Dissertação/Estágio

Mestrado em Engenharia Informática

Plataforma de Gestão de Importação

Hugo Miguel Rijo Sousa da Conceição

hmiguel@student.dei.uc.pt

Júris:

Bruno Miguel Brás Cabral

Joel Perdiz Arrais

Orientadores:

Jorge Cardoso

Hugo Duarte da Fonseca

Departamento de Engenharia Informática

Faculdade de Ciências e Tecnologia

Universidade de Coimbra

Coimbra, 7 de setembro de 2015

Resumo

A área de transportes e logística é uma área de enorme importância na economia mundial. O grau de complexidade dos vários processos que a envolvem é elevado. Dentro desses processos podemos encontrar a prospeção, a negociação, o transporte, a documentação, entre outros. Todos estes processos envolvem necessariamente duas entidades, que representam a origem e destino de um transporte internacional, ou seja, uma entidade que exporta um produto e outra que importa esse mesmo produto. A complexidade e a competitividade deste processo leva a que nos dias correntes seja necessário existir uma forma eficaz de gerir o processo de importação aos olhos do importador, o que não acontece. Este projeto tem como principal objetivo desenvolver um protótipo de uma plataforma de gestão *online* de processos de importação que permita à entidade importadora gerir e ter uma visão global sobre o todo processo de transporte, ou seja, desde que adquire ao exportador determinada mercadoria até ao momento em que o importador a recebe nas suas instalações.

Palavras Chave: Importação, Gestão, Transportes, Plataforma, Web, Framework, API

Abstract

The area of transport and logistics is an extremely important area in the world economy. The degree of complexity of the various processes that involve is high. Within these processes we can find the prospecting, negotiation, transportation, documentation, among others. All these processes necessarily involve two entities, which represent the source and destination of an international transport, one entity which exports a product and one that import that product. The complexity and competitiveness of this process means that at this days is necessary to have an effective way to manage the import process, which is not happen casualy. This project aims to develop a prototype of an online management platform for import processes that allow the importing entity to manage and have an overview of the whole transport process, since from acquiring the exporting certain goods until the importer receives on its premises.

Keywords: Import, Management, Transport, Software, Web, Framework, API

"Don't do something because it's easy.
Do something because you want to do it.
Give yourself that challenge and you will not be sorry for it."

- Alice Bowman

Agradecimentos

Agradeço a todas as pessoas que me ajudaram a chegar até aqui. Familiares, amigos, colegas, professores, mentores e conhecidos, o meu eterno agradecimento.

Conteúdo

Lista de Figuras

Lista de Quadros

Acrónimos

API	Application Programming Interface
CRUD	Create Read Update Delete
CSS	Cascading Style Sheets
CSV	Comma-separated values
DOM	Document Object Model
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JS	JavaScript
MVC	Modal-View-Controller
OWIN	Open Web Interface for .NET
PaaS	Product as a Service
PDF	Portable Document Format
REST	Representational State Transfer
SaaS	Security as a Service
XLS	Microsoft Excel Spreadsheet File
XSS	Cross Site Scripting

Capítulo 1

Introdução

O presente documento tem como base o trabalho realizado no âmbito da disciplina de estágio do Mestrado em Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra no ano letivo 2014/2015.

O estágio foi realizado na empresa Maeil, Information Systems Engineering, uma empresa privada da área de tecnologias e sistemas de informação, fundada em 1999, com raízes no Instituto Superior Técnico e que ao longo destes anos se especializou na área de transportes de carga e logística.

O estágio teve a duração de dois semestres, sendo que o primeiro semestre foi dedicado ao estudo do negócio, estado da arte e levantamento parcial de requisitos. No segundo semestre procedeu-se à implementação do projeto, onde se incluiu também o desenho de arquitetura. De notar a ocorrência de um atraso no projeto, por motivos profissionais do autor, o que levou à extensão do mesmo em dois meses.

Todo este processo foi acompanhado pelo Doutor Jorge Cardoso, professor no Departamento de Engenharia Informática e pelo Engenheiro Hugo Duarte da Fonseca, *Managing Partner* da empresa Maeil.

Neste capítulo introdutório podemos encontrar o âmbito e respetivo enquadramento do projeto, quais os objetivos e importância para a realização do mesmo, uma nota sobre contribuições já existentes associadas ao projeto e por fim, um breve resumo sobre estrutura do documento.

1.1 Âmbito

No panorama atual, entre os diversos mercados existentes, é cada vez mais importante tirar partido da rede global, *internet*, no negócio ou atividade de uma entidade, tendo esta, ou não, uma implementação tecnológica na base do seu negócio. Este benefício pode ir desde o vulgar correio eletrónico até uma integração da atividade de negócio com uma componente tecnológica, como por exemplo, um canal de comunicação dedicado entre clientes ou parceiros de negócio. Existem diversas atividades económicas onde a troca de informação entre entidades associadas ao negócio é abundante, tratada de forma manual, muitas vezes desorganizada, e que consome quantidades enormes de recursos, humanos e de tempo.

Neste sentido, tomando como ponto de partida a área dos transportes de mercadorias e logística, nomeadamente, a importação de mercadorias internacionais, pretende-se desenvolver uma plataforma web que dê a possibilidade a entidades que tenham como atividade a importação de mercadorias, poder efetuar uma gestão completa do seu negócio, com a finalidade de aumentar a produtividade do mesmo. Esta gestão passa pela gestão de compras, gestão da procura das melhores cotações de transporte, gestão de mercadorias e marcações, gestão de faturação, gestão de armazéns e logística, entre outas funcionalidades.

1.2 Contribuições

Nesta secção podemos encontrar, de forma sintética, a especificação do trabalho que já estava desenvolvido pela Maeil, e que se enquadrava neste projeto, até à data de início do estágio deste estágio e aquilo que foi realizado pelo autor durante o estágio.

No que toca ao estudo e desenvolvimento da plataforma de importação, não existia até à data de início do estágio nenhum estudo ou desenvolvimento vocacionado à importação de mercadorias, contudo, foram fornecidos dados relativos ao processo de exportação, nomeadamente documentação e acesso a uma plataforma de exportação, que foram úteis para efetuar o estudo do modelo de importação e levantamento de requisitos.

Relativamente à plataforma de exportação existente, desenvolvida pela Maeil, foi desde o primeiro instante transmitido ao autor a necessidade, por diversos motivos, de redesenhar toda arquitetura da plataforma existente, quer em termos de *backend* e *frontend*. Neste sentido, o desenvolvimento de uma nova plataforma teria no futuro que suportar ou facilitar a integração da mesma no âmbito de exportação.

A nível do modelo de negócio de importação, o autor teve a colaboração do aluno Pedro Macedo, que ao mesmo tempo se encontrava a realizar um estágio curricular de Mestrado Integrado em Engenharia Civil pelo Instituto Superior Técnico da Universidade de Lisboa, com o estudo de fluxos de informação em transportes intermodais, na vertente de importação de mercadorias.

1.3 Estrutura do Documento

De maneira a que se tenha uma perceção melhor do conteúdo do relatório, nesta secção é descrita a síntese de cada um dos capítulos, que totalizam um total de 8:

- No capítulo 1 é feita uma introdução ao estágio e ao respetivo projeto, procurando dar ao leitor uma ideia clara que trabalho a desenvolver, através da caraterização do âmbito do projeto.

- O capítulo 2 trata o estudo do estado de arte. Neste estudo é contemplado o estado atual do negócio de transportes e de importação de mercadorias, quer a nível de tecnologias usadas, quer a nível os fluxos de informação existentes e a sua relevância. No final do capítulo, pretende-se que o leitor tenha uma noção básica do negócio e dos benefícios que este projeto pretende trazer às entidades envolvidas.

- No capítulo 3 é apresentado o planeamento do estágio ao longo do ano letivo, bem como a metodologia de trabalho seguida.

- No capítulo 4 é apresentada ao leitor a descrição dos requisitos, funcionais e não funcionais. São também descritas as restrições tecnológicas que devem ser seguidas durante o estudo e implementação do projeto.

- No capítulo 5 é abordada toda a arquitetura do projeto. Neste capítulo incluem-se também, para além do desenho da arquitetura, as ferramentas e tecnologias que irão ser utilizadas na implementação, com a respetiva justificação para utilização das mesmas. No final deste capítulo pretende-se que o leitor tenha uma noção exata do como estará construída a aplicação, quer a nível de *backend*, quer a nível de *frontend*.

- O capítulo 6 apresenta detalhadamente o modo de implementação com base no que foi estudado na secção de arquitetura.

- No capítulo 7 são abordados os testes unitários, de usabilidade e de desempenho que foram aplicados no trabalho desenvolvido, bem como a validação de funcionalidades.

- No capítulo 8 são apresentadas as conclusões finais sobre todo o percurso realizado durante o estágio e também sobre o trabalho futuro.

Capítulo 2

Estado da Arte

Neste capítulo irá ser apresentado o estudo do estado da arte efetuado no contexto do projeto.

Este estudo incide sobre o modelo de transportes, exportação e importação, mais concretamente sobre as trocas de informação existentes entre as várias entidades envolvidas num cenário típico de transporte internacional de mercadorias. Com este estudo foi possível ao autor adquirir conhecimentos sobre uma área para si desconhecida e que num âmbito menos técnico se mostrou ser bastante útil no desenvolvimento da plataforma.

O estado da arte teve também como objetivo perceber e demostrar a razão pela qual o desenvolvimento da plataforma de gestão de importação será uma mais-valia para os futuros utilizadores da aplicação e qual o tipo de informação que irá ser tratada e processada na plataforma. Para o leitor, este capítulo torna-se também um suporte para compreender o contexto do negócio e aprender alguns conceitos.

No final do capítulo, serão também analisadas e comparadas as caraterísticas de algumas ferramentas de desenvolvimento web, com vista a selecionar aquela que melhor se adaptará e que mais benefícios trará ao desenvolvimento projeto e à concretização dos objetivos do mesmo.

2.1 Transportes, Exportação e Importação

Atualmente vivemos numa época em que a internacionalização dos mercados atingiu níveis nunca antes experienciados. Globalmente temos assistido a um aumento gradual das trocas internacionais e um dos indicadores que nos mostra este aumento é transporte de mercadorias.

De forma a perceber a dimensão deste crescimento, tomemos o exemplo do transporte de carga por via marítima. A via marítima é atualmente o meio de transporte intercontinental de carga mais utilizado, na medida em que é mais económico, com a desvantagem de ser mais lento, comparativamente ao transporte por via aérea. No transporte por via marítima, são utilizados contentores como acondicionadores de carga, estes podem ter vários tamanhos e formas, sendo estes classificados pelo seu tamanho e tipo. Geralmente, é utilizada a unidade de medida TEU, *twenty feet equivalente unit*, para contabilizar o número de contentores, sendo que 1 TEU será um contentor de 20 pés e 2 TEU será, por exemplo, um contentor de 40 pés. Um navio de grande porte pode levar até 19.000 TEU, sendo que em média são transportados 10.000 TEU por viagem. O número de contentores transportados depende da categoria do navio, que agrupa os navios em função do comprimento, da largura, da capacidade e da profundidade. Para além do transporte de contentores, existe também o transporte de carga não contentorizada, que abrange veículos rolantes, carga granel sólida e liquida.

Na figura 2.1, é apresentado um gráfico com a evolução no número TEU transportados por ano, a nível global.

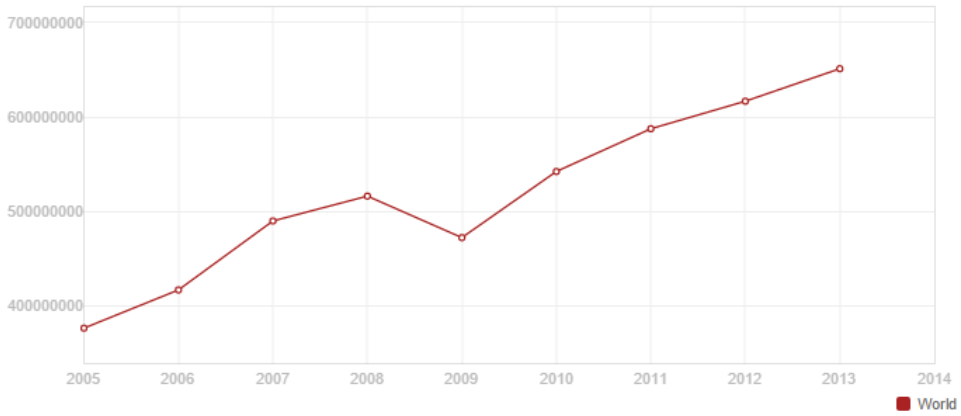

Figura 2.1: Número de TEU transportados mundialmente (2005-2013). Fonte: worldbank.org [1].

Se excluirmos o ano de 2009, ano marcado pela crise económica mundial, temos uma tendência crescente no mercado dos transportes de carga por via marítima. No entanto, o crescimento tecnológico, no que à gestão diz respeito, não tem acompanhado a evolução dos mercados e é cada vez mais difícil gerir o negócio, quer da exportação, quer da importação.

2.1.1 Agentes

Nesta secção pretende-se saber quais os agentes envolvidos num processo de transporte de mercadorias. Neste caso serão destacados aqueles que são mais significativos para o caso de estudo em questão.

Exportador

O exportador, também conhecido por expedidor ou consignador, é o agente responsável pelo envio da carga, que pode ou não ser o dono da respetiva mercadoria. Em grande parte dos casos, o exportador é também a entidade que vende a carga ao importador.

Importador

O importador está do lado oposto da cadeia de transporte, relativamente ao exportador. É também conhecido como consignatário e este será, por norma, o destinatário da carga, sendo que poderá ser o responsável pelo transporte de mercadoria. A responsabilidade é definida entre o importador e o exportador no ato da compra da mercadoria, esta responsabilidade decidida tendo em consideração os *Incoterms* (ver anexo A).

Transitário

O transitário é o agente responsável por fazer chegar a mercadoria ao importador, sendo este contratado pelo exportador ou pelo importador para executar o serviço. Cabe ao transitário gerir

toda a cadeia de transportes entre o local principal de carga e o local principal de descarga, de acordo com o contrato celebrado com o cliente. Para além do transporte, é comum o transitário tratar de toda a documentação e formalidades necessárias para o transporte da mercadoria.

Despachante Oficial

O Despachante Oficial é a entidade que trata as questões legais acerca da movimentação de bens entre dois países, quer a nível de exportação, quer a nível de importação. Existem diversas formalidades que devem ser cumpridas durante um processo de transporte e em grande parte dos casos, a melhor maneira de ter a certeza que todas essas regras e formalidades são cumpridas, é conveniente contatar um Despachante Oficial.

Transportadores

Os agentes transportadores são as empresas responsáveis pelo transporte físico das mercadorias, sendo que estes podem ser operadores rodoviários, ferroviários, aéreos ou marítimos. O nosso caso de estudo está focado no transporte marítimo, onde também intervêm os transportes rodoviários e ferroviários para a movimentação de mercadorias, desde a origem até ao porto de embarque e desde o porto de destino até ao destino final.

Armador

O armador é a entidade que explora o navio de transporte, sendo que esta pode ou não ser proprietária do navio. Os seus proveitos são obtidos com a cobrança de fretes marítimos às agências de navegação, pelo transporte de carga entre dois portos.

Operador de Terminal

Os operadores de terminais estão encarregues de gerir os terminais de embarque e desembarque de mercadorias. Estes controlam o próximo destino da mercadoria, que poderá ser o armazenamento, a consolidação de carga, mudança de transporte, entre ouras funções.

2.1.2 Fluxo de Informação

Durante a cadeia de transporte de determinada mercadoria, há diversas informações que são trocadas entre os vários agentes envolvidos no processo de transporte. Estas informações podem ser trocas de *emails*, envio de formulários web, telefonemas, etc. Existem também informações físicas, na forma de documentos, que são obrigatórias e que fazem parte do processo de transporte. Entre esta documentação podemos encontrar:

- Fatura Comercial – Documento de venda de mercadoria que contém a descrição da mercadoria e respetivo preço. É habitualmente fornecida pelo exportador.

- *Packing List* – Documento simples com a listagem da mercadoria a transportar.

- *Bill of Lading* ou Conhecimento de Embarque – Documento comprovativo do contrato de transporte marítimo entre o Armador e o Transportador. É emitido pelas agências de navegação e é um dos documentos mais importantes no transporte de mercadorias, pois este também serve de título de crédito e de propriedade da mercadoria.

- Certificado de Origem – Documento que permite ao importador, garantir a autenticidade de origem do produto importado, é fornecido ao importador pelo exportador.

- Documento Administrativo Único ou DAU – Documento a apresentar na alfândega, juntamente com o *Bill of Lading*, no caso de transportes marítimos, por forma a ter autorização para levantar a mercadoria.

Para aprofundar o estudo da troca de informação envolvida entre as várias entidades, foram contatados alguns intervenientes deste processo, nomeadamente um importador, um transitário e uma agência de navegação. A partir das informações recolhidas, foi possível esquematizar o fluxo de informação por forma a compreender melhor o seu funcionamento.

Na figura 2.2, temos uma esquematização simplificada de um processo de transporte, com a variante de crédito bancário. É comum os bancos do importador e exportador participarem no negócio de importação, esta ação tem o nome de crédito documentário. Com uso desta modalidade, o banco emitente (do importador) só efetua o pagamento da mercadoria se todos os requisitos necessários (a serem cumpridos pelo exportador) forem satisfeitos. As informações sobre este processo foram averiguadas junto do banco Millennium BCP [2].

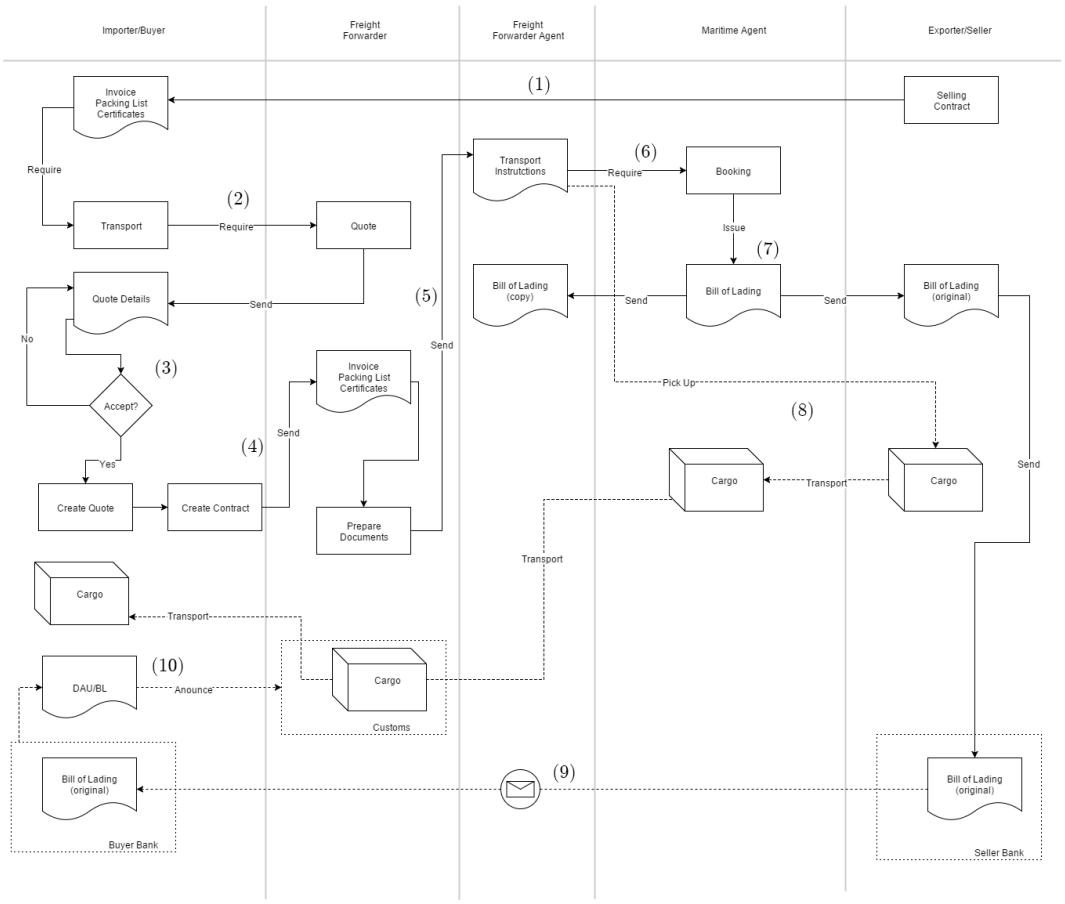

Figura 2.2: Fluxo de informação de um processo de transporte.

