Projeto

Mestrado em Engenharia Informática - Computação Móvel

Desenvolvimento de API para aplicação cloud

Ana Isabel Alves Marques

Leiria, abril de 2018

Projeto

Mestrado em Engenheria Informática - Computação Móvel

Desenvolvimento de API para aplicação cloud

Ana Isabel Alves Marques

Projeto de Mestrado realizada sob a orientação do Doutor Marco António de Oliveira Monteiro, Professor da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria .

Leiria, abril de 2018

Agradecimentos

Na elaboração deste projeto foram várias as pessoas que, direta ou indiretamente, contribuíram para que esta etapa da minha formação académica, chegasse a bom termo. A elas quero expressar a mais sincera gratidão.

Ao orientador Professor Doutor Marco António de Oliveira Monteiro por toda a ajuda, motivação e disponibilidade concedida ao longo deste projeto.

Às empresas Sinmetro e Aferymed, pela excelente oportunidade que me foi dada e pela confiança que depositaram em mim, bem pelo facto de me terem proporcionado todas as condições necessárias para a elaboração do estágio.

À minha mãe e meu irmão, um enorme obrigada por acreditarem sempre em mim e nas minhas capacidades.

Aos meus amigos, colegas de curso e a todos aqueles não mencionados que, direta ou indiretamente, contribuíram a concretização deste projeto.

A todos dedico este projeto.

Resumo

Graças ao aparecimento do estilo de arquitetura *REST* (*Representational State Transfer*) surgiu uma nova alternativa para a implementação de serviços *Web*. Este permitiu que, de um modo mais simples e flexível, fosse possível desenvolver APIs (*Application Programming Interface*) para a comunicação entre cliente e servidor.

Deste modo, a Sinmetro, empresa onde foi realizado o estágio, sentiu a necessidade de desenvolver uma API REST para aplicação *cloud* em modelo *SaaS* (*Software as a Service*) com vista a integração de uma aplicação *Web* destinada ao controlo estatístico da quantidade em pré-embalados. Essa aplicação *Web* designa-se por Accept *cloud* e é um dos módulos do sistema Accept, *software* criado pela empresa.

Este relatório descreve todo o processo de desenvolvimento da *API* REST, desenvolvida em *Python*, que tem como principais objetivos a gestão, recolha de dados e tratamento estatístico de dados necessários ao controlo metrológico de pré-embalados.

Palavras-chave: *API REST, Python, Django REST, Metrologia, Pré-embalados*

Abstract

Thanks to the appearance of the REST (Representational State Transfer) arquitectural style a new alternative for the implementation of web services has emerged. This allowed that, in a simpler and more flexible way, it was possible to develop APIs (Application Programming Interface) for the communication between client and server.

This way, Sinmetro, the company where the internship was held, felt the need to develop a REST API for cloud application in the SaaS (Software as a Service) model, aiming for the integration of the web application for the statistical control of quantity in prepackaged. This web application is called Accept cloud and is one of the Accept system modules, software created by the company.

This report describes the entire process of developing the REST API, developed in Python, whose main objectives are the management, data collection and statistical treatment of data necessary for the pre-packaged metrological control.

Keywords: *API REST, Python, Django REST, Metrology, Prepackages*

Lista de Figuras

Lista de Tabelas

Lista de Siglas

API Application Programming Interface

ASAE Autoridade de Segurança Alimentar e Económica

CGPM Conferência Geral de Pesos e Medidas

CL Critério Legal

CP Capacidade do Processo

CRUD Create, Read, Update e Delete

CSS Cascading Style Sheets

CSV Comma-Separated Values

ddof Delta Degrees Of Freedom

DRF Django Rest Framework

DT Desenvolvimento Tecnológico

EAD Erro Admissível por Defeito

ENG Inglês

ESTG Escola Superior de Tecnologia e Gestão

GUI Graphical User Interface

HATEOAS Hypertext As The Engine Of Application State

HTTP Hypertext Transfer Protocol

I Investigação

IPL Instituto Politécnico de Leiria

IPQ Instituto Português da Qualidade

JSON JavaScript Object Notation

KPI Key Performance Indicators

LCL Lower Specification Limit

MVC Model-View-Control

MVT Model-View-Template

OIML Organização Internacional de Metrologia Legal

ORM Object-Relational Mapping

OVM Organismo de Verificação Metrológico

PDF Portable Document Format

PT Português

REST Representational State Transfer

SaaS Software as a Service

SGBD Sistema de Gestão de Base de Dados

SI Sistema International de Unidades

SOAP Simple Object Access Protocol

SPC Statistical Process Control

SQL Structured Query Language

TI Tecnologias de Informação

URI Uniform Resource Identifier

URL Uniform Resource Locator

USL Upper Specification Limit

WSGI Web Server Gateway Interface

WWW World Wide Web

XML eXtensible Markup Language

YAML YAML Ain't Markup Language

Índice

Capítulo 1

Introdução

O controlo metrológico dos produtos pré-embalados é obrigatório para todo e qualquer produto embalado na ausência do consumidor em quantidades entre os 5 g/mL a 10 kg/mL, abrangendo a maioria dos produtos alimentares. Deste modo, o enchimento dos pré-embalados é um processo que obedece a um conjunto de normas e leis para ser considerado dentro dos conformes.

Com a constante evolução da tecnologia e do seu acesso cada vez facilitado, a sociedade está cada vez mais dependente das Tecnologias de Informação (TI) e são cada vez mais os utilizadores que recorrem a aplicações informáticas como forma de gestão e processamento de dados. Foi neste contexto que a empresa Sinmetro desenvolveu o sistema Accept, *software* que permite às empresas embaladoras, nas diferentes fases de produção, controlar todos os parâmetros críticos do seu processo e reagir em conformidade.

No entanto, graças à evolução da Internet e do respetivo aumento da sua utilização, as aplicações informáticas para dispositivos específicos têm vindo a ser substituídas por aplicações que nos oferecem serviços para qualquer tipo de dispositivo e de forma ubíqua.

Uma vez que o sistema Accept só se encontra disponível em ambientes *desktop* e tendo em conta que o mercado atual está cada vez mais competitivo, a empresa sentiu a necessidade de desenvolver um novo módulo, o Accept *cloud*. Este módulo permite às empresas embaladoras aceder às informações internas das suas fábricas, de uma forma ubíqua e em tempo real. Deste modo, a empresa Sinmetro decidiu transpor um dos módulos Accept, o Accept *gml* para a *cloud*.

Este conceito é designado por *Software* como Serviço, do inglês *Software as a Service*(SaaS), e tem vindo a revolucionar o negócio das indústrias TI. Na computação em nuvem, o *software* passa a ser operado pelo fornecedor de *software* e entregue ao cliente como um serviço [1].

Um serviço *Web* é um sistema desenvolvido para suportar interoperabilidade en-

tre máquinas sobre uma rede baseada em protocolos e padrões *Web*. Este pode ser implementado com base em dois tipos de arquiteturas distintas: arquiteturas baseadas em SOAP (*Simple Object Access Protocol*) ou arquiteturas baseadas em REST (*Representational State Transfer*).

O trabalho desenvolvido neste projeto insere-se no contexto de desenvolvimento de um serviço *Web* com base na arquitetura REST, nomeadamente uma API (*Application Programming Interface*) responsável pela troca de informações entre o servidor da aplicação *Web* e a base de dados, bem como efectuar todo o tratamento estatístico dos dados e a gestão de todo o seu conteúdo. A API REST irá integrar com a aplicação *Web* que está a ser desenvolvida. Atualmente, o REST é uma das arquiteturas mais populares para o desenvolvimento de serviços distribuídos, permitindo a implementação simplificada de APIs para a comunicação entre os diferentes componentes do sistema, nomeadamente entre o servidor e os diferentes clientes.

1.1 Estrutura do relatório

Este relatório descreve todo o processo de investigação e desenvolvimento da API REST realizados durante o estágio na empresa Sinmetro. Para além deste capítulo (Introdução), este relatório contém mais seis capítulos.

No capítulo 2 (Enquadramento) será feito o enquadramento contextual do estágio através de uma breve caracterização da empresa onde decorreu o mesmo e de uma descrição do trabalho realizado.

No capítulo 3 (Estado de Arte) irão ser abordados com mais profundidade os conceitos e tecnologia utilizados que são a base de conhecimento necessária ao desenvolvimento deste projeto. Este capítulo é dividido em duas secções: a Metrologia, nomeadamente no contexto do controlo metrológico dos produtos pré-embalados e a arquitetura REST.

No capítulo 4 (Arquitetura) é descrito a arquitetura do Accept *cloud*, os padrões arquiteturais implementados pela *framework Django* REST e por fim, a arquitetura da API REST *Python* desenvolvida.

No capítulo 5 (Implementação) é apresentada a metodologia de desenvolvida imposta pela empresa, as várias tecnologias utilizadas no desenvolvimento da API e uma descrição de como a API foi implementada.

No capítulo 6 (Testes) são apresentados os testes realizados à API.

Por fim, no capítulo 7 (Conclusão) é feito uma conclusão em relação ao estado do projeto na fase final do estágio e dadas algumas sugestões para trabalho futuro.

Capítulo 2

Enquadramento

Neste capítulo pretende-se fazer um enquadramento contextual do estágio através de uma breve caracterização da empresa onde decorreu o mesmo e de uma descrição geral do trabalho realizado.

2.1 Âmbito

No âmbito do mestrado em Engenharia Informática - Computação Móvel, da Escola Superior de Tecnologia e Gestão (ESTG) [29] do Instituto Politécnico de Leiria (IPL) [33], foi realizado um projeto com aplicação prática (estágio) numa entidade empresarial, integrando assim o mercado de trabalho.

O estágio teve uma duração de 9 meses, tendo sido iniciado a 28 de setembro de 2016 e concluído a 27 de junho de 2017.

2.2 Local de Estágio

A entidade empresarial escolhida para realização do estágio foi a Sinmetro [59], empresa situada em Leiria.
Fundada em 2002, a Sinmetro - Sistemas de Inovação em Qualidade e Metrologia, LDA é uma empresa que trabalha com a Indústria e Serviços no Desenvolvimento de *Software*, Formação e Consultadoria em Projetos de Investigação e Desenvolvimento Tecnológico (I & DT), Inovação e Otimização & Reengenharia de Processos, Gestão

Estratégica e Sistemas de Financiamento.

As principais atividades e competências da empresa são:

- Consultaria em Projetos de Inovação, Gestão Estratégica e Financiamento.
  - Definição estratégica de projetos para o desenvolvimento de novos produtos, serviços e/ou processos, bem como na otimização de operações;
  - Definição e implementação de planos estratégicos e operacionais, modelos de inovação, entre outras áreas, com a respetiva definição de KPIs (*Key Performance Indicators*) específicos para cada atividade.
- Implementação de Sistemas de Informação: Accept by Sinmetro
  - O sistema Accept é um Software de Controlo Estatístico e Otimização de Processos focada na metrologia de pré-embalados, 100 % desenvolvido pela Sinmetro (Ver Secção 3.1.2 do Capítulo 3).
- Formação em Ferramentas no âmbito da Otimização Industrial & *Coaching* de Projetos de Re-engenharia.

Os clientes da Sinmetro são, na totalidade, do sector industrial (alimentar, automóvel e indústria química) para os quais a empresa presta serviços de desenvolvimento de *software*, consultaria e/ou formação. Atualmente, o número de clientes Sinmetro é de 180, segundo a responsável da empresa.

A área de atuação do estágio está relacionada com a Metrologia no conceito de pré-embalados, nomeadamente com o sistema Accept.

O sistema Accept permite auxiliar as empresas no enchimento dos produtos de pré-embalados, processo que necessita de obedecer a regras e leis para que seja considerado dentro dos conformes. Atualmente, são aproximadamente 60 as empresas que utilizam o sistema Accept, num total de 414 licenças Accept, espalhados por Portugal, Espanha e Angola, sendo que, em Angola, está instalado o sistema Accept na fábrica SUMOL+COMPAL, cliente da empresa em Portugal [60].

A Sinmetro, em 2016, apresentou uma faturação de 386.463,25 euros, um aumento de 62 % em relação ao ano anterior tendo sido faturado, em 2015, um total de 238.333,76 euros. No gráfico apresentado na Figura 2.1 podemos visualizar a faturação das diferentes atividades da empresa (Accept, consultoria e formação) no ano 2016 e compará-la

com o ano anterior. O sistema Accept representa 44,5% da faturação (que corresponde ao total de 173.193,61 euros) da empresa. Em relação ao ano anterior, as vendas do sistema Accept tiveram um acréscimo de 29%.

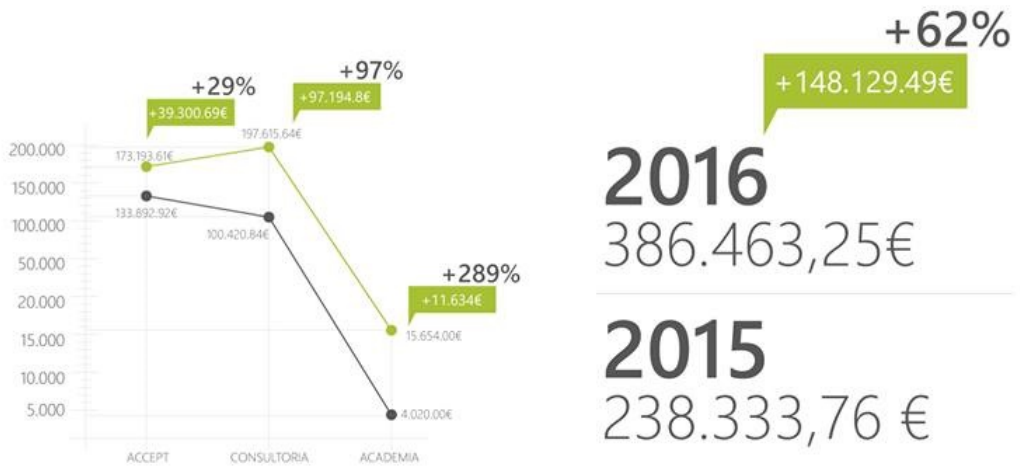

Figura 2.1: Faturação da empresa, no ano 2016, dividida pelas principais atividades. Gráfico disponibilizado pela Sinmetro.

Segundo informações dos responsáveis da empresa em 2016, a Sinmetro previu que a faturação de 2017 atingisse um volume de negócios no valor de 400.000,00 euros, um acréscimo de 4% em relação ao ano anterior. O gráfico representado na Figura 2.2 mostra as metas que a empresa pretende alcançar, no ano 2017, para as suas principais atividades.

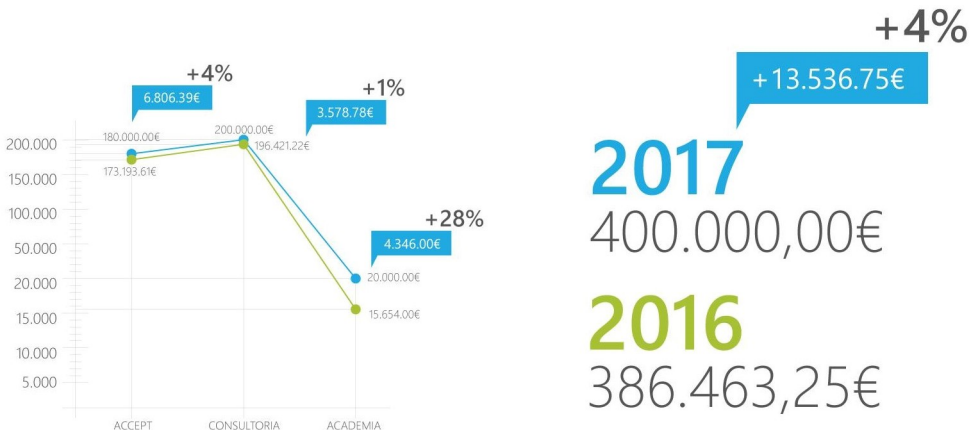

Figura 2.2: Faturação que a empresa pretende atingir no ano 2017, dividida pelas principais atividades. Gráfico disponibilizado pela Sinmetro.

2.3 Objetivo do Estágio

O estágio teve como objetivo o desenvolvimento e alojamento de uma API(*Application Programming Interface*) para aplicação *cloud* em modelo SaaS (*Software as a Service*) com vista à integração de uma aplicação *Web* em desenvolvimento.

Essa aplicação *Web* transporta as funcionalidades já existentes num dos módulos do sistema Accept, o Accept *gml* versão *desktop*, para a plataforma *Web*, permitindo às empresas, de uma forma ubíqua e em tempo real, fazer uma gestão de utilizadores, perfis, produtos, embalagens, fornecedores, linhas de produção e dispositivos das empresas (*backoffice*). Para além disso, realiza também uma recolha de dados sobre pesagens e processos de produção de produtos de pré-embalados para tratamento estatístico (cálculos estatísticos, elaboração de histogramas, cartas de controlo de *Shewhart*, entre outros).

A aplicação *Web* consome a informação da API e transforma-a em interface gráfica para o cliente. Assim, a API será responsável pela troca de informações entre o servidor da aplicação *Web* e a base de dados bem como o tratamento estatístico de dados.

A API é baseada no REST e foi desenvolvida em *Python* devido ao vasto conjunto de bibliotecas de estatística e de análise de dados que suporta.

A aplicação *Web* e a API desenvolvidas pertencem a um dos módulos Accept, o Accept *cloud*, módulo que se encontra em fase de desenvolvimento.

2.4 Condições do Estágio

O espaço físico da Sinmetro é partilhado com a empresa Aferymed [5], um Organismo de Verificação Metrológica de Pré-embalados Líquidos e Sólidos para todo o território nacional, num total de 12 colaboradores (9 colaboradores da Sinmetro e 4 colaboradores da Aferymed).

A empresa disponibilizou um computador portátil com o sistema operativo *Windows* 10 onde foi desenvolvido todo o trabalho realizado no estágio. Também foi instalada uma máquina virtual *Linux* (*Ubuntu* versão 16.04.1 LTS) no computador da empresa para alojar a API. A máquina virtual é acedida através da rede local.

No desenvolvimento da API foi necessário integrar a equipa de desenvolvimento do sistema Accept constituída por 9 elementos bem como compreender o funcionamento global do sistema. Além disso, foi necessário existir uma adaptação às metodologias de desenvolvimento que a Sinmetro utiliza.

Capítulo 3

Estado de Arte

O Estado de Arte ou Estado do Conhecimento é uma das partes mais importantes de todo trabalho científico.

É neste capítulo que é feito uma pesquisa sobre as áreas de conhecimento necessárias ao projeto fazendo referência a estudos já existentes.

As áreas de conhecimento necessárias inserem-se na Metrologia, nomeadamente no contexto dos produtos pré-embalados e na arquitetura REST.

3.1 Metrologia

Nesta secção pretende-se fazer uma introdução ao conceito de Metrologia, nomeadamente à Metrologia Legal no contexto dos produtos pré-embalados.

A metrologia é uma "*ciência da medição*"que tem como principal objetivo garantir o rigor nas medições efetuadas, recorrendo a unidades padronizadas para a comparação de grandezas da mesma natureza. Entende-se por medição o "*conjunto de operações que tem por objetivo determinar um valor de uma grandeza*". O rigor de uma grandeza é, geralmente, garantido através da calibração de equipamentos de medição de acordo com a grandeza a medir [73].

A Tabela 3.1 representa, num sentido geral, os sete tipos de grandezas e as respetivas unidades de base, segundo o Sistema Internacional de Unidades (SI), um sistema de unidades de medida adotado e recomendado pela Conferência Geral de Pesos e Medidas (CGPM).

