UNIVERSIDADE
BEIRA INTERIOR

Classificação automática de abastecimento de gasóleo profissional

Carlos Alberto Cabral Vitória Ferrão Lopes

Relatório do Projeto de Estágio para a obtenção do grau de Mestre
Engenharia Informática
(2º ciclo de estudos)

Orientador da UBI: Prof. Doutor Nuno Gonçalo Coelho Casta Pombo
Orientador da Opensoft: Engenheira Ana Fernandes

Outubro de 2023

Declaração de Integridade

Eu, Carlos Alberto Cabral Vitória Ferrão Lopes que abaixo assino, estudante com o número de inscrição M6935 de/o Metrado de Engenharia Informática da Faculdade Universidade da Beira Interior declaro ter desenvolvido o presente trabalho e elaborado o presente texto em total consonância com o **Código de Integridades da Universidade da Beira Interior**.

Mais concretamente afirmo não ter incorrido em qualquer das variedades de Fraude Académica, e que aqui declaro conhecer, que em particular atendi à exigida referenciação de frases, extratos, imagens e outras formas de trabalho intelectual, e assumindo assim na íntegra as responsabilidades da autoria.

Universidade da Beira Interior, Covilhã 04 /10 /2023

Carlos Alberto Cabral Vitoria Ferrão Lopes

(assinatura conforme Cartão de Cidadão ou preferencialmente
assinatura digital no documento original se naquele mesmo formato)

Dedicatória

Dedico este trabalho a todos que acreditaram a mim.

Agradecimentos

Tendo tomado a decisão de terminar o Mestrado de Engenharia Informática, depois de uma pausa de alguns anos, cumprindo, assim, uma promessa feita às pessoas mais importantes da minha vida, é chegado o momento de expressar a minha gratidão a todas as pessoas que contribuíram para a concretização desta minha etapa.

Agradeço ao meu Orientador de Estágio, Professor Doutor Nuno Pombo, pela sua disponibilidade, compreensão e apoio constantes.

Agradeço também à minha coorientadora de estágio, Engenheira Ana Fernandes, cujos conhecimentos e orientação foram imprescindíveis para o desenvolvimento deste trabalho.

À minha entidade empregadora, Opensoft, na pessoa da Doutora Engenheira Carla Farinha, agradeço o apoio prestado desde o primeiro dia em que tomou conhecimento da minha decisão de terminar o Mestrado.

A todos os meus colegas da minha equipa, o meu obrigado pelo bom ambiente proporcionado e pela coragem que me deram, em particular nas alturas mais complicadas.

Aos meus pais, que me ofereceram a possibilidade de frequentar o Mestrado nesta Universidade, o meu especial agradecimento por acreditaram em mim, de forma incondicional, e encorajarem sempre a lutar pelos meus sonhos.

Agradeço aos meus irmãos e à minha cunhada, também pela coragem e apoio quer na minha decisão de retomar o Mestrado, quer no incentivo que me deram sobretudo nos momentos em que eu estava mais cansado ou desanimado.

Aos meus familiares, amigos e colegas da empresa, o meu mais sentido obrigado pelo vosso apoio, incentivo e compreensão durante esta minha jornada e, em especial, à minha prima, Rosa M. Ferrão, pela palavras encorajadoras e por todo o seu auxílio na redação do presente trabalho.

Aos demais que contribuíram para que este estudo fosse possível, fica também um gesto de gratidão.

A todos os meus sinceros agradecimentos,

Carlos Alberto Cabral Vitória Ferrão Lopes

Resumo

A Unidade Curricular de Projeto de Dissertação ou de Estágio Curricular tem como intuito a planificação de um projeto a ser desenvolvido num futuro próximo, cujo tema é "Classificação Automática de Abastecimento de Gasóleo Profissional". Muito haveria a dizer sobre o assunto, no entanto, tendo em conta a situação difícil que atravessamos presentemente, julgamos pertinente tecer algumas considerações sobre o aumento excessivo dos valores dos combustíveis ao nível dos transportes e, simultaneamente, sobre as respetivas consequências, cujo impacto é extremamente negativo no que a esta área diz respeito, com o intuito de se proceder à contextualização/enquadramento do nosso tema. Mediante esta dura realidade, o Governo, face à escalada dos preços dos combustíveis, em reunião com as associações representativas do setor dos transportes, decidiu reforçar as medidas de apoio.

Neste contexto, pretende, então, o Governo alargar o mecanismo de reembolso parcial para outras áreas de transporte e outros tipos de combustíveis, para ajudar as empresas mais afetadas nesta subida súbita de preços. Este alargamento do mecanismo de reembolso permite fazer melhorias na aplicação **GASPRO**, uma ferramenta de produtividade que possibilita o reembolso do valor parcial dos combustíveis pago pelas empresas, sendo, no entanto, atualmente, uma aplicação muito estática no que concerne à parametrização do tipo de empresas, da diversidade de combustíveis e das especificações de veículos. O grande desafio é, então, a nível da parametrização desta ferramenta, que se pretende melhorar de forma a torná-la mais flexível e mais eficaz. Neste desafio, também se propõe a análise de algumas das principais técnicas de processamento de **XML** (*Extensible Markup Language*), no plano aplicacional, bem como em relação aos vários métodos de validação do ficheiro ao nível sintático e semântico.

Assim, em primeiro, convém registar que a planificação deste projeto foi realizada na Empresa Opensoft (onde se irá realizar o estágio). Porém, antes desta fase, frequentámos a Formação *Spring*, promovida pela empresa acima referida. Finalizada a formação, foi, então, escolhido o tema a ser estudado, preparado e desenvolvido na aplicação *Spring* já existente, **GASPRO**, sob a alçada do Ministério das Finanças e da responsabilidade da Autoridade Tributária Aduaneira (AT).

Fazem ainda parte deste relatório as tecnologias e as ferramentas que foram utilizadas no decurso deste trabalho e sem as quais não seria exequível a realização do presente projeto.

Na secção final, apresenta-se uma conclusão para evidenciar os pontos cruciais que desenvolveram desde a planificação até à concretização do projeto, bem como outros aspetos que poderão ser introduzidos em futuras atualizações da aplicação.

Palavras-chave

Gasóleo, Opensoft; Aplicação GASPRO; Processamento; Validação; Planificação.

Abstract

The Curricular Thesis Project Unit or of Internship Programme has as purpose the planning of a project to be developed in a near future and whose topic is “Automatic Classification of Professional Diesel Supply”. A lot could be said on the subject. However, having in mind the difficult situation we are going through nowadays, we believe it is important to make a few comments on the excessive rise of the fuel price regarding transports and, simultaneously, on the respective consequences, which are extremely negative, as far as this area is concerned, so that we can procede to bringing context to our topic. Facing this harsh realty of rocketing fuel prices, the Government, in meetings with the representative associations of the transport sector, has decided to reinforce supporting measures. In this context, the Government wishes, then, to extend the partial reimbursement mechanism to other areas of transport and types of fuel to help the most affected companies in this sudden rise of fuel price.

This measure allows for upgrades in applying GASPRO, a productivity tool that enables the refundig of the parcial fuel value paid by companies, being, nevertheless, a very static measure regarding the setting of comapnies, diversity of fuel and specification of vehicles. The huge challenge is, then, on the level of the setting of this tool, whose enhancement is intended to make it more flexible and efficient. For this challenge, the analysis of some oh the main **XML** (Extensible Markup Language) processing techniques are suggested, both in the application plam, as well as regarding several validation methods on syntactic and semantic levels.

Therefore, first hand, it should be noted that the planning of this project has been done at the Opensoft company (where the insternship will take place). However, before this phase, we have attended the Spring training promoted by the company mentioned above. Having ended the training it was then chosen the topic to be studied, prepared and deceloped on the Spring application already existing, GASPRO, under the Finance Ministry and the Revenue Authority responsibility.

Also part of this report are the technologies and tools that were used in the course of this work and without which the accomplishment of this project would not be possible.

In the final section, a conclusion is presented to highlight the crucial points that developed from the planning to the project’s implementation, as well as other aspects that could be introduced in future updates of the application.

Keywords

Diesel; Opensoft; GASPRO Application; Processing; Validation; Planning

Índice

Lista de Figuras

Lista de Tabelas

Lista de Listagens

Lista de Acrónimos

AT	Autoridade Tributária e Aduaneira
API	Application Programming Interface
BD	Base de Dados
CIEC	Código dos Impostos Especiais sobre o Consumo
CAE	Classificação Portuguesa das Atividades Económicas
DIPPE	Divisão do Imposto sobre os Produtos Petrolíferos e Energéticos
DSIECIV	Direção de Serviços dos Impostos Especiais de Consumo e do Imposto sobre Veículos
DSD	Document Structure Description
EM	Estado Membro
GASPRO	Gasóleo Profissional
HTTP	Hypertext Transfer Protocol
ICP	Instalação de Consumo Próprio
IEC	Impostos Especiais de Consumo
IDE	Integrated Development Environment
JVM	Java Virtual Machine
MVC	Model-View-Controller
SGBD	Sistema de Gestão de Bases de Dados
UBI	Universidade da Beira Interior
XML	Extensible Markup Language
XSD	XML Schema Definition
XPath	XML Path Language

Capítulo 1

Introdução

A carga fiscal aplicada no combustível tem sido, ao longo dos anos, mais agravada do que a carga fiscal praticada em Espanha. Como consequência, as empresas de transportes públicos optaram por abastecer em Espanha em vez de o fazerem em Portugal.

Com a entrada em vigor do regime de reembolso parcial de gasóleo profissional, este combustível passou a ter uma carga fiscal igual à praticada em Espanha. Para este efeito, as gasolineiras comunicam os abastecimentos nos seus postos e as empresas autorizadas com veículos elegíveis neste regime beneficiam de uma redução no abastecimento de gasóleo profissional, através de um reembolso do imposto efetuado pela Autoridade Tributária (AT).

Atualmente, o sistema informático que implementa o regime de Gasóleo Profissional tem a capacidade de receber as declarações de abastecimentos das gasolineiras, classificar os abastecimentos, de acordo com as atuais regras, e proceder à liquidação do reembolso. Assim, nos abastecimentos de veículos pesados e elegíveis para reembolso, é apurado o montante a reembolsar, de acordo com a Autoridade Tributária.

1.1 Motivação

Em setembro de 2016, foi publicada, no Diário da República, a Portaria n.º 246-A/2016 que regulamenta o regime de reembolso parcial dos combustíveis, referido no seu artigo primeiro: ”[...] reembolso parcial de impostos sobre combustíveis para as empresas de transportes de mercadorias, [...] ”. A portaria esclarece também, no artigo segundo, que ”O presente regime é aplicável aos abastecimentos com gasóleo rodoviário, [...]” e, como se pode ler no primeiro ponto do artigo quinto, ”[...] em veículos tributados na categoria D do Imposto Único de Circulação (IUC)” e ainda no primeiro ponto do artigo sétimo ”[...] licenciado como empresa de transporte de mercadorias, [...]” [13].

Mas o Orçamento de Estado de 2023 prevê um alargamento do regime, abrangendo adquirentes de outra natureza de atividade bem como veículos com outras caraterísticas.

Assim, para conseguir aliviar o aumento dos preços dos produtos petrolíferos, com reflexos prejudiciais na estrutura de custos dos transportes, nas medidas orçamentais propõe-se reforçar o mecanismo de reembolso parcial para o gasóleo profissional, alargando-o ao transporte e, bem assim, à utilização de outro tipo de combustíveis, utilizado no transporte por conta de outrem [14].

Nesse intuito, é nosso objetivo aproveitar a necessidade do alargamento do regime, para melhorar o sistema e torná-lo mais flexível, não só para abranger o que está previsto no alargamento proposto no Orçamento de Estado 2023, mas também para futuras alterações semelhantes que possam surgir.

É neste âmbito que surge a proposta desta dissertação, numa perspetiva de elaborar uma

solução eficaz através da criação de um módulo automático de classificação dos abastecimentos que, mediante diferentes critérios, seja capaz de categorizar o abastecimento como elegível ou não, no âmbito do regime de reembolso.

Neste sentido, serão focados dois aspetos principais. O primeiro relativo à definição de validações sobre elementos do ficheiro de abastecimentos e o segundo relativo ao modelo de classificação de abastecimento.

1.2 Apresentação da Empresa

A Opensoft é uma empresa portuguesa com mais de 20 anos de experiência em engenharia de software e consultoria tecnológica que procura apoiar os seus clientes na transformação digital dos seus negócios, integrando ou desenvolvendo soluções à medida das suas necessidades. Detém uma vasta experiência no desenvolvimento de mais de 1000 projetos tecnológicos, colaborando sucessivamente com clientes de referência, quer na Administração Pública Portuguesa e no setor privado português, quer em instituições públicas internacionais, destacando, a título de exemplo, entidades como a Autoridade Tributária e Aduaneira, a SIBS, a Caixa Geral de Depósitos, a Embaixada da República de Cabo Verde em Portugal, o Ministério das Finanças de Cabo Verde, entre outras.

Paralelamente, ao longo dos anos, a empresa tem também aplicado a sua experiência no desenvolvimento de produtos de software, como são exemplo o SAFTPRO, solução de recolha e validação massiva de dados através de ficheiros, o aplicações SIMN, ERP para a gestão de cartórios notariais, que comercializa diretamente, um software de comercialização direta, assegurando o necessário acompanhamento pós-venda ou o Lightweightfom, uma framework open source concebida com o objetivo de facilitar o desenvolvimento de formulários web exigentes, que se caracterizam por aportar um elevado número de campos e de regras de negócio (validações).

1.3 Apresentação do Estagiário

O estagiário, Carlos Alberto Cabral Vitória Ferrão Lopes, é um trabalhador-estudante de 36 anos, natural de Viseu e residente em Santa Iria de Azoia. Concretizou com sucesso os seus estudos no Colégio da Via-Sacra – Viseu, e, posteriormente, finalizou o ensino secundário na escola Emídio Navarro, no curso de Tecnologias de Informática, também na cidade de Viseu. Licenciou-se em Engenharia Informática na Universidade da Beira Interior (UBI), em 2014 e, neste momento, encontra-se a trabalhar na empresa Opensoft, estando a finalizar o 2.º ciclo (Mestrado) em Engenharia Informática, na mesma universidade. Em 2015, começou por trabalhar na Altran, com escritórios na cidade do Fundão, durante dois meses, tendo depois vindo trabalhar e viver para Lisboa. Não conseguindo, nessa época, conciliar o trabalho em Lisboa com o desenvolvimento do Mestrado, o estagiário considerou, então, mais oportuno adiar o Mestrado para uma fase posterior que proporcionasse condições de estabilidade sobretudo a nível profissional, tal como acontece presentemente na empresa onde trabalhamos que nos dá um apoio muito significativo.

Atualmente, o estagiário conclui o 2.º ano de Mestrado em Engenharia Informática, no qual realiza o seu plano de estágio que irá decorrer neste ano letivo 2022/2023, tendo como objetivos a obtenção do grau de Mestre em Engenharia Informática.

1.4 Objetivos do Estágio

O resultado deste estágio será apresentar a evolução do atual processo de classificação de combustíveis, de forma a adotar um modelo automático e flexível de categorização dos abastecimentos.
Para este efeito, pretende-se que o sistema seja capaz de:

- receber diferentes tipos de critérios (sejam associados ao veículo ou ao adquirente);
- saber interpretar e classificar um abastecimento de acordo com o tipo de critério recebido;
- categorizar automaticamente os abastecimentos de acordo com os (novos) critérios.

Em suma, o presente estudo e a consecução dos objetivos delineados levarão a um só objetivo final e principal, que consiste na melhoria do produto aplicação GASPRO.

1.5 Trabalho Proposto

O sistema atual do regime de gasóleo profissional é muito estático no que diz respeito à parametrização do tipo de empresas, da diversidade de combustíveis e das especificações de veículos.

O primeiro aspeto a considerar neste nosso trabalho é aprender o processo de processamento de ficheiro de **XML** e o processo de validação do mesmo, mas, também, tornar esta aplicação muito mais flexível e parametrizável.

Desse modo, propomos alterar o formato do ficheiro de abastecimento fornecido pelos emitentes, de forma a contemplar outros tipos de combustíveis e, assim, também melhorar o processamento de ficheiros e o processo de validações sintáticas e semânticas.

Outro aspeto muito importante, já referido, é o que diz respeito à questão flexível e parametrizável das configurações da aplicação, com a possibilidade de abranger vários tipos de combustíveis, vários tipos de empresas, vários tipos de transportes, vários tipos de veículos, assim como as suas características, por forma a estarem contemplados no regime de reembolso parcial de impostos.

1.6 Contexto de Investigação

O desenvolvimento deste projeto decorrerá sempre sobre constante supervisão e acompanhamento da coordenadora, Engenheira Ana Fernandes, que delineou o planeamento dos diversos pontos a abordar no presente projeto.

Todas as fases decorrentes do processo de investigação e implementação foram realizadas em contexto empresarial na Opensoft.

1.7 Organização do Documento

O segundo capítulo, Estado da Arte, aborda a arquitetura do GASPRO, o processamento de ficheiros, a estrutura das validações, o modelo de classificação de abastecimento e, por fim, as tecnologias a utilizar.

O terceiro capítulo, Planificação, centra-se na Formação *Spring* e, posteriormente, regista as tarefas e o respetivo plano de execução das mesmas durante o estágio.

O quarto capítulo, Metodologia, Arquitetura, Tecnologias e Ferramentas, apresenta a arquitetura e as metodologias de trabalho utilizada no projeto e, por fim, as tecnologias e ferramentas usadas durante o estágio.

O quinto capítulo, Implementação, aborda todos os dados de implementação desenvolvidos durante o estágio, considerando os mais relevantes, bem como a sua respetiva explicação.

O sexto capítulo, Testes, abordada a metodologia utilizada para a validação do trabalho efetuado, tanto em relação à qualidade do código produzido como aos testes automáticos.

No sétimo capítulo, Conclusões e Trabalho Futuro, mencionam-se todas as conclusões relacionadas com o projeto, bem como o trabalho a ser desenvolvido futuramente.

Capítulo 2

Estado da Arte

2.1 Introdução

Neste capítulo será feita uma descrição da arquitetura atual do sistema, assim como uma apresentação e análise de algumas das ferramentas e técnicas utilizadas.

Na secção 2.2, são apresentados os utilizadores finais do sistema e descritas as funcionalidades disponibilizadas para cada um deles, numa perspetiva de enquadrar o sistema num contexto real de utilização.

Na secção 2.3, apresenta-se a arquitetura, com foco na visão lógica que mostra a informação sobre as várias partes do sistema e a visão da implementação que expõe o sistema no ponto de vista do desenvolvimento.

Na secção 2.4, evidenciam-se as várias especificações de processamento de ficheiros de **XML**.

Na secção 2.5, apresentamos as múltiplas validações de **schemas** para validar os ficheiros **XML**.

Na secção 2.6, expomos as conclusões tiradas das apresentações das especificações de processamento de ficheiros **XML** e das várias validações de **schemas** para validar os ficheiros **XML**.

2.2 Categorias de Utilizadores no sistema Gasóleo Profissional

Antes de entender a arquitetura do sistema é necessário esclarecer dois pontos principais. O primeiro é relativo às categorias de utilizadores existentes no sistema e saber qual a sua representação em entidades do mundo real. O segundo ponto surge como uma sequência do primeiro para apresentar as funcionalidades disponibilizadas aos diferentes utilizadores. O **GASPRO** define dois sites principais, um destinado aos funcionários da DSIECIV e outro destinado aos emitentes e adquirentes. As funcionalidades disponibilizadas a cada utilizador do sistema são listadas abaixo.