O processo inicia-se com o envio da documentação para o importador por parte do exportador (1). De seguida, conforme a modalidade de transporte acordada, o importador contata um transitário a fim de pedir uma cotação de transporte para um determinado tipo de mercadoria entre duas localizações (2), poderá ser o local de origem da carga e o local de destino. Neste ponto, o transitário irá oferecer um preço pelo transporte da mercadoria, que será ou não aceite pelo importador (3). Caso não seja, poderá eventualmente renegociar ou contatar outro transitário. Depois de conseguir um transitário que faça o transporte, o importador envia toda a documentação necessária para o transitário, habitualmente é a fatura comercial, a *packing list* e certificados de origem caso seja necessários (4).

Após receber a documentação, o transitário contacta uma agência parceira internacional, que normalmente é um transitário no país de origem da carga, no sentido de dar seguimento ao processo transporte da mercadoria (5). No país de origem, o transitário irá contatar com uma agência de navegação, caso o transporte seja por via marítima, para agendar o transporte da carga com destino ao país do importador (6).

O agente de navegação irá emitir o *Bill of Lading* da mercadoria e enviar para o Exportador o documento, entregando também uma cópia ao Agente Transitário (7), eventualmente poderá existir mais do que uma cópia original, que irá permitir fazer o levantamento da carga junto da alfândega no país de destino.

O passo seguinte será levantar a carga no local indicado pelo exportador e levá-la até ao terminal de contentores do porto de origem (8). Durante este processo, já ocorre a negociação entre o banco

representativo do exportador e do banco que representa o importador (9). Ao chegar a carga ao destino, o importador ou quem o representa, poderá levantar o documento de conhecimento de embarque junto do banco, para efetuar o levantamento da mercadoria na alfândega, juntamente com o documento administrativo único (10). Posteriormente, poderá haver necessidade de transportar a carga até ao destino final e assim termina o processo de importação.

Ao longo deste processo de transporte analisado foi possível perceber que os métodos de comunicação por parte do importador têm como base telefonemas e trocas de *emails*, seja a nível de pedidos de cotações, marcações e monitorização de carga. Este é um dos principais aspetos que uma aplicação de gestão de importações deve ter em conta, facilitar a comunicação entre agentes.

O levantamento descrito anteriormente, foi comparado com o padrão *standard* de transporte de mercadorias identificado no estudo de indicadores de desempenho de uma cadeia de transporte intermodal elaborado por Martin Posset, Hans Hauslmayer e Manfred Gronalt [3].

A figura 2.3 mostra-nos o fluxo de negócio *standard* de um transporte intermodal.

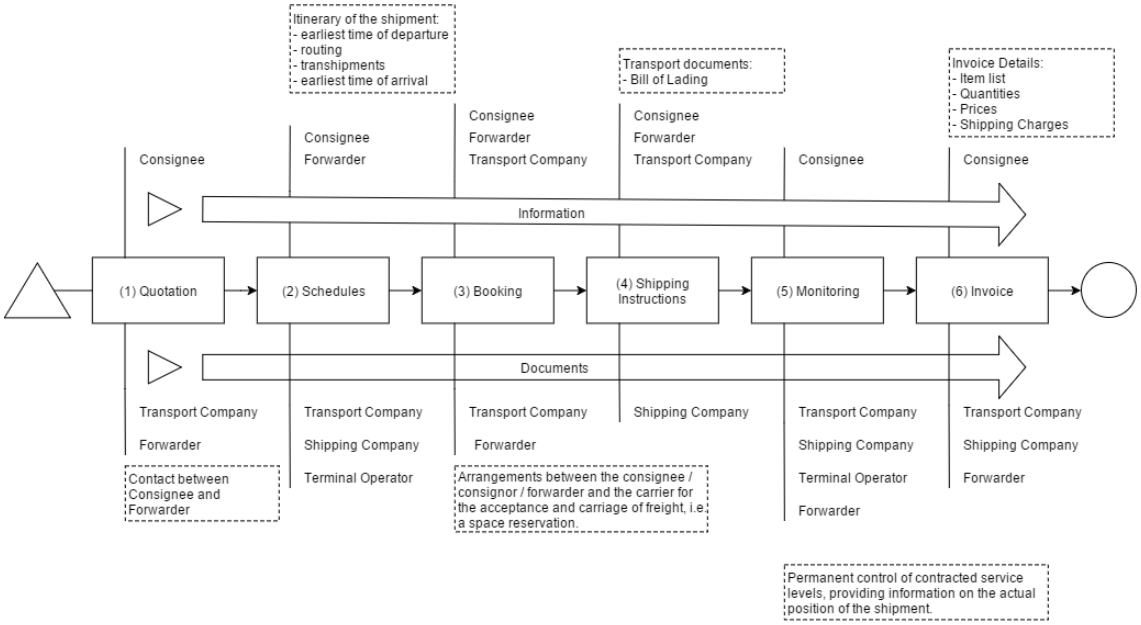

Figura 2.3: Fluxo de negócio de importação. Adaptado de COCKPIIT[3].

Neste esquema é possível identificar 6 fases distintas ao longo do processo: Cotações, Calendarização, Marcação, Instruções de Envio, Monotorização e Faturação. Este perfil de processo de transporte vai de encontro ao levantamento efetuado e caraterizado na figura 2.2. Ao longo deste documento iremos ver que este padrão acabou por ser a base da modularização da aplicação final, com algumas alterações devido ao levantamento de requisitos.

2.2 Produtos Concorrentes

Foi efetuada uma pesquisa sobre produtos concorrentes, no entanto não foi possível apresentar dados suficientes para uma análise comparativa sólida, isto aconteceu devido ao fato de esses produtos se apresentarem de modo fechado. No entanto, nesta secção são descritos os produtos encontrados com base na informação disponível nas suas páginas web.

2.2.1 Integration Point Import Management

Este produto faz parte de uma gama de produtos exclusivamente dedicados à área de comercio externo, importação e exportação. Este oferece uma visão global sobre o processo de importação, desde a compra do produto até à entrega final. É uma solução paga e é disponibilizada por uma empresa norte-americana, a Integration Point, Inc.

2.2.2 Apprise Software Import Management

O Apprise Software Import Management é um módulo integrado num ERP disponibilizado pela empresa Apprise Software, que permite a gestão de cargas. Do que foi possível avaliar, está limitada à gestão de cargas marítimas e não permite um controlo global sobre um processo de importação.

2.2.3 Amber Road Import Management

Esta é uma solução é um módulo disponibilizado pela empresa Amber Road, Inc. e que se foca no controlo e gestão de custos de um processo de importação. De acordo com os dados fornecidos pelo website da empresa, demonstra ser um produto sólido e bastante completo, sendo que, é por exemplo utilizado pela Santander Trade Portugal[1] na calculadora de exportação e importação disponibilizada no seu site, que é bastante completa a nível de funcionalidades.

2.2.4 Análise Comparativa

Funcionalidade	Int. Point	Apprise	Amber Road
Gestão de Transporte	Sim	Sim	Não
Gestão de Custos	Sim	Sim	Sim
Gestão de Produtos	Sim	Não	Sim
Comunicação Alfandegária	Não	Sim	Não
Validação de Dados	-	Sim	Sim
Gestão de Documentação	Sim	Não	Sim
Reutilização de Cenários	Não	Não	Sim
Monotorização de Carga	Sim	Não	Não
Integração ERP	Sim	Sim	-

Quadro 2.1: Funcionalidades oferecidas pelos produtos concorrentes.

No decorrer do segundo semestre, mais concretamente no mês de julho, foi lançado um novo produto pela empresa portuguesa Primavera – Business Software Solutions[5] intitulado de Comércio Externo e que segundo as informações apuradas, permite fazer a gestão de processos de importação. Devido ao *timing* do seu aparecimento no mercado, este produto não foi analisado ao detalhe, mas poderá ser um possível concorrente em determinadas funcionalidades.

2.3 Ferramentas de Desenvolvimento

Durante o processo de estudo da arquitetura do sistema, foi necessário reunir alguma informação sobre que ferramentas e tecnologias poderiam ser utilizadas na implementação do projeto. Umas das componentes do projeto que necessitou de um estudo mais aprofundado foi a escolha de uma framework de JavaScript. A opção do uso da tecnologia JavaScript para o desenvolvimento do projeto é baseada na arquitetura desenhada e que pode ser lida no capítulo 5 deste documento.

Nesta secção é descrito o levantamento efetuado acerca das frameworks existentes, sendo que apenas se efetuou o estudo sobre aquelas que reuniam um boa maturidade de desenvolvimento, de documentação e suporte. Para além destas características foi importante escolher uma framework que permitisse o desenvolvimento rápido face à extensão deste projeto. Este ponto é particularmente importante, porque o autor não tinha um profundo conhecimento sobre este tipo de frameworks, então a curva acentuada de aprendizagem era algo que deveria ser mitigada.

[1] Santander Trade - https://pt.santandertrade.com/expedicoes-internacionais/

2.3.1 Backbone.JS

Backbone.JS [6] é uma framework baseada na arquitetura MVC, foi criada em 2010 por Jeremy Ashkenas. Esta possui um licenciamento MIT, o que significa que pode ser usada comercialmente. As suas principais valências são a versatilidade e o seu minimalismo, no contexto em que precisa de poucas componentes para se conseguir criar um projeto funcional. Este atributo confere-lhe o grau de aplicação de excelência para projetos simples. Devido a isto, quando se entra em projetos de alguma complexidade, esta exige bastante investigação para se conseguir resolver alguns problemas, principalmente para programadores sem experiência de trabalho com a framework ou linguagem JavaScript. A ausência de mapeamento bidirecional nativo entre o JavaScript e o DOM HTML é um dos principais handicaps desta framework.

2.3.2 Ember.JS

Ember.JS [6] é também uma framework MVC, criada em 2011 por Yehuda Katz, um conhecido contribuidor da biblioteca JQuery[7] e Ruby-on-Rails[8]. Apresenta-se como uma framework de código aberto sob a licença MIT e tem sido mantida e desenvolvida pela sua comunidade de utilizadores. A Ember.JS surgiu numa altura em que a framework Backbone.JS dominava o panorama de frameworks Javascript e trouxe com ela uma série de funcionalidades nativas, que no caso do Backbone.JS só eram conseguidas com recurso a *plugins*, muitos deles sem suporte. Portanto, ter uma framework com todos os recursos disponíveis, sem requerer a procura extensiva de possíveis *plugins*, era uma grande vantagem. No entanto, o facto de ser uma framework bastante robusta, acaba por trazer a desvantagem da sua estrutura ser demasiado complexa e de difícil gestão. Mesmo sendo uma framework popular, tem bastantes lacunas a nível de documentação e suporte, o que torna difícil alguém inexperiente conseguir tirar partido do seu potencial.

2.3.3 Angular JS

Angular JS[9] é uma framework que nasceu em 2009, criada por Misko Hevery and Adam Abrons, sob o nome de GetAngular. Rapidamente foi noticiado pela Google o seu potencial e então surgiu a oportunidade de a Google apoiar o projeto. Atualmente a framework é mantida pela Google e pela comunidade Angular que também contribui para o projeto, uma vez que é um projeto de código aberto [10]. AngularJS é neste momento a framework mais popular entre todas frameworks Javascript, o que faz com que Angular JS seja a framework com melhor qualidade a nível de suporte. A nível de documentação a framework também está bem documentada e a sua curva de aprendizagem é relativamente baixa.

2.3.4 Análise Comparativa

Por forma a facilitar a comparação entre as três *frameworks* estudadas, reuniram-se os principais fatores que determinam a melhor escolha de uma *framework* javascript.

Características

Existem várias caraterísticas que uma boa *framework javascript* deve possuir, entre as quais, roteamento, uso de vistas parciais, mapeamento de dados bidirecionais, validação de formulários,

filtragens, entre outros. Para esta análise foram selecionadas aquelas que são as funcionalidades essenciais que uma framework deve ter para o desenvolvimento de uma aplicação web, são elas:

- Roteamento – Capacidade da framework alterar e aceder ao URL da aplicação e agir em concordância.
- Mapeamento em Vistas – Atualização automática de vistas quando ocorre alteração de um ou mais objetos pelo utilizador.
- Vistas Parciais – Possibilidade de incluir vistas em outras vistas.
- Mapeamento Bidirecional – Possibilidade de o valor de um objeto mudar simultaneamente durante a inserção de dados, por exemplo, num campo.
- Filtragens em Vistas – Possibilidade de filtragem de objetos presentes em vistas.

No quadro 2.2, é apresentada a comparação das características de cada uma das frameworks analisadas.

Característica	Ember.JS	Backbone.JS	Angular JS
Roteamento	Sim	Sim	Sim
Mapeamento em Vistas	Sim	Sim	Sim
Vistas Parciais	Sim	Não[2]	Sim
Mapeamento Bidirecional	Sim	Não[2]	Sim
Filtragem em Vistas	Sim	Não[2]	Sim

Quadro 2.2: Comparação de caraterísticas das frameworks analisadas.[2]

Flexibilidade

A flexibilidade, neste caso particular, entende-se pela flexibilidade da framework integrar outras tecnologias ou bibliotecas javascript. Para cada uma das frameworks estudadas, foi feita uma análise sobre a sua flexibilidade perante a integração de bibliotecas externas.

- Backbone.JS – Extremamente flexível no que toca a integração de bibliotecas externas, como por exemplo, JQuery. Isto deve-se ao facto de ser uma framework básica e que dependente de bibliotecas externas para implementar funcionalidades mais complexas.

- Ember.JS – Existem bastantes limitações quanto ao uso de bibliotecas de javascript externas, no entanto, a framework já oferece os recursos necessários para implementação das funcionalidades habitualmente utilizadas numa aplicação web.

- Angular JS – Na mesma linha que a framework Ember.JS, também a framework Angular tem bastantes limitações relativamente a bibliotecas de javascript externas, mas também oferece praticamente todas as funcionalidades necessárias.

Curva de Aprendizagem e Desenvolvimento

A curva de aprendizagem é um fator bastante importante quando estamos perante uma tecnologia da qual desconhecemos. Foi o caso deste projeto e tendo em conta o tempo disponível para a sua implementação, é importante perceber qual das frameworks poderia facilitar a tarefa de aprendizagem.

- Backbone.JS – Tem uma curva de aprendizagem relativamente pequena, no entanto, é bastante limitada a nível de funcionalidades, o que requer bastante tempo na pesquisa de soluções para implementar funcionalidades básicas.

- Ember.JS – Tem uma curva de aprendizagem mediana devido à sua complexidade, é necessário o investimento de algum tempo para conseguir iniciar a implementação de uma aplicação. A resolução de pequenos bugs pode ser difícil para quem não tem experiencia em Ember.JS.

[2] Em Backbone.JS também é possível ter as caraterísticas enunciadas através do uso de *plugins*.

- Angular JS – Tem uma curva de aprendizagem variável. No início é relativamente fácil de compreender a filosofia da framework e implementar pequenas funcionalidades, mas à medida que se vai explorando a framework, há diversas questões que surgem perante a complexidade e potencialidade da mesma e que podem atrasar o desenvolvimento do produto.

Documentação

A quantidade e qualidade da documentação é um importante complemento na aprendizagem de uma nova tecnologia, pois estas andam de mãos dadas. Uma boa documentação pode facilitar o processo de aprendizagem, é essencial para a implementação de funcionalidades e resolução de problemas.

- Backbone.JS – Possui uma boa documentação a nível de conteúdos, no entanto, poderia ter mais exemplos do que aqueles que são disponobilizados.

- Ember.JS – Documentação sólida, com qualidade, mas é notório que alguns exemplos estão ultrapassados e não refletem aquelas que são as boas práticas sugeridas atualmente.

- Angular.JS – Possui uma documentação sólida, com bastantes conteúdos, no entanto, pode tornar a sua consulta difícil quando não se está familiarizado com os conceitos próprios desta framework.

Comunidade e Popularidade

A presença de uma comunidade vasta e ativa de um produto significa que este é popular e tem um grau de satisfação bastante aceitável, tendo em conta que existem outros concorrentes. Este aspeto torna-se particularmente importante, porque significa que teremos apoio por parte de outros utilizadores a curto-médio prazo, o que pode ser um complemento importante à documentação existente, muito importante na resolução de problemas.

Existem inúmeras formas de proceder à comparação entre a popularidade entre as três frameworks. Uma delas é analisar a popularidade com recurso à ferramenta Google Trends [11], que mostra um gráfico comparativo entre termos pesquisados no motor de pesquisa da Google, que pode ser interpretado como indicador de popularidade.

Na figura 2.4 temos então o gráfico obtido através da plataforma Google Trends, para as três frameworks estudadas. Os valores verticais do gráfico representam o grau de interesse do termo relativamente ao ponto mais elevado do gráfico.

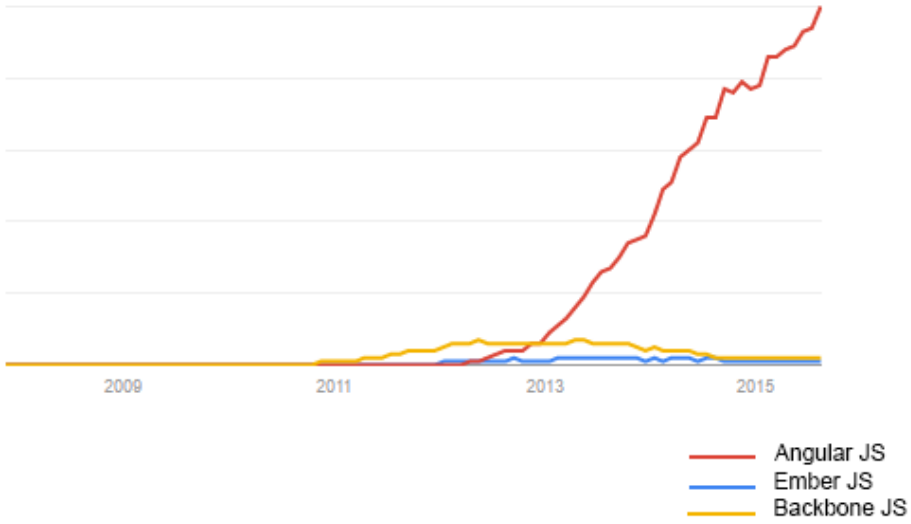

Figura 2.4: Gráfico de popularidade através dos termos que dão nome às frameworks.

A figura 2.4 mostra-nos que a framework Angular JS assume com clarividência a primeira posição no que diz respeito à popularidade entre os termos pesquisados. Este indicador é consolidado com os dados obtidos na plataforma GitHub[12] e StackOverflow[13], conforme os dados apresentados no quadro 2.5.

Indicadores	Ember.JS	Backbone.JS	Angular JS
Seguidores no GitHub	14528	22694	41662
Questões no StackOverflow[3]	2844	1634	47532

Quadro 2.3: Indicadores Github e StackOverflow.

Conclusão

Qualquer uma das frameworks analisadas parece ter um grau de maturidade suficiente para ser a escolha base de uma plataforma web, no entanto, é evidente o destaque da framework Angular JS sobre as outras, principalmente por ser aquela que atualmente dá maior confiança a quem desconhece o mundo das frameworks Javascript. No capítulo 5, o leitor perceberá que a opção selecionada recaiu sobre esta framework.

[3] Questões colocadas nos últimos seis meses com a *hashtag* das respetivas frameworks.

Capítulo 3

Metodologia e Planeamento

Neste capítulo é feita a descrição da metodologia de trabalho adotada, bem como a definição do plano de trabalho para ambos os semestres.

3.1 Metodologia de Trabalho

Para o desenvolvimento do projeto, foi utilizada a metodologia *scrum*.