A metrologia pode ser dividida em 3 áreas distintas [72]:

Tabela 3.1: Tipos de grandezas existentes e respetivas unidades de base segundo o SI. Adaptado do documento Vocabulário Internacional de Metrologia.

Grandeza	**Unidade SI**	
	Nome	Símbolo
Comprimento	metro	m
Massa	quilograma	kg
Tempo	segunndo	s
Corrente Elétrica	ampère	A
Temperatura Termodinâmica	kelvin	K
Quantidade de Matéria	mol	mol
Intensidade Luminosa	candela	cd

- a **Metrologia Científica** recorre à ciência para realizar as unidades de medida. Utiliza instrumentos laboratoriais e de pesquisa e metodologias científicas que têm por base padrões de medição nacionais e internacionais para o alcance de altos níveis de qualidade metrológica, apresentando o nível mais alto de exatidão;

- a **Metrologia Industrial** (ou aplicada) é utilizada em sistemas de medição que controlam processos produtivos industriais garantindo a qualidade dos produtos acabados. As medições são feitas em processos de fabrico;

- a **Metodologia Legal** está relacionada com sistemas de medição usados nas áreas de saúde, segurança e/ou meio ambiente. É realizada por organismos competentes que garantem que as unidades, métodos e instrumentos de medição cumprem com a legislação aplicável aos mesmos.

No âmbito do estágio foi realizado um estudo sobre a Metrologia Legal no contexto dos pré-embalados, apresentado na secção seguinte.

3.1.1 Controlo Metrológico de Pré-embalados

Um produto pré-embalado, segundo o Decreto Lei n.º 199/2008 de 8 de outubro, é "*um produto cujo acondicionamento foi efetuado antes da sua exposição ao consumidor, numa embalagem que solidariamente com ele é comercializada, de tal modo que a quantidade de produto contida na embalagem tenha um valor previamente escolhido e não possa ser alterado sem que a embalagem seja aberta ou sofra uma alteração percetível.*"[14].

Os pré-embalados podem ser classificados de acordo com a sua natureza [6]:

- Pré-embalados Líquidos (Bebidas, Tintas, Detergentes, entre outros);
- Pré-embalados Sólidos (Sal, Massa, Bolos, entre outros):
  - Pré-embalados Congelados e Ultra-congelados Vidrados (Pescado Congelado, por exemplo);
  - Pré embalados com Peso Escorrido (Conservas de Peixe, por exemplo).

De modo a garantir ao consumidor que a quantidade paga pelo produto é a que efetivamente vem descrita na embalagem, é necessário submeter os produtos a operações de controlo metrológico.

O controlo metrológico é um requisito legal controlado pelo Estado, da competência do Instituto Português da Qualidade (IPQ), um organismo do Ministério da Economia, participando os Serviços de Metrologia das Câmaras Municipais, Serviços Concelhios de Metrologia e os Organismos de Verificação Metrológica (OVM) [36].

Segundo o IPQ, o controlo metrológico assume um "*papel determinante na defesa do consumidor e dos cidadãos em geral e na arbitragem de conflitos entre os vários parceiros interessados na ciência da medição*"[34] garantindo, assim, o rigor das medições efetuadas com os instrumentos de medição [35].

No contexto dos pré-embalados podemos definir dois tipos de controlo metrológico [74]:

- o **controlo metrológico legal** que impõe critérios de aceitação e gamas para os lotes de pré-embalados (verifica e fiscaliza);
- o **controlo metrológico em fábrica** que controla os processos de modo a que se cumpra todos os requisitos legais e se evite desperdícios.

Enquadramento Legal do Controlo Metrológico de Pré-embalados

De modo a conciliar as regulamentações e os controlos metrológicos aplicados pelos serviços nacionais e/ou organizações de metrologia dos Estados Membros, a Organização Internacional de Metrologia Legal (OIML), uma organização intergovernamental mundial, elaborou, em 1989, a recomendação 87.

A recomendação 87 (OIML R87-e04) é um documento que descreve os procedimentos das atividades de verificação do conteúdo efetivo e os métodos estatísticos, que

devem ser adotados, aquando do controlo metrológico de pré-embalados. Este documento tornou-se numa referência para legisladores e estatísticos, sendo que, em 2004, foi publicada uma nova revisão do documento que se encontra disponível no sítio *Web* `https://www.oiml.org/en/files/pdf_r/r087-e04.pdf` [71].

Com base no documento OIML R87-e04 e na Diretiva Comunitária 2007/45/CE do Parlamento Europeu e do Conselho de 5 de setembro de 2007 que estabelece regras relativas às quantidades nominais dos produtos pré embalados, revoga as Diretivas 75/106/CEE e 80/232/CEE do Conselho e altera a Diretiva 76/211/CEE do Conselho [17], surgiram regras que o embalador/produtor deve cumprir.

Essas regras dão suporte às atividades do controlo metrológico de produtos pré-embalados e são baseadas em Legislação, dos quais se destacam os seguintes documentos legais :

- o Regulamento do Controlo Metrológico das Quantidades dos Produtos Pré-embalados (Portaria n.º 1198/91 de 18 de dezembro) que regulamenta o controlo metrológico de produtos pré-embalados, estabelecendo os critérios da média e dos conteúdos [43];
- o Decreto Lei n.º 291/90 de 20 de setembro que regulamenta o controlo metrológico dos métodos e instrumentos de medição [13];
- o Decreto Lei n.º 199/2008 de 8 de outubro [12] e a Declaração de Retificação 71/2008 [11] que estabelecem as regras relativas às quantidades nominais a que os pré-embalados devem obedecer;
- o Despacho 8146/2004 (2ª série) [16] e o Despacho 18853/2008 [15] que estabelecem as taxas aplicadas aos ensaios de controlo metrológico de pré-embalados.

É importante referir que os ensaios definidos na OIML R87-e04 e transpostos para o direito nacional pela Portaria 1198/91 de 18 de dezembro não devem ser implementados pelos embaladores/produtores mas sim pelas entidades oficiais no âmbito de uma fiscalização. A entidade competente e responsável pela fiscalização do controlo metrológico designa-se por Autoridade de Segurança Alimentar e Económica, mais conhecida como ASAE.

Segundo a Portaria 1198/91 de 18 de dezembro, o controlo metrológico de produtos pré-embalados destinados à comercialização deve ser feito a "*quantidades ou capacidades nominais unitárias iguais ou superiores a 5 g ou 5 mL e iguais ou inferiores a 10*

kg ou 10 L", pelo menos uma vez por ano, por uma entidade competente [43].
Essa entidade é uma entidade externa reconhecida pelo IPQ, conhecida como OVM (Organismo de Verificação Metrológica). Um OVM de pré-embalados é responsável pelos ensaios de verificação metrológica da Portaria 1198/91 de 18 de dezembro. Este deve avaliar os sistemas de medição usados pelo embalador/produtor no controlo da quantidade pré-embalada, analisar os registos do controlo do processo realizados periodicamente pelo embalador, emitir um relatório de avaliação da conformidade metrológico dos lotes verificados e emitir um certificado caso o controlo metrológico se enquadre nos limites legais.

De acordo com presente regulamento, o responsável pelos pré-embalados deve colocar as suas instalações à disposição das entidades competentes bem como os meios auxiliares indispensáveis à execução do controlo metrológico [75].

Existem diferentes métodos e instrumentos de medição que são escolhidos de acordo com o tipo de produto pré-embalado a ser analisado. Quanto ao tipo de ensaio de verificação metrológica, existem dois tipos de ensaios:

- **Ensaio não destrutivo**
  Neste tipo de ensaio é possível determinar o peso do produto através da subtração da tara da embalagem vazia ao peso da embalagem cheia. A embalagem não é violada na verificação e, assim, o produto pode ser comercializado, conforme o resultado do controlo metrológico.
  Geralmente dá-se preferência a este tipo de ensaio;

- **Ensaio destrutivo**
  Neste tipo de ensaio é necessário abrir a embalagem do produto para que se possa ter acesso ao peso real do produto. No final de um ensaio destrutivo o produto não pode ser comercializado. Este ensaio só é realizado quando o seu processo de embalamento não permita que este seja feito de outra forma.

Cada tipo de ensaio apresenta um plano de amostragem diferente de acordo com o presente regulamento.

Quanto à escolha do tipo de instrumento de medição a utilizar, de acordo com o Decreto Lei n.º 291/90 de 20 de setembro que regulamenta um regime de controlo metrológico dos métodos e instrumentos de medição abrangidos pela regulamentação relativa a pesos, medidas e aparelhos de medição [13], esta deve ser a mais adequada para que as medições apresentem um elevado grau de exatidão. A exatidão das medi-

ções é "*altamente dependente das propriedades metrológicas do instrumento de medição*"(nível de sensibilidade e precisão) "*determinadas pela sua calibração*"[73]. Desta forma, os instrumentos de medição também estão sujeitos a um controlo metrológico, de modo a garantir a exatidão do resultado das medições dentro de limites legalmente estabelecidos [34].

Relativamente às regras relativas às quantidades nominais a que os pré-embalados devem obedecer estas são estabelecidas pelo Decreto Lei n.º 199/2008 de 8 de outubro [12]. Pelo ponto 3, do Artigo 5.º do presente Decreto Lei, deve existir um controlo estatístico às quantidades pré-embaladas. De acordo com o artigo 13.º do Decreto-Lei n.º 291/90 [13], de 20 de setembro, a infração às normas relativas às operações de controlo metrológico legal incorre em contraordenação punível com coima. O montante mínimo da coima será de 5000 euros e o máximo de 150 000 euros quando a contraordenação for praticada por pessoa singular e de 50 000 a 1 500 000 euros quando praticada por pessoa coletiva. Estes montantes encontram-se definidos no n.º 2 do referido artigo.

Para além do controlo metrológico, o embalador deverá não só conservar os registos das medições periódicas como proceder às correções e ajustes necessários ao cumprimento da lei. Esses registos devem ser conservados num período que poderá ir de 1 a 5 anos, em função do prazo de validade dos produtos.

(a) Um ano para produtos com prazo de validade até 3 meses;

(b) Três anos para produtos com prazo de validade entre 3 e 18 meses;

(c) Cinco anos para produtos com prazo de validade mínimo superior a 18 meses.

O embalador deve garantir que a média acumulada dos seus processos de enchimento não é inferior à quantidade nominal e que a probabilidade de obter unidades defeituosas (quantidades inferiores ao erro admissível por defeito) não é superior a 2,5%, de acordo com as alíneas a) e b) do ponto 3 do Artigo 4.º. Nenhum pré-embalado deve ter um erro, por defeito, superior ao erro admissível. Os erros admissíveis por defeito para diferentes quantidades nominais estão representados na Tabela 3.2 que representa o quadro I da Portaria n.º 1198/91 de 18 de dezembro [43] que apresenta os erros admissíveis por defeito (EAD) nos conteúdos efetivos.

Tabela 3.2: Erros admissíveis por defeito. Adaptado do quadro I da Portaria n.º 1198/91 de 18 dezembro [43].

Quantidade nominal (grama ou mililitro)	**Erros admissíveis por defeito**	
	Percentagem	**Em massa ou volume (grama ou mililitro)**
Até 50	9.0	-
De 50 a 100	-	4.5
De 100 a 200	4.5	-
De 200 a 300	-	9.0
De 300 a 500	3.0	-
De 500 a 1000	-	15.0
De 1000 a 10 000	1.5	-
De 10 000 a 15 000	-	150.0
Superior a 15 000	1.0	-

Se o controlo metrológico apresentar um resultado positivo, isto é, se o pré-embalado se enquadrar nos critérios legais definidos, é emitido um certificado. Após a obtenção do certificado, com validade até dia 31 de dezembro do ano seguinte, o pré-embalado poderá conter, no seu rótulo, a marca de conformidade "e"representada na Figura 3.1.

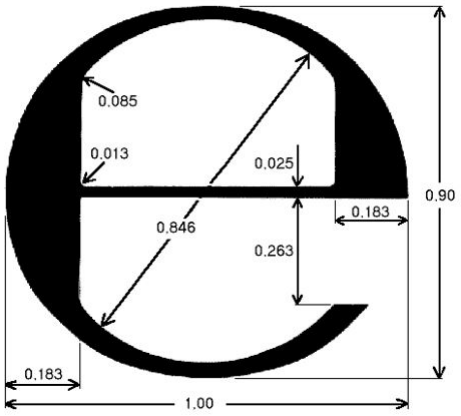

Figura 3.1: Marca de conformidade "e". Imagem retirada do anexo II do Decreto Lei n.º 199/2008 de 8 de outubro de 2010.

Segundo o Decreto Lei n. º 199/2008 de 8 de outubro de 2010, esta marca garante ao consumidor que a embalagem satisfaz os critérios legais definidos. Para isso, deverá "*obedecer ao grafismo indicado no anexo ii e ser colocada no mesmo campo visual que a*

indicação da quantidade nominal". Deverá ter uma altura mínima de 3 mm de altura e deverá manter todas as proporções caso sofra mudanças de tamanho [12].

Se um pré-embalado não obedecer aos critérios legais, deverão ser tomadas as medidas necessárias para corrigir os erros detetados. As medidas tomadas serão feitas nas própria hora ou numa data futura para que o ensaio seja novamente refeito até que se obtenha um resultado positivo.

Em suma, um embalador para cumprir o controlo metrológico de pré-embalados tem de possuir os equipamentos necessários, apresentar os procedimentos escritos e instrumentos de trabalho do controlo metrológico, com a indicação das técnicas estatísticas, arquivar e manter os registos do controlo metrológico durante pelo menos um ano e notificar anualmente uma entidade competente externa para a verificação metrológica.

Controlo Metrológico em Fábrica

O controlo metrológico em fábrica tem como principais objetivos:

- Caracterizar os processos de enchimento;
- Controlar estatisticamente os processos de enchimento e principais métodos de registos;
- Analisar os registos do controlo estatístico dos processos de enchimento;
- Avaliar o desempenho dos processos de enchimento de modo a evitar desperdícios com o sobre enchimento e tempos de paragens;
- Garantir os requisitos legais.

Deste modo, deve ser adotada a seguinte metodologia:

- Definição da dimensão da amostra a recolher em cada linha de embalamento;
- Definir uma frequência de amostragem para cada linha de embalamento;
- Avaliar o sistema de medição e se este apresenta a resolução adequada;

- Registar os dados de modo a assegurar a rastreabilidade das amostras (referência do produto, data, hora, n.º lote, operador, turno, etc);
- Analisar através de cartas de controlo o comportamento da variável volume ou massa e atuar de modo a evitar incumprimento legal ou desperdício por produto em excesso;
- Registar ocorrências e atuações sobre o processo;
- Analisar o histograma dos dados para avaliar o seu comportamento;
- Avaliar os indicadores de capacidade do processo.

3.1.2 *Software* de Controlo Metrológico

De acordo com um responsável técnico de realização de ensaios de verificação metrológica de pré-embalados da Aferymed, o registo das medições realizadas num ensaio metrológico pode ser feito através de um processo manual de inserção dos valores da amostra para uma folha de cálculo num computador ou para papel, através de balanças preparadas para recolher os dados das pesagens de uma amostra ou através de *software* de controlo metrológico. Estas diferentes alternativas auxiliam o processo do ensaio metrológico:

- **Processo manual**

  O registo manual dos valores das medições torna-se um processo pesado em termos de registo e armazenamento, sendo mais suscetível a erros. No caso do registo em papel, o acesso à informação e a análise estatística torna-se um processo demorado e difícil.

- ***Processo recorrendo a Software***

  A utilização de *software* no controlo metrológico permite otimizar a recolha dos valores da amostra, efetuar um tratamento estatístico e pesquisar registos, de forma eficiente e rápida.

Consoante o ritmo de produção, cabe ao embalador/ produtor definir o método que pretende para efetuar o registo e armazenamento das medições e, consequentemente, a análise estatística de dados de medições de pré-embalados. Essa escolha deve tornar o controlo mais eficiente.

Software de Balanças

O uso de *software* integrado em balanças permite o controlo e registo das medições feitas pelo embalador/produtor. Estas balanças permitem imprimir talões com a estatística de várias medições realizadas. A Figura 3.2 mostra um exemplo de talão impresso por uma balança.

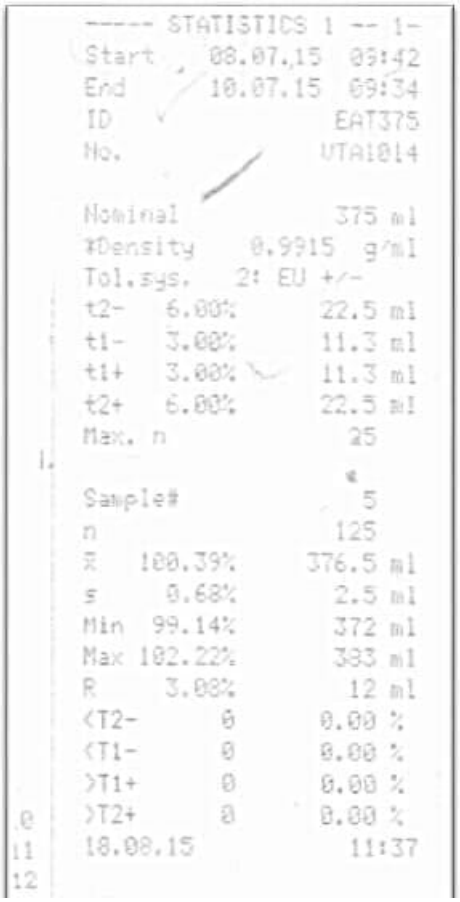

Figura 3.2: Talão impresso por uma balança que contém dados estatísticos sobre pesagens realizadas. Talão fornecida pelo técnico da Aferymed.

Uma das balanças mais utilizadas é a balança *FPVO Waage FKTF*. Esta balança de controlo obedece às normas de aferição de pré-embalados, permitindo, através de uma impressora externa própria, obter os resultados das pesagens realizadas bem como alguns dados estatísticos [8].

A desvantagem do uso deste *software* é o facto de ser bastante limitado em termos do tamanho de amostra e de estatísticas. Além disso, o armazenamento dos resultados é feito em formato de papel o que torna o acesso à informação num processo mais demorado.

Microsoft Excel

O uso da ferramenta *Excel*, um dos *softwares* desenvolvidos pela *Microsoft*, passa pela introdução do resultado da medição/pesagem de uma amostra na folha de cálculo. Essa introdução pode ser feita manualmente ou através da importação dos valores diretamente da balança para o *Excel*.

As folhas de cálculo *Excel* variam entre empresas, sendo que a informação, funcionalidades e análise estatística (gráficos, cálculos estatísticos, etc) depende das necessidades das mesmas.

As vantagens deste tipo de ferramenta são o armazenamento digital, o acesso à informação de forma rápida e fácil, apresentando um custo de manutenção bastante reduzido.

Sistema Accept

O sistema Accept é um Software de Controlo Estatístico e Otimização de Processos responsável pelos registos, medições e controlo da qualidade e eficiência da produção das empresas, criado em 2001 pela empresa SINMETRO.

É um sistema de informação integrado focado na metrologia de pré-embalados que tem como principais objetivos: garantir a qualidade da produção em todas as fases do processo, maximizar a capacidade das linhas de produção e reduzir custos e não conformidades.

Foi desenvolvido na linguagem de programação *Delphi* e está disponível para o sistema operativo *Windows*, na versão portuguesa. Permite a integração com outros sistemas de Informação e/ou equipamentos de medição para aquisição de dados e pode ser implementado através de módulos (módulos Accept).

- Módulos Accept

  O sistema Accept é constituído por 8 módulos *standard* que se encontram representados na Figura 3.3.

Figura 3.3: Módulos que constituem o Sistema Accept.

