REM

Desenvolvimento de *Widgets* para *Internet Banking*

RELATÓRIO DE ESTÁGIO DE MESTRADO

Maria Joselyn Dos Ramos Ferreira
MESTRADO EM ENGENHARIA INFORMÁTICA

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

setembro | 2017

Desenvolvimento de *Widgets* para *Internet Banking*

RELATÓRIO DE ESTÁGIO DE MESTRADO

Maria Joselyn Dos Ramos Ferreira
MESTRADO EM ENGENHARIA INFORMÁTICA

ORIENTADOR
Eduardo Leopoldo Fermé

CO-ORIENTADOR
Pedro Miguel Santos Camacho

Desenvolvimento de *Widgets* para *Internet Banking*

Maria Joselyn Dos Ramos Ferreira

Constituição do júri de provas públicas:

Karolina Baras, Prof.ª Auxiliar da Universidade da Madeira, Presidente

Pedro Campos, Prof. Auxiliar da Universidade da Madeira, Vogal

Eduardo Fermé, Prof. Associado com Agregação da Universidade da Madeira, Vogal

Novembro 2017

Funchal – Portugal

ABSTRACT

This abstract emerges to expose the complete experience for the project developed during the curricular internship included in the conclusion plan for the Master's Degree in Computer Engineering from the University of Madeira. The internship took place in EXICTOS, a company specialised in the development of software for the financial sector.

Technology is being commonly used to facilitate everyday tasks, one of those tasks being banking operations which are a perfect fit for adaptation and usage with the Internet using Web applications or mobile devices. Given the complexity of the financial sector the end result can easily be an application that is hard for a user to interact so, this project is focused on the development of widgets for the company's Internet Banking, called *eBanka*, with intent to simplify the interaction and execution of certain tasks.

This report will be describing the development process and technology used for the conclusion of this project including, a comparison analysis of some popular technologies and tools in contrast to the ones used in the company.

During the internship project development the software engineering processes were followed so, requirements were defined and promptly chased by the prototyping of the graphical interface for the widgets which were then tested by users and iterated numerous times.

Afterwards the widgets functionalities development started which divided the project in two versions, for the desktop and for mobile devices. Lastly, the usability tests were done and the results evaluated, which allowed to conclude that the widgets work correctly and have a friendly interface corresponding to the initial project prospects.

KEYWORDS

Software Development

Web Applications

Widgets

Internet Banking

Prototyping

Usability

RESUMO

O presente relatório surge de modo a expor toda a experiência vivida no projeto desenvolvido ao longo do estágio curricular inserido no plano de conclusão do Mestrado em Engenharia Informática da Universidade da Madeira, o qual foi realizado na empresa EXICTOS, especializada no desenvolvimento de *software* para setores financeiros.

Hoje em dia é cada vez mais habitual o uso de tecnologias para facilitar muitas das tarefas quotidianas, tal como as operações bancárias. Estas são fáceis de enquadrar neste contexto tecnológico sendo frequentemente efetuadas através da Internet utilizando aplicações *Web* ou aplicações para dispositivos móveis. Devido à complexidade destas aplicações para o setor bancário estas podem tornar-se difíceis de interagir para os utilizadores, pelo que este projeto se centra no desenvolvimento de *widgets* para o sistema de *Internet Banking* da empresa, o *eBanka*, de forma a simplificar a execução de determinadas tarefas e facilitar a interação.

Neste relatório será descrito o processo de desenvolvimento do projeto e as tecnologias utilizadas, onde será também apresentada a análise prévia realizada sobre algumas tecnologias e ferramentas estudadas como modo de comparação entre as tecnologias mais populares na atualidade e as utilizadas na empresa.

Durante o estágio, foi seguido o processo de desenvolvimento da Engenharia de *Software*, onde foram levantados os requisitos do projeto e criados protótipos para estudar e definir a interface gráfica dos *widgets*; estes protótipos foram testados com utilizadores e iterados várias vezes.

Posteriormente, foi iniciada a implementação das funcionalidades dos *widgets*, que dividiu o projeto em duas versões, uma versão para *desktop* e outra para dispositivos móveis. Finalmente foram realizados os testes de usabilidade e avaliados os resultados obtidos, no qual foi possível concluir que os *widgets* funcionaram corretamente, possuem uma interface fácil de utilizar e correspondem às expetativas iniciais do projeto.

PALAVRAS-CHAVE

Desenvolvimento de *Software*

Aplicações *Web*

Widgets

Internet Banking

Prototipagem

Usabilidade

AGRADECIMENTOS

Por toda a compreensão e apoio prestado, considero necessário dedicar as seguintes linhas às pessoas que, além de ajudar de forma significativa à concretização deste projeto, foram também parte essencial de toda a experiência vivida durante esta etapa da minha vida.

Agradeço à empresa EXICTOS pela oportunidade de estágio, a todos os seus colaboradores por sempre se mostrarem dispostos a ajudar, aos *designers* João Camacho e Guida Santos, e especialmente a Vítor Camacho, pela sua importante contribuição neste projeto e pelos conhecimentos partilhados ao longo do estágio.

À Universidade da Madeira e a todos os professores que fizeram parte do meu percurso académico e contribuíram para o meu desenvolvimento profissional e pessoal. Um agradecimento em especial aos meus orientadores, o Professor Eduardo Fermé e o Engenheiro Pedro Camacho, pela orientação, disponibilidade e experiências partilhadas.

À minha família, por estarem sempre atentos e por todo o apoio dado. Quero agradecer principalmente à minha irmã, Fátima Dos Ramos, por ser o meu grande suporte, por toda a sua paciência e conselhos e pelo seu apoio incondicional em todos os aspetos da minha vida.

Aos meus colegas e amigos, obrigada por me acompanharem neste percurso e por todas as experiências que vivemos juntos, quero agradecer em especial a Luis Sousa, pela sua boa disposição em me ajudar e motivar sempre.

Muito obrigada a todos os que participaram direta ou indiretamente nesta etapa tão importante da minha vida.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

ACRÓNIMOS

API – *Application Programming Interface*

BCP – Banco Comercial Português

BPI – Banco Português de Investimento

CAOP – *Context Aware Omnichannel Platform*

CRUD – *Create, Read, Update and Delete*

CSS – *Cascading Style Sheets*

DCS – *Distribution Channel Solutions*

DOM – *Document Object Model*

DVCS – *Distributed Version Control Systems*

ES6 – ECMAScript 6

HATEOAS – *Hypermedia As The Engine Of Application State*

HTML – *HyperText Markup Language*

HTTP – *HyperText Transfer Protocol*

HTTPS – *HyperText Transfer Protocol Secure*

JSON – *JavaScript Object Notation*

MVC – *Model View Controller*

MVVM – *Model View ViewModel*

NPM – *Node Package Manager*

OTP – *One Time Password*

PAF – Protótipo de Alta Fidelidade

PBF – Protótipo de Baixa Fidelidade

PFS – *Promosoft Financial Suit*

REST – *Representational State Transfer*

SEO – *Search Engine Optimization*

SOAP – *Simple Object Access Protocol*

SPA – *Single Page Application*

Sass – *Syntactically Awesome StyleSheets*

URI – *Uniform Resource Identifier*

URL – *Uniform Resource Locator*

VCS – *Version Control Systems*

XML – *eXtensible Markup Language*

XP – *Extreme Programming*

GLOSSÁRIO

A

Acoplamento

Indicador qualitativo do grau em que um módulo do sistema está ligado aos outros e ao o mundo exterior. 21, 27

Aplicações nativas

São as aplicações desenvolvidas e otimizadas especificamente para um determinado sistema operativo e para a plataforma de desenvolvimento do fabricante (Android, iOS, etc.), adaptando-se a 100% às funcionalidades e características do dispositivo. 38

Aplicações *Web* de uma única página (SPA)

É um tipo de aplicação *Web* onde todos os ecrãs apresentam a mesma página, sem atualizar o *browser*. Apesar de ter uma única página, a aplicação pode ter várias vistas, portanto, na mesma página é possível intercambiar vistas distintas, produzindo o efeito de ter várias páginas. 32, 33, 34, 77

Atributo de qualidade

Propriedade mensurável ou testável de um sistema que é usado para indicar o quão bem o sistema satisfaz as necessidades das suas partes interessadas. 22

B

Back-end

A parte de um sistema de computador ou aplicação que não é acedido diretamente pelo utilizador, geralmente responsável por armazenar e tratar dados. 2, 17, 18, 72, 73

C

Cache

Secção de memória de alta velocidade que armazena, temporariamente, informação utilizada com frequência, permitindo o acesso mais rápido. 8, 21

Código aberto

Refere-se ao *software* para o qual o seu código fonte original é disponibilizado abertamente e pode ser redistribuído e modificado. 31, 32, 35, 77, 79, 80

Compatibilidade

Capacidade que um sistema tem de se adaptar a vários ambientes (distinto *hardware*, sistemas operativos, *browsers*, etc.). 59

Confiabilidade

Grau em que um sistema deve cumprir a sua função e com a precisão necessária. 21, 22, 58

D

Dashboard

Página inicial num *site*, que dá acesso a diferentes elementos da funcionalidade do *site*. 5, 6, 8, 44, 45, 70, 90, 100, 117, 131, 132

***Data Binding* (ligação de dados)**

Consiste em unir, em tempo real, os dados de dois elementos. 32, 40, 42, 78

Debugging

Identificar e remover erros de um *software*..... 40

Deploy

Instalação de um sistema ou aplicação em todos os dispositivos necessários. 28

Desempenho

O tempo e a capacidade do sistema de *software* para atender aos requisitos de tempo especificados. 8, 32, 35, 38, 39, 40, 41, 43, 58, 65, 76, 78, 97

Disponibilidade

Capacidade de um sistema para mascarar ou reparar falhas, de modo a que o período de interrupção do serviço não exceda o valor exigido durante um intervalo de tempo especificado.. 21, 22, 36, 58

DOM (*Document Object Model*)

É uma interface de programação de aplicações (API) para documentos HTML e XML. Define a estrutura lógica dos documentos e a maneira em que é acedido e manipulado um documento. . 33, 34, 40, 42

E

Encapsular

Ocultar os detalhes de implementação internos de um objeto aos outros, permitindo assegurar que o conteúdo da informação se encontra seguro do mundo exterior. 7, 25, 74

Escalabilidade

Capacidade de adaptação e resposta de um sistema no que diz respeito ao seu rendimento à medida que aumenta de forma significativa o número de utilizadores. 20, 21, 22, 59

Extensibilidade

Facilidade que tem um sistema de se adaptar às mudanças dos requisitos. 19, 21, 22

F

Flexibilidade

Esforço necessário para modificar um programa que já opera. 20, 22, 39, 41, 43, 79

Framework

Estrutura real ou conceitual composta de componentes personalizáveis e trocáveis que serve como suporte para o desenvolvimento de aplicações. 3, 7, 31, 32, 38, 39, 41, 43, 45, 59, 73, 74, 75, 76, 77, 78, 80, 90, 97, 99

Front-end

A parte de um sistema de computador ou aplicação com o qual o utilizador interage diretamente. 2, 3, 4, 17, 18, 28, 32, 42, 72, 73, 97, 98, 99

I

Internet Banking

Ferramentas que disponibilizam os bancos aos seus clientes para aceder aos serviços que estes oferecem através de Internet. 2, 3, 5, 44

Interoperabilidade

Esforço necessário para acoplar um sistema a outro. 19

J

Janela modal

Janela emergente que possui uma função específica e aparece a partir de uma janela principal, apresenta conteúdo ou informação relevante e não permite interagir com outras janelas até não ser fechada. 79

L

Latência

Tempo que tarda um dado em estar disponível desde o momento em que foi realizado o seu pedido. 21

M

Manutenção

Esforço necessário para detetar e corrigir problemas num sistema. 25, 26, 27, 38, 59, 60, 96, 97

Middleware

Software que atua como uma ponte entre um sistema operativo ou base de dados e as aplicações, especialmente numa rede. 2, 16, 17, 43, 72, 73, 99

Mobile Banking

Ferramentas que disponibilizam os bancos aos seus clientes para aceder aos serviços que estes oferecem através de dispositivos móveis, como um telemóvel ou *tablet*. 2, 43

Modificabilidade

Medida do custo de mudança do *software*, tanto no tempo quanto no dinheiro..... 59

O

Operações CRUD

Criar, ler, atualizar e eliminar recursos nas bases de dados. 23

P

Padrão de desenho

Solução geral para problemas típicos e recorrentes que ocorrem no desenvolvimento de aplicações. 25, 38, 75, 78

Portabilidade

Facilidade com que o *software* criado para funcionar em uma plataforma pode ser alterado para ser executado em uma plataforma diferente. 59

Protocolo

Conjunto de regras que torna possível a execução de um programa de modo eficiente e sem erros. 19, 20, 22, 24

Proxy

Programa ou dispositivo que realiza uma tarefa de acesso a Internet em vez de outro computador, é um ponto intermediário entre o *browser* e o servidor. 7, 8

R

Renderizar

Conjunto de ações que realiza um *browser* para apresentar um *site* ao utilizador. 33, 40, 43

Repositório

Localização central na qual os dados são armazenados e geridos. 35, 60, 78, 79, 91

Responsivo

Diz-se de um *site* ou página da Internet programada para apresentar diferentes configurações, adequando-se automaticamente ao formato do ecrã em que é exibida. 90

Reutilização

Grau em que um programa (ou partes de um) pode ser reutilizado em outras aplicações. 26, 27, 39, 41, 74, 97

Routing

Processo de envio de informações de uma rede de computadores para outra. 32

S

Script

Conjunto de instruções em código para que uma função seja executada numa aplicação. 15, 16, 21

Segurança

Medida da capacidade do sistema de proteger dados e informações do acesso não autorizado enquanto ainda permite acesso a pessoas e sistemas autorizados. 8, 18, 21, 22, 48

SEO

Processos de melhorar a visibilidade de um *site* nos diferentes motores de pesquisa, como *Google* ou *Bing*, para conseguir acesso a uma posição destacada nos resultados, segundo um determinado critério de pesquisa. 40

Slideshow

Estrutura dinâmica comumente usada nos *sites* para a apresentação de conteúdo de forma sequencial. 47, 48, 62, 79, 82, 84, 99

Superset

Linguagem escrita sobre outra linguagem. No caso do TypeScript e JSX são linguagens baseadas em JavaScript, que oferecem novas características além das que já existem na linguagem original. 33, 39

U

Usabilidade

Facilidade que tem o utilizador de realizar a tarefa desejada e o tipo de suporte que o sistema lhe oferece. 4, 59, 60, 61, 63, 65, 66, 67, 76, 91, 92, 97, 99

W

Widget

Pequena aplicação que, em computadores, *tablets* e certos telemóveis, permite aceder não só ao programa associado como também a algumas das suas funcionalidades. 3, 5, 6, 7, 45, 54, 55, 56, 57, 58, 60, 61, 62, 63, 65, 66, 67, 68, 69, 70, 71, 74, 76, 79, 82, 83, 84, 85, 86, 87, 88, 92, 93, 94, 95, 99, 100, 121, 131, 132

I. INTRODUÇÃO

I.1. CONTEXTO DO ESTÁGIO

O estágio curricular, previsto pelo plano do Mestrado em Engenharia Informática da Universidade da Madeira, encontra-se inserido na Unidade Curricular "Projeto/Estágio/Dissertação", e representa para mim uma oportunidade de contextualizar a experiência académica adquirida num ambiente real, obter novos conhecimentos sobre a área e melhorar e reforçar as minhas competências.

A oportunidade do estágio surgiu após a minha participação numa apresentação da empresa EXICTOS realizada na Universidade da Madeira, em que expuseram o funcionamento da empresa e convidaram todos os assistentes a visitar as suas instalações. Aceitei o convite para participar e foi nessa visita que tomei conhecimento da existência de propostas para projetos de estágios de mestrado.

Após analisadas todas as propostas, escolhi o projeto que considerei mais adequado ao meu perfil, tendo como base o objetivo a atingir e as tecnologias a serem utilizadas. Posteriormente contactei o responsável do Departamento DCS (*Distribution Channel Solutions*) da empresa, o Engenheiro Pedro Camacho, onde definimos o necessário para o iniciar o estágio.

A **EXICTOS,** nascida no ano 1998 em Funchal e nomeada na altura como *Promosoft*, é uma empresa especializada na produção e implementação de *software* para setores financeiros. Atua em 8 países e possui filiais em Portugal, Angola, Moçambique, e Cabo Verde [1].

Figura 1 – Logótipo da empresa EXICTOS.

Na sede de EXICTOS localizada na Estrada Comandante Camacho de Freitas, nº 905 e 907 9050-222 Funchal, lugar onde decorreu o estágio curricular, o desenvolvimento dos seus produtos é realizado com a colaboração de duas equipas.

A equipa com maior número de colaboradores é a que desenvolve a plataforma de ***Core Banking*** - ***Banka*** - esta é uma solução destinada a fazer a gestão integrada e completa do negócio das instituições financeiras [1]. Esta equipa encarrega-se do desenvolvimento *back-end,* ou seja, trata de todas as operações bancárias e das bases de dados, além de desenvolver os serviços *Web* que permitem o passo de informação ao **departamento DCS**, onde são desenvolvidas as soluções que estão diretamente relacionadas com os clientes bancários.

O departamento DCS conta com 4 equipas divididas por áreas de especialização, nomeadamente:

- ***Middleware*****:** a equipa de *Middleware* desenvolve os serviços *Web* e trata das informações das bases de dados provenientes da *Banka*. É a intermediária entre a equipa da *Banka* e as equipas de desenvolvimento *front-end* (Canais Presenciais e Canais Não Presenciais).
- **Canais Presenciais:** a equipa de Canais Presenciais cumpre a função de desenvolver o *software* utilizado pelos funcionários dos bancos. O produto desenvolvido pela equipa é o Portal de Operações Bancárias PFS (*Promosoft Financial Suit*).
- **Canais Não Presenciais:** a equipa de Canais Não Presenciais desenvolve produtos para *Internet Banking* e *Mobile Banking*, os quais podem ser acedidos pelos clientes finais, particulares ou empresas, através da *Web* ou dispositivos móveis, sem necessidade de se dirigir a um banco. <u>É nesta equipa onde se encontra enquadrado o projeto curricular</u>.
- Além das quatro equipas anteriormente descritas, o departamento DCS conta com a equipa de ***designers***, responsável pelo desenho de todos os elementos das interfaces gráficas dos produtos e prototipagem dos mesmos.

I.1.1. Canais Não Presenciais

O projeto de estágio decorreu no âmbito da equipa dos **Canais Não Presenciais**; esta equipa está a desenvolver vários projetos, entre os quais se destacam: o projeto para *Internet Banking* ***eBanka***, que possibilita aos utilizadores a realização de operações bancárias através da *Web*, e o projeto **CAOP (*Context Aware Omnichannel Platform*)**, ainda em fase de desenvolvimento, que é uma *framework* interna da empresa que permite construir aplicações tendo como base a arquitetura apresentada no capítulo II.2 deste documento.

O projeto CAOP está a ser criado utilizando tecnologias modernas e é onde futuramente serão integrados os projetos já existentes da equipa, como o *eBanka* (na sua nova versão 2.5), e novos projetos. É neste projeto onde serão incluídos os *widgets* desenvolvidos durante o estágio curricular.

I.2. OBJETIVOS

O objetivo principal deste estágio curricular foi o desenvolvimento *front-end* de *widgets* para *Internet Banking*, a fim de oferecer uma fácil interação com as funcionalidades do sistema e acesso rápido as opções disponíveis.

Para este projeto, os *widgets* propostos pela empresa a serem desenvolvidos foram os que facilitam aos utilizadores a execução das seguintes tarefas:

- *Widget* destinado a facilitar os pagamentos de serviços;
- *Widget* destinado a apresentar a contagem de operações pendentes da aprovação do utilizador e aprovação das mesmas.

No capítulo II.1 serão detalhadamente expostos cada um dos propósitos dos *widgets* acima referidos.

É importante salientar que um dos objetivos do projeto foi desenvolver uma solução (à medida das necessidades da empresa) enquadrada nas características do sistema *eBanka* e ao projeto CAOP onde será futuramente integrado.

I.3. ESTRUTURA DO RELATÓRIO

Este relatório encontra-se dividido em cinco capítulos, correspondentes às cinco principais etapas do desenvolvimento do projeto para o estágio curricular.

Depois de realizada a contextualização do estágio e apresentados os objetivos a atingir (capítulo **I. Introdução**) serão introduzidas as características dos *widgets*, a arquitetura do projeto da empresa onde será integrado este projeto e o processo de *software* seguido durante o seu desenvolvimento (capítulo **II. Estado da Arte**).

Adicionalmente, serão apresentados os aspetos essenciais das aplicações e serviços *Web*, o padrão MVC e a arquitetura baseada em componentes, para posteriormente introduzir as várias abordagens de tecnologias existentes na atualidade para prototipagem e para o desenvolvimento *front-end* de aplicações *Web*. Serão também expostas algumas aplicações com características semelhantes às pretendidas neste projeto.

Já no capítulo **III. Desenvolvimento da Aplicação *Web***, será descrito todo o processo adotado, iniciando pelo levantamento de requisitos do projeto e passando pela elaboração dos protótipos de baixa e alta fidelidade da interface do utilizador. É neste capítulo onde são definidos os serviços *Web* e a arquitetura utilizada.

No capítulo **IV. Implementação**, serão mencionadas as linguagens e tecnologias usadas durante a implementação, será definida a estrutura de ficheiros do projeto e descritas as funcionalidades implementadas e o processo de *refactoring*. Posteriormente, serão apresentados os resultados dos testes de usabilidade e a documentação do projeto.

As conclusões sobre a aplicação, sobre o estágio curricular e as perspetivas futuras do projeto serão descritas no capítulo **V. Conclusão**. Finalmente, serão apresentadas as **Referências** e os **Anexos** referentes ao projeto e a este relatório.

II. ESTADO DA ARTE

II.1. *WIDGETS* PARA O SISTEMA *EBANKA*

A inclusão de *widgets* no sistema *eBanka* permite aos utilizadores realizar facilmente operações bancárias através destes *widgets*, os quais se encontram disponíveis no *dashboard* do utilizador. Deste modo, as tarefas principais podem ser realizadas sem a necessidade de navegar pelos menus e opções do sistema. O *eBanka* também permite ao utilizador escolher quais são os *widgets* que estarão visíveis no seu *dashboard*.

Os *widgets* do projeto foram desenvolvidos para um contexto genérico do *Internet Banking* da empresa, para que desta maneira possam ser integrados em qualquer das aplicações inseridas no projeto CAOP, no entanto, para o caso deste projeto foi definida a sua integração no sistema *eBanka*, especificamente no *dashboard* do utilizador. Por outro lado, para garantir uma adequada visualização dos *widgets* em diferentes dispositivos foram desenvolvidas duas versões, uma versão *Desktop* e uma versão para dispositivos móveis (versão *Mobile*).

A seguir apresentam-se as características mais relevantes dos *widgets* desenvolvidos, nomeadamente, o *widget* para pagamento de serviços e o *widget* para a aprovação de operações pendentes.

II.1.1. *Widget* para Pagamento de Serviços

Para realizar o pagamento de serviços na maioria dos sistemas atuais, é necessário que o utilizador aceda ao sistema e selecione a opção pretendida, introduza os dados do pagamento, nomeadamente, a conta a debitar, o número da entidade e da referência, e o montante a pagar, para finalmente confirmar o pagamento. Esta operação é muito simples e de uso frequente, portanto, num sistema com muitas funcionalidades é importante que o pagamento possa ser realizado de uma forma rápida e direta; uma maneira de o conseguir é simplificar a operação através de um *widget*.

O *widget* para pagamento de serviços está associado a uma nova característica que será introduzida no *eBanka*, que permitirá ao utilizador gerir na página de

pagamentos de serviços as suas entidades favoritas, ou seja, quais são as entidades que utiliza com maior frequência no momento de efetuar um pagamento ou as que pretende pagar com maior facilidade.

Estas entidades estabelecidas como favoritas aparecerão no *widget* apresentado no *dashboard*, permitindo ao utilizador escolher uma delas para realizar a operação. Além disso, o *widget* permite selecionar a conta a debitar, introduzir os restantes dados e finalmente efetuar o pagamento do serviço.

O público-alvo deste *widget* são os utilizadores particulares do sistema.

II.1.2. *Widget* para Operações Pendentes

Geralmente, as grandes empresas possuem um elevado número de funcionários responsáveis por efetuar diversas tarefas, entre estas, as operações bancárias, para tal, as soluções bancárias permitem aos gestores das empresas criar vários utilizadores com diferentes níveis de autorização (**A**, **B** ou **C**, onde **A** corresponde aos utilizadores com maior autorização e **C** com a menor). Desta forma os utilizadores podem aceder ao perfil da empresa no banco e efetuar as respetivas operações autorizadas.

Existem operações bancárias que depois de serem realizadas pelos respetivos utilizadores devem ser aprovadas/assinadas por um ou mais utilizadores (normalmente que possuem um nível de autorização superior, por exemplo, o gestor da empresa) para serem devidamente executadas. É tarefa do utilizador com autorização aprovar todas as operações pendentes da sua assinatura, por este motivo, foi proposto o desenvolvimento de um *widget* como solução para agilizar este processo, que permita ao utilizador visualizar diretamente no *dashboard,* informação relevante sobre estas operações.

O *widget* para as operações pendentes consiste principalmente em apresentar a contagem das operações que se encontram pendentes da assinatura do utilizador e facilitar o acesso à página do sistema onde é possível consultar e aprovar todas estas operações. Além disso, apresenta a contagem das operações que já foram aprovadas/assinadas pelo utilizador, mas cuja execução ainda depende da aprovação de outro(s) utilizador(es).

Adicionalmente, oferece a opção de aprovar diretamente desde o *widget* a operação mais antiga das que se encontram pendentes da aprovação do utilizador. O público-alvo deste *widget* são os utilizadores empresariais do sistema.

II.2. ARQUITETURA DO PROJETO CAOP

Os produtos desenvolvidos pelo departamento DCS da EXICTOS baseiam-se na arquitetura cliente-servidor, que será exposta no capítulo II.4.1. É importante destacar que este projeto será integrado no projeto **CAOP** da empresa e, como referido anteriormente, o CAOP é uma *framework* interna da EXICTOS que possui a sua própria arquitetura bem definida, a seguir apresentada.

O CAOP está estruturado para trabalhar sobre o **Padrão de Arquitetura de Microserviços** (em inglês, *Microservice Architecture Pattern*), em que os sistemas são desenhados como um conjunto de microserviços que escalam de maneira independente. Um microserviço é um componente independente que encapsula a lógica necessária para oferecer todas as funcionalidades pretendidas. Em geral, estas funcionalidades são bastante pequenas (de aqui vem o nome de *micro*) [2].

Cada microserviço oferece uma fronteira bem definida entre módulos, isto permite, entre outras vantagens, que diferentes microserviços possam ser escritos em diferentes linguagens de programação [3].

A nível arquitetural, no CAOP se encontram claramente definidas as aplicações que são compostas pelos módulos cliente e servidor, para além dos módulos *middleware*. Neste contexto as aplicações são consideradas os componentes funcionais e independentes que fazem parte do sistema *eBanka*, como por exemplo, pode ser considerada uma aplicação o componente encarregado de tratar todo o relacionado com a funcionalidade de pagamentos no *eBanka*.

Por outro lado, no CAOP existe apenas um servidor que comunica com o cliente, denominado por *proxy*, e é neste que estão mapeadas todas as aplicações existentes na rede, ou seja, o *proxy* sabe qual é a aplicação e onde se encontra localizada. Os clientes se comunicam com o *proxy* para realizar os pedidos, e por sua vez, o *proxy* comunica com os diferentes microserviços (cada um responsável por uma aplicação). A utilização

do *proxy* evita a exposição dos microserviços ao cliente, aumentando a segurança do sistema.

Os microserviços comunicam entre si quando existem dependências entre eles, além disso, podem existir vários microserviços responsáveis pela mesma aplicação, de modo a reduzir a sobrecarga dos pedidos feitos pelos clientes e aumentar o desempenho do sistema.

Adicionalmente, o CAOP possui um servidor chamado Redis onde estão implementadas as bases de dados, além de funcionar como uma *cache*. Cabe salientar que apenas uma aplicação fornece ao cliente o documento que trata do arranque do sistema e contém tudo o que é comum nas aplicações (index), para depois as outras aplicações apenas a complementarem.

Nos servidores das aplicações são realizados os pedidos para o *middleware*, desta forma, os servidores oferecem uma camada de segurança ao não expor o *middleware* diretamente ao cliente. Cada microserviço no CAOP comunica com um ou mais microserviços que se encontram no *middleware*. Por outro lado, o *middleware* também utiliza um *proxy* para comunicar de forma segura com a *Banka*. O esquema da arquitetura do CAOP pode ser visualizado no Anexo A.

O projeto do *dashboard* com os *widgets* será integrado como uma aplicação dentro do CAOP. A grande vantagem de utilizar esta arquitetura é que os microserviços são pequenos e independentes, o que facilita a escala e instalação das aplicações, além do versionamento das mesmas. Adicionalmente, permite o uso de múltiplas tecnologias e simplifica o desenvolvimento da aplicação, porque cada equipa pode trabalhar num respetivo microserviço sem restringir o desenvolvimento das outras equipas. Por outro lado, a desvantagem principal do uso de esta arquitetura é a complexidade de gestão das aplicações.

II.3. PROCESSO DE *SOFTWARE*

Na Engenharia de *Software* um **Processo de *Software*** cumpre o propósito de desenvolver de forma eficaz e eficiente produtos de *software* que reúnam todos os requisitos pretendidos pelos clientes. Este processo define quem é que está a fazer o

quê, quando e como atingir um determinado objetivo, e o seu uso tem sido um fator fundamental para o sucesso das empresas de desenvolvimento [4].

Cada uma das atividades, ações e tarefas que fazem parte do processo de *software* se encontram dentro de um **modelo** que define a sua relação quer com o processo como entre si. Um modelo geral para a engenharia de *software* define 5 atividades: comunicação, planeamento, modelação, construção e lançamento do *software*. Além das atividades anteriormente mencionadas, são aplicadas ao longo do processo atividades como o controlo do projeto, gestão de riscos, garantia de qualidade, etc. [5].

Cada um dos modelos de processo existentes descrevem um fluxo distinto do processo (como se organizam sequencialmente e cronologicamente as atividades, ações e tarefas). Entre os modelos de processo tradicionais mais conhecidos estão o modelo linear, em cascata, modelos evolutivos, entre outros [5].

Além dos modelos anteriormente mencionados existem modelos de processo modernos que se centram no desenvolvimento ágil de *software*. Entre os mais populares estão o Scrum e a Programação Extrema (XP).

É importante referir que o processo de desenvolvimento deste projeto foi baseado no **paradigma de prototipagem**, juntamente com o **modelo ágil XP**. O paradigma de prototipagem forma parte dos modelos evolutivos, os quais se caracterizam por permitir desenvolver versões cada vez mais completas do *software* [5].

II.3.1. Paradigma de Prototipagem

O paradigma de prototipagem é comumente usado nos projetos em que as suas funcionalidades e características são difíceis de identificar ou detalhar. Os protótipos são extremamente úteis para obter uma visão do produto sem funcionalidade e ajudam a compreender melhor o que deve ser feito nos casos em que os requisitos não estão bem definidos. Apesar de ser possível fazer protótipos como um modelo de processo isolado, o mais comum é utilizar esta técnica como parte de qualquer outro modelo de processo [5].

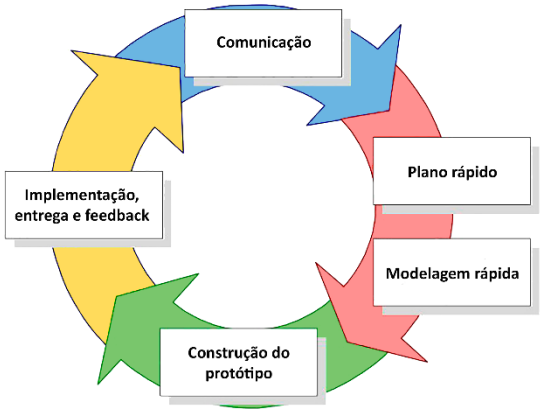


Figura 2 – O Paradigma de Prototipagem [5].

Como ilustrado na Figura 2, o paradigma de prototipagem começa pela comunicação, onde todos os participantes do projeto se reúnem para definir os objetivos gerais do *software* e identificar os requisitos já conhecidos. Depois é planeada rapidamente uma iteração para a criação do protótipo e feita uma modelação rápida do desenho, representando os aspetos do *software* que serão visíveis aos utilizadores finais [5].

A fase seguinte consiste em criar um protótipo (baseado no desenho rápido) para ser entregue e avaliado pelos participantes e receber *feedback* para melhorar os requisitos. A iteração ocorre à medida que o protótipo é melhorado até satisfazer as necessidades de todos os participantes e compreender tudo o que é necessário desenvolver. O protótipo serve como mecanismo para identificar os requisitos do *software* e avançar para a etapa de implementação [5].

II.3.2. Modelos de Processos Ágeis

Nos últimos tempos os modelos ágeis têm ganhado muita popularidade por serem uma ótima solução para projetos em que os seus requisitos são alterados constantemente e para projetos que devem ser desenvolvidos em curto prazo. Estes modelos centram-se em dar mais valor aos indivíduos, à colaboração com o cliente e ao desenvolvimento incremental do *software* com iterações muito curtas, mas mantendo sempre uma alta qualidade [6]. A seguir será exposto o modelo ágil XP, o qual foi seguido para o desenvolvimento deste projeto.

II.3.2.1. Programação Extrema (XP)

O modelo de processo ágil XP é um dos mais conhecidos e consiste basicamente em desenvolver um produto de boa qualidade em pouco tempo, focando o processo no trabalho em equipa e em satisfazer as necessidades do cliente. Este modelo é adequado para projetos com requisitos imprecisos, que estão continuamente a ser alterados e para equipas de desenvolvimento pequenas onde é mais fácil garantir a comunicação entre programadores, *managers* de projetos e clientes [6].

Na Programação Extrema (XP) existem 7 papéis importantes: os clientes que escrevem as histórias e testes de aceitação[1]; os programadores que criam o código do sistema, os *testers* encarregados de validar o *software*; os consultores que são pessoas externas à equipa e possuem conhecimento sobre um tema em específico para ajudar a resolver problemas; o *coach* que ajuda e ensina à equipa; o *tracker* que recolhe métricas e avisa quando encontra uma estimativa de risco; e finalmente o gestor, que coordena todo o projeto e é o vínculo entre clientes e programadores [6].

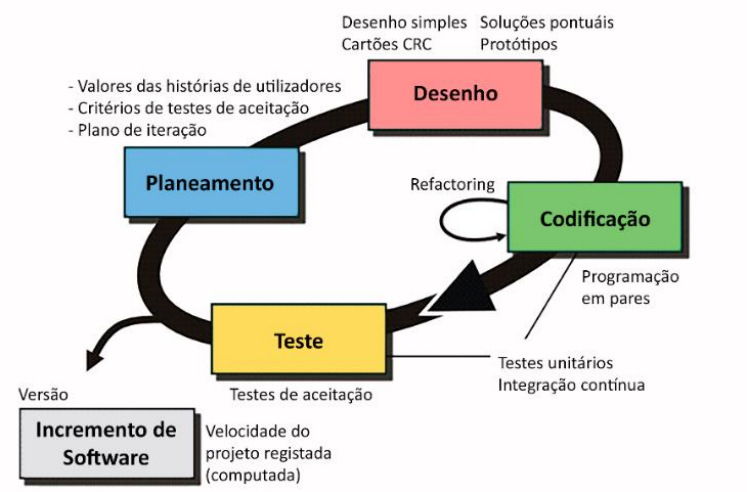


Figura 3 – O processo da Programação Extrema [5].

Por outro lado, o modelo XP ressalta uma série de valores que devem ser considerados e praticados durante o tempo de desenvolvimento do projeto, estes valores são a comunicação, simplicidade, *feedback*, valentia e respeito; e as atividades

[1] Execução de testes em todo o sistema para garantir se a funcionalidade da aplicação satisfaz a especificação [7].

que fazem parte do seu processo são o planeamento, o desenho, a codificação e os testes, como ilustrado na Figura 3 [5].

- O **planeamento** permite a toda a equipa entender o contexto do projeto e definir juntamente com o cliente as principais características e funcionalidades requeridas no *software*, que são descritas na forma de "histórias do utilizador". Cada história é criada pelo cliente e diz respeito a uma funcionalidade, onde o cliente também estabelece uma prioridade e a equipa avalia o seu custo de desenvolvimento. São também definidas as iterações a serem realizadas e as suas respetivas datas de entregas [5].

- A atividade de **desenho** foca-se em criar desenhos o mais simples possível e permite guiar a implementação de uma história de utilizador. Para projetos orientados a objetos é recomendado o uso de cartões CRC [2] (Classes, Responsabilidades e Colaboradores), e para projetos com alta complexidade pode ser necessária a elaboração e avaliação de protótipos para determinadas histórias [5].

  Um conceito central em XP é que o desenho pode ocorrer antes ou depois de iniciar a codificação, e neste modelo também existe o conceito de *refactoring*, que consiste em melhorar a estrutura interna do *software* sem alterar o seu funcionamento [5].

- Após criadas as histórias e feito o trabalho de desenho, o modelo sugere que a equipa de desenvolvimento inicie a atividade de **codificação** com a criação prévia de testes unitários[3] para cada uma das histórias, para serem posteriormente implementados no código desenvolvido [6].

  Durante a codificação é importante garantir uma comunicação ativa com o cliente para assim obter todos os detalhes necessários para o correto desenvolvimento do *software*. Além disso, XP propõe a programação em pares,

[2] Mecanismo eficaz para pensar em *software* num contexto orientado a objetos, identificam e organizam as classes que são relevantes [5].
[3] Um teste unitário consiste em testar uma pequena unidade de uma funcionalidade em particular [7].

onde dois programadores trabalham em conjunto no mesmo computador para minimizar os erros e criar um *software* de melhor qualidade [6].

- Ao realizar a atividade de **testes** deve garantir-se que, antes do lançamento do projeto, todos os módulos do *software* passem os testes unitários previamente definidos. Além disso, deve ser corrigido imediatamente qualquer erro detetado e tomadas as precauções necessárias para posteriormente não ocorrerem problemas semelhantes.

  Por outro lado, existem também testes de aceitação que são criados com base nas histórias do utilizador e onde o cliente é o responsável por verificar se todos os resultados estão corretos e comprovar a implementação satisfatória das histórias [6].

Para finalizar, apresenta-se na Tabela 1 as vantagens e desvantagens de usar o modelo XP no desenvolvimento de *software* [6].