Scrum é uma metodologia de desenvolvimento ágil de *software*, cuja principal mais valia é potenciar e controlar de forma eficaz o desenvolvimento de um produto. Funciona sob a forma de pequenas iterações de desenvolvimento, chamados de *sprints*. O *scrum* exige um controlo de calendarização bastante elevado a nível de desenvolvimentos e sabemos que nos dias de hoje é fundamental concretizar projetos dentro dos prazos previstos, devido a toda a competitividade existente. Para além disso, proporciona também uma melhor comunicação entre a equipa. A partir do momento que existem objetivos e requisitos claros num projeto, é possível implementar esta metodologia.

Depois de definidos os requisitos de um projeto, estes são inicialmente agregados e colocados num relatório de produto. A partir desse relatório de produto, é feita uma compilação de tarefas prioritárias a trabalhar, esta compilação de tarefas tem o nome de relatório de *release*. Deste relatório de *release*, nasce então o relatório de *sprint*, que representa o tempo de execução para o qual a tarefa terá que ser desenvolvida, verificada e testada.

Por norma, o espaço temporal de cada *sprint* não vai além das quatro semanas de trabalho. Caso o *sprint* não seja concluído com sucesso, dentro do espaço de tempo inicialmente definido, volta ao relatório de produto, dando-se assim início a uma próxima iteração. O fim de uma iteração é marcada por uma reunião de revisão de *sprint* e por uma reunião de retrospetiva de *sprint*, dois importantes acontecimentos que definem o sucesso da última iteração e o caminho a seguir na próxima.

De notar que para além destas reuniões no final de cada *sprint*, existe aquela que será talvez a mais importante, que é a reunião diária. Esta, como o nome sugere, tem uma periodicidade diária e é de curta duração, habitualmente é feita no início de cada de dia de trabalho, não ocupando mais do que quinze minutos. Serve essencialmente para controlo de progresso.

Outra das componentes da metodologia *scrum*, e que potencia o trabalho em equipa, é a obrigatoriedade da atribuição de papéis pré-definidos entre os membros da equipa. Este projeto, por ser de caracter académico/curricular, tem a particularidade de ser um trabalho de uma só pessoa, portanto, o conceito de equipa de desenvolvimento fica restrito a uma pessoa singular, no entanto, existem outros papéis que devem ser atribuídos com o propósito de seguir esta metodologia de trabalho. Portanto, seguindo as normas *scrum*, temos três papéis que devem ser atribuídos [14].

Papéis presentes na metodologia *scrum*:

- $ScrumMaster$: Papel responsável por garantir que todo o processo é compreendido e seguido nos termos definidos. Estará no cargo deste papel o Eng. Hugo Duarte da Fonseca.

- Proprietário do Produto: Papel responsável por maximizar o trabalho efetuado pela equipa de desenvolvimento. Estará no cargo deste papel o Eng. Hugo Duarte da Fonseca.

- Equipa Scrum: Equipa de profissionais com a responsabilidade de garantir que todos os requisitos do proprietário do produto são satisfeitos. A equipa neste caso será o autor do projeto.

Na figura 3.1, podemos então visualizar, sob forma figurada, o conceito de metodologia *scrum*.

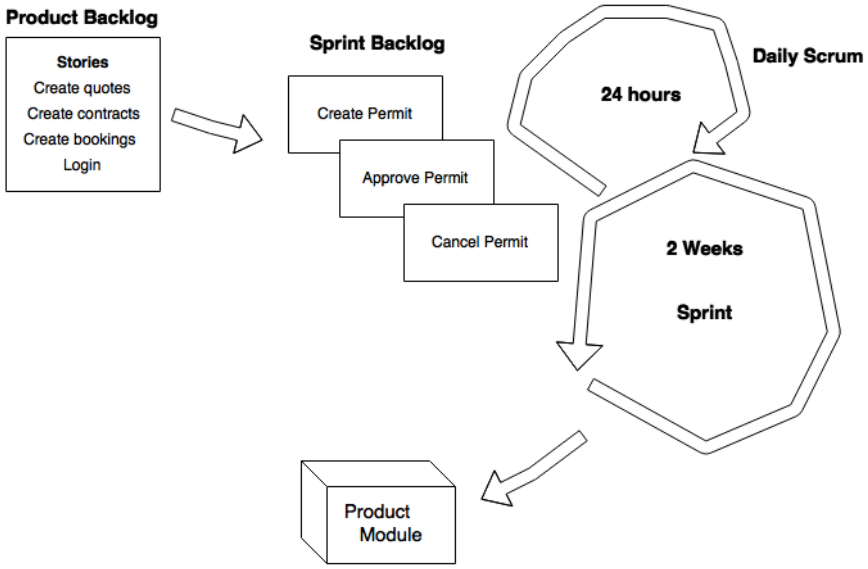

Figura 3.1: Metodologia *Scrum* (adaptado de Jonathan Rasmussom, 2014 [15]).

Como percecionámos, o *scrum* providência um bom controlo sobre o progresso de desenvolvimento do produto. Consegue-se adaptar, por exemplo, a situações onde seja necessário proceder a alterações de projeto sem prejudicar todo o trabalho feito até então, não comprometendo os prazos previamente estabelecidos.

A razão pela qual foi escolhida esta metodologia reside no facto de ir de encontro daquilo que se esperava deste projeto, porque ainda sem requisitos definidos, foi possível prever que o projeto se encaixava nesta metodologia.

Outro aspeto tido em conta foi facto de este ser um método de trabalho, em parte, já utilizado pela Maeil, fazendo com que ambas as partes saíssem beneficiadas neste processo de desenvolvimento.

3.2 Planeamento

Nesta secção são descritas as tarefas realizadas ao longo do ano, divididas pelos dois semestres. No anexo B, encontramos as diferenças entre os dois planeamentos efetuados, pré e pós estágio. Os planeamentos presentes no anexo estão representados sobre a forma de diagrama de Gantt [16].

3.2.1 Primeiro Semestre

No primeiro semestre estava inicialmente prevista a integração do autor no contexto de negócio e no âmbito da plataforma a desenvolver. Estava também previsto o estudo do estado da arte, a definição de requisitos e a especificação de arquitetura. Por razões profissionais, no primeiro semestre não foi possível realizar todas as tarefas previstas, sendo que o planeamento teve que ser reajustado e o estágio estendido até setembro de 2015. No quadro 3.1, estão listados os dados de plano de tarefas revisto relativo ao primeiro semestre.

Tarefa	Início	Fim
Integração do estagiário		
Configurações gerais do ambiente de trabalho	01/10/2014	05/10/2014
Estudo de ferramentas de trabalho	06/10/2014	08/10/2014
Metodologia e Planeamento		
Análise de metodologia de trabalho	09/10/2014	13/10/2014
Planificação e calendarização anual de trabalho	14/10/2014	16/10/2014
Contexto de Negócio		
Análise do âmbito do estágio	17/10/2014	25/10/2014
Estudo dos conceitos relativos ao negócio	26/10/2014	14/11/2014
Importação e exportação – Fluxo de Informação	15/11/2014	26/11/2015
Estado da Arte		
Estado da arte no contexto de negócio	27/11/2014	05/12/2014
Estudo de soluções existentes	06/12/2014	12/12/2014
Estudo de ferramentas disponíveis	15/01/2015	20/01/2015
Testes		
Análise e Testes de ferramentas	15/01/2015	29/01/2015
Relatório Intermédio		
Escrita do relatório	17/12/2014	14/01/2015

Quadro 3.1: Calendarização do primeiro semestre.

3.2.2 Segundo Semestre

À semelhança do primeiro semestre, também o segundo semestre teve que ser reajustado. No segundo semestre foram incluídas as tarefas inicialmente previstas para o primeiro semestre que não foram realizadas. Como referido no ponto anterior, houve necessidade do segundo semestre ser estendido até ao mês de Setembro.

No quadro 3.2, estão listados os dados de planeamento de tarefas relativo ao segundo semestre.

Tarefa	Início	Fim
Especificação de Requisitos		
Requisitos funcionais	01/02/2015	07/02/2015
Requisitos não-funcionais	08/02/2015	13/02/2015
Restrições tecnológicas	14/02/2015	16/02/2015
Arquitetura		
Análise e desenho de arquitetura	17/02/2015	27/02/2015
Análise e desenho do modelo de dados	28/02/2015	07/03/2015
Desenvolvimento		
Configuração do ambiente de desenvolvimento	08/03/2015	14/03/2015
Sistema de autenticação e autorização	15/03/2015	23/03/2015
Módulo de Ofertas	24/03/2015	04/04/2015
Módulo de Contratos	05/04/2015	09/04/2015
Módulo de Marcações	10/04/2015	19/04/2015
Módulo de Monotorização de Mercadoria	20/04/2015	28/04/2015
Dashboard	01/05/2015	10/05/2015
Módulo de Configuração e *BackOffice*	11/05/2015	25/05/2015
Servidor de Documentação	26/05/2015	02/06/2015
Documentação e Templates	08/06/2015	14/06/2015
Revisão de Interface Gráfica	15/06/2015	24/06/2015
Análise e Testes		
Testes sobre requisitos funcionais	25/06/2015	30/06/2015
Testes sobre requisitos não funcionais	01/07/2015	04/07/2015
Relatório Final		
Revisão de conteúdos relativos ao primeiro semestre	06/07/2015	12/07/2015
Escrita do relatório final	10/07/2015	20/08/2015
Revisão final de conteúdos	16/08/2015	30/08/2015

Quadro 3.2: Calendarização do segundo semestre.

3.2.3 Revisão de Planeamento

Como referido, houve necessidade de reajustar o plano de trabalho face à disponibilidade profissional do autor. Nos requisitos existem alguns módulos que não constam no plano indicado, estes foram descartados em função da prioridade definida aquando a definição de requisitos. A modularidade da arquitetura permitiu ter esta flexibilidade sem afetar o desenvolvimento e funcionamento da plataforma, afetando apenas o número de funcionalidades disponíveis. A nível de desenvolvimento, grande parte das tecnologias utilizadas eram novidade para o autor, sendo que foi preciso incluir no tempo de desenvolvimento a respetiva curva de aprendizagem necessária para cada uma delas. Este aspeto consumiu algum tempo extra no desenvolvimento que, aquando

a definição de requisitos e arquitetura, não estava totalmente ajustado à realidade. No entanto, ao assumir este risco, o autor foi recompensado com a aquisição de novos conhecimentos em termos de experiência tecnológica.

Capítulo 4

Análise de Requisitos

A definição de requisitos compõe uma das peças fundamentais no processo de desenvolvimento de *software*. É importante clarificar bem o problema (ou necessidade) de modo a elaborar um conjunto de requisitos sólidos e precisos que nos irão conduzir à solução do problema.

No caso particular deste projeto, os requisitos foram projetados com base na visão da Maeil sobre os objetivos da plataforma e também através da análise estado da arte por parte do autor, assumindo este, em conjunto com a Maeil, o papel de cliente. Foram também consultados dois parceiros da Maeil, a *Ibero Linhas* [17] e a *Sea World* [18], com o objetivo de esclarecer e validar alguns aspetos relativos ao fluxo de negócio e de informação, no âmbito da importação e transporte de carga.

A nível de definição de requisitos do sistema, estes são classificados em dois tipos: funcionais e não funcionais. Estes dois tipos de requisitos compõem respetivamente as secções deste capítulo.

4.1 Requisitos Funcionais

Por definição, os requisitos funcionais devem descrever todas as funcionalidades e serviços requeridos pela aplicação. Estes são construídos com base nas informações fornecidas pelo cliente, neste caso, a tarefa foi em parte facilitada, pois, como foi referido anteriormente, o autor e a Maeil assumiram o papel de cliente.

Contudo, foi preciso definir e clarificar como é que o sistema a implementar deveria comportar-se de modo a que o fluxo da aplicação fluísse de acordo com aquilo que são as diretivas do fluxo de negócio. Só deste modo seria possível corresponder à expetativa de potenciais clientes da aplicação, relembrando, que esse é um dos objetivos do estudo e desenvolvimento da plataforma de gestão de importação.

O quadro 4.1 apresenta os requisitos funcionais a definir, com a respetiva prioridade de implementação.

Requisito Funcional	Prioridade
Módulo de Negócio	2
Módulo de Ofertas	1
Módulo de Contratos	1
Módulo de Marcações	1
Módulo de *Tracking*	1
Módulo de Alfândega	2
Módulo de Logística	3
Módulo de Faturação	3
BackOffice	1
Configuração	2
Dashboard	2
Documentação	2

Legenda: 1 – Alta; 2 – Média; 3 – Baixa.

Quadro 4.1: Quadro de requisitos funcionais da plataforma de gestão de importação.

4.1.1 Módulo de Negócio

O fluxo de negócio começa precisamente com o módulo negócio. Este módulo tem o objetivo de permitir ao utilizador principal da plataforma, importador, gerir o seu processo de negócio com o exportador. Portanto, este módulo deve permitir ao importador fazer a gestão dos seus negócios com entidades exportadoras, registando os produtos adquiridos, condições de disponibilização da mercadoria, valores negociados, gestão de créditos documentários para com entidades bancárias e associação de documentação referente ao negócio, como faturas, certificados, entre outros documentos relacionados com o processo.

4.1.2 Módulo de Ofertas

O módulo de ofertas deve permitir ao importador solicitar e receber propostas de cotações de transporte de mercadorias. Após concretizar o negócio com o exportador e caso o transporte de mercadorias fique a cargo do importador, este tem que solicitar a uma entidade prestadora de serviços de transporte de mercadorias, o transporte da carga adquirida.

De forma a conseguir o melhor preço, o importador tem, por norma, que contatar diversas entidades do ramo dos transportes e solicitar uma cotação para esse transporte, este passo é um pedido de cotação. Por sua vez, as entidades contatadas face aos pedidos recebidos, devem enviar a sua cotação e condições de transporte para o importador. Posteriormente, o importador deverá analisar e selecionar aquela que será a proposta mais favorável, de entre todas as propostas recebidas. Neste âmbito, a aplicação deve disponibilizar um mecanismo de recolha de cotações para um determinado transporte de mercadorias.

Em resumo, sobre este ponto, o sistema deve permitir:

- Criar, alterar e apagar pedidos de cotação;
- Submeter pedidos de cotação a entidades prestadoras de serviços de transporte;

- Receber cotações de transporte de carga por parte de entidades prestadoras de serviços de transporte;
- Possibilidade de comparar e selecionar as melhores cotações recebidas.

4.1.3 Módulo de Contratos

O módulo de contratos tem como objetivo dar a possibilidade a um importador de fazer a gestão de contratos entre o importador e um transportador. A utilidade dos contratos prende-se com o facto de não existir necessidade de efetuar um novo pedido de cotação, caso o transporte se repita, por exemplo, na ocorrência das mesmas condições de transportes para os mesmos destinos e dentro do espaço de tempo em que uma cotação seja válida. Este acordo é assim um contrato celebrado entre as duas partes.

Em suma, o sistema deve permitir converter uma cotação selecionada em contrato e este deve ser um passo obrigatório. Para além da criação do contrato com base numa cotação, o sistema deve permitir alterar ou apagar um contrato.

O passo seguinte à criação de um contrato será a realização de uma marcação com base no contrato selecionado.

4.1.4 Módulo de Marcações

O módulo de marcações deve permitir ao importador efetuar a gestão das suas marcações de transporte. Uma vez contratualizada a intenção de efetuar um transporte de mercadorias, é necessário proceder à confirmação da marcação com o transportador. Antes da confirmação com o transportador, o sistema deve permitir registar os detalhes do transporte e da respetiva mercadoria que são exigidos pelo transportador. Os dados relativos ao transporte devem coincidir com os dados da cotação (e respetivo contrato), estes devem ser herdados pela marcação a fim do preenchimento da marcação ser menos exaustivo para o utilizador. Ficará a cargo do utilizador preencher os restantes detalhes da marcação. Após este passo, a confirmação da marcação deve ser enviada à entidade que irá efetuar transporte da mercadoria para que esta seja informada da escolha da sua cotação.

4.1.5 Módulo de Monotorização de Mercadoria

O módulo de monotorização de mercadoria, ou *tracking,* tem como objetivo permitir ao importador, a qualquer momento, ter informações sobre a posição terrestre ou marítima da sua mercadoria. Não sendo possível saber a posição em tempo real da mercadoria, o sistema deve pelo menos indicar a última posição conhecida da mercadoria. O sistema deve permitir pesquisar a posição da mercadoria através de uma referência da marcação.

4.1.6 Módulo Alfândega

O módulo alfândega deve permitir ao importador gerir o processo de desalfandegagem da sua mercadoria. Aquando a chegada de uma mercadoria de determinada marcação ao país e que seja necessária a intervenção do importador relativamente a processos de desalfandegagem, este módulo deverá possibilitar efetuar os registos sobre esse processo. O sistema deverá possibilitar a criação,

alteração e remoção de processos de desalfandegagem em função das marcações. Deverá também permitir agregar documentos ao processo de desalfandegagem.

4.1.7 Módulo de Logística

O módulo de logística deve permitir realizar a gestão de armazém e respetiva distribuição da mercadoria. Uma vez chegada a mercadoria ao importador, é necessário proceder à desconsolidação da mercadoria e armazená-la em armazém. Neste caso podem existir diferentes armazéns e zonas de armazém. Após o armazenamento, o sistema deve permitir atribuir a carga a um destino, para sua distribuição. Este é um dos módulos não prioritários da plataforma, mas que seria interessante implementar.

4.1.8 Módulo de Faturação

O módulo de faturação deverá permitir ao importador gerir as suas contas correntes. Todo processo de importação irá conter dados financeiros que permitem obter uma análise geral relativa ao estado do negócio de importação. Este módulo deve permitir ao utilizador saber quais os gastos e proveitos obtidos com cada processo de importação, desde a compra até à entrega. O módulo deverá estar preparado para poder integrar com *softwares* de sistema de informação, conhecidos por ERP [19].

4.1.9 *BackOffice*

A plataforma deverá ter um painel de *BackOffice*, que permita aos utilizadores gerir dados administrativos da aplicação. Este painel administrativo deverá permitir criar, alterar e apagar dados de utilizadores da plataforma.

O acesso à zona administrativa deverá estar restrita a utilizadores autorizados, como tal, deverá também ser possível gerir cargos e permissões de utilizadores. Estas permissões devem também permitir restringir o acesso por módulo em função do utilizador ou do tipo de utilizador.

4.1.10 Configuração

O sistema deve conter um painel de configuração de apresentação de dados da aplicação. Nas configurações deve ser permitido criar, alterar ou apagar, diferentes tipos de dados usados na aplicação. No tipo de dados associados a esta configuração incluem-se, por exemplo, os códigos de localização, de zonas, de países, de produtos e de equipamentos.

4.1.11 *Dashboard*

Deve existir um painel que permita aos utilizadores visualizar de forma resumida, todos os dados relevantes ao negócio. Numa primeira abordagem, estes devem conter indicadores com base nos módulos de Negócio, Ofertas, Contratos e Marcações.

4.1.12 Documentação

O módulo de documentação deverá permitir fazer a gestão de documentação que é associada aos registos nos diferentes módulos.

Para além da documentação, deverá ser possível gerar documentos customizados à medida de cada utilizador com as informações que cada módulo de negócio disponibiliza, numa primeira fase, os documentos a ter em consideração são os seguintes:

- Documento de cotação;
- Documento de contrato;
- Documento de marcação;

4.2 Requisitos Não Funcionais

Os requisitos não funcionais não pertencem à categoria de funcionalidades de negócio da aplicação. São requisitos que, sendo respeitados, contribuem para a qualidade da aplicação a ser desenvolvida. No quadro 4.2 estão listados os requisitos não funcionais e a respetiva prioridade em função do tempo de desenvolvimento.

Requisito Não Funcional	Prioridade
Segurança	1
Confiabilidade	1
Usabilidade	1
Desempenho	1
Disponibilidade	2
Portabilidade	2
Modularidade	2
Reusabilidade	2
Versionamento	1

Legenda: 1 – Alta; 2 – Média; 3 – Baixa.

Quadro 4.2: Quadro de requisitos não funcionais da plataforma de gestão de importação.