Cada módulo Accept pode ser adaptado às necessidades específicas de cada perfil de utilizador. São caracterizados pela sua simplicidade, facilidade na sua utilização e podem ser adquiridos individualmente. Através da Figura 3.4 é possível visualizar o funcionamento dos diferentes módulos que integram o sistema Accept.

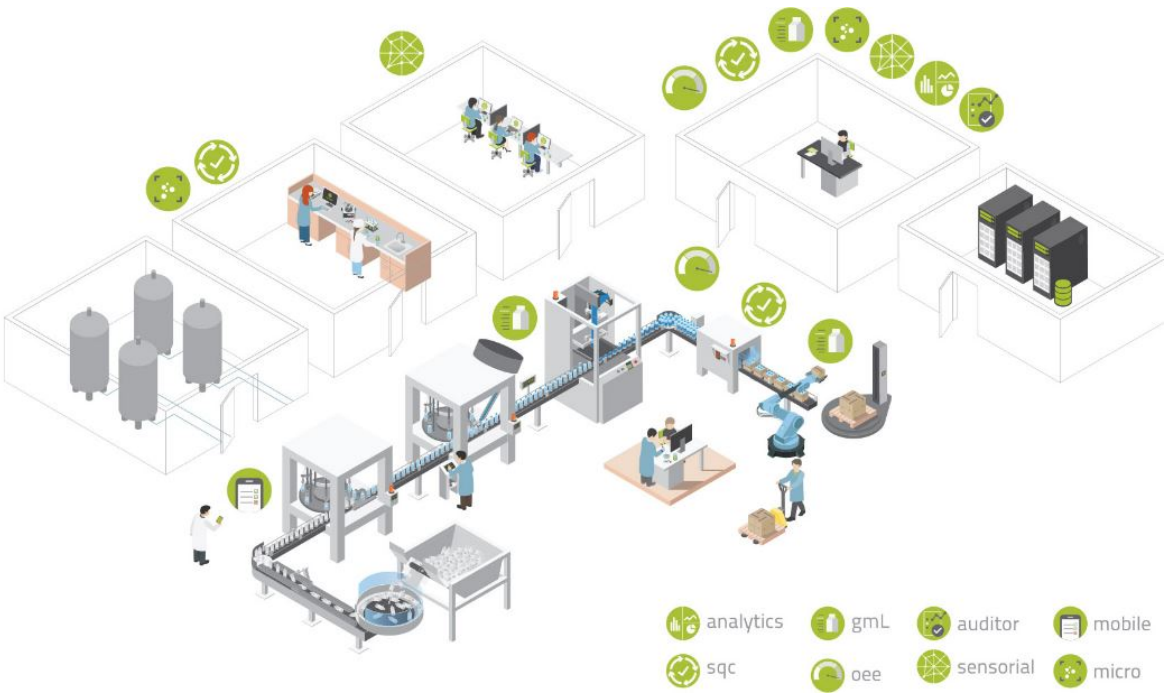

Figura 3.4: Funcionamento dos diferentes módulos que integram o sistema Accept. Imagem facultada pela Sinmetro.

Na Tabela 3.3 são descritos os diferentes módulos Accept mencionando os objetivos e funcionalidades de cada um.

Tabela 3.3: Descrição dos Módulos Accept. Informação retirada dos slides sobre o Sistema Accept disponibilizados pela empresa SINMETRO.

Módulo ACCEPT	Definição	Funcionalidades
analytics	**Definição de indicadores de avaliação do desempenho dos processos e análise da sua evolução** Este módulo agrega informação dos módulos ACCEPT: gml, sqc, oee, micro, sensorial e mobile.	- Permite comparar indicadores; - Verifica se os objetivos estão a ser atingidos; - Deteta as áreas onde o desempenho está abaixo do esperado.
gml	**Sistema de Controlo Estatístico da Quantidade de Pré-embalados**	- Evita o enchimento de quantidade por excesso em cada linha/ produto; - Garante o cumprimento dos critérios legais dos produtos de pré-embalados; - Tratamento estatístico, análise de dados e consequente análise de desvios, em tempo real.
sqc	**Sistema de Controlo Estatístico de Qualidade**	- Registo de Análises; - Análise de Não Conformidades; - Tratamento estatístico; - Elaboração de relatórios de acordo com as necessidades do cliente (relatórios à medida), com base na recolha dos dados e no tratamento estatístico realizado.
oee	**Sistema de Controlo da Eficácia das Linhas de Produção**	- Cálculo em tempo real dos indicadores de disponibilidade, velocidade e qualidade das linhas de produção.
micro	**Rastreabilidade de Amostras Microbiológicas**	- Identificação de amostras microbiológicas através de etiquetas de código de barras utilizadas em todas as fases do processo; -Análise estatística de amostras microbiológicas detetadas ao longo do processo.
mobile	**Interface móvel de recolha de dados** Aplicação para PDA destinada à gestão do ciclo de vida de provas sensoriais, desde a sua definição até à avaliação rápida dos dados.	- Registo, em qualquer local, das inspeções de controlo da qualidade integrando-as diretamente no ACCEPT.
sensorial	**Gestão de Provas de Análise Sensorial**	- Gestão do ciclo de vida das provas sensoriais, desde a sua definição até à análise estatística dos dados.
audit	**Gestão de auditorias**	- Realização de auditorias pela equipa de controlo de qualidade.

Arquitetura Accept e o seu funcionamento

A Figura 3.5 representa a arquitetura tradicional do sistema Accept (versão desktop) em funcionamento numa fábrica.

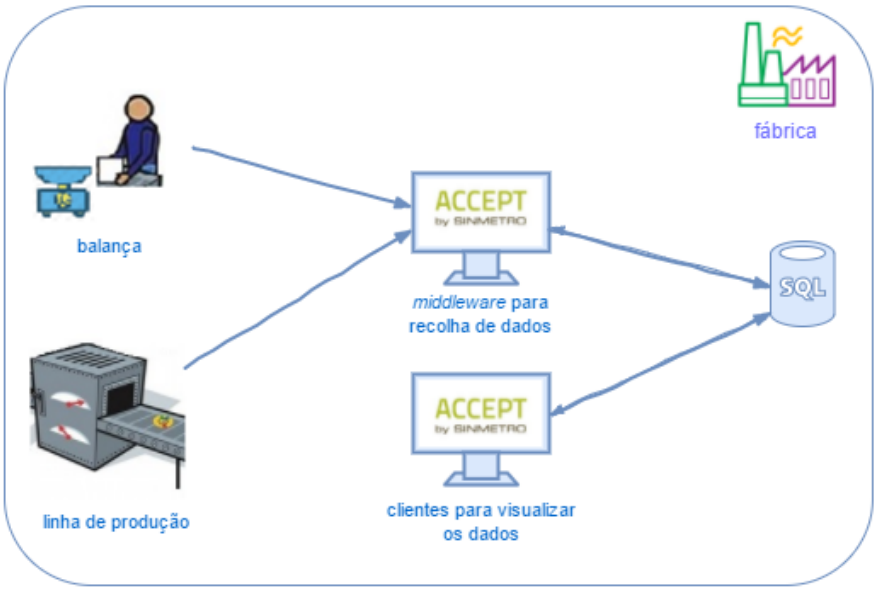

Figura 3.5: Arquitetura tradicional do sistema Accept em funcionamento numa fábrica.

Numa fábrica, a recolha dos dados das pesagens podem ser realizadas a partir de dois métodos distintos: através da pesagem manual utilizando uma balança ligada ao computador que está a executar o Accept ou através de sensores colocados ao longo das linhas de produção.

Na introdução dos dados no sistema Accept através do recurso à balança (processo manual), o computador utiliza a interface (ou portos) adequada a cada tipo de balança. O operador recolhe as amostras da linha de produção e efetua a aquisição das pesagens diretamente das balanças para o Accept, em tempo real, evitando enganos na introdução de dados e automatizando o processo de pesagem. Trata-se de um controlo metrológico manual. Este processo está representado na Figura 3.6.

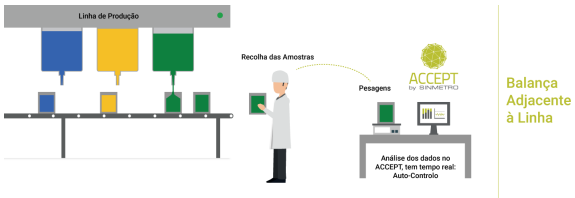

Figura 3.6: Processo manual de pesagem utilizando uma balança.

Na aquisição dos dados a partir de sensores (aquisição direta), os valores das pesagens são enviadas, em tempo real, através de sensores de peso situados nas linhas de produção. O número de sensores existentes depende da fábrica e dos produtos que ela produz. Na Figura 3.7 está representado este método de pesagem.

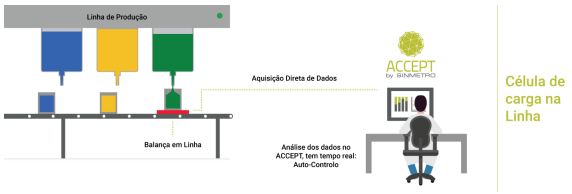

Figura 3.7: Processo manual de pesagem utilizando uma balança.

Em alguns casos, os sensores de peso são usados para determinar o peso da embalagem vazia, e posteriormente, são usados outros sensores colocados mais à frente na linha de produção para determinar o peso do produto após o seu enchimento. Assim, é automaticamente calculada a massa líquida do produto tendo em conta o seu peso antes e depois do enchimento. Noutros casos é utilizado um sensor no final da linha de produção para determinar a massa bruta do produto, fazendo uso de uma tara média previamente calculada para determinar a massa líquida.

Em ambos os métodos de recolha de dados, os dados das respetivas pesagens são enviados para um computador com o Accept instalado que será responsável pelo tratamento estatístico dos mesmos bem como guardar toda a informação recolhido (dados das pesagens e dados estatísticos) numa base de dados Microsoft SQL (*Structured Query Language*) *Server*. Este computador funciona como *middleware* entre a balança/linha de produção e a base de dados. Posteriormente, os dados das pesagens e das estatísticas podem ser visualizados em qualquer computador da fábrica desde que tenha o módulo Accept específico para o efeito instalado.

3.2 REST e REST*ful*

Nesta secção é feito um estudo sobre o estilo de arquitetura REST e sobre os serviços *Web* que implementam este estilo de arquitetura, o RESTful.

3.2.1 REST

O *Representational State Transfer* (REST), que em português significa Transferência de Estado Representacional, é um estilo de arquitetura de *software* da *World Wide Web* (WWW) desenvolvido para servir aplicações em *Web*.

O REST oferece um conjunto de linhas de orientação necessárias para se desenvolver um serviço coeso, escalável e com elevada performance. É independente da plataforma e da linguagem.

Segundo a dissertação do Dr. *Roy Fielding*, este estilo de arquitetura baseia-se nos seguintes princípios básicos [2]:

- Cliente-Servidor;
- *Stateless*;
- *Cacheable*;
- Interface Uniformizada;
- Sistema em camadas.

Cliente-Servidor

Uma aplicação REST deve separar a arquitetura e as responsabilidades em dois ambientes (cliente e servidor), tornando-se independente e consequentemente mais escalável.

O consumidor do serviço preocupa-se apenas com a interface enquanto o fornecedor de serviço (servidor) é responsável por devolver uma resposta ao cliente através da execução de um pedido enviado pelo cliente.

A Figura 3.8 representa a comunicação entre clientes e servidor através do protocolo HTTP.

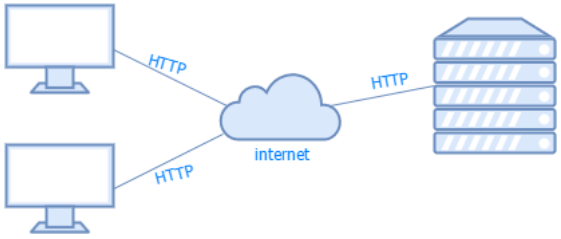

Figura 3.8: Modelo Cliente-Servidor

Stateless

O servidor não guarda nenhuma informação relativa ao estado do cliente. Essa informação é guardada no cliente. Um cliente pode fazer vários pedidos ao servidor. Cada pedido ao servidor é feito de forma independente e padronizada passando apenas a informação necessária ao servidor para que este possa processá-la de forma mais adequada e correta.

Cacheable

De modo a evitar processamento desnecessário e aumentar significativamente a performance, quando um cliente faz um pedido ao servidor, a resposta fica armazenada temporariamente em *cache*. Assim, se vários clientes fizerem o mesmo pedido ao servidor, este devolve o que está em *cache* sem ter que voltar a processá-lo, melhorando a eficiência, escalabilidade e desempenho pelo utilizador. No entanto, a informação em *cache* quando muito utilizada pode diminuir a confiabilidade dos dados tornando-os obsoletos, pelo que será importante a implementação de um *cache* de *gateway* (ou *proxy* reverso), isto é, um servidor de rede (camada independente) que armazena em *cache* as respostas à medida que são retornadas, reutilizando-as para pedidos futuros, diminuindo assim o número de interações diretas ao servidor.

Interface Uniformizada

A comunicação entre clientes e servidor obedece a regras simples que seguem determinadas linhas orientadoras, que quando bem definidas, tornam a comunicação mais uniforme e normalizada. De seguida é apresentado um conjunto de linhas orientadoras baseadas no padrão de arquitetura definido.

- Cada recurso é identificado através de um URI (*Uniform Resource Identifier*) específico e coeso;
- Uma URI representa recursos e não ações;
- As ações são representadas por verbos (ou métodos) HTTP;
- O formato em que o recurso pode ser devolvido ao cliente é escolhido de acordo com as necessidades específicas do fornecedor de serviço, sendo que os formatos

mais utilizados são o *JavaScript Object Notation* (JSON) e o *eXtensible Markup Language* (XML);

- O formato da comunicação cliente/ servidor deve ser definido no *Content Type*;
- O cliente recebe todas as informações necessárias na resposta para que possa navegar e ter acesso a todos os recursos da aplicação;
- Nos pedidos e nas respostas deve ser passada informação metadata (por exemplo, código HTTP, *Content-Type*, *Host*, entre outras).

Sistema em camadas

A aplicação deve ser composta por camadas que possam ser facilmente alteradas, adicionadas e/ou removidas.

Cada camada comunica ou processa informação entre cliente e servidor de forma específica ou individualizada.

Desta forma, por norma, o cliente não deverá chamar diretamente o servidor da aplicação sem antes passar por um *middleware*, por exemplo um *load balancer* ou uma máquina que faça interface com o servidor. O *middleware* fica responsável por distribuir os pedidos ao servidor, garantindo que cada uma das camadas execute funções específicas o que conduz a uma estrutura muito mais flexível a mudanças, proporcionando uma melhor performance, escalabilidade, simplicidade, flexibilidade, visibilidade, portabilidade, confiabilidade e segurança.

A Figura 3.9 representa um diagrama de rede que utiliza um *load balancer* para distribuir a carga pelos vários servidores .

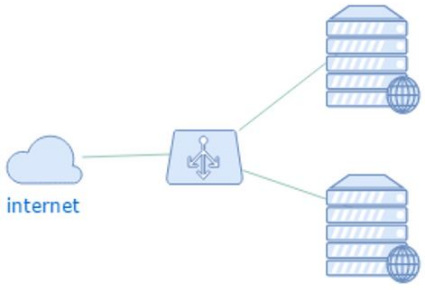

Figura 3.9: Modelo Cliente-Servidor com um *load balancer*

3.2.2 REST*ful*

A REST*ful* é um serviço *Web* que segue os princípios definidos na arquitetura REST. Para uma API (*Application Programming Interface*) ser considerada REST*ful* esta deve seguir estritamente as regras definidas na arquitetura REST bem como o modelo de *Richardson*, o modelo de Maturidade, representado na Figura 3.10 [53].

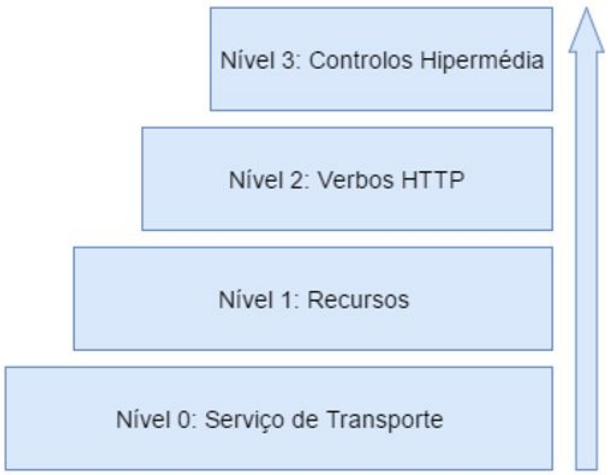

Figura 3.10: Modelo de Maturidade de *Richardson*

- **Nível 0 - Serviço de Transporte**

  O sistema de transporte utilizado para operações no servidor é o protocolo HTTP. Este utiliza um *endpoint* de serviço numa URI e um verbo HTTP;

- **Nível 1 - Recursos**

  A API é dividida em diferentes *endpoints* que apontam para um ou mais recursos (recursos individuais);

- **Nível 2 - Verbos HTTP**

  Implementação dos verbos (ou métodos) HTTP para as diferentes operações CRUD (*Create*, *Read*, *Update*, *Delete*) [7].

  – GET: Lê (ou recupera) um recurso identificado pela URI. Este método solicita o recurso ao servidor e retorna a informação no formato JSON e XML. É considerado um método seguro pois pode ser chamado sem risco de modificação ou corrupção de dados;

- POST : Cria novos recursos com a informação enviada no corpo do pedido. Através da URI consegue enviar informações ao servidor. Uma vez que essa não é retornável ao cliente, torna o POST mais seguro uma vez que os dados enviados não são expostos;
- PUT: Cria ou altera (atualiza) um conteúdo, substituindo toda a informação do recurso. A diferença entre o POST e o PUT é que no POST a URI indica qual o recurso onde se vai guardar informação enquanto o PUT indica o recurso específico onde a informação será armazenada. No método PUT, se o recurso já existir nessa URI, substitui o recurso. Caso contrário, é criado um novo recurso;
- PATCH - Modifica parcialmente o conteúdo de um recurso. Contém apenas a informação a alterar (partes) e não o recurso completo;
- DELETE - Elimina um recurso identificado por uma URI;

Na resposta ao pedido do cliente, o servidor HTTP retorna um código *status* que informa o resultado do pedido.
Esse código é composto por 3 dígitos e pode ser dividido em 5 classes *status* HTTP que classificam o tipo de resposta do servidor.

Na tabela 3.4 está representado as classes de *status* HTTP existentes e os códigos de *status* mais utilizados.

Tabela 3.4: Classes de *status* HTTP e respetivos códigos *status* HTTP mais utilizados. [31]

Classe de *status* HTTP	**Definição**	**Códigos *status* mais utilizados**
1XX	Informativa	100 (*Continue*)
2XX	Sucesso	200 (Ok) 201 (*Created*) 204 (*No Content*)
3XX	Redireccionamento	304 (*Not Modified*)
4XX	Erro do Cliente	400 (*Bad Request*) 401 (*Unauthorized*) 403 (*Forbidden*) 404 (*Not Found*) 409 (*Conflict*)
5XX	Erro do Servidor	500 (*Internal Server Error*)

Os métodos utilizados para as diferentes operações CRUD têm associados códigos de resposta que indicam o *status* da resposta HTTP.

A Tabela 3.5 representa os métodos utilizados para as diferentes operações CRUD e os código de *status* HTTP mais utilizados na arquitetura REST [69].

Tabela 3.5: Métodos HTTP utilizados para as diferentes operações CRUD.