Tabela 1 – Vantagens e desvantagens do Modelo XP

PROGRAMAÇÃO EXTREMA (XP)	
Vantagens	**Desvantagens**
Criação rápida de *software* com alta qualidade.	Dificuldade na planificação do projeto e estabelecimento de custos e duração do mesmo.
Boa compreensão das necessidades dos utilizadores e requisitos dos clientes.	Não deve ser aplicado em projetos que requerem de uma equipa numerosa.
O cliente tem o controlo das suas prioridades.	Altos custos no caso de o projeto falhar.
Planeamento do projeto otimizado e organizado.	
Fomenta a comunicação entre todos os interessados do projeto.	
Facilidade nas alterações.	
Programação organizada e eficiente.	
Diminui o número de falhas do *software*.	

II.4. APLICAÇÕES *WEB*

Nos últimos tempos a tecnologia para o desenvolvimento *Web* tem evoluído significativamente, permitindo assim a criação de aplicações cada vez mais complexas e não apenas destinada à troca de informação. Uma Aplicação *Web* expande o conceito de *Website* ao adicionar funcionalidades ao sistema, estas funcionalidades permitem que os utilizadores possam executar ações envolvidas na lógica de negócio através de um *browser* (como o *Google Chrome*, *Mozilla Firefox*, *Internet Explorer*, etc.).

Nesta altura é importante fazer uma distinção entre os termos Aplicações *Web* e *Websites*, embora os dois termos estejam associados à presença de informação disponível em médios digitais fazendo uso da Internet, a principal diferença entre estes conceitos encontra-se na origem dos dados que são expostos, nos *Websites* estáticos o conteúdo é um ficheiro ou documento pré-formatado. Já no caso das aplicações *Web*, o conteúdo apresentado é construído de forma dinâmica em função da interação do utilizador, com dados provenientes das bases de dados, onde ficam armazenados os dados que são processados pelos servidores *Web*. Estes servidores enviam os dados que são solicitados aos dispositivos dos utilizadores [8].

II.4.1. A Arquitetura Cliente-Servidor

As aplicações *Web* estão baseadas na **arquitetura Cliente-Servidor**, esta arquitetura segue o princípio fundamental de que todos os utilizadores devem poder visualizar, através de um *browser* (também conhecido como navegador *Web*), a mesma informação independentemente da plataforma de *software* e *hardware* utilizada [8].

O processamento da informação nesta arquitetura é dividido em módulos ou processos, o módulo chamado Cliente é onde se encontra o programa (o *browser*). Este solicita os dados ao módulo Servidor e apresenta-os aos utilizadores. O Servidor pode atender diferentes clientes bem como outros servidores e é responsável pelo processamento da informação e pelo envio dos resultados dos pedidos aos clientes [8].

Com base nesta arquitetura, é apresentado um esquema onde se encontra representado o fluxo de informação entre o cliente e o servidor no momento em que o

utilizador acede à aplicação através do *browser* (cliente), ou seja, o pedido inicial que ocorre entre o cliente e o servidor para mostrar à aplicação *Web* aos utilizadores.

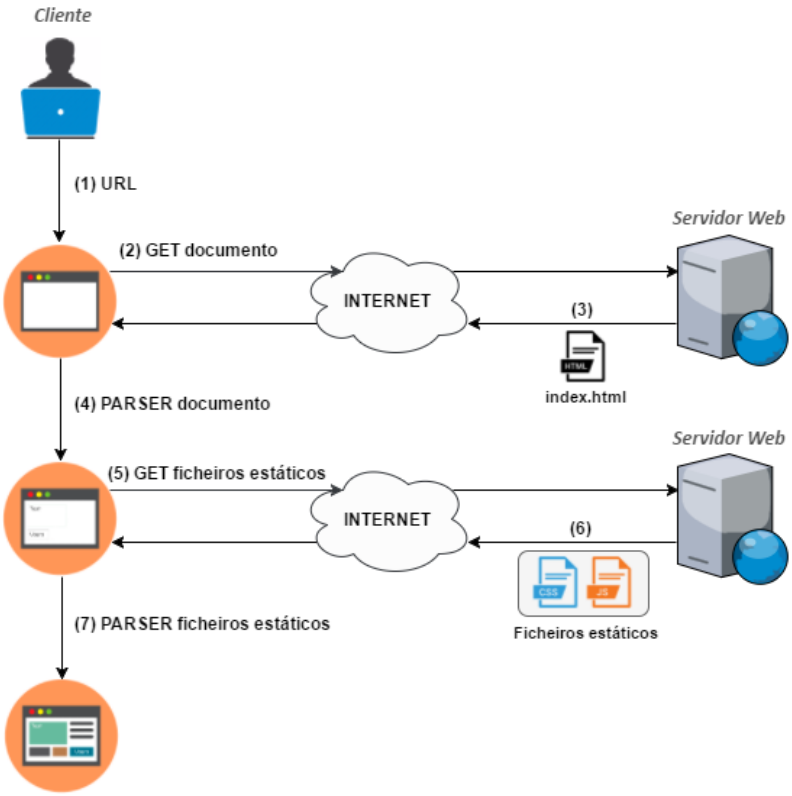


Figura 4 – Fluxo de informação entre o Cliente e o Servidor.

Com se pode visualizar na Figura 4, o utilizador acede inicialmente à aplicação *Web* ingressando a respetiva URL no *browser* que se encontra no lado do cliente (1). Em seguida, o *browser* envia um pedido ao servidor (2) solicitando o documento que inicia à aplicação (chamado por convenção *index*) e este pedido é enviado através da Internet utilizando os protocolos de comunicação HTTP ou HTTPS.

O servidor responde (3), devolvendo o documento solicitado para o *browser* que recebe e interpreta o seu conteúdo para depois ser apresentado (4). Nesta fase são carregados no *browser* os conteúdos e elementos da aplicação que se encontram definidos neste ficheiro, como por exemplo, o HTML puro com os textos e botões da página principal da aplicação.

Se o *browser* encontra referências a ficheiros estáticos neste documento, por exemplo, *links* que chamam ao CSS ou *scripts* para incluir na aplicação conteúdo em JavaScript, voltará a realizar um pedido ao servidor solicitando estes ficheiros (5).

Quando o servidor devolve o conteúdo dos ficheiros estáticos (6) estes são interpretados pelo *browser* e finalmente é apresentada ao utilizador a aplicação completamente carregada com os respetivos estilos CSS e *scripts* (7).

Por outro lado, quando o utilizador interage com a aplicação *Web*, podem surgir novos pedidos ao servidor para obter ou alterar as informações que se encontram nas bases de dados. Estes pedidos são realizados de forma transparente para o utilizador. Na Figura 5 é apresentado um esquema que mostra o fluxo básico de um pedido de informação ao servidor *Web*.

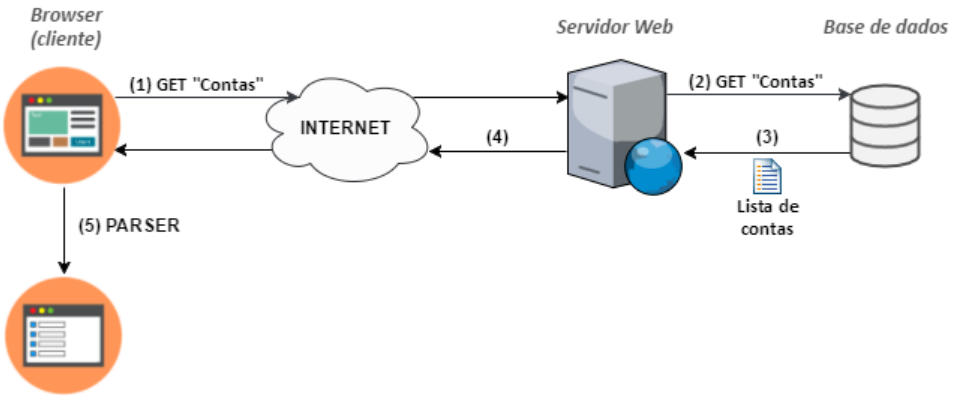


Figura 5 – Fluxo básico de um pedido de informação ao Servidor *Web*.

No caso deste exemplo, ao clicar num botão para visualizar uma lista de contas, o *browser* encarrega-se de fazer o pedido da informação (1) ao servidor através de Internet (usando algum dos protocolos de comunicação). O servidor *Web* recebe o pedido e por sua vez acede à base de dados para obter os dados armazenados que são necessários (2 e 3). Finalmente, o servidor processa e, caso não existir nenhuma falha, envia de volta esta informação ao *browser* (4) que será o encarregado de apresentar a informação obtida ao utilizador (5).

Na figura anterior foi exemplificado o fluxo de execução para um pedido de informação ao servidor, o que ocorreria da mesma forma se o *browser* solicitasse inserir, editar ou eliminar os dados.

É de salientar que uma aplicação *Web* pode possuir vários clientes e também pode requerer vários servidores. Por outro lado, existem casos em que é adicionada uma camada intermédia na arquitetura cliente-servidor. Esta camada tem o nome de *Middleware*.

O *Middleware* é um módulo intermediário que atua como ponte entre o Cliente e o Servidor, permitindo a comunicação entre estes módulos, pelo que é executado em ambas partes. A existência do *middleware* permite a independência dos clientes e servidores, facilitando a inter-relação entre eles e evitando a dependência de tecnologias [9].

II.4.2. O *Front-end* e *Back-end* das Aplicações *Web*

No desenvolvimento de aplicações *Web* existem dois conceitos que são considerados fundamentais: o *front-end* e *back-end*. O ***front-end*** é a parte da aplicação *Web* que se dedica a criar uma representação visual da aplicação e de interagir com os utilizadores [10], em outras palavras, trata dos componentes da aplicação que se encontram no lado do cliente e que são apresentados ao utilizador através do *browser*.

No outro extremo (lado do servidor) está o ***back-end***, que trata da lógica das aplicações *Web* e geralmente inclui um servidor *Web* que aloja a aplicação e comunica com as bases de dados. Encarrega-se, essencialmente, da manipulação dos dados, processa a entrada de dados provenientes do *front-end* e devolve uma resposta dos pedidos feitos pelos utilizadores [10], [11].

Quando um utilizador acede a uma aplicação *Web* através da sua interface gráfica, o *front-end* recolhe os dados relevantes para que o *back-end* os processe. Esta informação é verificada através das bases de dados no *back-end*, para depois ser apresentada de forma entendível ao utilizador. De maneira conjunta o *front-end* e *back-end* interagem para resolver as necessidades dos utilizadores [10].

A ideia é manter uma separação das diferentes partes da aplicação *Web* com o fim de obter um melhor controlo sobre a aplicação [11]. Por este motivo, hoje em dia, muitas das equipas de desenvolvimento *Web* encontram-se divididas em 3 áreas especializadas: a equipa de *front-end*, de *back-end* e os *designers*.

Como se ilustra na Figura 6, os desenvolvedores *front-end* trabalham em conjunto com os *designers*. Estes últimos tratam essencialmente da criação de todos os elementos visuais que serão apresentados na interface do utilizador.

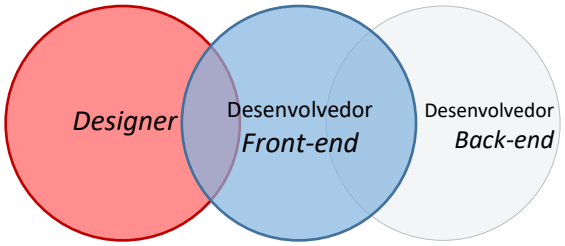


Figura 6 – Relação entre *Front-end*, *Back-End* e *Design* [12].

A equipa de *front-end* encarrega-se de estruturar e organizar a aplicação *Web*, preocupa-se pela interface do utilizador, dá formato aos conteúdos, cria as páginas, menus, botões e todos os elementos que o utilizador pode visualizar e com os quais pode interagir, e manipula os dados obtidos do *back-end*. Utiliza as tecnologias que correm no lado do cliente como HTML, CSS e JavaScript, que são as únicas que podem ser interpretadas pelos *browsers*.

Diretamente relacionada com a equipa de *front-end* encontra-se a equipa *back-end*, a qual em termos gerais, se encarrega de manipular todos os dados que chegam do *front-end* e do seu processamento; são também responsáveis pela criação de APIs (siglas de "*Application Programming Interface*") para que os dados possam ser consumidos no *front-end* e de garantir a segurança das aplicações. Utilizam tecnologias que permitem a construção das aplicações no lado do servidor, como por exemplo, as linguagens Java, PHP e Node.js, além de tecnologias para tratar das bases de dados como o MySQL, Oracle e MongoDB.

II.5. SERVIÇOS *WEB*

Na atualidade, a maioria das grandes aplicações utilizam serviços *Web* (em inglês, *Web services*) para que distintas aplicações de *software*, desenvolvidas em linguagens de programação diferentes e executadas sobre qualquer plataforma, possam trocar dados em rede [13].

Os **Serviços *Web*** são um conjunto de protocolos e estândares que expõem métodos para consultar, inserir, atualizar ou eliminar informação. A informação de um pedido e de uma resposta do serviço *Web* é feita com mensagens, através de protocolos de comunicação. O serviço *Web* processa esta informação e interage com outros

componentes, como por exemplo, as bases de dados, aplicações para dispositivos móveis, dispositivos físicos ou até outros serviços *Web* [14].

Os programas no lado do Cliente podem utilizar APIs para se comunicarem com os serviços *Web*. De um modo geral, uma **API** expõe um conjunto de dados e funções que facilitam as interações entre programas e permitem a troca de informações entre eles. Como apresentado na Figura 7, uma API é a interface de um serviço *Web*, a qual se encontra diretamente escutando e respondendo às petições dos clientes [15].

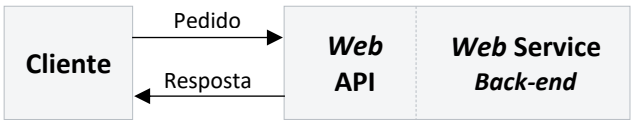


Figura 7 – *Web* API [15].

Uma API e um Serviço *Web* ambos servem como um médio de comunicação, a única diferença é que os serviços *Web* facilitam a interação entre duas máquinas através da rede e uma API atua como uma interface entre duas aplicações diferentes para que se possam comunicar entre si [16].

É importante salientar que os serviços *Web* oferecem um componente de comunicação estândar, proporcionam interoperabilidade e extensibilidade entre diferentes aplicações, independentemente da tecnologia utilizada [14].

II.5.1. REST

Nos últimos anos tem-se vindo a popularizar um estilo arquitetural de *software* chamado **REST (*Representational State Transfer*)**, que se refere a um conjunto de princípios para o desenho de arquiteturas na rede que resumem como os recursos são definidos, por outras palavras, definem a interação entre os distintos componentes e as regras que devem seguir. O protocolo mais utilizado que cumpre esta definição é o HTTP [17], [18].

Em REST, são nomeados "**recursos**" todos os elementos de informação, como as páginas de um *Website* ou aplicação *Web*, assim como as informações de uma secção e os ficheiros. O recurso é toda a informação a qual se quer aceder, modificar ou eliminar, independentemente do seu formato [19].

É considerada REST qualquer interface entre sistemas que usem HTTP para obter dados ou gerar operações sobre os dados em todos os formatos possíveis, como XML (*eXtensible Markup Language*) ou JSON (*JavaScript Object Notation*). REST é uma alternativa a outros protocolos estândares para troca de dados como o SOAP (*Simple Object Access Protocol*), que dispõem de uma grande capacidade, mas também de muita complexidade. Por tal motivo, hoje em dia é recomendado o uso de REST que resulta uma solução mais simples para a manipulação dos dados [20].

Por outro lado, é importante destacar quais as restrições presentes neste estilo arquitetural, sendo estas [15], [17]:

- **Arquitetura Cliente-Servidor:** a restrição principal presente em REST é que deve ser utilizada a arquitetura cliente-servidor (exposta no capítulo II.4.1), onde o cliente e o servidor podem ser implementados de forma independente, usando qualquer linguagem ou tecnologia desde que estejam em conformidade com a *interface uniforme* da *Web*, o que permite uma flexibilidade muito alta em todos os sentidos.

- **Interface uniforme:** as interações entre os componentes da *Web*, ou seja, dos clientes, servidores e intermediários da rede, dependem da uniformidade das suas interfaces. O pretendido é que a interface de comunicação não dependa nem do cliente nem do servidor, pelo que esta restrição garante que independentemente de quem execute os pedidos ou quem os receba, o importante é que ambos componentes cumpram com uma interface previamente definida. Se algum dos componentes não cumpre com os padrões estabelecidos, o sistema de comunicação não funcionará. Esta restrição simplifica o protocolo e aumenta a escalabilidade e rendimento do sistema.

- **Sistema em camadas:** as restrições de sistema em camadas permitem que intermediários baseados em rede, como *proxies* [4] e *gateways* [5], sejam

[4] Um *proxy* é um programa ou dispositivo que realiza uma tarefa de acesso a Internet em vez de outro computador, é um ponto intermediário entre o *browser* e o servidor [21].
[5] Um *gateway* (porta de ligação) é um ponto de uma rede que atua como ponto de entrada a outra rede [22].

implementados de forma transparente entre um cliente e um servidor usando a *interface uniforme*. O sistema não deve forçar o cliente em saber por quais camadas se transmite a informação, o que permite que o cliente conserve a sua independência. Estes intermediários são comumente usados para a implementação do balanceamento de carga, respostas em *cache* e para o cumprimento das políticas de segurança.

- ***Cache*:** a restrição de *cache* implica que um servidor *Web* deve colocar em *cache* todos os dados de cada resposta dos pedidos, o que ajuda a reduzir a latência do cliente, a aumentar a disponibilidade e a confiabilidade de uma aplicação e controlar a carga de um servidor *Web*.

- **Sem estado (*stateless*):** a restrição *stateless* diz que não é necessário que um servidor *Web* armazene o estado das aplicações cliente. Como resultado, cada cliente deve incluir todas as informações contextuais que considera relevantes em cada interação com o servidor. Além disso, os servidores pedem aos clientes para gerir a complexidade de comunicar o seu estado. Desta forma podem atender a um número muito maior de clientes, melhorando a escalabilidade do estilo arquitetural.

- **Código em demanda (*Code-On-Demand*):** a restrição *Code-On-Demand* é a única opcional dentro do REST e refere-se à capacidade de modificar o comportamento do cliente e estender as suas funcionalidades enviando desde o servidor os fragmentos de código (*scripts*) que o cliente deve processar [23]. As aplicações *Web* que utilizam JavaScript são um exemplo de uso, o JavaScript é descarregado no cliente (*browser*) desde o servidor junto com o HTML, o que permite melhorar a interface do utilizador e interagir com o servidor. Desta forma, o cliente não está limitado a apenas receber dados [23], [24]. A utilização do *Code-On-Demand* aumenta a extensibilidade do *software*, no entanto, tende a estabelecer um acoplamento de tecnologias entre os servidores e os seus clientes, já que o cliente deve ser capaz de compreender e executar o código que descarrega sob demanda desde o servidor [25].

Todas as restrições anteriormente apresentadas permitem que o estilo arquitetural REST disponha de um conjunto de atributos de qualidade, como sendo a flexibilidade, performance, escalabilidade, segurança, disponibilidade, confiabilidade e extensibilidade do *software*.

II.5.1.1. API REST

O estilo arquitetural REST é comumente aplicado no desenho de APIs para serviços *Web* modernos, as APIs que estão de acordo com este estilo têm o nome de **APIs REST** e utilizam o HTTP como protocolo de comunicação [15].

No momento de criar uma API REST, é importante ter em conta os seguintes princípios básicos:

- **Utilizar nomes de recursos apropriados:** os nomes de recursos apropriados fornecem o contexto para um determinado pedido, o que aumenta a compreensão da API. Os recursos são vistos hierarquicamente através dos seus nomes URIs, os quais permitem identificá-los de forma única [26].

  Existem regras básicas que devem ser seguidas ao dar nomes às URIs dos recursos, entre as mais destacadas encontram-se [19]:

  1. Os nomes não devem implicar uma ação, portanto, deve ser evitado o uso de verbos. Exemplos:

     - Nome incorreto: */contas/001/editar* (errado pelo verbo "editar").

     - Nome correto: */contas/001* (para o recurso "conta" com o identificador "001" esta URI é correta independentemente da ação que se pretende realizar).
  2. As URIs devem ser únicas, não pode existir mais do que uma URI para identificar o mesmo recurso.
  3. Devem ser independentes do formato utilizado. Exemplos:

     - Nome incorreto: */contas/001.pdf* (errado pela indicação do uso do formato PDF).

     - Nome correto: */contas/001* (não indica o formato utilizado).
  4. Devem manter a hierarquia lógica. Exemplos:

- Nome incorreto: */contas/001/cliente/005* (não segue a hierarquia lógica).

- Nome correto: */clientes/005/contas/001* (segue a hierarquia, lê-se: o recurso "conta" com o identificador "001" do cliente "005").

5. Para filtrar, ordenar, paginar ou pesquisar informação de um recurso, deve-se realizar uma consulta sobre a URI utilizando parâmetros HTTP, em vez de incluir estes filtros diretamente na URI.

   - Nome incorreto: */contas/ordem/desc/data-desde/2017/pagina/1* (errado porque o nome obriga a utilizar uma URI distinta para filtrar, ordenar ou paginar o mesmo recurso).

   - Nome correto: */contas?data-desde=2017&ordem=DESC&pagina=1* (O símbolo "?" permite filtrar as contas com base nos parâmetros recebidos a seguir do símbolo).

- **Utilizar métodos HTTP:** as operações CRUD estão fortemente ligadas ao REST, esta série de métodos (GET, POST, PUT, DELETE e PATCH) facilitam a gestão das informações nas bases de dados e aumenta a clareza dos pedidos [26].

Na Tabela 2 apresentam-se os métodos HTTP e as ações que estes realizam, para além de um exemplo de URI usando nomes apropriados [26], [27]:

Tabela 2 – Métodos HTTP

Método HTTP	CRUD	Ação	Exemplo de URI
GET	Ler	Ler um recurso específico (através de um identificador) ou uma coleção de recursos.	*http://www.exemplo.com/clientes/id* **OU** *http://www.exemplo.com/clientes*
POST	Criar	Criar um novo recurso.	*http://www.exemplo.com/clientes*
PUT	Atualizar/ Substituir	Atualizar um recurso específico (através de um identificador) ou uma coleção de recursos. Também pode ser usado para criar um recurso específico se o identificador do recurso é já conhecido.	*http://www.exemplo.com/clientes/id*

PATCH	Atualizar/ Modificar	Atualiza parcialmente o recurso. O pedido PATCH só tem de conter as alterações para o recurso, não o recurso completo.	*http://www.exemplo.com/ clientes/id*
DELETE	Eliminar	Eliminar um recurso específico através de um identificador.	*http://www.exemplo.com/ clientes/id*

Os métodos PUT, PATCH e DELETE são considerados idempotentes, ou seja, os resultados devem ser os mesmos independentemente das vezes que seja realizado o mesmo pedido. No entanto, o método GET é comumente mencionado como um método seguro ou *nullipotent*, porque as chamadas ao método não devem ter efeito no lado do servidor [28].

Por outro lado, as URIs para cada método HTTP possuem sempre os mesmos nomes independentemente do método a utilizar, isto facilita a compreensão da API e a leitura da URI.

- **Utilizar códigos de resposta HTTP para indicar o estado:** o protocolo HTTP define um conjunto de códigos de estados que devem ser devolvidos pelas APIs, estes códigos permitem saber se uma determinada operação foi realizada com sucesso ou com falha e qual o seu estado em específico [26].

- **Tipos de formato de respostas:** as APIs REST devem enviar a informação usando tipos de formato válidos. Atualmente o formato de resposta mais usado para o envio e receção dos dados é o JSON, mas existem outros formatos como o XML [28].

- **Hipermédia:** no caso de uma API REST o conceito de hipermédia explica a capacidade que tem uma interface para desenvolvimento de aplicações de proporcionar ao cliente os *links* adequados para executar ações sobre os dados. As APIs REST seguem o princípio HATEOAS (das suas siglas em inglês, *Hypermedia As The Engine Of Application State*) que define que cada vez que é realizado um pedido ao servidor e é devolvida uma resposta, parte da informação contida na

resposta são os *links* de navegação associados a outros recursos do cliente [20]. Isto resulta útil por exemplo, para que o cliente não tenha que conhecer as URLs dos recursos, o que facilita a sua manutenção além de garantir independência entre o cliente e o servidor [19].

Na atualidade, a maioria dos grandes projetos e aplicações existentes, como *Twitter* e *YouTube*, dispõem de uma API REST para permitir a criação de serviços profissionais a partir do seu *software*. Utilizam REST porque é considerado o estândar mais lógico, eficiente e habitual na criação de APIs para serviços *Web* [20].

As APIs REST têm a vantagem de serem independentes do tipo de plataforma ou linguagem utilizada, o que oferece uma grande liberdade no momento de alterar ou testar novos entornos dentro do desenvolvimento. Com uma API REST, é possível ter servidores em PHP, Java, Node.js ou Python, o único indispensável é que as respostas aos pedidos sejam feitas sempre utilizando o formato de resposta estabelecido, o qual normalmente é JSON ou XML [20].

II.6. MODELO-VISTA-CONTROLADOR (MVC)

Um dos padrões de desenho mais utilizados e eficientes para o desenvolvimento de *software* é o denominado **Modelo-Vista-Controlador (MVC)**. Um padrão de desenho é uma solução conhecida para um problema recorrente, ou seja, um conceito e uma estratégia a seguir para resolver uma situação. Os padrões arquiteturais como o MVC permitem estabelecer quais componentes terá o sistema a desenvolver e a maneira em que estes se relacionam [29].

O MVC permite simplificar o desenvolvimento e manutenção das aplicações ao separar o código da visualização da informação (interface do utilizador) do código que trata da manipulação dos dados e da lógica da aplicação. Segundo este padrão, cada aplicação é dividida em três componentes lógicos: o Modelo, a Vista e o Controlador, a seguir apresentados [30], [31].

- **Modelo:** o Modelo implementa o processamento dos dados e é o responsável pela lógica de negócio da aplicação, ou seja, possui o código principal e realiza o trabalho interno da aplicação, além de encapsular o acesso às bases de dados. O

Modelo não precisa de saber sobre o que se encontra nas Vistas e nos Controladores, só precisa que lhe sejam passados os dados necessários para retornar o resultado.

- **Vista:** a Vista é a que interage com o utilizador, controla a aparência dos dados e fornece recursos para recolher os dados provenientes dos utilizadores, além disso, recebe os dados fornecidos pelos Controladores e exibe-os na interface gráfica. As tecnologias encontradas nas Vistas são HTML, CSS e JavaScript.
- **Controlador:** o Controlador é o intermediário entre a Vista e o Modelo, combina o estilo da Vista com a funcionalidade do Modelo. É responsável por recolher os dados de entrada provenientes das Vistas e de chamar aos Modelos para executar as suas respetivas funcionalidades; depois interpreta os resultados retornados para que possam ser renderizados pelas Vistas. Além disso, gere o fluxo da aplicação e trata de todas as exceções que ocorrem.

A continuação, na Figura 8 apresenta-se um diagrama com os componentes deste padrão arquitetural e a comunicação entre eles.

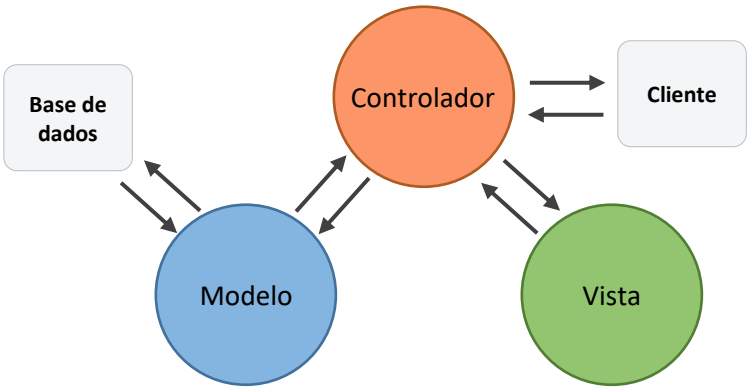


Figura 8 – Padrão arquitetural MVC [32].

A utilização do padrão MVC permite, entre outras coisas, a separação de conceitos e a criação de um código que facilita a sua reutilização; estas características melhoram o desenvolvimento das aplicações e a sua posterior manutenção, além de beneficiar o fluxo de trabalho das equipas.

II.7. ARQUITETURA BASEADA EM COMPONENTES

A complexidade dos sistemas de *software* leva a que os desenvolvedores procurem reutilizar *software* já existente. Uma **arquitetura baseada em componentes** centra-se na decomposição do desenho em componentes lógicos individuais, divide o problema em subproblemas, proporcionando um alto nível de abstração. Um componente é um conjunto modular, portátil, substituível e reutilizável de funcionalidades bem definidas que está destinado a interagir com outros componentes [33].

Para facilitar o raciocínio e reutilização, os componentes podem ser divididos em duas categorias: **Dumb** e **Smart** [34].

Os componentes **Dumb** estão preocupados com a aparência, não possuem dependências com o resto da aplicação, simplesmente adquirem as entradas de dados através do seu componente ascendente e não sabem de onde vêm e para onde se dirigem os dados. Este enfoque faz que o código seja reutilizável, porque permite integrar os componentes Dumb em qualquer parte do código [34], [35].

Os componentes **Smart** preocupam-se com o funcionamento, fornecem dados e comportamentos aos componentes Dumb ou a outros componentes Smart. Estes têm a responsabilidade de contactar os serviços e de passar os dados que adquirem aos componentes Dumb [34], [35].

Além de facilitar a reutilização de componentes, a abordagem baseada em componentes fornece um grande número de benefícios, como por exemplo, a diminuição do acoplamento entre componentes, que permite alterar e testar cada componente por separado, simplificando a realização de testes e a manutenção do sistema. Além disso, os componentes podem ser criados e melhorados continuamente com o passar do tempo, o que aumenta a qualidade do *software*.

Por outro lado, a desvantagem principal que pode ser considerada nesta abordagem é a comunicação entre os componentes, porque nesta arquitetura é criada uma árvore de componentes que cresce verticalmente (onde existem componentes pais, filhos e assim sucessivamente), e quando a aplicação é muito complexa a

comunicação entre estes componentes pode ser difícil de gerir. Contudo, atualmente as *frameworks* JavaScript já integram soluções para melhorar este problema.

Outra desvantagem que apresenta esta arquitetura é a complexidade que pode surgir ao gerir aplicações de grande escala quando são divididas em muitos componentes pequenos, pelo que o recomendado é realizar um bom planeamento de como será dividida a aplicação em componentes antes de iniciar a fase de implementação.

Para concluir esta secção, é importante destacar que a abordagem baseada em componentes permite criar aplicações extremamente modulares, altamente testáveis e fáceis de manter e de realizar o *deployment*. Pelas vantagens aqui referidas esta foi a arquitetura para desenvolvimento do lado do Cliente utilizada neste projeto.

II.8. FERRAMENTAS MAIS UTILIZADAS

Para o desenvolvimento de qualquer projeto de *software* é importante avaliar quais as ferramentas disponíveis no mercado que permitem agilizar e otimizar o processo de desenvolvimento, para assim escolher as que melhor se adequam aos requisitos do projeto e às condições da equipa e da empresa.

Serão apresentadas em seguida algumas das ferramentas existentes para a elaboração de protótipos de baixa e de alta fidelidade, os *frameworks* e livrarias JavaScript mais conhecidos para desenvolvimento de aplicações *Web front-end* e finalmente serão avaliadas as opções disponíveis para o controlo de versões e alojamento dos projetos de *software*.

II.8.1. Ferramentas para Prototipagem

Para a fase de prototipagem foram consideradas principalmente as ferramentas de utilização gratuita que permitem a elaboração de protótipos de baixa e de alta fidelidade. A continuação apresenta-se uma breve descrição de algumas das ferramentas encontradas para prototipagem de baixa fidelidade, nomeadamente Moqups, MockFlow e Axure.

- **Moqups** [36] é um *Website* desenhado em HTML5 que apresenta uma quadrícula onde podem ser adicionados elementos que simulam as aplicações ou páginas *Web*. Destaca-se pela sua simplicidade e possui já incorporada uma livraria com os principais objetos gráficos e formas predeterminadas para facilitar a criação dos protótipos, porém, na sua versão gratuita a quantidade de objetos no projeto é limitada e só permite a criação de um único projeto por conta registada.

  Esta ferramenta não precisa de *plugins* ou *software* extra para trabalhar e pode ser acedida a partir de qualquer dispositivo. Permite também a colaboração de vários participantes no projeto e a inserção de comentários. Além disso, é possível adicionar eventos aos elementos do protótipo para simular a interação do utilizador e a navegação entre os diferentes ecrãs.

  Na sua versão gratuita, o projeto criado pode ser acedido publicamente e é possível partilhá-lo através de um *link*. Já na sua versão paga, permite a criação de projetos privados e a exportação nos formatos PDF ou PNG.

- **MockFlow** [37] oferece muita liberdade na criação de protótipos e possui uma interface minimalista. Tem uma grande variedade de pacotes com elementos modernos e formas predeterminadas. Estes elementos encontram-se organizados por categorias (elementos para *Web*, para aplicações iPhone e Windows 10, gráficos, mapas, entre outros) o que permite encontrar facilmente os objetos necessários durante a elaboração do protótipo.

  Em semelhança à ferramenta Moqups, é possível partilhar o projeto através de um *link* de acesso público, adicionar comentários e interação aos elementos. Adicionalmente, MockFlow oferece uma aplicação para escritório com as mesmas características que se encontram na sua versão *Web*, está disponível para as plataformas Mac e Windows.

  MockFlow conta com bastantes opções de exportação na versão gratuita, nomeadamente, exportação nos formatos HTML5, PDF, PowerPoint, PNG e documentos Word. Esta versão está limitada à criação de um projeto com até 4 páginas e com a colaboração de até 2 pessoas.

- **Axure** [38] é uma aplicação muito completa que permite a realização de protótipos simples e complexos. Uma das suas maiores fortalezas é a sua interatividade, porque possui uma grande variedade de eventos para simular o comportamento das interfaces.

  Incorpora uma livraria com os clássicos elementos gráficos utilizados no desenho de páginas *Web* e é possível encontrar em Internet diversos módulos, ícones e uma série de elementos criados pela comunidade que podem ser importados e utilizados no projeto. Além disso, oferece a possibilidade de utilizar vistas adaptativas, o que permite criar de forma simples um desenho que se adapte aos diferentes tamanhos de ecrãs dos dispositivos.

  Com Axure também é possível trabalhar em colaboração com outras pessoas e conta com o serviço AxShare, que permite subir o protótipo a um servidor de Axure, o que facilita a partilha do desenho com os clientes, a equipa ou qualquer pessoa interessada; esta característica é sumamente útil para a realização de testes aos protótipos. Adicionalmente é possível a exportação nos formatos HTML, Word, CSV e alguns formatos de imagem como Bitmap, PNG, JPG e GIF.

  Esta ferramenta encontra-se disponível para as plataformas Windows e Mac, e a sua versão gratuita é limitada a 30 dias. Para estudantes e professores Axure oferece uma licença gratuita [39].

Depois de apresentadas algumas das ferramentas para a realização de protótipos de baixa fidelidade, serão agora expostas alguma das ferramentas existentes para o desenvolvimento de protótipos de alta fidelidade, nomeadamente InVision e Framer.

- **InVision** [40] é uma ferramenta muito completa e interativa para a criação de protótipos para *Web* ou dispositivos móveis, pelo que é possível testar facilmente o desenho, as funcionalidades e a experiência do utilizador. A ideia principal é importar as imagens do desenho final da interface para adicionar a interação com esta ferramenta.

  Permite sincronizar os ficheiros criados em ferramentas de desenho como Photoshop ou Sketch e atualizar os protótipos em tempo real dentro destes programas. Além disso, é possível partilhar os projetos e adicionar

comentários e discussões, pelo que facilita o fluxo de trabalho em equipa. Com a funcionalidade de *LiveShare* a colaboração pode ocorrer em tempo real [41].

A sua versão gratuita permite a criação de um único protótipo, com um número ilimitado de ecrãs e colaboradores. É possível exportar nos formatos HTML e PDF, e também partilhar o protótipo através de um *link*.

- **Framer** [42] é ideal para desenvolver protótipos altamente interativos. Com esta ferramenta os protótipos são construídos com CoffeeScript (uma maneira mais simples e legível de escrever JavaScript) e permite realizar animações e interações completas para aplicações *desktop* e móveis [43].

  Possui um entorno de desenvolvimento - **Framer Studio** - que oferece um conjunto de código já construído para as animações básicas e apresenta uma visualização em tempo real do protótipo. Ao igual que InVision, os ficheiros criados em aplicações de desenho (Photoshop ou Sketch) podem ser importados conservando as suas características [43].

  O facto de ser um protótipo construído em JavaScript permite fazer tudo o que é possível nos *browsers*, como por exemplo, incorporar dados em tempo real. Por outro lado, os protótipos podem ser partilhados através de *links* e exportados em formato *Web* [44]. Framer Studio encontra-se disponível apenas para Mac, sem custos associados por um período de teste de 14 dias, mas a *framework* Framer.js é gratuita e de código aberto, pelo que pode ser utilizada em qualquer editor de código-fonte.

II.8.2. *Frameworks* e Livrarias de JavaScript

Quando o pretendido é realizar um produto que se adapte aos diferentes sistemas e dispositivos, o ideal é conseguir uma solução que possa ser comum a todos. Hoje em dia, existe um consenso no mundo da tecnologia para dar suporte a HTML e JavaScript; estas soluções *Web* baseadas em HTML e JavaScript competem com as soluções nativas e em termos de rendimento continuam em desvantagem. Além disso, as linguagens nativas têm maior facilidade de acesso às características específicas e funcionalidades dos dispositivos. Porém, com os avanços mais recentes, como a chegada do HTML5 e de diversas APIs e *frameworks* JavaScript, esta linguagem

estendeu-se e já pode ser considerada robusta, capaz de cumprir com as necessidades de grandes aplicações [45].

Nesta altura é importante fazer uma distinção entre os conceitos Livraria e *Framework*. Uma **livraria** proporciona uma série de funções que podem ser incorporadas no projeto para simplificar o desenvolvimento das tarefas mais complexas. Por outro lado, uma ***framework*** define uma estrutura completa onde será integrado o projeto, estabelece um esquema com as condições que devem ser seguidas e define a arquitetura da aplicação. As *frameworks* podem ser vistas como um conjunto de componentes genéricos personalizáveis que podem ser utilizados para construir uma aplicação concreta com um desenho reutilizável [46].

Entre as *frameworks* JavaScript mais conhecidas na atualidade para o desenvolvimento *front-end* de aplicações *Web* encontram-se as *frameworks* AngularJS e Angular 2. Existem também livrarias JavaScript tão amplas e com alto desempenho que podem ser usadas em vez de *frameworks* (ou em conjunto). Entre as mais usadas estão Vue.JS e ReactJS [47], [48]. A continuação apresenta-se as principais características destas *frameworks* e livrarias JavaScript.

