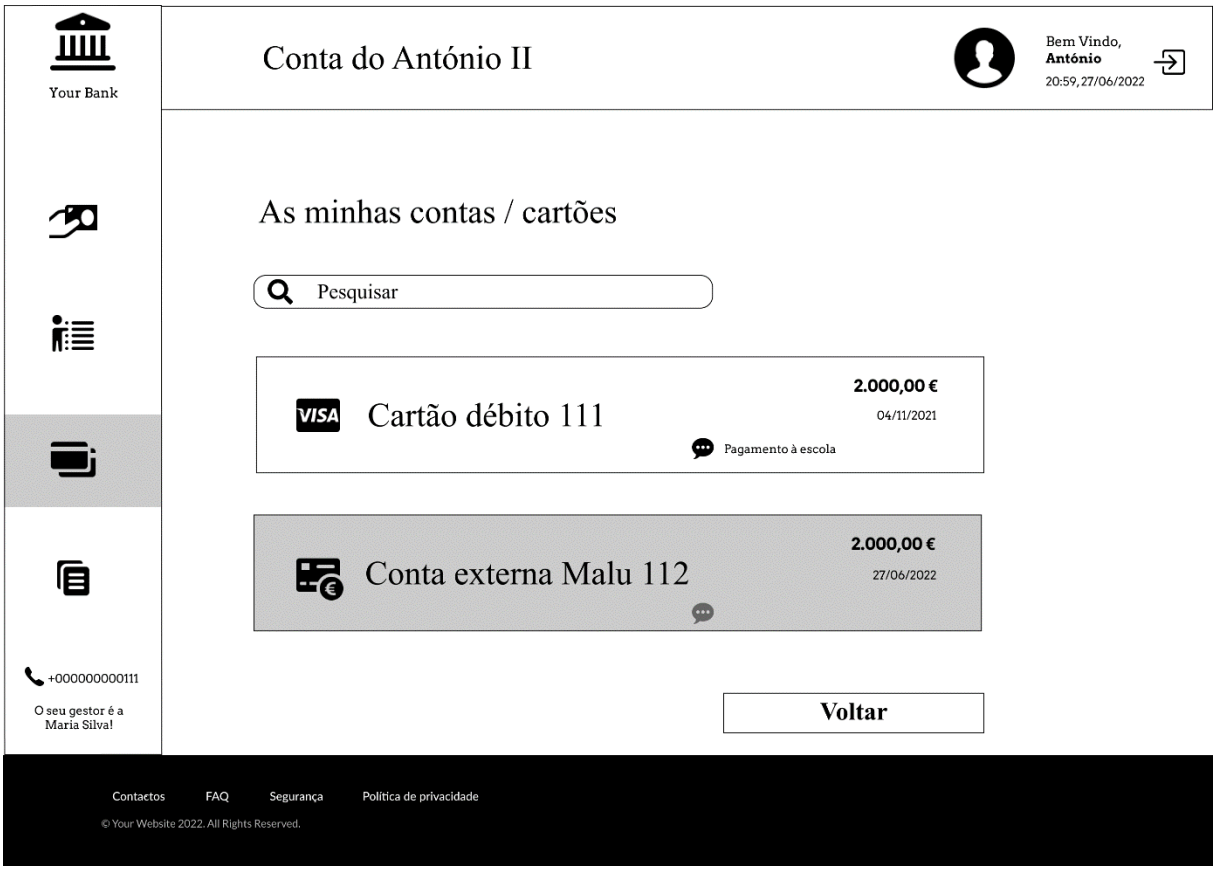

Figura C. 37 – *Mockup* que representa o detalhe de uma conta com outras contas e/ou cartões.

Figura C. 38 – *Mockup* que representa o detalhe de uma conta.

Figura C. 39 – *Mockup* que representa o download de um ficheiro através de uma conta.