4.2.1 Segurança

O sistema deve garantir a integridade da informação da informação presente na aplicação, o que significa que, uma vez satisfeito este requisito, é assegurada a integridade do sistema face a ataques intencionais ou não intencionais. A concretização deste requisito implica a não autorização de acesso à aplicação ou dados associados da aplicação, a entidades ou pessoas que não estejam autenticadas e autorizadas para tal. A salvaguarda da informação também deve ser garantida, por exemplo, com recurso a um sistema eficiente de *backups*.

4.2.2 Confiabilidade

O requisito de confiabilidade carateriza-se pela não existência de comportamentos indesejáveis do sistema em tempo de execução. Significa isto que deve ser minorados erros não desejáveis, ou erros críticos de sistema. Assim sendo, podemos afirmar que este requisito é satisfeito quando o sistema está livre de erros catastróficos e que os resultados, ou eventuais erros, são previsíveis e compreendidos pelo utilizador da aplicação. Eventuais erros que possam surgir, cuja resolução não dependa do utilizador, devem ser recuperados pelo próprio sistema.

4.2.3 Usabilidade

O requisito de usabilidade diz respeito à interação que existe entre o utilizador e a aplicação, nomeadamente a nível de contato com a interface gráfica da aplicação. Nos termos deste requisito, a usabilidade pode ser medida pelo grau de facilidade com que o utilizador consegue manipular a aplicação.

Nesta interação utilizador-aplicação, deve ser tomada em consideração a capacidade da tecnologia informar o utilizador sobre as suas ações. De acordo com a norma ISO 9241, que diz respeito à usabilidade e interação humano-máquina, encontramos cinco características, ou requisitos, que um produto de *software* deve apresentar e que devem ser respeitados no âmbito de desenvolvimento:

- Aprendizagem: a interface deve ser de aprendizagem fácil e intuitiva, para quem a utiliza pela primeira vez.

- Memorização: a conceção da interface da aplicação deve ter em conta a fácil memorização da mesma por parte do utilizador, caso este não interaja com esta durante um período de tempo.

- Produtividade: a interface deve maximizar a produtividade do utilizador, permitindo que o utilizador realize as suas tarefas de forma rápida e eficiente.

- Erros: a interface deve procurar minimizar a taxa de erros por parte do utilizador, informando o utilizador sobre eventuais erros e permitir que estes sejam facilmente corrigidos.

- Satisfação: a interface deve maximizar a satisfação do utilizador através da confiança e segurança dada ao utilizador.

4.2.4 Desempenho

O requisito não funcional de desempenho é um importante requisito de qualidade da aplicação, pois é aquele que, do ponto de vista do utilizador, mais sobressai. É um requisito que pode entrar facilmente em conflito com outros requisitos de qualidade, por exemplo, com o requisito da usabilidade. Assim sendo, a prioridade deste requisito deve ser pensada e balanceada de modo a que haja um equilíbrio entre o desempenho e outros requisitos que dependam deste.

4.2.5 Disponibilidade

É importante garantir que o sistema, sendo a aplicação orientada a acessos remotos por vários utilizadores, tenha uma disponibilidade de funcionamento na ordem dos 99%. Significa este valor percentual que em 100 pedidos efetuados pelo utilizador à aplicação, 99 terão que ser satisfeitos por forma a não prejudicar o trabalho do utilizador. Note que não se incluem nos pedidos não satisfeitos, pedidos cuja resposta seja afetada por condicionantes externas à aplicação.

4.2.6 Portabilidade

O sistema deve garantir a portabilidade entre diferentes ambientes de *hardware* ou *software*. No caso particular desta aplicação, que será acedida via *browser*, é necessário garantir que é possível utilizá-la nos diferentes browsers presentes no mercado, ou pelo menos, naqueles que são mais utilizados.

4.2.7 Modularidade

Deve ser aplicado, se possível, o conceito de modularidade no desenho de arquitetura de *software*, por forma a garantir a não dependência entre componentes e facilitar a integração ou substituição das mesmas com futuros desenvolvimentos.

4.2.8 Reusabilidade

No desenvolvimento da aplicação, deve ser tido em consideração o conceito de reusabilidade de *software*. Significa isto que, o projeto deve desenvolvido com vista a potenciar novos projetos de *software*, ou seja, de onde se possa extrair conhecimento no futuro, reaproveitando esforço e minimizando custos de desenvolvimento. Esta reutilização de conhecimento pode ser entendida como, reutilização do modelo de negócio, da arquitetura, de módulos, de código, de interfaces, entre outras partes integrantes do projeto.

4.2.9 Versionamento

Deve existir um sistema estável de controlo de versões, com o objetivo de controlar as versões desenvolvimento e manter um histórico de alterações.

4.3 Restrições de Implementação

O uso de ferramentas externas para o desenvolvimento de um projeto *software* tem a vantagem de acelerar o processo de desenvolvimento. É vantajoso, por exemplo, para equipas com menos recursos humanos e que tenham um curto espaço de tempo para o seu desenvolvimento. No entanto existem contrapartidas que algumas destas ferramentas requerem e que podem não ir de encontro à nossa disponibilidade ou, neste caso concreto, da Maeil. O uso destas ferramentas tem que ser ponderado e avaliado, para isso foram delineadas algumas recomendações sobre o uso destas ferramentas.

- Licenciamento – Não devem ser utilizados recursos de apoio ao desenvolvimento que acarretem custos de licenciamento à Maeil. Devem ser priorizadas ferramentas de parceiros da Maeil ou de código aberto.

- Integração – Deve estar presente a visão de integração nas tecnologias a utilizar, ou seja, deve ser evitado utilizar tecnologias que não permitam uma fácil integração com outras tecnologias que possam vir a ser utilizadas no futuro.

- Distribuição – A distribuição do *software* deve considerar uma distribuição SaaS, mas também uma distribuição do tipo PaaS. Assim sendo, terá que ser pensada uma solução que tenha em consideração estes dois casos.

Capitulo 5

Modelo de Arquitetura

Neste capítulo será abordada a arquitetura da plataforma de gestão de importação, que foi estudada e desenhada com base no estudo do estado da arte, tendo em consideração os requisitos funcionais e não funcionais apresentados no capítulo 3.

Esta proposta de resolução do problema é assim composta pela descrição de cada uma das componentes do projeto e algumas comparações relativas a decisões tomadas. Na parte final do capítulo podemos também encontrar as tecnologias que foram escolhidas para a implementação do projeto, onde são dadas as respetivas justificações para as suas escolhas.

5.1 Modelo de Arquitetura

A arquitetura concebida assenta no modelo clássico *cliente-servidor* por camadas, onde podemos encontrar as três seguintes componentes:

- Camada de Apresentação: Esta é camada responsável pelo interface humano-máquina. É nesta camada que os utilizadores interagem com aplicação e tem acesso às suas funcionalidades. O meio ou dispositivo com que utilizam a aplicação fica a critério do utilizador, sendo que neste protótipo apenas será disponibilizada uma plataforma com acesso via *browser*, que obviamente estará disponível em qualquer dispositivo *smartphone* ou *tablet*.

- Camada Lógica: É a camada que faz a ponte entre o utilizador e os dados da aplicação. Nela é feita toda a transformação de lógica de dados e é nesta camada que estão presentes as regras sobre aquilo que pode ser mostrado, criado, modificado ou apagado, por parte do utilizador, mediante as regras do negócio.

- Camada de Dados: Na camada de dados são guardados todos os dados relativos à plataforma.

5.2 Camada Lógica

Conforme abordado no capítulo anterior, um dos requisitos seria que a plataforma deveria funcionar em ambiente web, desde modo, o primeiro esboço da arquitetura resultou no esquema que pode ser visto na figura 5.1.

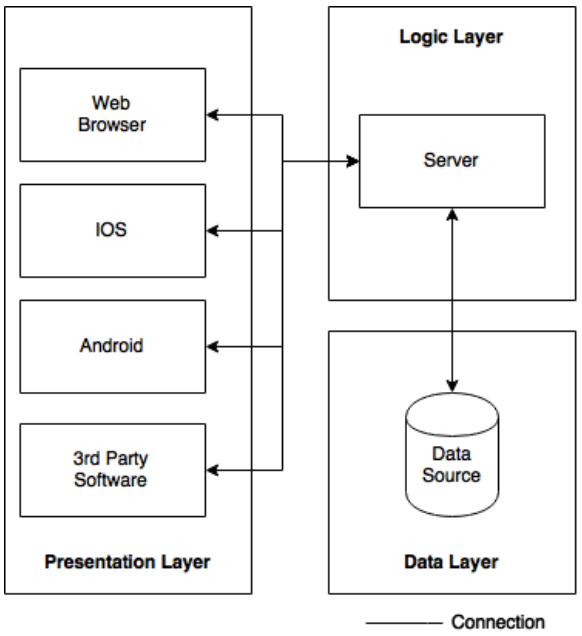

Figura 5.1: Arquitetura tipo de uma aplicação a correr em ambiente web.

A camada lógica faz a ponte entre a camada de apresentação e a camada de dados. É nesta camada que definimos as regras do negócio e se aplicam todas as transformações de dados que chegam do utilizador à base de dados e vice-versa.

O primeiro passo foi definir como seria feita a distribuição de dados que são encaminhados para camada de apresentação. A forma clássica de o fazer, em aplicações web, segue o estilo de renderização de conteúdo no servidor, onde os documentos HTML são renderizados e enviados para o *browser* do utilizador, como resposta aos seus pedidos.

Note a figura 5.2, onde é ilustrada a arquitetura de renderização no servidor.

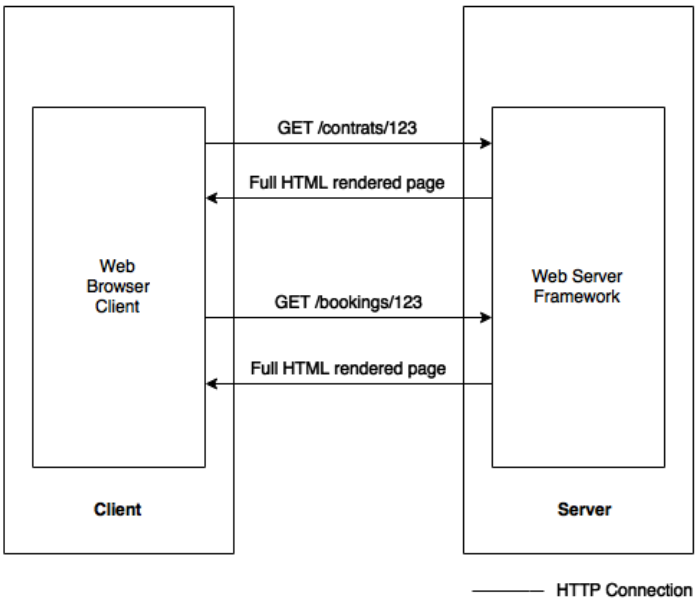

Figura 5.2: Arquitetura de renderização de páginas HTML no servidor.

Como podemos visualizar na figura acima, todos os pedidos efetuados ao servidor devolvem um documento HTML renderizado de acordo com o pedido do cliente, este documento é interpretado pelo *browser*. Deste modo, o *browser* só tem que efetuar uma ação de leitura do documento recebido e mostrar o resultado ao utilizador.

A situação descrita é a forma clássica em desenvolvimento de aplicações web, contudo, recentemente começaram a surgir aplicações web onde a renderização de documentos não é feita no servidor, mas sim no *browser*. Isto é possível graças às *frameworks* que têm sido desenvolvidas nos últimos anos, cujo desenvolvimento foi acompanhado pela evolução dos *browsers* de internet.

As aplicações que fazem uso desta arquitetura, são normalmente conhecidas por *Single Page Aplications*. Uma das aplicações web que utiliza esta arquitetura é o Gmail ou Google Mail[19], desenvolvido pela Google. Facilmente nos conseguimos aperceber quando estamos perante uma aplicação que é renderizada no servidor, pois quando assim acontece, não existe o típico refrescamento de página quando acedemos às diferentes hiperligações presentes na página, tornando-se a interação do utilizador com a aplicação extremamente fluida.

Na figura 5.3, podemos ver como funciona, de uma forma simples, a renderização de documentos HTML no cliente.

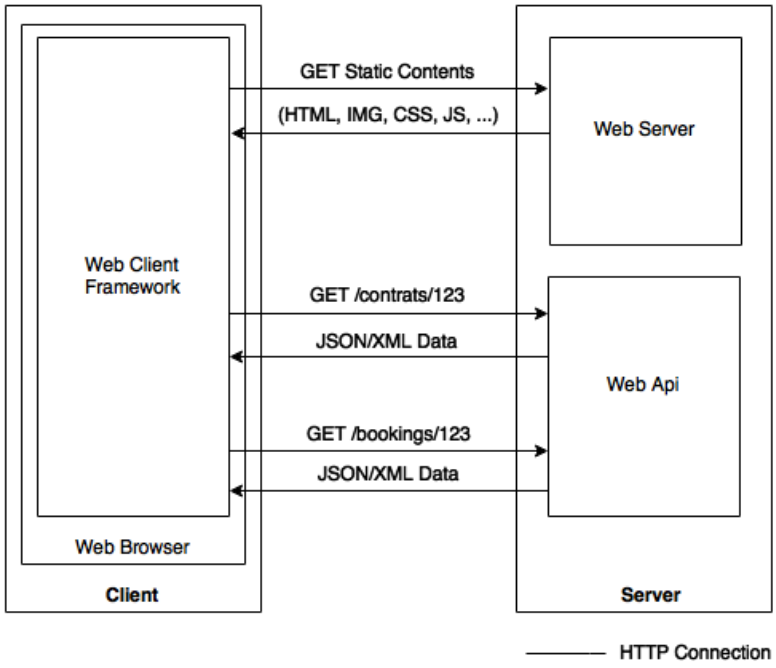

Figura 5.3: Arquitetura de renderização de páginas HTML no cliente.

No exemplo acima, todos documentos estáticos, como páginas de modelos (*templates*), imagens, ficheiros CSS e JS, estão alojados num servidor web e podem ser todos carregados quando a página principal é acedida (ficando posteriormente guardados em cache). Neste exemplo, todos os dados dinâmicos da página são obtidos através de uma Web API e a ligação de dados (*data binding*) com o template HTML é concretizado através do uso de uma *framework* que corre diretamente sobre o *browser* do utilizador. Estes dados dinâmicos chegam habitualmente à *framework* em formato JSON (ou XML) e são obtidos através pedidos AJAX [20].

Uma das grandes vantagens deste método é que permite reduzir bastante o nível de processamento do servidor da aplicação, uma vez que o trabalho de renderização das páginas fica totalmente do lado do cliente. Para além disso, o tráfego de dados entre cliente e servidor é drasticamente reduzido, o que é benéfico quando a aplicação é acedida por *smartphone*s.

Existem algumas desvantagens em usar um renderização de páginas no cliente. O tempo de carregamento inicial da página é quase sempre maior do que o tempo da renderização de páginas no servidor, dependendo no número de dados estáticos a carregar. A performance da aplicação também pode ser afetada negativamente em função da máquina do utilizador, *hardware* mais antigo reduz a performance da aplicação, ao passo que no caso da renderização no servidor, a performance depende da capacidade de processamento do servidor e não da máquina do utilizador. Esta foi a principal razão pela qual a empresa Twitter[21], decidiu abandonar em 2012 a renderização de páginas de internet no cliente e passou a renderizar as páginas no servidor. No entanto, nos últimos três anos houve uma evolução significativa a nível de *hardware* nos utilizadores comuns e das respetivas web *frameworks*, o que causa que as diferenças de performance não são tão significativas como eram quando este tipo de abordagem de arquitetura começou a

ser utilizada. A satisfação dos utilizadores a nível de usabilidade das aplicações renderizadas no cliente, por certo, supera as diferenças de níveis de performance de *frameworks* servidoras e *frameworks* cliente.

Tendo estas duas possibilidades em aberto para o desenvolvimento da aplicação, optou-se pelo uso de uma *framework* de renderização de páginas HTML no lado cliente. Esta escolha prendeu-se com o facto de no futuro existir a possibilidade de vir desenvolver uma aplicação móvel nativa e/ou integração com outras aplicações externas. Este cenário passaria sempre pelo desenvolvimento de uma Web API, de modo que era crucial preparar a arquitetura para facilitar os desenvolvimentos futuros. Esta arquitetura trás também uma redução de custos substancial face à arquitetura clássica, pois os recursos de processamento são menores.

A nível de desenvolvimento da aplicação, esta arquitetura tem a vantagem de existir a total separação de conceitos entre *backend* e *frontend*. Como estas camadas correm em ambientes distintos, não existe dependência de *software* entre as duas camadas e pode existir um maior controlo entre aqueles que são objetivos das duas componentes, evitando-se assim incluir lógica de negócio na camada de apresentação e lógica de usabilidade na camada de lógica do negócio, como habitualmente acaba, erradamente, por acontecer com o desenvolvimento de funcionalidades.

Assim sendo, o modelo de arquitetura ficou desenhado da seguinte forma (ver figura 5.4):

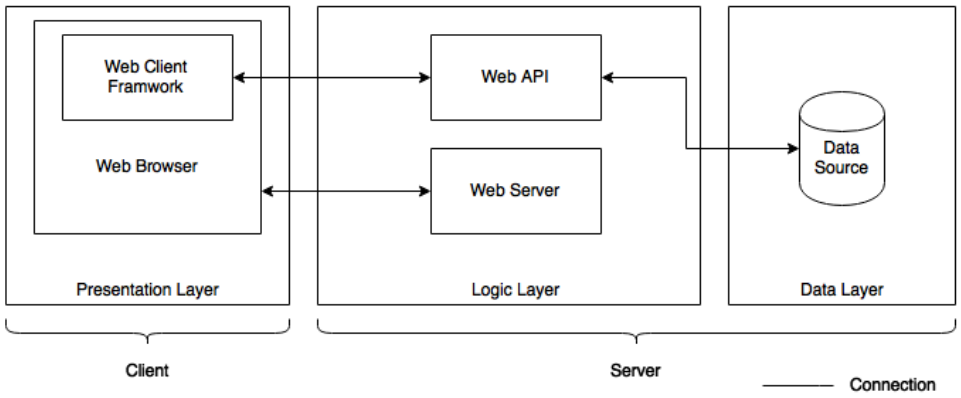

Figura 5.4: Modelo de arquitetura da plataforma de gestão de importação.

5.2.1 Comunicação e Transporte de Dados

A melhor forma de tirar partido da abstração da camada de apresentação no servidor é ter uma camada de transporte robusta, simples e segura que sirva de ponte entre a camada de apresentação e a camada de lógica, que no nosso caso, são respetivamente, as componentes cliente e servidor.

Como a aplicação a desenvolver irá correr em ambiente *browser*, o protocolo a usar será naturalmente HTTP/HTTPS, que no futuro também poderá ser usado em aplicações móveis nativas, já que hoje em dia praticamente todas as aplicações móveis e não móveis têm bibliotecas HTTP para acesso de dados na internet.

Como o protocolo HTTP não serve unicamente para expor páginas de internet, podemos usá-lo para outros serviços web como, por exemplo, comunicar com Web APIs externas. Para isso,

fazemos uso da arquitetura REST, que é uma alternativa aos clássicos mecanismos de transmissão de informação, como o WSDL ou SOAP.

A escolha da arquitetura REST, em prol das acima referenciadas, prende-se com o facto de ser a tecnologia que melhor se adapta ao conceito da aplicação. Existem algumas vantagens em relação à arquitetura SOAP, no âmbito deste projeto, que é o facto de ser a arquitetura se adaptar a ambientes de aplicações web e móveis, porque nesse caso, a nível de implementação, a tecnologia SOAP é muito mais complicada de implementar, para além de a manutenção de SOAP ser mais difícil de gerir. Outra vantagem prende-se com o facto de o SOAP utilizar um grande consumo de dados relativamente ao REST. A principal desvantagem no uso de REST em relação ao SOAP é a segurança, que no caso do SOAP já faz parte da arquitetura da tecnologia, no caso do REST tem que ser tratada à parte.

5.2.2 Web Application-Programming Interface

As Web APIs são geralmente usadas como porta de acesso às funcionalidades de determinados serviços em ambiente servidor, por parte do ambiente cliente, daí ser uma interface. As três funções principais das APIs são a recolha de dados, a formatação de dados e o envio de dados.