- **AngularJS** [49] desenvolvida pela equipa de *Google*, converteu-se rapidamente na *framework* JavaScript do lado do cliente mais popular na atualidade, permite criar aplicações *Web* de uma única página (SPA, *Single Page Application*) e está especificamente pensada para projeto complexos.

  Esta *framework* é de código aberto e está baseada em jQuery Lite, uma variante da livraria jQuery, oferece ferramentas para a gestão de dependências, modelos, controladores e *routing*. A partir da sua versão 1.5 possibilita a utilização de componentes [50]. Além disso, uma das características mais ressaltantes deste *framework* é o "*Two-Way Data Binding"* que permite que as alterações realizadas nos Modelos sejam refletidas imediatamente nas Vistas.

- **Angular 2** [51] é comumente conhecida como a nova versão da *framework* AngularJS, mas na realidade trata-se de uma nova *framework* construída desde zero com o fim de substituir a AngularJS. Da mesma forma que AngularJS, esta *framework* é útil para criar aplicações *Web* do lado do cliente e mantém muitas

das suas características. Encontra-se baseada em componentes e no uso de classes, e faz uso do *superset* TypeScript, mas continua a ser possível o desenvolvimento das aplicações na linguagem JavaScript.

A diferença do AngularJS onde a renderização é feita no lado do cliente (no *browser*), uma das novidades mais interessantes no Angular 2 é a renderização universal, que permite renderizar as vistas no lado do servidor [52].

- No que diz respeito às livrarias JavaScript úteis para o desenvolvimento de aplicações *Web*, uma das que está a ganhar muita popularidade hoje em dia é a livraria **ReactJS** [53] desenvolvida pela equipa de *Facebook*. Permite criar aplicações de uma única página de forma muito eficiente e funciona quer no lado do cliente quer no lado do servidor, também promove a criação de componentes reutilizáveis [54].

  Esta livraria encarrega-se só da parte da vista da aplicação. Porém, *Facebook* recomenda uma arquitetura para modelar o resto da aplicação, chamada Flux [55]. Por outro lado, usa o *superset* JSX, que é uma extensão da linguagem JavaScript altamente recomendada pela sua clara sintaxe que facilita a programação em ReactJS, mas a sua utilização não é obrigatória, podendo o código ser escrito em JavaScript [56].

  ReactJS tem como característica principal que possui um DOM[6] virtual que contribui para a melhora da velocidade no carregamento da informação das aplicações, pois só são carregadas as alterações realizadas na página e não toda a informação [56]. Muitas das aplicações usadas na atualidade foram construídas utilizando esta livraria, como por exemplo, *Facebook*, *Netflix* e *Dropbox*.

- Outra livraria JavaScript existente é **Vue.js** [58], assim como ReactJS permite a construção de interfaces *Web* modernas e interativas, e possui a maioria das características observadas em ReactJS. Foi desenvolvida pela comunidade e a sua livraria principal centra-se apenas na camada de visualização, mas permite a

[6] DOM (*Document Object Model*) é uma interface de programação de aplicações (API) para documentos HTML e XML. Define a estrutura lógica dos documentos e a maneira em que é acedido e manipulado um documento [57].

construção de aplicações completas de uma única página quando é usada em combinação com outras ferramentas e livrarias complementares [59]. Oferece a renderização no lado do servidor (como o Angular 2 e ReactJS) e suporte opcional a JSX. Vue.js também possui um DOM virtual como ReactJS [60].

Além das *frameworks* e livrarias anteriormente apresentadas, é importante mencionar a popular livraria **jQuery** [61], esta é um dos clássicos no desenvolvimento em JavaScript e ainda bastante usada em muitos projetos *Web*. Apesar da existência de novas soluções muito mais extensas em características e funcionalidades, JQuery se destaca principalmente pela simplicidade que oferece para manipular o DOM, os documentos HTML e para adicionar interação à aplicação. Esta livraria também pode servir como complemento das *frameworks* JavaScript utilizadas na atualidade, conforme o tipo de projeto e as suas necessidades.

Finalmente, apresenta-se na Figura 9 um gráfico com as *frameworks* e livrarias para desenvolvimento no lado do cliente mais populares no ano 2016, onde Vue.js figura como a mais popular, seguida de ReactJS e Angular nas suas duas versões.

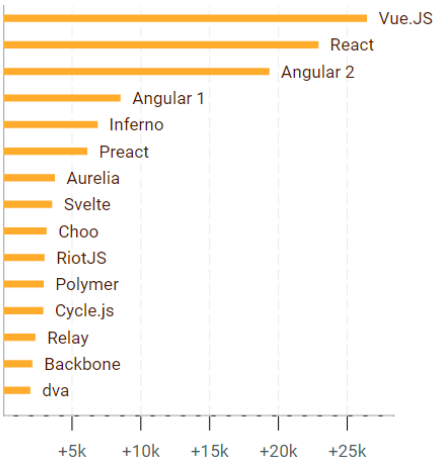


Figura 9 – *Frameworks* e livrarias *Front-end* mais populares no ano 2016 [62].

II.8.3. Sistema de Controlo de Versões

Para gerir o desenvolvimento de projetos de *software* agilmente, é considerada uma boa prática a utilização de um Sistema de Controlo de Versões (VCS) que permita registar as mudanças realizadas sobre um ficheiro ou conjunto de ficheiros ao longo do tempo, de modo que seja possível reverter o projeto a uma versão anterior ou comparar as alterações [63]. Além disso, resulta indispensável em projetos desenvolvidos em

equipa, porque simplifica a gestão de conflitos de ficheiros e melhora a velocidade de comparação de códigos escritos por diferentes programadores num mesmo projeto [64]. Isto facilita a elaboração de projetos de forma colaborativa e a centralização do código desenvolvido.

Das várias soluções existentes no mercado tem-se o **Subversion** [65], comumente conhecido como SVN, este é um VCS de código aberto desenvolvido pela empresa *Apache*. Utiliza um servidor centralizado para guardar todas as alterações e atualizações, isto permite que as pessoas em diferentes partes do mundo possam trabalhar no mesmo projeto, contudo, estes sistemas centralizados estão dependentes de ligação à Internet para funcionar, de outra maneira as alterações nunca poderiam ser refletidas no servidor central [66].

Para ultrapassar as desvantagens dos sistemas centralizados, surgiram os Sistemas de Controlo de Versões Distribuídos (DVCS), como o **Git** e o **Mercurial**. O **Git** [67] é um DVCS muito popular na atualidade; pelo facto de ser distribuído cada um dos membros da equipa pode manter uma cópia local do repositório completo e atualizar o código sem a necessidade de ligação à Internet, facilitando a sua autonomia [64].

Além do Git, também é bastante usado o DVCS **Mercurial** [68]; em termos práticos e de desempenho ambas ferramentas são muito parecidas [69].

II.8.4. Serviços de Alojamento de Repositórios de *Software*

Ao desenvolver uma aplicação, o mais recomendável é hospedar o código do projeto num Serviço de Alojamento de Repositórios de *Software*, para tal, existe na atualidade diferentes alternativas, sendo a mais popular o **GitHub** [70]. Esta plataforma está baseada em Git e é mantida pela comunidade. Permite um número ilimitado de repositórios e de colaboradores e está destinada principalmente a alojar projetos de código aberto. Na sua versão paga oferece repositórios privados [71], [72].

Além do GitHub, tem-se o **BitBucket** [73], desenvolvido pela empresa *Atlassian*; é compatível com Git e Mercurial e possui a maioria das características de GitHub, mas esta oferece gratuitamente repositórios privados ilimitados para um máximo de 5 colaboradores [72].

Outra alternativa é o **GitLab** [74], também baseada em Git, é um serviço de alojamento completamente gratuito muito semelhante ao GitHub, proporciona repositórios privados e permitir a instalação de servidores próprios [75].

II.8.5. Considerações Finais

Expostas algumas das ferramentas disponíveis no mercado para prototipagem e desenvolvimento de aplicações *Web*, será feita de seguida uma análise e comparação das suas características.

II.8.5.1. Análise das Ferramentas para Prototipagem

Em relação às ferramentas para a criação de protótipos de baixa fidelidade, as ferramentas expostas Moqups e MockFlow têm a vantagem de não estar limitadas no número de dias, são ideais para projetos pequenos, que não requerem a colaboração de muitas pessoas e de um vasto número de objetos (que são limitados na ferramenta Moqups) nem de muitas páginas (que são limitadas em MockFlow). Ambas ferramentas podem não ser a melhor alternativa para projetos complexos.

Por outro lado, MockFlow destaca-se por possuir uma livraria muito completa de elementos que facilitam o trabalho de desenho, e na sua versão gratuita permite diferentes formatos de exportação, coisa que não é possível com a ferramenta Moqups.

Para projetos mais complexos, Axure pode ser considerada uma excelente alternativa. Para um período de 30 dias, disponibiliza a sua versão gratuita sem nenhum tipo de restrições, além disso, facilita a criação de protótipos para vistas adaptativas e possui mais opções de interação em comparação com as ferramentas anteriores, pelo que se converte também numa alternativa para a criação de protótipos de alta fidelidade. Por outro lado, não conta com tanta variedade de elementos predeterminados como MockFlow e a limitação de 30 dias gratuitos pode não ser suficiente para a criação sem custos de protótipos de grandes aplicações.

Para o caso deste projeto, foi decidido utilizar a ferramenta Axure, devido ao número ilimitado de páginas e de objetos que oferece, além da grande variedade de opções de interação que dispõe. O limite de dias gratuitos não foi considerado um problema para este caso porque existiu a disponibilidade de realizar os protótipos em menos do tempo estabelecido.

No que diz respeito às ferramentas para a criação de protótipos de alta fidelidade, Framer é ideal para obter resultados inovadores e altamente interativos, ao contrário da InVision, que possui um número limitado de interações disponíveis.

Por outro lado, InVision é mais intuitiva e fácil de aprender, porque não é necessário saber nenhuma linguagem de programação, contudo, para ajudar no processo de aprendizagem, Framer possui uma documentação bastante completa e bem estruturada disponível para a linguagem CoffeeScript.

Pelo facto de Framer Studio estar disponível unicamente para Mac e a sua versão gratuita ser por tempo limitado foi escolhida a ferramenta InVision para ser usada neste projeto; a sua simplicidade permite criar protótipos de alta fidelidade interativos em um curto período de tempo fácil e sem custos.

Na Tabela 3 apresentam-se as características mais relevantes que possuem as ferramentas de prototipagem estudadas.

Tabela 3 – Características das ferramentas de prototipagem estudadas

Características	**Ferramenta de prototipagem**				
	Moqups	**MockFlow**	**Axure**	**InVision**	**Framer**
***Site* oficial**	moqups.com	www.mockflow.com	www.axure.com	www.invisionapp.com	framer.com
Fidelidade de prototipagem	Baixa	Baixa	Alta e baixa	Alta	Alta
Livraria de objetos	Muito boa	Excelente	Boa	Não possui	Não possui
Formatos de exportação	PDF ou PNG[7]	HTML5, PDF, PowerPoint, PNG e Word	HTML, GIF, PNG, JPG, CSV e Word	HTML e PDF	HTML
Vistas adaptativas	Não	Não	Sim	Sim	Sim
Interação	Boa	Muito boa	Excelente	Excelente	Excelente

[7] Exportação dos protótipos disponível unicamente nas versões pagas.

Para finalizar, a Tabela 4 apresenta as restrições de utilização impostas nas versões gratuitas destas ferramentas de prototipagem.

Tabela 4 – Limites da versão gratuita das ferramentas de prototipagem

Limites da versão gratuita	Ferramenta de prototipagem				
	Moqups	**MockFlow**	**Axure**	**InVision**	**Framer**
Nº de objetos de desenho	300 objetos	Ilimitado	Ilimitado	Não aplicável	Não aplicável
Nº de projetos por conta	1 projeto	1 projeto	Ilimitado	1 projeto	Ilimitado
Nº de páginas no projeto	Ilimitado	4 páginas	Ilimitado	Ilimitado	Ilimitado
Nº de colaboradores no projeto	Não determinado	2 colaboradores	Ilimitado	Ilimitado	Não aplicável
Nº de dias de utilização	Ilimitado	Ilimitado	30 dias	Ilimitado	14 dias
Exportação do projeto	Não	Sim	Sim	Sim	Sim

II.8.5.2. Análise das *Frameworks* e Livrarias JavaScript

As *frameworks* JavaScript trazem um conjunto de paradigmas e padrões que facilitam o desenvolvimento e manutenção de produtos de *software*, entre os quais se destaca o padrão de desenho arquitetural MVC e as suas variantes. Estas *frameworks* reduzem a carga de trabalho do servidor, isto é possível porque ao ocorrer alguma mudança na navegação da aplicação o servidor envia ao cliente unicamente a vista que foi alterada, e não o documento HTML inteiro, permitindo uma transferência de dados bastante ligeira. Adicionalmente, os utilizadores percebem uma melhoria no desempenho nestas aplicações *Web* porque as suas ações recebem uma resposta mais rápida por parte do servidor [45].

Frameworks JavaScript oferecem muitas facilidades para o desenvolvimento de aplicações *Web*, com um desempenho semelhante às aplicações nativas [45]. Para o caso deste projeto, foi estabelecido pela empresa o uso da *framework* AngularJS na sua

versão 1.5, uma vez que é a utilizada pela equipa de Canais Não Presenciais no projeto CAOP.

Em seguida, apresenta-se uma análise comparativa das ferramentas estudadas, onde são considerados os aspetos essenciais destas *frameworks* e livrarias JavaScript, nomeadamente a facilidade de utilização e de aprendizagem, as formas de renderização, o desempenho que oferecem, e a reutilização e flexibilidade das aplicações desenvolvidas fazendo uso destas ferramentas.

- **Facilidade de Utilização e de Aprendizagem**

No que diz respeito à facilidade de aprendizagem das *frameworks* e livrarias JavaScript estudadas, as livrarias ReactJS e Vue.js são consideradas pela comunidade mais fáceis de aprender em comparação com as *frameworks* AngularJS e Angular 2. Isto deve-se, entre outras coisas, ao facto de não serem tão complexas quanto às funcionalidades oferecidas e pela alternativa que ambas oferecem de desenvolver os projetos utilizando o *superset* JSX ao invés da linguagem JavaScript, que proporciona uma sintaxe mais entendível e fácil de aprender.

A *framework* Angular 2 permite usar como alternativa ao JavaScript o *superset* TypeScript que também se destaca pela sua simplicidade, mas uma das suas desvantagens é a configuração inicial dos projetos (*Setup*), que é mais complexa em comparação com o AngularJS e as livrarias estudadas.

Para finalizar esta secção, destaca-se como uma vantagem do AngularJS a sua alta popularidade hoje em dia, o que permite que os desenvolvedores consigam um vasto número de livrarias, componentes e documentação sobre esta *framework*, facilitando o desenvolvimento dos projetos.

- **Renderização e Desempenho**

Uma das vantagens do AngularJS em comparação às outras *frameworks* existentes é que esta requer escrever menos código, devido às novas características que

oferece, como a injeção de dependência[8] e o "*Two-Way Data Binding*" [77], [45]. Apesar de que esta última característica foi um dos motivos do sucesso do AngularJS, esta implica um custo em tempo de processamento no *browser*, penalizando o desempenho das aplicações mais complexas e dificultando o processo de *debugging* (porque o Modelo da aplicação pode ser alterado em múltiplos pontos resultando difícil conseguir a origem dos erros) [52].

Para solucionar este problema de desempenho, o Angular 2 e as livrarias ReactJS e Vue.js para além de oferecer o "*Two-Way Data Binding"* (apesar de não ser uma boa prática) permitem também o uso do "*One-Way Data Binding*" em que o fluxo de dados é feito de forma mais controlada, otimizando as aplicações desenvolvidas [52].

Por outro lado, o AngularJS diminui o seu desempenho quando existe uma grande quantidade de *watchers*[9], porque cada vez que surge uma alteração no *scope* (âmbito) todos os *watchers* têm de ser reavaliados novamente. Angular 2, ReactJS e Vue.js resolvem este problema com a utilização de *observables*[9], em que todas as alterações são desencadeadas de forma independente exceto quando existe alguma relação de dependência explícita [78].

No que se refere à renderização das vistas das aplicações, com a aparição do Angular 2 e das novas livrarias apresentadas, é possível renderizar as vistas no lado do servidor (a diferença do AngularJS em que só permite renderizar no lado do cliente), o que é uma vantagem porque permite uma potente renderização e SEO nas aplicações, além de reduzir significativamente à espera do utilizador na primeira visita da aplicação, pois as vistas já haverão sido renderizadas no servidor e são enviadas diretamente ao cliente [52].

Ao contrário do Angular 2 em que a renderização é feita no DOM regular (o que prejudica o seu desempenho), as livrarias ReactJS e Vue.js utilizam um DOM virtual. Este DOM virtual é mais leve no Vue.js que no ReactJS porque sabe exatamente quais componentes precisam de ser renderizados novamente quando existe alguma alteração

[8] Uma vez que seja importada ou criada alguma livraria no projeto é unicamente necessário injetar a sua dependência em qualquer parte do código para que se encontre disponível na aplicação, sem necessidade de implementações complexas [76].
[9] Os "*watchers"* e "*observables"* são mecanismos para observar as alterações do *scope* (âmbito).

num componente, o que traz um incremento positivo no seu desempenho. Em ReactJS, a renderização é feita para todos os componentes da subárvore do componente alterado, o que introduz sobrecarga [78].

Entre as soluções estudadas, a que se encontra com maior vantagem quanto ao desempenho é a livraria Vue.js, porém ReactJS possui também um excelente desempenho. Ambas livrarias oferecem um desempenho superior ao AngularJS e Angular 2, principalmente por não ter a sobrecarga que possuem estas *frameworks* que são mais complexas.

- **Reutilização e Flexibilidade**

A partir da versão 1.5 do AngularJS é possível seguir uma abordagem orientada a componentes que facilita a reutilização de código, para além das outras vantagens apresentadas no capítulo II.7. A utilização desta abordagem é opcional no AngularJS, mas foi a seguida neste projeto devido às vantagens que oferece e pela recomendação da equipa de desenvolvimento. Cabe destacar que as *frameworks* e livrarias JavaScript mais recentes, como o Angular 2, ReactJS e Vue.js, já se encontram todas seguindo esta abordagem.

No que diz respeito à flexibilidade no desenvolvimento das aplicações, Angular 2 permite a implementação de classes e de muitos dos conceitos do paradigma de programação orientado a objetos que podem ser vantajosos em certas circunstâncias. Contudo, estas características não estão totalmente maduras como no caso das *frameworks* baseadas em outras linguagens (como a linguagem Java). O facto de Angular 2 forçar o programador a utilizar esta abordagem, sem ser de maneira opcional como no AngularJS, pode ser considerada uma desvantagem desta *framework*.

AngularJS simplifica o desenvolvimento de aplicações, apresentando um nível mais alto de abstração para o programador. Como qualquer abstração, isto implica um custo na flexibilidade, pelo que esta *framework* não é adequada para todo tipo de aplicações [77]. Por outro lado, uma das características que pode ser considerada como desvantagem nas livrarias ReactJS e Vue.js em comparação com as *frameworks* que permitem o desenvolvimento completo das aplicações, é o facto de estas só tratarem das vistas das aplicações. Contudo, isto possibilita uma alta flexibilidade, já que

permitem complementar as suas funcionalidades com a integração de outras ferramentas e livrarias que se adaptam melhor ao projeto em específico.

É apresentada, de seguida, uma tabela comparativa (Tabela 5) entre as *frameworks* e livrarias JavaScript estudadas.

Tabela 5 – Características das *frameworks front-end* estudadas

Características	AngularJS	Angular 2	ReactJS	Vue.js
Logótipo				
Definição	*Framework*	*Framework*	Livraria JS	Livraria JS
Data de lançamento	2010	2016	2013	2014
Autor	Empresa *Google*	Empresa *Google*	Empresa *Facebook*	Evan You
***Site* oficial**	angularjs.org	angular.io	facebook.github.io/react/	vuejs.org
Tecnologias	JS/HTML	JS/TypeScript/ HTML	JS/JSX	JS/JSX/ HTML
Código aberto	Sim	Sim	Sim	Sim
Curva de aprendizagem	Alta	Média	Baixa	Baixa
***Data Binding*[10]**	2 Vias	1 ou 2 Vias	1 ou 2 Vias	1 ou 2 Vias
Renderização	Cliente	Cliente e Servidor	Cliente e Servidor	Cliente e Servidor
DOM	DOM Regular	DOM Regular	DOM Virtual	DOM Virtual
Desempenho	Médio	Alto	Muito Alto	Excelente
Nº de estrelas no Github[11]	56,554	26,191	71,849	61,259

[10] O *data binding* é a união em tempo real dos dados de dois elementos.
[11] Dados recolhidos até o dia 01/07/2017.

Nº de *commits*[12] no Github[11]	8,546	8,183	8,813	2,111
Nº de descargas no NPM[13]	12,071,805	1,601,711	48,488,769	4,502,005

Para concluir a análise comparativa destas ferramentas, é importante referir que existe uma grande variedade de soluções disponíveis para desenvolvimento de aplicações *Web*, todas contam com seus aspetos positivos e negativos. Dependerá dos requisitos de cada projeto e dos conhecimentos técnicos da equipa de desenvolvimento a decisão de qual é a melhor *framework* ou livraria a utilizar, muitas vezes o desempenho e a flexibilidade da aplicação são as características mais importantes a considerar, outras vezes é a linguagem de programação, a facilidade de implementação e aprendizagem ou a documentação disponível.

Neste projeto a *framework* utilizada foi **AngularJS**, mas a equipa de desenvolvimento recomendou estruturar o código de modo a permitir uma fácil migração ao Angular 2 quando for necessário. Quando se avalia a possibilidade de migrar os projetos de AngularJS para Angular 2 são consideradas todas as suas características, entre as vantagens do Angular 2 ressalta o incremento do desempenho geral das aplicações e a possibilidade de renderizar no lado do servidor.

Por outro lado, para aplicações *Web* que requerem um ótimo desempenho estão comumente adotadas na atualidade às livrarias ReactJS e Vue.js, que não só se destacam pelo seu alto desempenho, mas também pela sua alta flexibilidade e fácil aprendizagem.

II.8.5.3. Análise dos Sistemas de Controlo de Versões e dos Serviços de Alojamento de Repositórios de *Software*

No que diz respeito aos sistemas de controlo de versões, na empresa são utilizados o SVN e o Git. O **SVN** é usado pela equipa de *Middleware* e pela equipa de Canais Não Presenciais para os projetos *Mobile Banking* e *eBanka*; os projetos mais

[12] A ação de guardar num repositório de *software* as alterações realizadas aos ficheiros do projeto é chamada no Git de "*commit*".
[13] Dados recolhidos desde o dia 01/07/15 até o dia 01/07/17.

recentes, como o portal PFS e o projeto CAOP, já foram migrados ao sistema de controlo de versões distribuído **Git** devido as grandes vantagens que este apresenta.

Atualmente, o serviço de alojamento de repositórios de *software* mais popular é o **Github**, esta é uma ótima escolha para projetos que não possuem restrições quanto à sua privacidade. Além disso, permite visualizar outros projetos existentes e pode ser usado como portefólio pessoal pelos desenvolvedores de *software*. Como o Github tem a limitação de não ser gratuito para repositórios privados, **BitBucket** e **GitLab** são excelentes alternativas para estes casos. Na empresa é usado o **GitLab**, porque permite utilizar servidores próprios da empresa além de alojar os repositórios de forma privada e gratuita.

II.9. *WIDGETS* PARA *INTERNET BANKING*

Antes de dar início ao desenvolvimento dos *widgets*, foi realizada uma pesquisa das aplicações já existentes que possuem funcionalidades semelhantes às requeridas nos *widgets*, para obter uma visão geral do âmbito em que se inserem e os elementos que as compõe, e analisar a interação de estas aplicações com os utilizadores.

Ao tratar-se de *widgets*, foi decidido orientar esta análise mais concretamente às aplicações de *Internet Banking* para dispositivos móveis, isto porque a própria natureza destas aplicações apresentam uma série de característica que são equiparáveis à natureza dos *widgets*, nomeadamente, a simplicidade da interface do utilizador, o aspeto minimalista e a rapidez e facilidade para realizar as operações pretendidas; centram-se em oferecer as funcionalidades mais importantes ao utilizador e na utilização de baixos recursos de *hardware* para serem executadas corretamente.

São apresentadas a seguir as aplicações mais destacadas encontradas durante esta fase de análise e que possuem características que se adequam aos *widgets* do projeto.

II.9.1. Aplicações Semelhantes com *Dashboard* de *Widgets*

Durante a análise das aplicações já existentes no mercado relacionadas com o âmbito do projeto foram avaliadas aplicações e *Websites* que possuem um *dashboard*

com *widgets* de forma semelhante ao que oferece o sistema *eBanka*, com a finalidade de obter uma visão geral das alternativas e soluções mais usadas na atualidade que possuem estas características.

Um dos melhores exemplos encontrados durante esta fase de pesquisa é o *dashboard* apresentado na Figura 10, este *dashboard* se encontra no *Website* oficial da **livraria de gráficos "NVD3"** e está disponível para a *framework* Angular [79]. Possui uma interface gráfica amigável que oferece as funcionalidades de adicionar ou remover *widgets* gráficos no *dashboard*. Além disso, o utilizador pode determinar o tamanho dos *widgets* e a organização dos mesmos. Cada *widget* tem uma opção de configuração onde o utilizador pode estabelecer qual tipo de gráfico pretende visualizar no respetivo *widget*.

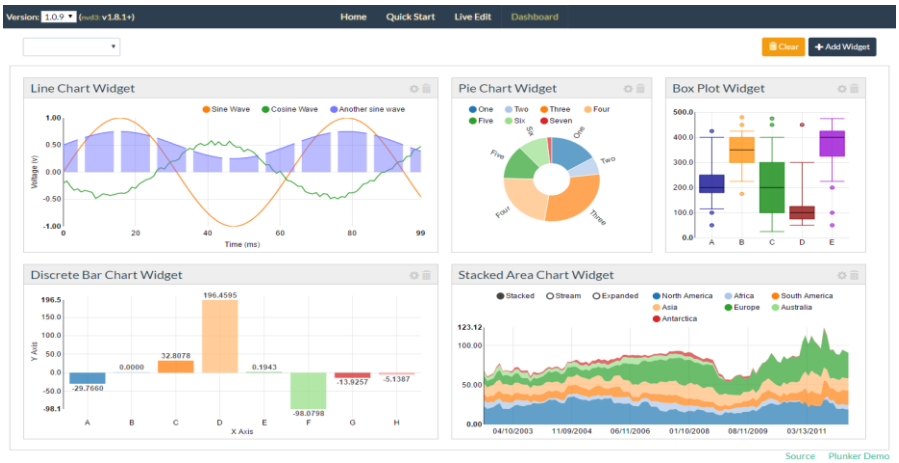


Figura 10 – *Dashboard* da livraria de gráficos "NVD3".

Outro exemplo de *dashboard* com *widgets* é o da aplicação ***Google Analytics***, já que esta aplicação oferece ferramentas para a análise dos dados de utilização de *Websites* e apresenta estas informações através de *widgets*. Como se pode visualizar na Figura 11, a maioria das informações são apresentadas sob a forma de gráficos interativos o que facilita a análise dos dados apresentados.

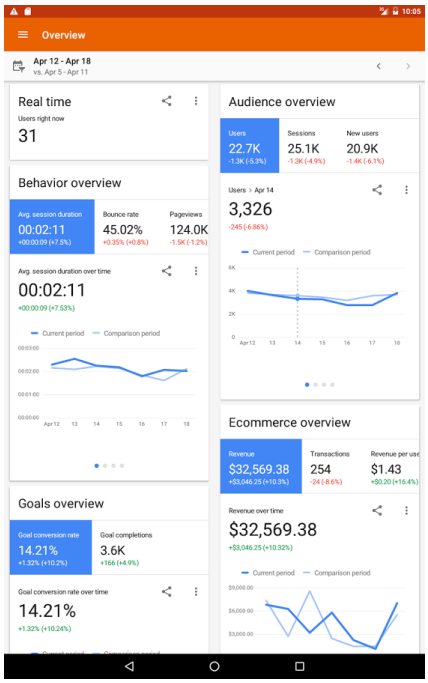


Figura 11 – *Dashboard* da aplicação "*Google Analytics*" [80].

O uso de gráficos interativos para apresentar dados é uma das soluções mais usadas nos *widgets*. Estes mostram de forma detalhada e simplificada as informações mais importantes para o utilizador. Na Figura 12 ilustra-se um gráfico da aplicação ***Google My Business*** que permite visualizar, através de um gráfico interativo, o número de vezes que os utilizadores encontraram a ficha de informações de uma dada empresa ao introduzir (no motor de pesquisa de *Google*) o nome ou endereço da empresa (pesquisa direta), ou ao pesquisar alguma categoria, produto ou serviço relacionado (ficha descoberta).

Figura 12 – Gráfico da aplicação "*Google My Business*".

II.9.2. Aplicações Semelhantes para Pagamento de Serviços

Com a finalidade de estudar os aspetos mais importantes de um sistema em relação aos pagamentos de serviços, foram estudadas diversas aplicações já existentes para bancos portugueses que oferecem aos seus clientes, entre outras operações bancárias, a execução desta operação. A seguir apresenta-se uma breve descrição destas aplicações, nomeadamente, *App Millennium*, *App Santander Totta* e *NB Smart*, onde serão destacadas as características mais relevantes de cada uma no que diz respeito aos pagamentos de serviços.

II.9.2.1. *App Millennium*

App Millennium é uma aplicação que permite aos clientes do banco *Millennium BCP* (*Banco Comercial Português*) realizar as operações bancárias mais destacadas através dos dispositivos *smartphones*, *tablets* e *Apple Watch*. Esta aplicação oferece, entre outras operações, a realização de pagamento de serviços através de uma interface bastante simples e amigável para o utilizador, e possui a funcionalidade de reconhecimento de voz para facilitar a sua utilização.

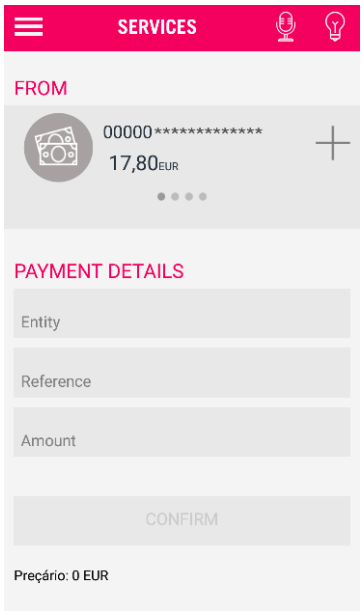


Figura 13 – Pagamento de serviços na aplicação "*App Millennium*".

Como se pode visualizar na Figura 13, a interface para pagamentos de serviços apresenta um *slideshow* com as contas do utilizador e o respetivo saldo; adicionalmente permite visualizar estas mesmas informações através de uma lista. Os restantes dados de pagamentos, como a entidade, referência e montante são introduzidos

manualmente pelo utilizador, que depois deverá confirmar o pagamento para completar a operação.

II.9.2.2. *App Santander Totta*

A aplicação do banco ***Santander Totta*** (Portugal) para particulares caracteriza-se pelo seu alto nível de segurança e simplicidade. Encontra-se disponível para os dispositivos *smartphones*, *tablets* e *Apple Watch* [81].

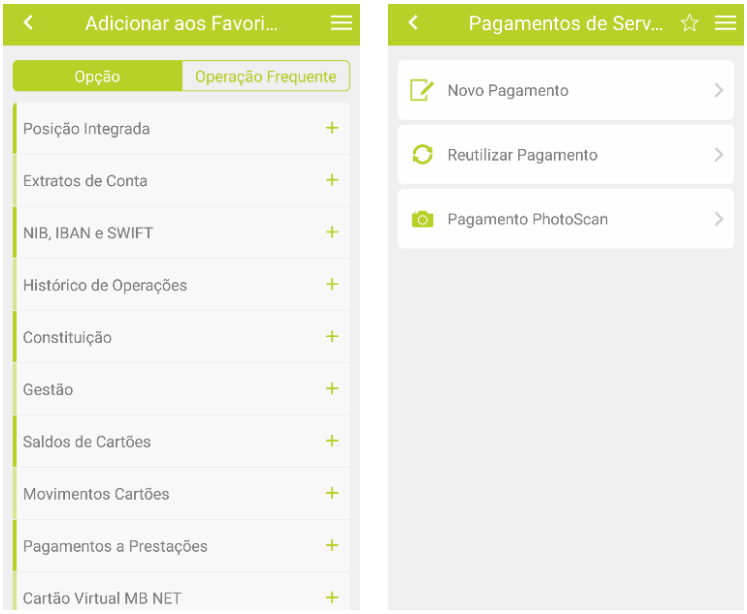
À semelhança da aplicação anteriormente apresentada, esta também permite ao utilizador realizar pagamentos de serviços. Os dados das contas são visualizados num *slideshow* e o utilizador pode estabelecer os seus destinatários (entidades) frequentes como favoritos, os quais são apresentados sob a forma de lista ao clicar no respetivo botão, desta forma é possível escolher o destinatário pretendido sem a necessidade de introduzir os dados repetidamente em cada pagamento.

Figura 14 – Pagamento de serviços na aplicação "*Santander Totta*".

A Figura 14 ilustra a interface desta operação, são pedidos os dados necessários para o pagamento, nomeadamente a entidade, referência, montante e a data do pagamento. Estes dados podem também ser preenchidos ingressando uma fotografia da fatura através da câmara.

II.9.2.3. *NB Smart*

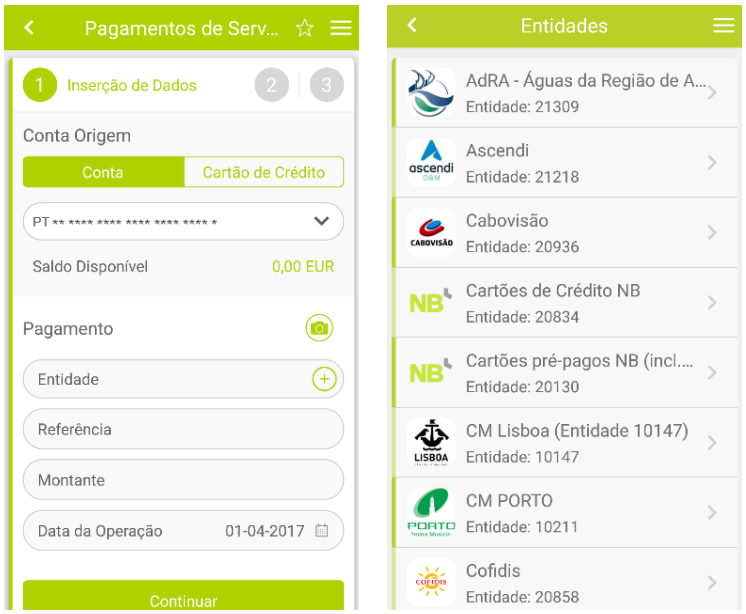
A aplicação ***NB Smart*** do "*Novo Banco*" encontra-se disponível para dispositivos *smartphones* e *tablets*, e tal como as aplicações anteriores, apresenta um conjunto de opções para pagamento de serviços. Além disso, permite ao utilizador estabelecer quais das operações oferecidas pela aplicação são as suas favoritas, desta forma, as operações escolhidas são guardadas numa lista de favoritos facilitando o seu acesso, como se pode visualizar na Figura 15-a.



(a) **(b)**

Figura 15 – Aplicação "*NB Smart*", (a) Estabelecer operações como favoritas; (b) Menu para iniciar o pagamento de serviços.

Em relação ao pagamento de serviços, a aplicação apresenta inicialmente um menu (ilustrado na Figura 15-b) onde o utilizador pode escolher se deseja realizar um novo pagamento ou reutilizar algum dos pagamentos efetuados anteriormente (que já possui os dados preenchidos) ou também a opção de realizar o pagamento com o preenchimento dos dados necessários através de uma fotografia.



(a) **(b)**

Figura 16 – Pagamento de serviços na aplicação "*NB Smart*", (a) Formulário para a inserção dos dados; (b) Lista de entidades.

Se o pretendido é realizar um novo pagamento de serviço, a aplicação permite alterar a conta a debitar e apresenta o saldo disponível (como ilustrado na Figura 16-a). Os dados do pagamento podem ser introduzidos através da câmara ou manualmente pelo utilizador, onde a entidade pode ser escolhida entre uma lista de entidades, como se apresenta na Figura 16-b.

II.9.3. Aplicações Semelhantes para Operações Pendentes

De seguida são apresentadas duas aplicações para bancos, *OnMobile* e *BPI empresas*, que se caracterizam por oferecer aos clientes empresariais a opção de aprovar as suas operações pendentes de forma simplificada através de dispositivos móveis.

II.9.3.1. *Popular OnMobile*

A aplicação ***Popular OnMobile*** do *Banco Popular* (Portugal) encontra-se disponível para dispositivos móveis e permite visualizar através de uma interface bastante amigável e intuitiva, o número de operações pendentes da assinatura do utilizador, para além do número das já assinadas e as que possuem erros. Oferece também a opção de visualizar todas as assinaturas.

Figura 17 – Resumo das operações pendentes na aplicação "*Popular OnMobile*" [82].

Como ilustrado na Figura 17, a aplicação apresenta um resumo das operações pendentes, com os dados mais relevantes como o nome, número, montante e a data da operação. Segundo a descrição oficial da aplicação [83], também permite assinar as operações e notifica ao utilizador quando existe alguma operação pendente da sua assinatura.

II.9.3.2. *BPI Empresas*

A aplicação ***BPI Empresas*** do banco *BPI* (*Banco Português de Investimento*) é uma aplicação orientada às empresas que facilita a gestão empresarial e permite o acesso a um conjunto de funcionalidades através de dispositivos móveis, entre as quais se encontra a autorização de operações pendentes [84].

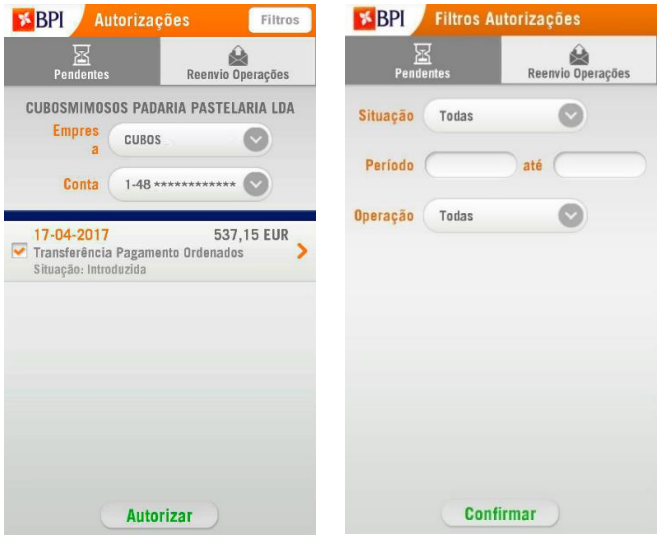


Figura 18 – Operações pendentes na aplicação "*BPI Empresas*".