Verbos HTTP	**Função CRUD**	**Status HTTP**
POST	*Create*	Sucesso: 201 (*Create*) Erro: 400 (*Bad Request*) 404 (*Not Found*) 409 (*Conflict*)
GET	*Read*	Sucesso: 200 (*Ok*) Erro: 400 (*Bad Request*)
PUT	*Update/Replace*	Sucesso: 200 (*Ok*) 204 (*No Content*) Erro: 404 (*Not Found*)
PATCH	*Update/Modify*	Sucesso: 200 (*Ok*) 204 (*No Content*) Erro: 404 (*Not Found*)
DELETE	*Delete*	Sucesso: 200 (*Ok*) 204 (*No Content*) Erro: 404 (*Not Found*)

Outros métodos utilizados embora não tão frequentemente são o HEAD, o TRACE, o OPTIONS e o CONNECT. Estes métodos são utilizados para recuperar metadados da URI.

– HEAD: Retorna os cabeçalhos de uma resposta sem que o corpo contenha o recurso. Este método é utilizador para testar a validade do último acesso

sem que seja enviada informação ao cliente;

  - TRACE: Este método é utilizado para enviar mensagens do tipo *loopback* para teste. A informação é enviada de volta ao cliente para que este veja se houve alterações/adições feitas por servidores intermediários.
  - OPTIONS: Retorna ao cliente as propriedades do servidor.
  - CONNECT: Este método utilizado para permitir a comunicação com servidores *Proxy*.

- **Nível 3 - Controlos Hipermédia**: Introdução de Hipertexto como mecanismo de Estado da Aplicação (HATEOAS - *Hypertext As The Engine of Application State*). O cliente interage com a aplicação através de hipertexto que identifica o recurso. Desta forma, a API deve fornecer ao cliente toda a informação necessária para interagir com a aplicação, permitindo uma maior navegabilidade.

Comparação entre SOAP e REST

Um serviço *Web* é, nos dias de hoje, uma tecnologia bastante utilizada na comunicação entre aplicações *Web* de diferentes plataformas. Essa comunicação é feita em rede através de protocolos de comunicação HTTP e HTTPS. As principais alternativas de arquiteturas de serviços *Web* que permitem a comunicação entre o *frontend* e o *backend* são o SOAP (*Simple Object Access Protocol*) e o REST. De seguida é apresentado uma análise comparativa destes dois tipos de arquitetura.

O SOAP, comparando com o REST, é um protocolo de transferência de mensagens em formato XML para uso em ambientes distribuídos e é independente da plataforma e transporte (HTTP, TCP, SMTP, *etc*). Enquanto o REST requer apenas o uso do protocolo HTTP e assume uma comunicação direta de ponto a ponto. Numa arquitetura SOAP, o serviço *Web* é orientado a ações enquanto numa arquitetura REST o serviço é orientado aos recursos, em que as ações (verbos HTTP) manipulam diretamente os recursos.

O REST poderá utilizar diferentes tipos de formatos sendo que, o formato das mensagens deve ser definido de acordo com a linguagem/aplicação que está sendo desenvolvida enquanto o SOAP está limitado ao XML como formato de resposta. Apesar do SOAP ser uma linguagem mais fácil de ser interpretada, uma vez que os dados estão armazenados em elementos XML, o documento XML torna-se mais pesado devido a uma maior quantidade de informação a ser transportada entre os sistemas, exigindo uma maior capacidade de processamento o que, consequentemente, exige mais tempo. Em contrapartida, o REST não utiliza elementos XML para identificar recursos, ocupando assim menos espaço. Os recursos e as diferentes operações sobre os mesmos estão

disponíveis através de URLs. Assim, a transferência e o processamento dos dados são mais rápidos.

Por ser considerado um dos tipos de serviço *Web* mais rápido, objetivo, simples e de fácil aprendizagem e utilização, o REST ganhou uma grande popularidade no desenvolvimento *Web* e na disponibilização de APIs de acesso público.

Em suma, apesar das vantagens e desvantagens que cada tipo de serviço *Web* apresenta, podemos concluir que o REST é um excelente opção para casos onde há limitação de recursos e de largura de banda, para realizar operações CRUD sem estado e para casos onde a informação precisa ser guardada em *cache*. É um serviço *Web* extremamente flexível uma vez que não existe restrições no formato em que a informação é devolvida mas no comportamento dos componentes envolvidos. O SOAP é a melhor solução para instituições com padrões rígidos e ambientes complexos onde a integridade, segurança e confiabilidade dos dados são fatores importantes a ter em conta na transferência dos mesmos [61].

Capítulo 4

Arquitetura

Neste capítulo é apresentada a arquitetura do ACCEPT *cloud* definida pela Sinmetro, bem como os padrões arquiteturais que definem a API REST *Python* desenvolvida.

4.1 Arquitetura Accept *cloud*

O Accept *cloud* é constituído por diferentes componentes responsáveis pelo processo decorrido desde o preenchimento da base de dados até à visualização da informação por parte do cliente.

A Figura 4.1 representa a arquitetura geral do Accept *cloud* e nela podemos visualizar como os vários componentes comunicam entre si, bem como as suas dependências.

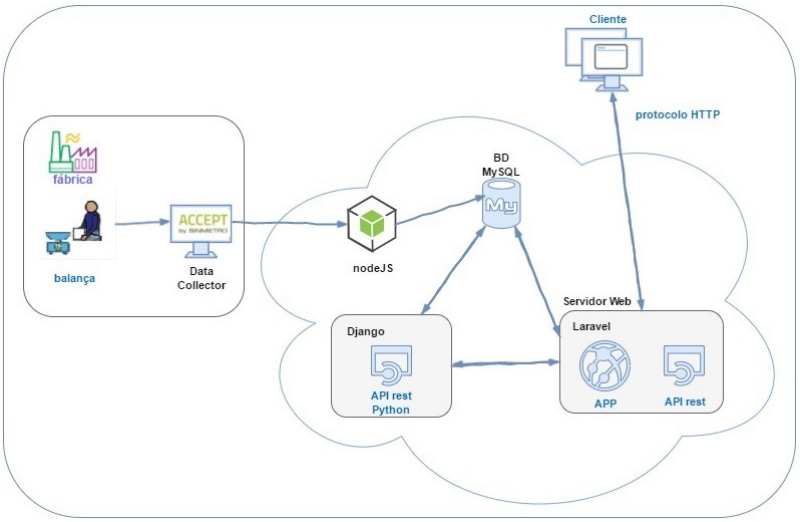

Figura 4.1: Arquitetura Geral do Accept *cloud*

A API REST desenvolvida no contexto deste estágio foi desenvolvida através do *framework Django* REST e encontra-se alojada num servidor *Linux*. Esta API contém os métodos relativos à gestão de configurações (*backoffice*) bem como os métodos relacionados com o tratamento estatístico. Esses métodos são utilizados pelo servidor *Web* e são responsáveis por realizar as operações CRUD na base de dados MySQL.

O cliente (*browser*) é responsável pelas configurações de produtos, embalagens, fornecedores, linhas de produção, dispositivos das empresas e processos (gestão de *backoffice*). A recolha de dados das pesagens pode ser obtida através de balanças ou diretamente das linhas de produção da empresa. Para isso, o cliente Accept existente na fábrica deverá conter a aplicação *Data Collector* a executar.

O *Data Collector*, ao receber o valor da pesagem, será responsável por o enviar para um servidor *nodeJS*, através de um canal que é criado a partir de um *token* que representa uma balança/linha de produção. A aplicação *Web* está à escuta do canal do servidor *nodeJS* que contem o respetivo *token*.

O servidor *Web* foi desenvolvido através do *framework Laravel* e aloja a aplicação *Web* e a API REST responsável pela gestão dos dados das empresas e utilizadores, bem como pela autenticação no sistema.

O utilizador realiza pedidos HTTP ao servidor *Web* através do *browser*. Após a receção do pedido HTTP, o servidor *Web* utiliza um mecanismo de rotas que associa o caminho do URL num método interno responsável por adquirir/tratar informação da base de dados.

4.2 Padrões Arquiteturais

Nesta secção são apresentados os padrões arquiteturais que a tecnologia *Django* REST recorre: o padrão MVT (*Model-View-Template*) e o padrão ORM (*Object Relational Mapping*). Com base nestes padrões foi possível estruturar a arquitetura da API REST desenvolvida.

4.2.1 Padrão MVT

Antes de começar a implementar a API REST foi necessário compreender a arquitetura que o *framework Django* implementa. Para isso foi feita uma pesquisa sobre a mesma, cujos resultados se apresentam de seguida.

De acordo com o *Django Book*, o *Django* é considerado um *framework Model-View-Controller* (MVC), no entanto segue o seu próprio padrão de arquitetura, o *Model-View-Template* (MVT), em que considera o *Controller* como "*View*"e a *View* como "*Template*"[64].

Na Figura 4.2 é representado o padrão arquitetural MVT implementado pelo Django.

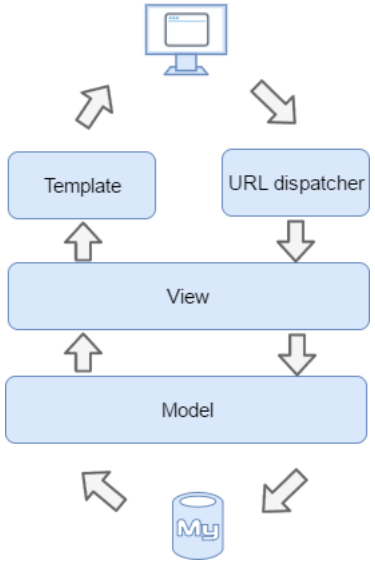

Figura 4.2: Padrão Arquitetural MVT

A estrutura do padrão MVT é composta por três camadas:

- ***Model***

  A camada *Model* (camada de acesso à base de dados) é composto por várias classes associadas às tabelas existentes na base de dados.

  Esta camada não contém os dados mas sim uma interface para os mesmos, que inclui toda a informação relacionada com a base de dados incluindo a sua estrutura, o acesso, validação e comportamento dos dados e relações entre os mesmos.

  A *framework Django* facilita na interface com a base de dados permitindo ao utilizador não se preocupar com a conexão entre as classes de domínio e a base de

dados. Para isso, recorre à técnica designada por Mapeamento Objeto-Relacional, do inglês *Object-Relational Mapping* (ORM).

- ***View***

  A camada *View* (camada lógica) define quais os dados que são apresentados ao cliente sendo responsável pelo tratamento lógico dos dados. É a camada *View* que comunica com o *Model* e o *Template*.

  Cada "*View*"é uma função de retorno para um URL específica. Essa função retorna ao cliente a informação no formato definido (XML, HTML, JSON, entre outros) ou os erros encontrados.

  Comparando com o modelo MVC, esta camada corresponde ao *Controller*. Neste padrão arquitetural, o controlador é responsável por controlar o fluxo de informação entre o modelo e a vista, isto é, decide quais as informações a serem obtidas na base de dados através do modelo e quais serão enviadas para a visualização.

- ***Template***

  A camada *Template* (camada de apresentação) descreve a forma visual em que os dados são apresentados ao utilizador. Esta camada é composta por HTML, CSS, JavaScript, entre outros.

  No modelo MVC, esta camada corresponde à camada *View*.

4.2.2 Padrão ORM

O ORM é uma técnica que permite manipular e consultar dados de uma base de dados através de uma perspetiva orientada a objetos adaptado à linguagem de programação que está a ser utilizada em vez de recorrer a instruções SQL [41]. A Figura 4.3 mostra o padrão arquitetural ORM.

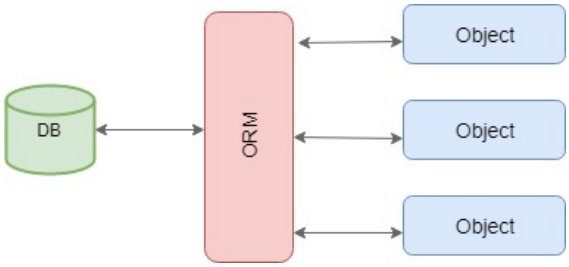

Figura 4.3: Padrão Arquitetural ORM

O ORM serve de ponte entre o modelo relacional (base de dados) e o mundo orientado a objetos (classes e métodos) fazendo o mapeamento às tabelas que constituem

a base de dados relacional para estruturas de dados. É na entidade *Model* que esse mapeamento é feito.

É importante que referir que o mapeamento apenas é feito à base de dados e não aos registos. Os registos de cada tabela são representados por instâncias das classes correspondentes.

A Figura 4.4 mostra um exemplo de uma tabela pertencente à base de dados MySQL, a tabela *Suppliers*.

id	description	code	company_id
1	Fornecedor1	CodFornecedor1	133
2	Fornecedor2	CodFornecedor2	133

Figura 4.4: Tabela *Suppliers* pertencente à base de dados

O código abaixo mostra o mapeamento da tabela *Suppliers* realizado pelo *Django* ORM.

```
class Suppliers(models.Model):
    description = models.CharField(max_length=100, blank=True, null=True)
    code = models.CharField(max_length=100, blank=True, null=True)
    company_id = models.IntegerField(blank=True, null=True)

    def delete(self, *args, **kwargs):
        ...
        self.save()
        pass

    class Meta:
        managed = False
        db_table = 'suppliers_view'
        pass
```

Listagem 4.1: Mapeamento da tabela *Suppliers*

A API do ORM acede aos objetos do modelo através de um *manager*. Um *manager* é um objeto presente em cada *model*. Trata-se de uma classe que representa um tipo de objeto armazenado numa tabela da base de dados que permite consultar ou alterar a coleção de objetos do *model* na base de dados através de métodos como *all()*, *filter()* ou *delete()*.

Graças ao ORM há uma independência dos modelos em relação à base de dados, um acesso direto às diferentes componentes existentes na base de dados (objetos relacionados), uma implementação fácil e flexível das operações CRUD e uma validação

dos campos. Através do ORM é possível manipular e compreender os dados, de forma fácil e intuitiva. No entanto, o ORM trata-se de um procedimento que pode reduzir potencialmente a performance [39].

4.3 Arquitetura da API REST *Python*

Com base no padrão MVT definido pelo *framework Django* e na arquitetura REST que o *framework Django* REST implementa podemos visualizar através na Figura 4.5 uma representação esquemática do funcionamento da API REST *Python* quando esta recebe um pedido HTTP do cliente.

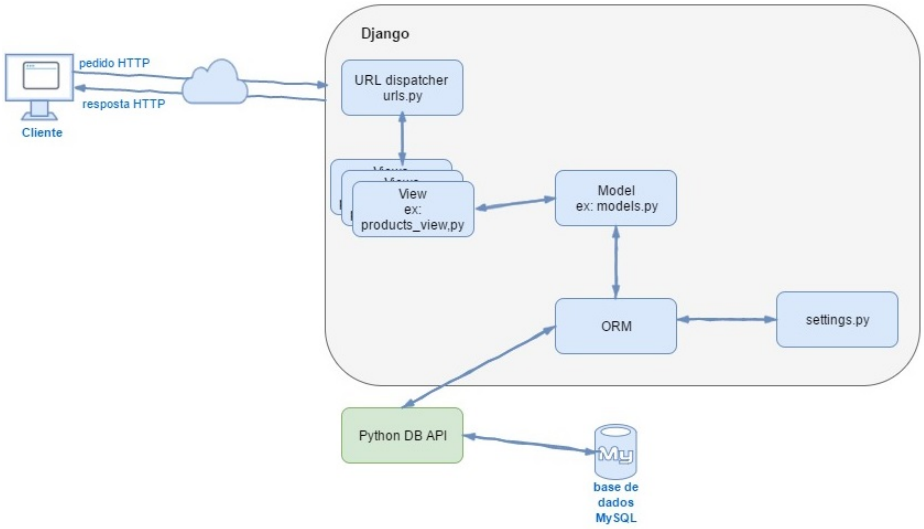

Figura 4.5: Arquitetura da API REST *Python*

Na Figura 4.5 podemos visualizar as componentes que fazem parte do projeto e como estes comunicam entre si no processo. De seguida é apresentada uma explicação detalhada do seu funcionamento.

Quando um cliente executa uma ação, o servidor *Web* recebe o pedido e através de um mecanismo de rotas, converte-o para um URL específico.
O distribuidor de URL do *Django*, com base no ficheiro *urls.py*, escolhe o primeiro método da API correspondente ao URL solicitado, que será responsável por invocar a função de retorno (*view*) associada. O ficheiro *urls.py* é responsável pelo mapeamento dos padrões de URL para as funções de retorno.

Cada *view* retorna um objeto *HttpResponse* com a informação a apresentar ao

cliente no formato JSON ou gera uma exceção/erro HTTP. Para obter informação da base de dados, a *view* recorre ao ficheiro *models.py*. Este ficheiro faz o mapeamento das tabelas/*views* que constitui a base de dados para classes Modelo. Esse mapeamento é feito recorrendo ao padrão ORM.

O ORM recorre ao *Python* Db API que contém um conjunto de módulos que permitem aceder à base de dados. O acesso à base de dados é feito através objetos de conexão e cursores que são utilizados para operações de consulta e de registos na base de dados. A API contém também métodos que fazem a validação dos dados, isto é, especificam o tipo de dados de uma coluna da tabela de modo a garantir que o campo a inserir naquela coluna tem o formato especificado.

O ficheiro *settings.py* contém todas as configurações do projeto, nomeadamente a informação necessária para a conexão à base de dados, fuso horário, idioma e informações sobre as aplicações instaladas no projeto.

Capítulo 5

Implementação

Antes da implementação da API REST foi necessário ter conhecimento sobre a metodologia que a Sinmetro utiliza bem como as tecnologias utilizadas. Após esse estudo, iniciou-se a implementação da API REST.

Neste capítulo serão apresentados os aspetos mais relevantes desta fase através de uma explicação detalhada das várias funcionalidades implementadas.

5.1 Metodologia

Na realização do estágio, de modo a desenvolver um projeto de *software* de qualidade e dentro do prazo definido, foram adotadas metodologias ágeis baseadas num conjunto de orientações que ajudam a gerir e planear projetos de desenvolvimento de software de forma dinâmica e rápida.

A metodologia de desenvolvimento implementada pela empresa foi a Prototipagem Evolutiva, um dos tipos de metodologia de Prototipagem de *Software*.

A Prototipagem de *Software* trata-se de uma metodologia iterativa que permite gerir modelos de *software* no ciclo de desenvolvimento do mesmo. A Figura 5.1 ilustra o processo de prototipagem em quatro fases.

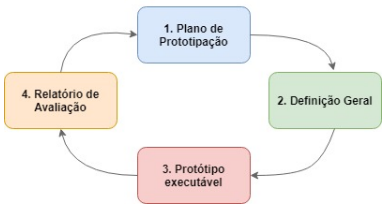

Figura 5.1: Processo de Prototipagem de *Software*

Esta abordagem envolve a produção de protótipos com base nos requisitos de *software* definidos tendo em conta a sua funcionalidade e interface. Desta forma, com base no processo de prototipagem apresentado na Figura 5.1, é necessário definir qual o objetivo do protótipo (fase 1) e, consequentemente, a definição das suas funcionalidades (fase 2). Após estas fases estarem definidas, o protótipo é desenvolvido (fase 3) para posteriormente ser testado e avaliado pelo utilizador final com o objetivo de obter *feedback* sobre o mesmo (fase 4), ou seja, através dos testes realizados aos protótipos é possível validar os requisitos definidos, detetar falhas e propor melhorias [57].

O protótipo pode sofrer várias alterações até que o resultado final corresponda as necessidades e expetativas do cliente, apresentando uma maior usabilidade no sistema, minimizando os riscos e maximizando os lucros.