Na figura 5.5 estão esquematizadas as três principais funções de uma Web API.

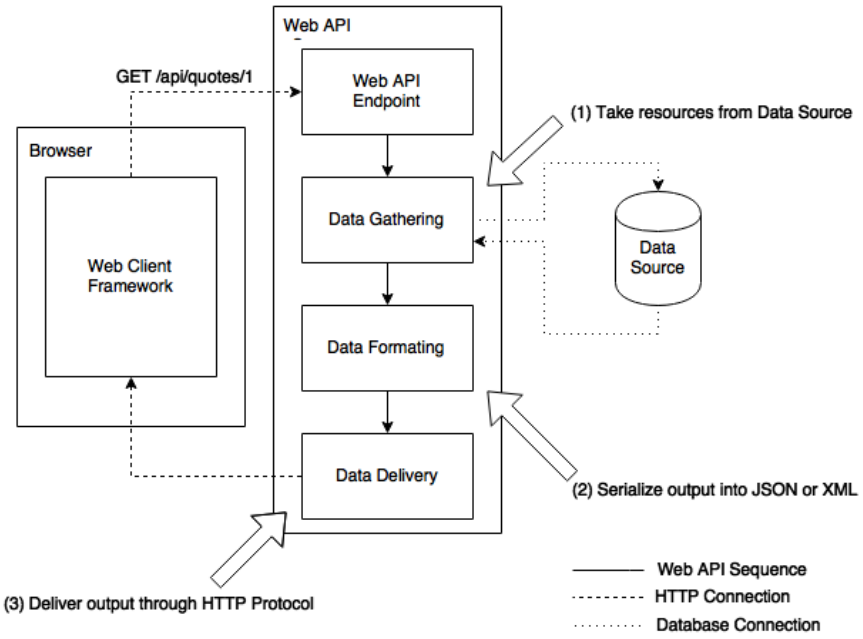

Figura 5.5: As três principais funções de uma Web API.

Para além destas três funções, existem outras funcionalidades que são a base de suporte destas três, onde se incluem, por exemplo, a autenticação, a autorização, a orquestração, o escalonamento, a consistência de dados, entre outras.

Note que a função de recolha de dados ilustrada na figura não é exclusivamente de leitura, mas também de escrita.

5.3 Segurança

Uns dos aspetos a ter em consideração, e de extrema importância, quando se trabalha com dados sensíveis, sejam pessoais ou de negócio, é a segurança dos dados. De acordo com a especificação de requisitos, existe a necessidade de conceber um sistema de autenticação e de autorização para restringir o acesso a diferentes módulos da aplicação. Para além deste aspeto, há que considerar que a aplicação será implementada para o modelo SaaS e neste contexto é imprescindível controlar quem terá acesso a determinados recursos, uma vez que os recursos de diferentes utilizadores estarão sob a mesma base de dados.

Para implementar a segurança no acesso aos recursos de negócio da aplicação, teremos em mente duas caraterísticas, a autenticação e a autorização.

- Autenticação: Consiste na verificação das credenciais no acesso à aplicação, habitualmente, através de uma identificação de utilizador e uma palavra passe. Quando a autenticação é efetuada com sucesso, o utilizador tem acesso à aplicação.

- Autorização: A autorização concede ao utilizador, previamente autenticado, o acesso a determinados recursos aos quais tem acesso. Por exemplo, se o utilizador for administrativo, tem acesso ao painel administrativo, caso contrário esse acesso seria negado.

5.3.1 Autenticação

Em aplicações web, por norma, é usado o protocolo HTTP, que é um protocolo *stateless*, o que significa que se um utilizador do protocolo se autenticar no primeiro pedido que efetuar, no segundo pedido o protocolo, por si só, não sabe quem o está a fazer.

O modo clássico para resolver este problema é criar um sistema de *cookies* de sessão, onde é criada uma sessão no servidor para o utilizador que fez a requisição da sessão e uma chave, *cookie de sessão*, que é enviado a esse utilizador. Esta chave é guardada no *browser* e enviada nos próximos pedidos, assim, não há necessidade de repetir o processo de autenticação por cada pedido efetuado ao servidor. Este envio é feito automaticamente pelo *browser*.

Atualmente, esta abordagem pode ter alguns problemas, sobretudo, porque a internet cresceu e já não é a mesma internet que existia quando esta solução foi pensada. Hoje em dia temos muitas aplicações que utilizam Web APIs onde, como vamos ver mais à frente, não é a abordagem mais indicada.

Na figura 5.6 está ilustrada um exemplo de autenticação por *cookies* de sessão.

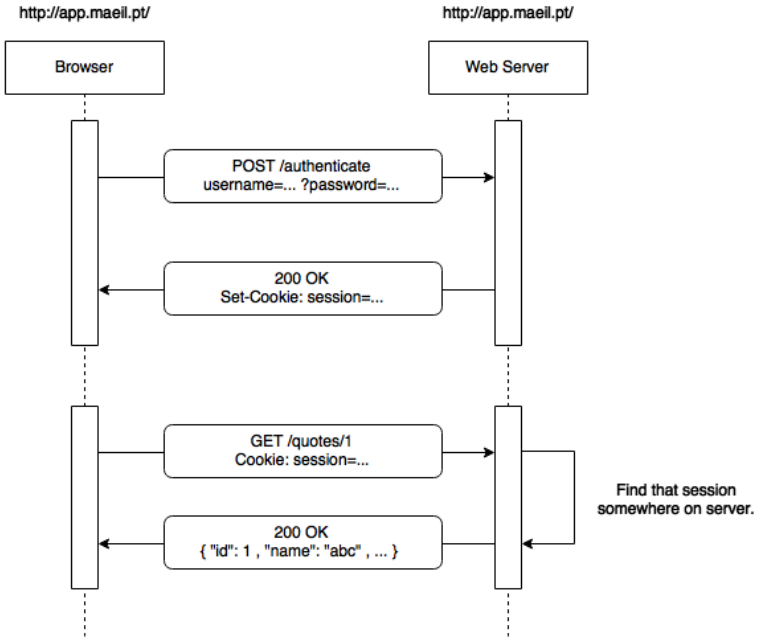

Figura 5.6: Autenticação com recurso a *cookies* de sessão.

Uma alternativa ao cenário acima descrito é o uso de *JSON Web Tokens* (JWT)[22]. Ao contrário dos *cookies* de sessão, o uso de JWT não necessita de guardar sessões no servidor. O utilizador ao enviar os dados corretos de acesso para o servidor, é-lhe devolvido um *token* que servirá de acesso aos recursos da aplicação.

Este *token* é encriptado e assinado pelo servidor emissor. No caso de o utilizador usar uma aplicação através de um *browser*, este irá guardar o *token* na sua *store* e estará disponível para a aplicação que o solicitou. Nos próximos pedidos, será enviado o *token* de acesso como cabeçalho do pedido ao servidor e este será validado como sendo autêntico, disponibilizando assim os recursos ao utilizador.

Na figura 5.7 temos uma ilustração de uma autenticação com o uso de JWT.

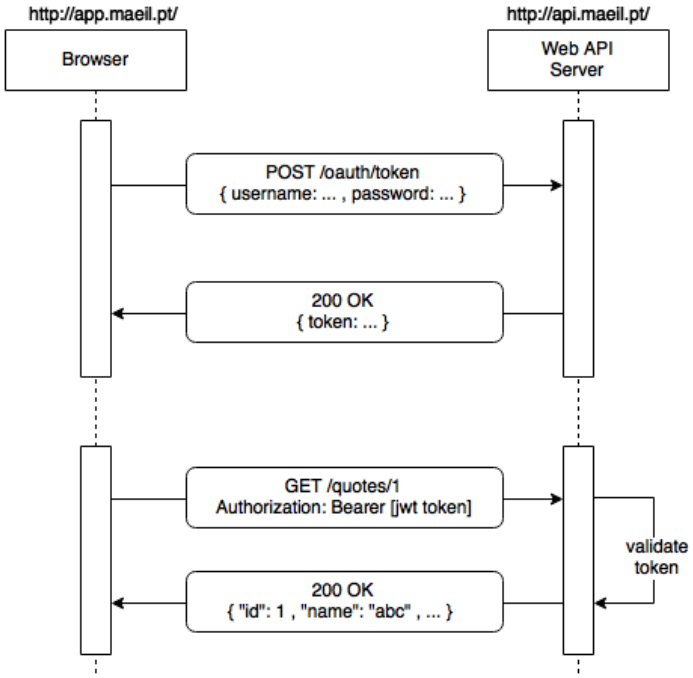

Figura 5.7: Autenticação com recurso a JWT.

Uma das características únicas do JWT diz respeito à constituição do próprio *token*. Nele estão encapsuladas três partes de informação, separadas por um ponto, o cabeçalho, o *payload* e a assinatura.

No cabeçalho segue a informação do tipo de *token* e a indicação do algoritmo de encriptação, no *payload* segue a informação que se pretende passar ao destinatário (habitualmente dados sobre o utilizador) e finalmente a assinatura, que é obtida através do cabeçalho, do *payload* e de uma chave apenas conhecida pelo servidor gerador do *token*. Esta assinatura permite ao servidor validar a autenticidade e a integridade do *token*. As três partes constituintes do token não estão encriptadas, apenas são codificadas individualmente em *base64*, quer isto dizer que de forma alguma deve ser passada informação confidencial através do *token*.

No quadro seguinte temos um *JSON Web Token* com as suas três partes (separadas por um ponto), cabeçalho, *payload* e assinatura, respetivamente.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwOi8vaHlkcmEubWFlaWwucHQiLCJzdWIiOiIxMjM0
NTY3ODkwIiwibmFtZSI6Ikh1Z28gQ29uY2Vpw6fDo28iLCJhZG1pbiI6dHJ1ZSwiZXhwIjo0MjkzODc0Mjl9.GmmAxJ
U3DNFmONxx1HbQkUsckWFgFG3e3poTGz7wJfo
```

Quadro 5.1: Exemplo de um JSON Web Token.

Descodificando o cabeçalho para texto, obtemos o tipo de algoritmo de encriptação utilizado e o tipo de *token*, neste exemplo, HS256 de HMAC-SHA256 e JWT de *JSON Web Token*.

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Quadro 5.2: Exemplo do cabeçalho de um JSON Web Token.

Descodificando a segunda parte, o *payload*, obtemos as informações que queremos transmitir ao destinatário do *token*. Nenhuma destas informações é obrigatória, mas é comum usar a identificação do utilizador e o tempo de expiração do *token*. No caso do exemplo, acrescentaram-se mais alguns, nomeadamente, o emissor do token, o indicador booleano de que confere o título de administrador ao utilizador e o nome do utilizador.

Alguns destes campos fazem parte dos padrões de implementação do JWT, no caso do exemplo, são os campos "*iss*", "*sub*" e "*exp*", que significam, *Issuer*, *Subject* e *Expiration*, respetivamente. Os restantes são campos customizados pelo responsável da emissão do *token*.

```
{
  "iss": "http://hydra.maeil.pt",
  "sub": "1234567890",
  "name": "Hugo Conceição",
  "admin": true,
  "exp": 429387429
}
```

Quadro 5.3: Exemplo do *payload* de um JSON Web Token.

Por último, temos a assinatura, uma *hash* gerada pelas duas primeiras partes e pela chave apenas conhecida pelo servidor, que neste caso era "secret".

Face ao facto de os dados que seguem no *token* não serem encriptados, deve ser usada uma ligação segura para transmissão de dados segura, recorrendo ao uso de SSL/TLS sobre o protocolo HTTP.

Algumas vantagens do uso de JWT em relação a cookies de sessão:

- CORS / *Cross-Domain*: Quando é necessário autenticar o utilizador, em domínios ou subdomínios diferentes, por exemplo quando tempos um subdomínio para a aplicação e outro para a API, no caso do uso de *cookies* pode ser problemático.

- Escalabilidade: No uso de *cookies*, é necessário manter e gerir as sessões de cada utilizador no servidor, situação que não acontece com o uso JWT, onde o JWT de acesso é apenas validado em termos de consistência de assinatura.

- Performance: Baseado no que foi descrito no ponto da escalabilidade, podemos concluir que temos um ganho de performance quando validamos apenas o JWT comparado com o fato de termos que consultar uma sessão na base de dados, usando *cookies* de sessão.

5.3.2 Autorização

A autorização é uma componente de segurança que limita o acesso a determinados recursos da aplicação, no caso deste projeto, somente a utilizadores autenticados. O exemplo mais comum de autorização em aplicações web é por exemplo, bloquear o acesso a painéis administrativos por parte de utilizadores comuns. No caso da plataforma de importação, para além de ser possível bloquear o acesso a painéis administrativos, deve aplicar-se também a limitação de acesso a qualquer tipo de painel, sendo que esta limitação deve ser configurável tendo em conta os diferentes grupos de utilizadores.

5.4 Modelo de Dados

A conceção do modelo de dados da plataforma de importação teve como base um modelo de dados fornecido pela Maeil, modelo concebido pela Maeil e usado no âmbito da exportação. O facto de existirem alguns padrões na estrutura das duas realidades, importação e exportação, possibilitou a reutilização de parte da estrutura de dados existente. Ao modelo foram então efetuadas as alterações necessárias conforme o projeto de gestão de importação exigia. Na figura seguinte, temos os relacionamentos sobre as 4 tabelas principais relativas ao negócio, Booking, Contract, Quote e QuoteRequest (figura 5.8). Os restantes relacionamentos podem ser consultados no anexo C.

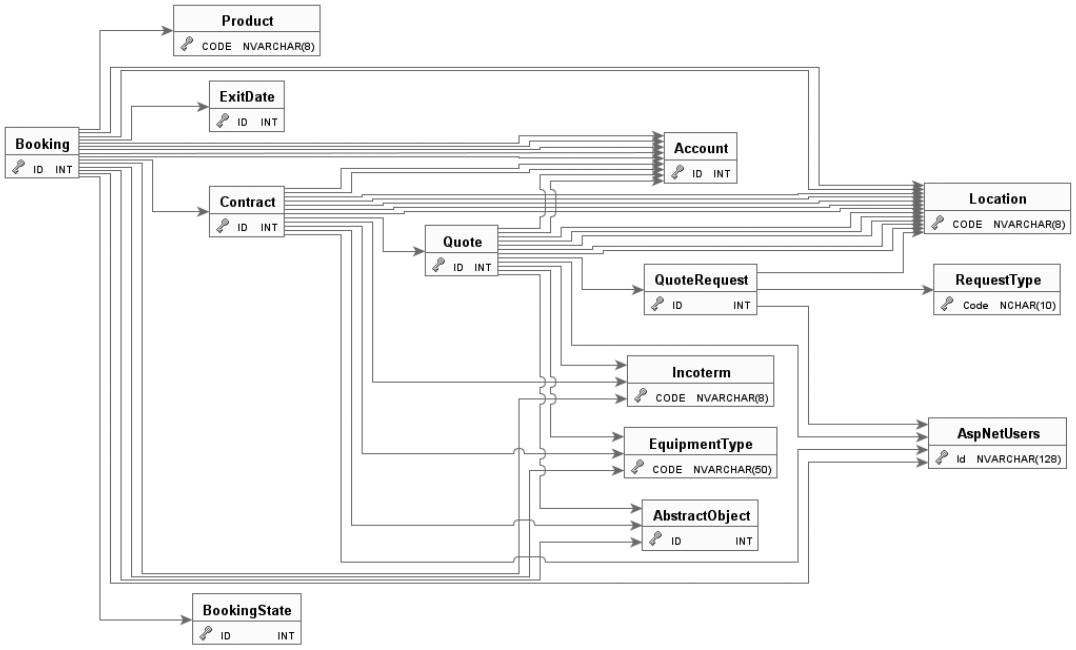

Figura 5.8: Modelo de dados parcial da plataforma de importação.

5.5 Cenários de Implementação

Foi definido internamente que umas das premissas que a arquitetura deveria respeitar a necessidade da existência de dois cenários possíveis de distribuição da plataforma. O primeiro cenário é conhecido como *Product-as-a-Service (PaaS)*, o segundo cenário é conhecido por *Software-as-a-service (SaaS)*.

Cenário 1 - PaaS

Neste cenário, a aplicação é disponibilizada no cliente, dentro da sua rede interna. Todos os dados da aplicação ficarão sob a responsabilidade do cliente, ou da entidade que fizer a gestão dos seus dados. Numa implementação deste tipo é conveniente ser usada a virtualização de sistema, de modo a facilitar a distribuição do produto.

Cenário 2 - SaaS

Neste cenário, é garantido o acesso à aplicação por parte do cliente em qualquer lugar, com a salvaguarda que os seus dados ficam em segurança. A aplicação ficará neste caso na infraestrutura da Maeil ou, por exemplo, na infraestrutura de um provedor de computação em nuvem. Com este cenário é garantido o suporte da aplicação e o cliente tem a garantia que o *software* tem sempre a atualização mais recente.

Nas figuras seguintes temos uma representação simples de ambas as infraestruturas, respetivamente.

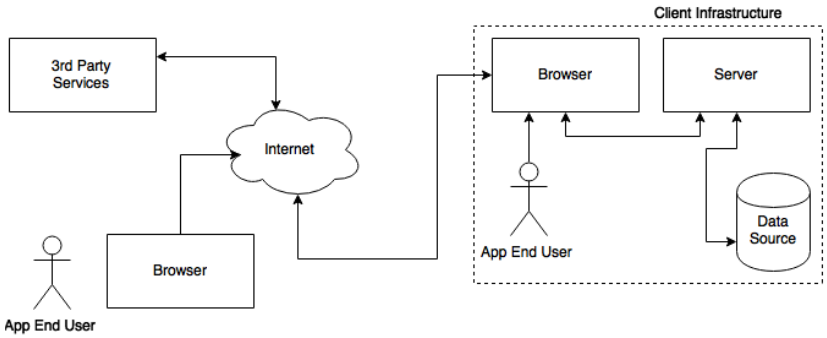

Figura 5.9: Cenário 1 - Arquitetura *Product-as-a-Service*.

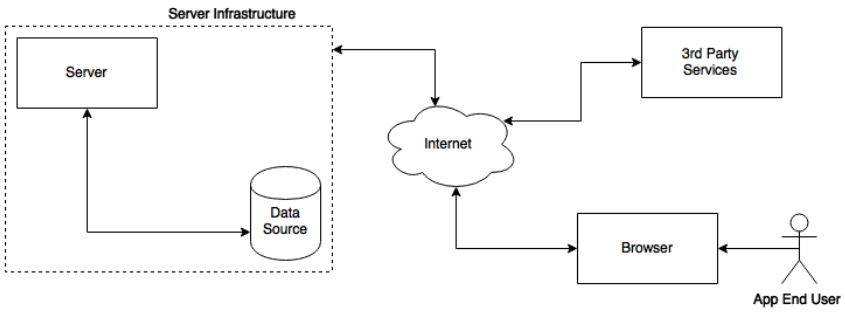

Figura 5.10: Cenário 2 - Arquitetura *Software-as-a-Service*.

5.6 Servidor de Documentos

Um dos requisitos mais comuns entre as aplicações de gestão de negócio é a geração de documentos. Habitualmente esta componente é desenvolvida dentro da própria aplicação como se fosse qualquer outro requisito de negócio, sendo um requisito repetitivo no desenvolvimento de aplicações.

No âmbito deste projeto, o autor optou por desacoplar esta componente da aplicação de modo a que pudesse ser tirado proveito por parte outras aplicações existentes ou que eventualmente pudessem a vir a ser desenvolvidas.

Deste modo, a solução passou por desenvolver um serviço web, independente da plataforma de gestão de importação, cujo objetivo seria apenas a criação e gestão de documentos. No fundo, esta componente poderá ser vista como um serviço *3rd Party*, perante a plataforma de gestão de importação ou qualquer outra aplicação.

A arquitetura do servidor de documentos poderá ser vista na figura seguinte (fig. 5.11).

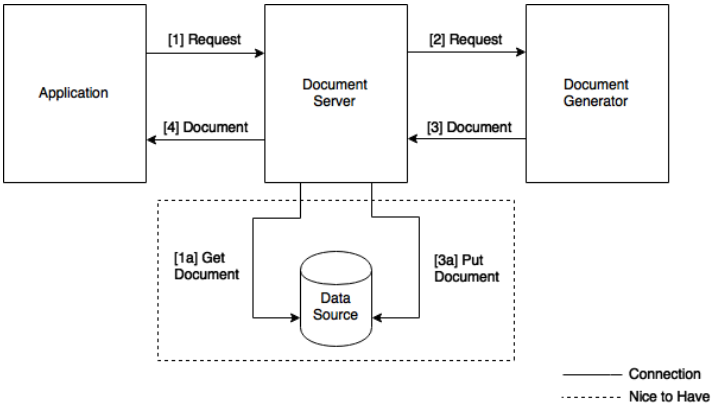

Figura 5.11: Arquitetura do servidor de documentação.

O fluxo começa com um pedido ao servidor, enviando determinados parâmetros e respetivos dados para a geração do documento. Esses dados irão ser processados consoante o tipo de ficheiro que se deseje gerar e irá retornar, em caso de sucesso, um documento como resposta ao pedido. De forma a facilitar a integração e comunicação de dados, o servidor será uma Web API, com implementação de arquitetura REST, semelhante à plataforma de gestão de importação.

A nível de questões de segurança, será implementado o mesmo sistema que a plataforma de gestão de importação, usando JWT. Neste caso, cada utilizador do servidor de documentação será autenticado como uma aplicação e não uma pessoa.

Relativamente ao tipo de documentos a gerar, foi dada prioridade à geração de ficheiros PDF. De notar a importância de que os ficheiros gerados serem baseados em *templates*, com objetivo facilitar a sua construção e potenciar a sua customização. De modo a facilitar a sua customização, os *templates* devem ser construídos com recurso a linguagem HTML e CSS, para posteriormente serem convertidos em PDF.

5.7 Camada de Apresentação

Após a decisão tomada pela renderização do contexto de *frontend* ser efetuado pela máquina cliente, o próximo passo foi escolher uma *framework* que permitisse solucionar esta questão. Dado o desconhecimento técnico deste tipo de arquitetura, conforme visto no estado da arte, procurou-se estudar e comparar cada uma das principais *frameworks* existentes no panorama atual, por forma a escolher aquela que melhor se adaptava ao projeto, tendo também em conta a mitigação de riscos associados a esta escolha.

Após a análise, optou-se pela *framework* de *javascript*, Angular JS.

Angular JS segue uma arquitetura MVC, onde temos uma estrutura bem definida de componentes e de fácil gestão. Dentro desta estrutura podemos encontrar três elementos principais: modelos ou templates, controladores e documentos estáticos.

Templates

São documentos HTML que no caso de Angular JS, contêm elementos e atributos especiais da *framework*. Mais especificamente, existem 4 tipos de elementos e atributos que podem ser usados:

- Diretivas – são atributos presentes nos elementos DOM que informam o compilador HTML de Angular JS sobre a ação lógica a tomar sobre aquele elemento.