Funcionário da DSIECIV: Funcionário dos serviços centrais da DSIECIV com responsabilidade para verificar a veracidade dos dados registados sobre um adquirente ou veículo de outro Estado Membro.

Adquirente: Sujeito passivo, comprador do gasóleo profissional e beneficiário do regime de reembolso parcial, cumprindo os requisitos legalmente previstos.

Emitente: Entidade emissora de cartões profissionais (empresas petrolíferas) ou então outros representantes em Portugal ou emissores de outros mecanismos de controlo certificados pela Autoridade Tributária (AT), incluindo instalações de consumo próprio autorizadas,

que cumpram os requisitos legalmente previstos.

Fornecedor: Entidade responsável pelo fornecimento de combustíveis às instalações de consumo próprio.

Tendo presente as diferentes categorias de utilizadores que interagem com o sistema, podemos, agora, partir para uma análise mais detalhada da arquitetura do mesmo. Assim, na secção seguinte, será realizada uma análise arquitetural do sistema, partindo de uma visão de mais alto nível que será concretizada em cada um dos módulos isoladamente. O objetivo deste ponto é identificar e descrever a arquitetura destes componentes, identificando as suas limitações e condicionalismo que serão alvo de análise no âmbito deste projeto.

2.3 Arquitetura do sistema Gasóleo Profissional

As representações da arquitetura pretendem patentear as diferentes visões focadas em aspetos específicos distintos de uma arquitetura.
As visões que aqui reproduzimos centram-se, assim, em ilustrar os seguintes aspetos:

- Visão **Lógica** – ilustra informações sobre as várias partes do sistema, preocupando-se com as funcionalidades disponibilizadas aos utilizadores;
- Visão de **Implementação** – ilustra o sistema do ponto de vista do desenvolvimento, concentrando-se na organização do software em módulos e subsistemas.

2.3.1 Visão Lógica

A visão lógica ilustra as estruturas da solução que respondem aos requisitos identificados. Estas estruturas são evidenciadas na figura 2.1, a seguir apresentada, com a representação dos componentes da solução e dos sistemas externos com que interagem.

2.3.1.1 Os componentes

O componente Gaspro é a representação do gasóleo profissional, disponível na Internet e tem como objetivo disponibilizar as funcionalidades previstas neste regime aos principais utilizadores, incluindo adquirentes, fornecedores e emitentes de cartões de gasóleo profissional (postos de abastecimento e consumo próprio).

2.3.1.1.1 O componente GASPRO

O componente **GASPRO** é a representação do gasóleo profissional, disponível na ***Internet*** e tem como objetivo disponibilizar as funcionalidades previstas neste regime aos principais utilizadores, incluindo adquirentes, fornecedores e emitentes de cartões de gasóleo profissional (postos de abastecimento e consumo próprio).

2.3.1.1.2 O componente GASPROIntra

O componente **GASPROIntra** representa a ***Intranet*** do gasóleo profissional disponível aos funcionários da DSIECIV responsáveis por monitorizar e acompanhar este regime e a

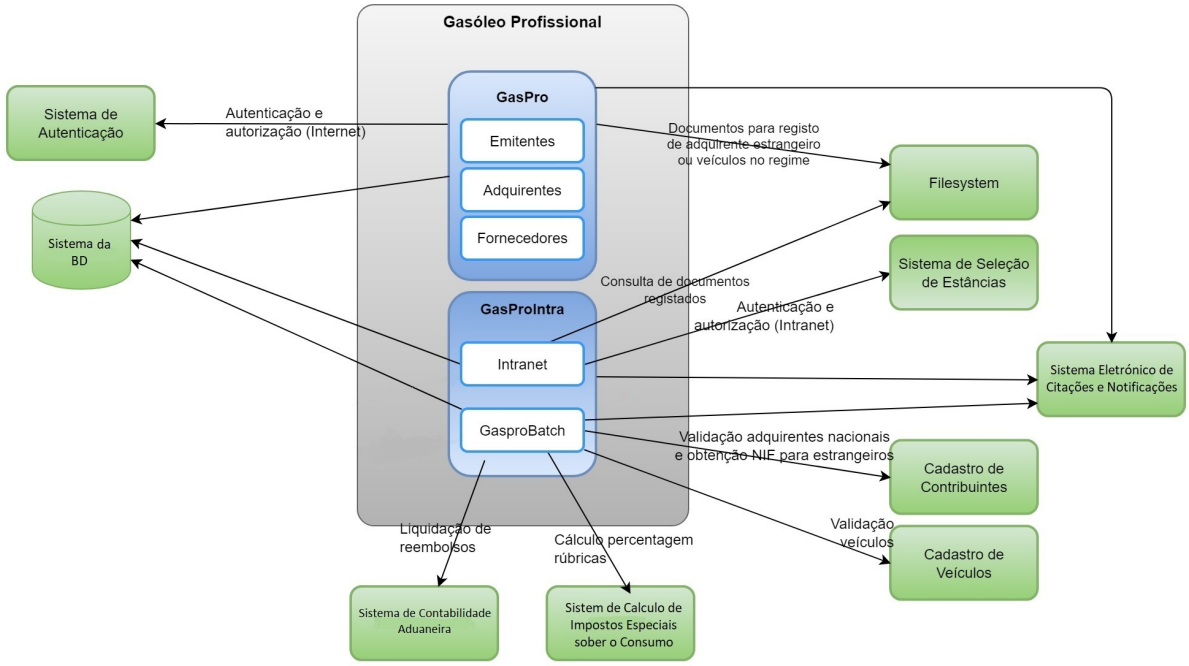

Figura 2.1: Visão Lógica.

responsabilidade deste componente é fornecer todas as operações de controlo, monitorização, acompanhamento e intervenção no contexto deste regime de gasóleo profissional.

2.3.1.1.3 O componente GASPROBatch

O componente **GASPROBatch** representa um ***batch*** para processamento de tarefas periódicas e a responsabilidade deste componente é processar periodicamente conjuntos de informação, incluindo a liquidação mensal de reembolsos, no âmbito deste regime.

2.3.1.2 As interfaces

Interfaces são entidades externas à solução com as quais a solução precisa de interagir.

2.3.1.2.1 Sistema de Autenticação

Esta interface, conhecida como SA, assegura os mecanismos de autenticação e autorização de utilizadores dos portais *Internet* da AT. Para o efeito, a aplicação encaminha o utilizador para este sistema que valida a autenticidade do utilizador e permite a consulta dos perfis de acesso autorizados.

Os mecanismos de autorização permitem, para cada utilizador, a atribuição de perfis específicos que definem as diferentes ações e páginas que o utilizador pode ver. Com base nas informações registradas no sistema, a aplicação é capaz de:

- garantir o acesso autenticado de modo que só contribuintes autenticados perante esta interface possam interagir com a aplicação;

- restringir o acesso a funcionalidades específicas com base no perfil do utilizador. De acordo com o perfil do utilizador, este terá ou não permissão para executar operações específicas;
- possuir acesso seguro de acordo com o próprio utilizador. Desta forma, além de filtrar o acesso e as operações com base no perfil, algumas informações só podem ser consultadas ou editadas por utilizadores específicos.

2.3.1.2.2 Sistema de Seleção de Estâncias

Esta interface verifica a autenticidade e autorização dos técnicos das alfândegas para acederem aos sistemas das estâncias aduaneiras. Assim, a interface valida os funcionários em determinada estância aduaneira por eles selecionada e acrescenta à sessão as suas credenciais e dados da estância ao pedido do utilizador.

2.3.1.2.3 Sistema (Central) de Cadastro

O Sistema (Central) de Cadastro é uma interface que contém os dados de cadastro dos contribuintes. A comunicação com esta interface pretende obter dados cadastrais necessários a determinados fluxos da solução.

2.3.1.2.4 Cadastro de Veículos

Esta interface contém os dados do cadastro de veículos portugueses a uma data específica, tendo em conta o histórico do veículo.

2.3.1.2.5 Sistema de Impostos especiais sobre o Consumo

Trata-se de uma interface que representa um método de cálculo de Imposto Especial de Consumo. Para o efeito, a interface recebe um conjunto de *inputs* fundamentais ao cálculo, incluindo estância aduaneira, data e produto(s) da pauta aduaneira. Permite, ainda, identificação de isenções e certificados afetos ao operador. O resultado da interface é o cálculo do imposto, no seu valor global, mas, também, a sua distribuição por rubricas (despesas públicas para subsetores do sector público administrativo) e medidas.

2.3.1.2.6 Sistema de Contabilidade Aduaneira

O sistema de Contabilidade Aduaneira pretende integrar os Sistemas Declarativos e os Sistemas Fornecedores de serviços necessários para o circuito declarativo. Para o efeito, disponibiliza várias operações, incluindo criar, anular, consultar ou retificar registos de liquidação ou validar garantias.

2.3.1.2.7 Sistema Eletrónico de Citações e Notificações

Esta interface representa o sistema de notificações da AT, permitindo a comunicação com o contribuinte ou outro utilizador em diversos formatos. Esta interface admite também a produção de Composição de Citações ou Notificações (PDFs ou TXTs) e o envio destas Citações e Notificações (por carta registada, email, SMS, Via CTT).

2.3.1.2.8 Sistema da BD

A base de dados (BD) permite o armazenamento de dados de maneira estruturada e com a menor redundância possível. Estes dados devem poder ser utilizados por aplicações e utilizadores diferentes. Assim, a noção básica de dados é acoplada a uma rede, a fim de poder reunir estas informações. A gestão da BD é feita graças a um sistema de administração de bancos de dados (SGBD) que não é mais do que um conjunto de serviços (*softwares*) que permite gerir as bases de dados, incluindo acessos, autorizações a utilizadores e manipulação dos dados (inserção, eliminação e alteração). Esta base de dados estende-se de um modelo relacional para um modelo objeto-relacional, possibilitando o armazenamento de modelos de negócios complexos num banco de dados relacional.

2.3.2 Visão de Implementação

A visão de implementação configura a organização dos componentes do sistema. Os módulos e as suas dependências são organizados numa hierarquia de camadas, cada uma fornecendo uma *interface* às camadas acima. A figura 2.2, que a seguir se apresenta, mostra a arquitetura da solução, com foco na integração entre os componentes. Observando a imagem podemos, ainda, constatar que a solução adota um modelo de camadas que fornece uma abstração ao nível da hierarquia dos componentes, em que a camada inferior fornece uma interface às camadas superiores, o que aumenta, de forma significativa, a modularidade do código, facilitando a sua manutenção e evolução.

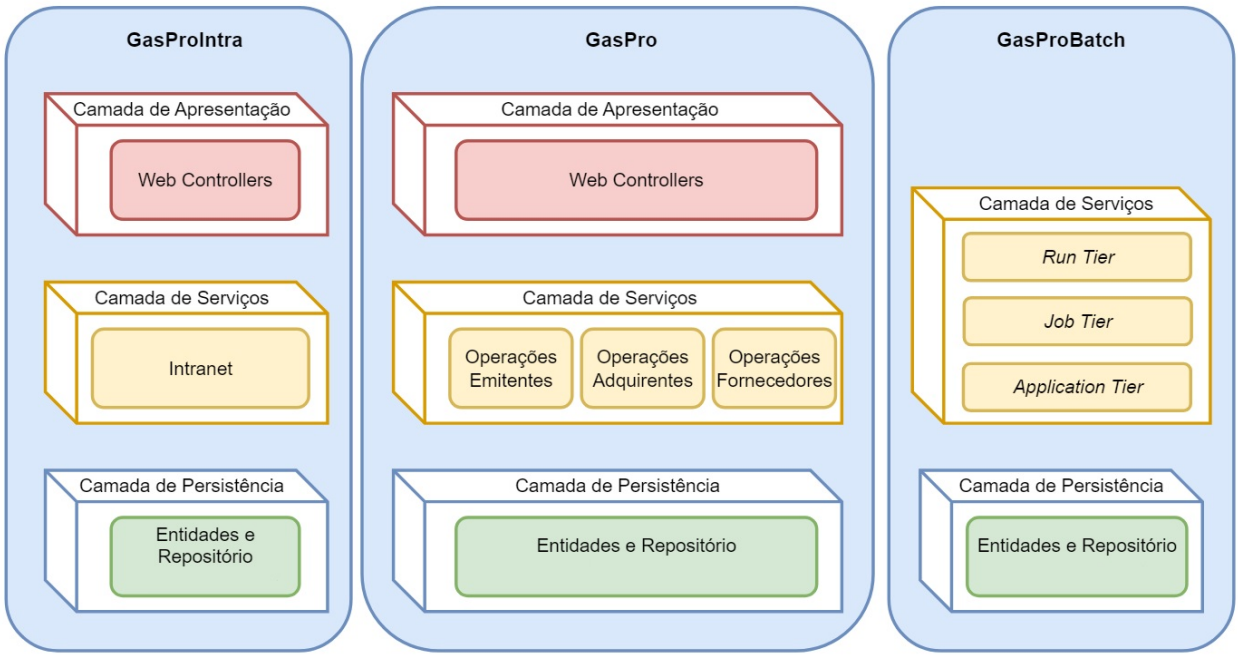

Figura 2.2: Visão da Implementação.

2.3.2.1 Camada de Apresentação

A camada de apresentação contém a lógica para lidar com as interações entre o utilizador e o sistema através de HTTP. A camada é responsável por produzir todas as páginas da *web* usadas para interagir com o utilizador, processando os seus pedidos e devolvendo a resposta adequada. Também é responsável por lidar com exceções lançadas pelas restantes camadas e chamar funções de alto nível, como a autenticação e autorização, fornecidas na Camada de Serviço.

2.3.2.2 Camada de Serviços

A camada de serviço contém a lógica de negócio estruturada em métodos de alto-nível. Os serviços disponibilizados nesta camada fornecem API's públicas que se comunicam com recursos externos como ficheiros de sistema.

2.3.2.3 Camada de Persistência

A camada de persistência é a camada inferior da aplicação web que representa as entidades da Base de Dados para comunicação com a mesma. Esta camada atua como um meio que fornece interfaces para controlar vários objetos de dados e abstrair os detalhes técnicos sobre as estruturas de dados e seu armazenamento. A camada de acesso a dados sabe como e onde as entidades persistentes estão armazenadas.

2.3.2.4 O componente GASPRO

O componente **GASPRO** encontra-se dividido nas três camadas que foram referidas nas secções 2.3.2.1, 2.3.2.2 e 2.3.2.3, sendo que a camada de serviços está logicamente dividida em módulos orientados a funcionalidades dos utilizadores:

- **Emitentes**, incluindo públicos e instalações de consumo próprio tanto para comunicação de dados de abastecimentos como consulta dos mesmos;
- **Adquirentes**, permitindo, designadamente a consulta de abastecimentos comunicados pelos emitentes e de reembolsos emitidos pelo sistema Gasóleo Profissional;
- **Fornecedores**, abastecendo um meio de realizar a comunicação de fornecimentos a instalações de consumo próprio.

2.3.2.5 O componente GASPROIntra

O componente **GASPROIntra** encontra-se dividido nas três camadas, já mencionadas nas secções 2.3.2.1, 2.3.2.2 e 2.3.2.3, e está logicamente dividida no seguinte módulo:

- **Intranet**, responsável por todas as funcionalidades disponibilizadas aos funcionários aduaneiros;

2.3.2.6 O componente GASPROBatch

O componente **GASPROBatch** é dividido em duas camadas: serviços e persistência, pois, neste caso, a camada de serviços contém serviços automatizados, que correm periodicamente, não necessitando de ser despoletados pelos utilizadores. A camada de serviços está logicamente organizada de acordo com a estrutura do ***Spring Batch***:

- **Run Tier:** preocupa-se com o agendamento e o lançamento da aplicação, ou seja, permite o planeamento interdependente e é baseado em tempo de tarefas em lote, além de fornecer, também, recursos de processamento paralelo;

- **Job Tier:** responsável pela execução global do job. Executa sequencialmente steps do batch, assegurando que todos os steps estão no estado correto e que todas as políticas apropriadas são aplicadas;

- **Application Tier:** contém os componentes necessários para a execução e reforça as políticas de concretização, incluindo intervalos de commit ou indicadores estatísticos.

2.4 Processamento de Ficheiros (XML)

Existem três tipos de ficheiros que são submetidos na aplicação para serem processados automaticamente pelo componente **GASPROBatch**.

Esses ficheiros são:

- Ficheiros de **Abastecimento Profissional** onde são registados os abastecimentos de gasóleo profissional processados diariamente, de forma a identificar os abastecimentos elegíveis para reembolso parcial de Gasóleo Profissional;

- Ficheiros de **Posto de Abastecimento** que são dados a conhecer pelo posto de abastecimento e são processados automática e periodicamente. Por cada ficheiro, e de acordo com a estrutura definida, o processo faz a leitura de cada posto, validando-o e registando-o. Cada posto registado fica sujeito a apreciação de regularização ou não pela DSIECIV;

- Ficheiros de **Fornecimento** que registam os fornecimentos de consumo próprio que são processados periodicamente.

Pelo facto de a sintaxe de ficheiros **XML** seguir uma estrutura muito semelhante ao paradigma orientado a objetos, existiu um grande esforço em mapear estes ficheiros em objetos, de modo a poderem ser manipulados de forma mais intuitiva dentro do código aplicacional[1].

Nesta secção são analisadas algumas das principais modelos para processamento de **XML** a nível aplicacional.

[1] [15] e [16]

2.4.1 Especificações

No passado, foi realizada uma tarefa para migrar o *Daemon Threads* para o Batch, para processamento dos ficheiros de Abastecimento e de Fornecimento e, na altura, a equipa de desenvolvimento escolheu uma especificação **StAX**. No entanto, dado que o processamento dos ficheiros de Estabelecimento foi o primeiro a ser desenvolvido desde da raiz, para ser um Batch, por se o mais antigo dos três, veio a aplicar-se uma especificação **SAX**.

2.4.1.1 SAX *Simple API for XML*

O *Simple API for XML* (**SAX**) é um modelo de parser de baixo nível baseado em eventos para processamento de ficheiros **XML**. Isto significa que aplicações que adotem o modelo **SAX** irão receber eventos à medida que o ficheiro **XML** é processado.

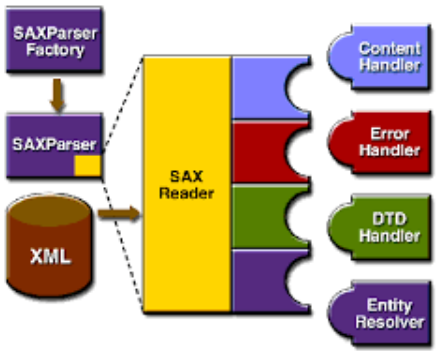

Figura 2.3: *SAX Simple API for XML*. [1]

O processamento é realizado de forma linear em modo de *stream*, o que impossibilita o acesso aleatório a um campo do ficheiro **XML**. O uso desta especificação, apesar de ser de muito baixo nível, é aconselhada para processamento de ficheiros de grande dimensão que podem não caber integralmente em memória, como é o caso dos ficheiros Estabelecimento[2].

2.4.1.2 DOM - *Document Object Model*

A especificação *Document Object Model* (**DOM**) surge como uma alternativa aos *parsers* baseados em eventos. Neste modelo, o **XML** é carregado integralmente para memória, tornando, assim, possível um acesso aleatório a qualquer campo do ficheiro, o que dá um nível de flexibilidade adicional. O **DOM** é, contrariamente ao **SAX**, muito mais simples de utilizar, dado que o ficheiro **XML** é encarado como um objeto composto por vários campos que podem ser acedidos e alterados livremente pelo código aplicacional[3].