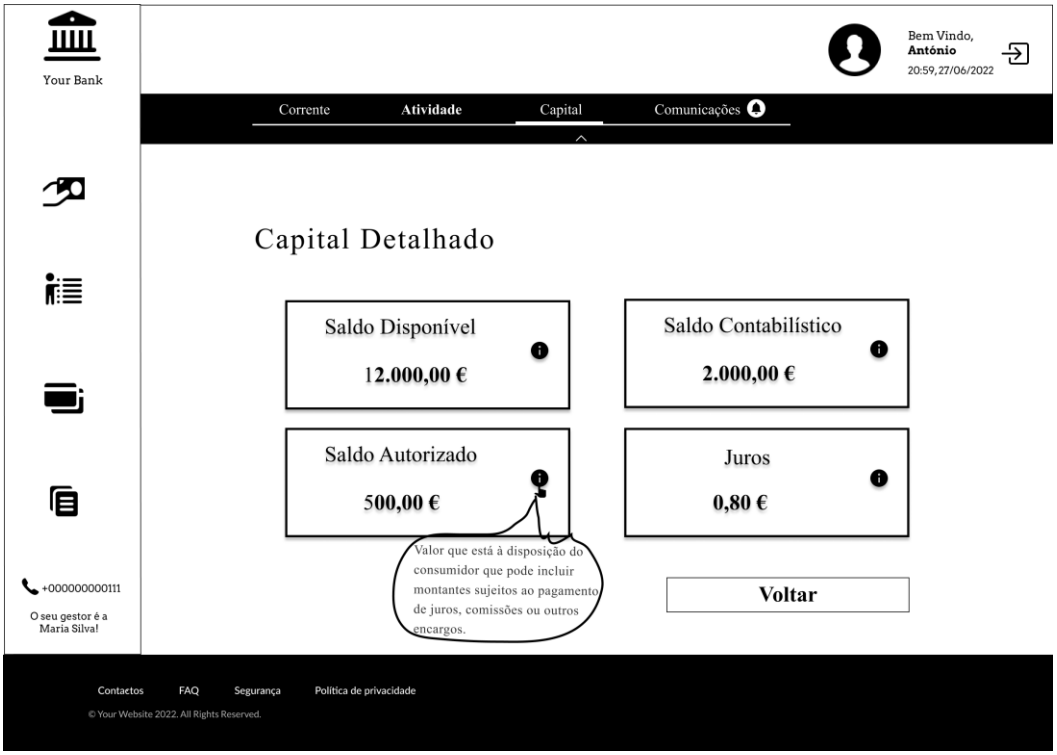

Figura C. 40 – *Mockup* que representa o capital detalhado.

Todos os *mockups* estão disponibilizados no seguinte link https://www.figma.com/file/3uYndFvvdLscxQj3jwUeyw/Est%C3%A1gio?node-id=0%3A1 . Apenas com acesso para orientadores e professores do estágio desenvolvido no Instituto Politécnico de Bragança.

Apêndice D.

User Stories envolventes no trabalho realizado

- Cliente

Identificador	Nome	Prioridade	Descrição
USC01	Editar dados pessoais	Muito alto	Como cliente quero editar as informações da conta.
USC02	Consultar a segurança e as recomendações	Muito alto	Como cliente quero poder consultar a segurança e as recomendações.
USC03	Consultar dados pessoais	Muito alto	Como cliente quero poder consultar em detalhe todas as informações da minha conta.
USC04	Consultar as contas associadas	Muito alto	Como cliente posso consultar todas as contas associadas.
USC05	Verificar o saldo da conta associada	Muito alto	Como cliente posso verificar o saldo da conta bancária associada.
USC06	Pesquisar contas associadas	Muito alto	Como cliente posso pesquisar contas associadas.
USC07	Consultar detalhe da(s) conta(s) associada(s)	Muito alto	Como cliente posso ver o detalhe das contas associadas
USC08	Pesquisar os consultores	Muito alto	Como cliente posso pesquisar todos os consultores.
USC09	Consultar movimentos	Muito alto	Como cliente posso consultar os movimentos.
USC10	Consultar detalhe dos movimentos	Muito alto	Como cliente quero poder consultar em detalhe os movimentos.
USC11	Consultar o capital financeiro	Muito alto	Como cliente posso consultar o capital financeiro.
USC12	Consultar o detalhe do capital financeiro	Muito alto	Como cliente quero poder consultar em detalhe o património financeiro.
USC13	Descarregar comprovativos	Muito alto	Como cliente posso descarregar comprovativos.
USC14	Pesquisar os contactos de suporte	Muito alto	Como cliente quero poder pesquisar os contactos de suporte que tenho direito se tiver dúvidas.
USC15	Adicionar consultor	Muito alto	Como cliente posso adicionar um certo consultor caso o mesmo não se encontre na plataforma.
USC16	Consultar comunicações	Alto	Como cliente posso consultar