```
<button ng-click=submit()></button>
```

Quadro 5.4: Exemplo de aplicação da diretiva Angular *ng-click*.

- Markup – é identificado pelas duplas chavetas curvas $\{\{x\}\}$ e usado para passar valores para variáveis no código HTML de uma determinada região.

```
<div class="example">{{value}}</div>
```

Quadro 5.5: Exemplo de aplicação de *markups*.

- Filtros – servem para efetuar a formatação de dados, por exemplo, para a deserialização de uma lista.

```
<span ng-repeat="city in ctrl.cities">{{city.name}} - {{city.population}}</span>
```

Quadro 5.6: Exemplo da aplicação de um filtro.

- Controladores de formulário – servem para validar formulários, permitem por exemplo, informar o utilizador de campos preenchidos incorretamente antes de este submeter a informação.

Controlador

O controlador é o pedaço de código Javascript onde é definido comportamento dos nossos objetos e dos dados. O controlador é no fundo o local onde está definida a "lógica" da nossa interface, onde por exemplo poderá ser feita a chamada a Web Services / Web API. É identificado no código HTML pela diretiva *ng-controller*.

Documentos estáticos

Os modelos estáticos não são mais que ficheiros não dinâmicos, como imagens, ficheiros css e html.

5.8 Ferramentas e Tecnologias Adotadas

Nesta secção são descritas as ferramentas e as tecnologias adotadas tendo em consideração os requisitos não funcionais indicados no capítulo 4 e a arquitetura do sistema.

5.8.1 Sistema Operativo

O servidor irá correr sobre o sistema operativo Microsoft Windows, sendo que as versões suportadas são as seguintes (e superiores):

- Windows Server 2012
- Windows Server 2008 r2
- Windows 8
- Windows 7

A escolha desta solução prende-se principalmente com o facto de a Web API ser desenvolvida em C#/.NET e assim a sua implementação estar limitada a este sistema operativo. Esta situação não é crítica, uma vez que a Microsoft é parceira da Maeil e os desenvolvimentos da Maeil serem habitualmente e preferencialmente feitos neste ambiente. Contudo, não é descartada a utilização no futuro de outro sistema operativo, como Unix, para a implementação de algum componente.

5.8.2 Servidor Web

O servidor utilizado foi IIS da Microsoft[23] e o Node.JS[24]. A utilização do IIS está relacionado com aquilo que foi mencionado no ponto 5.8.1, o uso de uma aplicação C#/.NET. Esta está limitada ao uso ao IIS. Já aplicação cliente pode ou não utilizar o IIS, no ambiente de desenvolvimento e testes foi usado um servidor web Node.JS. Dependendo do modo de distribuição do produto, poderá ser usado um CDN para alojar a aplicação cliente, isto trairia a vantagem de existirem vários servidores dedicados por forma a garantir a melhor performance, escalabilidade e disponibilidade. A principal desvantagem é que esta solução pode implicar custos elevados, mediante o trafego gerado. Esta solução seria ideal para o modelo de distribuição *SaaS*, no caso do *PaaS*, não existiria grande vantagem uma vez que que aplicação pode estar presente dentro da rede local do cliente.

5.8.3 Base de Dados

A nível de base de dados, optou-se por usar uma base de dados relacional e esta corre sobre SQL Server 2012 ou superior. Eventualmente, pode ser usada uma versão mais antiga, mas não é garantido o seu correto funcionamento. A escolha do SQL Server vem de encontro ao que foi referido nos pontos 5.8.1 e 5.8.2.

5.8.4 Linguagens

Durante a implementação do projeto foram usadas diversas linguagens de programação. No caso da componente servidora, Web API, foi usada a *framework* .NET [25] que assenta sobre a linguagem C#. Também na componente servidora, mas relativamente ao servidor de documentação foi usada a linguagem Python [26], que neste âmbito mostrou ser uma linguagem mais robusta e com melhores soluções, para além do fator tempo de implementação, que era reduzido e que contribuiu para esta escolha. Ainda no servidor de documentação podemos encontrar as linguagens HTML e CSS, para construção de *templates*.

Na componente de cliente, *frontend*, foi maioritariamente usada a linguagem *Javascript* e também HTML e CSS, para os *templates* de apresentação.

Na componente frontend, foram usadas diversas ferramentas no apoio ao desenvolvimento de javascript sobre a framework, nomeadamente a ferramenta de gestão de pacotes Bower [39] e a ferramenta de gestão de tarefas Grunt [40]. Para suporte ao desenvolvimento de código, foi usado maioritariamente o editor Brackets [41].

Capítulo 6

Implementação

Neste capítulo são descritos os procedimentos de implementação com base nos requisitos de cada componente, conforme foram estipulados no capítulo 4.

6.1 Web API

No capítulo anterior vimos que a arquitetura da camada lógica da aplicação tem como base uma Web API. Esta Web API dá acesso às várias funcionalidades disponibilizadas pela aplicação e que compõem a camada lógica. Os recursos disponibilizados pela Web API podem ser consultados no anexo D.

Existem três pilares na base da construção da Web API, os modelos, os serviços e os controladores.

6.1.1 Modelos

Os modelos não são mais do que um conjunto de classes com a caracterização dos objetos usados. Os modelos foram inicialmente criados com recurso à funcionalidade ADO.NET [27], que permite construir todos os objetos que representam as tabelas presentes numa base de dados já existente, para além disto, permite também definir automaticamente como é que estes objetos se relacionam entre si.

A definição correta destes relacionamentos e dos próprios objetos foram fundamentais para o bom funcionamento da Web API, uma vez que a validação de entrada de dados e a serialização de resposta são baseadas nestes modelos. Por essa razão, foi necessário corrigir alguns aspetos nos modelos, uma vez que a geração necessita quase sempre de afinações, seja a nível de atributos de campos, seja a nível de correções de referências entre tabelas. Um dos exemplos que, por desconhecimento, gerou mais dificuldade, foi a eliminação ciclos de referência entre modelos. Os ciclos de referência acontecem, por exemplo, quando duas tabelas se referem mutuamente e por consequência da geração automática de objetos essa situação pode refletir-se numa ramificação infinita, aquando a serialização de resposta no acesso aos dados. A eliminação deste ciclo, por ser feita definindo na classe de referências de tabelas da base de dados quem é a tabela principal.

6.1.2 Serviços

Os serviços são as classes que tratam a parte da lógica de negócio, incluem cálculos de negócio, acessos à base de dados, acessos a componentes externas, etc. As classes de cada serviço são acedidas pelos controladores que serão explicados no ponto seguinte.

Relativamente aos acessos à base de dados, estes são feitos através da tecnologia LINQ[28] da Microsoft, que assenta sobre C#, cujo objetivo é integrar e facilitar chamadas SQL na linguagem C#. Para além destas duas vantagens, podem ser prevenidos ataques via SQL Injection[4] à aplicação com o uso deste tipo de tecnologia.

Tome-se o exemplo, no quadro 6.1 e 6.2, onde podemos ver, respetivamente, uma chamada à base de dados usando uma *query* LINQ e a *query* SQL resultante da *query* LINQ, obtida através do *SQL Profiler*, ferramenta presente no *SQL Server* para análise de acessos à base de dados.

```
db.Quotes.Where(q => q.USERID.Equals(userid));
```

Quadro 6.1: Exemplo de uma *query* LINQ.

```
SELECT
        (...)
FROM [dbo].[Quote] AS [Extent1]
WHERE [Extent1].[USERID] = @p__linq__0
```

Quadro 6.2: Exemplo de uma *query* SQL, obtida com base no quadro 6.1.

6.1.3 Controladores

Os controladores são as classes que definem os acessos externos à Web API, é nestas classes que controlamos os dados que entram e saem do servidor. Para isso ser possível, seguindo uma arquitetura REST, são definidos para cada função, os tipos de métodos HTTP de acesso suportados pela Web API, GET, POST, PUT e DELETE. A definição dos métodos é feito definindo o prefixo da função de chamada, com o nome do respetivo método. Note o exemplo do quadro 6.3.

```
public async Task<IHttpActionResult> PostQuote(int id, Quote quote)
{
...
}
```

Quadro 6.3: Exemplo de definição do método HTTP POST.

Nos controladores são também definidos os *routings* de acesso à Web API, ou seja, o caminho URL de cada método disponibiliza. Este *routing* pode ser feito adicionando um atributo à classe e/ou à função pretendida. Note o exemplo do quadro 6.4, onde se define o atributo correspondente ao *routing* de acesso às funções do controlador de cotações, através dos atributos *RoutePrefix* e *Route*.

[4] SQL Injection - tipo de ameaça de segurança que se aproveita de falhas em sistemas que se ligam com bases de dados via comandos SQL.