[2] [17], [18], [19], [20], [21], [22], [23], [19] e [24]
[3] [18], [25], [21], [22], [19], [23], [19] e [24]

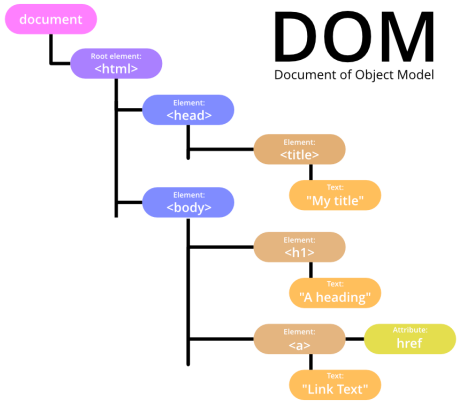

Figura 2.4: *DOM - Document Object Model.* [2]

2.4.1.3 StAX - *Streaming API for XML*

O *Streaming API for XML* (**StAX**) surge como uma melhoria relativamente ao **SAX**, encaixando, por isso, na categoria de processamento em *stream*, com um baixo consumo de memória. O *StAX parser* destaca-se do *SAX parser* por adotar um *Pull Model* enquanto o **SAX** utiliza *Push Model*. Isto implica que, ao invés de o *parser* chamar o *handler* definido pelo programador, é o *handle* que realiza chamadas ao *parser*. Esta pequena alteração facilita bastante o controlo do fluxo de processamento, pois, agora, o código aplicacional pode controlar o *parser* de modo a poder avançar para o próximo evento.

O uso de *Pull Model* torna ainda possível a realização de *subparsing* do ficheiro **XML**. O *subparsing* consiste em permitir que um dado bloco do ficheiro *XML* seja processado de uma determinada forma pré-definida. Deste modo, sempre que um bloco de determinado tipo é detetado o *parser* irá atuar no sentido de realizar o processamento definido[4].

2.4.1.4 JAXB - *Java XML Binding*

O **JAXB** é uma especificação de alto nível que fornece um nível adicional de abstração sobre a forma de uma *API*, que permite invocar *parsers* **DOM** ou **SAX** para processamento de ficheiros **XML**.

Usando a abordagem **SAtX**, o documento começa a ser processado desde o início e, à medida que a aplicação faz chamadas ao *parser*, este vai avançando pelo ficheiro de forma sequencial. A premissa inicial de que nada é persistido em memória é mantida durante todo o processamento do ficheiro. Também não é possível realizar qualquer tipo de modificação ao ficheiro **XML** que está a ser processado, apenas é possível ler e realizar computações com base nos dados.

Na abordagem **DOM**, o ficheiro **XML** passa pelo *parser*, sendo criada uma representação em árvore deste mesmo ficheiro que pode, depois, ser acedida pela aplicação sobre a forma de um objeto. Deste modo, é possível navegar livremente pela estrutura da árvore e, até mesmo,

[4] [26], [27], [26], [28] e [19]

manipular esta estrutura, caso seja necessário[5].

2.4.1.5 Resultados

Nesta secção, serão apresentados os resultados de processamento de exemplos das especificações referidas anteriormente. O exemplo utilizado para estas execuções é adaptado a cada especificação, sendo completamente distinto do utilizado no **GASPRO**.

```
<Empresas>
  <Empresa registo = "1">
    <NomeEmpresa>100Limpar</NomeEmpresa>
    <CAE>01610</CAE>
    <TipoFicheiro>C</TipoFicheiro>
    <NIF>123456789</NomeEmpresa>
    <MoradaEmpresa>
      <Rua>Rua XXXX</Rua>
      <Cidade>Viseu</Cidade>
      <CodigoPostal>5600-123</CodigoPostal>
      <Pais>PT</Pais>
    </MoradaEmpresa>
    <AnoFiscal>2023</AnoFiscal>
    <DataInicio>20213-01-01</DataInicio>
    <DataFim>2023-12-31</DataFim>
  </Empresa>
  <Empresa registo = "2">
    <NomeEmpresa>100Carne</NomeEmpresa>
    <CAE>95110</CAE>
    <TipoFicheiro>U</TipoFicheiro>
    <NIF>987654321</NomeEmpresa>
    <MoradaEmpresa>
      <Rua>Rua XXXX</Rua>
      <Cidade>Lisboa</Cidade>
      <CodigoPostal>1000-123</CodigoPostal>
      <Pais>PT</Pais>
    </MoradaEmpresa>
    <AnoFiscal>2023</AnoFiscal>
    <DataInicio>20213-01-01</DataInicio>
    <DataFim>2023-12-31</DataFim>
  </Empresa>
</Empresas>
```

Listagem 2.1: Exemplo da estrutura utilizado

O ficheiro ***XML*** tem o tamanho aproximado de **4GB** e, quando compactado, fica sensivelmente **1GB**. Antes de apresentar os resultados obtidos com as várias execuções de cada especificação acima apresentada, é importante definir qual o ambiente de desmonstração utilizado para a realização destas demonstrações.

[5] [29], [24], [21], [7], [30], [31], [32] e [33]

Esprecificação	Tempo de execução
SAX - Simple API for XML	00:03:20
DOM - Document Object Model	00:09:40
StAX - Streaming API for XML	00:02:01
JAXB - Java XML Binding	00:02:21

Tabela 2.1: Resultados da demonstração

- **Modelo:** HP ProBook 450 G8
- **Sistema Operativo**: Windows 11 Pro
- **RAM:** 32GB
- **Disco:** 500GB

Para conseguirmos uma informação relativamente ao tempo de execução das especificações usadas, aplicámos a ferramenta de desenvolvimento *IntelliJ idea*. Os resultados obtidos nesta demonstração podem ser visualizados na tabela 2.1.

2.5 Validação de *Schemas*

Como já foi referido, os três tipos ficheiros são documentos em formato *XML* submetidos a uma posterior validação, sendo, por isso, necessário definir qual a estrutura destes mesmos ficheiros. A criação desta aplicação foi feita com base num um tipo de validação do formato do ficheiro de **XML**, na altura, o **XSD**. Será este tipo de validação (**XSD**), feito no início do desenvolvimento da aplicação, o mais adequado e eficaz ou haverá uma outra alternativa melhor que possa oferecer mais vantagens?

2.5.1 *Schematron*

O **Schematron** é um método de validação de ficheiros baseado na definição de asserções, usando **XPath** (linguagem para interrogação de ficheiros **XML**), que é, depois, confrontado com a estrutura do ficheiro **XML**.

Permite a definição de regras dependentes do conteúdo de cada elemento, algo que não é possível em nenhum dos outros métodos. Contudo, o seu uso não é aconselhado para a definição de restrições de conteúdo em múltiplos campos presentes no ficheiro **XML**. Deve, assim, ser usado como um complemento aos outros métodos e não de forma isolada[6].

2.5.2 DTD – *Document Type Definition*

É de todos os modelos o mais rápido e fácil de usar, simplificando significativamente o código aplicacional. Contudo, apresenta, em contrapartida, um poder expressivo muito reduzido quando comparado com os modelos anteriores, permitindo apenas definir a estrutura do documento, sem qualquer tipo de restrição sobre o seu conteúdo. Este facto decorre diretamente da inexistência de um sistema de tipos, que permita definir tipos de elementos.

[6] [34], [35] e [36]

O *Document Type Definition* (**DTD**) apresenta, ainda, limitações relativas aos *namespaces*, pois os prefixos utilizados para distinguir elementos com o mesmo nome são considerados caracteres pertencentes ao nome do elemento. Isto torna muito difícil a produção de ficheiros **DTD** que possam ser integrados entre si. O modelo **XML Schema**, por ser mais recente, já contempla este cenário, permitindo uma fácil integração de *tag* igual entre ficheiros diferentes[7].

2.5.3 XML *Schema* (XSD)

Sistema de validação muito mais poderoso que o **DTD**, pois permite a associações de tipos aos vários elementos, assim como a definição de restrições mais sofisticadas ao nível das estruturas. O **XSD** torna possível a realização da maioria das verificações diretamente no *schema*, o que faz com que o processo de criação destes *schemas* seja mais complexo. Deste modo, a lógica de validação é implementada diretamente pelo **XSD**, o que diminui significativamente o código aplicacional dedicado à validação do ficheiro **XML**. O **XML** tem um sistema de tipos mais poderoso que o **DSD** (*Document Structure Description*), permitindo criar estruturas de grande complexidade através da composição de estruturas mais simples, formando, assim, uma estrutura em árvore que permite a navegação pelos campos do ficheiro, usando caminhos definidos e uma sequência de elementos aninhados. Outra grande vantagem de usar ficheiros **XSD** é que estes seguem um padrão orientado aos objetos, que encaixa perfeitamente no contexto de linguagens como o **Java** na qual está desenvolvido o projeto **GASPRO**[8].

2.5.4 *Relax NG*

O **Relax NG** foi criado com o objetivo de oferecer um poder semelhante ao **XSD**, mas simplificando significativamente toda a complexidade característica deste tipo de ficheiros. Desta forma, o **Relax NG** oferece dois tipos de sintaxe, um em formato **XML** e outro num formato mais compacto que torna a leitura e o entendimento muito mais simples. Existe, ainda, uma grande variedade de ferramentas, disponibilizadas para linguagens como o **Java** e **NET** que permitem realizar validações utilizando **Relax NG**[9].

2.6 Conclusão

Como conclusão às abordagens definidas anteriormente, são apresentadas abaixo, sobre a forma de tabela 2.2, algumas das vantagens e desvantagens associadas a cada especificação. Analisando a tabela 2.2, podemos facilmente identificar que as *frameworks* para processamento de ficheiros de **XML** foram evoluindo de forma sólida ao longo dos anos, criando, assim, um nível de abstração que permite ao programador navegar na estrutura de um ficheiro **XML** de forma simples. Atualmente, de todas as especificações anteriormente descritas, o

[7] [37], [38], [39], [40] e [23]
[8] [41], [42], [43], [44], [45], [46] [42] e [47]
[9] [3], [48] e [49]

JAXB é a mais utilizada, existindo diversas distribuições/implementações, cada uma com as suas vantagens e desvantagens.

Deve acrescentar-se que **SAX**, **DOM**, **StAX** e **JAXB** são apenas especificações e não implementações concretas para processamento de ficheiros **XML**. Para cada especificação apresentada anteriormente existem diversas implementações que podem ser *open source* ou comerciais. Na tabela 2.3, são apresentadas algumas das implementações mais populares usadas por aplicações **Java**.

Pela observação da tabela, podemos verificar que, apesar de todas as implementações seguirem a especificação **JAXB**, nem todas são totalmente fiéis à mesma. Como exemplo, podemos tomar a *Framework* **Simple** que não permite a validação com base em ficheiros **XSD**, o que vai contra o que está definido na especificação [12].

O **GASPRO** é uma aplicação que foi desenvolvida em linguagem **Java**. Por este motivo, temos a oportunidade de passar a usar **JAXB** como *parser* de ficheiros **XML**. Contudo, devemos ter em atenção as restrições impostas ao produto, relembrando que o **GASPRO** foi desenhado com o objetivo de ser uma ferramenta para validação e processamento de ficheiros de elevada dimensão em *real time*.

Para satisfazer a especificação do produto é necessário que o ficheiro vá sendo validado à medida que os segmentos de ficheiro vão chegando ao servidor. Este fator, juntamento com o facto de os ficheiros submetidos poderem ser na ordem das centenas de *gigabytes*, impossibilita o seu mapeamento em memória, o que nos permite descartar, de forma imediata, especificações como o **DOM**.

Especificação	Vantagens	Desvantagens
SAX	>Baseado em eventos; >Gestão de memória eficiente; >Mais rápido que o modelo **DOM**; >Apresenta suporte para validação de usando **schema** (**XSD**).	>Ficheiro **XML** não é mapeado diretamente em objetos, sendo necessário tratar os eventos lançados e criar os objetos manualmente; >*Parser* é muito básico, permitindo apenas leituras para a frente e nunca para trás (no estilo de processamento de *streams*); >Não permite alteração do ficheiro **XML**, apenas leitura; >Não apresenta suporte para interrogação usando **XPath**; >Difícil de usar.
DOM	>**XML** é tratado como um objeto em memória; >Preserva a ordem dos elementos; >Permite leitura e escrita do **XML**; >Suporte para validação usando **XSD**.	>Consumo excessivo de memória, pois o **XML** é carregado integralmente como um objeto único; >Muito lento; >Modelo muito genérico, apenas existe o conceito de Nó.
StAX	>Combina a eficiência do **SAFX** com a facilidade de uso do **DOM**; >Permite *sub-parsing*; >Permite a leitura de diversos ficheiros usando um único *thread*; >Processamento paralelo de ficheiros **XML** é simples.	>Não apresente suporte para validação usando **XSD**; >Só permite leituras sequenciais assim como o **SAX**; >Não permite manipulação do **XML**.
JAXB	>Abstração de alto nível que torna possível manipulação do ficheiro **XML** sem ter conhecimento de **XML**; >Leitura pode ser feita de forma bidirecional; >Gestão de memória otimizada em relação ao **DOM**; >O **JAXB** ao contrário do **DOM** e **SAX** cria um *parser* específico tendo em conta o ficheiro **XSD**; >Permite a conversão direta de **XML** para sistema de tipos do *Java*; >Permite manipulação do **XML** através de uma *API* disponibilizada por um objeto representativo do ficheiro **XML**.	>Só permite realizar o *parsing* de **XML** válido.

Tabela 2.2: Vantagens e Desvantagens das especificações.

Resta-nos, assim, utilizar especificações baseadas em *streams* como o **SAX** ou **StAX**, sendo o **StAX** preferível devido às vantagens apresentadas anteriormente e, por isso, vamos manter as especificações escolhidas no passado.

Nome	Licenciamento	Geração de código Baseada em XSD	Mapeamento Costumizável
Apache Commons Betwixt	Apache	Desconhecido	Desconhecido
Apache XMLBeans	Apache License 2.0	Sim	Desconhecido
Castor	Apache 2.0	Desconhecido	Desconhecido
Java Architecture for XML Binding (JAXB)	Não	Sim	Sim
XStream	Sim, mas com poucas limitações	Desconhecido	Desconhecido
Simple	Apache 2.0	Não	Sim

Tabela 2.3: Exemplos de implementações [7], [8], [9], [10], [11] e [12].

Podemos concluir que a validação e processamento de ficheiro **XML** não se resume apenas ao uso de **XML Schema**. As soluções disponíveis são muito diversas, apresentando características próprias que lhes dão vantagens e desvantagens que devem ser ponderadas durante a fase de planeamento de um projeto, sendo, por isso, necessário encontrar o equilíbrio entre a simplicidade de uso e o poder expressivo, desejados no sistema.

Em jeito de resumo, apresentamos a figura 2.5 com a comparação entre os vários modelos, tendo em conta a facilidade de uso e o poder expressivo de cada modelo.

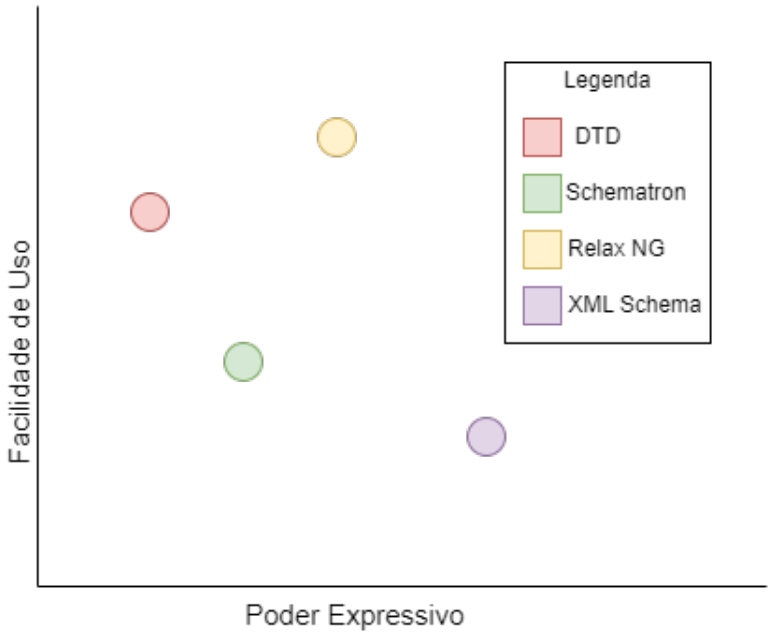

Figura 2.5: Plano de execução. [3]

Como podemos observar, existe já no mercado um conjunto de tecnologias que dão suporte à definição de regras as quais permitem validar a estrutura de um documento **XML**. O **GASPRO**, por ser uma aplicação que possibilita a validação e processamento de ficheiros, torna a adoção do **XML Schema** quase que obrigatória, mantendo, assim, a opção utilizada no passado. Isto é justificável pelo facto do **XSD** oferecer um enorme poder e flexibilidade nas regras definidas, tornando o processo de validação a nível aplicacional mais simples. Para além disso, o **XML Schema** é um dos formatos *standard* adotado pela grande maioria

das aplicações para troca de largos volumes de dados, o que contribuiu também para a sua adoção.

Como foi referido, os ficheiros **XML**, tal como a sua validação, usando ficheiros **XSD**, são standards muito utilizados na indústria de desenvolvimento de *software*, facto que criou uma enorme necessidade de tornar o processo de validação de *Schemas* um procedimento eficiente. Como consequência, sugiram, no mercado, uma panóplia de soluções que tentam otimizar estas validações. Muitas das soluções para o processamento e validação de ficheiros **XML** são já disponibilizadas pelas próprias linguagens e *frameworks*, como é o caso do **Java** e da *framework Spring* com a disponibilização de bibliotecas para validação de ficheiros, usando o **XML Schema**.

No caso concreto do **Java**, é disponibilizada a biblioteca **javax.xml**, que oferece diversas operações sobre ficheiros **XML**. Assim, operações como parsing e validação por meio de um ficheiro **XSD** podem ser realizados de uma forma simples e rápida.

É muito importante referir que o **XSD** apenas permite definir a estrutura de um ficheiro **XML**. O **XSD** deve ter todo o tipo de validações que envolva validação do conteúdo de determinados campos que está fora do escopo do **XSD**, sendo essas validações da total responsabilidade do **GASPRO**. Por esse motivo, é necessário implementar um motor de validação adicional que execute as regras com base nos tipos definidos no ficheiro **XSD**.

Capítulo 3

Planificação

3.1 Introdução

Neste capítulo, iremos descrever resumidamente o que foi feito na primeira etapa de desenvolvimento – a realização da Formação de *Spring* (secção 3.2). Finalizada esta fase e posteriormente à mesma, foi iniciado o período de conhecimento e integração com a aplicação **GASPRO**. Seguidamente, foram definidas as tarefas a realizar (secção 3.3). Por fim, definimos as etapas a realizar durante o estágio e, posteriormente, especificamos o plano de execução (secção 3.4).

3.2 Formação de *Spring*

A formação *Spring* permitiu alargar os conhecimentos já adquiridos por nós anteriormente através dos projetos que temos vindo a desenvolver em contexto profissional. Esta formação, além dos conteúdos teóricos, também consistiu numa componente prática, onde houve oportunidade para concretizar o conteúdo aprendido. Foram abordados conceitos do *Spring*, como o tema *Spring Bean* que proporcionou o tratamento de vários tópicos como, por exemplo, o ciclo de vida dos objetos, os vários *beans scopes*, os *múltiplos contexts*, entres outros pontos e os outros temas como a injeção de dependência baseada em anotações, bem como o AOP (*Aspect Oriented Programming with Spring*), os vários aspetos dos acessos a base de dados e, por fim, o *Spring Tests*. Em suma, foram temas muito enriquecedores, essenciais e necessários a nível dos conceitos do *Spring*, que se revelaram de suprema importância para o desenvolvimento deste projeto.