Como se pode visualizar na Figura 18, o utilizador pode autorizar as operações pendentes escolhendo inicialmente a empresa e a conta pretendida, depois a aplicação apresenta os dados mais importantes destas operações, nomeadamente, nome, situação, data e montante. Além disso, permite filtrar as operações a visualizar por tipo de situação, período e tipo de operação.

II.9.4. Considerações Finais

Após uma breve análise das aplicações apresentadas anteriormente, é importante destacar que todas se caracterizam pela sua simplicidade na hora de realizar as diversas operações.

No que diz respeito aos pagamentos de serviços, na maioria das aplicações analisadas destaca-se uma apresentação clara dos dados e a fácil introdução dos mesmos. Possuem elementos como *slideshows* para a escolha rápida de dados e oferecem também outras formas de introduzir as informações necessárias, como por exemplo, as listas de entidades já preestabelecidas onde o utilizador só tem que selecionar a entidade pretendida. Além disso, os utilizadores podem estabelecer quais são os seus destinatários favoritos, o que simplifica o processo nos próximos pagamentos, e algumas aplicações permitem a introdução de dados através da câmara ou por reconhecimento de voz.

Por outro lado, as aplicações analisadas que permitem a aprovação das operações pendentes oferecem aos utilizadores uma forma simplificada de realizar este processo, destacando-se o uso de listas com o resumo das operações pendentes e os seus dados mais relevantes, assim como a apresentação visualmente atrativa do número total de operações pendentes pela aprovação do utilizador, para além de outras informações relevantes.

Ao se tratar do desenvolvimento de *widgets* é importante mencionar as características mais relevantes dos *dashboards* já existentes, estes se adequam às preferências do utilizador e se destacam pelo uso de elementos gráficos interativos para a apresentação dos dados.

Todas as características apresentadas anteriormente foram tidas em conta durante o processo de desenvolvimento dos *widgets*, descrito no próximo capítulo.

III. DESENVOLVIMENTO DA APLICAÇÃO *WEB*

III.1. INTRODUÇÃO

A maioria das equipas de desenvolvimento seguem um ou mais Modelos de Processos de *Software* para especificar as fases e as tarefas que devem ser executadas durante o desenvolvimento e respetivas prioridades. Estes modelos são normalmente escolhidos com base nas características dos projetos e das equipas, os métodos e ferramentas que usam e as limitações no tempo de entrega dos seus produtos. Para o projeto curricular foi considerado o modelo de processos XP (exposto no capítulo II.3.2.1) como base para o desenvolvimento, adaptando as atividades realizadas quanto possível às que o modelo apresenta.

Foram seguidas todas as etapas que pressupõe ter o desenvolvimento de *software* de forma iterativa e incremental, começando pelo levantamento de requisitos, para posteriormente iniciar a fase de desenho, onde foi seguido o Paradigma de Prototipagem (apresentado no capítulo II.3.1) para a definição da interface do utilizador e das características mais específicas dos *widgets*; ainda nesta fase foi definida a arquitetura do projeto e as estruturas de dados dos serviços *Web*. Após todas estas atividades começou a implementação do *software*, testes e documentação do mesmo.

É fundamental numa fase inicial saber qual o propósito do projeto e as características que este deve possuir, pelo que foram realizadas reuniões presenciais com a equipa de desenvolvimento da empresa e o responsável do departamento DCS, o Engenheiro Pedro Camacho, para a discussão das funcionalidades e características pretendidas.

Identificados os requisitos, o passo seguinte consistiu em criar protótipos de baixa e de alta fidelidade de forma iterativa, para assim definir com maior detalhe todas as características do projeto e a interface do utilizador, cujos protótipos de alta fidelidade foram elaborados pelos *designers* da empresa.

Previamente à fase de implementação foram definidas as estruturas de dados dos serviços *Web* a utilizar e a arquitetura do projeto, a qual se encontra baseada em componentes e segue o padrão arquitetural MVVM.

Finalmente foi realizada a implementação do projeto, onde um dos integrantes da equipa de Canais Não Presenciais da empresa, Vítor Camacho, exerceu o papel de *coach* do projeto, segundo os papéis que define o modelo XP.

É importante salientar que foi utilizada a aplicação **Trello** [85] para facilitar a gestão das tarefas do projeto, que permite organizar facilmente todos os apontamentos e ideias na forma de cartões e agrupar estes conforme o contexto.

III.2. Levantamento de Requisitos

Durante a fase de levantamento de requisitos do projeto foram realizadas várias reuniões com a equipa de desenvolvimento da empresa, *designers* e o representante do departamento DCS para recolher as características e funcionalidades pretendidas para o projeto, as quais foram de seguida traduzidas em requisitos. Todas estas reuniões decorreram nas instalações da empresa, onde foram tiradas notas de todos os aspetos relevantes discutidos sobre o projeto.

Como ilustrado na Figura 19, dividiu-se o projeto em dois módulos, cada módulo representa um *widget* e possui um conjunto de funcionalidades relacionadas entre si. O primeiro módulo diz respeito ao *widget* de pagamento de serviços e o segundo módulo ao *widget* de operações pendentes de aprovação.

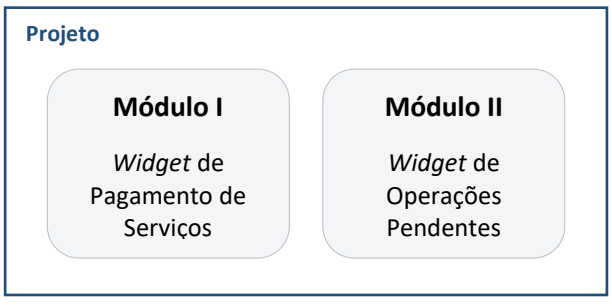


Figura 19 – Módulos do projeto.

III.2.1. Requisitos dos *Widgets*

Para o desenvolvimento bem-sucedido de um projeto, é necessário conhecer os requisitos do sistema e documentar os mesmos de maneira adequada. Os requisitos são condições ou capacidade que devem ser alcançadas ou estar presentes nos sistemas a desenvolver [86].

Com base nas informações recolhidas nas reuniões realizadas com os membros da empresa e no estudo realizado às aplicações já existentes no mercado, foram determinados um conjunto de requisitos para cada *widget*. Adicionalmente, é importante estabelecer uma prioridade que indique que tão essencial é cada requisito, portanto, para este projeto foi utilizada a **técnica de priorização MoSCoW**, a qual permite centrar-se nas necessidades reais do produto ao invés de características desejadas, mas que não são necessárias; esta técnica classifica os requisitos nas seguintes quatro categorias [87], [88]:

- **M (*Must have*) – Deve ter:** requisitos considerados os mais importantes e devem estar implementados na versão final do produto;
- **S (*Should have*) – Deveria ter:** requisitos que deveriam estar incluídos na versão final, mas podem ser prescindíveis em dadas circunstâncias;
- **C (*Could have*) – Poderia ter:** requisitos que são desejados, mas não necessários, são características opcionais que são implementadas se há tempo e pressuposto;
- **W (*Won't have*) – Não terá desta vez:** requisitos que não vão ser incluídos na solução, mas que poderão ser considerados nas futuras versões do produto.

III.2.1.1. Requisitos Funcionais

Os requisitos funcionais definem a funcionalidade oferecida pelo sistema a ser desenvolvido [86]. Na Tabela 6 e Tabela 7 apresentam-se os requisitos funcionais que foram determinados para cada *widget* e as respetivas prioridades.

Tabela 6 – Requisitos funcionais do *widget* de pagamento de serviços

WIDGET DE PAGAMENTO DE SERVIÇOS		PRIORIDADE
RF01	O *widget* deve permitir ao utilizador realizar o pagamento dos seus serviços favoritos.	***Must***

RF02	O *widget* deve apresentar ao utilizador o nome e número da conta a utilizar para o pagamento.	***Must***
RF03	O *widget* deve apresentar ao utilizador o valor do saldo e tipo de moeda da conta a utilizar para o pagamento.	***Must***
RF04	O *widget* deve permitir ao utilizador alterar a conta a utilizar para o pagamento.	***Must***
RF05	O *widget* deve diferenciar visualmente o saldo das contas do utilizador em negativo.	***Could***
RF06	O *widget* deve apresentar ao utilizador o nome e logótipo de todas as suas entidades favoritas.	***Must***
RF07	O *widget* deve permitir ao utilizador escolher uma entidade das suas favoritas para realizar o pagamento.	***Must***
RF08	O *widget* deve permitir ao utilizador estabelecer novas entidades como favoritas.	***Could***
RF09	O *widget* deve limitar ao utilizador de estabelecer até um máximo de 5 entidades favoritas.	***Could***
RF10	O *widget* deve permitir ao utilizador eliminar entidades estabelecidas como favoritas.	***Could***
RF11	O *widget* deve permitir ao utilizador introduzir o número de referência do pagamento.	***Must***
RF12	O *widget* deve apresentar ao utilizador uma lista com os montantes predefinidos disponíveis para a entidade selecionada.	***Must***
RF13	O *widget* deve apresentar ao utilizador montantes sugeridos quando não existir predefinidos.	***Could***
RF14	O *widget* deve permitir ao utilizador introduzir o montante a pagar quando a entidade selecionada o permita.	***Must***
RF15	O *widget* deve limitar ao utilizador de inserir montante fora do intervalo determinado nos casos correspondentes (valor mínimo e máximo).	***Must***

RF16	O *widget* deve apresentar ao utilizador o montante escolhido para o pagamento.	***Must***
RF17	O *widget* deve apresentar ao utilizador um resumo com os dados do pagamento antes de finalizar a operação.	***Must***
RF18	O *widget* deve apresentar ao utilizador uma mensagem com o estado da operação após realizado o pagamento (sucesso ou insucesso).	***Must***
RF19	O *widget* deve permitir ao utilizador aceder à página de pagamentos.	***Could***
RF20	O *widget* deve confirmar o pagamento utilizando o mecanismo de *Chave de Confirmação*[14].	***Won't have***
RF21	O *widget* deve confirmar o pagamento utilizando o mecanismo *OTP (Código de Confirmação)*[15].	***Won't have***
RF22	O *widget* deve confirmar o pagamento utilizando o mecanismo de *Cartão de Coordenadas*[16].	***Won't have***

Tabela 7 – Requisitos funcionais do *widget* de operações pendentes

WIDGET DE OPERAÇÕES PENDENTES DE APROVAÇÃO		**PRIORIDADE**
RF23	O *widget* deve permitir ao utilizador visualizar o número total de operações pendentes.	***Must***
RF24	O *widget* deve permitir ao utilizador visualizar o número total de operações pendentes da sua aprovação.	***Must***
RF25	O *widget* deve permitir ao utilizador visualizar o número total de operações já aprovadas por si, mas cuja execução ainda depende da aprovação de terceiros.	***Must***
RF26	O *widget* deve apresentar um gráfico com a contabilização do número total de operações pendentes da aprovação do utilizador e de terceiros.	***Must***

[14] Mecanismo para validar uma operação bancária através do pedido da chave de confirmação do cliente.
[15] Mecanismo para validar uma operação bancária através do pedido de um código que é enviado por mensagem de texto ao telemóvel do cliente.
[16] Mecanismo para validar uma operação bancária através do pedido de coordenadas aleatórias apresentadas no cartão de coordenadas do cliente.

RF27	O *widget* deve permitir ao utilizador aceder à página de operações pendentes na secção de operações pendentes da sua aprovação.	***Must***
RF28	O *widget* deve permitir ao utilizador aceder à página de operações pendentes na secção de operações pendentes da aprovação de terceiros.	***Must***
RF29	O *widget* deve permitir ao utilizador aprovar a operação com a data mais antiga das que se encontram na sua lista de pendentes por aprovação.	***Should***
RF30	O *widget* deve permitir ao utilizador rejeitar a operação com a data mais antiga das que se encontram na sua lista de pendentes por aprovação.	***Should***
RF31	O *widget* deve apresentar ao utilizador informações sobre a operação com data mais antiga das suas pendentes (nome da operação, nome do operador quem efetuou a operação, montante, data da operação e as respetivas condições de aprovação).	***Should***
RF32	O *widget* deve diferenciar visualmente as condições já aprovadas da operação pendente do utilizador que possui a data mais antiga.	***Should***
RF33	O *widget* deve permitir ao utilizador aceder à página de operações pendentes para visualizar os detalhes completos da operação pendente com data mais antiga.	***Could***
RF34	O *widget* deve apresentar ao utilizador um resumo com os dados da operação pendente antes de finalizar a aprovação ou rejeição da mesma.	***Should***
RF35	O *widget* deve apresentar ao utilizador uma mensagem com o estado da operação após realizada a aprovação ou rejeição da operação pendente (sucesso ou insucesso).	***Should***
RF36	O *widget* deve permitir ao utilizador aceder à página de operações pendentes.	***Could***

III.2.1.2. Requisitos Não Funcionais

Os requisitos não funcionais, também chamados requisitos de qualidade, definem as qualidades desejadas do sistema a ser desenvolvido. Tipicamente, requisitos de qualidade definem o desempenho, a disponibilidade, a confiabilidade, a

escalabilidade ou a portabilidade de um sistema [86]. Neste relatório, os requisitos não funcionais estão organizados de acordo com as seguintes categorias: usabilidade (Tabela 8), aparência e implementação (Tabela 9), modificabilidade (Tabela 10), compatibilidade (Tabela 11) e manutenção (Tabela 12), sendo que podem existir outras.

Tabela 8 – Requisitos não funcionais (Usabilidade)

USABILIDADE		PRIORIDADE
RNF01	A realização das tarefas fazendo uso dos *widgets* deve ser simples para todos os utilizadores.	***Must***
RNF02	Os *widgets* devem proporcionar mensagens de erros que sejam descritivos, informativos e orientados ao utilizador.	***Should***
RNF03	Os *widgets* devem possuir um desenho adaptado aos diversos tamanhos de ecrã, garantindo uma visualização adequada em todos os dispositivos.	***Must***

Tabela 9 – Requisitos não funcionais (Aparência e Implementação)

APARÊNCIA E IMPLEMENTAÇÃO		PRIORIDADE
RNF04	A interface gráfica dos *widgets* deve cumprir com os estândares da empresa.	***Must***
RNF05	Os *widgets* devem ser desenvolvidos com as ferramentas indicadas pela equipa da empresa (HTML5, CSS3/Sass, JavaScript (ES6) e a *framework* AngularJS na sua versão 1.5).	***Must***
RNF06	O projeto deve ser desenvolvido utilizando uma arquitetura baseada em componentes (a utilizada no projeto CAOP).	***Must***

Tabela 10 – Requisitos não funcionais (Modificabilidade)

MODIFICABILIDADE		PRIORIDADE
RNF07	Os *widgets* devem admitir facilmente mudanças na sua implementação, como correção de erros e modificações nos requisitos iniciais.	***Should***

Tabela 11 – Requisitos não funcionais (Compatibilidade)

COMPATIBILIDADE		PRIORIDADE
RNF08	Os *widgets* devem permitir a sua fácil integração no projeto CAOP da empresa.	***Must***
RNF09	Os *widgets* devem funcionar corretamente em todos os *browsers*, a exceção de *Internet Explorer* nas suas versões anteriores à 11.	***Should***

Tabela 12 – Requisitos não funcionais (Manutenção)

MANUTENÇÃO		PRIORIDADE
RNF10	O projeto deve ser introduzido num repositório de *software* privado para facilitar a futura manutenção dos *widgets*.	***Must***

Os requisitos anteriormente apresentados já incluem as alterações que surgiram ao longo da fase de prototipagem em relação ao definido nas primeiras reuniões de exploração das características e funcionalidades pretendidas para o projeto. Foi durante o desenho dos protótipos (de baixa e alta fidelidade) e análise dos resultados dos testes de usabilidade que foram completamente definidos todos os requisitos dos *widgets*. Algumas das alterações mais relevantes destes requisitos são apresentadas de seguida.

- Foi adicionado o requisito **RF15** (intervalo de montantes). Funcionalidade que possui atualmente o *eBanka* mas que não foi considerada no levantamento de requisitos inicial do *widget* de pagamento de serviços.
- Acrescentados os requisitos **RF08**, **RF09** e **RF10** para gerir as entidades favoritas dentro do *widget* de pagamento de serviços.
- De forma a complementar o *widget* de operações pendentes foi adicionada a funcionalidade que permite aprovar a operação pendente do utilizador com data mais antiga (**RF29**, **RF30**, **RF31**, **RF32**).

Por outro lado, é importante referir que os requisitos **RF20**, **RF21** e **RF22** não vão ser incluídos nesta versão do projeto, pois estes tratam das possíveis formas de confirmar o pagamento de serviços, e estas funcionalidades já se encontram

implementadas no *eBanka*, pelo que serão acrescentadas ao *widget* quando esteja integrado no sistema.

III.3. PROTOTIPAGEM

Depois de definir os requisitos para cada *widget* da aplicação, foi iniciada a fase de prototipagem, permitindo assim representar os aspetos essenciais da aplicação, esquematizando os elementos que a constituem e a forma como será organizada a informação. Esta etapa facilitou a comunicação ativa com a equipa de desenvolvimento, o responsável do departamento DCS e a equipa de *design*, os quais se encontram estreitamente vinculados com este projeto. Outras vantagens que oferece a elaboração de protótipos é a de permitir avaliar de forma simples a interface do utilizador, melhorar a usabilidade da aplicação e identificar requisitos previamente omitidos, tudo isto antes de iniciar a fase de implementação.

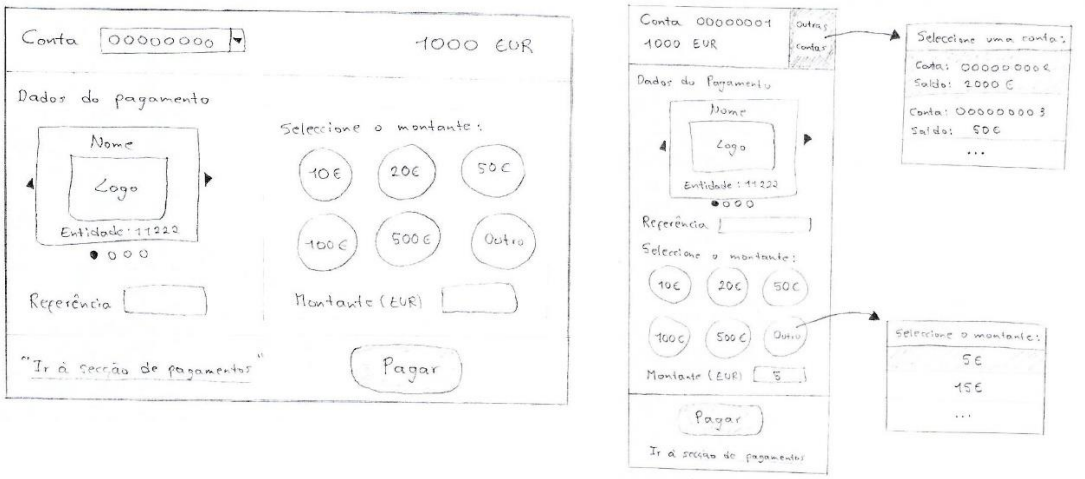
Neste projeto foram desenhados protótipos de baixa e de alta fidelidade para ambos *widgets*. O objetivo de criar os protótipos de baixa fidelidade (PBF) foi o de representar, de uma forma rápida e económica, os aspetos gerais de cada *widget* e a organização dos elementos que compõem as suas respetivas interfaces. Por outro lado, a elaboração dos protótipos de alta fidelidade (PAF) permitiu representar os aspetos mais precisos da interface e criar um desenho mais semelhante à interface final do utilizador.

Para a elaboração dos protótipos de baixa fidelidade, foram utilizadas duas técnicas de prototipagem: *sketching* e *wireframes*, a seguir apresentadas.

III.3.1. *Sketching*

A primeira técnica de prototipagem utilizada nesta fase foi a criação de esboços informais da interface gráfica (*sketches*) efetuados a lápis e sobre papel, que permitiram uma rápida comunicação das ideias e conceitos a todas as partes interessadas do projeto. Foi escolhida esta técnica para trabalhar agilmente com várias ideias e esquematizar os diferentes ecrãs dos *widgets* de forma fácil e com baixo custo, pois não requer investir muito tempo nem aprender a usar ferramentas de prototipagem ou desenho. Além disso, permite realizar facilmente alterações iterativas do desenho [89].

Durante esta fase de prototipagem realizaram-se duas versões dos *sketches* para cada *widget* que depois foram apresentadas ao responsável do departamento DCS para obtenção de *feedback*. Para o caso do *widget* de pagamento de serviços, foi proposto numa primeira versão apresentar no ecrã principal uma única entidade como favorita e introduzir os restantes dados do pagamento de forma explícita pelo utilizador (a referência e montante). Depois de receber o respetivo *feedback* foi criada uma versão final dos *sketches*, onde as alterações mais destacáveis foram: apresentar um maior número de entidades favoritas (num *slideshow*) e listar os valores predefinidos dos montantes (caso existirem para a entidade selecionada, caso contrário, listar valores sugeridos) para assim agilizar a introdução dos dados do pagamento. Esta última versão encontra-se apresentada na Figura 20.

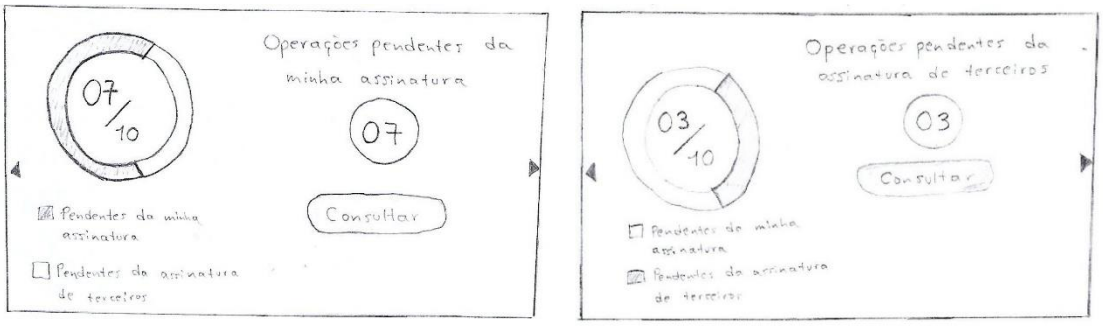


(a) **(b)**

Figura 20 – *Sketches* do *widget* de pagamento de serviços, (a) Versão *Desktop*; (b) Versão *Mobile*.

No que diz respeito ao *widget* de operações pendentes de aprovação, na primeira versão dos *sketches* apresentou-se um desenho que permitia a aprovação e cancelamento das operações pendentes da assinatura do utilizador, listando todas as operações pendentes e um conjunto de funcionalidades. Durante uma reunião realizada com o responsável do departamento DCS, a equipa de desenvolvimento e os *designers*, foi discutido o facto de esta proposta apresentar um conjunto de funcionalidades que podem sobrecarregar o *widget*, portanto, decidiu-se alterar o objetivo do *widget* de maneira que apresente informação visual sobre as operações pendentes e também permita o encaminhamento à respetiva página onde se encontram todas as restantes

funcionalidades. Com a informação obtida do *feedback* foram realizadas as alterações necessárias aos *sketches* para finalmente criar a versão final, apresentada na Figura 21.



(a) **(b)**

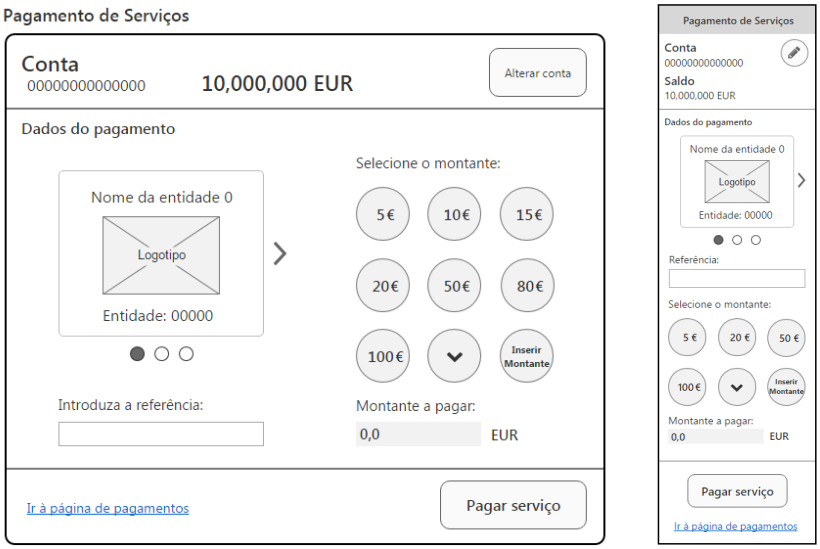
Figura 21 – *Sketches* do *widget* de operações pendentes, (a) Ecrã principal; (b) Ecrã secundário.

É importante referir que as primeiras versões dos *sketches* para cada um dos *widgets* podem ser consultadas no Anexo B.

III.3.2. *Wireframes*

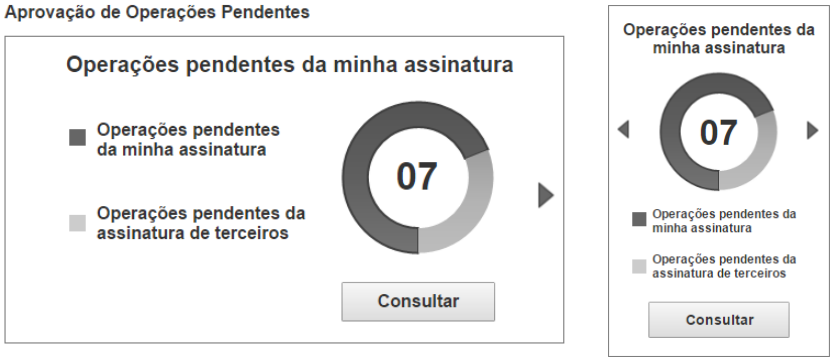
Depois de realizados os *sketches*, foi utilizada a ferramenta de prototipagem **Axure RP** [38] na sua versão gratuita para criar os *wireframes* de cada *widget*. Os *wireframes* são esquemas da interface do utilizador que apresentam a maneira na que será estruturado o conteúdo (sem especificar detalhadamente a sua aparência final) e o comportamento e navegação entre os diferentes ecrãs da aplicação [89]. Com os *wireframes* é possível melhorar a comunicação entre os desenvolvedores, a equipa de *design* e se for o caso com os clientes, além disso, são uma base para a criação do desenho final da interface e para realizar testes aos utilizadores (que permitem detetar possíveis problemas de usabilidade).

Para ambos os *widgets* foram elaboradas duas versões de *wireframes*, a primeira versão, a qual pode ser consultada no Anexo C, foi apresentada numa reunião com o responsável do departamento DCS e integrantes da equipa de desenvolvimento; com base nas ideias e sugestões de melhoria que surgiram realizou-se uma versão final dos protótipos, efetuando as alterações necessárias sobre a versão inicial. Os principais *wireframes* desenhados encontram-se apresentados a seguir (Figura 22 - Figura 24).



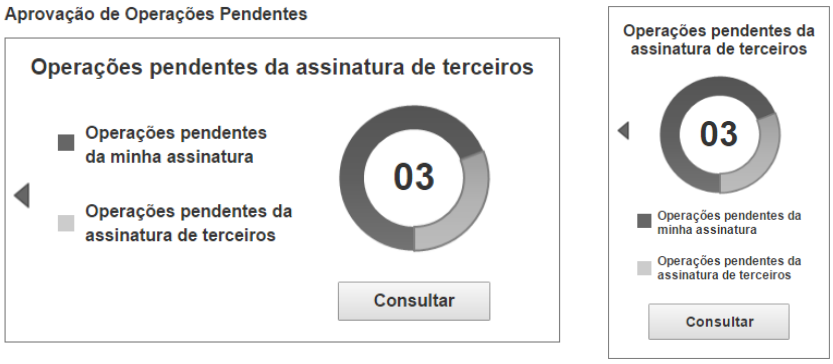
(a) (b)

Figura 22 – *Wireframes* do ecrã principal do *widget* de pagamento de serviços, (a) Versão *Desktop*; (b) Versão *Mobile*.



(a) (b)

Figura 23 – *Wireframes* do ecrã principal do *widget* de operações pendentes, (a) Versão *Desktop*; (b) Versão *Mobile*.



(a) (b)

Figura 24 – *Wireframes* do ecrã secundário do *widget* de operações pendentes, (a) Versão *Desktop*; (b) Versão *Mobile*.

Os *wireframes* dos restantes ecrãs do *widget* de pagamento de serviços podem ser consultados no Anexo C.

III.3.2.1. Testes ao PBF do *Widget* de Pagamento de Serviços

Para garantir uma boa usabilidade e receber o respetivo *feedback* do protótipo de baixa fidelidade (PBF) do *widget* de pagamento de serviços, realizou-se testes de usabilidade aos *wireframes* para um grupo de 15 pessoas, sendo 6 delas parte da equipa de desenvolvimento e 9 potenciais utilizadores finais.

Como o *widget* é parte das funcionalidades do *eBanka* e este sistema é desconhecido pelos utilizadores que participaram no teste, foi exposto um cenário (a modo de descrição) juntamente com o objetivo pretendido a atingir, desta forma os utilizadores entraram em contexto e simularam a realização de um pagamento de serviço, explorando a interface do protótipo e os diferentes casos possíveis de utilização.

Durante os testes foram observadas e apontadas as interações realizadas pelos utilizadores, todos conseguiram realizar as tarefas pretendidas com sucesso, requerendo para isto a utilização de poucos cliques e num curto período de tempo, isto é essencial ao tratar-se de um *widget*. Além disso, os utilizadores apresentaram sugestões, entre as quais se destacam:

- Ante a falta de informações sobre a conta usada para o pagamento, mostrar o nome da conta no ecrã principal e na lista de contas;
- Na janela com o resumo dos dados da operação, apresentar o nome da entidade da qual foi feita o pagamento, por ser esta informação mais intuitiva para o utilizador que o número da entidade;
- Para a interface final, utilizar um ícone representativo ou um descritivo no botão que apresenta a lista de outros montantes, pois gerou confusão em certos casos.

A equipa de desenvolvimento sugeriu possíveis funcionalidades a acrescentar no *widget*. Estas propostas foram apresentadas ao responsável do departamento DCS e depois de analisar as vantagens e desvantagens de cada uma (incluindo os efeitos destas novas características no desempenho e usabilidade do *widget*) foram adicionadas ao *widget* as seguintes:

- Limitar ao utilizador de inserir um montante fora do intervalo que é preestabelecido pela *Banka* para a entidade selecionada, esta característica não tinha sido considerada na fase de levantamento de requisitos.
- Adicionar a opção de remover uma entidade favorita dentro do próprio *widget*.

Ao concluir a fase de testes, foi enviado o protótipo de baixa fidelidade à equipa de *design* da empresa, juntamente com os resultados finais dos testes.

III.3.2.2. Testes ao PBF do *Widget* de Operações Pendentes

Para o protótipo de baixa fidelidade do *widget* de operações pendentes foram realizadas duas fases de testes de usabilidade, na primeira fase participou um grupo de 12 pessoas, sendo 6 participantes parte da equipa de desenvolvimento, 3 utilizadores particulares e 3 utilizadores empresariais.

Da mesma forma que no descrito para o *widget* anterior, foi exposto um cenário para contextualizar aos participantes e foram apontadas todas as interações e comentários obtidos durante os testes.

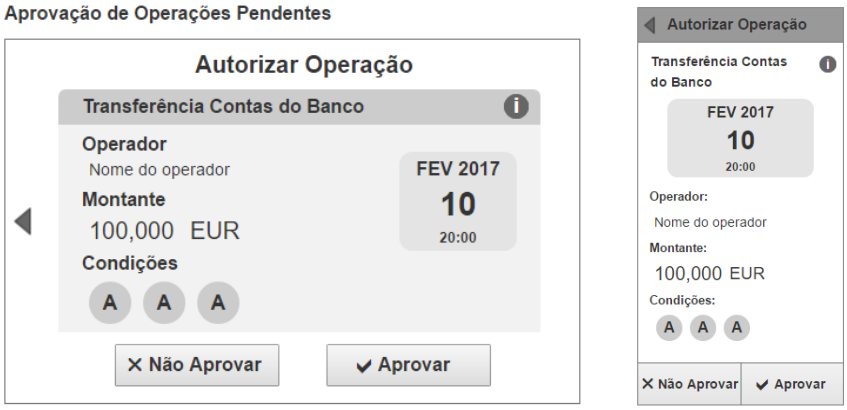
Na primeira fase de testes os participantes coincidiram em que resultavam confusos os dados apresentados nos gráficos, pelo que recomendaram acrescentar no ecrã principal o número total de operações pendentes e agrupar num mesmo ecrã a visualização do número total de operações pendentes da aprovação do utilizador e de terceiros (de outros utilizadores).

Estas recomendações foram apresentadas ao responsável do departamento DCS e aos *designers* da empresa, onde se concluiu uma nova versão do protótipo tendo em consideração o *feedback* obtido nesta fase de testes (os novos *sketches* podem ser consultados no Anexo B).

Como o *widget* ficou bastante simplificado depois destas alterações, foi proposto adicionar uma funcionalidade complementaria, a qual permite ao utilizador aprovar diretamente no *widget* a operação pendente da sua assinatura com a data mais antiga. Por tal motivo foi realizada uma nova versão do protótipo; os principais *wireframes* encontram-se apresentados na Figura 25 e Figura 26.

(a) (b)

Figura 25 – *Wireframes* finais do ecrã principal do *widget* de operações pendentes, (a) Versão *Desktop*; (b) Versão *Mobile*.



(a) (b)

Figura 26 – *Wireframes* finais do ecrã secundário do *widget* de operações pendentes, (a) Versão *Desktop*; (b) Versão *Mobile*.

Esta nova versão do protótipo foi apresentada novamente a um grupo de 11 pessoas para testar a sua usabilidade, entre elas se encontram 4 pessoas da equipa de desenvolvimento, 3 utilizadores particulares e 4 empresariais.

Os resultados foram positivos nesta fase de testes, os participantes conseguiram realizar as tarefas pretendidas sem nenhum inconveniente e usando poucos cliques. As sugestões que apresentaram foram as seguintes:

- Permitir ao utilizador aceder à página principal de operações pendentes em todos os ecrãs do *widget*;
- Apresentar no ecrã de autorização da operação o nome do operador quem executou a operação;

- Alterar o descritivo do botão "*Cancelar*" no ecrã de autorização da operação, pois gerou confusão em certos casos.

Ao concluir esta segunda fase de testes, foi enviado o novo protótipo de baixa fidelidade do *widget* e os resultados dos testes à equipa de *design*.

III.3.3. Protótipos de Alta Fidelidade

Nesta fase foram criados protótipos de alta fidelidade para representar de forma mais detalhada a interface do utilizador e obter um desenho mais próximo ao aspeto final de cada *widget*. Partindo da base de que os protótipos de alta fidelidade se caracterizam por ter em consideração as cores, animações, tipos e tamanhos de letras, interações e todos os elementos que formam parte da interface, foram tidas em conta todas estas características na criação dos protótipos, por este motivo, a sua elaboração tem um custo mais elevado em comparação com os protótipos de baixa fidelidade, mas são muito importantes para conseguir um *feedback* completo da interface e do comportamento dos *widgets* a desenvolver.

Para a criação dos protótipos de alta fidelidade, foi realizada uma reunião com os *designers* da empresa onde foram discutidos os pontos mais importantes a considerar para o desenho da interface final, tendo como base os *wireframes* desenvolvidos anteriormente e as regras internas da empresa aplicadas à interface do sistema *eBanka*.

Foi ponderado por parte dos *designers* alterar algumas formas de apresentação das informações que tinham sido definidas no protótipo de baixa fidelidade, para assim conseguir uma correta adaptação do desenho dos novos *widgets* à interface atual do *eBanka*, utilizando os mesmos elementos e apresentando as informações de forma semelhante, garantindo desta forma a consistência na interface do utilizador.

Por outro lado, decidiu-se nesta fase adicionar uma nova funcionalidade ao *widget* de pagamentos de serviços, a gestão de entidades favoritas. Desta forma, em vez de existir unicamente a opção de remover as entidades favoritas (como tinha sido proposto no protótipo de baixa fidelidade), agora também será possível adicionar novas entidades dentro do próprio *widget*, sem a necessidade de reencaminhar ao utilizador para a página de pagamentos.

Adicionalmente, foi concluído durante uma reunião com os *designers* e a equipa de desenvolvimento o facto de o *widget* de pagamento de serviços ser muito semelhante na versão *Mobile* ao que futuramente será a página de pagamentos do *eBanka* (ainda não desenvolvida) para esta versão, pois a simplicidade que caracteriza o *widget* também deverá existir na versão *Mobile* da página de pagamentos. Pelo que se decidiu mudar a sua finalidade, sendo que a partir desta fase passará a ser considerado como a própria página de pagamentos do *eBanka* (ao invés de um *widget*).

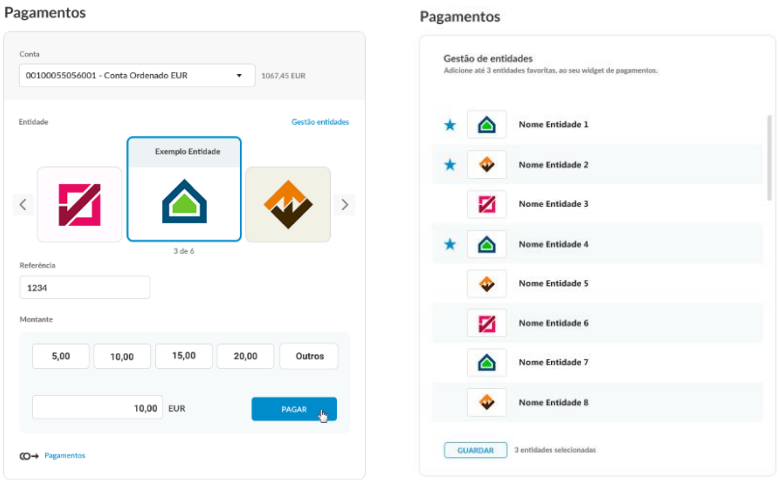
Para finalizar, os protótipos de alta fidelidade foram realizados utilizando a ferramenta de prototipagem **InVision** [40], a qual permitiu simular a interação entre os diferentes ecrãs e obter protótipos mais realistas e adequados para os testes com os utilizadores, que servem como base para o desenvolvimento final da interface. Os desenhos usados para estes protótipos foram criados pelos *designers* da empresa, João Camacho e Guida Santos.

III.3.3.1. PAFs da Versão *Desktop* dos *Widgets*

São apresentados, em seguida, os ecrãs principais dos protótipos de alta fidelidade (PAFs) da versão *Desktop* e descritas as alterações mais significativas que foram realizadas aos *widgets* em relação aos protótipos de baixa fidelidade. Os restantes ecrãs podem ser consultados no Anexo D.