```
[RoutePrefix("api/quotes")]
public class QuotesController : ApiController
{
...
    // GET: api/quotes/{id}
    [Route("{id}")]
    public async Task<IHttpActionResult> GetQuote(int id)
    {
    ...
    }
...
}
```

Quadro 6.4: Exemplo de definição de *routings* / url de acesso

Cada função pode ter um tipo de resposta diferente, podemos querer obter uma cotação, uma marcação ou outro tipo de objeto diferente. Para isso, temos que definir, para cada função do controlador o seu tipo de resposta como atributo e não como habitualmente se faz, como definição da função. Isto dependendo do tipo de resposta que pretendemos dar ao pedido.

Nos atributos do controlador, ou das funções do controlador, podemos também inserir outras anotações, sendo que a mais usada para além dos referenciados anteriormente foram as definições de permissões, que iremos abordar na secção relativa à implementação da segurança.

6.2 Módulos da Plataforma

Nesta secção é descrita a implementação e respetiva tomada de decisão para cada um dos módulos implementados que compõe a plataforma de gestão de importação.

6.2.1 Módulo de Ofertas

O módulo de ofertas gere as ofertas de cotações de transporte. Para o utilizador comum, o importador, é constituído por 2 painéis, um onde este pode enviar pedidos de cotação a transportadores e outro onde são listadas as cotações recebidas. Para os transportadores, existe um painel neste módulo onde estes podem submeter as suas ofertas de cotação e outro para alterar as cotações submetidas.

Na figura 6.1, podemos ver o painel de pedidos de cotações implementado, onde no canto inferior direito o utilizador pode adicionar um pedido de cotação.

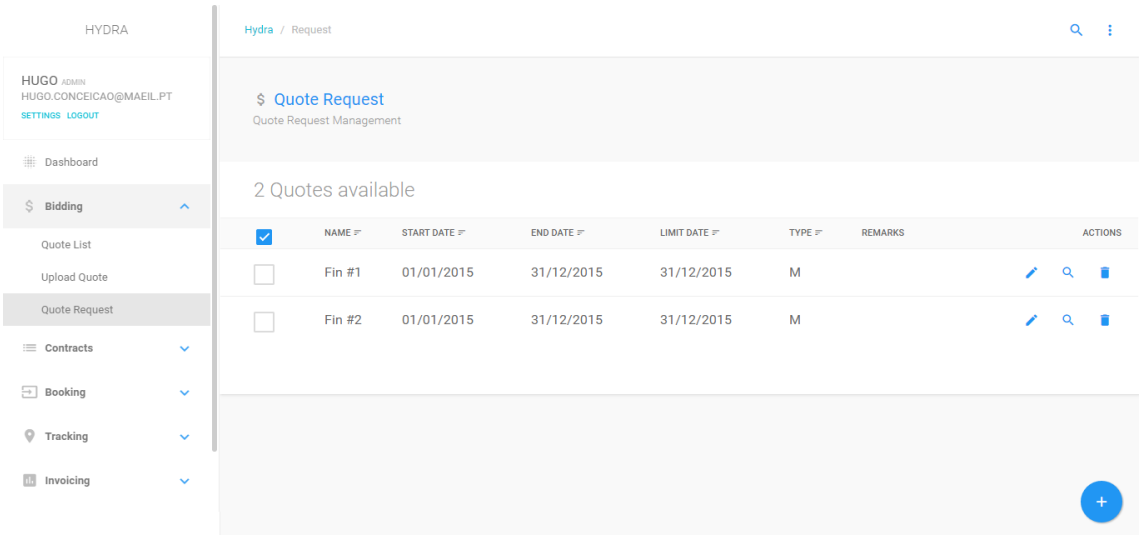

Figura 6.1: Painel de listagem de pedidos de cotação implementado.

Na figura 6.2, temos o painel de listagem de cotações recebidas, onde ao selecionar uma cotação surge o ícone que permite a conversão de cotação em contrato.

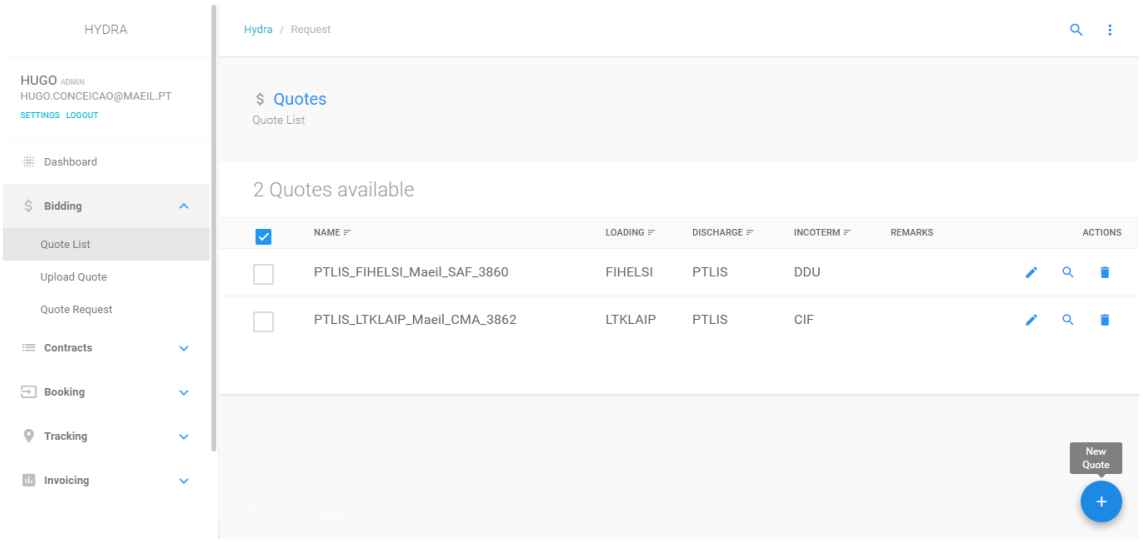

Figura 6.2: Painel de listagem de cotações implementado.

Na figura 6.3, temos a ilustração no painel de submissão de cotações, que deverá estar disponível apenas para transportadores.

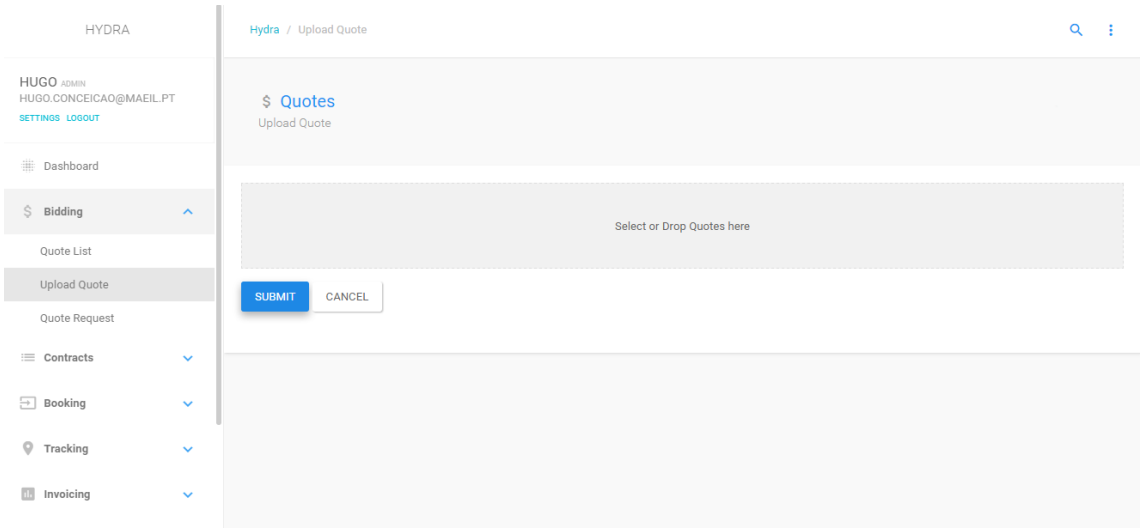

Figura 6.3: Painel de submissão de cotações implementado.

6.2.2 Módulo de Contratos

O módulo de contratos permite a gestão de contratos. É um painel simples, onde são listados os contratos criados a partir das cotações. Ao selecionar um contrato podemos criar uma marcação de transporte, clicando sobre o ícone destinado a esse efeito. Fazendo duplo clique sobre o contrato, podemos ver as informações do mesmo.

Na figura 6.4 temos ilustrado o painel de contratos implementado.

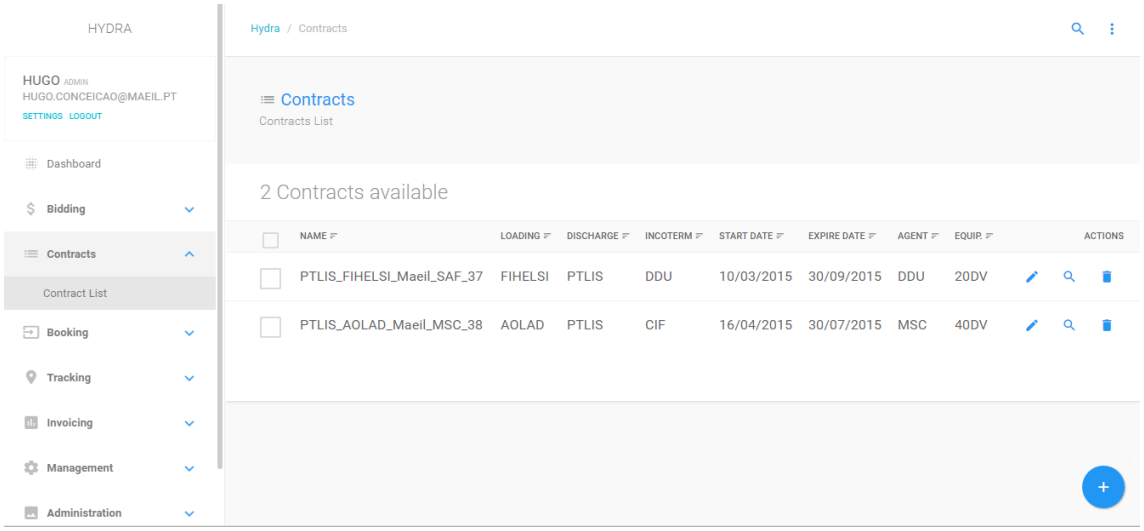

Figura 6.4: Painel de contratos implementado.

6.2.3 Módulo de Marcações

O módulo de marcações é constituído por um painel com a listagem de marcações. Sobre as marcações podemos completar as informações da marcação, com os detalhes que não são definidos nos módulos anteriores.

Na figura 6.5 temos ilustrado o painel de marcações implementado.

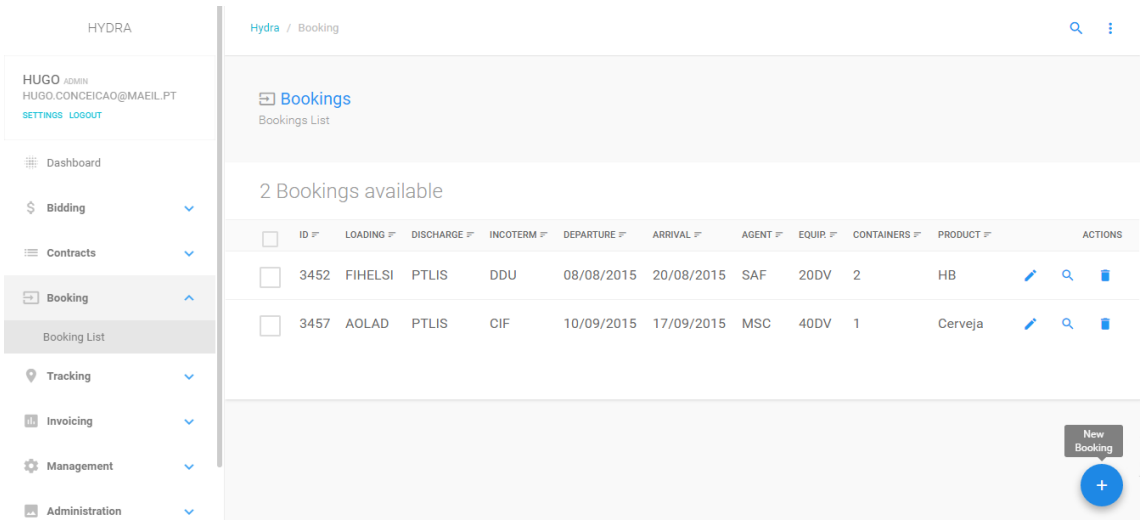

Figura 6.5: Painel de marcações implementado.

6.2.4 Módulo de Monotorização de Mercadoria

O módulo de monotorização de mercadoria permite ao utilizador saber em que localização se encontra a sua mercadoria. A pesquisa da mercadoria é feita com base na marcação, o utilizador insere o número da marcação e é-lhe apresentado o mapa global, com a indicação da posição da carga.

Para a implementação do mapa foi utilizada API do Google Maps [29]. As informações da posição da carga foram obtidas através da API Marine Traffic [30], que fornece informações sobre a posição e rotas de navios. Com base na informação da marcação que contém a informação do navio e datas das viagens, é possível pedir informações sobre a posição do navio.

Na figura 6.6 podemos ver como é feito o processo de recolha de informação da posição do navio de mercadorias.

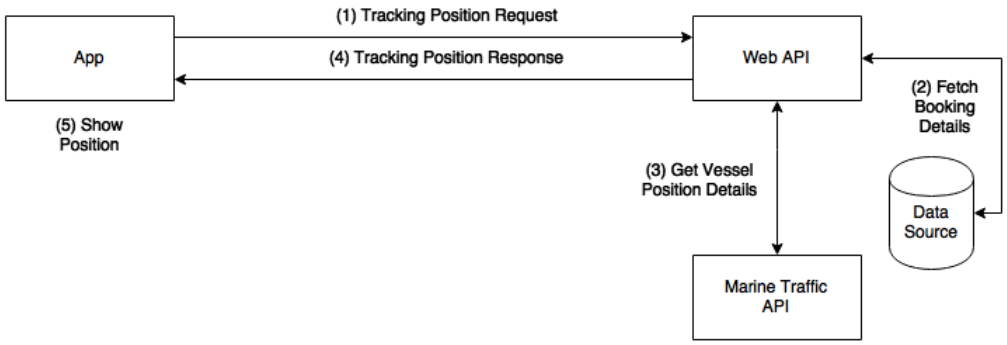

Figura 6.6: Pedido de posição de mercadoria.

O utilizador ao requerer a posição do navio, a aplicação envia para a Web API um pedido de *tracking* para uma determinada marcação(1) e de seguida a API efetua a pesquisa da marcação e recolher as informações da mesma na base de dados(2). Após ter os dados da marcação, irá proceder ao pedido da posição da mercadoria(3), caso a data do pedido esteja dentro do período das datas de transporte, data de partida e data de chegada no navio, que constam nos detalhes da marcação. Só assim é garantido que a posição fornecida, corresponde à viagem que transporta a carga em questão. Por fim, é enviado à aplicação a resposta ao pedido(4), com a posição no navio e é mostrado o resultado ao utilizador(5).

Na figura 6.7, temos ilustrado o módulo de monotorização de carga implementado.

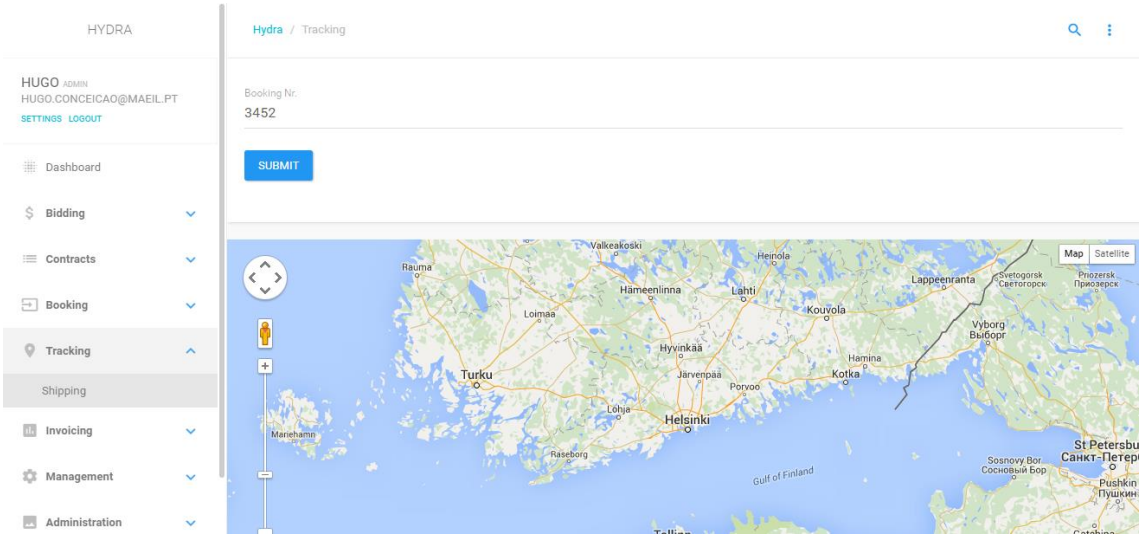

Figura 6.7: Módulo de monotorização de mercadoria implementado.

6.2.5 *BackOffice*

O BackOffice permite fazer a gestão de utilizadores e permissões dos mesmos na aplicação. Foram criados dois painéis que permitem fazer esta gestão. Temos então um painel onde o administrador da aplicação pode registar ou desativar utilizadores, e outro painel onde o administrador pode conceder permissões a utilizadores a nível de módulos.

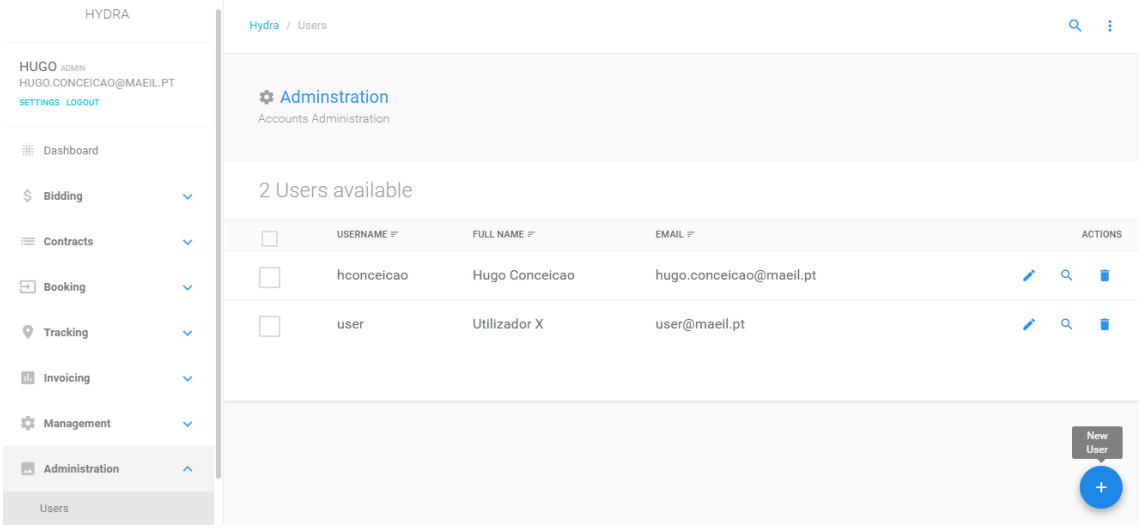

Figura 6.8: Painel BackOffice implementado.

6.2.6 Configurações

O módulo de configurações permite fazer a gestão de dados estáticos de negócio, ou seja, permite gerir aqueles dados que por norma não sofrem grandes alterações ao longo do tempo. Por exemplo, códigos de moeda, de *Incoterms*, de localizações ou de produtos. São códigos ocasionalmente alterados e este módulo permite efetuar essa gestão.

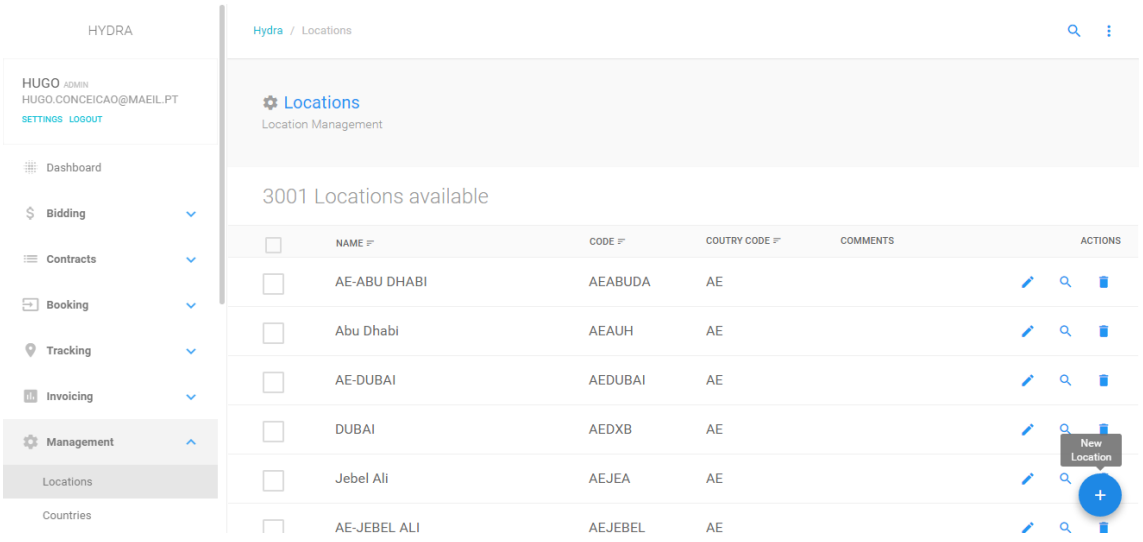

Figura 6.9: Painel de Configurações implementado, neste caso das localizações.

6.2.7 *Dashboard*

O Dashboard é um painel que resume os dados relevantes da aplicação, este contém instrumentos para uma rápida análise de negócio. Assim, foram criados alguns indicadores de negócio representados sob a forma de gráficos e tabelas. Estes indicadores são adquiridos via Web API, tal como acontece nos restantes painéis.

Na figura 6.10 temos ilustrado o painel *Dashboard* implementado.

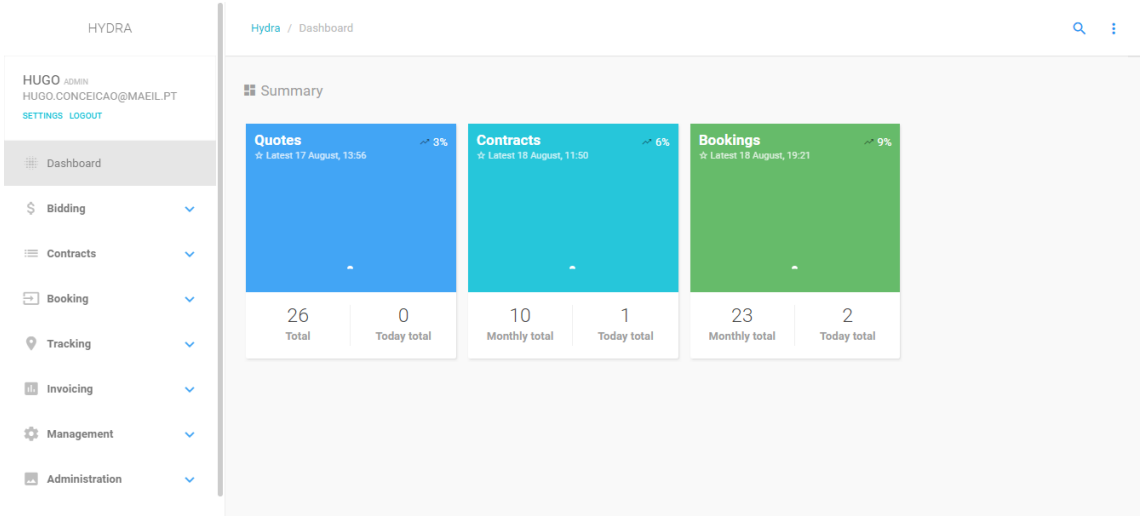

Figura 6.10: Painel de *Dashboard* implementado.

6.2.8 Documentação

No que diz respeito à documentação necessária, foram criados os *templates* referidos nos requisitos de documentação. Os *templates* foram feitos com recurso à linguagem HTML e CSS. Estes podem ser visualizados no Anexo E.

6.3 Segurança

A nível de segurança da aplicação, no que diz respeito ao acesso aos dados da web API, foi implementado um sistema de autenticação com recurso a JSON Web Tokens, descrito na secção 5.3 do capítulo de Arquitetura. Os JSON Web Tokens não são o tipo de tokens de autenticação utilizados por defeito pela Web API, portanto, foi necessário implementar esta funcionalidade.

Uma das vantagens aqui, é que o uso dos JSON Web Tokens funcionam com diferentes linguagens de programação, tais como, C#/.NET, Python, Java, Ruby, PHP, Javascript, Go, entre outras. Portanto, podem ser usados nos mais variados cenários de implementação, no caso deste projeto, foram usadas as linguagens C#, Javascript e Python. Sendo que os casos de Javascript e Python serviram apenas para interpretação de tokens e não de geração.

No que diz respeito ao controlo de acesso aos métodos da Web API, foi necessário definir quais os controladores que exigiam autenticação. Neste caso, todos os controladores estão protegidos, com exceção da função de validação de *login*, que é naturalmente pública. Esta classificação de acesso é feita adicionando o atributo *Authorize* à classe ou método no caso de esta exigir um utilizador autenticado ou *AllowAnonymous* no caso de querermos que um determinado método seja público. Este atributo garante assim a autenticidade do utilizador. Note o exemplo no quadro 6.5.

```
[Authorize]
public class ...
```

Quadro 6.5: Apenas utilizadores autenticados terão acesso.

```
[AllowAnonymous]
public class ...
```

Quadro 6.6: Qualquer utilizador terá acesso, usado por exemplo, no acesso ao login.

No que diz respeito à autorização, lembrando que serve para proteger determinados recursos de utilizadores autenticados, existem duas situações. Uma delas é a definição de grupo de utilizadores e a outra é a proteção por utilizador. A funcionalidade que trata grupos de utilizadores já existe por defeito na Web API se optarmos por usar ASP.NET *Identity* [88] como método de autenticação. Assim sendo, corretamente configurado o uso de JWT com sistema ASP.NET *Identity*, podemos simplesmente colocar o atributo que define os grupos autorizados nas classes ou métodos em questão. Note o exemplo no quadro 6.7.