Nesta metodologia, um protótipo permite responder a diferentes questões pelo que, segundo as suas finalidades e objetivos, enquadram-se em vários tipos de prototipagem. Existem quatro tipos de Prototipagem de *Software* que são apresentados de seguida:

1. **Prototipagem *Throw-Away* (Prototipagem descartável)**

   Neste tipo de prototipagem, os protótipos são desenvolvidos com o intuito de obter *feedback* do utilizador final que é responsável por testá-los e avaliá-los.

   Após *feedback* recebido, os protótipos são descartáveis. Este tipo de protótipos não fazem parte do sistema final. Tratam-se de protótipos incompletos, desenvolvidos de forma rápida, que têm como objetivo garantir que os requisitos estão bem definidos e bem compreendidos pela equipa de desenvolvimento, reduzindo assim o risco de requisitos mal definidos [65].

2. **Prototipagem Incrementativa**

   Neste tipo de prototipagem, os protótipos são desenvolvidos de forma independente para as diferentes funcionalidades do sistema. À medida que os protótipos são desenvolvidos, vão sendo integrados entre si até a obtenção do sistema final.

   Os utilizadores finais podem testar os protótipos em desenvolvimento com o objetivo de fornecer *feedback* que, pode ou não, levar a alterações no protótipo [32].

3. **Prototipagem Extreme**

   Este tipo de prototipagem é utilizada para o desenvolvimento de aplicações, nomeadamente aplicações *Web*. Tratam-se de protótipos funcionais de alta fidelidade que dividem o desenvolvimento de uma aplicação *Web* em três fases distintas:

(a) Criação de páginas estáticas apenas com elementos HTML (protótipo estático) e de um modelo de dados lógicos que deverá dar suporte às páginas criadas;

(b) Programação dos ecrãs e dos elementos HTML utilizando dados simulados (processo de codificação);

(c) Implementação dos serviços e funcionalidades e integração com a interface.

Este tipo de metodologia foca-se essencialmente na fase 2, com o objetivo de desenvolver uma página 100 % funcional antes da sua fase de implementação. Nesta fase, os utilizadores finais podem testar o protótipo funcional de modo a fornecer o *feedback* sobre o estado e possíveis alterações.
Além disso, permite que os elementos da equipa de desenvolvimento se foquem apenas nas suas tarefas e tomem, antecipadamente, decisões que garantam a integração de páginas e serviços com sucesso [51].

4. **Prototipagem Evolutiva**

Neste tipo de prototipagem, é desenvolvido um protótipo inicial que serve como base para todos os outros protótipos desenvolvidos. O protótipo define apenas os requisitos iniciais mais importantes e já bem definidos.

Este protótipo é apresentado ao utilizador final que é responsável por validá-lo, fornecer *feedback* do mesmo e sugerir possíveis melhorias. Com base no *feedback* obtido, a equipa de desenvolvimento é responsável por refinar o protótipo e apresentá-lo novamente ao utilizador final. Este processo é feito repetidas vezes até que o sistema final responda às expetativas do mesmo.

O utilizador final tem um papel importante no desenvolvimento do sistema pois garante que o sistema obedece aos requisitos necessários, evitando riscos causados pelo fraco conhecimento dos requisitos por parte da equipa de desenvolvimento. No entanto, é importante saber parar de ajustar o sistema com alterações que em nada influenciam os requisitos do sistema pois assim o desenvolvimento do sistema tornar-se-á interminável [30].

Tal como referido, no desenvolvimento do projeto, a Prototipagem Evolutiva foi o tipo de prototipagem que mais se adequou à metodologia adotada pela empresa sendo que, os processos de desenvolvimento de protótipo e do sistema final são essencialmente os mesmos.

Graças a esta metodologia foi possível identificar falhas, alterar funcionalidades que não iam de acordo com as necessidades do utilizador final e/ou adicionar novas funcionalidades que surgiram no desenvolvimento do *software*.

Uma vez que a API REST é um dos módulos do projeto Accept *cloud* foi necessário a integração na equipa de desenvolvimento e adequar a implementação da API REST à metodologia que a mesma utiliza.

Com base nos protótipos definidos, o chefe de equipa de desenvolvimento foi responsável pela atribuição das tarefas pelos vários elementos que constituem a equipa bem como pelas tomadas de decisão que visam garantir o sucesso da integração da aplicação *Web* com a API. A atribuição das tarefas foi feita de acordo com a prioridade das mesmas e com recurso a uma ferramenta online denominada *Trello*.

O *Trello* é uma ferramenta que integra a metodologia *Scrum*, uma metodologia *Agile* que permite controlar de forma eficiente e eficaz o projeto a realizar através da definição dos objetivos/tarefas e do cumprimento dos prazos estabelecidos de modo a que o cliente final fique satisfeito com o produto final [70]. Esta ferramenta é composta por um *board* que permite, em forma de listas, organizar tarefas de forma dinâmica e funcional, permitindo gerir e controlar o desenvolvimento do projeto de forma intuitiva, iterativa e atualizada [66].

Uma vez que as tarefas se alteram com frequência ao longo do desenvolvimento do projeto, as mesmas são facilmente adicionadas e organizadas na lista respetiva. Deste modo, é possível gerir as tarefas através do seu estado, bem como detetar se existe algum processo (tarefa com bugs ou tarefa bloqueada) que impeça o desenvolvimento de uma outra tarefa, reduzindo assim os riscos/ custos e aumentando a produtividade.

O *board* do *Trello* é composto pelas seguintes listas:

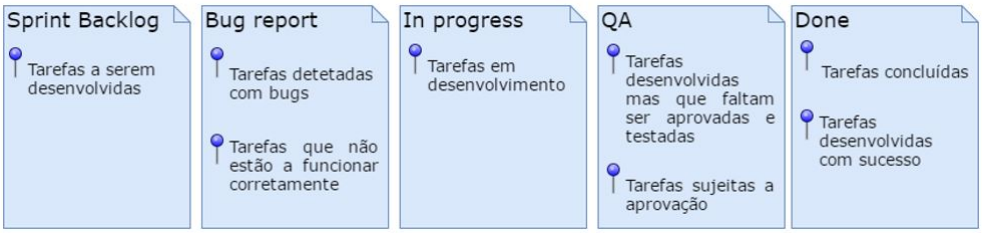

Figura 5.2: Board da ferramenta *Trello*

Com base no *board* do *Trello*, o chefe de equipa de desenvolvimento organiza reuniões regulares (previamente planeadas) onde atribui novas tarefas e/ou faz um acompanhamento do desenvolvimento das mesmas, de modo a comprovar se estão a ser desenvolvidas de forma correta, se é necessário alguma alteração ou testar se a mesma tem o resultado esperado.

Desta forma, a equipa de desenvolvimento facilmente se adapta às necessidades do cliente, otimizando o tempo de produtividade através da diminuição de reuniões desnecessárias e do sobre carregamento dos vários processos na realização de uma tarefa.

5.2 Tecnologias Utilizadas

A API REST foi desenvolvida na linguagem de programação *Python* com o apoio de *frameworks* que disponibilizam os recursos necessários à sua implementação e de uma base de dados *MySQL*.

Estas tecnologias foram decididas pela empresa, na fase inicial do estágio, por serem de código aberto, multiplataforma e por disponibilizarem os recursos necessários bem como uma forte e extensa documentação.

As tecnologias utilizadas para o desenvolvimento desta API, bem como as suas principais características, são apresentadas nesta secção.

5.2.1 *Python*

Criada por *Guido van Rossum*, em 1991, o *Python* é uma linguagem de programação de alto nível orientada a objetos, de código aberto e disponível para vários sistemas operativos (multiplataforma) [46]. Caracteriza-se pela sua sintaxe simples e clara, facilidade na sua utilização e por conter uma forte e extensa biblioteca de objetos e funções, nomeadamente na componente de matemática e estatística.

Segundo *Guido van Rossum*, o *Python* é utilizado para programação de interfaces gráficas de utilizador (GUI, do inglês *Graphical User Interface*), bases de dados, aplicações *Web*, tanto do lado do cliente como do lado do servidor, e para testar aplicações [50].

A versão do *Python* utilizada para o desenvolvimento da API REST foi a 2.7.

5.2.2 *Django*

Desenvolvido em 2003, o *Django* é um *framework* de alto nível que permite criar, de forma simples e rápida, aplicações *Web* de alto desempenho. É gratuito, de código aberto e recorre o *Python* como linguagem de programação [18].

Este *framework* tem suporte nativo a uma grande variedade de recursos úteis em aplicações *Web*, quer a nível de desenvolvimento quer a nível de administração. A nível de desenvolvimento o *Django* permite modelar dados através de classes *Python* bem como gerir e manipular tabelas de dados através do ORM, gerar automaticamente formulários através dos modelos de dados, gerar ficheiros CSV, autenticação, possui uma interface de administração de modelos criados através do ORM, autenticação, sistema de *cache*, entre outras funcionalidades. Quanto ao nível de administração, o *Django* permite a gestão de utilizadores e perfis, por exemplo.

O *Django* implementa o padrão de arquitetura MVT, padrão que segue fortemente o padrão de arquitetura MVC. A versão do *Django* utilizada foi a 1.10.1.

5.2.3 *Django* REST

O *Django REST Framework* (DRF) é um *framework* composto por um "*conjunto de ferramentas poderoso e flexível*"direcionado para a construção de forma fácil e rápida APIs REST baseadas em modelos *Django* [23].

A combinação do *Django* com o REST é uma prática bastante comum pois o *Django* aplica de forma rigorosa os princípios da arquitetura REST, permitindo o desenvolvimento de APIs intuitivas e simples.

O DRF têm suporte para a criação de recursos a partir de dados ORM e não ORM, suporte a uma variedade de métodos de serialização, utilização de filtros de conteúdos e definição do formato dos dados (JSON, XML, YAML e HTML) em que os recursos são apresentados. Permitem também a autenticação dos utilizadores, controlo de acessos através de permissões (Suporte para OAuth 1 e 2) e ainda a capacidade de teste à API [52].

O DRF permite a criação de uma interface Web navegável da APIs com documentação própria, caracterizada pelo seu grau de facilidade e simplicidade.

O DRF dispõe de uma documentação extensa, detalhada e explicativa e de um grande apoio da comunidade que permite que o suporte e o desenvolvimento utilizando este *framework* seja mantido.

Este *framework* é utilizado por várias empresas internacionalmente reconhecidas como a *Red Hat* e a *Mozilla*. A versão utilizada foi a 3.5.1.

5.2.4 *Pycharm*

O ambiente de desenvolvimento integrado (IDE) utilizado foi o *PyCharm Professional Edition*. A versão do *PyCharm Professional Edition* foi a 2016.3.2.

O *PyCharm* foi desenvolvido pela empresa *JetBrains* e é direcionado para o desenvolvimento de *software* na linguagem *Python*. Está disponível para vários sistemas operativos e caracteriza-se pela disponibilização dos seguintes recursos [45]:

- Integração com sistemas de controlo de versões;
- Integração de uma unidade de testes;
- Suporte a *frameworks* de desenvolvimento web como o *Django*, *Flask*, *Google App Engine*, *Pyramid* e *web2py*;
- Suporte de tecnologias de desenvolvimento como o *JavaScript*, *CoffeeScript*, *TypeScript*, *Cython*, *SQL*, *HTML/CSS*, *Angular JS*, *Node.js*, entre outros;
- Assistência Inteligente do *Python* (análise de código, inspeções de código, sintáxe e erros destacados, disponibilização de recursos de navegação);
- Depurador gráfico;
- Execução, teste e implementação de aplicações em computadores remotos ou máquinas virtuais com intérpretes remotos, terminal *ssh* e/ou integração *Docker* e *Vagrant*.

5.2.5 *MySQL*

O *MySQL* é um Sistema de Gestão de Base de Dados (SGBD), de código aberto, que recorre a linguagem SQL *Structured Query Language* para interagir com os dados (inserir, aceder e gerir) armazenados numa base de dados [40].

A empresa escolheu o *MySQL* devido à sua popularidade, elevada penetração no mercado/área devido ao suporte para diferentes sistemas operativos, entre eles *Oracle Linux*, *Oracle Solaris*, *Red Hat*, *Debian*, *Microsoft* e *Apple* que oferece.

Devido aos fatores acima mencionados e pelo facto do *MySQL* suportar base de dados de grandes dimensões, o *MySQL* foi utilizado com a finalidade de armazenar dados

relacionados com a empresa assim como o resultado desse processamento (tratamento estatístico).

A administração da base de dados MySQL utilizada foi feita através da ferramenta *MySQL Workbench*, versão 6.3.7.

5.3 Desenvolvimento da API

Nesta secção serão detalhados os aspetos mais relevantes relacionados com as funcionalidades da API implementadas.

5.3.1 Estrutura do Projeto

Na criação de um projeto, o *Django* cria, por defeito, um conjunto de pastas e ficheiros cuja estrutura está apresentada na Figura 5.3.

Figura 5.3: Estrutura de pastas e ficheiros.

Na pasta *apicloud* encontram-se os ficheiros de configuração:

- *settings.py* –ficheiro de configuração de instalação do *Django*;
- *urls.py* –ficheiro de configuração de *URLs*, ficheiro responsável pelo mapeamento de cada URL para a *view* correspondente;

- *wsgi.py* –ficheiro de configuração do *WSGI* (da sigla *Web Server Gateway Interface*), padrão utilizado para desenvolver aplicações *Web* em linguagem *Python*.

Na pasta *acceptgml* encontram-se os ficheiros associados à API desenvolvida que serão explicados ao longo deste capítulo.

A pasta *templates* contém os *templates* do projeto. Os *templates* são normalmente ficheiros em HTML que têm como objetivo separar a apresentação dos dados.

5.3.2 Base de Dados

Com base nos requisitos da aplicação *Web*, a equipa de desenvolvimento desenhou, através da ferramenta *MySQL Workbench*, uma base de dados que armazena todas as informações decorrentes e necessárias ao correto funcionamento da API REST.

A base de dados destina-se ao armazenamento de informação relacionada com produtos, embalagens, fornecedores, linhas de produção e dispositivos existentes nas empresas. E, além disso, armazena toda a informação obtida através da recolha de dados sobre pesagens e processos de produção de produtos de pré-embalados e todo o tratamento estatístico realizado aos mesmos.

A base de dados designa-se por "*accept_ gml*"e foi-me fornecida pelo chefe da equipa de desenvolvimento. Esta é composta por um conjunto de tabelas que, ao longo do desenvolvimento do projeto, sofreram algumas alterações/adaptações.

Na realização deste projeto, em termos de base dados, foram desenvolvidas *views* e *stored procedures* permitindo uma recolha/pesquisa de dados mais rápida e eficiente. As *views* e *stored procedures* foram desenvolvidas recorrendo à ferramenta *MySQL Workbench*.

- *Views*

  As *views* são objetos da base de dados que são definidas através de declarações *SELECT*, retornando uma visualização específica de dados de uma ou mais tabelas.

  Por exemplo, todas as tabelas têm uma *view* associada que retorna apenas os registos não eliminados da mesma, uma vez que na eliminação de registos é praticado o *SOFT DELETE*. No *SOFT DELETE*, o registo não é eliminado da

tabela, mas sim é marcado como tal. A tabela contém o campo *deleted_at* que guarda a data/hora em que o registo foi marcado como eliminado.

- *Stored Procedures*

  As *stored procedures* são funções escritas em SQL que ficam armazenadas no servidor e que retornam valores ou informações de uma ou mais tabelas. Esta técnica evita duplicar código da base de dados, aumentando a performance.

  As *stored procedures* foram implementadas para fazer pesquisas (*SELECT*) à base de dados quando este recebe filtros de pesquisa. Esses filtros são passados através de parâmetros da função que são enviados no pedido à API.

Ligação do *Django* com a base de dados *MySQL*

A plataforma *Django REST* permite a ligação à base de dados *MySQL* através da seguinte configuração no ficheiro *settings.py*:

```
1
2 DATABASES = {
3     'default': {
4         'ENGINE': 'django.db.backends.mysql',
5         'NAME': 'accept_gml',
6         'USER': 'sinmetro',
7         'PASSWORD': '*****************',
8         'HOST': 'sinmetro.no-ip.biz',
9         'PORT': '3306',
10        'TEST': {
11            'MIRROR': 'default',
12        },
13        'OPTIONS': {
14            'autocommit': True,
15        },
16    }
17 }
```

Listagem 5.1: Exemplo de configuração que permite a ligação da API à base de dados MySQL

Migração da Base de Dados

A migração da base de dados é importante pois não só mantém a aplicação atualizada como garante uma maior segurança dos dados, evitando a perda de informações iniciais

caso haja uma falha humana ou no sistema.

O sistema de migração do *Django* permite trabalhar com um grande número de migrações. Os comandos utilizados que permitiram fazer as migrações à base de dados foram:

```

python manage.py makemigrations
python manage.py migrate
```

Listagem 5.2: Comandos que permitem realizar as migrações à base de dados.

5.3.3 *Model*

O modelo defini a estrutura dos objetos da API. De acordo com a estrutura de diretorias definida pelo *Django*, o modelo encontra-se definido no ficheiro *models.py*. Este ficheiro contém toda a informação sobre os dados (campos) e o comportamento dos mesmos. Cada modelo mapeia uma única tabela/ *view* da base de dados, onde cada atributo do modelo representa um campo da tabela. Cada tabela corresponde a uma classe *Python* que estende a sub-classe *django.db.models.Model* [21].

O excerto do código apresentado na 5.3 representa a forma como uma tabela da base de dados é mapeada. Neste caso, o mapeamento é feito à tabela *Suppliers*, tabela destinada ao armazenamento de informação sobre os fornecedores.

```
class Suppliers(models.Model):
    description = models.CharField(max_length=100, blank=True, null=True)
    code = models.CharField(max_length=100, blank=True, null=True)
    ...

    def delete(self, *args, **kwargs):
        self.deleted_at = timezone.now()
        self.save()
        pass

    class Meta:
        managed = False
        db_table = 'suppliers_view'
        pass
```

Listagem 5.3: Excerto do código correspondente ao modelo da tabela Suppliers

Os campos *description* e *code* são exemplos de campos pertencentes ao modelo.

Cada campo é especificado como um atributo de classe e cada atributo é mapeado para uma coluna da tabela *Suppliers*.
O método *delete()* é um dos métodos do modelo que define o comportamento que o objeto deve ter na base de dados. Neste caso, este método é responsável por executar a instrução *SQL DELETE* do objeto. No entanto, uma vez que para este projeto é aplicado o conceito de *SOFT DELETE*, o objeto não será eliminado mas marcado como tal através do *campo delete_at* que irá armazenar na base de dados, a data/hora em que o registo foi "eliminado".

5.3.4 *Querysets*

Um *queryset* representa um conjunto de objetos de um dado modelo resultante de uma pesquisa à base de dados. O resultado da pesquisa à base de dados pode ser restringido através de filtros. Um *queryset* pode ter zero, um ou vários filtros. Os filtros são parâmetros dos métodos que devem ter o mesmo nome do campo na base de dados.
Em termos de *SQL*, um *queryset* equivale a uma instrução *SELECT* e os filtros equivalem ao *WHERE* ou *LIMIT*.

Existe um conjunto de métodos do *queryset* utilizados para obter, filtrar e ordenar objetos da base de dados. Na Tabela 5.1 são apresentados os métodos mais utilizados [20].