3.3 Enumeração das Tarefas

Iniciou-se a parte de definição das etapas a realizar futuramente, de modo a executá-las iterativa e sequencialmente. Para efetuar a enumeração foi necessário determinar aspetos fulcrais do conhecimento e integração da aplicação *GASPRO*, de modo a permitir uma perceção dos pontos que necessitam de ser alterados e a criação do novo módulo de automação. As tarefas foram crias levamos em conta a um modelo de desenvolvimento que encaixa na metodologia de trabalho que nos utilizamos na empresa, neste caso utilizados as **Metodologia Ágil**. De seguida, são mencionadas e explicadas as tarefas realizadas durante o estágio.

Conceção

1. Definir o âmbito do problema.
2. Identificar os requisitos e *stakeholders*.
   Nesta tarefa, apresentamos os diferentes *stakeholders* e os requisitos do projeto, assim como os principais problemas por eles identificados e que devem ser aplicados à solução a implementar.
3. Produzir o documento de Visão.

Elaboração

1. Investigar e Desenho.
   Nesta tarefa bastante importante será feito um reconhecimento das funcionalidades da aplicação **GASPRO** e o modo como estão desenvolvidas, permitindo, deste modo, uma maior facilidade na continuação do desenvolvimento e melhoria desta alteração ao modelo de dados;
2. Documentação de especificação de requisitos.

Construção

1. Desenvolvimento.
   Nesta tarefa, concretizamos a arquitetura da solução, implementando as alterações na aplicação **GASPRO** que permitem atingir os objetivos propostos;
2. Testes unitários e Testes integrados no sistema.

Transição

1. Revisão da Documentação do projeto.
2. Acompanhamento de entrada em produção.

As tarefas vão ser realizadas com ***scrum***, aplicando-se a metodologia acima referia.

3.4 Plano de Execução

O plano de execução que determinou a realização das tarefas anteriormente mencionadas foi realizado em conjunto com a Gestora/Orientadora de projeto da Opensoft, da melhor forma possível, para que existisse tempo suficiente para a finalização de cada etapa a ser concretizada, bem como a melhoria de aspetos não menos importantes, constituintes ou não das tarefas.

Através da Figura 3.1, que expomos seguidamente, é possível, então, verificar-se, de uma forma mais simples, o tempo que foi planeado para cada uma das tarefas definidas anteriormente, assim como o tempo total para a sua finalização.

Concluído o plano de execução, constituído pelas respetivas tarefas que foram enumeradas anteriormente, estavam reunidas todas as condições e orientações necessárias para dar início ao desenvolvimento.

Figura 3.1: Plano de execução.

3.5 Análise de Riscos e Plano de Mitigação

Nesta secção, é feita a identificação dos riscos que poderão acontecer no desenvolvimento do projeto de estágio, a probabilidade de ocorrência, mecanismos a efetuar para minimizar/-prevenir os riscos e, ainda, o que deve acontecer com este plano de mitigação.

Posteriormente, são, então, detalhados e enumerados os riscos e o respetivo plano de mitigação.

Risco 1 – Dispositivos de programação Para o desenvolvimento do projeto de estágio, é indispensável um dispositivo de programação (computador). A probabilidade de existir neste dispositivo algum problema é baixa, embora possa acontecer.

Sendo o computador da empresa, caso este risco ocorra, as medidas de mitigação a tomar passam pela comunicação imediata com a empresa, solicitando um computador de substituição temporária até que o anterior esteja funcional.

A ocorrência deste tipo de problema seria bastante prejudicial ao desenvolvimento do projeto. A medida de mitigação deste risco será acionada, com risco de avaria do dispositivo de programação (computador).

Risco 2 – Serviços

A aplicação utiliza alguns serviços, como, por exemplo, a Internet. Caso este serviço falhasse, não seria possível que os pedidos dos seus utilizadores se concretizassem. A probabilidade deste risco acontecer é média.

As medidas de mitigação, neste caso, passam, então, por utilizar serviços quase iguais, que permitam, de alguma forma, manter a normalidade das funcionalidades executáveis da aplicação. Estas medidas serão ativadas, caso algum serviço não funcione, ou funcione apenas parcialmente, uma vez que os serviços são indispensáveis numa aplicação, se o seu funcionamento decorrer normalmente e sem qualquer tipo de falha, de modo a evitar problemas futuros.

Risco 3 – Ferramentas

As ferramentas que irão ser utilizadas no desenvolvimento do projeto poderão ficar indisponíveis temporariamente por um curto ou longo espaço de tempo. A probabilidade deste

risco acontecer é baixa.

Como medidas de mitigação para este risco, será feito sempre um backup de todo o projeto, salvaguardando, desta forma, os dados do projeto ligados a estas ferramentas ou recursos. Caso este risco surja, terão de ser substituídas as ferramentas utilizadas por outras que permitam executar as funcionalidades da mesma forma.

3.6 Conclusão

Terminado este capítulo, resta acrescentar que a Formação *Spring* foi uma boa forma de ampliar conhecimentos, sobretudo através das experiências com os principais recursos, que incluem configuração, acesso a dados, configuração automática, entre outros, proporcionando uma base sólida para criar aplicativos corporativos.

Nesta etapa da planificação, ficam também definidas as tarefas a realizar durante o desenvolvimento do projeto de estágio, bem como a sua breve descrição e o plano que vai ser seguido na execução das mesmas.

Por fim, apresenta-se uma análise dos possíveis riscos e o plano de mitigação correspondente, com a preparação do desenvolvimento a ser realizado durante o estágio de maneira equilibrada e bem estruturada, tornando possível a execução e implementação das tarefas da melhor forma no contexto da aplicação.

Capítulo 4

Metodologia, Arquitetura, Tecnologias e Ferramentas

4.1 Introdução

Neste capítulo, serão abordadas, na secção 4.2, a metodologia utilizada no âmbito do desenvolvimento deste projeto; na secção 4.3, abordaremos a arquitetura implementada; na secção 4.5, concentraremos todas as tecnologias utilizadas no âmbito do desenvolvimento, nomeadamente as linguagens de programação e ainda as ferramentas utilizadas na secção 4.4 e, por fim, uma breve conclusão (secção 4.6).

4.2 Metodologia Utilizada

Metodologias ***Ágeis*** constituem o método para o desenvolvimento de produtos que estão alinhados com os princípios descritos no Manifesto ***Ágile*** para o desenvolvimento de *software*. Estas metodologias ágeis são compostas por pequenos e frequentes entregáveis com algumas funcionalidades que acrescentem valor incremental ao produto. São pequenas equipas multifuncionais que permitem uma reação mais rápida e constante.

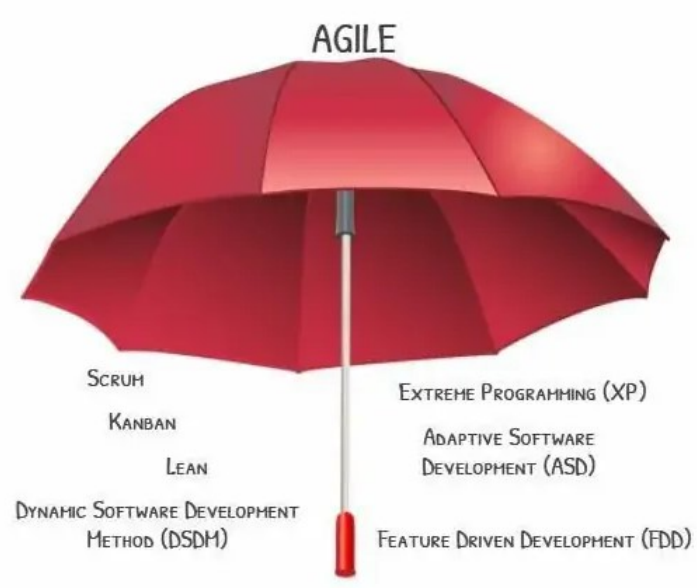

Figura 4.1: Metodologias Ágeis [4].

Esta metodologia ***Ágile***, como se pode ver na figura 4.1, tenta corrigir os desafios duma abordagem tradicional tendo em vista obter uma entrega final do produto bastante mais considerável, dado que, durante o seu período de desenvolvimento, os requisitos do cliente mu-

davam com alguma frequência. Este facto poderia levar a entregas erradas do produto por nem sempre poderem estarem de acordo com as exigências do cliente [4].

Das oito principais metodologias ***Ágeis***, nos utilizamos 2:

4.2.1 Scrum

Esta metodologia é uma ferramenta para trabalhar com projetos e foi criada por *Ken Schwaber* e *Jeff Sutherland*. As metodologias Ágeis de desenvolvimento de software são mini projetos, o que significa que o trabalho é dividido em pequenos projetos. O ***Scrum*** é constituído por pequenas equipas de 7 a 9 pessoas e também incluiu um ***Scrum Master*** e ***Product Owner***.

Figura 4.2: Processo *Scrum* [4].

No ***Scrum*** [4], os projetos são divididos em ciclos que são chamados de ***Sprints***. O ***Sprint*** representa um intervalo de tempo no qual um conjunto de funcionalidades deve ser desenvolvido. Podem combinar-se vários ***sprints*** ou também se pode fazer a entrega de *software*/produto ao cliente no final de cada ***sprint***, como se pode ver na figura 4.2.

A metodologia ***Scrum*** é caracterizada por fases específicas, tais como a reunião ***Daily Meeting***; a reunião ***Sprint Review***, que é a Demonstração ao ***Product Owner*** do *software*/produto para a entrega; a reunião ***Sprint Retrospective*** e a ***reunião Sprint Planning***. Todas estas reuniões proporcionam um trabalho de colaboração e oportunidades de revisão que permitem assegurar que este método se desenvolve e progride como pretendido, e também garantem a resolução de quaisquer questões rapidamente.

Nós utilizamos o ***Scrum*** para projetos de desenvolvimentos ou evolutivas, que costumam ser os projetos mais complexos.

4.2.2 Kanban

Kanban é um vocábulo de origem japonesa e significa ”quadro de sinal” ou ”sinal visual”, como se pode ver na figura 4.3. É um conceito que foi desenvolvido e aplicado em empresas de fabricação em série, como a *Toyota*, no ano de 1940. A produção é baseada na tarefa do cliente, em vez da prática padrão de produzir certas quantidades de mercadorias e entregá-las no mercado. O seu propósito fulcral é minimizar as atividades, sem sacrificar a

produtividade. O objetivo principal é criar mais valor para o cliente, sem gerar mais custos [4].

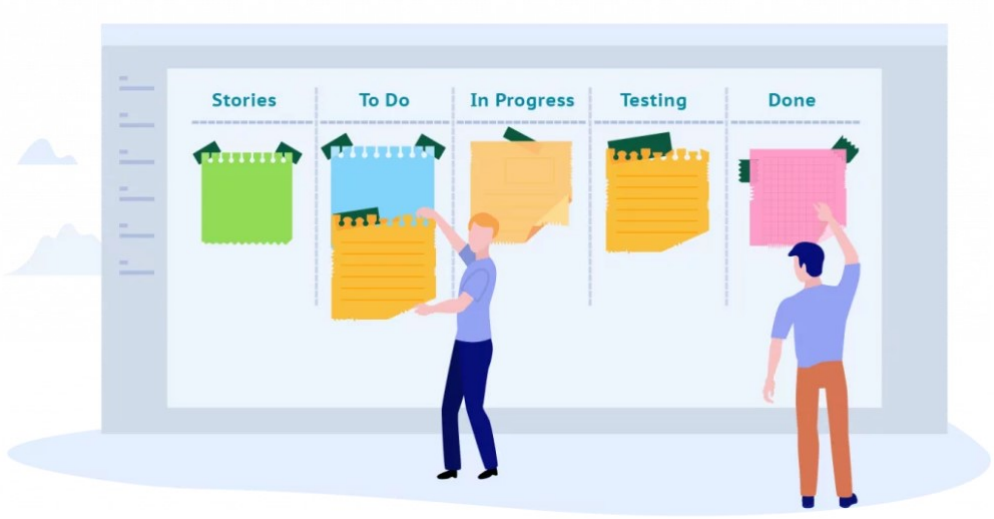

Figura 4.3: Processo *Kanban* [4].

A metodologia ***Kanban*** pode ser resumida como uma forma de gerir os processos que envolvem as equipas de um projeto, priorizando as tarefas e tornando o foco bem definido para todos. Desta maneira, é possível identificar e resolver problemas no fluxo de trabalho.

Assim, o ***Kanban*** tem como objetivo a evolução dos fluxos de trabalho, introduzindo restrições ao processo para otimizar o fluxo de valor. Basicamente, institui-se um processo incremental para mudança de processos e sistemas nas empresas. Deste modo, são promovidas pequenas alterações e melhorias no processo, de forma a criar uma menor resistência à mudança e facilitando a evolução das equipas.

Utilizamos o ***Kanban*** para tarefas de correção ou manutenção, que costumam ser tarefas menos complexas.

4.3 Arquitetura Utilizadas

4.3.1 Arquitetura *MVC (Model-View-Controller)*

Este padrão de arquitetura software foi criada nos anos 80 pela *Xerox Parc*, por *Trygve Reenskaug*. Focaliza-se na reutilização de um código e na separação de conceitos em três camadas interligadas. A arquitetura tornou-se popular para projetar aplicações web e até mesmo para aplicações móveis, para *desktop* e para outros clientes. Linguagens de programação populares como **Java**, **C#**, **Object Pascal/Delphi**, **Ruby**, **PHP**, **JavaScript** e outras possuem ***frameworks MVC*** que são usadas no desenvolvimento de aplicações ***web*** [50].

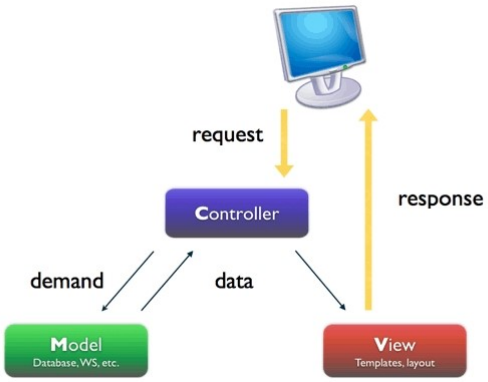

Figura 4.4: Daigrama *MVC* [5].

Esta arquitetura divide-se em três tipos de componentes. O desenho **MVC** define as interações entre eles, como se pode ver na figura 4.4.

- ***Model*** - Esta camada é responsável por gerir e controlar a forma como os dados se comportam através das funções, da lógica e das regras de negócios estabelecidas.
- ***Controller*** - Esta camada é responsável por intermediar as requisições enviadas pelo ***View*** com as respostas fornecidas pelo Model.
- ***View*** - Esta camada tem a responsabilidade de apresentar as informações de forma visual ao utilizador.

4.4 Ferramentas

4.4.1 *IntelliJ IDEA*

O ***intellij*** é um *Integrated Development Environment* (*IDE*), para o desenvolvimento de software na plataforma ***java***, ***Kotlin***, ***Groovy*** e outras linguagens baseado em *JVM(Java Virtual Machine)*. São desenvolvidas pela *JetBrains* e têm uma edição com licença ***Apache 2 Licensed community*** e outra edição comercial.

A primeira versão foi lançada em janeiro de 2001 e foi um dos primeiros *IDEs Java* disponíveis com recursos avançados e refaturação de códigos integrados. Em 2009, a *JetBrains* lançou o código-fonte do ***IntelliJ IDEA*** sob a licença *Apache 2.0* de código aberto.

No relatório *InfoWorld* de 2010, o ***IntelliJ*** recebeu a pontuação mais alta do centro de testes entre as quatro principais ferramentas de programação *Java* que são *Eclipse*, *IntelliJ IDEA*, *NetBeans* e *JDeveloper* .

Em dezembro de 2014, o Google anunciou a versão 1.0 do *Android Studio*, um *IDE* de código aberto para desenvolvimento de aplicativos *Android*, baseado na edição da comunidade de código aberto [51].

4.4.2 *SoupUI*

SoupUI é uma ferramenta de código aberto escrita em ***Java*** cuja principal função é consumir e testar *Web Services(WS)* com os protocolo *Simple Object Access Protocol(SOAP)* e *Representational State Transfers(REST)*. Esta ferramenta serve para importar e gerar automaticamente as solicitações descritas na *Web Services Description Language (WSDL)*; serve para validar as requisições e respostas definidas no *WSDL*; serve também para realizar testes funcionais, testes de carga e testes de stress e ainda muitas outras funcionalidades [52].

4.4.3 *PowerDesigner*

Esta ferramenta de modelagem é produzida pela ***Sybase***, que é atualmente da ***SAP***. Ela oferece um suporte ao design de software de *model-driven architecture* e armazena modelos usados. Também pode armazenar modelos num arquivo de banco de dados.

Com a modelagem de dados, *link* e sincronização e gestão de metadados, é possível capturar imediatamente camadas e requisitos de arquitetura, aceder a um poderoso repositório de metadados e compartilhá-lo com as equipas de desenvolvimento [53].

4.4.4 *SonarQube*

SonarQube é uma ferramenta de análise estática que assegura a qualidade do código que está a ser desenvolvido. Ela efetua diversas análises durante o processo de compilação da aplicação para, por exemplo:

- detetar *Bugs* gerados durante o desenvolvimento;
- prevenir a repetição de código desenvolvido;
- identificar questões de segurança.

As análises resultantes desta aplicação atendem às métricas de qualidade configuradas na própria ferramenta, seguindo algumas configurações padronizadas. No entanto, neste caso, as configurações das métricas são de responsabilidade do cliente [54] (AT).

4.5 Tecnologias

4.5.1 *Java*

O ***Java*** é uma linguagem de programação orientada para objetos, incrementada na década de 90, que foi desenvolvida por um grupo de programadores, tendo como membro principal *James Gosling*, na empresa *Sun Microsystems*, sendo atualmente, parte da empresa **Oracle**.

Esta linguagem é compilada para um *bytecode*, que é executado numa máquina virtual, com benefícios de portabilidade, ou seja, o mesmo código pode ser executado em diferentes dispositivos, desde que estejam suportados por uma **máquina virtual *Java*** [55].

4.5.2 *XML*

O **XML** (*Extensible Markup Language*) é uma linguagem de marcação que deriva de outra linguagem **SGML** (*Standard Generalized Markup Language*), através da criação de um documento com uma estrutura organizada hierarquicamente para ser utilizada em vários sistemas. O seu objetivo é representar objetos num formato comum para garantir uma interpretação única [56].

4.5.3 *SQL*

O ***SQL*** (*Structured Query Language*) é uma linguagem de pesquisa padrão para base de dados. Foi desenvolvido originalmente nos anos 70 pela **IBM**, que tem o propósito de demonstrar a viabilidade da implementação do modelo relacional. Em 1986, o *American National Standards Institute* (*ANSI*) tentou normalizar a linguagem ***SQL***, devido ao facto de, entretanto, terem surgido novas linguagens de base de dados e, em 1987, o padrão foi aprimorado várias vezes com funcionalidades e recursos adicionais pela *International Organization for Standard*(*ISO*) [57].

4.5.4 *HTML*

O **HTML** (HyperText Markup Language) é uma linguagem de marcação utilizada para o desenvolvimento de páginas na web. Os documentos **HTML** podem ser interpretados por todos os browsers [58].