			as minhas comunicações.
USC17	Consultar os tipos de pagamento	Alto	Como cliente poso consultar os tipos de pagamento.
USC18	Encontrar propostas de empréstimo	Alto	Como cliente posso pesquisar propostas de empréstimo.
USC19	Simular um empréstimo	Alto	Como cliente posso simular propostas de empréstimo.
USC20	Remover o acesso às contas associadas	Médio	Como cliente posso eliminar contas agregadas à minha conta.
USC21	Consultar perguntas frequentes	Muito baixo	Como cliente posso consultar as perguntas frequentes.

Tabela D.1 – Tabela que gere as US do Cliente.

- Agente Bancário

Identificador	**Nome**	**Prioridade**	**Descrição**
USA01	Editar dados pessoais	Muito alto	Como agente bancário autenticado quero editar as minhas informações da conta.
USA02	Consultar dados pessoais	Muito alto	Como agente bancário quero poder consultar em detalhe todas as informações da minha conta.
USA03	Consultar a segurança e as recomendações	Muito alto	Como agente bancário quero poder consultar a segurança e as recomendações.
USA04	Consultar detalhe da conta dos clientes	Muito alto	Como agente bancário quero poder consultar em detalhe todas as informações das contas.
USA05	Consultar as contas dos clientes	Muito alto	Como agente bancário posso consultar todas as contas associadas ao banco.
USA06	Pesquisar contas dos clientes	Muito alto	Como agente bancário posso pesquisar contas dos clientes.
USA07	Consultar os consultores	Muito alto	Como agente bancário posso consultar todos os consultores.
USA08	Consultar movimentos dos clientes	Muito alto	Como agente bancário posso consultar os movimentos.
USA09	Consultar detalhe dos movimentos dos clientes	Muito alto	Como agente bancário quero poder consultar em detalhe os movimentos.
USA10	Consultar capital financeiro dos clientes	Muito alto	Como agente bancário posso consultar o capital financeiro dos clientes associados.
USA11	Consultar detalhe do capital financeiro dos clientes	Muito alto	Como agente bancário quero poder consultar em detalhe o capital financeiro.
USA12	Pesquisar os contactos de suporte	Muito alto	Como agente bancário quero poder pesquisar os contactos de suporte.
USA13	Adicionar consultor a uma conta de algum cliente	Muito alto	Como agente bancário posso adicionar um certo consultor a uma conta de um dado

			cliente.
USA14	Consultar comunicações	Alto	Como agente bancário posso consultar as minhas comunicações.
USA15	Encontrar propostas de empréstimo	Alto	Como agente bancário posso encontrar propostas de empréstimo.
USA16	Validar propostas de empréstimo	Alto	Como agente bancário posso validar propostas de empréstimo.
USA17	Consultar a informação do consultor	Médio	Como agente bancário posso consultar a informação do consultor.
USA18	Consultar perguntas frequentes	Muito baixo	Como agente bancário posso ver as perguntas frequentes.

Tabela D.2 – Tabela que gere as US do Agente Bancário/ Operador.