Tabela 5.1: Métodos *Queryset* mais utilizados

Método Queryset	Definição
.filter()	Retorna um *queryset* que combina com os parâmetros de pesquisa fornecidos.
.exclude()	Retorna um *queryset* com os objetos que não correspondem aos parâmetros de pesquisa fornecidos.
.order_by()	Retorna um *queryset* ordenado pela campo definido.
.distinct()	Retorna um *queryset* eliminando registos duplicados dos resultados da consulta.
.all()	Retorna um *queryset* com todos os objetos.
.count()	Retorna um número de resultados devolvidos no *queryset*.

Com base nos métodos *queryset* é-nos apresentado um excerto do código que mostra um exemplo de um *queryset* que retorna todos os objetos da tabela *Suppliers* que pertencem a uma determinada empresa (filtro *HTTP_COMPANYID*). O filtro é obtido através do parâmetro que é passado no *HEADER* no pedido à API.

```
1 queryset = Suppliers.objects.all().filter(company_id=request.META['
    HTTP_COMPANYID'])
```

Listagem 5.4: Exemplo de código que mostra a utilização de um método *queryset*

5.3.5 *Serializers*

Os *serializers* permitem traduzir estruturas de dados (*querysets* e instâncias de modelo) em tipos de dados *Python* para que possam ser facilmente convertidos em *JSON*, *XML* ou *YAML* [26].

Numa estrutura *REST*, os *serializers* são uma forma poderosa e genérica para controlar as respostas graças à classe *ModelSerializer*. Esta classe mapeia, de forma automática, os campos de uma classe pertencente ao modelo bem como valida se esses campos obedecem aos critérios definidos no mesmo. Por omissão, os campos que constituem a classe pertencente ao Modelo correspondem aos campos do *serializer*, no entanto, podem ser enviados apenas os campos necessários ao correto funcionamento da API [25]. O excerto do código apresentado na Listagem 5.5 mostra-nos o *serializer* do modelo correspondente à tabela *Suppliers*.

```
class SupplierSerializer(ModelSerializer):
    class Meta:
        model = Suppliers

        fields = [
            'id',
            'description',
            'code',
            ...
        ]
```

Listagem 5.5: Excerto do código correspondente ao *serializer* que retorna os dados do modelo correspondente à tabela *Suppliers*.

A classe *SupplierSerializer* define a estrutura de dados que permite "traduzir"o modelo definido para outros formatos como, por exemplo, o JSON ou o XML.
Neste projeto, a transferência dos dados entre a API e a aplicação *Web* é feita no formato JSON. Para isso, foi necessário recorrer à biblioteca *simplejson*.

simplejson - codificador e descodificador JSON

Para codificar (*encode*) e descodificar (*decode*) objetos JSON foi utilizada a *package* *simplejson*. Esta *package* é extensível para *Python* 2.5+ e *Python* 3.3+ e caracteriza-se pela sua simplicidade e rapidez. É puro código *Python*, sem dependências, embora inclua, para aumentar a velocidade, uma extensão C opcional [49].

A documentação mais recente desta *package* encontra-se disponível no sítio *Web* `https://simplejson.readthedocs.io/en/latest/` [58]:

- Codificação (*Encoding*) de objetos JSON

  Para serializar um objeto *Python* (que, neste caso, será um dicionário) para o formato JSON foi utilizado o método *simplejson.dumps*.

  Na Listagem 5.7 é apresentado é feita uma pesquisa à base de dados que obtém todos os lotes de produção de uma determinada empresa. Essa informação é armazenada num *queryset*, conjunto de pesquisa que permite ler dados da base de dados, filtrá-los e ordená-los. O *queryset* (lista de objetos do modelo *ProductionLotAllInformation*) é serializado para um dicionário que será codificado para um objeto JSON.

```
queryset = ProductionLotAllInformation.objects.all().filter(
    company_id=request.META['HTTP_COMPANYID'])

serializer = ProductionLotSerializer(queryset, many=True)

jsonData = simplejson.dumps(serializer.data)
```

Listagem 5.6: Exemplo de uma pesquisa à base de dados e da serialização da informação para um dicionário.

- Descodificação (*Decoding*) de objetos JSON

  Para deserializar um objeto JSON para um objeto *Python* (codificação UTF8, por padrão) foi utilizado o método *loads*, cujo o processo é semelhante ao processo descrito no ponto acima mas de forma inversa.

```
queryset = ProductsAllInformation.objects.all().filter(company_id=
    request.META['HTTP_COMPANYID'])

serializer = ProductSerializer(queryset, many=True)

dataAux = simplejson.dumps(serializer.data)

productsData = simplejson.loads(dataAux)
```

Listagem 5.7: Exemplo de uma pesquisa à base de dados e da deserialização da informação para um dicionário.

5.3.6 *Views*

O *Django REST* permite trabalhar com *views* baseadas em funções.

Uma função do tipo *view* (*@api_view()*)é uma função em *Python* que recebe um pedido (*request*) e retorna uma resposta (*response*). Esta função contém toda a lógica necessária para consultar os dados através do Modelo e retornar a resposta que, neste caso, encontra-se no formato JSON [24].

Por omissão, as funções do tipo *view* encontram-se definidas no ficheiro *views.py*. No entanto, devido à grande quantidade de funções implementadas, estas foram separadas de acordo com os requisitos. Por exemplo, as funções do tipo *view* relacionadas com as operações CRUD dos fornecedores encontram-se no ficheiro *suppiers_views.py*.

As funções do tipo *view* que implementam os métodos HTTP utilizadas foram: GET, POST, PUT e DELETE. A Listagem 5.8 mostra um exemplo de uma função do tipo *view* implementada, neste caso a função recorre ao método HTTP POST.

```
@api_view(['POST'])
def create_supplier(request):
    """
        Create a supplier instance.
    """
    data = request.data
  ...
  serializer = SupplierCreateSerializer(data=request.data)

    if serializer.is_valid():
        serializer.save()
        jsonData = simplejson.dumps(serializer.data)
        return HttpResponse(jsonData, content_type="application/json",
   status=status.HTTP_201_CREATED)

    return HttpResponse(simplejson.dumps(serializer.errors), content_type
   ="application/json", status=status.HTTP_400_BAD_REQUEST)
```

Listagem 5.8: Exemplo de uma função do tipo view que recorre ao método HTTP POST.

Este método é chamado quando o utilizador pretende criar um fornecedor. Para isso, o utilizador envia através do URL a informação que pretende guardar na base de dados. Essa informação encontra-se no formato JSON e é convertida para um dicionário. O dicionário é sujeito a uma validação para que os dados a armazenar

na base de dados obedeçam aos critérios definidos no Modelo. Se os dados passarem na validação, a função retorna um JSON com os dados guardados e o código status *HTTP_201_CREATED* que corresponde a criação do fornecedor com sucesso. Caso contrário, retorna o erro e o código status *HTTP_400_BAD_REQUEST*.

5.3.7 URLs

Um URL (*Uniform Resource Locator* - Localizador Padrão de Recurso) é o endereço virtual de um recurso que se encontra disponível na rede. O URL é composto pelo protocolo de comunicação HTTP e o seu formato é definido pela norma RFC 1738 [68].

Cada recurso tem um URL específico. Quando este é solicitado, o *Django* acede ao ficheiro que contém os URLs definidos e que é responsável pelo mapeamento entre os URLs e as funções de retorno (*views*). Este ficheiro designa-se por *urls.py* e encontra-se definido no módulo *ROOT_URLCONF* do ficheiro *settings.py*. O módulo *URLconf* (configuração de URL) é um conjunto de padrões que o *Django*, através de expressões regulares simples, associa a URL com a *view* correta.

O ficheiro *urls.py* é composto pela variável *urlpatterns*. Esta variável é uma lista de URLs e é percorrida pelo *Django* que, ordenadamente, é responsável por encontrar o URL requisitado e retornar a *view* associada. Para que o mapeamento entre o URL e a função de retorno seja feito corretamente, o *Django* recorre a expressões regulares que obedecem a certas normas [22]. Na Tabela 5.2 são mostradas algumas das expressões regulares mais utilizadas e o seu respetivo significado [67]:

Tabela 5.2: Expressões Regulares mais utilizadas

Expressão Regular	**Significado**
^	Início do URL.
$	Fim do URL.
+	Item anterior ocorre 1 ou mais vezes.
{n}	n ocorrências.
{n, m}	Entre as ocorrências n e m.
[]	Agrupamento de caracteres.
(?P_)	Ocorrência de captura.
\d+	1 ou mais dígitos.
\D+	1 ou mais não dígitos.
[a-zA-Z0-9_]	Um ou mais caracteres de palavras minúsculos, maiúsculos, números ou sublinhados.
\w+	Um ou mais caracteres.

É importante referir que o "r"no início de cada *string* é opcional mas recomendado.

Este indica que o conteúdo da *string* deve ter em conta os caracteres "\"em vez de os interpretar como um caracter especial.

De acordo com a Tabela 5.2, para capturar um parâmetro, a expressão regular que o *Django* utiliza é o ?P. De seguida é apresentado um exemplo de sintaxe sobre a passagem de parâmetros num URL.

```
1 url(r'^suppliers/(?P<supplier_id>\d+)$', suppliers_views.get_supplier),
```

Listagem 5.9: Exemplo de sintaxe de um URL que recebe parâmetros.

Neste exemplo, o parâmetro encontra-se declarado entre <> e tem o nome de *supplier_id*. De acordo com a expressão regular d+, o parâmetro pode ter um ou mais dígitos. Caso contrário, o URL não é chamado. Para que a captura do parâmetro funcione corretamente, o nome do parâmetro de URL deve ser o mesmo que o nome da variável de contexto do método da função de retorno (*view*).

5.4 Funcionalidades Implementadas

No sistema Accept, a análise dos dados recolhidos no controlo do processo de embalamento recorre a boas práticas de controlo estatístico de modo a garantir a credibilidade dos serviços de avaliação da conformidade metrológica junto dos embaladores e das entidades inspectoras e judiciais. As boas práticas do controlo estatístico do processo utilizadas pela empresa SINMETRO são definidas recorrendo a inúmeras referências bibliográficas, nomeadamente o livro "*Introduction to Statistical Quality Control*" escrito por *Douglas C. Montgomery* [76].

Desta forma, a implementação dos recursos que, inicialmente, constituem a aplicação Accept *cloud* dividiu-se em quatro grupos de funcionalidades distintas:

1. **Gestão de Configurações**
   A gestão de configurações permite ao utilizador, de forma rápida e eficaz, realizar as operações CRUD que permitem consultar informação da base de dados bem como adicionar, editar e/ou eliminar registos à mesma;

2. **Registo de Controlo dos processos de embalamento**
   Neste processo é feito o registo dos ensaios que determinam o conteúdo efetivo em cada embalagem, segundo um plano de amostragem. Este registo é feito através do preenchimento de formulários. Essa informação é enviada no pedido à API e

esta é responsável por armazenar essa informação nas tabelas da base de dados respetivas.

O valor das pesagens podem ser obtidos automaticamente através do dispositivo da empresa ou inseridos manualmente pelo utilizador. Este passo é fundamental para garantir a rastreabilidade;

3. **Tratamento estatístico dos dados**
   Neste processo é feito o tratamento estatístico com base na recolha de dados sobre pesagens e processos de pré-embalados;

4. **Análise dos dados**
   A análise dos dados é feita com base nos dados recolhidos no controlo do processo e respetivo procedimento.

   Este processo permite avaliar se o embalador garante, em tempo real, os direitos do consumidor e se, a sua fábrica não está a ter prejuízo no processo de embalamento.

Todos os métodos da API que permitem interagir com a base de dados seguem a estrutura REST já apresentada.

Para além da gestão de configurações, todos os cálculos estatísticos necessários para o controlo estatístico da quantidade dos pré-embalados também é feito a nível da API.

5.4.1 Gestão de Configurações

A aplicação Accept *cloud* dispõe de um conjunto de configurações que são necessárias à implementação do controlo metrológico da quantidade pré-embalada. As configurações são as seguintes:

1. **Desenvolvimento de *backoffice***
   Configuração de produtos, embalagens, fornecedores, linhas/lotes de produção e dos sistemas de medição (dispositivos) das empresas (balanças, densímetros, etc);

2. **Configuração dos processos de embalamento**;

   Após o desenvolvimento de *backoffice*, o utilizador pode definir os processos de embalamento. Um processo associa-se a um dispositivo (balança/ linha de produção) que irá efetuar as medições ao produto bem como à quantidade nominal que o produto deverá obedecer. Para cada processo são definidos os limites de especificação, o modo de aquisição de dados (manual ou automática), as regras que

podem aumentar a sensibilidade das cartas de controlo e o plano de amostragem que deve obedecer ao Princípio de Subgrupos Racionais. Segundo *Montgomery*, este princípio defende que as amostras devem ser recolhidas ao mesmo tempo de modo a maximizar a probabilidade de se detetarem causas especiais de variação entre amostras e minimizar a probabilidade de se detetarem essas mesmas diferenças dentro de uma amostra.

3. **Configuração de uma amostra;**

   Após a criação do processo, o utilizador inicia a configuração de uma amostra, identificando o produto a controlar, o lote de produção, a densidade (caso se trate de um produto líquido), o fornecedor e o tipo de embalagem (podem existir vários tipos de embalagem para um mesmo fornecedor).

Para além destas configurações, o sistema Accept *cloud* permite às empresas embaladoras configurar utilizadores e perfis.

As configurações dos dados considerados mestres (produtos, embalagens, fornecedores e linhas de produção) podem ser efetuadas através da importação em *bulk* dos dados.

Importação de ficheiros

De modo a importar dados de configuração para a base de dados de forma rápida e eficiente, foi implementada a funcionalidade de importação de ficheiros *EXCEL* e *CSV*, que é a funcionalidade que permite a importação em *bulk* dos dados.

O ficheiro a importar é enviado por *URL* juntamente com o *JSON* que contém informação sobre a versão da língua (*PT/ENG*) em que o ficheiro se encontra, o nome da tabela para onde os dados vão ser importados, um *array* com a ordem em que os campos estão distribuídos nas colunas do ficheiro (por exemplo, o campo "Nome do Fornecedor"encontra-se na coluna 1) e se a opção "ignorar duplicados"está ou não ativa (*True/ False*). Esta opção quando ativada ignora os registos duplicados, ou seja, os campos considerados únicos, se já existirem na base de dados, são ignorados e a leitura do ficheiro é passada para a linha seguinte. Se a opção estiver desativada, os campos duplicados não são ignorados, isto é, se os campos únicos da linha a inserir já existirem na base de dados, a informação restante, se for diferente da que está inserida, é substituída/atualizada na base de dados. Caso contrário, a leitura do ficheiro é passada para a linha seguinte.

Antes de ser efetuada a leitura do ficheiro é feita uma validação à extensão do mesmo. Se a extensão obedecer aos formatos suportados (.xls, .xlsx ou .csv), o ficheiro é armazenado localmente no disco para poder ser lido. Caso contrário, é retornado uma mensagem de erro "*Invalid extension file*".

Após validação da extensão do ficheiro, o ficheiro a importar é armazenado numa pasta temporária. De modo a evitar conflitos em nome de ficheiros, o formato do nome do ficheiro é *NOME_FICHEIRO_DATA_HORA*, em que o nome do ficheiro corresponde ao nome da tabela para onde os dados serão importados e a data/hora correspondente à data atual do sistema.

A Listagem 5.10 mostra como o armazenamento do ficheiro é feito no servidor.

```
def save_file(file):

    pathStr = 'tmp/{0}'.format(file)

    path = default_storage.save(pathStr, ContentFile(file.read()))

    tmp_file = os.path.join(settings.MEDIA_ROOT, path)

    return tmp_file
```

Listagem 5.10: Excerto de código que mostra como o armazenamento de um ficheiro é feito no servidor.

O método utilizado foi o *save()* pertencente à API *file storage*. Esta API contém um conjunto de métodos que permitem ao *Django* aceder aos ficheiros carregados pelo utilizador. Por norma, o armazenamento de ficheiros do *Django* é dado pela configuração *DEFAULT_FILE_STORAGE* [19].

A biblioteca que implementa um conjunto de métodos que permitem lidar com os nomes de caminho de pasta/ficheiros é a biblioteca "*os*". Neste caso o método utilizado foi o *join()* que concatena o caminho do sistema de ficheiros absoluto (*settings.MEDIA_ROOT*) com a pasta */tmp/NOME_FICHEIRO_DATA_HORA* [42].

Por fim, após o ficheiro ser armazenado localmente, é feita a leitura do mesmo. À medida que um ficheiro é lido, cada linha é armazenada numa variável do tipo dicionário tal como apresentado na Listagem 5.11.

```
newData = {
  'description': descriptionData,
  ...
```

```
}
```

Listagem 5.11: Estrutura do dicionário onde será armazenado os dados obtidos da leitura do ficheiro.

A informação indexada à chave é baseada no dicionário que contém a ordem em que os campos estão inseridos no ficheiro (dicionário com o nome *arrangeOrder* enviado no formato JSON, por URL), tal como é mostrado na Listagem 5.12.

```
def settings(arrangeOrder):

    dataJSON = simplejson.loads(arrangeOrder, encoding='latin-1')
    description = -1
  ...

    for key, value in dataJSON.items():

        if key == 'description':
            description = value
            pass
        ...

        pass

    return description, ...
```

Listagem 5.12: Exemplo de leitura da informação obtida de um ficheiro para um dicionário.

Após a leitura do ficheiro, a informação contida no dicionário é validada pelo *serializer* associado à tabela a que se destina a importação dos dados. Se os dados a armazenar obedecerem aos critérios definidos no Modelo, é retornado uma mensagem a informar que a importação foi feita com sucesso, bem como o número de registos inseridos. Caso contrário, é enviado o número de erros encontrados e os detalhes do mesmo (linha em que deu o erro, o campo em que deu o erro e a respetiva mensagem de erro).

Conforme a extensão dos ficheiros, a sua leitura é feita recorrendo a bibliotecas diferentes apesar de ambas apresentar a mesma lógica.

- Leitura de um ficheiro CSV

  O formato de importação/exportação de dados para base de dados ou folhas de cálculo mais comum é o *Comma-Separated Values*(CSV). Um ficheiro CSV é um

ficheiro sem formatação, onde os dados estão separados por vírgulas e em que cada linha corresponde a um registo diferente.

Para fazer a importação dos dados para a base de dados foi necessário recorrer à biblioteca csv. Esta biblioteca implementa um conjunto de classes que permitem ler e escrever dados neste tipo de ficheiros.

A Listagem 5.13 mostra como o processo de leitura do ficheiro CSV é feito. Inicialmente é necessário abrir o ficheiro CSV do qual queremos extrair a informação. Depois, é necessário criar um objeto do tipo *reader* (leitor) que irá ler cada linha que constitui o ficheiro e guardar a informação lida num dicionário para posteriormente ser armazenada na base de dados. Quando a leitura do ficheiro estiver concluída, o ficheiro é fechado.

```
with open(pathFile, 'r') as file:
        reader = csv.reader(file, delimiter=str(delimiter))

        next(reader)
    ...
        for row in reader:

            codeData =unicode(str(row[code]), encoding='latin-1',
    errors='ignore')
            descriptionData  = unicode(str(row[description]),
    encoding='latin-1', errors='ignore')

            newData = {
                'code': codeData,
                'description': descriptionData
            }

            dictData.append(newData)
            pass
        pass
```

Listagem 5.13: Processo de leitura de um ficheiro CSV.

Neste tipo de ficheiros foi necessário fazer o *encode* a cada linha lida para que a informação guardada na base de dados suporta-se caracteres especiais.

- Leitura de um ficheiro EXCEL

  Para extrair informação de ficheiros EXCEL (formato .xls ou .xlsx) foi necessário instalar a biblioteca xlrd. Esta biblioteca contém um conjunto de métodos que permitem manipular os ficheiros EXCEL.