4.5.5 *JUnit*

O **JUnit** é um *framework* de código aberto, criado por *Erich Gamma* e *Kent Beck*, com suporte à criação de testes automatizados na linguagem de programação **Java**. O programador tem a possibilidade de utilizar um instrumento para criar um modelo padrão de testes [59].

4.5.6 *Spring*

O ***Spring***(4.5) é um *framework* de código aberto para a plataforma **Java**, criado por *Rod Johnson*, com o propósito de facilitar o desenvolvimento de aplicações, baseado nos padrões de projeto Inversão de controle (*IoC - Inversion of Control*) e injeção de dependências. Esta *framework* oferece uma variedade de recursos indispensáveis numa grande parte de aplicações através de módulos para persistência de dados, integração, segurança, testes, desenvolvimento *web*, e também permite criar soluções mais coesas e, consequentemente, mais fáceis de compreender e manter [6].

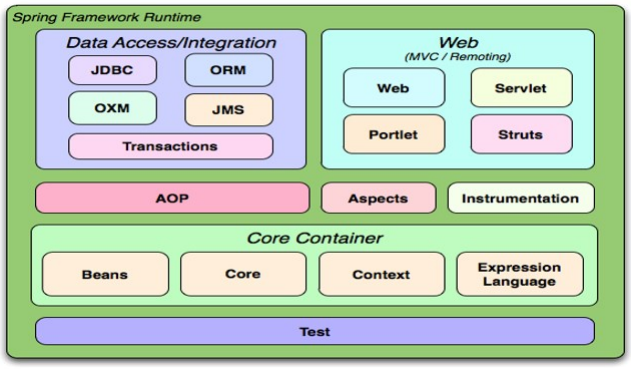

Figura 4.5: *Spring Framework* [6].

4.5.7 *Mockito*

O ***Mockito*** é uma ferramenta de teste de código aberto para o **Java**, tendo como principal finalidade a criação de instâncias de classes e o controle do comportamento de seus métodos. Isso possibilita que, ao simular a dependência de uma classe, a classe que está a ser testada acredite que está a invocar o método real, quando, na realidade, está a invocar o *mock* [60].

4.5.8 *Apache Maven*

Maven é uma tecnologia de automação de compilação, que é utilizada nos projeto de **Java**. Esta tecnologia, muito parecido com a tecnologia *Ant*, também é usada para a construção e gestão de projetos com outra linguagem de programação como **C#**, **Ruby Scala** e outras. Este projeto, que fazia parte do projeto *Jakarta*, é atualmente controlado pela *Apache Software Foundation*.

O **Maven** utiliza um ficheiro **XML** (**POM**) para descrever a construção do software e as suas dependências e componentes externos, bem como a ordem da compilação, diretórios e *plug-ins* necessários para o projeto. O ***Maven*** faz o download das dependências indicadas no arquivo **XML** do *Projeto Object Model* (*POM*), de forma dinâmica, a partir de um ou mais repositórios remotos.

O **POM** é o arquivo de configuração central do ***Maven***, que especifica as informações do projeto, incluindo as dependências necessárias e guarda-as numa área de *cache* local chamada *"Maven Repository"*, no seu diretório do computador.

Em suma, o ***Maven*** faz o *download* dinâmico das dependências com base no arquivo **POM**, consulta os repositórios remotos e armazena-os num *cache* local para uso posterior, simplificando, assim, a gestão de dependências e garantindo a reprodutibilidade e a consistência nas construções do projeto [61].

4.6 Conclusão

A construção de uma aplicação *web* em **Java** com várias tecnologias envolvidas é um projeto complexo, mas compensador. Através da utilização dessas tecnologias, é possível criar uma aplicação robusta, e com um bom desempenho.

O **Java**, como uma linguagem de programação, oferece uma ampla gama de recursos e bibliotecas que facilitam o desenvolvimento de aplicações web. Através do **Java**, é possível implementar a lógica de negócios da aplicação, manipular dados, gerenciar a segurança e interagir com outras tecnologias.

Além do **Java**, várias outras tecnologias são utilizadas no desenvolvimento de aplicações web que fornecem ferramentas poderosas para simplificar o desenvolvimento e melhorar a produtividade. Esses *frameworks* oferecem recursos como injeção de dependência, autenticação e autorização, entre outros.

A utilização dessas várias tecnologias permite que o desenvolvimento de uma aplicação *web* em **Java** seja flexível e adaptável às necessidades do projeto.

No geral, a construção de uma aplicação *web* em **Java** com várias tecnologias envolvidas é um processo desafiador, mas altamente viável. Com as ferramentas e linguagens disponíveis, é possível criar uma aplicação *web* poderosa e completa, capaz de atender às questões dos utilizadores e fornecer uma experiência agradável e eficiente.

Capítulo 5

Implementação

5.1 Introdução

Neste capítulo, serão detalhados os passos efetuados na direção pretendida, isto é, com vista a uma melhoria e flexibilidade de uma aplicação já desenvolvida.

Serão explicados e expostos na secção 5.3, alguns dos resultados obtidos no final da aplicação, bem como os métodos utilizados para chegar até aos mesmos e os motivos que levaram à escolha do método de implementação que irá ser especificada na secção 5.2.

Na secção 5.4 é feita uma breve conclusão acerca deste capítulo.

5.2 Implementação

Esta secção mostra as implementações realizadas no que diz respeito às melhorias que foram desenvolvidas e integradas na aplicação para a elaboração das etapas ou tarefas definidas e desenvolvidas na secção 3.3.

Antes de ser iniciada a explicação das tarefas planeadas e mencionadas anteriormente, é importante referir que o projeto de estágio previa fazer uma evolução do atual processo de classificação de combustíveis, de forma a adotar um modelo automático e flexível de categorização dos abastecimentos.

No entanto, houve necessidade de acrescentar outros aspetos relacionados com a realidade do mercado de trabalho, para complementar este estágio.

Assim, os itens que se adicionam centram-se na possibilidade da aplicação **GASPROIntra**. Os funcionários da DSIECIV (Direção de Serviços dos Impostos Especiais de Consumo e do Imposto sobre Veículos) têm o poder adicionar, alterar e remover as autorizações de abastecimento em postos privados que não pertençam ao adquirente para abastecerem os seus veículos de passageiros nesse postos privados e no processamento de ficheiro de abastecimento. Neste sentido, há necessidade adicionar uma validação para verificar se aquele adquirente tem autorização para abastecer naqueles posto de abastecimento privado.

5.2.1 Conceção

Tal como foi descrito na secção, 3.3, a primeira tarefa a realizar consistiu em definir o âmbito do problema, identificar os requisitos e *stakeholders*.

Descrição do Problema

- **O problema**
  O sistema atual do regime de gasóleo profissional é muito estático no que diz respeito

à parametrização do tipo de empresas, da diversidade de combustíveis e das especificações de veículos.

- **Afeta**
  Todos os adquirentes licenciados, como empresas de transporte de mercadorias e de passageiros, bem como os emitentes de gasóleo abrangidos neste regime, tanto públicos como titulares de instalações de consumo próprio;
  Funcionários da DSIECIV que pretendem monitorizar, controlar, acompanhar e dar apoio a estes utilizadores.

- **O que origina**
  No Orçamento de Estado de 2023, podemos observar orientações políticas no sentido de expandir o uso de outros combustíveis [14].
  Neste contexto, percebemos que, uma vez que a aplicação deixa de ser exclusivamente para uso de gasóleo profissional, como é o caso atualmente, surgem dificuldades na modificação das parametrizações, ou até mesmo na adição de novos tipos de CAE (Classificação Portuguesa das Atividades Económicas), combustíveis e outros parâmetros essenciais para o processamento dos arquivos de combustíveis.
  Para efetuar tais alterações, é necessário editar o código e, posteriormente, solicitar a instalação de uma nova versão da aplicação ao cliente. Aproveitamos, assim, essa oportunidade para realizar melhorias e tornar a aplicação mais dinâmica, facilitando a inclusão de futuras alterações semelhantes que possam surgir.

As Restrições são:

1. Ambientes de desenvolvimento, qualidade e produção.

   (a) A plataforma de servidor é definido por:

      i. Java 1.8
      ii. Weblogic Application Server 12.2.1.4
      iii. Oracle v12c ou superior

5.2.2 Elaboração

Esta secção é uma das mais importantes, antes do início do desenvolvimento, porque nesta fase é feita a investigação e a produção dos desenhos do Modelo de Dados e os desenhos de possíveis soluções para o desenvolvimento final.

A investigação foi realizado em duas fases:

1. Conhecer a aplicação tal como está desenvolvida até a data presente;

2. Conhecer o Modelo de dados aprovado pela AT e a data deste desenvolvimento.

3. Investigar a melhor solução para o desenvolvimento que se reflete nessa investigação no capítulo [2]: Estado de Arte.

No desenho, foram realizados vários esboços para as alterações necessárias nos ecrãs que precisam de ser modificados ou até mesmo criar novos ecrãs.

No caso das alterações ao Modelo de Dados, houve várias evoluções até se chegar à versão definitiva.

5.2.3 Construção

Nesta fase, traçam-se os desenvolvimentos que foram planeados no início do estágio e outros novos que apareceram durante o estágio.

5.2.3.1 Geração do Processo *Batch*

Nesta secção, serão apresentadas as várias soluções propostas e implementadas durante a fase de desenvolvimento deste processo, os problemas encontrados durante o desenvolvimento e as soluções alternativas exploradas.

O objetivo é, assim, dar uma visão cronológica dos factos, capaz de satisfazer os requisitos definidos no arranque do estágio.

5.2.3.2 *Spring Batch*

O ***Spring Batch*** é uma *framework* desenhada para o processamento de largos volumes de dados, oferecendo um conjunto de abstrações que permitem a implementação de processo *batch* robusto. Funcionalidades como *logging*, *tracing*, gestão de transações já são asseguradas pela *framework*, assim como a possibilidade de configurar o início, o fim e a reinicialização de processos. A *framework* permite ainda a definição de estratégias de recuperação de eventuais erros, podendo estas resultar numa nova tentativa ou num salto (*skip*) para o elemento seguinte.

Existem dois fatores essenciais que permitem ao ***Spring Batch*** oferecer elevada eficiência no processamento de elementos, sendo estes a capacidade de realizar operações *batch* com a base de dados e a facilidade de introduzir processamento paralelo em passos particulares.

A realização de operações em modo *batch* consiste em dividir os dados processados em blocos de informação de um tamanho predefinido, o que permite que estes sejam preservados através de uma única comunicação com a base de dados.

Note-se que o ***Spring Batch*** é uma *framework* complexa e com um amplo conjunto de funcionalidades que vão além das apresentadas nesta secção, na qual serão apenas cobertos os conceitos e funcionalidades utilizadas durante o desenvolvimento. Deste modo, caso se pretenda uma explicação mais detalhada da *framework*, recomenda-se a leitura da sua documentação oficial[1].

[1] [62], [63]

5.2.3.3 Arquitetura *Spring Batch*

O ***Spring Batch*** apresenta uma arquitetura bastante simples e intuitiva, sendo a sua unidade de trabalho de maior complexidade designada por ***Job***, sendo um ***Job*** composto por um ou mais ***Steps***.

Job

Uma execução de um processo *batch* corresponde à execução de um ***Job***, que pode ser composto por um conjunto de um ou mais ***Steps***, podendo estes ***Steps*** ser executados de forma sequencial ou paralela.

Para que o ***Job*** seja executado é necessário criar uma instância do mesmo, sendo este processo realizado pelo *JobLauncher*. O *JobLauncher* irá, então, criar um *JobIntance* baseado em *JobParameters*, uma configuração que contém informação de *schedule* do processo *batch*. Um *JobInstance* é ainda composto por *JobExecution*, isto porque uma instância pode ser executada mais de uma vez devido a erros e/ou estratégias de recuperação.

A Figura seguinte mostra sobre a forma de esquema a relação entre os vários conceitos apresentados.

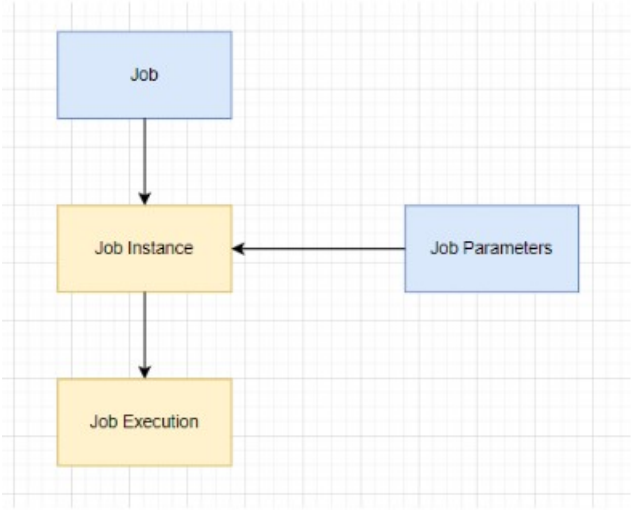

Figura 5.1: Estrutura de um *Job*.

Step

Como referido anteriormente, um *Job* é constituído por um ou mais ***Steps***. Cada ***Step*** é constituído por três componentes principais: uma unidade de leitura capaz de gerar e retornar objetos; uma unidade de processamento que recebe os objetos retornados pelo leitor, realiza o seu processamento e retorna um objeto já processado e, por fim, uma unidade de escrita que recebe elementos processados e efetua a sua escrita, normalmente para um ficheiro ou base de dados.

Esta arquitetura tripartida permite segregar de forma fácil as fases necessárias ao processo de integração, isolando os vários erros que podem surgir em cada uma das componentes. A Figura seguinte mostra a arquitetura global de um ***Step*** e a sua decomposição nas três componentes acima descritas[2].

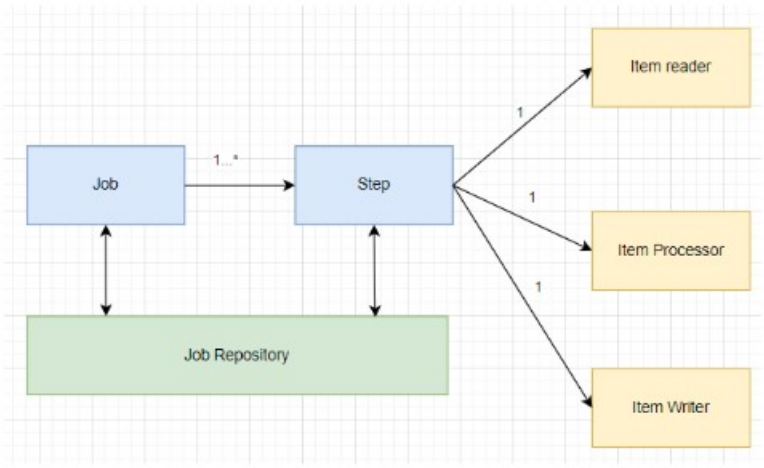

Figura 5.2: Estrutura de um *Step*.

5.2.3.4 Processamento Orientado ao Bloco

A execução de um *Step* pode ser realizada de dois modos distintos, que se podem dividir em execução orientada ao bloco (*Chunk Oriented Processing*) e *TaskletStep*. Podemos pensar no processamento em blocos como um processo dividido em três fases distintas que incluem uma fase de leitura, uma fase de processamento e uma fase de escrita, sendo a passagem dos dados entre cada uma das fases realizada em blocos de um tamanho definido à partida. Esta categoria de processamento deve ser utilizada quando o número de operações de escrita para a base de dados é muito elevado, isto porque o mecanismo de transações subjacente a esta arquitetura favorece o armazenamento em modo *batch*, garantindo que o número de comunicações com a base de dados é reduzido ao mínimo indispensável.

Durante o processamento de um bloco, todos os elementos persistentes durante a fase de escrita são enviados para a base de dados após o término do bloco, sendo, posteriormente, enviados para a base de dados numa única operação, permitindo uma enorme redução no tempo de execução do processo de integração.

O facto de o processamento ser realizado em blocos facilita o desenvolvimento de processos de recuperação, dado que, após o *commit* de um bloco para a base de dados, podemos

[2] [62], [63]

assumir que este foi guardado na base de dados. Esta informação permite determinar em que ponto o processo foi interrompido e, deste modo, retomar o processamento no último ponto estável, evitando que tenhamos que recomeçar o processo do ponto inicial [64].

5.2.3.4.1 *TaskletStep*

O ***TaskletStep*** representa uma categoria particular de *Step*, na qual a estrutura de leitura, processamento e escrita são substituídas por uma execução única, que contém apenas um método execute capaz de despoletar a execução do código pretendido.

A execução de um ***TaskletStep*** é também realizada dentro de uma transação à semelhança do que acontece com o processamento em blocos, sendo executado apenas um *commit* no final da execução do *Step* [64].

5.2.3.4.2 *Spring Batch e Parser XML*

De modo a conseguir adaptar o processo de integração já implementado à arquitetura da *framework*, houve a necessidade de converter o *push model* do *parser* para um *pull mode*, isto porque é necessário ter um controlo do fluxo de execução do *parser* que nos permita avançar para o próximo elemento de forma manual, quando desejado, o que não é possível no *push model* implementado pelo **SAX parser**. Assim, foi necessário adotar um modelo de processamento de **XML** que seguisse um modelo **StAX parser**, o que envolve compreender e implementar.

5.2.3.4.2.1 *StAX Parser - Modos de Execução*

Quando comparamos a estrutura de um ficheiro **XML** com um modelo de dados relacional, podemos observar que ambos têm aspetos bastante comuns, que permitem um processo de tradução relativamente simples.

Destas características podemos salientar as seguintes:

- Em ambos os modelos existe o conceito de relação entre pai e filho, no **XML** definida via aninhamento de elementos e no modelo relacional a relação é definida via definição de chaves estrangeiras.
- Possibilidade de criar relações entre elementos de diferentes cardinalidades, em particular cardinalidade 1-1 e 1-N.

Contudo, apesar de ambos os modelos parecerem encaixar de forma perfeita, tal não acontece na prática, facto este que se prende essencialmente com o uso de relações fortes, característico do modelo relacional.

Para entender melhor o problema, considere-se o ficheiro **XML** apresentado no código em baixo.

```
<FicheiroEmpresa>
    <Header>
        <IDEmpresa>509960910</IDEmpresa>
        <NumeroRegisto>509960910</NumeroRegisto>
        <TipoConteudoFicheiro>F</TipoConteudoFicheiro>
```

```
        <NomeEmpresa>100Limpar</NomeEmpresa>
        <MoradaEmpresa>
            <Rua>Rua XXXX</Rua>
            <Cidade>Viseu</Cidade>
            <CodigoPostal>5600-123</CodigoPostal>
            <Pais>PT</Pais>
        </MoradaEmpresa>
        <AnoFiscal>2023</AnoFiscal>
        <DataInicio>20213-01-01</DataInicio>
        <DataFim>2023-12-31</DataFim>
    </Header>
</FicheiroEmpresa>
```

Listagem 5.1: Definição do elemento FicheiroEmpresa

Após o processamento do ficheiro presente no código 5.1, seria desejável reproduzir as várias relações num modelo relacional, gerando um resultado semelhante ao apresentado na Figura 5.3.

Observando o resultado produzido, podemos afirmar que o elemento pai, independentemente da sua posição no ficheiro, deve ser sempre perseverado na base de dados antes dos seus filhos, isto porque, caso esta condição não se verifique, o próprio sistema de base de dados produzirá uma exceção de violação de integridade da chave estrangeira, o que é normal e desejável num modelo relacional.