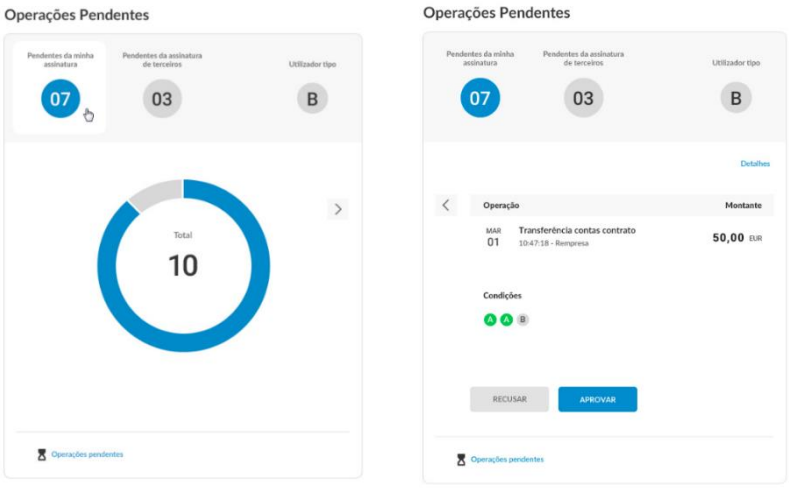
Para esta versão dos *widgets* foi decidido juntamente com os *designers* apresentar o resumo dos dados e a mensagem com o estado da operação fora dos *widgets*, desta forma, os utilizadores finalizarão o processo de pagamento de serviços e de aprovação de operações pendentes na respetiva página do *eBanka*. Esta alteração foi sugerida com o fim de manter a consistência na execução das operações entre os *widgets* e as suas respetivas páginas do *eBanka*.

Apresentam-se na Figura 27 e Figura 28 os ecrãs principais dos PAFs de ambos *widgets*.



(a) (b)

Figura 27 – PAF da versão *Desktop* do *widget* de pagamento de serviços, (a) Formulário de inserção dos dados; (b) Gestão de entidades favoritas.

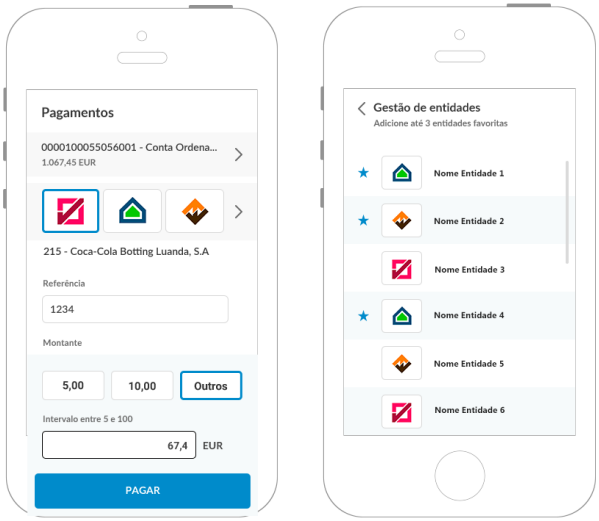


(a) (b)

Figura 28 – PAF da versão *Desktop* do *widget* de operações pendentes, (a) Ecrã principal com a contabilização das operações pendentes; (b) Ecrã secundário, para a aprovação das operações.

III.3.3.2. PAFs da Versão *Mobile*

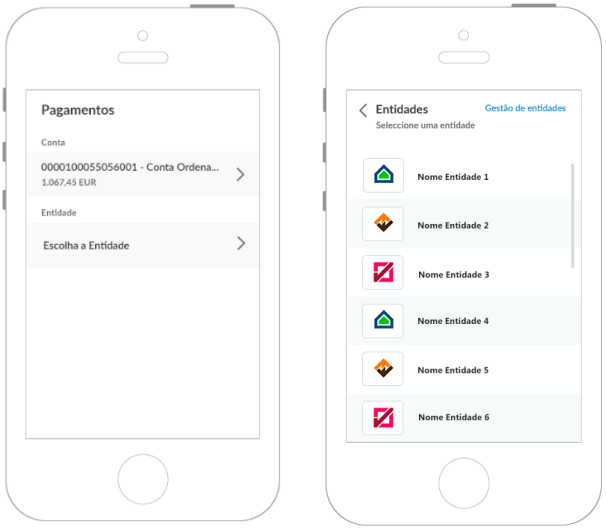
Nesta secção serão apresentados os ecrãs principais dos protótipos de alta fidelidade da versão *Mobile* da página de pagamento de serviços e do *widget* de operações pendentes (os restantes ecrãs podem ser consultados no Anexo D). É importante referir que o sistema *eBanka* não possui atualmente uma versão *Mobile*, pelo que a interface do *dashboard* e certas características ainda não se encontram totalmente definidas. Na Figura 29 ilustra-se os ecrãs principais da versão *Mobile* da página de pagamento de serviços.



(a) (b)

Figura 29 – PAF da versão *Mobile* do *widget* de pagamento de serviços, (a) Formulário de inserção dos dados; (b) Gestão de entidades favoritas.

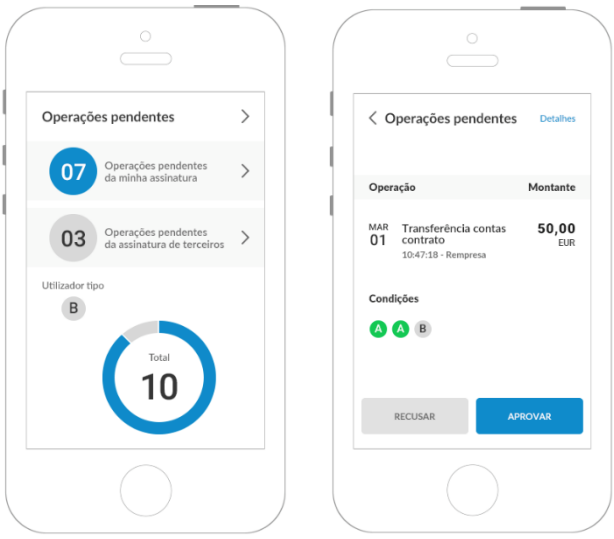
Como a partir desta fase está a ser pensada uma versão mais completa, foi necessário estudar novos casos possíveis que não tinham sido considerados ao tratar-se de um *widget*; esta versão deve permitir ao utilizador fazer pagamento de serviços mesmo sem a existência de entidades favoritas estabelecidas. Os novos ecrãs encontram-se ilustrados na Figura 30.



(a) (b)

Figura 30 – PAF da versão *Mobile* do *widget* de pagamento de serviços, (a) Ecrã principal sem entidades favoritas; (b) Lista de entidades predefinidas.

Por outro lado, os ecrãs principais do PAF do *widget* de operações pendentes para a versão *Mobile* encontram-se apresentados na Figura 31.



(a) (b)

Figura 31 – PAF da versão *Mobile* do *widget* de operações pendentes, (a) Ecrã principal com a contabilização das operações pendentes; (b) Ecrã secundário, para a aprovação das operações.

Depois de criados os protótipos de alta fidelidade, realizou-se um pequeno teste em que foram apresentados os protótipos a 5 pessoas com a finalidade de explorarem todas as suas características livremente e obter *feedback*. Os resultados indicaram que os protótipos já estavam adequados para passar a uma seguinte etapa, pelo que foi iniciada a implementação do projeto, que será descrita no capítulo IV.

III.4. Serviços *Web* e Estruturas de Dados

No departamento DCS da EXICTOS existe uma comunicação ativa entre as equipas de desenvolvimento *front-end* (Canais Presenciais e Não Presenciais) e a equipa de *back-end* (*Middleware*), isto deve-se a que equipa de *Middleware* é a encarregada de criar todos os serviços *Web* utilizados e de processar os dados (provenientes da equipa da *Banka*) para posteriormente serem transmitidos para as aplicações *front-end* através destes serviços. No *front-end* são efetuados os pedidos aos serviços *Web* e os dados recebidos são tratados e apresentados no sistema.

Como o desenvolvimento *back-end* não era parte do âmbito deste projeto, foi considerado o uso temporal do servidor **json-server** [90]. Este servidor oferece uma API REST já criada com todos os métodos HTTP (GET, POST, PUT, PATCH e DELETE) necessários para o consumo de serviços *Web*, os quais se encontram baseados na arquitetura REST apresentada no capítulo II.5.1. Desta forma não foi necessário

aguardar pela equipa de *Middleware* para a criação ou alteração dos serviços *Web* usados, e assim o desenvolvimento do *front-end* do projeto ficou independente da equipa de *back-end*.

Por outro lado, nesta fase foi analisada e construída a estrutura de dados adequada para cada serviço *Web* a utilizar, usando o formato de resposta JSON. No momento em que sejam integrados os *widgets* ao sistema *eBanka*, estas estruturas de dados serão usadas pela equipa de *Middleware* como base para a criação dos serviços *Web* necessários, usando a sua própria API REST ao invés do servidor JSON a ser utilizado para o desenvolvimento.

As estruturas de dados foram criadas de forma simplificada para o projeto, pelo que possuem unicamente as propriedades necessárias para o desenvolvimento dos *widgets*. No Anexo E apresentam-se as estruturas de dados dos serviços *Web* do projeto.

É importante salientar que os pedidos realizados aos serviços *Web* a partir do *front-end* são uma ponte para aceder às bases de dados e estas últimas são transparentes para o projeto, pelo que independentemente do tipo de base de dados a utilizar no *back-end* e as suas características, o tratamento de dados não será afetado no *front-end* sempre que o formato de resposta dos serviços *Web* seja válido.

III.5. ARQUITETURA

Neste capítulo será apresentada a arquitetura para desenvolvimento do lado do cliente utilizada neste projeto curricular, que implementa uma abordagem orientada a componentes e segue o padrão arquitetural MVVM.

III.5.1. Arquitetura Baseada em Componentes com o AngularJS

Com a utilização da *framework* AngularJS no projeto, na sua versão 1.5, consegue-se criar aplicações com uma arquitetura baseada em componentes. Desta forma, a aplicação é dividida em pequenos componentes sendo possível aproveitar as vantagens que esta arquitetura oferece (expostas no capítulo II.7), além de preparar o projeto para a versão mais recente da *framework*, o Angular 2, que está totalmente construída baseada nesta arquitetura.

Os componentes no AngularJS são elementos HTML personalizados que permitem estender o HTML com a criação de novas etiquetas que facilitam a reutilização do código e encapsulam a sua complexidade [91].

Nesta fase do projeto foram analisados os protótipos de alta fidelidade da interface do utilizador para determinar uma correta divisão da aplicação e decidiu-se a criação de um componente genérico para servir como modelo ou *template* de todos os *widgets* da aplicação; este componente determina que todo *widget* baseado neste *template* deve possuir um cabeçalho (*header*), um conteúdo (*content*) e um rodapé (*footer*) e permitirá partilhar as características em comum entre os *widgets*, como por exemplo, os estilos CSS.

Cada *widget* tem um componente principal (*Smart Component*) para interagir com os serviços *Web* (obtenção e atualização dos dados no servidor) e que se comunica com os seus descendentes (*Dumb Component*) para estes últimos tratar dos dados recebidos.

No Anexo F encontram-se esquemas com os componentes que integram o *widget* de pagamento de serviços e o *widget* de operações pendentes. É importante salientar que não são os únicos do projeto, já que existem outros componentes para casos específicos, como por exemplo, gestão de entidades, resumos dos dados e mensagens com o estado das operações.

Cada componente criado tem uma única responsabilidade e conhece apenas os dados do seu controlador. Para evitar dependências, estes componentes não devem aceder diretamente aos dados de outros componentes, pelo que a melhor solução é utilizar mecanismos para troca de dados, que são oferecidos pela própria *framework* (como por exemplo, os *bindings*).

Desenvolver uma aplicação utilizando esta arquitetura facilita significativamente a reutilização do código e, para o caso deste projeto, a maioria dos componentes criados para a versão *Desktop* foram reutilizados para a versão *Mobile* da aplicação, mesmo existindo certas diferenças na apresentação do conteúdo entre as duas versões. Desta forma, para a versão *Mobile* foram unicamente criados os novos componentes que são específicos da versão, como por exemplo, o menu.

Por outro lado, é importante mencionar que no momento de decidir quais são os componentes que integram uma aplicação *Web* e como esta será dividida se tem de garantir um equilíbrio no número de componentes criados, porque um projeto extremamente modular pode tornar-se difícil de gerir.

III.5.2. Padrão Arquitetural MVVM

A implementação dos *widgets* neste projeto também seguiu a arquitetura MVVM (*Model-View-ViewModel*) a qual é definida pela *framework* utilizada.

AngularJS implementa o padrão de desenho MVC (*Model-View-Controller*), mas frequentemente numa variante muito estendida no mundo de JavaScript, conhecida com o nome MV* ou MVVM (*Model-View-ViewModel*). Estes padrões permitem a separação do código dependendo da sua responsabilidade, no AngularJS o MVC funciona da seguinte forma [92]:

- **Vistas:** as Vistas são o HTML e tudo o que apresente dados ou informação.
- **Controladores:** os Controladores encarregam-se da lógica da aplicação.

No AngularJS o "Modelo" é algo mais do que normalmente se entende quando se fala do MVC tradicional, o Modelo nesta *framework* é o "*Scope*". O *Scope* define-se como um objeto JavaScript que permite comunicar as Vistas (HTML) com os Controladores (JavaScript) e vice-versa [92], [93].

- **Modelo da vista (*ViewModel*):** O *ViewModel* são os dados da aplicação e os dados adicionais necessários para serem mostrados adequadamente [92]. Neste padrão, em vez de controlar manualmente as alterações nas Vistas e nos dados, estes são atualizados diretamente quando sofrem alterações, por exemplo, uma Vista atualiza um dado que está a ser apresentado e, por sua vez, é atualizado no Modelo automaticamente, e vice-versa [94].

IV. IMPLEMENTAÇÃO

IV.1. INTRODUÇÃO

Após definidos os requisitos, o desenho da interface do utilizador e a arquitetura do projeto, foi iniciada a fase de implementação. Neste capítulo, serão primeiramente citadas as tecnologias utilizadas neste projeto, escolhidas com base nas ferramentas usadas pela equipa da EXICTOS no projeto CAOP.

Ao tratar-se de aplicações *Web* surge a necessidade de utilizar *frameworks* ou livrarias JavaScript para facilitar o seu desenvolvimento e otimizar o desempenho, para tal, foi usada a *framework* AngularJS (anteriormente estudada e comparada com outras alternativas possíveis no capítulo II.8.2). Esta *framework* será exposta com maior detalhe neste capítulo juntamente com as outras tecnologias utilizadas durante a implementação do projeto.

Adicionalmente, será apresentada a estrutura de ficheiros do projeto, realizada tendo em consideração as duas versões do projeto (*Desktop* e *Mobile*) e os dois módulos previamente definidos durante o levantamento de requisitos (o *widget* de pagamentos de serviços e o *widget* de operações pendentes de aprovação).

Posteriormente no capítulo, será descrito o processo para a implementação dos *widgets*, apresentando uma breve descrição das suas funcionalidades e os aspetos mais relevantes da sua implementação. Além disso, será exposto o processo de *refactoring*, que permitiu aumentar a qualidade do código durante o desenvolvimento sem afetar o funcionamento do projeto.

Finalmente serão explicados os testes de usabilidade realizados à interface do utilizador e apresentadas as alterações que foram feitas ao projeto com base nos resultados obtidos. Além disso, será descrita a forma como foi documentado o código do projeto, para o qual foi usado o gerador de documentação para JavaScript "**JSDoc**".

IV.2. LINGUAGENS E TECNOLOGIAS UTILIZADAS

Neste capítulo serão apresentadas as diferentes tecnologias utilizadas para o desenvolvimento do projeto, nomeadamente, a linguagem JavaScript e as ferramentas AngularJS, Node.js, Sass e Git, as quais são também usadas pela equipa de Canais Não Presenciais no projeto CAOP (onde serão integrados os *widgets*). Além destas tecnologias, serão descritas outras que foram escolhidas livremente para este projeto em específico como, por exemplo, a livraria HighCharts, o serviço de alojamento BitBucket e módulos externos do AngularJS criados pela comunidade de desenvolvedores.

IV.2.1. ECMAScript 6 e Babel

O projeto foi desenvolvido utilizando a linguagem de programação **JavaScript**, na sua versão mais recente **ECMAScript 6 (ES6)**. ECMAScript é o estândar que define como deve ser esta linguagem, a qual é interpretada e processada por múltiplas plataformas, entre estas, os *browsers* [95].

ES6 traz muitas alterações significativas na linguagem, não só na sintaxe, mas também na incorporação de novas funcionalidades, como por exemplo, a introdução de "classes" que são geralmente usadas noutras linguagens conhecidas como Java e PHP (que suportam à Programação Orientada a Objetos).

Os principais *browser*s já implementam a maioria destas funcionalidades. No entanto, como esta versão está ainda em processo de adaptação, foi utilizada a ferramenta **Babel** [96], que permite transformar o código JavaScript desenvolvido com ES6 para uma versão da linguagem mais antiga que possa ser interpretada por todos os *browsers*.

IV.2.2. *Framework* AngularJS

A *framework* **AngularJS** é um projeto de código aberto desenvolvido pela empresa *Google* que usa a linguagem JavaScript. Contém um conjunto de bibliotecas úteis para o desenvolvimento de aplicações *Web* avançadas no lado do cliente e permite criar aplicações de uma única página, ou seja, carregar diferentes partes da aplicação

sem ter de recarregar todo o conteúdo no *browser*. É possível encontrar o projeto do AngularJS no *site* oficial desta *framework* [49], [92], [93].

Como referido no capítulo III.5.2, AngularJS permite criar aplicações utilizando o padrão de desenho MVC ou alguma das suas variantes (MV*), e uma das suas características mais relevantes é o *Two-Way Data Binding*, que consiste em observar continuamente as mudanças que ocorrem, quer na Vista, quer no Modelo e sincronizar os dados entre eles, como ilustrado na Figura 32 [97].

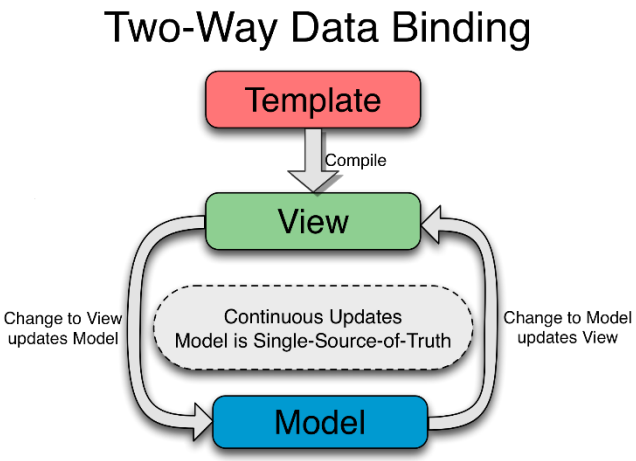


Figura 32 – *Two-Way Data Binding* no AngularJS [98].

Outra característica fundamental do AngularJS é a possibilidade de criar Módulos, os quais podem ser definidos como contentores isolados onde se encontra o código e que permitem o desenvolvimento de aplicações com código facilmente reutilizável e organizado [99]. AngularJS, a partir da sua versão 1.5, também permite desenvolver projetos seguindo uma arquitetura baseada em componentes.

IV.2.3. Node.js e NPM

Node.js [100] é uma plataforma para desenvolvimento de aplicações *Web* no lado do servidor baseadas em rede utilizando JavaScript. Foi concebida para construir aplicações de alto desempenho e escaláveis [101].

Node.js possui um poderoso gestor de pacotes chamado **NPM (*Node Package Manager*)** que permite instalar e atualizar pacotes, gerir as suas dependências e versões e também criar pacotes próprios e publicá-los *online* para toda a comunidade. Estes pacotes podem ser encontrados no repositório oficial do NPM [102], [103].

IV.2.4. Módulos externos – *Modal* e *Slideshow*

Para simplificar o desenvolvimento dos *widgets* foram utilizados módulos criados pela comunidade. A seguir apresenta-se uma breve descrição de cada um deles:

- ***ngDialog*** - Este módulo encontra-se disponível para o AngularJS e foi utilizado no *widget* de pagamento de serviços para apresentar janelas modais com a informação pretendida, nomeadamente, a janela para a gestão de entidades e para outros montantes. Este módulo pode ser encontrando no seu repositório do GitHub [104].
- ***ui-carousel*** - Este módulo permite criar *slideshows* no AngularJS; no caso deste projeto foi utilizado para o *slideshow* que apresenta as entidades favoritas no *widget* de pagamento de serviços. O módulo pode ser encontrado no seu repositório oficial do NPM [105].

IV.2.5. Livraria HighCharts

Para o gráfico do *widget* de operações pendentes foi utilizada a livraria **HighCharts** [106] a qual se encontra escrita na linguagem JavaScript e oferece um método fácil e interativo para adicionar gráficos às aplicações *Web* [107].

Esta livraria é de código aberto e as características dos gráficos podem ser personalizadas, permitindo assim uma alta flexibilidade. Pode ser utilizada de forma gratuita e é compatível com todos os *browsers* modernos [107].

IV.2.6. Pré-processador Sass e *Framework* Bootstrap

No desenvolvimento de aplicações *Web* são atualmente usados pré-processadores CSS[17] para otimizar a criação de estilos CSS; para este projeto foi utilizado o pré-processador Sass (*Syntactically Awesome StyleSheets*). **Sass** [109] é uma extensão de CSS que adiciona potência e elegância à linguagem CSS básica, permite utilizar variáveis, regras CSS aninhadas, importações internas, etc., pelo que as folhas de estilos ficam organizadas e o código otimizado [110].

[17] Ferramenta que converte um conjunto de instruções para CSS. Os mais conhecidos são Sass e Less, ambos são um subconjunto da linguagem CSS, mas com funcionalidades adicionais [108].

Por outro lado, foi usado no projeto Bootstrap para agilizar o processo de criação de estilos CSS. **Bootstrap** [111] é uma *framework* de código aberto desenvolvida pela empresa *Twitter* que permite criar de forma simples interfaces *Web* com desenho adaptativo, ou seja, que se adaptam aos diferentes dispositivos e tamanhos de ecrã, e oferece uma variedade de estilos e elementos HTML que podem ser personalizados conforme o projeto.

IV.2.7. Git, SourceTree e BitBucket

O sistema de controlo de versões **Git** [67], desenvolvido por Linus Torvalds, é um dos VCS (do inglês *Version Control System*) de código aberto e de distribuição livre mais conhecidos e usados na atualidade [64].

Para facilitar o uso do Git, foi recomendada pela equipa de desenvolvimento da empresa a aplicação **SourceTree** [112] que permite trabalhar com Git facilmente através de uma interface gráfica.

Adicionalmente foi utilizado para este projeto o serviço de alojamento **BitBucket** [73], sumamente útil para os projetos que utilizam o VCS Mercurial ou Git. BitBucket permite a gestão de repositórios de *software* online de forma gratuita e privada.

IV.3. Estrutura de Ficheiros

Seguindo as boas práticas recomendadas para o desenvolvimento de aplicações *Web* e com base na arquitetura utilizada foi decidido estruturar os ficheiros que compõem o projeto por componentes, onde criou-se um diretório para cada componente e dentro deste foram inseridos os ficheiros relacionados ao respetivo componente, nomeadamente, as vistas (HTML), os controladores (JS), as folhas de estilos (SCSS) e os restantes ficheiros relacionados.

Esta abordagem permite gerir facilmente as aplicações consideradas escaláveis, nas que são adicionados novos elementos ao projeto de forma recorrente, além disso, permite alterar ou remover componentes sem afetar os diretórios dos restantes.

Depois de desenvolver a versão *Desktop* do projeto seguindo esta abordagem, foram consideradas duas alternativas para reestruturar os ficheiros do projeto de forma a incluir a versão *Mobile*:

- A primeira alternativa consistia em utilizar os mesmos ficheiros da versão *Desktop* para a versão *Mobile* e usar *breakpoints*[18] para diferenciar o código CSS utilizado em cada versão.
- A segunda alternativa consistia em separar ambas versões em dois módulos diferentes (um diretório para *Desktop* e um para *Mobile*), e todos os ficheiros (vistas, controladores, etc.) que são partilhados em ambas versões passariam a estar num diretório em comum.

Como o uso de *breakpoints* gera um código CSS bastante carregado e no projeto existem diferenças entre as versões *Desktop* e *Mobile* não só nos estilos mas também ao nível funcional, foi escolhida a segunda alternativa, esta permite seguir as boas práticas de desenvolvimento, como ter um código modular, organizado e reutilizável. No Anexo G encontra-se exemplificado parte da estrutura de ficheiros do projeto.

IV.4. FUNCIONALIDADES IMPLEMENTADAS

Após definir a estrutura de ficheiros do projeto foram implementadas inicialmente as funcionalidades com maior prioridade, determinadas na fase de levantamento de requisitos (descrita no capítulo III.2). Para tal, o projeto foi dividido em dois módulos: um módulo para a versão *Desktop* e outro para a versão *Mobile*, como ilustrado na Figura 33.

[18] Os *breakpoints* são condições da largura do ecrã dos dispositivos, definem quando cada bloco de CSS será utilizado, ou seja, quando um desenho deve ser alterado para outro para se adaptar à largura do ecrã [113].

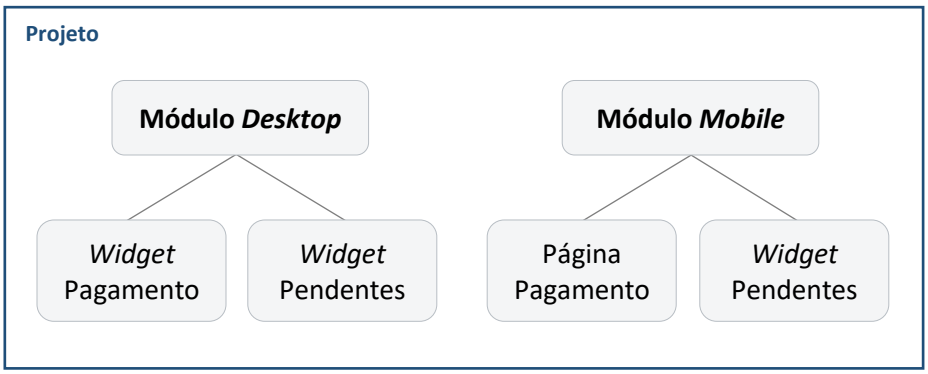


Figura 33 – Divisão do projeto por versões (*Desktop* e *Mobile*).

Como o sistema *eBanka* ainda não possui uma versão *Mobile*, foi considerado o desenvolvimento dos *widgets* para esta versão menos prioritários para a empresa, pelo que decidiu-se iniciar com a implementação da versão *Desktop*.

IV.4.1. Implementação da Versão *Desktop*

A seguir são descritos os aspetos mais relevantes da implementação dos *widgets* para a versão *Desktop* do projeto.

IV.4.1.1. *Widget* de Pagamento de Serviços

Foi iniciada a implementação do *widget* de pagamento de serviços com a obtenção, através de pedidos aos serviços *Web* criados (*ver capítulo* III.4), dos dados das contas que o utilizador possui no banco e de todas as entidades disponíveis para pagamentos.

Com o pedido para obter as entidades disponíveis é possível saber também quais são as entidades que se encontram estabelecidas como favoritas para o utilizador e a lista de montantes associada a cada entidade. As entidades favoritas são apresentadas no *widget* na forma de *slideshow*, permitindo ao utilizador selecionar a pretendida para o pagamento.

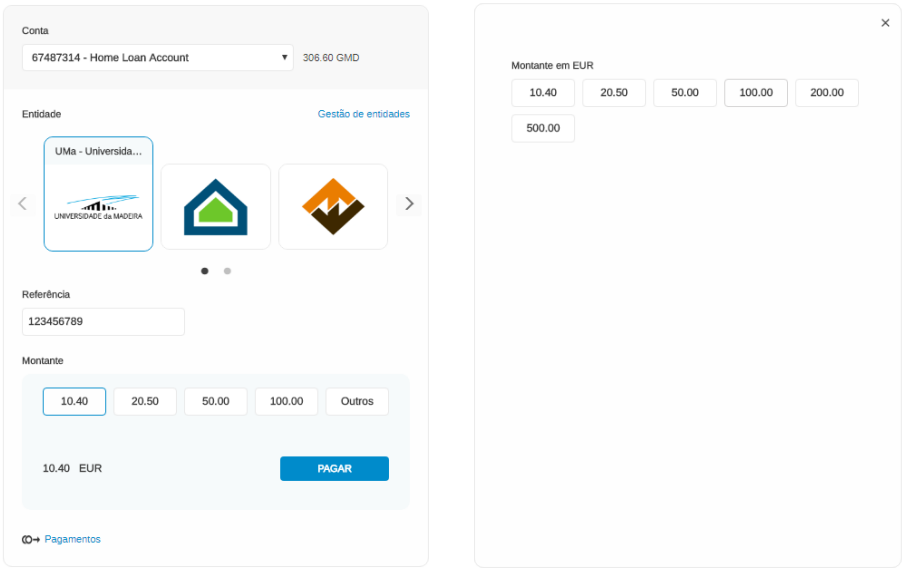
- **Casos dos Montantes**

Devido à natureza desta operação, os montantes permitidos nos pagamentos de serviços dependem da entidade escolhida, pelo que existem três cenários possíveis:

O cenário em que a entidade possui uma lista com montantes já predefinidos (em que o utilizador só pode escolher entre um deles) e aparecem estes valores como opções no formulário.

Neste caso quando a lista excede o número máximo de montantes que podem ser apresentados no ecrã principal do *widget* são mostrados os montantes de menor valor no formulário e os restantes passam a ser apresentados noutra janela, onde o utilizador pode escolher o valor pretendido entre todos os montantes disponíveis. A decisão de apresentar inicialmente os montantes de menor valor foi tomada juntamente com a equipa da empresa.

Na Figura 34, apresenta-se o ecrã principal do *widget* de pagamentos de serviços e a janela com todos os montantes predefinidos para a entidade escolhida.



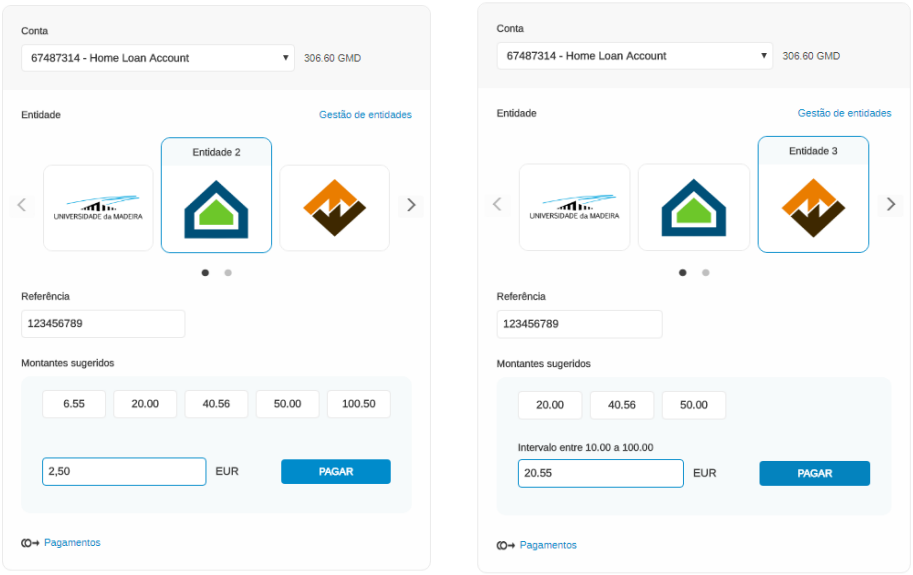
(a) **(b)**

Figura 34 – Versão *Desktop* do *widget* de pagamentos de serviços, (a) Formulário de inserção de dados; (b) Janela de montantes predefinidos.

Existem outros dois cenários onde a entidade escolhida permite ao utilizador introduzir um montante (seja este qualquer montante ou um valor dentro de um intervalo). Em ambos casos são mostrados no *widget* montantes sugeridos, pelo que o utilizador pode introduzir ele próprio o valor desejado ou escolher entre as opções sugeridas.

Estes montantes sugeridos são valores atualmente preestabelecidos, mas o pretendido é que numa versão futura do *widget* apareçam valores adaptados ao perfil do utilizador, por exemplo, durante o desenvolvimento do projeto foi proposta como ideia para implementação futura que estes montantes sejam os que o utilizador usa com maior frequência nos seus pagamentos de serviços.

Na Figura 35, apresenta-se o ecrã principal do *widget* nos casos em que o utilizador pode introduzir um montante qualquer ou dentro de um intervalo.



(a) **(b)**

Figura 35 – Introduzir o montante do pagamento, (a) Montante em aberto; (b) Montante dentro de um intervalo.

- **Gestão de Entidades**

Depois de implementados todos os requisitos de maior prioridade deste *widget*, foi acrescentada a funcionalidade para gerir as entidades favoritas (requisito adicionado na fase dos protótipos de alta fidelidade, mas com uma prioridade baixa).

Para tal, são apresentadas todas as entidades predefinidas pelo banco numa nova janela que permite ao utilizador estabelecer quais são as suas favoritas. Cabe destacar que inicialmente se pensou em limitar o número de entidades favoritas (RF09), mas este requisito funcional foi eliminado depois de avaliar o funcionamento do *slideshow*, pois concluiu-se que não era necessário, pelo menos nesta versão, limitar ao utilizador neste sentido. O ecrã da janela de gestão das entidades favoritas encontra-se ilustrado no Anexo H.

IV.4.1.2. *Widget* de Operações Pendentes

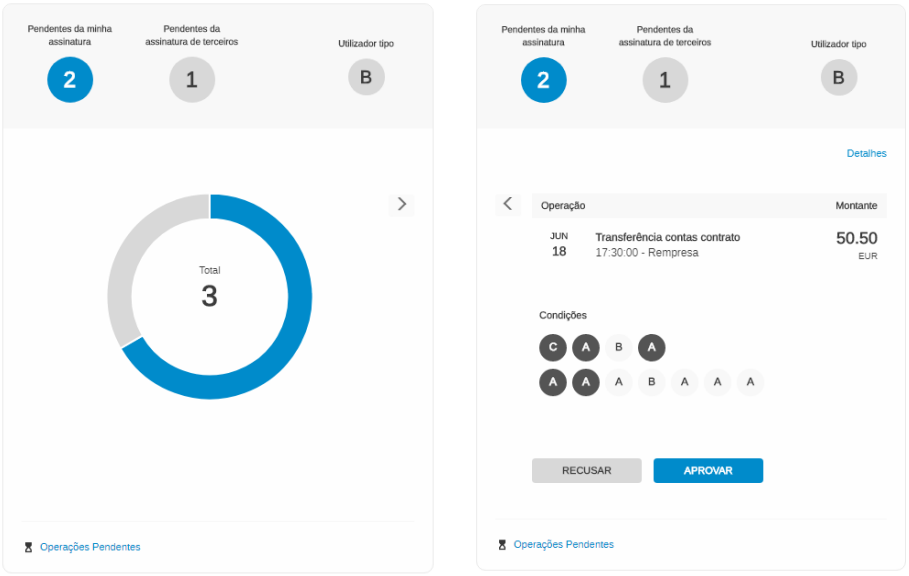
A implementação do *widget* de operações pendentes foi iniciada da mesma forma que o *widget* anterior, com a obtenção dos dados necessários através de pedidos

aos respetivos serviços *Web*, para assim saber o tipo do utilizador e as suas operações pendentes de aprovação.

Com estes dados, foi possível apresentar no ecrã principal do *widget* a contagem das operações pendentes da aprovação do utilizador e a contagem das que já aprovou mas cuja execução depende da aprovação de outros utilizadores.

O segundo ecrã do *widget* apresenta a operação pendente da aprovação do utilizador mais antiga (ou seja, aquela que tem mais tempo a sua espera para ser aprovada) e será possível aprovar (ou rejeitar) esta operação quando o *widget* se encontre integrado no *eBanka*.

Ambos ecrãs do *widget* de operações pendentes são ilustrados na Figura 36.



(a) **(b)**

Figura 36 – Versão *Desktop* do *widget* de operações pendentes, (a) Ecrã principal; (b) Ecrã secundário.

- **Condições de Aprovação**

No segundo ecrã do *widget* são apresentados os dados mais relevantes da operação pendente, incluindo as suas condições de aprovação. No âmbito bancário, estas condições funcionam da seguinte forma:

Cada utilizador possui um "tipo de assinatura" associado conforme as suas responsabilidades na empresa, e cada tipo tem um nível de autorização (**A**, **B** ou **C**, onde

o tipo **A** corresponde aos utilizadores com mais autorização e o tipo **C** com menos). A execução das operações pendentes é concluída quando se cumpre uma das condições de aprovação.

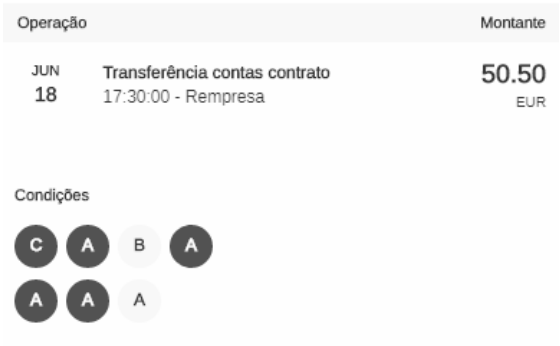


Figura 37 – Operação pendente e respetivas condições de aprovação.

Para tal, cada condição precisa das assinaturas dos utilizadores para serem cumprida, por exemplo, a operação pendente da Figura 37 (uma transferência) possui duas condições: (**C - A - B - A**) ou (**A - A - A**)**.** A primeira condição já tem uma assinatura do tipo C e duas do tipo A, pelo que está em falta unicamente uma assinatura do tipo B (ou de nível superior) para ser aprovada esta operação. No momento em que um utilizador do tipo B ou de nível de autorização superior (por exemplo, do tipo A) aprove/assine a operação, esta transferência será concluída.

Por outro lado, se um utilizador do tipo A aprovar esta operação, também será cumprida a segunda condição, pelo que igualmente será executada a transferência.

Para finalizar, é importante referir que neste *widget* existem 3 requisitos que foram parcialmente implementados: os requisitos RF27, RF28 e RF33, os quais indicam que o utilizador deve poder aceder à página de operações pendentes do *eBanka* para consultar informações mais específicas sobre as operações pendentes. Como é necessário que o *widget* esteja integrado no sistema para implementar estas funcionalidades, foi realizada unicamente uma simulação do reencaminhamento para a página.