  A Listagem 5.14 mostra como se faz a leitura ao ficheiro EXCEL.

```
file = xlrd.open_workbook(pathFile)
 ...
   for sheet in file.sheets():
       for row in range(1, sheet.nrows):
      ...
            for column in range(sheet.ncols):

                codeData = sheet.cell(row, code).value
                descriptionData = sheet.cell(row, description).value

                pass


            newData = {
                'code': codeData,
                'description': descriptionData,
            }

            dictData.append(newData)

            pass
        pass

   return dictData
```

Listagem 5.14: Processo de leitura de um ficheiro EXCEL.

Com base no código, verificamos que inicialmente é necessário abrir o ficheiro através do método *open_workbook* existente na biblioteca *xlrd*. De seguida, cada linha existente nas folhas de cálculos do ficheiro é percorrida e, consequentemente, é extraída a informação que será guardada num dicionário. Por fim, após a leitura do ficheiro estar concluída, toda a informação nele contida será armazenada na base de dados.

5.4.2 Tratamento estatístico dos dados

Para a realização do tratamento estatístico de dados foi necessário recorrer a um conjunto de bibliotecas matemáticas disponíveis no *Python*. O *Python* é uma linguagem de programação bastante popular para a análise estatística de dados porque já contém um vasto conjunto de bibliotecas que permitem realizar cálculos matemáticos, tais como:

- a biblioteca *math* que fornece um conjunto de funções matemáticas definidas pelo

padrão C, como por exemplo, a função *sqrt(x)* que retorna a raíz quadrada de um valor [3].

- a biblioteca *statistics* que contém um conjunto de funções de estatísticas matemáticas, como por exemplo a função *mean(listValues)* que permite calcular a média aritmética de uma amostra de dados [4] da qual se pode observar na listagem 5.15.

```
"""
Cálculo da média
:param lista: lista com os valores das pesagens/amostras
:return media: retorna a média
"""
def calculateAverage(listValues):

    if len(listValues) == 0:
        average = 0
    else:
        average = mean(listValues)

    return round(average, DECIMAL_PLACES)
```

Listagem 5.15: Exemplo de código que mostra como é efetuado o cálculo da média.

- a biblioteca *numpy* trata-se de um biblioteca que suporta operações com vetores e matrizes sendo fundamental para a computação científica com o *Python*. Tem uma sintaxe semelhante ao *Matlab*, mas como é implementada em linguagem C apresenta uma maior velocidade nas operações efetuadas [38].

  Como exemplo, temos o cálculo do desvio-padrão [54]. Para o tratamento estatístico (estudo de capacidade) efetuado, foram realizados dois tipos de desvio padrão:

  - o desvio-padrão simples que foi calculado através da função *numpy.std(listValues)* em que *listValues* representa a amostra de valores. A Listagem 5.16 mostra um exemplo de como este desvio foi implementado;

```
"""
Cálculo do desvio padrão
:param lista: lista com os valores das pesagens
:return desvio-padrão: retorna o desvio-padrão corrigido
"""
def calculateStdSimples(listValues):

    std = numpy.std(listValues)

```

```
10      return round(std, DECIMAL_PLACES)
```

Listagem 5.16: Exemplo de código que mostra como é calculado o desvio-padrão simples.

– o desvio-padrão corrigido foi calculado através da mesma função, no entanto recebe o parâmetro *ddof* (*Delta Degrees Of Freedom*). Este parâmetro representa os graus de liberdade. Por defeito, o *ddof* é 0. A Listagem 5.17 mostra como este desvio-padrão foi calculado.

```
1  """
2  Cálculo do desvio padrão corrigido
3  :param lista: lista com os valores das pesagens
4  :return desvio-padrão: retorna o desvio-padrão corrigido
5  """
6  def calculateStd(listValues):
7
8      if len(listValues) > 1:
9          std = numpy.std(listValues, ddof=1)
10     else :
11         std = numpy.std(listValues, ddof=0)
12
13     return round(std, DECIMAL_PLACES)
```

Listagem 5.17: Exemplo de código que mostra como é efetuado o cálculo do desvio-padrão corrigido.

Esta biblioteca também foi utilizada para o cálculo do histograma através da função *numpy.histogram*. Esta função retorna os dados que permitem desenhar o histograma. A Listagem 5.18 mostra como esses dados foram calculados.

```
1  '''
2  Calcular histograma
3  :param lista - lista de valores
4  :return xMin - valor min (eixo do x) - valor onde começa o
       histograma
5  :return sizeBucket - tamanho da barra
6  :return bucketsList - lista de barras
7  '''
8  def calculateHistogram(listValues):
9      n = len(sorted(listValues))
10     k = math.sqrt(n)
11     ki = int(round(k))  # número de classes
12
13     a, binEdges = numpy.histogram(listValues, bins=ki)
14
15     xMin = calculateMinimum(binEdges)
```

```
16
17      sizeBucket = binEdges[1] - binEdges[0]
18
19      bucketsList = a
20      return xMin, sizeBucket, bucketsList
```

Listagem 5.18: Exemplo de código que mostra como são calculadas as variáveis que permitem desenhar um histograma.

- a biblioteca *scipy* que contém um conjunto de funções destinadas à engenharia, ciência e matemática [55]. Esta biblioteca foi utilizada para obter a variável aleatória contínua de T de *Student* através da função *scipy.stats.t* [56]. Esta variável foi utilizada para o cálculo do fator *K*. O fator K é um valor que é calculado de acordo com a distribuição normal ou *t-Student* e indica, com base no nível de confiança requerido para um intervalo, a incerteza de um resultado de medição.

Para além destas bibliotecas, foram utilizadas funções disponíveis na biblioteca padrão do *Python* como, por exemplo, o cálculo do máximo através da função *max(listValues)*, função que retorna o valor máximo de uma amostra de dados [47].

Cálculos estatísticos

Para o estudo de capacidade de um processo de embalamento é necessário efetuar um conjunto de cálculos estatísticos. Nesta sub-seccção são mostrados exemplos de cálculos estatísticos efetuados.

Os cálculos estatísticos implementados foram agrupados nos seguintes grupos:

- Cálculos Básicos de Estatística Descritiva como o cálculo da média, máximo, amplitude, amplitude média, amplitude móvel, desvio-padrão e desvio-padrão médio de uma lista de valores;
- Cálculo de indicadores de avaliação da capacidade de um processo (*Cp*, *Cpk*);

  O *Cp* e o *Cpk* são índices de capacidade de um processo. O *Cp* compara a variabilidade do processo com a permitida pelas especificações, mas não informa sobre a centralidade do processo com o valor médio. O método que permite calcular o CP é apresentado na Listagem 5.19.

```
"""
CP - Capacidade do Processo (Taxa de tolerância (a largura dos
    limites de especificação) à variação atual (tolerância do
 processo)
(usl - lsl) / (6 * sigma)

:param usl: limite superior de especificação
:param lsl: limite inferior de especificação
:param sigma: sigma
:return cp: retorna a cp
"""
def calculateCp(usl, lsl, sigma):

    try:
        cp = (usl - lsl) / (6.0 * sigma)
    except ZeroDivisionError:
        cp = 0

    return cp
```

Listagem 5.19: Exemplo de código que mostra como é efetuado o cálculo do CP (capacidade do processo).

O *Cpk* compara a variabilidade do processo com a permitida pelas especificações, considerando a média do processo relativa ao ponto médio das especificações. É aconselhável o cálculo dos índices *Cpk* superior e *Cpk* inferior. Estes índices comparam o posicionamento da média face aos limites de especificação superior e inferior. A Listagem 5.20 é um exemplo de código que mostra como o CPK superior foi calculado.

```
"""
CPK Sup- Capacidade do Processo (Taxa de tolerância (a largura dos
    limites de especificação) à variação atual,
considerando a média do processo relativa ao ponto médio das
    especificações.)

:param average: média
:param usl: limite superior de especificação
:param sigma: sigma
:return cpkSup: retorna a cpk superior
"""
def calculateCpkSup(average, usl, sigma):
    try:
        cpkSup = (usl - average) / (3.0 * sigma)
    except:
        cpkSup = 0

```

```
16      return cpkSup
```

Listagem 5.20: Exemplo de código que mostra como é efetuado o cálculo do CPK superior do processo).

O objetivo é maximizar os índices *Cp* e *Cpk* e minimizar a percentagem de unidades defeituosas através da optimização dos processos.

- Implementação em *Python* da Tabela de Constantes para a Cartas de Controlo (tabela SPC). Esta tabela encontra-se disponível no sítio *Web* `http://www.bessegato.com.br/UFJF/resources/table_of_control_chart_constants_old.pdf` [9].

- Cálculo do nível *sigma*;

  O *sigma* é o desvio padrão do processo. O *sigma* foi calculado com base num pseudocódigo apresentado pelo chefe de equipa, cujo o método implementado encontra-se representado na Listagem 5.21.

```
1  """
2  Cálculo do sigma
3  :param rmedia: amplitude média
4  :param n: tamanho da amostra
5  :return sigma: retorna o sigma
6  """
7  def calculateSigma(rmedia, n):
8
9      d2p = 0
10     if n == 1:
11         d2p = 1.128
12         pass
13     else:
14         d2p =  table_spc.TableSPC().getConstantTableSPC("d2pequeno",
       n)
15         pass
16
17     sigma = rmedia / d2p
18
19     return sigma
```

Listagem 5.21: Exemplo de código que mostra como é efetuado o cálculo do *sigma*.

- Cálculo dos limites de especificação, limites de controlo e limites de rejeição.

  Os limites de especificação são os valores entre as quais os produtos pré-embalados devem operar [62].

Os limites de controle representam sua variação do processo e ajudam a indicar quando seu processo está fora de controle. Os limites de controle superior e inferior são baseados na variação aleatória no processo [63].

Os limites de rejeição definem faixas nas quais não há dúvidas de que o produto não obedece à tolerância [10].

Estes limites são calculados na configuração do processo. A largura dos limites pode apresentar três níveis de sigma diferentes:

- Nível 1 - largura apertada onde o nível do sigma é 1;
- Nível 2 - largura normal onde o nível do sigma é 2;
- Nível 3 - largura larga onde o nível do sigma é 3.

O sigma de rejeição tem o valor 20 por padrão e o ki (percentagem de tolerância) tem o valor 30.

- Cálculo do erro admissível por defeito, 1º limite legal (TL1) e 2º limite legal (TL2).

  O cálculo do EAD é baseado no Quadro n.º 1 presente na portaria 1198/91 de 18 dezembro.
  O cálculo dos limites legais (TL1 e TL2) é apresentado na Listagem 5.22.

```
"""
Cálculo do 1º Limite Legal (TL1)
:param nominal - quantidade nominal (em gramas ou mililitros)
:param tl1 - 1º limite legal
:return tl1 - retorna o 1º limite legal
"""
def calculateTL1(nominal, tl1, ead):
    if tl1 != None:
        return round(tl1, DECIMAL_PLACES)
    else:
        tl1 = nominal - ead
        return round(tl1, DECIMAL_PLACES)

    pass

"""
Cálculo do 2º Limite Legal (TL2)
:param nominal - quantidade nominal (em gramas ou mililitros)
:param tl2 - 2º limite legal
:return tl2 - retorna o 2º limite legal
"""
def calculateTL2(nominal, tl2, ead):
```

```
23      if tl2 != None:
24          return round(tl2, DECIMAL_PLACES)
25      else:
26          tl2 = nominal - (2 * ead)
27          return round(tl2, DECIMAL_PLACES)
28      pass
```

Listagem 5.22: Exemplo de código que mostra como são calculados os limites legais (1º e 2º).

- Cálculo da percentagem de unidades defeituosas;

  O cálculo da percentagem de unidades defeituosas é dada pela seguinte expressão matemática: % defeituosos = P(x>= USL) + P(x<= LSL).

  Para o cálculo da percentagem foi necessário implementar a função *normcdf*. Esta função retorna a distribuição normal cumulativa e encontra-se definida na Listagem 5.23

```
1  """
2  Cálculo da distribuição normal cumulativa (normcdf)
3
4  :param le: limite de especificação (superior ou inferior)
5  :param media: média
6  :param sigma: sigma
7  :return normcdf: retorna a distribuição normal cumulativa
8  """
9  def normcdf(le, media, sigma):
10
11      t = le - media
12      normcdf = 0.5 * (math.erfc(-t / (sigma * math.sqrt(2.0))))
13
14      if normcdf > 1.0:
15          normcdf = 1.0
16
17      return normcdf
```

Listagem 5.23: Exemplo de código que mostra como é calculada a distribuição normal cumulativa.

Após o cálculo da função *normcdf*, é possível determinar a percentagem de defeituosos. A Listagem 5.24 mostra como é calculado a P(x>=USL).

```
1  """
2  Cálculo da percentagem de valores das amostras superiores ao limite
       superior de especificação
3
4  :param usl: limite superior de especificação
5  :param average: média
```

```
:param sigma: sigma
:return p_usl: retorna a percentagem dos valores das amostras
    superiores ao lse
"""
def calculatePerUSL(usl, average, sigma):
    p_usl = 0
    valorAux = 0

    if sigma > 0:
        valorAux = 1 - normcdf(usl, average, sigma)
        p_usl = round(valorAux * 100, 2)
        pass
    else:
        p_usl = 0
        pass

    return p_usl
```

Listagem 5.24: Exemplo de código que mostra como é calculada a percentagem de valores das amostras superiores ao LSE.

- Cálculo da probabilidade de produção de unidades sem defeito;

  Este cálculo designa-se por *First Pass Yield* ou rendimento/ capacidade de um processo para produzir com zero defeitos.

  O método implementado para este cálculo esta apresentado na Listagem 5.25.

```
"""
Cálculo da probabilidade de produção de unidades sem defeito

:param x: percentagem de defeitos detetados
:param n: número de unidades inspecionadas
:return prob: retorna a probabilidade de produção de unidades sem
    defeito.
"""
def calculateProbability(x, n):

    p = x / n
    result = 100 * prob
    return result
```

Listagem 5.25: Exemplo de código que mostra como é calculado a probabilidade de produção de unidades sem defeito.

- Validação dos critérios legais dos conteúdos e da média previstos na portaria 1198/91 de 18 dezembro;

  Os critérios legais implementados foram:

- critério legal da média;
- critério legal do TL1;
- critério legal do número de amostras inferiores ao TL2;
- critério legal KS.

Para calcular o critério legal KS é necessário calcular o fator K. O fator K é calculado com base no código apresentado na Listagem 5.26.

```
'''
Obter Fator K
:param n - número total de pesagens
:return k - k
'''
def calculateK(n):
    k = 0

    if n > 1:
        t = scipy.stats.t
        normInv = t.ppf(0.005, n - 1)
        k = abs(normInv / math.sqrt(n))
        pass

    return k
```

Listagem 5.26: Exemplo de código que mostra como é calculado o fator K.

O fator K foi calculado com base no pseudo-código apresentado pelo chefe de equipa.

Com base nos cálculos dos critérios legais implementados é obtido um resultado do lote. Esse resultado indica se o lote está conforme os limites legais.

- Implementação das Cartas de Controlo de *Shewhart*

  As cartas de controlo de *Shewhart* da média com a indicação da dispersão dos valores individuais, média e amplitude ou desvio-padrão, ou valores individuais e amplitudes móveis.

  Os métodos implementados não devolvem o gráfico mas os dados que o permitem desenhar.

  A Figura 5.4 mostra uma carta de controlo desenhada a partir do excerto do JSON apresentado na Listagem 5.27. O JSON contém os valores que são necessários para o desenho da carta de controlo.

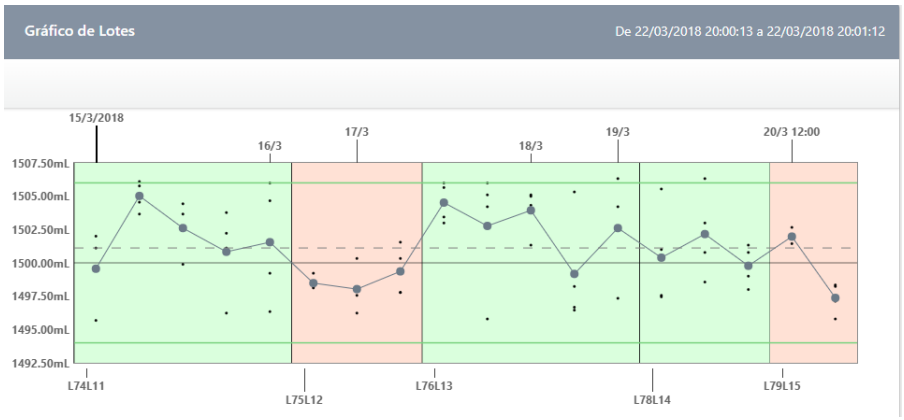

Figura 5.4: Exemplo de uma carta de controlo gerada pelo JSON devolvido pelo método da API.


```
"samples_data": [
{

  "process_id": 393,

  "stats_data": [
    {
      "usl": "1506.0000",
      "url": "1560.0000",
      "ucl": "1506.0000",
      "sample_id": 7027,
      "lrl": "1440.0000",
      "lsl": "1494.0000",
      "ccl": "1500.0000",
      "lcl": "1494.0000",
      "id": 7056,
      ...
    }

  ],

  "values_data": [
    {
      "gross_value": "1366.8000",
      "sample_id": 7027,
      "order": 1,
      "net_value": "1354.8000",
      "company_id": 274,
      "value_result": 1,
      "final_net_value": "1501.9956",
      "id": 30266,
```

```
        ...
    },
    ...
  ]
}
```

Listagem 5.27: Excerto do JSON que contém os dados necessários para o desenho da carta de controlo

Para cada carta de controlo é calculado a linha central e os limites de controlo superior e inferior.

- Implementação de regras SPC

  As regras de SPC são aplicadas às Cartas de Controlo e permitem aumentar a sensibilidade das mesmas. As regras de SPC implementadas devolvem:

  - os Pontos fora dos Limites de Controlo;
  - os N pontos consecutivos intercalados;
  - os N pontos consecutivos a descer ou a subir;
  - os N pontos consecutivos abaixo ou acima dos limites de controlo.

- Cálculo dos custos de sobre enchimento.

  A Listagem 5.28 mostra o método implementado para calcular o custo de sobre enchimento por unidade (I) ou por gramas/mililitros.

```
'''
Calcular o custo de sobre enchimento
'''
def calculateOverfillingCost(cost_per_product, unit_type_per_cost,
    lot_dimension, nominal, average):

    if average > nominal:
        overfilling = average - nominal
        pass
    else:
        overfilling = 0
        pass

    waste = overfilling * lot_dimension

    if unit_type_per_cost == "I":
        overfilling_cost = (cost_per_product * waste) / nominal
        pass

    elif unit_type_per_cost == "U":
```

```
20          overfilling_cost = (cost_per_product * waste) / 1000.0
21          pass
22
23      else:
24          overfilling_cost = 0.0
25          pass
26
27      return round(overfilling_cost, DECIMAL_PLACES)
```

Listagem 5.28: Exemplo de código que mostra como é calculado o custo de sobre enchimento por unidade ou por gramas.

A fórmula para o custo de sobre enchimento foi dada por um técnico da empresa *Aferymed*.

Análise de Dados

A análise dos dados pode ser feita não só através do tratamento estatístico acima descrito mas também através da emissão de relatórios automáticos.

- Relatórios

  A aplicação Accept *cloud* permite fazer o *download* de relatórios de avaliação da conformidade metrológica dos processos/lotes de produção analisados para que o embalador verifique o funcionamento das suas linhas de produção e se os seus produtos se enquadram nos limites legais. Estes são constituídos por informação estatística distribuída por tabelas e, em alguns casos, incluem gráficos para complementar a informação.
  Os relatórios emitidos pelo sistema Accept *cloud* são gerados pela API e encontram-se no formato PDF (*Portable Document Format* - Formato Portátil de Documento).