Assim, para garantir que estes erros não são produzidos, é necessário assegurar a ordenação entre os elementos *pai* e *filho*, o que, num cenário onde existem vários processos a executar de forma concorrente, envolve o desenvolvimento de processos de sincronia que têm um impacto significativo no tempo de execução.

Outra questão que surge no processo de integração está relacionada com a necessidade de o elemento pai ser sempre devolvido pelo parser antes dos seus filhos.

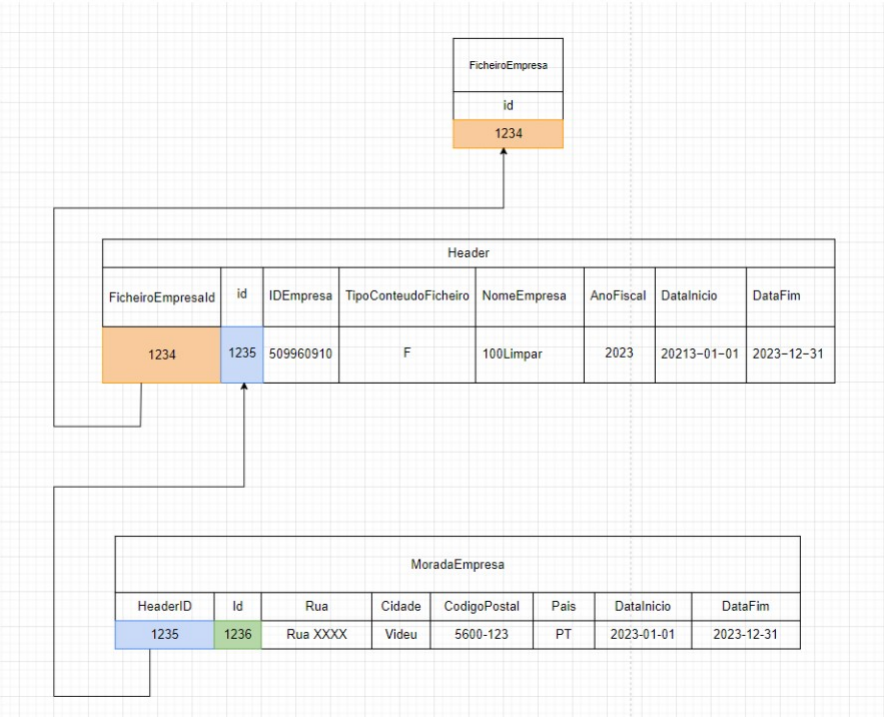

Figura 5.3: Mapeamento para modelo relacional do elemento FicheiroEmpresa.

Este passo, apesar de crucial para evitar quebras na integridade da base de dados, apresenta um grande desafio em cenário como o exibido no ficheiro **XML**, acima apresentado.

No caso exposto, entre os campos básicos do elemento *"Header"*, surge um elemento filho complexo MoradaEmpresa. Assim, é necessário que o elemento *Header*, ainda incompleto, seja retornado e persistido antes do seu filho, pelo que apenas uma das três opções abaixo pode ser considerada:

- *"Header"* é persistido antes do "MoradaEmpresa", sendo posteriormente atualizado após o processamento dos campos básicos em falta.
- Os elementos filhos complexos, caso existam, são sempre declarados após todos os campos básicos do elemento em questão. Por outras palavras, podemos considerar um elemento como preenchido assim que o primeiro elemento complexo filho for detetado.
- Relaxamento das restrições de integridade de modo a permitir uma ordem arbitrária no processamento de elementos.

Considerando os requisitos associados ao processo de integração, em particular o tempo de execução, podemos descartar, de forma imediata, a primeira opção sugerida, isto porque a persistência de um elemento completo na base de dados exige uma operação de inserção e uma de atualização, o que aumenta, de forma bastante considerável, o tempo de execução.

A segunda opção permite, ao contrário da primeira, que o elemento seja persistido numa única operação, o que reduz o tempo de execução de forma significativa.

A última opção é, de todas as anteriores, a que oferece mais liberdade e também maior eficiência no processamento de elementos, visto que a ordem de execução deixa de ser uma necessidade devido à existência de relações.

5.2.3.4.2.2 *StAX Parser - Processamento*

A primeira versão do ***StAX** parser* desenvolvida para o módulo de integração foi implementada de forma a garantir que o elemento pai é sempre retornado antes do elemento filho. Por este motivo, o processamento do ficheiro é realizado partindo do elemento raiz e descendo progressivamente na altura da árvore até se chegar aos elementos presentes nas folhas, sendo, por isso, designado como um **processamento** do ficheiro.

Esta categoria de processamento tem, contudo, um custo associado, pois obriga a que os elementos complexos presentes no ficheiro **XML** apresentem todos os seus campos simples antes dos elementos complexos (elementos filhos). Este requisito é fundamental para garantir que o elemento pai é sempre devolvido antes do filho, dado que só assim é possível afiançar que, ao chegar ao elemento filho, o seu pai já possui todos os campos preenchidos. Este facto permite que o armazenamento dos elementos possa ser realizado numa única operação de inserção da base de dados, ao invés de uma inserção seguida de uma atualização após o fecho do elemento.

A Figura abaixo ilustra sobre a forma de esquema o processamento efetuado pelo *parser* a cada chamada.

Analisando a figura apresentada, podemos observar que um elemento é considerado como preenchido quando é detetado o início de um elemento filho ou quando se deteta uma *tag* de fecho do elemento ativo, o que só é possível graças à assunção feita inicialmente.

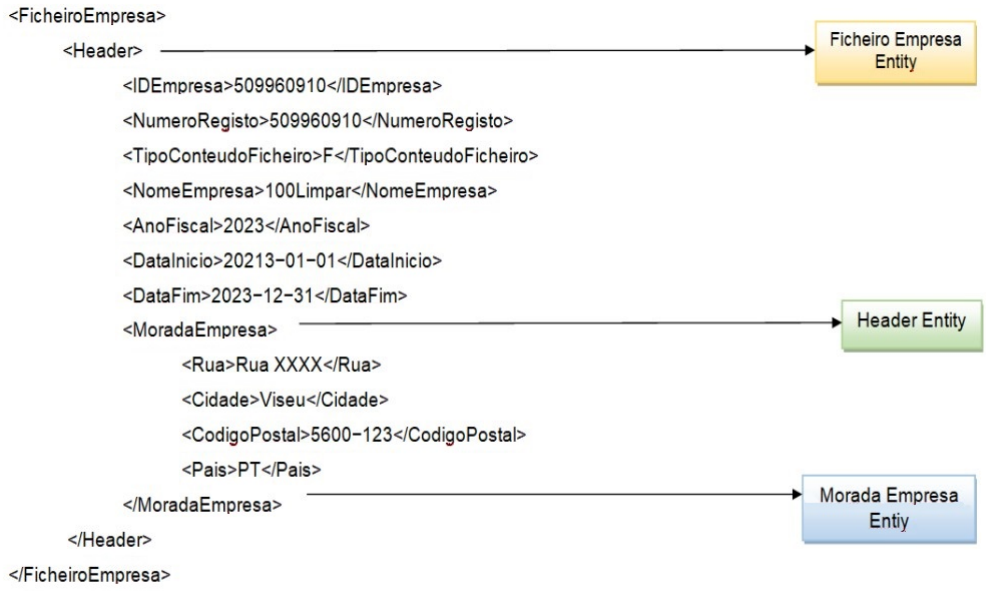

Figura 5.4: *StAX Parser* processamento.

5.2.3.4.2.3 Módulo de Integração em *Spring Batch*

A primeira versão do módulo de integração realizada com recurso à *framework* ***Spring Batch*** teve como principal processo de inserção os elementos extraídos para a base de dados.

A arquitetura existente consiste em realizar o processamento utilizando apenas um *Step*,

constituído por uma unidade de leitura, uma unidade de processamento e uma unidade de escrita. A fase de leitura é responsável pela extração dos elementos do **XML** com recurso a um ***StAX parser*** em modo de execução, que parte da raiz e termina nas folhas do ficheiro. Estes elementos são posteriormente submetidos a uma fase de processamento, onde são realizadas validações que permitem detetar erros ao nível do conteúdo do elemento. Caso o elemento seja válido, é passado à fase de escrita que é guardado na base de dados, resultando numa arquitetura semelhante à apresentada na Figura seguinte.

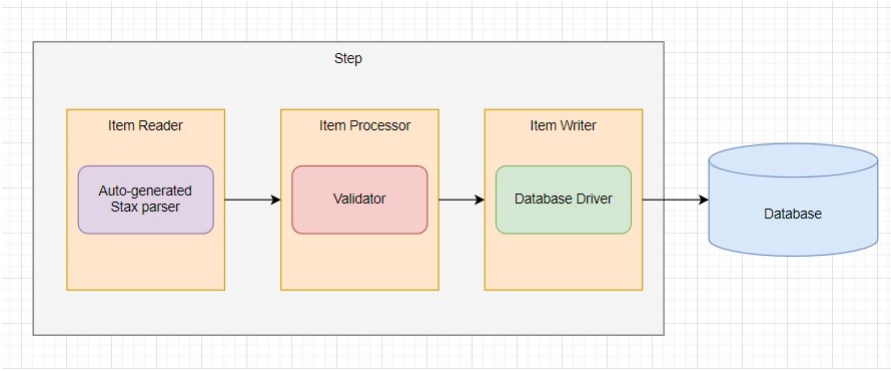

Figura 5.5: *Spring Batch* arquitetura inicial.

Um aspeto importante a ser considerado durante o desenvolvimento de uma aplicação em *Spring Batch* é que o mecanismo de paralelismo disponibilizado pela ferramenta é assegurado ao nível do *Step*. Na prática, um *Step* pode ser executado por um único fio de execução *(single-threaded)* ou por múltiplos fios *(multi-threaded)* de execução que executam em paralelo. O paralelismo garantido pela utilização de *Steps multi-threaded* traz um enorme ganho de performance, dado que o processamento de cada bloco *(chunk)* é processado por um único fio de execução, o que permite o processamento paralelo de vários segmentos do ficheiro, assim como processos de comunicação com a base de dados simultâneos. Observando a Figura 5.5, podemos pensar que, para paralelizar todo o processo de integração, bastaria apenas correr o *Step* utilizando múltiplos fios de execução. Contudo, esta solução não produz os efeitos desejados, dado que ao paralelizar todo o *Step* estamos também a paralisar a fase de leitura, o que, por consequência, não nos permite ter qualquer garantia em relação à ordem dos elementos presentes no ficheiro.

5.2.3.5 Modificações do Modelo de Dados

Um dos objetivos deste estágio é a evolução do atual processo de classificação de combustíveis, de forma a adotar um modelo automático e flexível de categorização dos abastecimentos. Essa evolução e a flexibilidade trouxeram várias alterações ao modelo de dados. Foram adicionadas novas tabelas para os tipos de combustíveis, tipo de veículos, categorias de veículos e códigos da atividade da empresa e também houve alterações em algumas tabelas que já existiam.

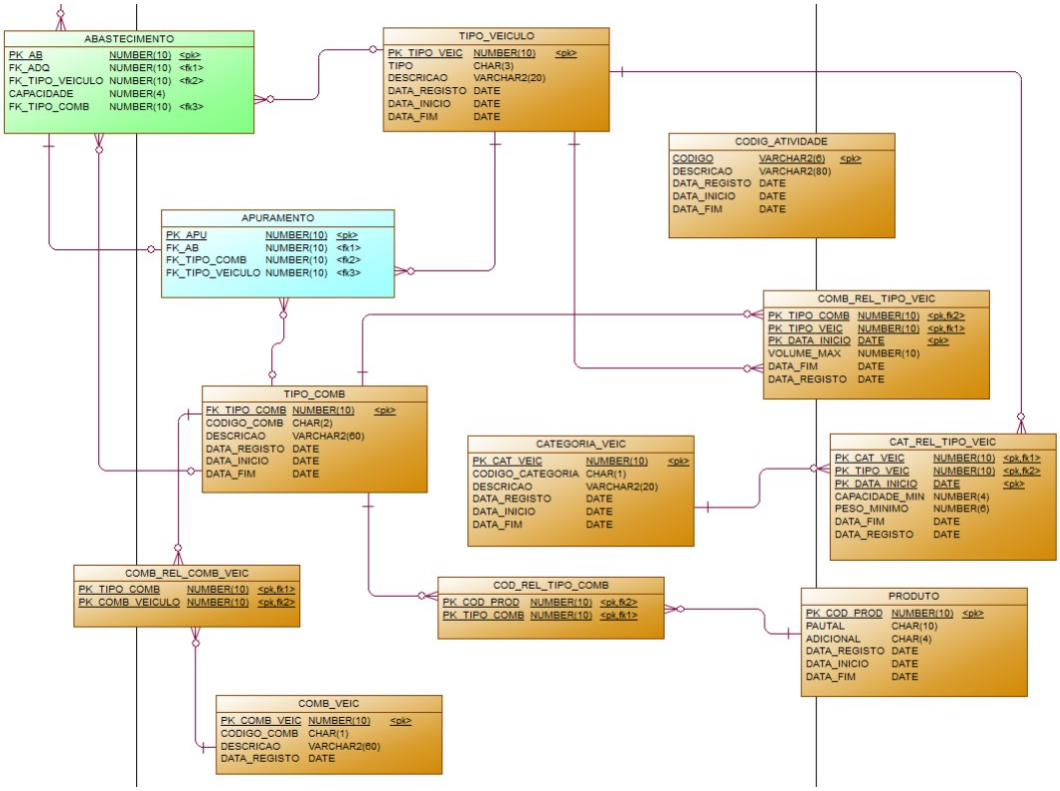

Figura 5.6: Alterações do Modelo de dados.

No desenvolvimento extra teve impacto o modelo de dados, na adição de uma tabela nova para guardar as matrículas e outra para relacionar o Adquirente e o Emitente.

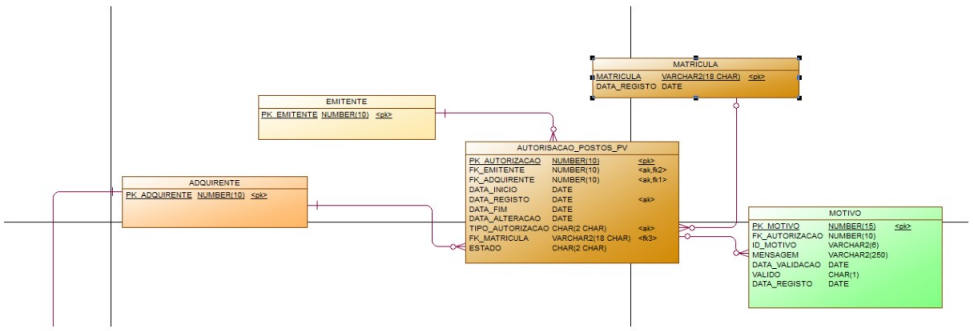

Figura 5.7: Alterações do Modelo de dados.

5.2.3.6 Modificações do Processamento Batch

Tendo em conta os objetivos principais do estágio, foram implementadas diversas mudanças no modelo de dados, sendo, por isso, necessário realizar algumas modificações nas validações.

As alterações nos processo validações de informação dos dados, provenientes dos dois *Web Services*, o de cadastro de veículos e o de contribuintes, foram realizadas. Algumas

validações já existentes passaram por reformulações, como o tipo de veículo, tipo de combustível, tipo de CAE e validação do peso mínimo nos veículos. Isso resultou na eliminação de grande parte do código referente à lógica estática.

Adicionalmente, foram introduzidas novas validações para verificar a lotação mínima em veículos de passageiros e garantir a correspondência entre o tipo de CAE e o tipo de veículo.

Além disso, foram implementadas novas funções com cache em todas as validações existentes. O uso dessas funções permite reduzir o número de comunicações com a base de dados, no entanto, a cache será aplicada somente durante o processamento de um arquivo e será renovada a cada novo ficheiro processado.

Devido aos novos desenvolvimentos que afetam a validação do adquirente de consumo próprio, as alterações nessa validação consistiram em adicionar uma condição adicional para verificar se o adquirente com veículo de passageiro possui autorização para abastecer naquele posto privado.

5.2.3.7 Modificações da aplicação GASPROIntra

As modificações realizadas na aplicação GASPROIntra durante o estágio foram impulsionadas pelos novos desenvolvimentos que foram adicionados. As alterações no modelo de dados têm impacto nos ecrãs que têm relação com os Emitentes.

Os ecrãs que passaram por modificações foram o ecrã de criação, o de alteração e o de consulta detalhada do Emitente. Essas alterações foram guiadas por requisitos específicos do cliente, conforme ilustrado nas imagens abaixo.

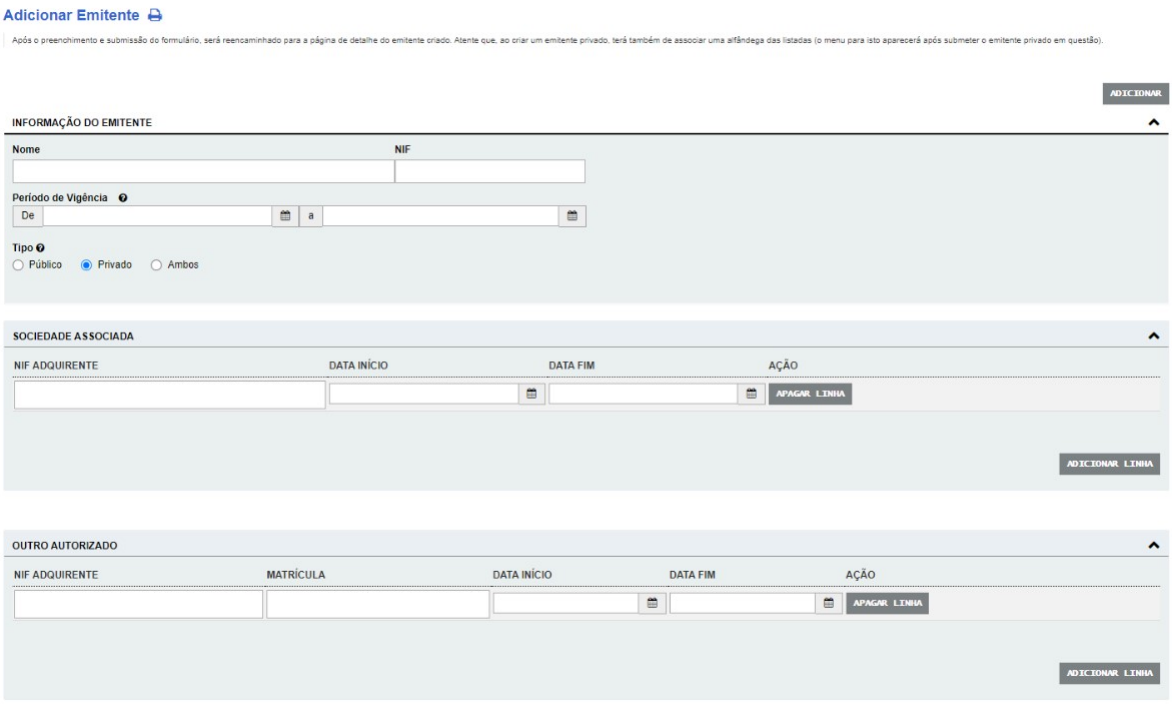

Figura 5.8: Adicionar Emitente.