```
[Authorize(Roles="Admin”)]
public class ...
```

Quadro 6.7: Exemplo de atribuição de restrição ao grupo “Admin”.

No caso de autorização de acesso a recursos privados por parte de utilizadores é sempre identificado o utilizador que está a fazer o pedido e apenas são mostrados os recursos que pertencem a esse utilizador com base numa cláusula condicional aquando a recolha dos dados da base de dados.

No que diz respeito ao *frontend*, primeiramente foi necessário implementar um ecrã de login para os utilizadores se autenticarem perante o servidor. De seguida procedeu-se à implementação da validação do *token* de acesso, baseada na data expiração do *token*, bem como na definição das regras de *routing* dentro da aplicação. Para além disso, foi implementada a funcionalidade que permite que a aplicação envie, em qualquer pedido que faça ao servidor, o cabeçalho HTTP com o *token* de acesso à Web API. Relembrando que o token é guardado na *store* do *browser* onde corre a aplicação.

No quadro 6.8, temos a aplicação do objeto *jwtInterceptorProvider,* que integra a biblioteca *angular-jwt.*

```
.config(function Config($httpProvider, jwtInterceptorProvider) {

    jwtInterceptorProvider.tokenGetter = ['store', function (store) {
        return store.get('jwt');
    }];

    $httpProvider.interceptors.push('jwtInterceptor');
}
```

Quadro 6.8: Aplicação do objeto *jwtInterceptorInterceptor* na directiva httpProvider.

Na figura 6.11 temos o ecrã de login implementado.

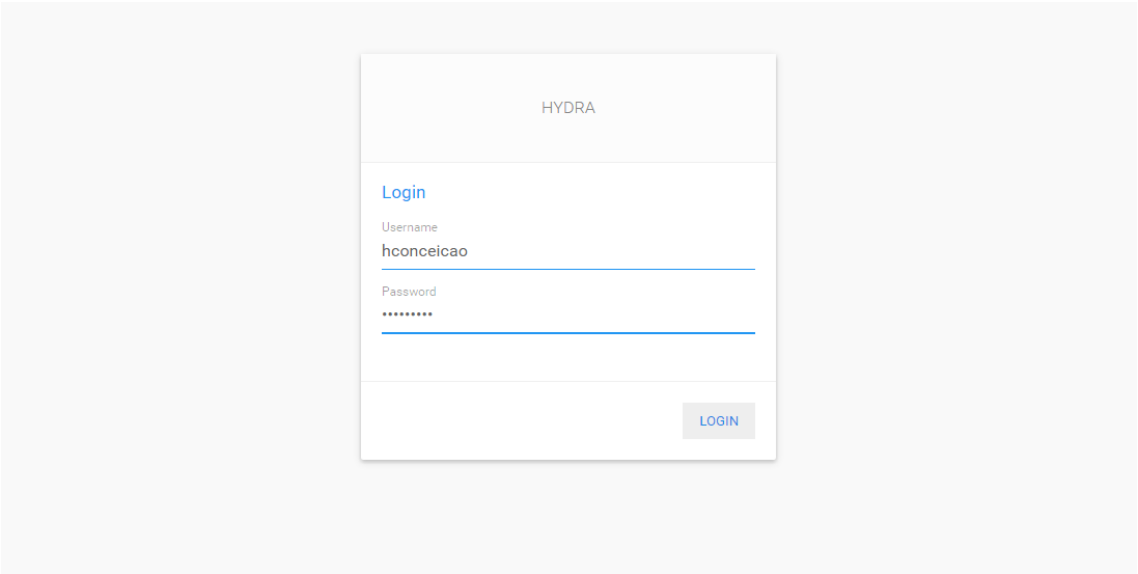

Figura 6.11: Ecrã de login implementado.

6.4 Servidor de Documentação

O servidor de documentação assume, perante a plataforma, o papel de aplicação *3rd Party*, conforme documentado no capítulo de arquitetura. Assim sendo, foi implementado um servidor dedicado somente à geração de documentos, sendo que no futuro poderá ser extensível à gestão dos respetivos documentos, podendo por exemplo serem guardados numa base de dados. No âmbito do projeto apenas foi requisitado que gerasse ficheiros em formato PDF.

O servidor foi implementado com recurso à linguagem Python, usando o módulo Flask [31] para a implementação da API. Foi escolhida esta linguagem, porque era a que oferecia as melhores ferramentas pela implementar o que tinha sido definido nos requisitos e também porque é uma linguagem conhecida do autor e que permitiu desenvolver esta ferramenta num curto espaço de tempo.

Uma vez desenvolvida a API, uma interface simples que permite o envio e obtenção de informação via protocolo HTTP, foi necessário implementar a parte de geração de documentos PDF através de *templates html*. Estes *templates* são previamente customizados a nível de apresentação do

documento, utilizando a linguagem HTML e CSS. O uso destas linguagens potencia a qualidade do documento a nível de apresentação e permite alterar rapidamente o aspeto do documento.

Após receber o pedido e respetivos dados, a API faz o mapeamento desses dados recebidos do utilizador e irá colocá-los no ficheiro HTML, que será convertido em PDF e por fim retornado ao utilizador. O mapeamento dos *templates* foi efetuado usando a biblioteca Jinja2 [32] e a conversão do ficheiro HTML foi possível com recurso à biblioteca xhtml2pdf [33].

6.5 Requisitos Não-Funcionais

Nesta secção é descrita a abordagem de implementação dos requisitos não-funcionais definidos no capítulo 4.

6.5.1 Confiabilidade

A confiabilidade é um requisito de qualidade que a plataforma deve oferecer aos utilizadores, neste caso abrange a capacidade de o sistema recuperar de erros inesperados. Os erros mais comuns em aplicações é o browser deixar de responder e que normalmente age por si para tentar resolver o problema. Poderá também ocorrer falhas a nível de JavaScript sendo que o utilizador é informado dessa situação pelo browser.

6.5.2 Usabilidade

No que à usabilidade diz respeito, foram consideradas as características definidas na norma ISO 9126 (2001) [34] para a qualidade de *software*, onde na usabilidade encontramos os seguintes atributos:

- Inteligibilidade – Procurou-se desenvolver uma interface que oferecesse ao utilizador uma rápida compreensão sobre as funcionalidades existentes, por exemplo, com o uso de legendas informativas sobre vários ícones presentes na aplicação.

- Apreensibilidade – Representa a capacidade de aprendizagem do utilizador na aplicação e neste aspeto procurou-se implementar uma interface intuitiva que permitisse ao utilizador, independentemente do seu nível de conhecimento tecnológico, aprender a utilizar a aplicação de forma simples.

- Operacionalidade – Neste aspeto, durante a aplicação são dadas indicações sobre tarefas em execução, de forma a haja uma boa operabilidade entre o utilizador e a aplicação.

- Atratividade – Procurou-se apresentar uma interface apelativa e atrativa ao utilizador, esta foi baseada nas orientações de design elaboradas pela Google [35]. Estas seguem os princípios do design clássico com as mais recentes inovações.

6.5.3 Desempenho

A nível de desempenho foram considerados dois aspetos no desempenho da aplicação, a escalabilidade e a disponibilidade dos serviços. De seguida são descritas as considerações acerca destes dois pontos.

Escalabilidade

A escalabilidade é um requisito de qualidade essencial para uma aplicação web, porque o número de utilizadores pode aumentar exponencialmente e a arquitetura de uma aplicação tem que estar preparada para lidar com esse crescimento. Embora neste caso, o número de utilizadores da plataforma seja de alguma forma controlável, este aspeto foi tido em consideração para um eventual crescimento no futuro.

No caso da Web API, que é o motor do negócio, foram utilizadas *threads* e chamadas assíncronas no tratamento de pedidos externos. O número de máximo de *threads* a correr simultaneamente pode ser configurável no servidor web, bem como o número máximo de pedidos em fila de espera. Todas estas configurações dependem do *hardware* e assim podemos dizer que uma aplicação é escalável se o aumento de recursos de hardware permitir aumentar o número de acessos simultâneos, que neste caso se verifica.

No caso da aplicação cliente, não é um aspeto tão crítico como o da Web API, uma vez que os pedidos são drasticamente menores, relembrando que por estarmos a falar de uma SPA, os recursos principais da aplicação são recolhidos apenas uma vez, na altura em que acedemos à aplicação. Contudo, mesmo a este nível é necessário garantir a escalabilidade da aplicação, onde poderemos alojar a aplicação num CDN e assim resolver este possível problema.

Disponibilidade

A disponibilidade da aplicação é um importante requisito de qualidade e que foi tido em consideração na arquitetura da aplicação. A Web API pode correr em vários servidores simultaneamente, sem que exista qualquer efeito negativo no contexto do negócio, isto permite a que no caso de algum servidor por alguma razão não der resposta, poderá ser usado um segundo servidor. Apesar de a solução estar preparada para este cenário, esta não foi implementada, sendo que para isso seria necessário usar o módulo Application Request Routing [36] do IIS. A nível de aplicação cliente, poderá ser usada a mesma solução, ou então, poderemos colocar a aplicação num CDN que em grande parte dos casos garante 100% disponibilidade do conteúdo da aplicação.

6.5.4 Portabilidade

Sendo a plataforma uma aplicação web esta tarefa foi facilitada, sendo esta à partida uma aplicação multiplataforma em que a sua única limitação neste aspeto é o *browser* utilizado pelo cliente. Deste modo, procuraram-se utilizar versões de ferramentas, bibliotecas e frameworks que não limitassem demasiado os browsers a utilizar e assim a aplicação é funcional nos principais browsers usados no mercado atual.

6.5.5 Reusabilidade

Ao longo do projeto foi tido em conta o conceito de reusabilidade, fazendo uma separação de funcionalidade genéricas, que podem ser uteis para futuros desenvolvimentos. A criação do servidor de documentação é também um exemplo da reutilização de código, sendo que neste caso, o conceito é a nível aplicacional e não a nível de código.

6.5.2 Modularidade

Conforme foi possível avaliar no capítulo de arquitetura, a aplicação foi desenhada de modo a seguir os padrões de modularidade, existindo uma evidente separação de funcionalidades na aplicação, onde cada funcionalidade representa um módulo. Esta modularidade é percecionada também no código da aplicação, onde por exemplo, temos um controlador na Web API por cada módulo da aplicação, bem como um conjunto de controladores e modelos Angular também por cada módulo.

6.5.7 Versionamento

Durante o desenvolvimento da aplicação foi usada a plataforma GitHub para controlo de versões. Esta ferramenta tornou-se vital para a boa gestão do processo de desenvolvimento. A escolha desta ferramenta prende-se com facto de ser uma ferramenta conhecida do autor e que oferece todas as necessidades para aquilo que se pretendia.

Capítulo 7

Validação e Testes

Este capítulo aborda a fase de validação e testes efetuados aos vários módulos da aplicação desenvolvida, quer a nível da componente servidora, quer a nível da componente cliente. Este processo de teste é importante na medida em que é possível validar possíveis falhas na plataforma e posteriormente corrigi-las. Neste sentido foram efetuados testes sobre os requisitos funcionais e requisitos não funcionais.

Foram então efetuados 3 tipos de testes: unitários, usabilidade e desempenho. Uma ressalva para o facto de os testes apenas terem sido realizados na reta final do estágio, após o término da implementação, e não durante o final da implementação de cada módulo, como seria a forma mais correta. Assim, não foi possível proceder à análise e correção das falhas encontradas, sendo que é um ponto a ter em consideração no futuro. Os resultados podem ser consultados no anexo F.

7.1 Testes Unitários

Os testes unitários tem como objetivo detetar falhas em blocos de código de um desenvolvimento, habitualmente aplicado em funções ou métodos. Os testes unitários devem ser automatizados, pois só assim se torna escalável essa tarefa de verificação de falhas, pois realizá-los manualmente seria uma tarefa extremamente exaustiva. Na aplicação de testes unitários, utilizando o exemplo de uma função, queremos que um determinado conjunto de entrada de dados, produza o resultado esperado à saída. Caso não aconteça temos uma falha nesse bloco analisado.

Para o caso deste projeto e devido à escassez de tempo, a realização de testes unitários a nível de *frontend* da aplicação não foi efetuado, sendo que apenas foram aplicados parcialmente na componente *backend*, que assume um papel mais preponderante na lógica da aplicação.

7.2 Testes de Usabilidade

Os testes de usabilidade dizem respeito à interação do utilizador com a aplicação, ou seja, têm como objetivo avaliar a interação dos utilizadores e o impacto que esta tem sobre eles. Neste sentido, foi delineado um guião com algumas tarefas a realizar pelos utilizadores de e no final tarefa foram registadas as avaliações destes.

7.3 Testes de Desempenho

Os testes de desempenho dividiram-se em duas partes: desempenho da componente *backend* e desempenho relativo à componente *frontend*. No caso da componente *backend*, o teste do desempenho foi medido através do número de pedidos efetuados à API da plataforma, sendo registadas as falhas e o tempo de resposta para os diversos *endpoints*. Para este teste foi usada a ferramenta Locust [37]. No caso da componente *frontend*, foi medido o desempenho da página no browser utilizando a ferramenta Google Insights [38].

7.4 Validação de Requisitos

Por forma analisar o trabalho concretizado e o que ficou por concretizar do que estava planeado, após o fecho da implementação e dos testes, procedeu-se à análise e verificação dos requisitos estipulados no capítulo 4, relativamente ao trabalho que foi realizado. Neste seguimento, esta análise foi dividida pelos dois tipos de requisitos, funcionais e não funcionais.

7.4.1 Requisitos Funcionais

Relativamente aos requisitos funcionais, consta no seguinte quadro a sua designação, o grau de implementação e o respetivo resultado.

Requisito Funcional	Grau de Implementação	Resultado
Módulo de Negócio	N	-
Módulo de Ofertas	I	P
Módulo de Contratos	I	P
Módulo de Marcações	P	F
Módulo de Monotorização	I	P
Módulo de Alfândega	N	-
Módulo de Logística	N	-
Módulo de Faturação	N	-
BackOffice	I	
Configuração	P	P
Dashboard	P	P
Módulo de Documentação	I	P

Quadro 7.1: Validação de requisitos funcionais.

Legenda:
N – Não implementado
I – Implementado
P – Parcialmente implementado
P – Passou
F- Falhou

Todos os módulos com prioridade mais elevada foram implementados, os restantes não foram executados devido à extensão do projeto em função do tempo disponível. O módulo de marcações foi assinalado como parcialmente implementado devido à falta de uma componente, nomeadamente detalhes de mercadoria. No caso das configurações, parte dos objetos configuráveis, não foram tratados na componente *frontend*, sendo que não eram cruciais para o protótipo da

plataforma. No caso do *backoffice,* ficou por tratar também sobre a componente *frontend*, nomeadamente a gestão de permissões de utilizadores.

7.4.2 Requisitos Não Funcionais

Para além da validação de requisitos da aplicação, procedeu-se também à análise de requisitos não funcionais de modo a validar a qualidade do *software*. Posto isto, procedeu-se ao mesmo tipo de análise conforme efetuado no ponto 7.4.1.

No quadro 7.2, temos o quadro de validação aos requisitos não funcionais.

Requisito Não Funcional	Grau de Implementação	Resultado
Segurança	P	P
Confiabilidade	I	P
Usabilidade	I	P
Desempenho	I	P
Disponibilidade	P	P
Portabilidade	I	P
Modularidade	I	P
Reusabilidade	I	P
Versionamento	I	P

Quadro 7.2: Validação de requisitos não funcionais.

Legenda:
N – Não implementado
I – Implementado
P – Parcialmente implementado
P – Passou
F- Falhou

Todos os requisitos não funcionais foram tidos em consideração, sendo que há diversas validações e melhoramentos que podem e devem ser efetuados. No caso da Segurança e Disponibilidade, foram assinalados como parcialmente implementados. Isto aconteceu devido a pormenores não aplicados e validados, como no caso da segurança do uso de uma ligação HTTP segura, por necessidade de aquisição de certificado, que numa fase de desenvolvimento não seria crucial. No caso da disponibilidade, faltou aplicar e testar o uso de um sistema de redundância, por forma a garantir a disponibilidade do serviço, caso o servidor tivesse alguma falha.

Capítulo 8

Conclusão

Neste capítulo é descrita uma reflexão do autor sobre o trabalho realizado durante o estágio e também é feita uma análise sobre o futuro da plataforma de gestão de importação.

8.1 Reflexão Final

Ao longo deste último ano letivo, foi-me proporcionada a tarefa de arquitetar e conceber o protótipo de uma plataforma de gestão de importação. Penso que, com o trabalho desenvolvido até ao momento, a Maeil poderá tirar bom partido do trabalho que foi efetuado, porque caso pretenda dar continuidade ao projeto, existe uma base sólida para o fazer. O facto de ter apresentado uma nova visão tecnológica à empresa poderá abrir novos horizontes no futuro, podendo ser aplicada a futuros projetos.

Relativamente ao projeto em si e excluindo o facto de no primeiro semestre a minha disponibilidade não ter estado de acordo com aquilo que era esperado, penso que este era um projeto bastante ambicioso e extenso em termos de requisitos, daí a necessidade de ter de descartar algumas funcionalidades para que fosse possível apresentar um produto funcional à Maeil no final do estágio.

Em termos de estágio, a nível pessoal só há a retirar aspetos positivos, a começar pela minha inserção num ambiente de *software house* de um mercado muito específico, de Transportes e Logística, que me permitiu ganhar bastantes competências dentro deste meio. O uso de novas tecnologias e filosofias de desenvolvimento de produto deu-me novas competências que pretendo continuar a explorar no futuro.

8.2 Trabalho Futuro

Concluída esta etapa, cabe à Maeil a decisão dar continuidade ao projeto. Como referido anteriormente, penso que o projeto já possui uma arquitetura sólida e um desenvolvimento modular de diversas funcionalidades que irão permitir concluir os requisitos não concretizados de uma forma progressiva. De qualquer forma, ficam neste ponto registados alguns aspetos que considero interessantes e que devem ser abordados no futuro, isto para além da conclusão da implementação do trabalho não desenvolvido.

Usabilidade

Na usabilidade existem diversos pontos que poderiam ser mencionados para trabalho de futuro, eis alguns que na ótica de utilizador/programador fariam sentido num futuro próximo:

- Possibilidade de filtragem resultados nas listas de dados;

- Existência de um sistema de paginação de resultados em *backend* paralelo ao *frontend*;
- Possibilidade de efetuar uma pesquisa transversal a toda a aplicação;
- Implementar um sistema de notificações ao utilizador, por exemplo, a nível de email ou de browser;
- Personalização de perfis de utilizadores;
- Implementação de uma aplicação mobile nativa.

Integração

Desenvolver um sistema de registo de aplicações externas para a utilização da API da plataforma, a fim de permitir integrar conteúdos de/para outras aplicações, tendo em consideração os limites de utilização da mesma, por exemplo, limitando o número de pedidos diários.

Segurança

A nível de segurança seria interessante realizar um estudo sobre vulnerabilidades quer a nível da Web API, quer a nível da framework Angular JS. Durante o estágio procurou-se implementar a segurança básica exigida, mas a segurança é sempre um ponto a considerar devido às vulnerabilidades que vão sendo descobertas ao longo do tempo.

Testes

Os testes realizados ficaram muito limitados face ao atraso mencionado e seria interessante efetuar um conjunto de testes com uma maior escala, quer a nível de tipo de testes, quer a nível de itens abordados.

Referências

[1] The World Bank Open Data: Container port traffic (TEU: 20 foot equivalent units), http://data.worldbank.org/indicator/IS.SHP.GOOD.TU/countries?display=graph

[2] Millennium bcp: Apoio à Exportação-Importação, http://ind.millenniumbcp.pt/pt/negocios/internacional/Pages/apoio_import_export.aspx

[3] Mag. Martin Posset, Univ. Prof. Dr. Manfred Gronalt, Mag. Hans Häuslmayer: COCKPIIT – Clear Operable and Comparable Key Performance Indicators for Intermodal Transportation, 2010.

[4] Primavera – Business Software Solutions. http://pt.primaverabss.com/pt/

[5] Backbone.JS. http://backbonejs.org/

[6] Ember.Js. http://emberjs.com/

[7] JQuery. https://jquery.com/

[8] Ruby on Rails. http://rubyonrails.org/

[9] Angular JS. https://angularjs.org/

[10] Angular Code. https://github.com/angular/angular

[11] Google Trends. https://www.google.pt/trends/

[12] GitHub. https://github.com/

[13] StackOverflow. http://stackoverflow.com/

[14] Schwaber K. and Sutherland J., The Scrum Guide, 2010.

[15] Jonathan Rasmussom, The Agile Samurai: How Agile Masters Deliver Great Software, 2010.

[16] Wilson, James M, Gantt charts: A centenary appreciation, 2003.

[17] Iberolinhas. http://www.iberolinhas.pt

[18] Sea World. http://www.seaworld.pt/

[19] Gmail - Google. https://mail.google.com/

[20] AJAX. http://www.wrox.com/WileyCDA/Section/id-303217.html

[21] Twitter, Improving performance on twitter.com. https://blog.twitter.com/2012/improving-performance-on-twittercom

[22] JSON Web Token. https://self-issued.info/docs/draft-ietf-oauth-json-web-token.html

[23] IIS. https://www.iis.net/

[24] Node.JS. https://nodejs.org/

[25] Microsoft .NET. http://www.microsoft.com/net

[26] Python. https://www.python.org/

[27] Microsoft ADO.NET. https://msdn.microsoft.com/en-us/library/h43ks021(v=vs.110).aspx

[28] LINQ. https://msdn.microsoft.com/en-us/library/bb397926.aspx

[29] Google Maps. https://developers.google.com/maps/

[30] Marine Traffic API. http://www.marinetraffic.com/en/ais-api-services

[31] Flask. http://flask.pocoo.org/

[32] Jinja2. http://jinja.pocoo.org/docs/dev/

[33] xhtml2dpf. https://github.com/xhtml2pdf/xhtml2pdf

[34] ISO 9126 (2001). http://www.iso.org/iso/catalogue_detail.htm?csnumber=22749

[35] Google Design. https://design.google.com/

[36] IIS Application Request Routing. http://www.iis.net/downloads/microsoft/application-request-routing

[37] Locust. http://locust.io/

[38] Google Insights. https://developers.google.com/speed/pagespeed/insights/

[39] Bower. http://bower.io/

[40] Grunt. http://gruntjs.com/

[41] Brackets. http://brackets.io/

Anexos