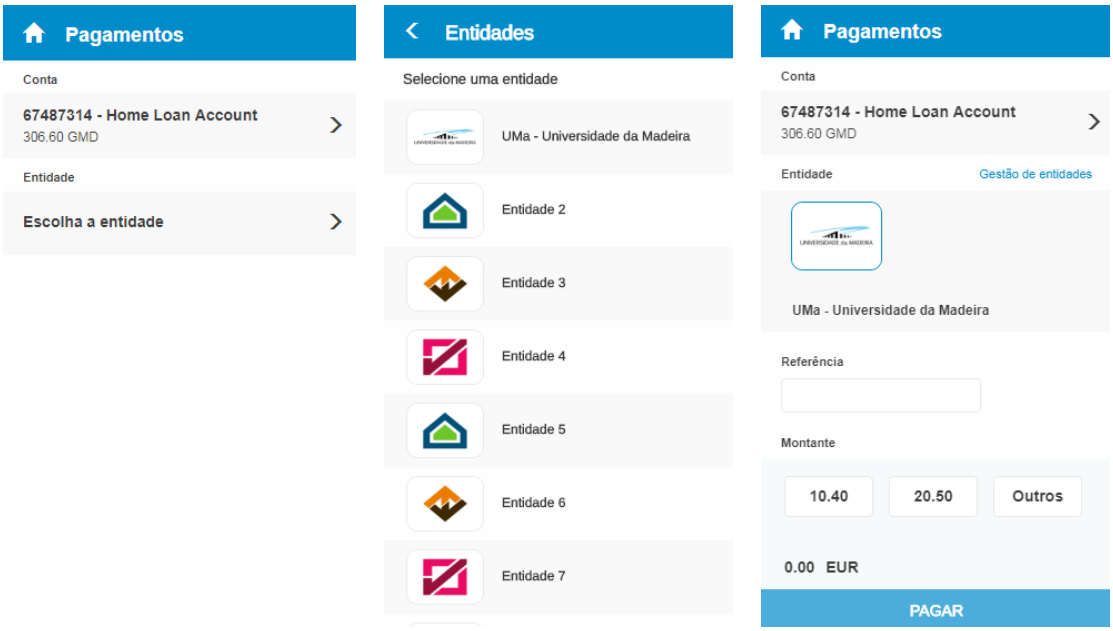
IV.4.2. Implementação da Versão *Mobile*

São descritos a seguir os aspetos mais relevantes da implementação da página de pagamentos de serviços e do *widget* de operações pendentes para *Mobile*. Muitas das características de esta versão são semelhantes às apresentadas na versão *Desktop*.

IV.4.2.1. Página de Pagamento de Serviços

Como nesta versão do projeto foi desenvolvida a própria página de pagamento de serviços (e não um *widget* como na versão *Desktop*), foi implementado o caso em que o utilizador pode realizar um pagamento mesmo sem a existência de entidades favoritas preestabelecidas.

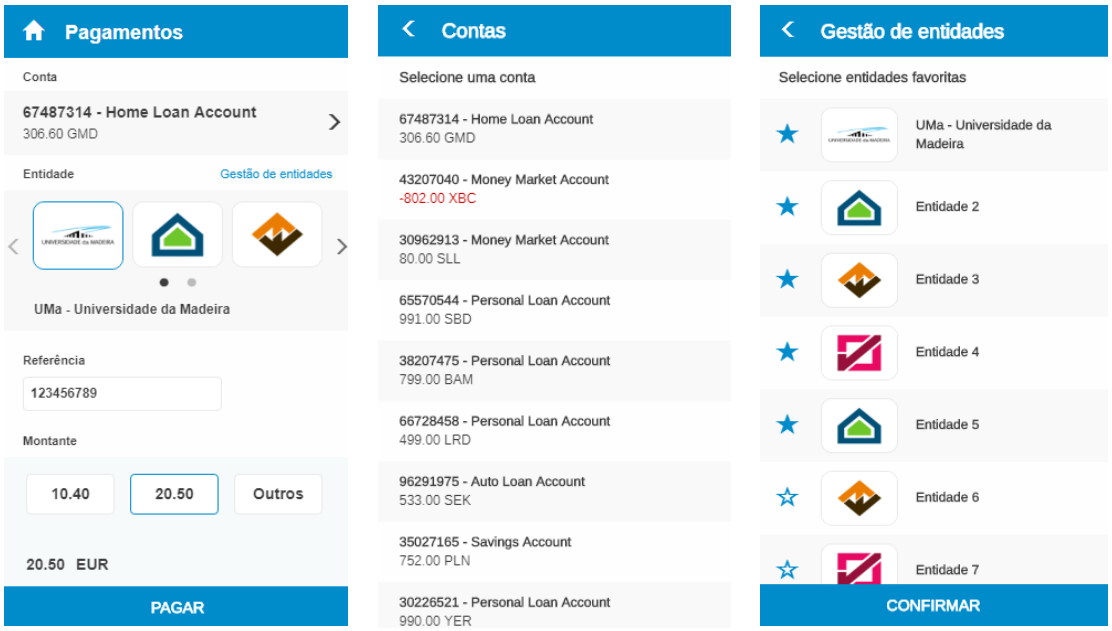
Para tal, foi criado um novo ecrã que, além de ser possível alterar a contar a debitar, permite também selecionar uma entidade para fazer o pagamento entre todas as entidades predefinidas do banco, como ilustrado na Figura 38. A entidade escolhida passa a ser favorita por defeito e, ao igual que na versão *Desktop*, esta versão também permite gerir as entidades favoritas no ecrã principal.



(a) (b) (c)

Figura 38 – Versão *Mobile* da página de pagamento de serviços sem entidades favoritas, (a) Ecrã principal; (b) Lista de entidades predefinidas; (c) Ecrã principal após escolhida uma entidade.

Por outro lado, a Figura 39 ilustra o ecrã principal da versão *Mobile* com o formulário para o preenchimento dos dados do pagamento, a lista de contas do utilizador e o ecrã para gerir as entidades favoritas.



(a) (b) (c)

Figura 39 – Versão *Mobile* da página de pagamento de serviços, (a) Formulário de inserção de dados; (b) Lista de contas do utilizador; (c) Gestão de entidades favoritas.

A diferença da versão *Desktop*, nesta versão foi implementado o ecrã do resumo da operação (com os dados do pagamento preenchidos pelo utilizador) e o ecrã com a mensagem do estado da operação (que notifica se esta foi ou não efetuada com sucesso). Estes ecrãs encontram-se ilustrados no Anexo H.

Cabe ressaltar que a execução completa do pagamento de serviços encontra-se parcialmente implementada (RF01), pois é necessário integrar esta funcionalidade ao sistema *eBanka* para a sua completa implementação.

IV.4.2.2. *Widget* de Operações Pendentes para *Mobile*

No que diz respeito ao *widget* de operações pendentes para *Mobile*, este possui características muito semelhantes ao *widget* da versão *Desktop*. Na Figura 40 a seguir apresentada ilustram-se os ecrãs principais.

(a) (b)

Figura 40 – Versão *Mobile* do *widget* de operações pendentes, (a) Ecrã principal; (b) Ecrã secundário.

Nesta versão encontra-se implementada a aprovação e rejeição das operações pendentes, que no momento de ser concretizada se atualiza automaticamente a contagem das operações pendentes no ecrã principal.

Todos os restantes ecrãs do projeto desenvolvido podem ser consultados no Anexo H.

IV.4.3. Interface do Utilizador

Entre os aspetos mais importantes considerados no momento de desenvolver a interface gráfica do projeto está que as características da interface do sistema *eBanka* variam conforme os seus clientes (entidades bancárias), ou seja, durante o desenvolvimento é criado um desenho base e posteriormente as suas propriedades são adaptadas para cada cliente (como por exemplo, as cores, elementos, comportamento, etc.).

Foi utilizado o pré-processador Sass para facilitar a criação dos estilos CSS dos *widgets*, pois este permite o uso de variáveis para reutilizar em todo o estilo do projeto os valores das propriedades que estão armazenados nelas. Além disso, foi criada uma folha de estilos que contém unicamente estas variáveis, isto com o fim de facilitar a modificação das propriedades CSS no futuro, quando o desenho tenha de ser adaptado às características dos clientes.

Para agilizar a criação dos estilos, foram utilizados componentes da *framework* Bootstrap, e para garantir que o projeto não depende diretamente desta *framework*, foram sobrescritas às classes de estilos do Bootstrap para classes próprias do projeto. Desta forma, quando seja necessário alterar ou eliminar alguma classe utilizada do Bootstrap só será modificado o CSS do projeto, o que evita alterar as vistas HTML.

Outro ponto a considerar é que o *eBanka* possui uma versão *Desktop* não responsiva, pelo que existem características da interface para dispositivos móveis (como o menu e *dashboard*) que ainda não foram completamente definidas e que para este projeto foi apenas feito um desenho provisório.

A interface gráfica da versão *Mobile* do projeto é responsiva, e na versão *Desktop* os *widgets* possuem dimensões fixas e iguais, com o fim de assegurar que as dimensões dos *widgets* se adequam corretamente ao *dashboard* do *eBanka*, juntamente com os outros *widgets* já existentes (que possuem também dimensões fixas).

IV.5. *REFACTORING*

Refactoring é o processo de engenharia de *software* que permite melhorar a estrutura interna do projeto modificando o seu código, sem alterar o seu comportamento. É sumamente útil para criar *software* fácil de manter e compreensível para todos os integrantes da equipa. Além disso, permite melhorar o produto e corrigir e evitar possíveis erros [114].

Durante o desenvolvimento do projeto, foi realizado este processo de maneira iterativa, onde foram seguidas as recomendações dadas pela equipa de desenvolvimento para otimizar o código e seguidas as boas práticas de JavaScript existentes na atualidade.

Para facilitar o seguimento destas boas práticas durante a etapa de implementação e *refactoring*, existem *plugins* que podem ser instalados nos editores de código-fonte e que oferecem um mecanismo para uniformizar a maneira como é escrito o código através de regras definidas. Estes *plugins* apresentam alertas durante a implementação nos casos em que sejam detetados erros ou incumprimento de alguma destas regras.

Ter um estilo consistente no código é importante para manter a capacidade de leitura e compreensão, pelo que foi utilizado para este projeto o *plugin* **ESLint** [115] para JavaScript e seguida a guia de regras de estilos de **Airbnb** [116] que oferece um conjunto de regras baseadas nas boas práticas de programação e está a ser mantida e melhorada continuamente pela comunidade de desenvolvedores. Esta guia de estilos pode ser encontrada no seu repositório de GitHub [117].

IV.6. TESTES DE USABILIDADE DA INTERFACE

Após a fase de implementação do projeto, segue-se a realização de testes com utilizadores finais para conseguir detetar problemas de usabilidade que não foram previamente identificados durante a fase de prototipagem. Além disso, a etapa de testes permite obter um *feedback* mais concreto do aspeto final e funcionalidades do projeto.

Por conseguinte, foi decidido fazer os testes de usabilidade a um número total de 10 utilizadores, divididos em 2 fases: numa primeira fase para um número de 5 pessoas e numa segunda fase para as outras 5. Segundo o gráfico de Nielsen apresentado na Figura 41, com 5 utilizadores já é possível detetar aproximadamente o 85% dos problemas de usabilidade da interface e para um número de 10 utilizadores o 95% dos problemas [118].

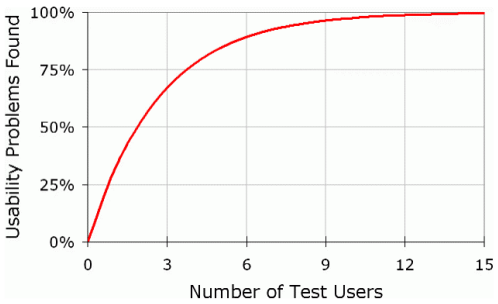


Figura 41 – Gráfico de Nielsen [118].

Nielsen [118] também afirma que, com poucos utilizadores, se consegue detetar rapidamente a maioria dos problemas de usabilidade, e que, à medida que são adicionadas novas pessoas, serão encontradas menos novidades e os problemas mais frequentes serão repetidos, pelo que sugere que, ao invés de fazer diretamente um teste para 10 ou 15 pessoas, o ideal é fazer testes mais pequenos e em várias iterações. Desta forma, numa primeira etapa será possível detetar os erros mais críticos, os quais

devem ser corrigidos para os testes posteriores. Um segundo teste permite saber se as correções realizadas funcionaram corretamente e serve como garantia de qualidade do resultado do primeiro estudo, além disso, ajuda a fornecer informações mais específicas sobre a interface [118].

Seguida esta análise, foram iniciados os testes de usabilidade do projeto, onde foi dada uma guia de tarefas a cada um dos utilizadores (disponível no Anexo I) e pedido para falarem em voz alta de forma a entender quais as dificuldades que surgiram durante a execução das tarefas. Além disso, foi gravado o ecrã do computador, registando o número de erros e cliques realizados ao fazer as respetivas tarefas para, posteriormente, fazer uma análise dos resultados obtidos e determinar as falhas mais recorrentes.

Foi testado o projeto com utilizadores que possuem conhecimento bancário básico e intermédio e oferecida ajuda sobre o âmbito dos *widgets* nos casos necessários. Entre os 10 utilizadores que participaram nos testes, 3 pessoas possuem conhecimentos sobre contas bancárias empresariais (2 pessoas na primeira fase e 1 pessoa na segunda fase), isto permitiu obter *feedback* específico do público objetivo para o *widget* de operações pendentes.

Após a conclusão de todas as tarefas, os utilizadores exploraram livremente os *widgets* e foram apontados todos os comentários e sugestões que surgiram para complementar a análise dos resultados.

IV.6.1. Resultados dos Testes de Usabilidade – Fase 1

Na primeira fase de testes foram avaliados ambos *widgets* nas suas duas versões do projeto, *Desktop* e *Mobile*. A Tabela 13 e Tabela 14 apresentam as principais tarefas realizadas durante os testes e o número de cliques incorretos ocorridos (são considerados cliques incorretos aqueles que não são significativos para completar a tarefa pretendida).

Tabela 13 – Resultados dos testes de usabilidade da versão *Desktop* (Fase 1)

Widget (*Desktop*)	Tarefa	Nº1	Nº2	Nº3	Nº4	Nº5
Pagamento de serviços	Alterar a conta a debitar	0	0	0	0	0
	Estabelecer entidade favorita	0	0	0	0	0
	Escolher outro montante	0	0	0	0	0

Operações pendentes	Ir a minhas operações pendentes	0	0	0	0	0
	Ver detalhes da operação	0	0	0	0	0
	Ir a aprovar operação pendente	1	0	1	0	0

Tabela 14 – Resultados dos testes de usabilidade da versão *Mobile* (Fase 1)

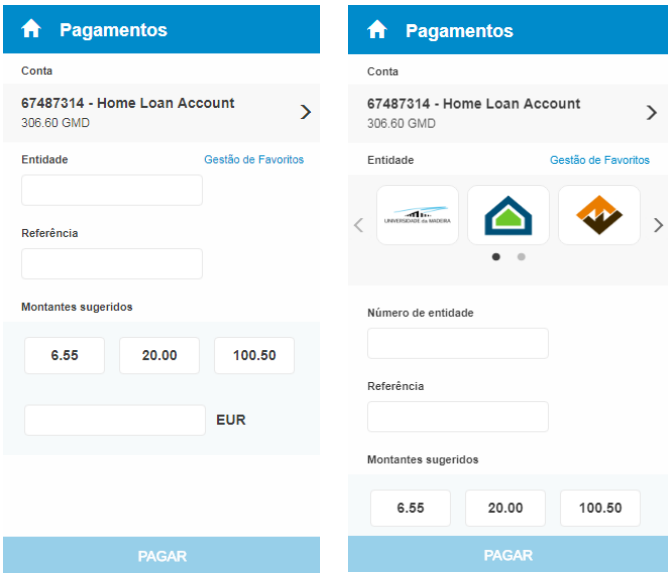
Widget (*Mobile*)	Tarefa	Nº1	Nº2	Nº3	Nº4	Nº5
Pagamento de serviços	Alterar a conta a debitar	1	0	0	0	0
	Estabelecer entidade favorita	0	0	0	0	0
	Escolher outro montante	0	0	0	0	0
Operações pendentes	Ir a minhas operações pendentes	1	0	1	4	0
	Ver detalhes da operação	0	0	0	0	0
	Aprovar operação pendente	3	5	6	1	1

Os resultados dos testes para a versão *Desktop* de ambos *widgets* foram considerados satisfatórios, pelo que se concluiu que não era necessária uma segunda iteração para esta versão. Entre o *feedback* obtido nesta fase foi recomendado adicionar o botão "fechar" na janela de "gestão de entidades favoritas" do *widget* de pagamento de serviços, esta alteração foi realizada.

No que diz respeito à versão *Mobile*, ocorreram vários cliques incorretos nas zonas da interface onde os utilizadores tinham que clicar em setas (por exemplo, para alterar a conta a debitar no pagamento). Pelo que foi modificada a zona clicável em todos os casos presentes, passando a ser clicável toda a área pretendida e não apenas a seta.

Por outro lado, na página de pagamentos de serviços para *Mobile* alguns utilizadores sugeriram que para o caso em que não existam entidades favoritas preestabelecidas deveria ser possível pagar qualquer entidade introduzida (e não unicamente as que estão disponíveis na lista predefinida do banco).

Tendo em consideração o *feedback*, foi resolvida esta limitação, adicionando um campo no formulário do pagamento que permite ao utilizador escrever o número da entidade pretendida, como ilustrado na Figura 42.



(a) (b)

Figura 42 – Nova versão do formulário da página de pagamento de serviços, (a) Sem entidades favoritas; (b) Com entidades favoritas.

Em relação ao *widget* de operações pendentes na versão *Mobile*, a dificuldade mais frequente encontrada nos testes foi na tarefa de aprovar a operação pendente (com data mais antiga), pois os utilizadores expressaram que nunca considerariam que a seta no menu permitiria aceder a essa opção, pelo que foi decidido juntamente com os *designers* da empresa remover a seta do menu e adicionar um *link* no conteúdo do *widget*. As alterações realizadas ilustram-se na Figura 43.

(a) (b)

Figura 43 – Ecrã principal do *widget* de operações pendentes na versão *Mobile*, (a) Versão anterior aos testes; (b) Nova versão após as alterações.

IV.6.2. Resultados dos Testes de Usabilidade – Fase 2

Após as devidas correções dos problemas encontrados na primeira fase, foi iniciada uma segunda fase de testes da versão *Mobile* para 5 utilizadores, onde os resultados expressaram uma melhoria significativa da interface, pois foi reduzido o número de cliques incorretos ao realizar as tarefas, como apresentado na Tabela 15.

Tabela 15 – Resultados dos testes de usabilidade da versão *Mobile* (Fase 2)

Widget (*Mobile*)	Tarefa	Nº1	Nº2	Nº3	Nº4	Nº5
Pagamento de serviços	Alterar conta a debitar	0	0	0	0	0
	Estabelecer entidade favorita	0	0	0	0	0
	Escolher outro montante	0	0	0	0	0
Operações pendentes	Ir a minhas operações pendentes	1	0	0	1	0
	Ver detalhes da operação	0	0	0	0	0
	Aprovar operação pendente	0	0	0	0	1

No que diz respeito à tarefa "*ir a minhas operações pendentes*" do *widget* de operações pendentes, alguns utilizadores clicaram no gráfico ao invés da opção disposta para o fim, pelo que foi proposto para uma versão futura o acesso à página de operações pendentes através do gráfico (para além da respetiva opção). Não foi considerado adicionar esta funcionalidade nesta versão pelas limitações que existem na alteração das características originais do gráfico utilizado. Finalmente, foi sugerida a visualização do "tipo de utilizador" no ecrã de aprovação, esta alteração pode ser vista na Figura 44.

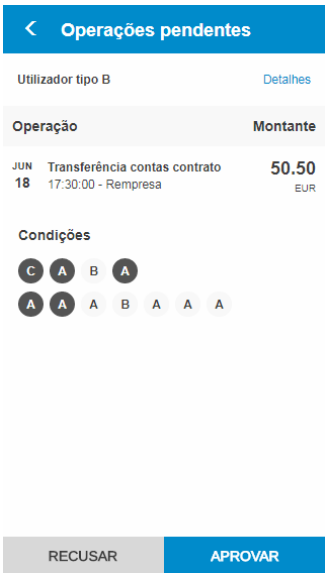


Figura 44 – Nova versão do ecrã secundário do *widget* de operações pendentes.

IV.7. DOCUMENTAÇÃO

Durante a implementação e manutenção do *software* um dos aspetos mais importantes a considerar é a documentação do código-fonte. Uma boa documentação favorece a comunicação entre os distintos membros da equipa de desenvolvimento e facilita a compreensão do código aos desenvolvedores e novos integrantes da equipa. Além disso, serve como referência para a manutenção do *software*.

Para este projeto, foi sugerido pela equipa de desenvolvimento da empresa o uso do gerador de documentação "**JSDoc**" [119] que permite criar documentação automática (em HTML) a partir de comentários adicionados no código JavaScript. JSDoc baseia-se numa série de *tags* (palavras precedidas de um símbolo "@") que são escritas antes de cada função, módulo ou classe e descrevem as diferentes características do código [120].

Seguir as convenções do JSDoc é uma boa maneira de garantir que os comentários sejam tão informativos e compreensíveis quanto possível [121]. Contudo, a incorporação de comentários tão detalhados em todo o código pode fazer com que os ficheiros fiquem sobrecarregados e o código seja difícil de ler e manipular. Por este motivo (e seguindo a mesma abordagem da equipa na empresa), foi utilizado no projeto o JSDoc unicamente para os módulos e funções nos que resulta indispensável ter uma descrição do seu comportamento ou retorno, maioritariamente para o código que pode ser reutilizado noutros módulos do sistema.

Decidiu-se gerar documentação automática com JSDoc para os módulos e funções do projeto relacionados com os pedidos feitos aos serviços *Web*, apresentando também exemplos do retorno de cada pedido na documentação, que pode ser consultada no Anexo J. Para o restante código não foi considerado indispensável descrever detalhadamente o seu comportamento, pelo que foi descrita brevemente a finalidade de cada módulo ou função através de comentários simples no código.

V. CONCLUSÃO

V.1. SOBRE A APLICAÇÃO

Nos últimos tempos, a automatização de tarefas do dia a dia através do uso de tecnologias modernas tem-se vindo a popularizar. Um exemplo claro disso é a aparição das aplicações *Web*, e o sistema *eBanka* encontra-se enquadrado neste contexto, permitindo executar através de Internet muitas das operações bancárias que eram realizadas no passado apenas dirigindo-se ao banco. O desenvolvimento de *widgets* para o *eBanka* representa uma solução para facilitar a interação com o sistema e permitem aceder de forma simplificada aos serviços que oferece.

O seguimento do paradigma de prototipagem foi uma ótima escolha para o desenvolvimento deste projeto, permitiu melhorar gradualmente o desenho dos *widgets* e detetar requisitos que não tinham sido considerados numa fase prévia de análise do projeto. Além disso, o *feedback* obtido durante os testes de usabilidade ajudou a melhorar as propostas do desenho e a chegar a uma solução final mais apropriada para os utilizadores.

Por outro lado, com a implementação de uma arquitetura baseada em componentes no projeto conseguiu-se incrementar a qualidade do *software*, favorecendo a reutilização do código desenvolvido e facilitando aspetos como a manutenção e testes do mesmo.

No que diz respeito aos recursos utilizados na fase de implementação, destaca-se a *framework* AngularJS, que apesar de ter sido necessário tempo e esforço para a aprendizagem inicial do uso desta tecnologia, foi observada a facilidade de desenvolver o *front-end* de aplicações *Web* utilizando esta *framework* e com um ótimo desempenho.

Para concluir, o principal objetivo do projeto, que seria o desenvolvimento de *widgets* para facilitar a interação do utilizador para dois dos serviços que oferece o sistema *eBanka* (pagamento de serviços e aprovação de operações pendentes) foi atingido, a empresa expressou a sua satisfação pelo projeto desenvolvido e é considerado um produto para ser integrado num futuro próximo no sistema *eBanka*. O

projeto encontra-se enquadrado com as características do *eBanka* e do projeto CAOP da empresa e os requisitos principais definidos (funcionais e não funcionais) foram implementados com sucesso.

V.2. SOBRE O ESTÁGIO CURRICULAR

O estágio curricular foi, sem dúvida, uma excelente experiência ao nível profissional e pessoal. Fazer parte do processo de desenvolvimento numa empresa bem-sucedida no mercado e conhecer profissionais da área foram dos mais gratos acontecimentos durante o meu percurso académico.

Ao longo do estágio tive a oportunidade de trabalhar e manter contacto com uma equipa de profissionais de várias áreas especializadas, que me transmitiram novos conhecimentos. Para o projeto, trabalhei com os *designers* da empresa que me deram o seu ponto de vista e ideias para o desenho dos protótipos e também com a equipa de *front-end* dos Canais Não Presenciais, que me orientaram na análise das características do sistema *eBanka* e no desenvolvimento do projeto.

Um dos aspetos da empresa que considerei interessante é o facto de os programadores realizarem formações sobre as tecnologias com que trabalham para transmitir os seus conhecimentos a todas as equipas. Tive a oportunidade de participar numa formação da linguagem JavaScript durante o estágio, o que não só me permitiu obter novos conhecimentos sobre a linguagem, mas também perceber os aspetos que eles consideram mais relevantes e presenciar a forma como são realizadas as formações na empresa.

Notei a dificuldade que existe em migrar os projetos desenvolvidos pela empresa para tecnologias mais modernas devido à complexidade do sistema. Contudo, as equipas estão sempre a par das novidades e dispostas a evoluir e melhorar a qualidade do *software*. Além disso, todas as pessoas partilham conhecimentos da área e, durante a minha experiência, mostraram-se dispostas a ajudar-me e esclarecer dúvidas.

No que diz respeito ao projeto, a etapa de prototipagem foi uma das mais destacadas de todo o processo. A empresa expressou, desde o início, o seu interesse pela criação de protótipos de baixa e alta fidelidade interativos e tive a iniciativa de

realizar testes de usabilidade com potenciais utilizadores, com o fim de obter *feedback* e assim melhorar as características dos *widgets*. Por outro lado, aprender tecnologias modernas para o desenvolvimento *front-end* de aplicações *Web* como a *framework* AngularJS foi uma mais valia para a minha experiência na área. Durante a implementação do projeto, a etapa que considero de maior evolução foi a de *refactoring*, pois permitiu-me otimizar o código desenvolvido e aplicar as sugestões dadas pela equipa da empresa, pelo que foi a etapa de maior aprendizagem técnica do estágio.

Por outro lado, considero o facto de este projeto estar orientado ao âmbito bancário como uma oportunidade para expandir os meus conhecimentos em outras áreas, o que é sempre útil do ponto de vista profissional e pessoal. As dificuldades encontradas por desconhecer aspetos sobre a banca foram superadas com o esclarecimento das dúvidas que foram surgindo durante o estágio.

Finalmente, a aproximação ao mundo do trabalho e a contextualização da experiência académica adquirida foi a minha maior motivação no momento de escolher um estágio curricular. Considerei sempre relevante ter um contacto com a realidade das empresas durante o percurso académico. Houve momentos de estresse e tensão, mas recomendaria a realização de um estágio curricular a qualquer colega que tenha a vontade de aprender, superar as suas inseguranças e conhecer a realidade do mundo profissional, pois esta é uma excelente oportunidade, tendo em consideração as experiências vividas e a evolução verificada durante o estágio, não só profissionalmente mas também ao nível pessoal.

V.3. PERSPETIVAS FUTURAS

Relativamente às perspetivas futuras do projeto, o passo a seguir é a integração do projeto ao sistema *eBanka* da empresa para, posteriormente, finalizar a implementação das funcionalidades que ficaram pendentes desta integração e passar a utilizar os serviços *Web* criados pela equipa de *Middleware*.

Outra situação que ficou pendente foi a atualização do módulo do *slideshow* do *widget* de pagamento de serviços para uma versão corrigida, pois este módulo

(desenvolvido por terceiros) se encontra atualmente com certas falhas que limitam o correto funcionamento do *widget*, falhas que já foram comunicadas aos desenvolvedores do módulo.

Por outro lado, uma das questões importantes para o futuro do projeto é a criação de testes automatizados, como os testes unitários, e também testar a versão *Mobile* em diferentes dispositivos móveis, uma vez que foram feitos testes apenas com o simulador que se encontra integrado nas ferramentas de desenvolvedores do *browser*.

Para concluir, sugiro algumas novas funcionalidades para serem implementadas a futuro, como a de apresentar montantes sugeridos com algum algoritmo inteligente no *widget* de pagamento de serviços (por exemplo, os montantes mais usados pelo utilizador nos pagamentos anteriores). Outra ideia sugerida é a de mostrar o número de operações pendentes da aprovação do utilizador como uma notificação no *dashboard* da futura versão para dispositivos móveis do sistema *eBanka*.

REFERÊNCIAS

[1] "Exictos." [Online]. Available: https://www.exictos.com/home/about-exictos/asseco/asseco-group. [Accessed: 16-Jan-2017].

[2] J. Hernández, "Los microservicios no son para mí," 09-May-2016. [Online]. Available: http://blog.koalite.com/2016/05/los-microservicios-no-son-para-mi/. [Accessed: 02-Apr-2017].

[3] J. Lewis and M. Fowler, "Microservices," *martinfowler.com*. [Online]. Available: https://martinfowler.com/articles/microservices.html. [Accessed: 02-Apr-2017].

[4] L. Nóbrega, "Processos de Desenvolvimento de Software," Universidade da Madeira, 2013.

[5] R. Pressman, *Ingeniería del Software: Un Enfoque Práctico*, 7th ed. New York, NY: McGraw-Hill, 2010.

[6] B. Yolanda, "Metodología Ágil de Desarrollo de Software – XP," Universidad de las Fuerzas Armadas, Sangolquí, Ecuador.

[7] R. Sicarelli, "Testes unitários vs aceitação," 14-May-2016. [Online]. Available: https://medium.com/android-dev-br/testes-unit%C3%A1rios-vs-aceita%C3%A7%C3%A3o-30691fc8578d. [Accessed: 14-Jan-2017].

[8] M. Winckler and M. Pimenta, "Avaliação de Usabilidade de Sites Web." [Online]. Available: https://www.irit.fr/~Marco.Winckler/2002-winckler-pimenta-ERI-2002-cap3.pdf. [Accessed: 28-Feb-2017].

[9] "Arquitectura cliente servidor," 17-Oct-2016. [Online]. Available: https://oposicionestic.blogspot.pt/2011/06/arquitectura-cliente-servidor.html. [Accessed: 01-Mar-2017].

[10] "¿En qué se diferencia el desarrollo de front-end del de back-end?," *SearchDataCenter en Español*. [Online]. Available: http://searchdatacenter.techtarget.com/es/respuesta/En-que-se-diferencia-el-desarrollo-de-front-end-del-de-back-end. [Accessed: 28-Feb-2017].

[11] "Frontend y Backend ¿Que son?" [Online]. Available: http://culturacion.com/frontend-y-backend-%C2%BFque-son/. [Accessed: 28-Feb-2017].

[12] A. Pedraza, "¿Qué es desarrollo frontend?," 01-Sep-2014. [Online]. Available: https://desarrollofrontend.com/que-es-desarrollo-frontend/. [Accessed: 28-Feb-2017].

[13] “¿Qué es y para qué sirve un web service?” [Online]. Available: http://culturacion.com/que-es-y-para-que-sirve-un-web-service/. [Accessed: 10-Mar-2017].

[14] G. Duarte, “Arquitectura para diseñar e implementar Web Services,” vol. 3, no. 2, 2015.

[15] M. Massé, *REST API Design Rulebook*. Sebastopol, CA: O’Reilly, 2012.

[16] “Qué diferencia hay entre API y Servicio Web,” 16-Feb-2017. [Online]. Available: http://quediferenciahay.com/que-diferencia-hay-entre-api-y-servicio-web/. [Accessed: 10-Mar-2017].

[17] A. Fernández, “Servicios web RESTful con HTTP. Parte I: Introducción y bases téoricas,” 12-Nov-2013. [Online]. Available: http://www.adwe.es/general/colaboraciones/servicios-web-restful-con-http-parte-i-introduccion-y-bases-teoricas#ref. [Accessed: 10-Mar-2017].

[18] R. Navarro, “Modelado, Diseño e Implementación de Servicios Web,” Universidad Politécnica de Valencia, Valência, Espanha, 2007.

[19] A. Marqués, “Conceptos sobre APIs REST,” 11-Apr-2013. [Online]. Available: http://asiermarques.com/2013/conceptos-sobre-apis-rest/. [Accessed: 11-Mar-2017].

[20] “API REST: qué es y cuáles son sus ventajas en el desarrollo de proyectos,” 23-Mar-2016. [Online]. Available: https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos. [Accessed: 11-Mar-2017].

[21] “¿Que es un proxy? ¿Para que sirven?,” 11-Mar-2013. [Online]. Available: https://www.taringa.net/posts/ciencia-educacion/16487890/Que-es-un-proxy-Para-que-sirven.html. [Accessed: 10-Mar-2017].

[22] “gateway no Dicionário Infopédia da Língua Portuguesa com Acordo Ortográfico,” *Infopédia - Dicionários Porto Editora*. [Online]. Available: https://www.infopedia.pt/dicionarios/lingua-portuguesa/gateway. [Accessed: 12-Mar-2017].

[23] J. Alarcón, “5 conceptos que todo programador JavaScript debería conocer,” 08-Apr-2015. [Online]. Available: https://www.campusmvp.es/recursos/post/5-conceptos-que-todo-programador-JavaScript-deberia-conocer.aspx. [Accessed: 11-Mar-2017].

[24] “Porqué uso arquitectura REST.” [Online]. Available: https://byteflair.com/es/2014/06/porque-uso-arquitectura-rest/. [Accessed: 12-Mar-2017].

[25] T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” University of California, Irvine, 2000.

[26] “RESTful Services Quick Tips.” [Online]. Available: http://www.restapitutorial.com/lessons/restquicktips.html. [Accessed: 11-Mar-2017].

[27] “HTTP Methods for RESTful Services.” [Online]. Available: http://www.restapitutorial.com/lessons/httpmethods.html. [Accessed: 11-Mar-2017].

[28] “Principios de un webservice RESTful,” 24-May-2014. [Online]. Available: https://kubide.es/principios-de-un-webservice-restful/. [Accessed: 11-Mar-2017].

[29] “Entendiendo el patrón MVC,” 02-Nov-2014. [Online]. Available: https://frisconet.com.ar/entendiendo-el-patron-mvc/. [Accessed: 12-Apr-2017].

[30] K. McArthur, *Pro PHP: Patterns, Frameworks, Testing and More*. Berkeley, CA: Apress, 2008.

[31] S. Holzner, *Design Patterns For Dummies*. Hoboken, NJ: Wiley, 2006.

[32] “Fundamentals of an MVC Framework.” [Online]. Available: http://www.programminghelp.com/mvc/fundamentals-mvc-framework/. [Accessed: 12-Apr-2017].

[33] “Arquitectura basado en componentes.” [Online]. Available: http://www.w3ii.com/es/software_architecture_design/component_based_architecture.html. [Accessed: 12-Apr-2017].

[34] D. Abramov, “Presentational and Container Components,” 23-Mar-2015. [Online]. Available: https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0. [Accessed: 12-Apr-2017].

[35] D. Grossman, “Writing component based app with angular 1.5,” 20-Jul-2016. [Online]. Available: http://blog.grossman.io/angular-1-component-based-architecture-2/. [Accessed: 12-Apr-2017].

[36] “Online Mockup, Wireframe & UI Prototyping Tool · Moqups.” [Online]. Available: https://moqups.com/. [Accessed: 05-Jan-2017].

[37] “MockFlow - Online Wireframe Tools, Prototyping Tools, UI Mockups, UX Suite.” [Online]. Available: https://www.mockflow.com/. [Accessed: 05-Jan-2017].

[38] “Prototypes, Specifications, and Diagrams in One Tool | Axure Software.” [Online]. Available: https://www.axure.com/. [Accessed: 05-Jan-2017].

[39] “Buy | Axure.” [Online]. Available: https://www.axure.com/buy. [Accessed: 05-Jan-2017].

[40] “InVision | Digital Product Design, Workflow & Collaboration.” [Online]. Available: https://www.invisionapp.com/. [Accessed: 05-Jan-2017].

[41] “Top 3 herramientas de prototipado para webs y aplicaciones,” 25-Sep-2015. [Online]. Available: https://www.lancetalent.com/blog/top-3-herramientas-de-prototipado-web-aplicaciones/. [Accessed: 05-Jan-2017].

[42] “Framer - Interactive Design & Collaboration Tool.” [Online]. Available: https://framer.com/. [Accessed: 05-Jan-2017].

[43] E. Schwartzman, “Designer’s Toolkit: Prototyping Tools,” 14-Jul-2015. [Online]. Available: https://www.cooper.com/prototyping-tools/Framer. [Accessed: 05-Jan-2017].

[44] T. Mat, “Five app prototyping tools compared: Proto.io, Pixate, Origami, Framer & Form,” 06-Aug-2015. [Online]. Available: https://medium.com/sketch-app-sources/five-app-prototyping-tools-compared-form-framer-origami-pixate-proto-io-c2acc9062c61. [Accessed: 05-Jan-2017].

[45] A. Basalo and M. Alvarez, “Por qué AngularJS,” 22-Aug-2014. [Online]. Available: https://desarrolloweb.com/articulos/por-que-angularjs.html. [Accessed: 26-Apr-2017].

[46] J. Gutiérrez, “¿Qué es un framework web?” [Online]. Available: http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf. [Accessed: 26-Apr-2017].

[47] “The Best of JavaScript and the web platform.” [Online]. Available: https://bestof.js.org/tags/framework/. [Accessed: 27-Apr-2017].

[48] “Web framework rankings.” [Online]. Available: https://hotframeworks.com/languages/javascript. [Accessed: 25-Apr-2017].

[49] “AngularJS — Superheroic JavaScript MVW Framework.” [Online]. Available: https://angularjs.org/. [Accessed: 24-Apr-2017].

[50] “Los frameworks imprescindibles para desarrollar la web del futuro,” 22-Dec-2015. [Online]. Available: https://bbvaopen4u.com/es/actualidad/los-frameworks-imprescindibles-para-desarrollar-la-web-del-futuro. [Accessed: 23-Apr-2017].

[51] “Angular.” [Online]. Available: https://angular.io/. [Accessed: 25-Apr-2017].

[52] A. Basalo, “Introducción a Angular 2,” 09-Jun-2016. [Online]. Available: https://desarrolloweb.com/articulos/introduccion-angular2.html. [Accessed: 27-Apr-2017].

[53] “React - A JavaScript library for building user interfaces.” [Online]. Available: https://reactjs.org/index.html. [Accessed: 26-Apr-2017].

[54] W. Turriate, “React js en español - tutorial básico, primeros pasos y ejemplos,” 02-Feb-2016. [Online]. Available: https://frontendlabs.io/3158--react-js-espanol-tutorial-basico-primeros-pasos-ejemplos. [Accessed: 26-Apr-2017].