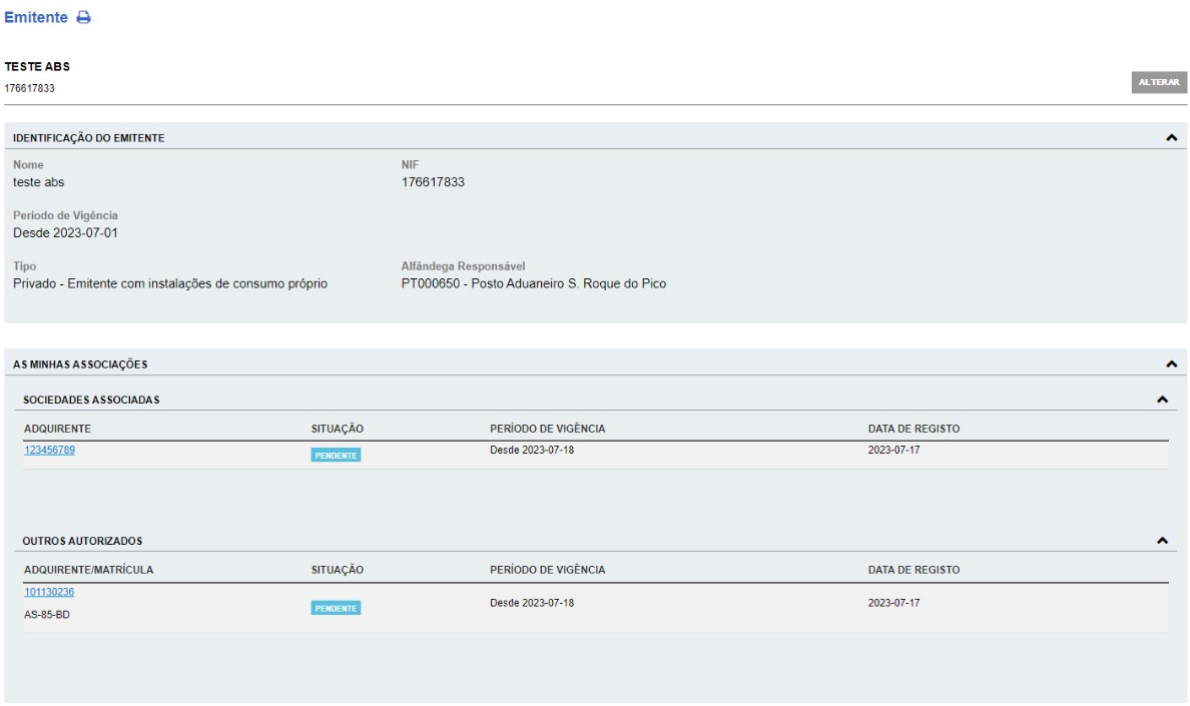

Figura 5.9: Consultar detalhe do Emitente.

5.3 Introdução

Concluídas as explicações da implementação do modelo de dados, do processamento do *batch* e da aplicação GASPROIntra, fica evidente que o projeto exigiu um trabalho árduo e um esforço considerável para ser plenamente realizado.

A implementação do modelo de dados envolveu o projeto e a estruturação de um esquema adequado para arquivar e gerir os dados da aplicação. O desenvolvimento das suas funcionalidades requereu um conhecimento aprofundado das tecnologias e ferramentas envolvidas, bem como habilidades de programação e resolução de problemas.

Assim, todas as melhorias e novas implementações levadas a cabo ocasionaram um aperfeiçoamento muito significativo na flexibilidade da aplicação, proporcionando a remoção do código estático para as configurações de tipo de combustíveis, veículo, peso mínimo e código CAE.

Em suma, foi um trabalho profundamente benéfico, visto que todos os procedimentos mencionados e detalhados neste capítulo foram concluídos com sucesso e de forma eficiente, atendendo aos requisitos do projeto. As melhorias realizadas trouxeram um impacto positivo substancial à aplicação, tornando-a mais adaptável, eficiente e flexível para futuras atualizações e implementações.

Capítulo 6

Testes

6.1 Introdução

Uma componente considerada importante neste trabalho consiste na validação da correção da solução e dos objetivos propostos alcançados durante esta fase.

Por se tratar de uma proposta realizada em contexto empresarial, o processo de validação segue um protocolo definido pela empresa, o qual permite não só garantir os padrões de qualidade, mas também apresentar provas concretas que permitam ao cliente atestar a qualidade do produto.

6.2 Metodologia de Testes Utilizada

Na **Opensoft**, o método utilizado para testar a correção de aplicações desenvolvidas assenta num conjunto de testes, abaixo apresentado.

Testes Unitários - Os testes unitários têm como principal objetivo validar componentes isoladas da solução.

Testes de Integração – Uma vez testadas individualmente as componentes do sistema, é necessário verificar se as várias componentes atuam entre si de modo a produzir o resultado desejado.

Testes de Qualidade – Para que a solução seja considerada válida na **Opensoft** é necessário que esta satisfaça métricas de qualidade, que garantem que o código desenvolvido é eficiente e apresenta baixo custo de manutenção. As métricas utilizadas serão apresentadas de forma detalhada mais adiante.

Testes de Aceitação – Após a solução ser confirmada em termos de qualidade, ela é considerada apta para a última fase de testes – os testes de aceitação. O objetivo principal desses testes é simular a utilização da ferramenta pelo utilizador final, replicando ao máximo as condições reais de uso. Nessa fase, a totalidade dos testes é realizada manualmente pelo próprio utilizador.

O objetivo dos testes de aceitação é detetar eventuais erros e identificar possíveis melhorias para proporcionar uma experiência de utilização ainda mais satisfatória. Nessa fase, os utilizadores têm a oportunidade de interagir com a ferramenta de forma mais próxima do cenário real, permitindo que qualquer falha ou dificuldade seja identificada e corrigida antes do lançamento final.

Na **Opensoft**, o processo de validação é normalmente realizado de forma incremental, o que significa que os testes, tanto os antigos quanto os novos, são executados integralmente no lançamento de uma nova versão da aplicação.

A utilização da ferramenta ***SonarQube*** (4.4.4) é uma excelente prática para medir a

qualidade do código produzido pela empresa. Essa ferramenta proporciona um conjunto de estatísticas e métricas que são fundamentais para avaliar a fiabilidade, a segurança e a manutenção do código.

Através da análise dessas métricas, a ferramenta compara os valores obtidos com as métricas de qualidade previamente definidas pela empresa. Com base nessa comparação, a ferramenta atribui uma classificação ao código, o que permite ter uma visão clara e objetiva sobre o nível de qualidade da solução desenvolvida.

Essas métricas e estatísticas fornecidas pelo ***SonarQube*** permitem uma avaliação mais precisa da qualidade do código, pelo que a empresa pode identificar áreas de melhoria, tomar decisões para melhorar o código e garantir que o produto final seja mais robusto, seguro e de fácil manutenção, atendendo aos padrões de qualidade estabelecidos.

O ***SonarQube*** desempenhou um papel crucial como ferramenta auxiliar no desenvolvimento das novas funcionalidades e das melhorias que foram desenvolvidas durante o estágio.

6.3 GASPRO - Teste Automáticos

Para comprovar o desenvolvimento realizado durante este estágio, foi essencial criar um conjunto abrangente de testes unitários, abordando diversos cenários de utilização da ferramenta.

Esses testes foram usados durante a fase de implementação para guiar o desenvolvimento. Como resultado, obtivemos métricas altamente positivas no ***SonarQube***, que comprovaram a qualidade do trabalho realizado. Na Figura 6.1, são apresentadas as estatísticas globais do projeto, calculadas pelo ***SonarQube***, incluindo o número total de testes, a percentagem de cobertura do código desenvolvido e o número de problemas identificados.

Para uma visão mais detalhada do contributo de cada um dos módulos desenvolvidos para as estatísticas é apresentada, na Figura 6.2, uma análise detalhada das estatísticas de cada um dos módulos.

6.4 GASPRO - Geração de Testes Unitários e de Integração

Para assegurar o correto funcionamento do sistema, foi elaborado um conjunto abrangente de testes unitários e de integração, abarcando todas as camadas do projeto. Para isso, foram criadas quatro categorias distintas de testes, as quais serão detalhadamente apresentadas a seguir.

6.4.1 Testes Unitários à camada de Persistência

A primeira camada de testes realizada e a mais simples foi a geração de testes unitários para a camada de repositórios. O objetivo destes testes é verificar se os dados estão a ser persistidos de uma forma correta, sendo, para este efeito, realizados testes de inserção, remoção e atualização de elementos 6.1.

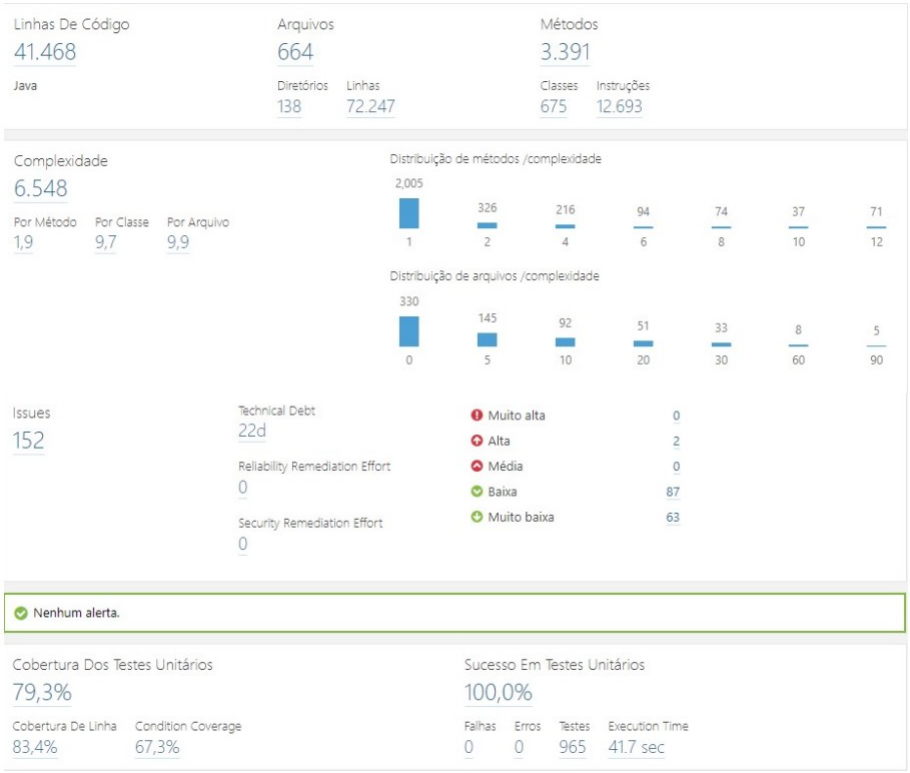

Figura 6.1: Estatística Globais do Modulo GASPROIntra.

6.4.2 Testes Unitários à camada de Serviços

Esta camada de testes tem como objetivo validar a camada de serviços de forma isolada. Para tal, a camada de persistência é simulada por meio do uso da *framework* ***Mockito***(4.5.7). Essa *framework* permite substituir a camada de uma função por um valor pré-definido, isolando, assim, a componente que está a ser testada. Com a camada de persistência devidamente isolada, são realizados testes com parâmetros variáveis para verificar se a lógica aplicacional executada produz os resultados esperados 6.2.

6.4.3 Testes Unitários à camada de Controlo

Os testes direcionados à camada de controlo são construídos através da realização de pedidos *POST* ou *GET*. O objetivo é perceber se os pedidos efetuados produzem os resultados desejados com base nos dados fornecidos 6.3.

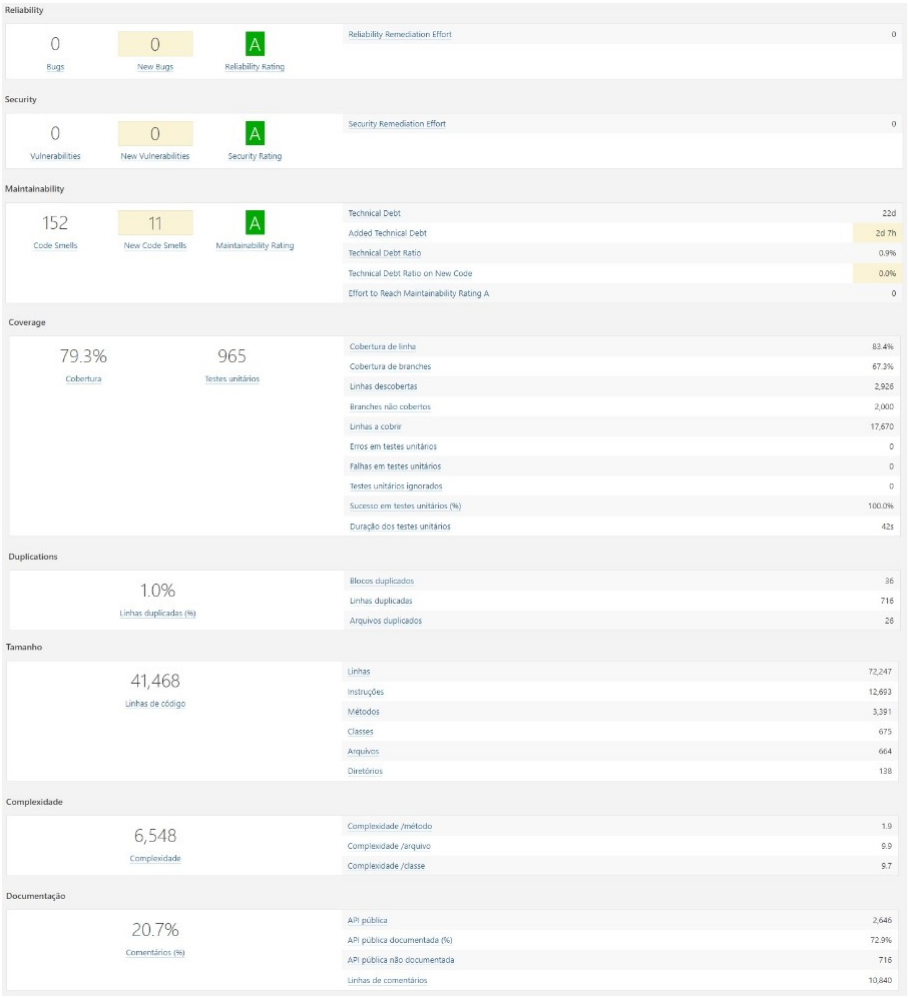

Figura 6.2: Estatística Detalhadas por Módulo GASPROIntra.

6.4.4 Testes de Integração das várias componentes

Os testes de integração formam um conjunto de testes de alto nível, cujo objetivo principal é confirmar que a interação entre as camadas de controladores, serviços e persistência ocorre conforme o esperado. Para isso, são efetuados diversos pedidos de inserção, remoção, atualização e consulta de elementos. Nesses testes, nenhuma das componentes é simulada utilizando o ***Mockito***, pois o propósito é simular uma execução real da aplicação e validar a correta interação entre os componentes.

```java
public class AbastRepositoryTest {
  @Test
  @Sql(scripts = IMPORT_TEST_DATA, executionPhase = BEFORE_TEST_METHOD)
  public void testUpdate() throws Exception {
    AbastecimentoPk abastecimentoPk = new AbastecimentoPk();
    Abastecimento abastecimento = AbastRepository.findOne(abastecimentoPk);
    Fich fich = abastecimento.getFich();
    Date startDate = DateTimeCommonUtils.getDateStringFormattedAsDate("2017-01-01
     00:00:00", DateTimeCommonUtils.DATE_TIME_FORMAT_STRING);
    Date endDate = Calendar.getInstance().getTime();
    assertNotEquals(dadoTesteCombustivel1, abastecimento.getState());
    assertTrue(fich.getDataEnvio().after(startDate));
    assertNotEquals(dadofich1, fich.getState());
    assertNotEquals(dadofich2, fich.getState());
    assertTrue(fich.getDataEnvio().before(endDate));
    assertTrue(abastecimento.getDataAbastecimento().after(startDate));
    assertEquals(dadoTesteCombustivel1, abastecimento.getSituacao());
    assertTrue(abastecimento.getDataAbastecimento().before(endDate));
    AbastRepository.updateSuppliesReprocessingProcess(nifTest, datdofich.name(),
    startDate, endDate, null, null, "PT", matriculaTest);
    Abastecimento abastecimento1 = AbastRepository.findOne(abastecimentoPk);
    assertEquals(dadoTesteCombustivel2, abastecimento1.getState());
  }
}
```

Listagem 6.1: Exemplo de uma testes unitário na camada de Persistência

```java
public class FichServiceTest {
  @Test
  public void testGetDetailFichModel() throws IOException {
        mockMvc = MockMvcBuilders.webAppContextSetup(webApplicationContext).build
    ();
        Fichs.createDirectories(Paths.get(Parameters.getUploadRcvDir()));
        Long fichID = TestUtils.simulateFichUpload(fichService, TestUtils.
    SAMPLEWITH2FUELSUPPLIES, new Date(), dadofich1);
        FichId fichId = new FichId();
        fichId.setId(fichID);
        Fich fich = fichService.getOne(fichId);
        fMotivoService.saveAndFlush(dadoTesteFmotivo);
        Fich insertedFich = fichService.findAll().get(0);
        DetailFichModel detailFichModel = fichService.getDetailFichModel(
    fichMapper.convertToFichModel(insertedFich), dadofich2, 10, "Success");
        Assert.assertNotNull(detailFichModel);
        Assert.assertEquals(TestUtils.SAMPLEWITH2FUELSUPPLIES,detailFichModel.
    getFichname());
        TestUtils.deleteDirectory(Parameters.getUploadRcvDir());
    }
}
```

Listagem 6.2: Exemplo de uma testes unitário na camada de Serviços

```
public class EmitenteControllerTest {
    @Test
    @BusinessTest(idAccount = 2022, idRequirement = RQ01, idUserStory = USW38,
    idBusinessRule = "RN07", idTestScenario = "Se ainda não existir registo do
    adquirente identificado na associação, o mesmo é criado no momento da
    associação.")
    public void testAddEmitenteAndAuthorizationNewPurchaser() throws Exception {
        PreAuthenticatedAuthenticationToken principal = Mockito.mock(
    PreAuthenticatedAuthenticationToken.class);
        Mockito.when(principal.getName()).thenReturn(TEST_USER);
        List<Adquirente> adquirenteList = adquirenteService.findByNif(nifteste);
        assertEquals(0, adquirenteList.size());
        Date now = new Date();
        String startDate = DateTimeCommonUtils.getDateFormattedAsString(
    DateTimeCommonUtils.getAddOrSubtractDays(now, 1), DateTimeCommonUtils.
    DATE_FORMAT_STRING);
        ResultActions resultActions = mockMvc.perform(get(linkAdd)
                        .principal(principal)
                        .param("nif", nifTeste2)
                        .param("nome", nomeTeste)
                        .param("tipo", tipoTeste)
                        .param("dataInicio", "2017-06-19")
                        .param("alfandega", dadoTesteAlfandega)
                        .param("autorizacao", dadoTesteAutorizacao)
                .andExpect(status().isFound());
        Emitente emitente = emitenteService.findBynif(nifTeste2);
        resultActions.andExpect(view().name("redirect:linkRedirect" + emitente.
    getId()));
        adquirenteList =  adquirenteService.findByNif(dadoTesteAutorizacao.
    nifTesteAdquirenteAutorizado);
        assertEquals(1, adquirenteList.size());

    }

    @Test
    @AcceptanceTest(idAccount = 2021, idRequirement = RQ05, idUserStory = USW35,
    idAcceptanceCriteria = "CA1",
            idTestScenario = "Quando o funcionário da DSIECIV preenche o
    formulário de associação com dados válidos, a associação é feita com sucesso.
    ")
    public void testAddEmitenteAuthorization() throws Exception {
        PreAuthenticatedAuthenticationToken principal = Mockito.mock(
    PreAuthenticatedAuthenticationToken.class);
        Mockito.when(principal.getName()).thenReturn(TEST_USER);

        Date now = new Date();
        String startDate = DateTimeCommonUtils.getDateFormattedAsString(
    DateTimeCommonUtils.getAddOrSubtractDays(now, 1), DateTimeCommonUtils.
```

```
    DATE_FORMAT_STRING);
36          mockMvc.perform(get(linkAdd)
37                          .principal(principal)
38                          .param("nif", nifTeste3)
39                          .param("nome", nomeTeste)
40                          .param("tipo", tipoTeste)
41                          .param("dataInicio", "2017-06-19")
42                          .param("alfandega", dadoTesteAlfandega)
43                          .param("autorizacao", dadoTesteAutorizacao)
44                  .andExpect(view().name("redirect:linkRedirect"+1))
45                  .andExpect(status().isFound());
46      }
47  }
```

Listagem 6.3: Exemplo de uma testes unitário na camada de Controlo

6.5 Conclusão

Face ao exposto, podemos concluir que esta fase de testes é essencial para garantir a qualidade da aplicação e prevenir a introdução de erros ou alterações involuntárias durante o seu desenvolvimento futuro.

Com a criação de testes, aumentar-se-á a fiabilidade da aplicação, tornando-a mais estável e menos propensa a falhas. Atualmente, os testes são ferramentas muito importantes para o desenvolvimento. No entanto, se os testes não forem rigorosos, a credibilidade da aplicação será significativamente reduzida, colocando em risco a sua estabilidade, facto que poderá levar a falhas e tornar o investimento de tempo e recursos no desenvolvimento de testes insatisfatório.

Resta acrescentar que os testes de software desempenham um papel fundamental na construção de uma aplicação de alta qualidade que, além de reduzir custos na correção de erros, também contribui para a satisfação do cliente através da entrega de um produto de alta qualidade. Ao assegurar que a aplicação seja estável, confiável e livre de falhas, os testes garantem uma experiência positiva para o utilizador, o que é essencial para o sucesso do produto no mercado. Através de um desenvolvimento cuidadoso dos testes e da implementação rigorosa dos mesmos, é possível obter uma aplicação que atenda às expetativas dos clientes, aumente a confiança na marca e promova um ambiente propício para o crescimento do negócio.

Capítulo 7

Conclusões e Trabalho Futuro

7.1 Conclusões Principais

Ao retrocedermos ao ponto inicial deste estágio e avaliarmos todos os objetivos definidos inicialmente, constatamos que foram explorados e alcançados com absoluto sucesso. O mesmo pode ser afirmado em relação aos objetivos adicionais incorporados durante o desenvolvimento do projeto, os quais também foram conseguidos na sua grande maioria. Alguns objetivos requereram mais tempo para serem concretizados, contudo esse trabalho resultou na realização técnica de outros objetivos.

A aplicação **GASPRO**, desenvolvida em 2016, passou por aprimoramentos significativos com a intenção de implementar uma evolução no atual processo de classificação de combustíveis. O objetivo era adotar um modelo automático e flexível de categorização dos abastecimentos.

Acrescenta-se que, na concretização deste projeto de estágio, foram necessárias diversas etapas, iniciando-se com a análise do estado da arte para se obter conhecimentos acerca dos diferentes tipos de processamento de arquivos *XML* e das respetivas validações. Em seguida, procedeu-se à implementação e aperfeiçoamento do processo de classificação de combustíveis na aplicação, adotando um modelo automático e flexível para a categorização dos abastecimentos.

Neste processo, algumas soluções exigiram maior esforço, enquanto outras, consideradas possíveis, falharam na sua implementação, o que consideramos ter sido uma aprendizagem crucial. Essa experiência permitiu estabelecer uma cultura de melhoria incessante da solução proposta, resultando, por fim, numa solução com um nível de qualidade extremamente satisfatório.

Em resumo, acredito que o projeto de estágio constituiu um contributo significativo não apenas para o meu desenvolvimento intelectual e profissional, mas também para o aperfeiçoamento do produto, facto que nos deixa excessivamente satisfeitos.

7.2 Trabalho Futuro

Como é comum dizer-se, nenhuma solução é perfeita, e essa máxima também se aplica à solução implementada no âmbito deste projeto de estágio.

Mas, em primeiro de lugar, é necessário focar o que faltou para cumprir o plano de desenvolvimento, designadamente a alteração do ecrã da edição do emitente, para permitir a sua edição, e também o cancelamento/revogação da(s) autorização(ões) para que o adquirente possa abastecer num posto privado de um emitente específico, no módulo **GASPROIntra**.

Uma das áreas a serem exploradas em futuras pesquisas e investigações nos dois módu-

los no projeto **GASPRO**.

No módulo **GASPROInter** a criação de novos ecrãs, visando possibilitar que um determinado emitente tenha acesso às seguintes funcionalidades:

1. Consulta as suas autorizações fornecidas, permitindo um melhor acompanhamento das permissões concedidas.

2. Submissão do(s) pedido(s) de autorização(ões) enviados à entidade responsável, para que um determinado adquirente possa abastecer nos postos privados, promovendo maior transparência e controle sobre essas solicitações.

3. Submissão do(s) pedido(s) de cancelamento(s)/revogação(ões) enviados à entidade responsável para o adquirente em questão, facilitando a supervisão e a gestão dessas ações.

No módulo **GASPROIntra** para ter um sistema mais abrangente e eficiente, atendendo às necessidades de controlo e gestão de autorizações de forma mais completa.

Precisamente, a criação de novos ecrãs teria a finalidade de facilitar a aprovação das submissões de cancelamento/revogação para que um determinado adquirente possa abastecer em outros postos privados. Além disso, adicionar novos ecrãs permitiriam que o(s) utilizador(es) responsável por realizar configurações e ajustar os parâmetros que são necessários para as configurações processos em *batch* e proporcionando maior flexibilidade. Com essas implementações, o sistema seria capaz de gerir de forma mais eficiente as permissões e as configurações relacionadas com os abastecimentos e autorizações, garantindo uma operação mais suave e adaptável às necessidades específicas dos envolvidos.

No entanto, para que qualquer trabalho futuro, como a criação de novos ecrãs e implementação de funcionalidades adicionais nos dois módulos da aplicação **GASPRO**, seja realizado, é essencial obter o aval e o suporte dos *stakeholders* da aplicação.

Bibliografia

[1] The java web services tutorial. [Online]. Available: http://www.cs.uccs.edu/~cs526/jwsdp/docs/tutorial/doc/JAXPIntro4.html xv, 12

[2] rishabh110511. What is document object in java dom? [Online]. Available: https://www.geeksforgeeks.org/what-is-document-object-in-java-dom/ xv, 13

[3] Sam Page. (2005) Package javax.xml.validation. [Online]. Available: http://www.usingxml.com/Basics/XmlValidation xv, 16, 19

[4] Visão geral de metodologia Ágil. [Online]. Available: https://www.nimblework.com/pt-br/agile/metodologia-agil/ xv, 25, 26, 27

[5] The mvc design pattern in abap for practical use. [Online]. Available: https://blogs.sap.com/2010/09/06/the-mvc-design-pattern-in-abap-for-practical-use-part-1/ xv, 28

[6] Spring framework. [Online]. Available: https://glysns.gitbook.io/spring-framework/ xv, 30, 31

[7] Ed Ort and Bhakti Mehta. (2003) Java architecture for xml binding (jaxb). [Online]. Available: https://www.oracle.com/technical-resources/articles/javase/jaxb.html xvii, 14, 19

[8] Commons betwixt. [Online]. Available: https://commons.apache.org/dormant/commons-betwixt/ xvii, 19

[9] Desconhecido. apache xml beans. [Online]. Available: https://xmlbeans.apache.org/ xvii, 19

[10] Castor. [Online]. Available: https://castor-data-binding.github.io/castor/reference-guide/reference/xml/xml-framework.html xvii, 19

[11] Xstream. [Online]. Available: https://x-stream.github.io/ xvii, 19

[12] Simple. [Online]. Available: https://simple.sourceforge.net/ xvii, 17, 19

[13] Ministério das Finanças. (2016) Portaria n.º 246-a/2016. [Online]. Available: https://files.dre.pt/1s/2016/09/17301/0000200004.pdf 1

[14] Ministério das Finanças XXIII Governo Constitucional. (2023) Orçamento do estado 2023. Pagina 65. [Online]. Available: https://www.dgo.gov.pt/politicaorcamental/OrcamentodeEstado/2023/Proposta%20do%20Or%C3%A7amento/Documentos%20do%20OE/OE2023_doc16_Relatorio.pdf 1, 34

[15] T. e. K. Colin Robertson. (2022) Processamento de arquivo de documentação xml. [Online]. Available: https://learn.microsoft.com/pt-br/cpp/build/reference/dot-xml-file-processing?view=msvc-170 11

[16] (2013) Relacionamento do processador com a api java para processamento xml (jaxp). [Online]. Available: https://www.ibm.com/docs/pt-br/was/8.5.5?topic=api-relationship-processor-java-xml-processing-jaxp 11

[17] Y. Pan, Y. Zhang, and K. Chiu, "Hybrid parallelism for xml sax parsing," in *2008 IEEE International Conference on Web Services*, 2008, pp. 505–512. 12

[18] Y. Xiang, B. Zhang, G. Li, Y. Li, and X. Gao, "The application and analysis of netconf subtree filtering based on sax and dom," in *2010 International Conference on Measuring Technology and Mechatronics Automation*, vol. 3, 2010, pp. 758–761. 12

[19] T. C. Lam, J. J. Ding, and J.-C. Liu, "Xml document parsing: Operational and performance characteristics," *Computer*, vol. 41, no. 9, pp. 30–37, 2008. 12, 13

[20] (2001) sax project. [Online]. Available: http://www.saxproject.org/ 12

[21] BrettMcLaugblin, *Java & XML Data Binding*, ser. ISBN: 0-596-00278-5. O'Reilly, 2002. 12, 14

[22] E. e W. Scott Means, *XML in a Nutshell*, ser. ISBN: 0-596-00292-0. O'Reilly, 2002. [Online]. Available: https://docstore.mik.ua/orelly/xml/xmlnut/index.htm 12

[23] Xml parsing for java. [Online]. Available: https://docs.oracle.com/en/database/oracle/oracle-database/21/adxdk/XML-parsing-for-Java.html#GUID-33E0EA2B-A08A-462E-86B4-3F944409663D 12, 16

[24] F. Simeoni, D. Lievens, R. Conn, and P. Mangh, "Language bindings to xml," *IEEE Internet Computing*, vol. 7, no. 1, pp. 19–27, 2003. 12, 14

[25] H. Son and B. Son, "Design of the xml document object model for the home management server," in *5th International Conference on Computer Sciences and Convergence Information Technology*, 2010, pp. 115–118. 12

[26] A. Sarasa-Cabezuelo, B. Temprado-Battad, A. Martinez-Aviles, J.-L. Sierra, and A. Fernandez-Valmayor, "Building an enhanced syntax-directed processing environment for xml documents by combining stax and cup," in *2009 20th International Workshop on Database and Expert Systems Application*, 2009, pp. 427–431. 13

[27] E. Song and S.-C. Haw, "Xml-reg: Transforming xml into relational using hybrid-based mapping approach," *IEEE Access*, vol. 8, pp. 177 623–177 639, 2020. 13

[28] (2015) Streaming api for xml. [Online]. Available: https://mediacenter.ibm.com/media/Streaming+API+for+XML/0_d9ropbsm 13

[29] H. Jiang, A. Lo, T. Ozyer, and R. Alhajj, "Xml views based approach for web services," in *IRI -2005 IEEE International Conference on Information Reuse and Integration, Conf, 2005.*, 2005, pp. 458–463. 14

[30] (2015) Jaxb. [Online]. Available: https://www.ibm.com/docs/pt-br/was-nd/8.5.5?topic=services-jaxb 14

[31] Y. Shulin, Y. Nan, H. Jieping, and R. Xuelei, "Research on the implement scheme of reconfigurable function navigation based on jaxb," in *2016 8th International Conference on Information Technology in Medicine and Education (ITME)*, 2016, pp. 724–727. 14

[32] A. Mukherjee, Z. Tari, and P. Bertok, "A spring based framework for verification of service composition," in *2011 IEEE International Conference on Services Computing*, 2011, pp. 258–265. 14

[33] K. Sakib, Z. Tari, and P. Bertok, *WEB SERVICES*, 2014, pp. 57–76. 14

[34] Chimezie Ogbuji. (2000) Validating xml with schematron. [Online]. Available: https://www.xml.com/pub/a/2000/11/22/schematron.html 15

[35] Johannes Becker. Schematron validation how to. [Online]. Available: https://ecad-wiki.prostep.org/specifications/vec/schematron-howto/ 15

[36] (2006) Schematron validation how to. [Online]. Available: https://schematron.com/home/implementation.html 15

[37] K. Bohm, K. Aberer, M. Ozsu, and K. Gayer, "Query optimization for structured documents based on knowledge on the document type definition," in *Proceedings IEEE International Forum on Research and Technology Advances in Digital Libraries - ADL'98P–*, 1998, pp. 196–205. 16

[38] L. Zhou and J. Wang, "A method for exporting relational schema into document type definition," in *2010 Third International Symposium on Information Science and Engineering*, 2010, pp. 30–32. 16

[39] F. Ye and X. Jingsheng, "Mapping xml dtd to relational schema," in *2009 First International Workshop on Database Technology and Applications*, 2009, pp. 557–560. 16

[40] Y. Papakonstantinou and P. Velikhov, "Enhancing semistructured data mediators with document type definitions," in *Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337)*, 1999, pp. 136–145. 16

[41] Package javax.xml.validation. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/javax/xml/validation/package-summary.html 16

[42] P. M. Pardeshi and G. Ramachandran, "Generic method for xml generation from given xsd," in *2011 3rd International Conference on Electronics Computer Technology*, vol. 5, 2011, pp. 404–408. 16

[43] L. Checiu and D. Ionescu, “A new algorithm for mapping xml schema to xml schema,” in *2010 International Joint Conference on Computational Cybernetics and Technical Informatics*, 2010, pp. 625–630. 16

[44] Y. Zhang, H. Zhou, J. Liu, Z. Liang, and P. Duan, “Efficient schema extraction from large xml documents,” in *2012 5th International Conference on BioMedical Engineering and Informatics*, 2012, pp. 1255–1260. 16

[45] W. Gong and P. Yao, “Based on grammar analysis’s expressiveness among the different xml-schema languages,” in *2013 IEEE 4th International Conference on Software Engineering and Service Science*, 2013, pp. 116–119. 16

[46] N. Gherabi, K. Addakiri, and M. Bahaj, “A framework for mapping uml class into xml data based on technical specifications,” in *2012 International Conference on Multimedia Computing and Systems*, 2012, pp. 749–754. 16

[47] Xml schema. [Online]. Available: https://www.w3.org/XML/Schema 16

[48] C. League and K. Eng, “Type-based compression of xml data,” in *2007 Data Compression Conference (DCC’07)*, 2007, pp. 273–282. 16

[49] H. Cui, B. Zhang, G. Li, X. Gao, and Y. Li, “Contrast analysis of netconf modeling languages: Xml schema, relax ng and yang,” in *2009 International Conference on Communication Software and Networks*, 2009, pp. 322–326. 16

[50] Introdução ao mvc. [Online]. Available: https://www.devmedia.com.br/introducao-ao-padrao-mvc/29308/ 27

[51] Intellij idea – the leading java and kotlin ide. [Online]. Available: https://www.jetbrains.com/idea/ 28

[52] Accelerating api quality through testing. [Online]. Available: https://www.soapui.org/ 29

[53] Sap powerdesigner. [Online]. Available: https://www.sap.com/products/technology-platform/powerdesigner-data-modeling-tools.html 29

[54] clean code for teams and enterprises with sonarqube. [Online]. Available: https://www.sonarsource.com/products/sonarqube/ 29

[55] History of java programming language. [Online]. Available: https://u-next.com/blogs/java/history-of-java/ 29

[56] CHRIS COLLINS. A brief history of xml. [Online]. Available: https://ccollins.wordpress.com/2008/03/03/a-brief-history-of-xml/ 30

[57] History of sql. [Online]. Available: https://docs.oracle.com/cd/B13789_01/server.101/b10759/intro001.htm 30

[58] Sharqa Hameed. Html history | explained. [Online]. Available: https://linuxhint.com/html-history/ 30

[59] RAJESH KUMAR. What is junit and how it works? an overview and its use cases. [Online]. Available: https://www.devopsschool.com/blog/what-is-junit-and-how-it-works-an-overview-and-its-use-cases/ 30

[60] Tasty mocking framework for unit tests in java. [Online]. Available: https://site.mockito.org/ 31

[61] History of maven. [Online]. Available: https://maven.apache.org/background/history-of-maven.html 31

[62] Spring batch. [Online]. Available: https://spring.io/projects/spring-batch 35, 37

[63] Introdução ao spring batch. [Online]. Available: https://www.devmedia.com.br/introducao-ao-spring-batch/33284 35, 37

[64] Giuliana Bezerra. Desenvolvimento com spring batch. [Online]. Available: https://giulianabezerra.medium.com/desenvolvimento-com-spring-batch-steps-4d42af2696ec 38

Glossário

.Net	É uma plataforma de código aberto para a criação de aplicações de desktop, Web e móveis que podem ser executadas nativamente em qualquer sistema operacional.
API	É um conjunto de serviços que foram programadas em um software, que são disponibilizados para que aplicativos possam utilizar dessas funcionalidades diretamente, sem envolver-se em detalhes da implementação do software.
Base de Dados	É uma ferramenta de recolha e organização de informações.
Framework	É um conjunto de códigos genéricos capaz de unir trechos de um projeto de desenvolvimento.
Gigabyte	Unidade de medida de informação, equivalente a 1024 megabytes.
HTTP	É um protocolo que permite a obtenção de recursos, como documentos **HTML**.
Intranet	É uma rede de computadores de acesso exclusivo de uma empresa ou corporação.
Internet	Rede informática utilizada para interligar computadores a nível mundial, à qual pode aceder qualquer tipo de utilizador , e que possibilita o acesso a toda a espécie de informação.
Java	é uma linguagem de programação orientada a objetos desenvolvida na década de 90 por uma equipe de programadores chefiada por James Gosling, na empresa *Sun Microsystems*, que em 2008 foi adquirido pela empresa *Oracle Corporation*.
Open source	Significa ”código aberto” e se refere ao código-fonte de um site ou aplicativo.
Parser	É um objeto que faz análise sintática de um texto para determinar a sua estrutura lógica.
SGBD	É o sistema de software responsável pela gestão de um ou mais base de dados.
Software	É um serviço computacional utilizado para realizar ações nos sistemas de computadores.
Sprint batch	É uma parte do projeto ***Spring***, e visa fornecer funções reutilizáveis para o processamento e gestão de dados em um determinado cenário de integração de sistemas através de aplicações *batch*.
XML	É uma linguagem de marcação que fornece regras para definir quaisquer dados.
XSD	É uma especificação de usadas para descrever o ”formato/padrão” que um arquivo **XML**.

XPath	É uma linguagem de consulta (*Query Language*) para selecionar nós de um documento **XML**. Ademais, **XPath** pode ser usada para computar valores do conteúdo de um documento **XML**. **XPath** foi definido pelo *World Wide Web Consortium* (**W3C**).
XML Schema	um arquivo codificado em linguagem baseada em padrão **XML** que contém a definição da estrutura de um documento **XML**, as definições de tipo, tamanho, ocorrência e regras de preenchimento dos elementos que compõe documento **XML**.