[55] D. Suarez, “Introducción a React,” 16-Feb-2015. [Online]. Available: https://www.adictosaltrabajo.com/tutoriales/introduccion-react/. [Accessed: 10-Aug-2017].

[56] “¿Qué es React.js? | React.” [Online]. Available: https://burgersbacon.gitbooks.io/react/content/chapter1.html. [Accessed: 26-Apr-2017].

[57] J. Robie, “¿Qué es el Modelo de Objetos del Documento?” [Online]. Available: http://html.conclase.net/w3c/dom1-es/introduction.html. [Accessed: 12-Aug-2017].

[58] “Vue.js.” [Online]. Available: https://vuejs.org/. [Accessed: 28-Apr-2017].

[59] “Introduction — Vue.js.” [Online]. Available: https://vuejs.org/v2/guide/#Getting-Started. [Accessed: 28-Apr-2017].

[60] “5 Best JavaScript Frameworks in 2017,” 18-Jan-2017. [Online]. Available: https://da-14.com/blog/5-best-javascript-frameworks-2017. [Accessed: 10-Aug-2017].

[61] “jQuery.” [Online]. Available: https://jquery.com/. [Accessed: 28-Apr-2017].

[62] “2016 JavaScript Rising Stars.” [Online]. Available: https://risingstars2016.js.org/#framework. [Accessed: 23-Apr-2017].

[63] “Git - Acerca del control de versiones.” [Online]. Available: https://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones. [Accessed: 04-May-2017].

[64] I. Alcázar and M. Alvarez, “Introducción a Git y Github,” 03-Jun-2014. [Online]. Available: https://desarrolloweb.com/articulos/introduccion-git-github.html. [Accessed: 05-May-2017].

[65] “Apache Subversion.” [Online]. Available: https://subversion.apache.org/. [Accessed: 06-May-2017].

[66] “Por qué Git es el sistema de control de versiones más popular,” 12-May-2014. [Online]. Available: https://hipertextual.com/archivo/2014/05/git-sistema-control-versiones/. [Accessed: 06-May-2017].

[67] “Git.” [Online]. Available: https://git-scm.com/. [Accessed: 05-May-2017].

[68] “Mercurial SCM.” [Online]. Available: https://www.mercurial-scm.org/. [Accessed: 06-May-2017].

[69] A. Dias, “Qual a melhor ferramenta de controle de versão: Subversion, Git ou Mercurial?,” 17-Jun-2009. [Online]. Available: https://blog.pronus.io/posts/qual-a-melhor-ferramenta-de-controle-de-versao-subversion-git-ou-mercurial/. [Accessed: 07-May-2017].

[70] “The world’s leading software development platform · GitHub.” [Online]. Available: https://github.com/. [Accessed: 06-May-2017].

[71] “SourceForge vs GitHub vs Bitbucket vs GitLab,” 05-May-2016. [Online]. Available: https://openwebinars.net/blog/sourceforge-vs-github-vs-bitbucket-vs-gitlab/. [Accessed: 07-May-2017].

[72] N. González, “Github vs Bitbucket. Mi opinión personal sobre cual usar.” [Online]. Available: http://www.nazariglez.com/2013/02/15/github-vs-bitbucket-mi-opinion-personal-sobre-cual-usar/. [Accessed: 07-May-2017].

[73] “Bitbucket | The Git solution for professional teams.” [Online]. Available: https://bitbucket.org/product. [Accessed: 07-May-2017].

[74] “The leading product for integrated software development - GitLab | GitLab.” [Online]. Available: https://about.gitlab.com/. [Accessed: 07-May-2017].

[75] P. Gutiérrez, “GitLab, la alternativa a GitHub, ligera y con repositorios privados,” 26-May-2014. [Online]. Available: https://www.genbetadev.com/sistemas-de-control-de-versiones/gitlab-la-alternativa-a-github-ligera-y-con-repositorios-privados. [Accessed: 05-May-2017].

[76] C. Solis, *Manual del Guerrero - AngularJS*. 2015.

[77] “AngularJS: Developer Guide: Introduction.” [Online]. Available: https://docs.angularjs.org/guide/introduction. [Accessed: 18-May-2017].

[78] “Comparison with Other Frameworks — Vue.js.” [Online]. Available: https://vuejs.org/v2/guide/comparison.html. [Accessed: 18-May-2017].

[79] “Angular-nvD3.” [Online]. Available: https://krispo.github.io/angular-nvd3/#/dashboard. [Accessed: 18-May-2017].

[80] G. Inc, *Google Analytics*. Google Inc., 2017.

[81] “App Santander Totta.” [Online]. Available: https://www.santandertotta.pt/pt_PT/Particulares/Servi%C3%A7os/App-Santander-Totta.html. [Accessed: 10-Jan-2017].

[82] B. P. Portugal, *Banco Popular*. Banco Popular Portugal, 2017.

[83] “Canal Popular OnMobile Particulares | Popular.” [Online]. Available: http://www.bancopopular.pt/particulares/canais/mobile. [Accessed: 10-Jan-2017].

[84] “App BPI Empresas | Empresas | Banco BPI.” [Online]. Available: http://www.bancobpi.pt/empresas/servicos-24-7/app-bpi-empresas. [Accessed: 10-Jan-2017].

[85] “Trello.” [Online]. Available: https://trello.com/. [Accessed: 19-Aug-2017].

[86] K. Pohl and C. Rupp, *Fundamentos da Engenharia de Requisitos*. Rocky Nook, 2012.

[87] “¿Cómo Priorizar requisitos? Técnica de priorización MOSCOW.” [Online]. Available: http://managementplaza.es/blog/priorizar-requisitos-tecnica-priorizacion-moscow/. [Accessed: 19-Jan-2017].

[88] S. Molina, “Metodologías Ágiles Enfocadas al Modelado de Requerimientos,” Universidad Nacional de la Patagonia Austral, Santa Cruz, Argentina, 2012.

[89] O. Montoto, “Wireframes,” 21-Jan-2011. [Online]. Available: https://olgacarreras.blogspot.pt/2007/02/wireframes.html. [Accessed: 19-Jan-2017].

[90] “GitHub - typicode/json-server: Get a full fake REST API with zero coding in less than 30 seconds (seriously).” [Online]. Available: https://github.com/typicode/json-server. [Accessed: 14-Jul-2017].

[91] M. Alvarez, “Componentes en AngularJS 1.5,” 22-Feb-2016. [Online]. Available: https://desarrolloweb.com/articulos/componentes-angularjs-15.html. [Accessed: 02-Jul-2017].

[92] A. Basalo and M. Alvarez, “Qué es AngularJS,” 28-Aug-2014. [Online]. Available: https://desarrolloweb.com/articulos/que-es-angularjs-descripcion-framework-javascript-conceptos.html. [Accessed: 09-Jul-2017].

[93] M. Dorta, *AngularJS Paso a Paso*, 2nd ed. Victoria, British Columbia, Canada: Lean Publishing, 2016.

[94] “MVC y MVVM,” 07-Oct-2012. [Online]. Available: https://www.adictosaltrabajo.com/tutoriales/zk-mvc-mvvm/#comparacion-entre-mvc-mvvm. [Accessed: 10-Jul-2017].

[95] M. Alvarez, “Qué es ES6 y cómo estudiarlo,” 16-Aug-2016. [Online]. Available: https://desarrolloweb.com/articulos/que-es-ecmascript6-como-estudiarlo.html. [Accessed: 15-Jul-2017].

[96] “Babel · The compiler for writing next generation JavaScript.” [Online]. Available: https://babeljs.io/. [Accessed: 15-Jul-2017].

[97] M. Flores, “Las principales características de Angularjs.” [Online]. Available: http://html5facil.com/tutoriales/las-principales-caracteristicas-de-angularjs/. [Accessed: 09-Jul-2017].

[98] “AngularJS: Developer Guide: Data Binding.” [Online]. Available: https://docs.angularjs.org/guide/databinding. [Accessed: 10-Jul-2017].

[99] P. Hurtado and X. Cerdá, “Módulo, objeto module en AngularJS,” 29-Sep-2014. [Online]. Available: https://desarrolloweb.com/articulos/modulo-objeto-module-angularjs.html. [Accessed: 10-Jul-2017].

[100] “Node.js.” [Online]. Available: https://nodejs.org/en/. [Accessed: 15-Jul-2017].

[101] C. Lopes, “O que é Node.js e saiba os primeiros passos,” 28-Jul-2014. [Online]. Available: https://tableless.com.br/o-que-nodejs-primeiros-passos-com-node-js/. [Accessed: 15-Jul-2017].

[102] “npm.” [Online]. Available: https://www.npmjs.com/. [Accessed: 15-Jul-2017].

[103] F. Monteiro, *Learning Single-page Web Application Development*. Birmingham, UK: Packt Publishing, 2014.

[104] “GitHub - likeastore/ngDialog: Modals and popups provider for Angular.js applications.” [Online]. Available: https://github.com/likeastore/ngDialog. [Accessed: 16-Jul-2017].

[105] “angular-ui-carousel.” [Online]. Available: https://www.npmjs.com/package/angular-ui-carousel. [Accessed: 16-Jul-2017].

[106] “Interactive JavaScript charts for your webpage | Highcharts.” [Online]. Available: https://www.highcharts.com/. [Accessed: 16-Jul-2017].

[107] H. Alvaro, “HighCharts: Librería para creación de gráficos,” 01-Mar-2013. [Online]. Available: https://enboliviacom.wordpress.com/2013/03/01/highcharts-libreria-para-creacion-de-graficos/. [Accessed: 16-Jul-2017].

[108] A. Rios, “¿Por qué deberías usar un preprocesador de CSS?,” *www.webbizarro.com*, 22-Jan-2014. [Online]. Available: https://www.webbizarro.com/noticias/485/por-que-deberias-usar-un-preprocesador-de-css/. [Accessed: 20-Jul-2017].

[109] “Sass: Syntactically Awesome Style Sheets.” [Online]. Available: http://sass-lang.com/. [Accessed: 20-Jul-2017].

[110] “Sass Documentation,” 13-Jul-2017. [Online]. Available: http://sass-lang.com/documentation/file.SASS_REFERENCE.html. [Accessed: 20-Jul-2017].

[111] “Bootstrap · The most popular HTML, CSS, and JS library in the world.” [Online]. Available: http://getbootstrap.com/. [Accessed: 16-Jul-2017].

[112] “SourceTree | Free Git GUI for Mac and Windows.” [Online]. Available: https://www.sourcetreeapp.com/. [Accessed: 16-Jul-2017].

[113] “Introdução sobre Media Queries.” [Online]. Available: /introducao-sobre-media-queries/. [Accessed: 20-Jul-2017].

[114] “Procesos de Mantenibilidad: Refactoring,” 24-Feb-2016. [Online]. Available: https://algodeingenieriablog.wordpress.com/2016/02/24/procesos-de-mantenibilidad-refactoring/. [Accessed: 19-Aug-2017].

[115] “ESLint - Pluggable JavaScript linter.” [Online]. Available: https://eslint.org/. [Accessed: 19-Aug-2017].

[116] “Style Guides | Airbnb Engineering.” [Online]. Available: http://airbnb.io/projects/styleguides/. [Accessed: 19-Aug-2017].

[117] “GitHub - airbnb/javascript: JavaScript Style Guide.” [Online]. Available: https://github.com/airbnb/javascript. [Accessed: 19-Aug-2017].

[118] J. Nielsen, “Why You Only Need to Test with 5 Users,” 19-Mar-2000. [Online]. Available: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/. [Accessed: 01-Aug-2017].

[119] “Use JSDoc.” [Online]. Available: http://usejsdoc.org/. [Accessed: 20-Aug-2017].

[120] M. Iglesias, “Commenting JavaScript code with JSDoc,” 22-Jan-2016. [Online]. Available: https://www.eventbrite.com/engineering/commenting-javascript-code-with-jsdoc/. [Accessed: 20-Aug-2017].

[121] “Front-end documentation with JSDoc,” 2015. [Online]. Available: https://caktus.github.io/developer-documentation/frontend-docs.html. [Accessed: 20-Aug-2017].

ANEXOS

ANEXO A. ARQUITETURA DO PROJETO CAOP

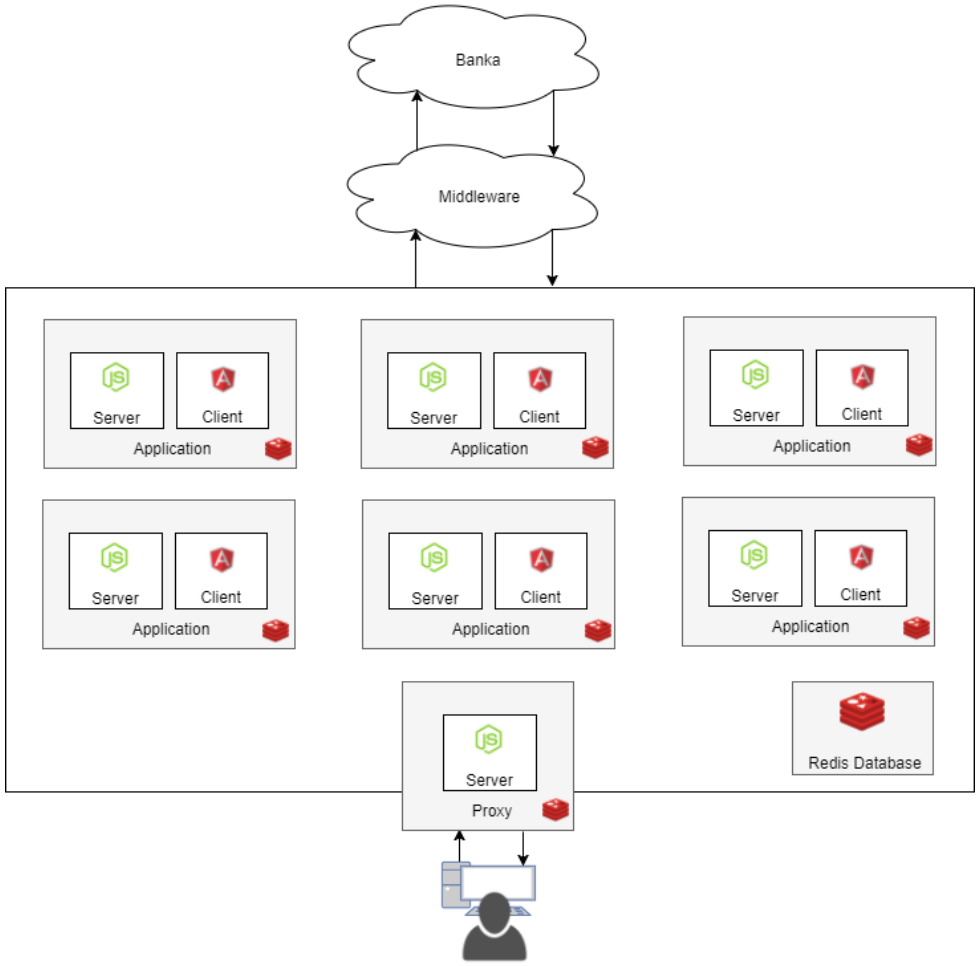


Figura 45 – Arquitetura do projeto CAOP.

ANEXO B. *SKETCHES*

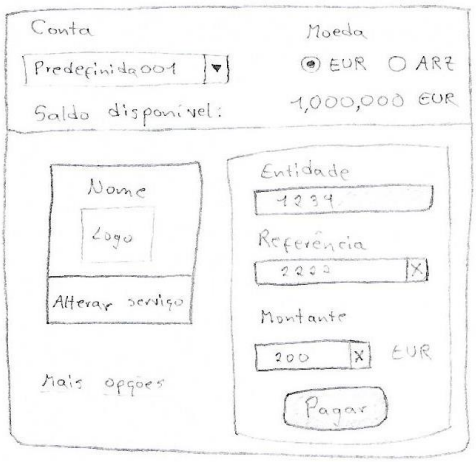


Figura 46 – Primeira versão dos *sketches* do *widget* de Pagamento de Serviços.

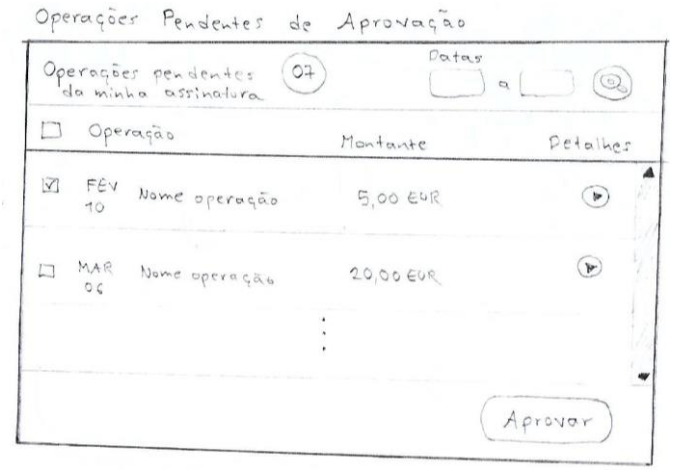
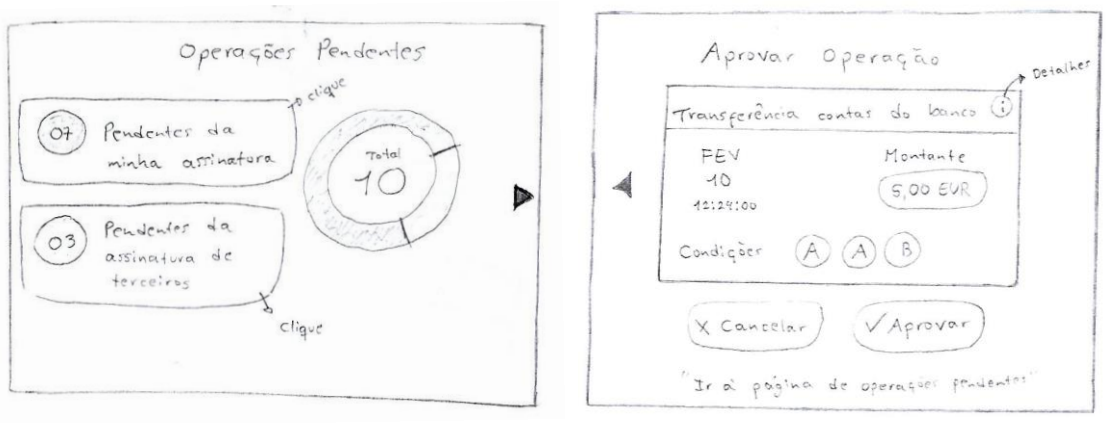


Figura 47 – Primeira versão dos *sketches* do *widget* de Operações Pendentes.



(a) **(b)**

Figura 48 – Nova versão dos *sketches* do *widget* de Operações Pendentes após os testes de usabilidade realizados aos *wireframes*, (a) Ecrã principal; (b) Ecrã de aprovação.

ANEXO C. *WIREFRAMES*

Versão Inicial dos *Wireframes*

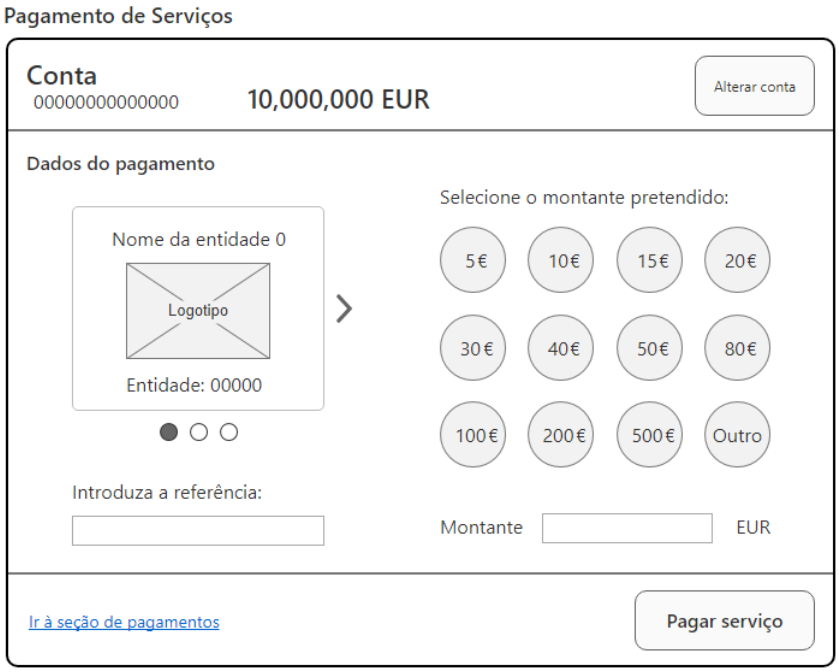
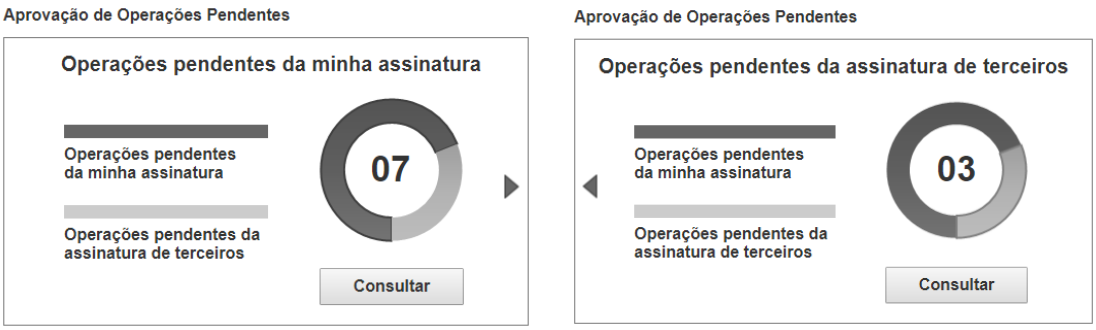


Figura 49 – Primeira versão dos *wireframes* do *widget* de Pagamento de Serviços.

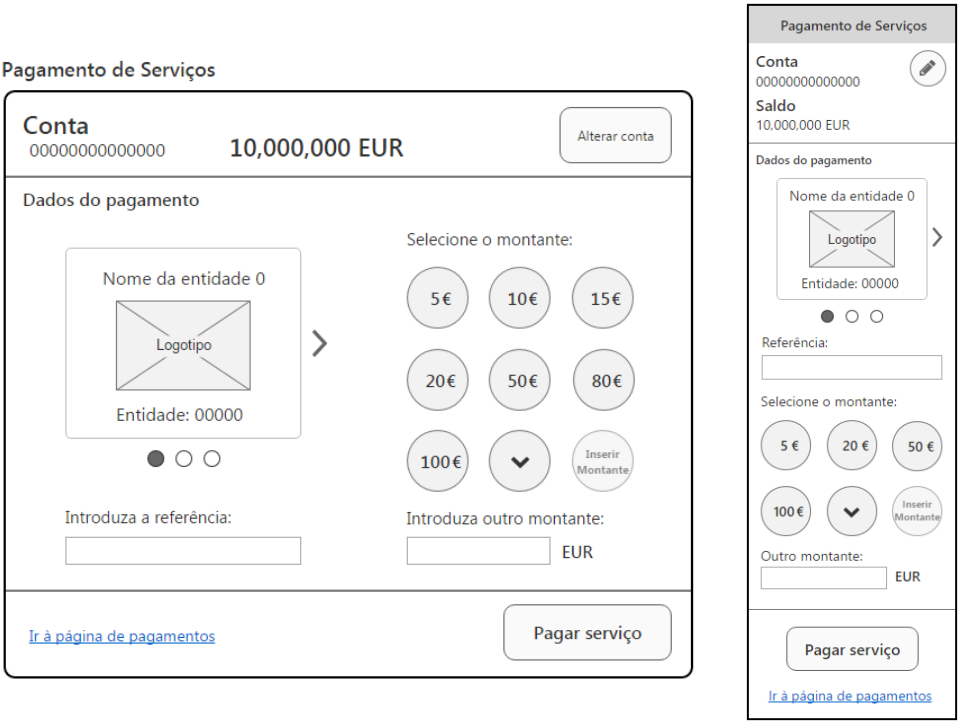


(a) **(b)**

Figura 50 – Primeira versão dos *wireframes* do *widget* de Operações Pendentes, (a) Ecrã principal; (b) Ecrã secundário.

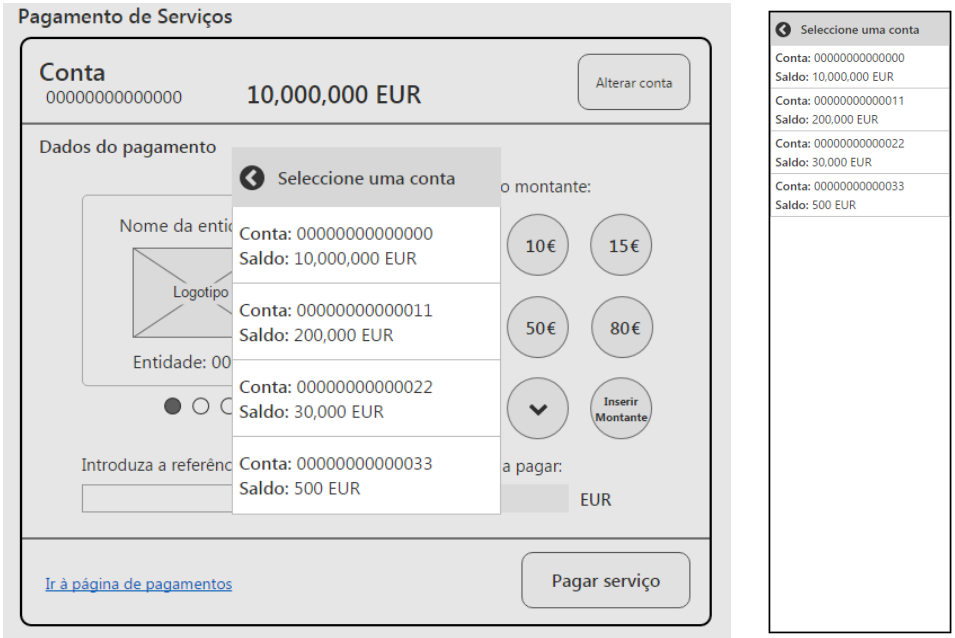
Versão Final dos *Wireframes* do *Widget* de Pagamento de Serviços

Figura 51 – *Wireframes* do ecrã principal do *widget* de pagamento de serviços com a lista de montantes predefinidos.



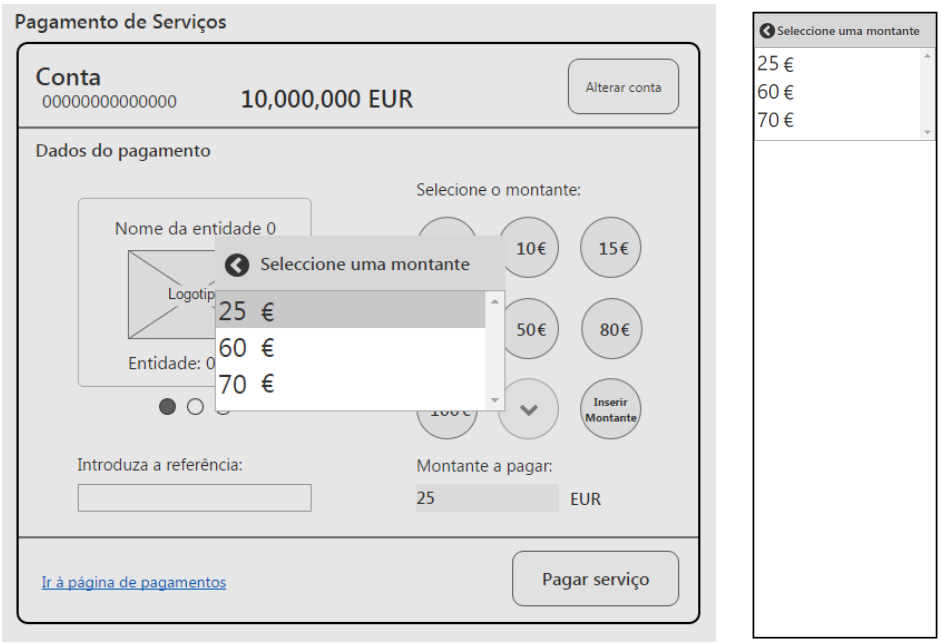
(a) **(b)**

Figura 52 – *Wireframes* do ecrã principal do *widget* de pagamento de serviços com a opção de inserir um montante, (a) Versão *Desktop*; (b) Versão *Mobile*.



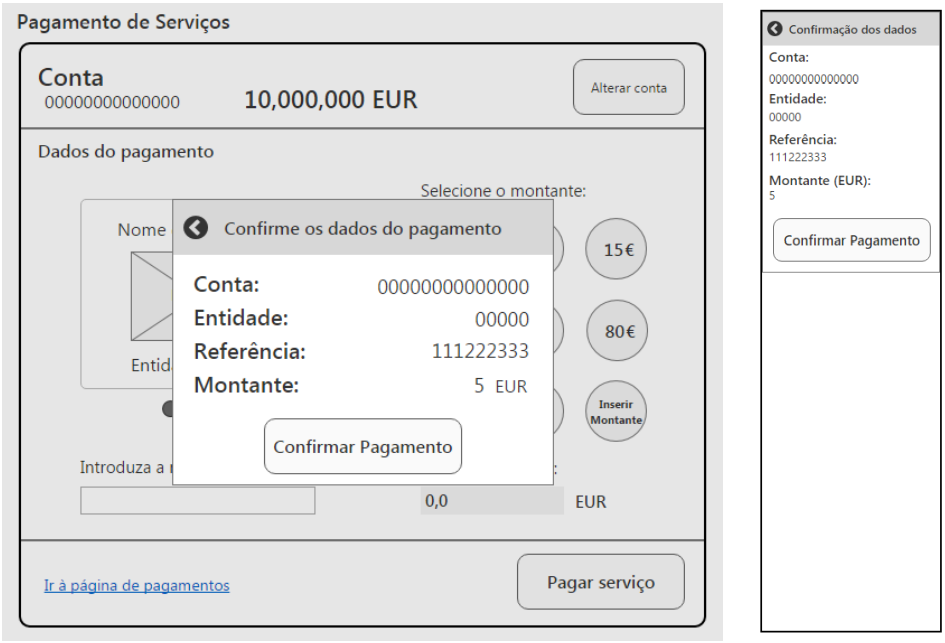
(a) **(b)**

Figura 53 – *Wireframes* da listagem das contas do utilizador, (a) Versão *Desktop*; (b) Versão *Mobile*.



(a) **(b)**

Figura 54 – *Wireframes* da listagem dos montantes predefinidos para a respetiva entidade escolhida, (a) Versão *Desktop*; (b) Versão *Mobile*.



(a) **(b)**

Figura 55 – *Wireframes* da janela de resumo da operação, (a) Versão *Desktop*; (b) Versão *Mobile*.

Figura 56 – *Wireframes* da janela da mensagem com o estado da operação, (a) Versão *Desktop*; (b) Versão *Mobile*.

ANEXO D. PROTÓTIPOS DE ALTA FIDELIDADE

Na Figura 57 ilustra-se o protótipo do *dashboard* do *eBanka* com os *widgets* a serem desenvolvidos neste projeto. Os ecrãs dos *widgets* apresentados são para os casos em que o utilizador não possui entidades favoritas estabelecidas nem operações pendentes pela sua aprovação.

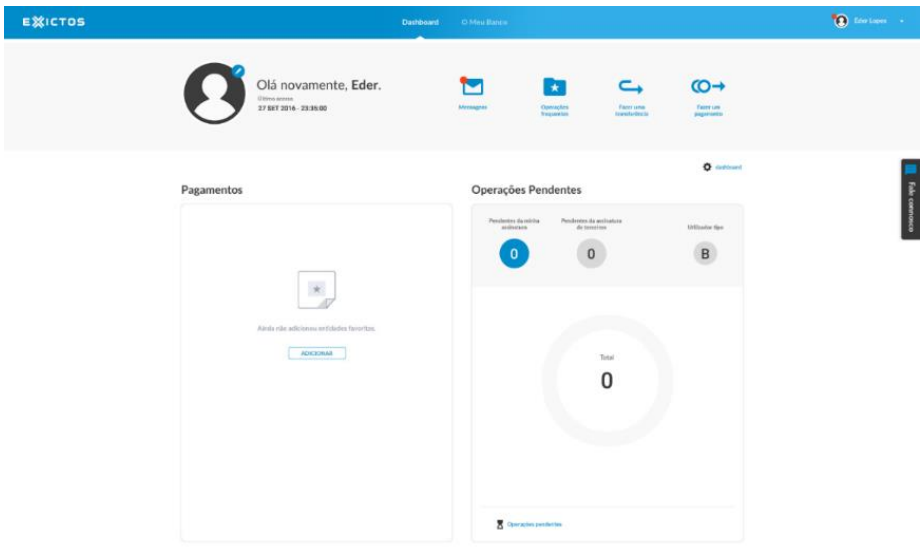
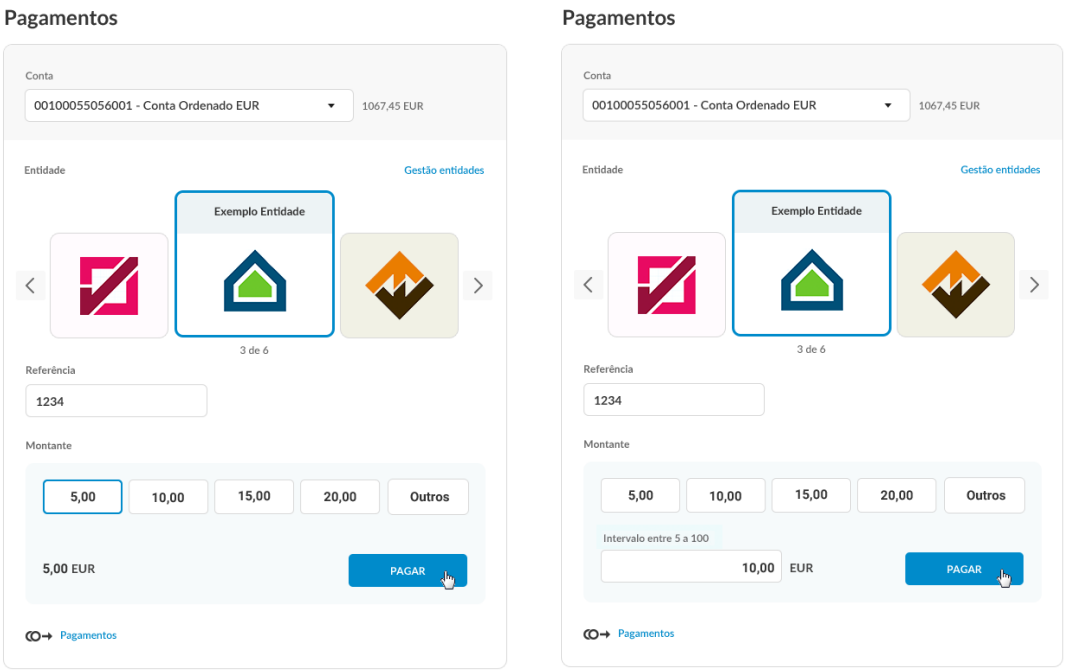


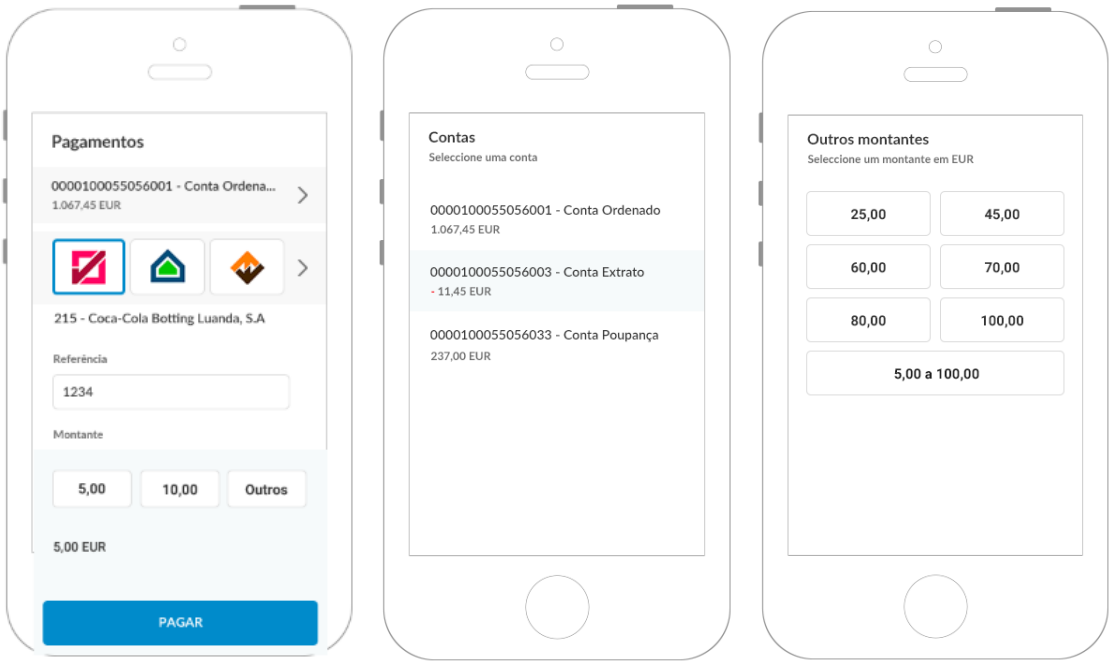
Figura 57 - PAF do *Dashboard* do *eBanka*.



(a) **(b)**

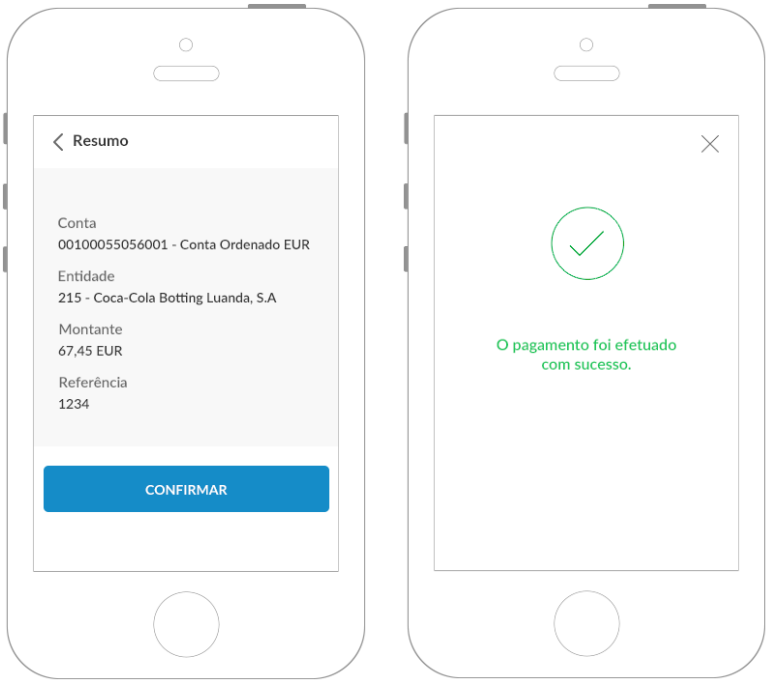
Figura 58 – PAF do ecrã principal do *widget* de Pagamento de Serviços da versão *Desktop*, (a) Listagem de montantes predefinidos; (b) Montante dentro de um intervalo.

Figura 59 – PAF da janela com a listagem de todos os montantes predefinidos da versão *Desktop* do *widget* de Pagamento de Serviços.



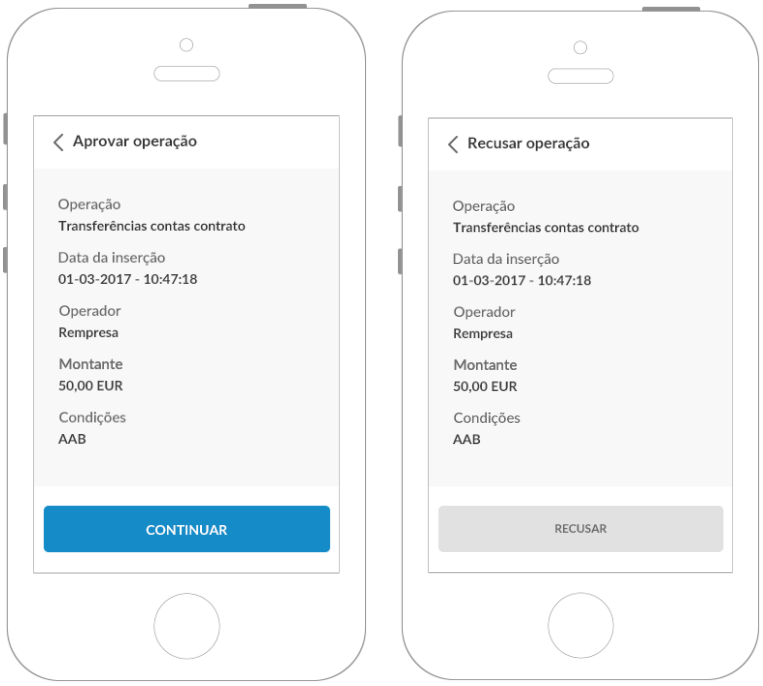
(a) **(b)** **(c)**

Figura 60 – PAF do *widget* de Pagamento de Serviços da versão *Mobile*, (a) Ecrã principal com montantes predefinidos; (b) Listagem das contas do utilizador; (c) Listagem de todos os montantes predefinidos.



(a) (b)

Figura 61 – PAF do *widget* de Pagamento de Serviços da versão *Mobile*, (a) Ecrã do resumo da operação; (b) Ecrã da mensagem com o estado da operação.



(a) (b)

Figura 62 – PAF do *widget* de Operações Pendentes da versão *Mobile*, (a) Ecrã do resumo da aprovação da operação pendente; (b) Ecrã do resumo da rejeição da operação pendente.

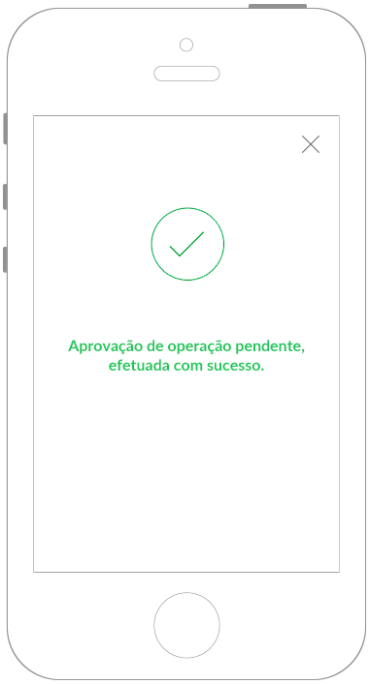


Figura 63 – PAF do ecrã da mensagem com o estado da operação do *widget* de Operações Pendentes, versão *Mobile*.

ANEXO E. ESTRUTURAS DE DADOS

Estruturas de dados dos serviços *Web* para o *widget* de Pagamento de Serviços:

- **Contas bancárias do utilizador:**

```
{
   accounts: [
      {
         account_id: (Number),
         account_number: (Number),
         account_name: (String),
         account_balance: {
            value: (Number),
            currency: (String)
         }
      }
   ]
}
```

- **Entidades:**

```
{
   entities: [
      {
         entity_id: (Number)
         entity_name: (String),
         type_amounts: (String),
         predefined_amounts: [Number],
         default_image: (String),
         isFavourite: (Boolean)
      }
   ]
}
```

Estruturas de dados dos serviços *Web* para o *widget* de Operações Pendentes:

- **Dados do atual utilizador:**

```
{
   user: [
      {
         user_id: (Number),
         user_name: (String),
         user_type: (String),
      }
   ]
}
```

- **Operações pendentes:**

```
{
   operations: [
      {
         operation_id: (Number),
         name_operation: (String),
         name_operator: (String),
         amount_operation: {
            value: (Number),
            currency: (String)
         },
         date: (String),
         state: (String),
         conditions_approved: [String],
         conditions_operation: [condition_operation],
         condition_operation: [condition],
         condition: {
            value: (String)
         }
      }
   ]
}
```

ANEXO F. COMPONENTES DOS *WIDGETS*

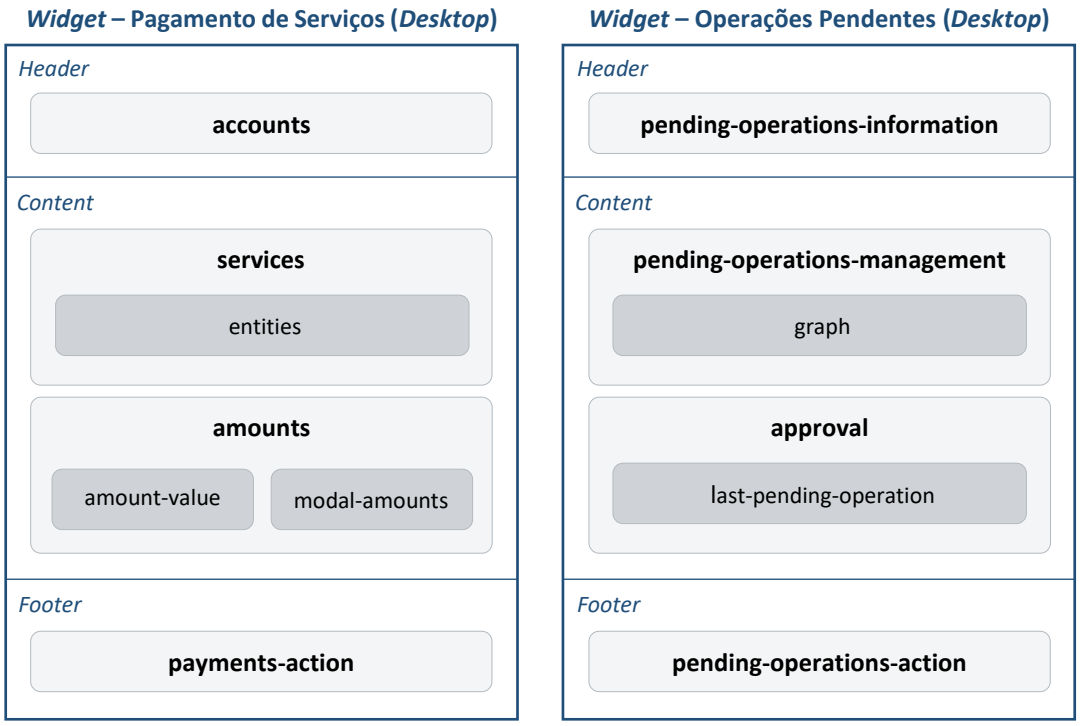


Figura 64 – Componentes que integram os *widgets* na versão *Desktop*, (a) Pagamento de serviços; (b) Operações Pendentes.

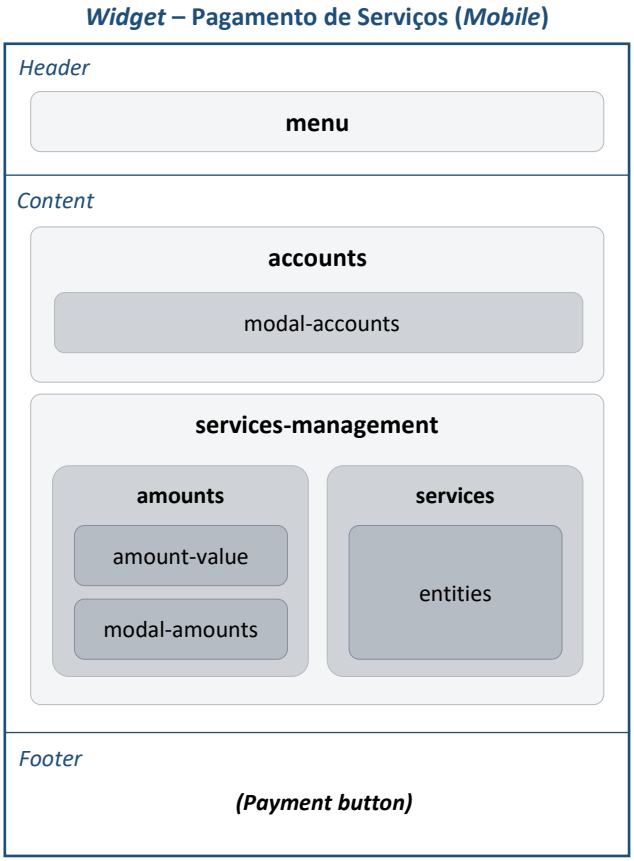
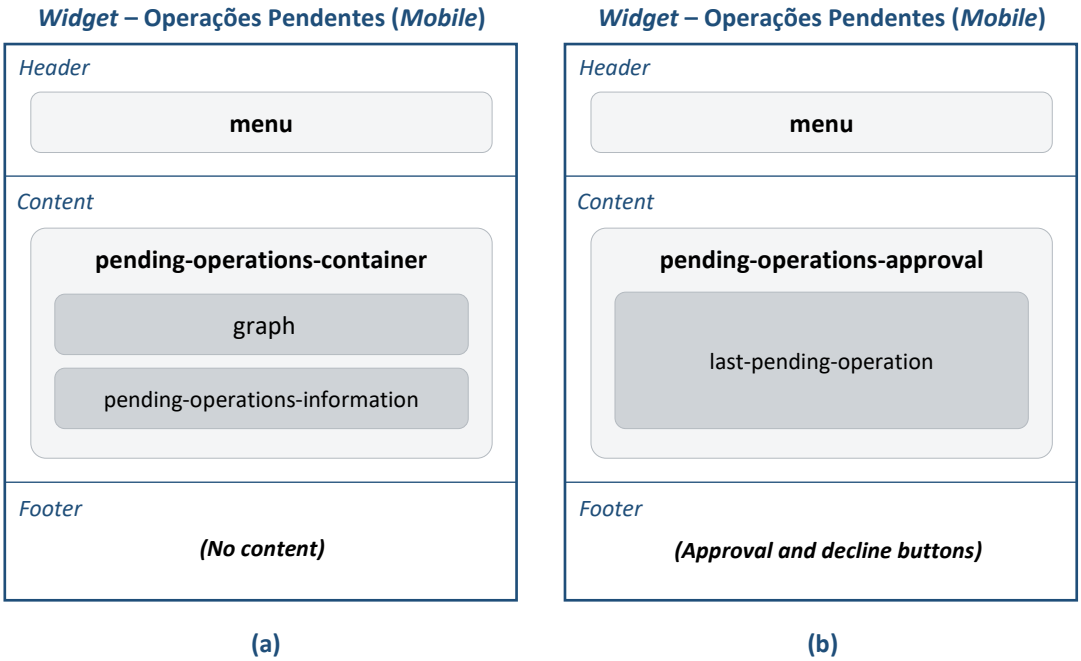


Figura 65 – Componentes que integram o *widget* de pagamento de serviços na versão *Mobile*.



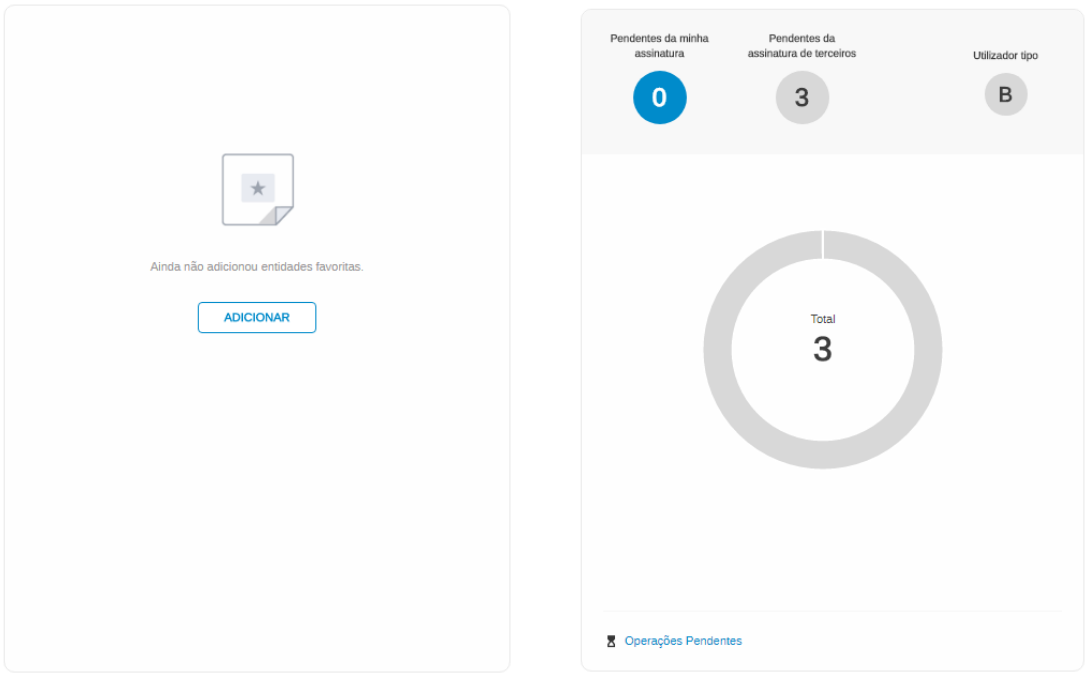
(a) **(b)**

Figura 66 – Componentes que integram o *widget* de operações pendentes na versão *Mobile*, (a) Ecrã principal; (b) Ecrã secundário.

ANEXO G. ESTRUTURA DE FICHEIROS

Figura 67 – Exemplos da estrutura de ficheiros do projeto.

ANEXO H. *WIDGETS* DESENVOLVIDOS



(a) (b)

Figura 68 – Na versão *Desktop*, (a) Ecrã do *widget* de Pagamentos de Serviços quando o utilizador não tem entidades favoritas; (b) Ecrã do *widget* de Operações pendentes quando o utilizador não tem operações pendentes para aprovação.

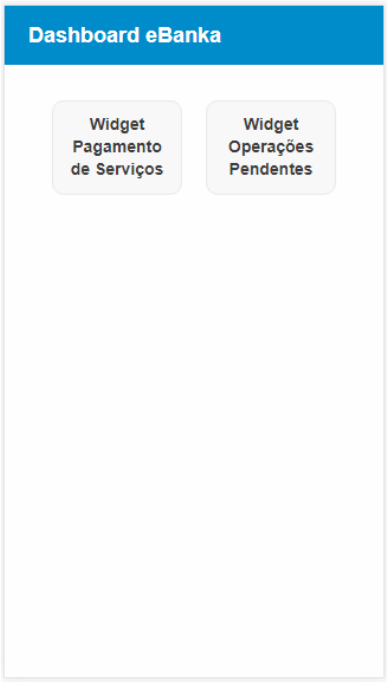
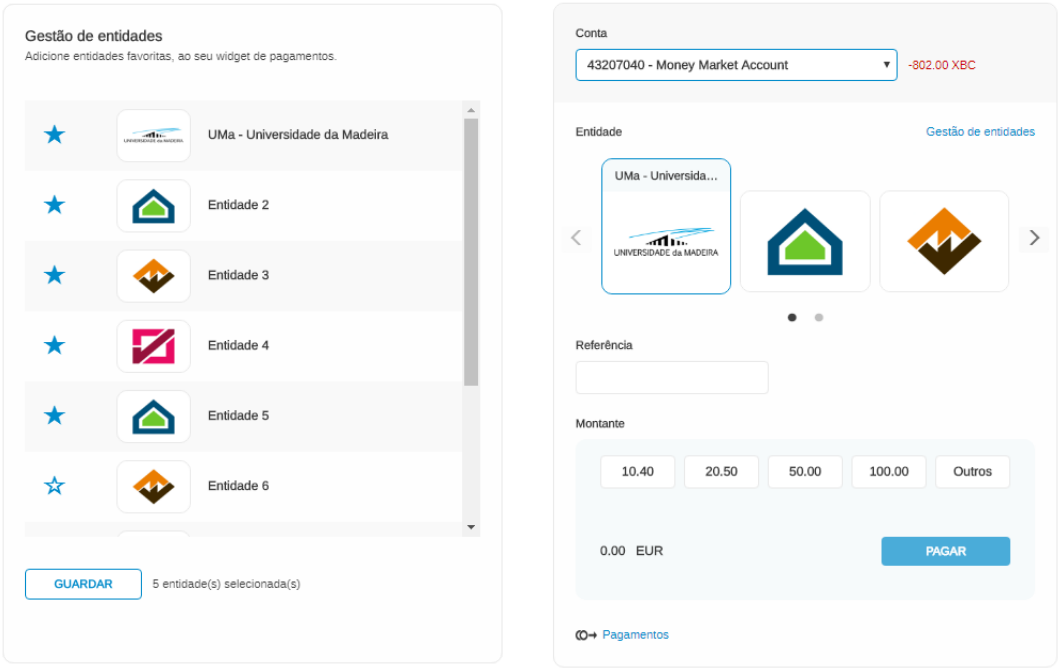


Figura 69 – *Dashboard* provisória da versão *Mobile* do *eBanka*.



(a) (b)

Figura 70 – Versão *Desktop* do *widget* de Pagamento de Serviços, (a) Janela de gestão de entidades favoritas; (b) Apresentação do saldo em negativo da conta do utilizador.

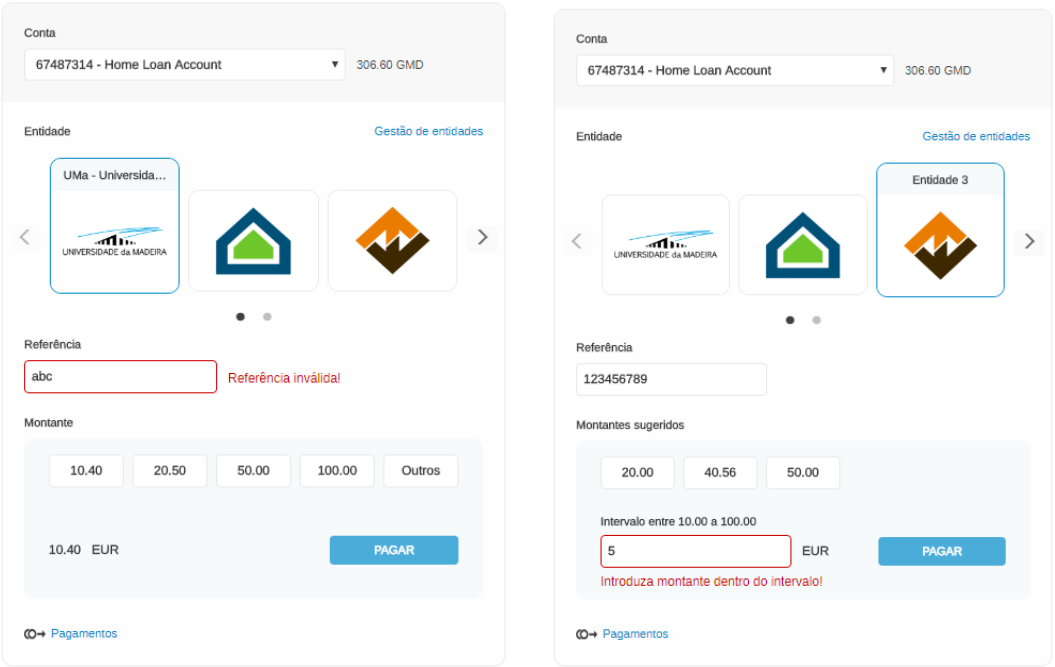
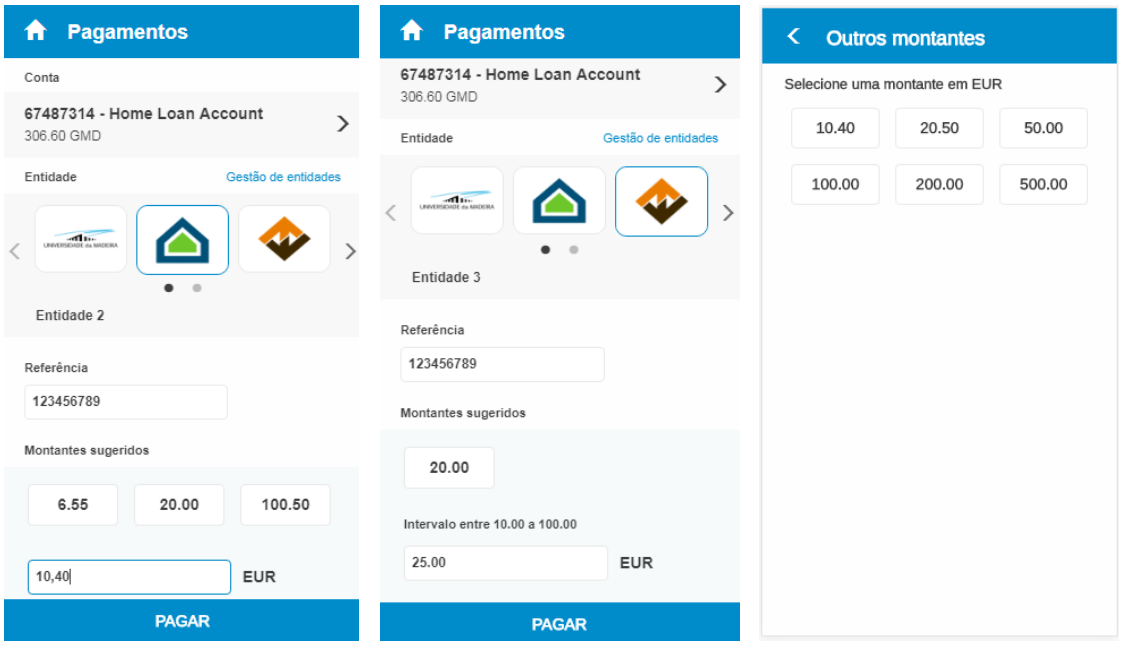
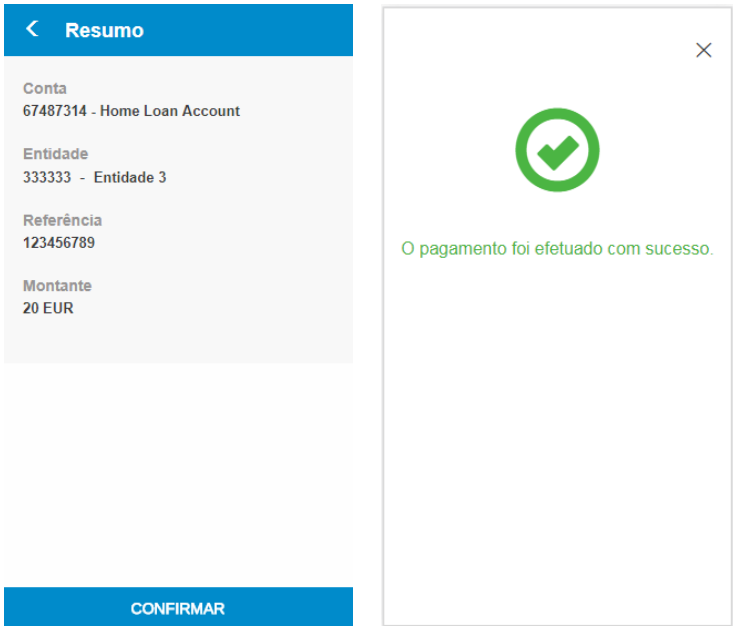


Figura 71 – Exemplos de mensagens de erros para o preenchimento do formulário de pagamento de serviços.



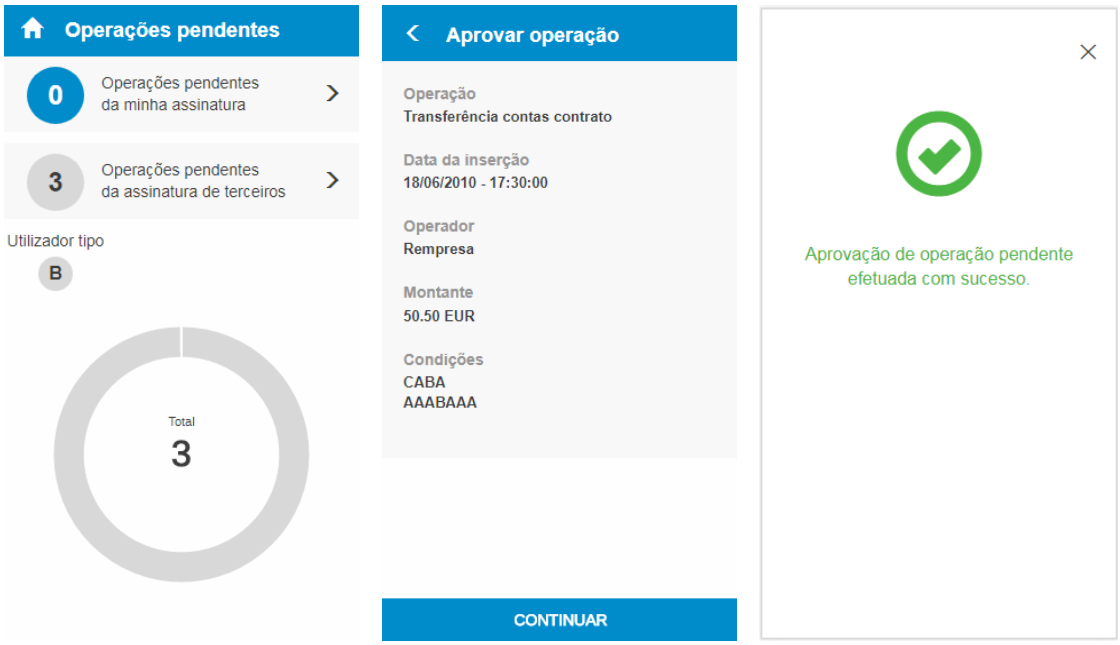
(a) (b) (c)

Figura 72 – Versão *Mobile* da página de Pagamento de Serviços, (a) Introduzir o montante em aberto; (b) Introduzir o montante dentro de um intervalo; (c) Listagem de todos os montantes predefinidos.



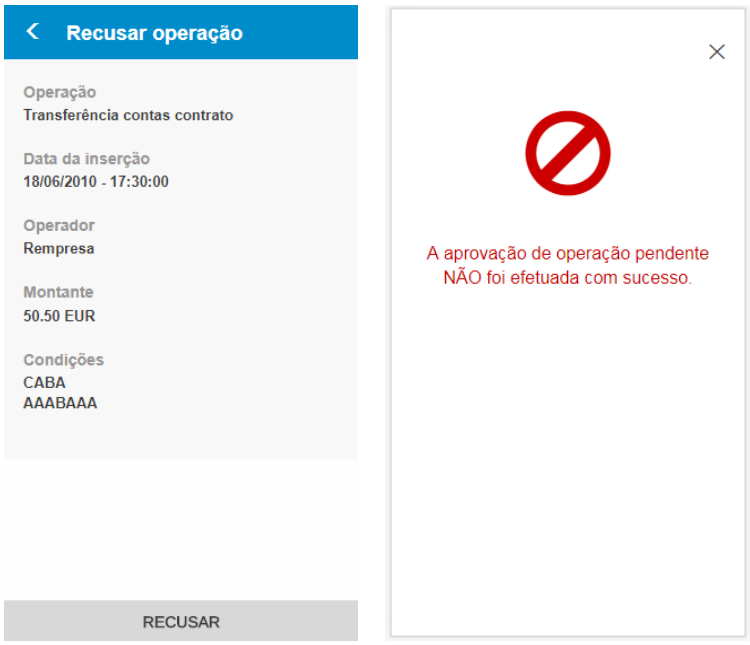
(a) (b)

Figura 73 – Versão *Mobile* da página de Pagamento de Serviços, (a) Ecrã do resumo com os dados do pagamento; (b) Ecrã da mensagem com o estado da operação.



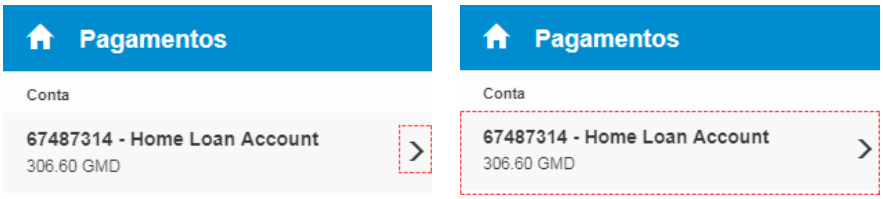
(a) (b) (c)

Figura 74 – Versão *Mobile* do *widget* de Operações Pendentes, (a) Ecrã principal quando o utilizador não tem operações pendentes para aprovação; (b) Ecrã do resumo da aprovação da operação pendente; (c) Ecrã da mensagem com o estado da operação.



(a) (b)

Figura 75 – Versão *Mobile* do *widget* de Operações Pendentes, (a) Ecrã do resumo para recusar a operação pendente; (b) Ecrã da mensagem com o estado da operação (quando existem falhas na aprovação da operação).



(a) (b)

Figura 76 – Zona clicável para alterar a conta a debitar no pagamento, (a) Versão anterior aos testes de usabilidade; (b) Nova versão após as alterações.

ANEXO I. GUIA DE TESTES DE USABILIDADE

Para o âmbito deste projeto, encontro-me a realizar dois *widgets* para um sistema bancário, que irão aparecer no teu *dashboard* quando entres na página do banco. Estes *widgets* estão ainda em fase de desenvolvimento, pelo que poderão existir falhas na sua interface. Para ajudar a identificar e corrigir os problemas que causam confusão, peço que completes as tarefas apresentadas e digas em voz alta tudo o que estás a pensar. São um total de 15 tarefas e podes pedir ajuda, se necessário. Todos os teus movimentos na interface serão gravados. Concordas?

Widget de Pagamento de Serviços

O primeiro *widget* ajudar-te-á a fazer o pagamento de algum serviço. Recordo que, para fazer um pagamento, precisas de uma conta bancária, uma entidade (serviço que pretendes pagar), o número de referência e finalmente o montante (valor a pagar). Tens, também, a opção de estabelecer entidades como favoritas, o que permitirá agilizar os próximos pagamentos!

- **Tarefas (Pagamento da mensalidade da UMa):**
  1. (**Versão *Mobile***) - Já te encontras na página do teu banco e estás a visualizar o teu *dashboard*, pretendes fazer um pagamento, pelo que vais escolher o respetivo *widget*.
  2. Agora queres alterar a conta a debitar, escolhe qualquer uma que tenha saldo positivo.
  3. Pretendes pagar, mensalmente, a mensalidade da universidade, pelo que o melhor será estabelecer essa entidade como favorita para que, a partir de agora, tenhas acesso direto a essa entidade no *widget*.
  4. Seleciona a entidade: “Universidade da Madeira”
  5. Introduz a seguinte referência: 123456789
  6. O montante a pagar são 100 euros.
  7. Já podes pagar!

Widget de Operações Pendentes

Imagina que tens uma empresa com vários funcionários. Muitas das operações que são feitas por eles na conta bancária da empresa precisam da tua aprovação. Este *widget* ajudar-te-á a saber quantas operações estão pendentes de ti. Além disso, poderás aprovar facilmente a operação que já depende da tua aprovação há algum tempo!

- **Tarefas:**
  1. (**Versão *Mobile***) - Estás na página do teu banco a visualizar o teu *dashboard*, pretendes saber se tens operações pendentes da tua aprovação, pelo que vais escolher o respetivo *widget*.
  2. **Responde a esta pergunta:** quantas operações estão pendentes de ti?
  3. **Responde a esta pergunta:** quantas operações já aprovaste mas estão pendentes de outras pessoas?
  4. Agora queres saber informações específicas sobre todas as tuas operações pendentes, acede a elas. Depois volta ao *widget*.
  5. Como tens várias operações pendentes e estás com pressa, decidiste que vais aprovar uma delas (por defeito é a que mais tempo está à espera da tua aprovação).
  6. Queres saber mais detalhes sobre essa operação, como farias?
  7. ***Para pessoas com conhecimento empresarial, responde a esta pergunta:*** *que condições estão em falta para concluir a aprovação da operação?*
  8. Já podes aprovar a operação pendente!

Já acabaste. Muito obrigada! Agora podes explorar livremente os *widgets*!

ANEXO J. DOCUMENTAÇÃO

A modo de comparação, a Figura 77 apresenta um exemplo de comentários para uma função seguindo as convenções do JSDoc e a Figura 78 um exemplo de comentários simples.

```
/**
 * Get all pending operations from web service
 * @function getPendingOperations
 * @return {Object} Return all pending operations
 * @example "operations": [
  {...
  },...
 */
function getPendingOperations() {
  return http.get('operations', {});
}
```

Figura 77 – Exemplo de comentários seguindo as convenções do JSDoc.

```
// Check total others pending operations
function getTotalOthersPendings(totalPendings, userPendings) {
  const othersPendings = totalPendings - userPendings;
  return othersPendings;
}
```

Figura 78 – Exemplo de comentários simples.

Module: accountsService

The accountsService module manages everything related to the web service that allows to get all user's accounts

Parameters:

Name	Type	Description
`http`	Object	HTTP Object

Source: accounts-service.js, line 1

Methods

`(inner)` `getAccounts()` → `{Object}`

Get accounts from web service

Source: accounts-service.js, line 15

Returns:

Return all user's accounts

Type
Object

Example

```
"accounts": [
    {
      "account_id": 70070131,
      "account_number": 67487314,
      "account_name": "Home Loan Account",
      "account_balance": {
        "value": 306.6,
        "currency": "GMD"
      },
      "account_contranct_number": 9884667
      "account_currency": {
        "code": "MOP",
```

```
        "description": "PATACA"
      },
      "account_iban": "PT5000020123123456
      "account_bic": "BDIGPTPL",
      "account_type": {
        "code": "DO",
        "description": "Contas a Ordem"
      },
      "in_total_balance": false,
      "mountly_spending_limit": {
        "value": 76,
        "currency": "OMR"
      },
      "created_date": "2016-02-05T13:50:0
      "spent_today": {
        "value": 814,
        "currency": "BWP"
      },
      "spent_this_month": {
        "value": 235,
        "currency": "SZL"
      },
      "spent_this_year": {
        "value": 719,
        "currency": "DKK"
      },
      "additional_fields": {
        "avaliable_balance": "949.00",
        "authorized_balance": "998.00",
        "last_extract_date": "2016-02-24T
      }
    },...
```

Module: entitiesService

The entitiesService module manages everything related to the web service that allows to get all entities

Parameters:

Name	Type	Description
`http`	Object	HTTP Object

Source: entities-service.js, line 1

Methods

`(inner)` **getEntities()** → `{Object}`

Get all entities from web service

Source: entities-service.js, line 16

Returns:

Return all entities

Type
Object

Example

```
"entities": [
    {
      "id": 1,
      "entity_id": 1,
      "entity_number": 111111,
      "entity_name": "UMa - Universidade
      "image_status": "enable",
      "type_amounts": "predefined",
      "predefined_amounts": [
        10.4,
        20.5,
        50,
```

```
        100,
        200,
        500
      ],
      "custom_image": "",
      "default_image": "",
      "isFavourite": true,
      "_links": {
        "self": {
          "description": "Get Entities",
          "method": "GET",
          "link": "/entities"
        },
        "favourite": {
          "description": "Get favourites I
          "method": "GET",
          "link": "/entities?favourite=1"
        },
        "toogleFavourite": {
          "description": "Toogle Entity's
          "method": "PATCH",
          "link": "/entities/{entity_id}/
        }
      }
    },
```

(inner) putEntities(id, entity) → {Object}

Parameters:

Name	Type	Description
id	Number	entity ID
entity	Object	entity

Source: entities-service.js, line 62

Returns:

Update entity

Type

Object

Documentation generated by JSDoc 3.5.4 on Tue Aug 15 2017 21:40:17 GMT+0100 (GMT Daylight Time)

Module: pendingOperationsService

The pendingOperationsService module manages everything related to the web service that allows to get all user's pending operations

Parameters:

Name	Type	Description
`http`	Object	HTTP Object

Source: pending-operations-service.js, line 1

Methods

`(inner)` `getPendingOperations()` → `{Object}`

Get all pending operations from web service

Source: pending-operations-service.js, line 16

Returns:

Return all pending operations

Type
Object

Example

```
"operations": [
    {
      "id": 1,
      "name_operation": "Transferência co
      "name_operator": "Rempresa",
      "amount_operation": {
        "value": 50.5,
        "currency": "EUR"
      },
      "date": "20100618T173000",
      "state": "pending",
      "conditions_approved": [
```

```
        "A",
        "A",
        "C"
      ],
      "conditions_operation": [
        [
          {
            "value": "C"
          },
          {
            "value": "A"
          },
          {
            "value": "B"
          },
          {
            "value": "A"
          }
        ],
        [
          {
            "value": "A"
          },
          {
            "value": "A"
          },
          {
            "value": "A"
          },
          {
            "value": "B"
          },
          {
            "value": "A"
          },
          {
            "value": "A"
          },
          {
            "value": "A"
          }
        ]
      ]
    },...
```


`(inner) putPendingOperations(id, operation)`

`→ {Object}`

Parameters:

Name	Type	Description
`id`	Number	operation ID
`operation`	Object	pending operation

Source: pending-operations-service.js, line 81

Returns:

Update pending operation

Type

Object

Module: userService

The userService module manages everything related to the web service that allows to get all user data

Parameters:

Name	Type	Description
`http`	Object	HTTP Object

Source: user-service.js, line 1

Methods

`(inner)` **getUser()** → `{Object}`

Get user data from web service

Source: user-service.js, line 14

Returns:

Return user data

Type
Object

Example

```
"user": [
      {
        "user_id": 1,
        "user_name": "Manuel Rodrigues",
        "user_type": "B"
      }
    ]
```