  Para gerar os ficheiros PDF foi necessário instalar a biblioteca *pdfkit* que converte código HTML para PDF [48]. Desta forma, os relatórios foram desenvolvidos recorrendo a código HTML para o conteúdo e CSS para a definição visual dos mesmos.
  De seguida é apresentada a Listagem 5.29 que mostra um código exemplo de como um ficheiro HTML denominado "*template_lot_report_pt.html*" é convertido para um ficheiro PDF. O ficheiro HTML é preenchido dinamicamente com a informação que se encontra dentro da variável *context*. Por fim, o ficheiro HTML é convertido para PDF e retornado para a aplicação.

```
@api_view(['POST'])
def lot_report(request, production_lot_id):
    template = get_template("template_lot_report_pt.html")
    ...
    context = Context({
        'lot_description': lot_description,
        'product_name': product_name,
        'dataSamples': dataSamples,
    })

    html = template.render(context)

    pdfkit.from_string(html, 'out.pdf', options=OPTIONS)

    pdf = open("out.pdf")

    response = HttpResponse(pdf.read(), content_type='application/
    pdf')

    response['Content-Disposition'] = 'attachment; filename=output.
    pdf'

    pdf.close()

    return response
```

Listagem 5.29: Exemplo de código que mostra a conversão de um ficheiro HTML para um ficheiro PDF.

Além da conversão do ficheiro HTML para PDF, através da biblioteca *pdfkit* é possível configurar a página do ficheiro PDF (tamanho da página, margens da página e o rodapé com a numeração da página).

A Listagem 5.30 representa a configuração da página de um ficheiro PDF gerado.

```
OPTIONS = {

    'page-size': 'A4',
    'margin-top': '1.5cm',
    'margin-right': '2cm',
    'margin-bottom': '1.5cm',
    'margin-left': '2cm',
    'encoding': "UTF-8",
    'no-outline': None,
    'footer-left': "Relatório gerado por Acceptcloud © SINMETRO, LDA
    ",
    'footer-right': 'Página [page] de [topage]',
    'footer-font-size': '8'
```

```
13
14 }
```

Listagem 5.30: Configuração da página de um ficheiro PDF gerado.

O ficheiro HTML é responsável por processar a informação recebida, neste caso através da variável *context*. A Listagem 5.31 apresenta um exemplo de como as variáveis definidas em *Python* são acedidas em código HTML.

```
<div id="process_data">
    <div>Produto: {{ product_name }}</div>
    <div>Quantidade Nominal: {{ product_nominal }}</div>
</div>

(...)

{% for element in dataSamples %}
    <h3>Data: {{ element.sample_date | date:"d-m-o" }} {{
                 element.sample_date|time:"H:i:s"    }} </h3>
{% endfor %}
```

Listagem 5.31: Exemplo de código que mostra como as variáveis em Python são acedidas em código HTML.

Os gráficos incluídos num relatório são gerados e guardados, temporariamente, no formato PNG, sempre que é gerado um relatório. A biblioteca responsável pela renderização dos gráficos foi a *matplotlib.pyplot*. Esta biblioteca é composta por um conjunto de funções que fazem o *matplotib* funcionar como o MATLAB. Cada função *pyplot* permite criar gráficos ou diagramas através da ligação de pontos que representam os valores definidos. A biblioteca pyplot cria uma figura e é nela que o gráfico é desenhado [37].

Os relatórios gerados pela API podem conter dois tipos de gráficos:

1. **Gráficos (ou Cartas) de Controlo**

   Com base no tratamento estatístico efetuado é possível produzir cartas de controlo de *Shewhart*. Estas cartas permitem fazer um acompanhamento dos processos de embalamento. Neste caso é possível produzir cartas de controlo da média com a indicação da dispersão dos valores individuais, média e amplitude ou desvio-padrão, ou valores individuais e amplitudes móveis.

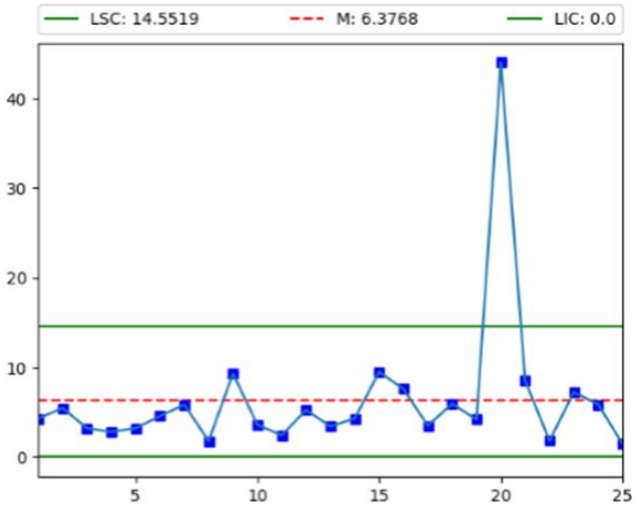

Figura 5.5: Exemplo de um Gráfico (ou Carta) de Controlo.

A Figura 5.5 representa o tipo de gráfico utilizado para acompanhar um processo de embalamento, com o objetivo de verificar, se o processo está controlado (dentro dos conformes legais). Os limites de controlo superior e inferior e a linha média foram previamente estatisticamente determinados ou definidas pelo utilizador na configuração do processo.

Para o desenho deste gráfico foi necessário recorrer às seguintes funções do *matplotlib.pyplot*:

- *plot* - Esta função permite traçar linhas com um estilo de linha/marcador definido. A Listagem 5.31 que mostra um exemplo de código que permite desenhar dois gráficos: um gráfico com o estilo "linha"e um gráfico que marca apenas os pontos que representam os valores com o estilo "*square*".

```
1
2 axisX = range(1, len(listStd) + 1)
3 axisY = listStd
4
5 plt.plot(axisX, axisY, 'bs')
6 plt.plot(axisX, axisY)
```

Listagem 5.32: Exemplo de código que mostra como se desenha um gráfico com o estilo "linha"e um gráfico que marca apenas os pontos com o estilo "*square*"

- *axhline* - Esta função permite traçar uma linha horizontal que, neste caso, representa um limite de controlo. A Listagem 5.33 apresenta um exemplo de código implementado para desenhar um limite de controlo (linha horizontal).

```
plt.axhline(y=dataControlChartS['lics'], label=label, color=
    'green', linestyle='-')
```

Listagem 5.33: Exemplo de código que mostra como se desenha uma linha horizontal

2. **Histogramas**

   Os histogramas permitem uma análise exploratória de dados e a validação da normalidade dos dados, condição necessária para o estudo da capacidade do processo.

   A Figura 5.6 representa um histograma, um tipo de gráfico em barras ou colunas que representa um conjunto de dados dividido em classes. O histograma é calculado com base na informação armazenada na base de dados. Os limites de especificação superior e inferior e a média foram estatisticamente determinados ou definidas pelo utilizador na configuração do processo.

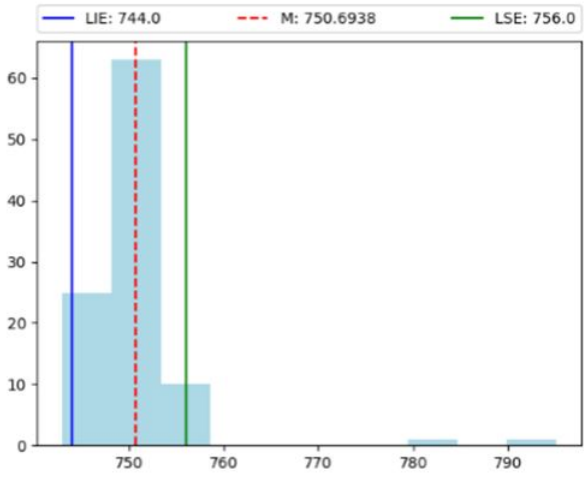

Figura 5.6: Exemplo de um Histograma.

Para o desenho deste gráfico foi necessário recorrer às seguintes funções:

– função *histogram* da biblioteca *NumPy*

Esta biblioteca é utilizada na computação científica e encontra-se disponível em `http://www.numpy.org/`. A função *histogram* determina os valores do histograma e o número de classes (representada por barras). O código implementado encontra-se na Listagem 5.34.

```
n = len(sorted(listValues))
k = math.sqrt(n)
ki = int(round(k))
a, binEdges = np.histogram(listValues, bins=ki)
```

Listagem 5.34: Exemplo de código que calcula os valores necessários que permitem desenhar o histograma.

3. função *hist* da biblioteca *matplotlib.pyplot* - Esta função é responsável pelo cálculo e desenho do histograma. A Listagem 5.35 mostra como se gere os histogramas.

```
plt.hist(listValues, bins=binEdges, color="#ADD8E6")
```

Listagem 5.35: Exemplo de código que permite gerar histogramas.

4. função *axvline* da biblioteca *matplotlib.pyplot* - Esta função permite traçar uma linha vertical que, neste caso, representa um limite de especificação. A Listagem 5.36 mostra como se desenha uma linha vertical, de cor verde, ao gráfico.

```
plt.axvline(x=lse, label=label, color='green', linestyle='-')
```

Listagem 5.36: Exemplo de código que permite desenhar uma linha vertical ao histograma.

Todos os gráficos são constituídos por uma legenda que se encontra colocada na parte superior do gráfico. Para isso foi necessário implementar o código apresentado na Listagem 5.37.

```
plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=4,
           ncol=3, mode="expand", borderaxespad=0.)
```

Listagem 5.37: Exemplo de código que permite definir legenda ao gráfico.

Quando o desenho do gráfico estiver terminado, é necessário guardá-lo numa figura e por fim, fechá-lo. A figura é guardada no formato PNG. De seguida, podemos visualizar na Listagem 5.38 um exemplo do código implementado neste processo.

```
ROOT_DIR = os.path.dirname(os.path.abspath(__file__))
IMAGES_DIR = '/static/images/'

path = "{0}{1}histogram.png".format(ROOT_DIR, IMAGES_DIR)
plt.savefig(path)
plt.close()
```

Listagem 5.38: Exemplo de código que mostra como se guarda um gráfico numa figura.

No Anexo 7 é apresentado um exemplo de um relatório gerado pela API. Este relatório corresponde ao Estudo da Capacidade de um Processo.

Overview - Vista geral da empresa em tempo real

A aplicação Accept *cloud* possui uma vista onde é possível ter um *overview* geral da empresa, consoante o perfil de utilizador. O *overview* tem como principal objetivo, responder a questões do tipo: "O que eu produzi?", "As minhas linhas de produção estão a trabalhar corretamente?"e/ou "Qual o meu desperdício + risco legal?". Desta forma, o utilizador pode, em tempo real, visualizar o comportamento dos processos de enchimento através de *widgets*.

As *widgets* mostram uma análise à informação sobre o tratamento estatístico efetuado sobre processos, linhas ou lotes de produção. O utilizador configura as *widgets* (tamanho, posição e conteúdo) que pretende ter no *overview*. Toda a informação visualizada em cada *widget* é carregada por um método específico da API. Esse método obtém a informação necessário recorrendo a uma *stored procedure* implementada na base de dados específica para o efeito.

A *stored procedure* é responsável pela pesquisa à base de dados com base nos filtros enviados no pedido à API. A chamada de uma *stored procedure* é feita pela estrutura apresentada na Listagem 5.39.

```
1 cursor = connection.cursor()
2
3 cursor.callproc("nome_da_stored_procedure", [parametro1, parametro2,
      parametroN])
4
5 rows = cursor.fetchall()
6
7 keys = map(lambda x: x[0], cursor.description)
8
9 cursor.close()
```

Listagem 5.39: Exemplo de código que mostra como é feita a chamada de uma *stored procedure*.

Inicialmente, as *widgets* que se encontram implementadas e que utilizam os dados fornecidos pela API são:

1. **Processos recentes**

   A *widget* "Processos Recentes" mostra os últimos N processos mais recentes. A pesquisa é filtrada por intervalo de tempo e linha de produção.

2. **Processos favoritos**

A *widget* "Processos favoritos" mostra informação sobre os processos favoritos do utilizador.

3. **Processo Simples**

   A *widget* "Processo" mostra informação sobre um processo específico.
   A pesquisa é filtrada pelo ID do processo e intervalo de tempo ou número das últimas pesagens.

4. **Processos Top**

   Nesta *widget* é possível visualizar os N processos de uma linha de produção agrupados por CPK, desvio médio, percentagem do desvio médio ou por percentagem de conformes. O valor do parâmetro escolhido é ordenado por ordem crescente ou decrescente.
   A pesquisa é filtrada pelas últimas N pesagens ou por filtro temporal.

5. **Lotes Lista**

   Esta *widget* tem como objetivo mostrar uma lista com todos o lotes produzidos para os filtros recebidos. Os filtros podem ser: linha de produção ou processo e filtro temporal ou os últimos n registos.
   A lista devolvida contém informação sobre a linha de produção, processo, produto e lote de produção, data da primeira e última pesagem, número de pesagens, desvio, operador, sobre-custo e respetivo resultado que indica se o lote está com lucro ou prejuízo e, por fim, o resultado final do lote que indica se este se encontra ou não nos conformes.

6. **Lotes CL**

   A *widget* "Lotes CL" permite ao utilizador consultar os lotes com problemas legais bem como o top de linhas/processos ou produtos com problemas.
   A pesquisa é filtrada por processo ou linha de produção e por filtro temporal.
   A informação devolvida é: o número total de lotes, número de lotes que não se encontram nos conformes e o desvio médio dos mesmos.
   Devolve também uma lista com a linha de produção, número de lotes nela produzida, número de lotes que não se encontram nos conformes e o desvio médio.
   A lista é agrupada por linha, processo ou produto.

7. **Lotes Perdas €**

   A *widget* "Lotes Perdas €" permite ao utilizador consultar as perdas monetárias de uma linha de produção ou processo, a variação relativamente ao último período em valor absoluto e em valor percentual (por exemplo, se o período for "semana atual", mostra a variação relativamente à semana anterior),

Devolve também o top de linhas, processos ou produtos com sobre-custo. Por exemplo, linha de produção / número de produtos / sobre-custo.
A pesquisa é filtrada por processo ou linha de produção e por filtro temporal.

8. **Lotes Perdas € + CL**

   A *widget* "Lotes Perdas € + CL"permite ao utilizador visualizar as perdas monetárias de uma linha de produção ou processo bem como os lotes com problemas legais.
   A pesquisa é filtrada por processo ou linha de produção e por filtro temporal.
   A informação devolvida é: o número de lotes produzidos, número de lotes com problemas legais, desvio médio e o custo total das perdas bem como a variação relativamente ao último período em valor absoluto e percentual (por exemplo, se o período for semana atual, mostra a variação relativamente à semana anterior).

A informação devolvida nas *widgets* 1, 2 e 3 é: nome do processo, nominal, linha de produção, média, CPK, número de pesagens e os dados necessários que permitem desenhar o histograma e a carta de controlo das médias.

O filtro temporal definido na pesquisa à base de dados pode ser: hoje, hoje e ontem, esta semana, última semana, este mês, último mês, este ano e último ano.

Capítulo 6

Testes

Neste capítulo pretende-se apresentar a forma como os métodos da API foram testados e consequentemente validados. As soluções apresentadas nos métodos desenvolvidos foram acompanhadas pela realização de testes unitários.

Os testes unitários têm como principal objetivo testar possíveis cenários através do comportamento dos métodos da API. O comportamento é testado através da validação dos dados de resposta e do código *status* que retorna. Essa validação recorre essencialmente a asserções entre valores devolvidos pelo bloco em teste e valores previamente esperados.

Neste caso, os testes unitários foram realizados individualmente aos métodos com maior impacto para a aplicação *Web* não só para garantir que as entidades envolvidas tem o comportamento correto mas também para detetar precocemente se a informação processada é ou não a correta.

6.1 Preparação dos Testes Unitários

Antes da realização dos testes aos métodos da *API* foi necessário criar cenários de teste e consequentemente fazer um levantamento dos pontos da cenário de teste que se pretende testar: método HTTP, dados de saída e código status que deve retornar.

Os valores que são resultado de cálculos matemáticos/estatísticos realizados pelos métodos da API responsáveis pelo tratamento estatístico foram obtidos recorrendo às aplicações *Accept gml* e *Accept sqc*.

Relativamente à informação devolvida pelo método *API* em teste foi utilizada a ferramenta *Postman* para auxiliar a validar os testes realizados. Esta ferramenta é um *REST Client* bastante popular para quem necessita realizar pedidos *HTTP* a partir de uma interface simples e intuitiva, facilitando o teste e depuração de serviços *REST* [44].

6.2 Implementação dos Testes Unitários

Após o levantamento dos pontos a serem testados, os testes foram implementados. Uma vez que os testes foram realizados aos métodos que representam apenas as funcionalidades mais importantes do sistema *Web*, não houve necessidade de criar um pacote de testes pelo que, todos os testes desenvolvidos encontram-se no ficheiro *tests.py*.

Para a realização dos testes unitários, o *Django REST framework* disponibiliza as classes *APIClient* e *APITestCase*, a partir das quais criamos as nossas próprias classes relativos aos nossos testes específicos.

A classe *APIClient* permite simular pedidos à *API* através da utilização de um cliente de teste (atributo *self.client* da classe) que substitui o cliente padrão do *Django*. Esta classe estende a classe *Client* do *Django*, no entanto é limitada aos métodos HTTP (*get()*, *post()*, *put()*, *delete()*, *head()*, *options()* e *trace()*) [28].

A classe *APITestCase* permite realizar testes aos métodos da API. Esta classe estende a classe *TestCase* do *Django* (subclasse que recorre à biblioteca padrão *Python: unittest*) [27].

De seguida na Listagem 6.1 é apresentado um exemplo de código que mostra como o método *HTTP delete()* é testado. Neste caso, é feito um teste ao método que permite eliminar um fornecedor através do seu *id*. Para isso é feito um pedido *DELETE* à *URL* fornecida.

```
1 def test_delete_supplier(self):
2
3     header = {"HTTP_COMPANYID": 1, "HTTP_USERID": 1}
4
5     response = self.client.delete('/suppliers/2/delete', content_type="
    application/json", **header)
6
7     self.assertEqual(response.status_code, status.HTTP_200_OK)
8
9     response = self.client.get('/suppliers/2', content_type="application/
```

```
    json", **header)
10
11      self.assertEqual(response.status_code, status.HTTP_404_NOT_FOUND)
12
13      pass
```

Listagem 6.1: Exemplo de código que mostra como o método HTTP *delete()* é testado.

Um teste unitário é feito através do método *assertEqual* que recebe como parâmetros o resultado do método associado à URL fornecida e o resultado esperado.

No exemplo apresentado, quando um fornecedor é eliminado com sucesso deverá ser retornado o *status HTTP* (*200_OK*). O teste unitário realizado está apresentado na linha 7 do código acima.

Para além deste teste unitário, foi feito um teste unitário ao método *get()* que pretende obter o fornecedor eliminado. Uma vez que o fornecedor já não existe pois foi eliminado, deverá retornar o *status HTTP*: *HTTP_404_NOT_FOUND*. O teste unitário realizado está apresentado na linha 11 do mesmo código.

É importante referir que, para os testes funcionarem corretamente, foi necessário fornecer os atributos de sessão e autenticação que se encontram definidos nos *headers*, que neste caso é o *id* da empresa e o *id* do utilizador.

6.3 Execução dos Testes Unitários

Para executar os testes desenvolvidos basta executar o comando de teste do *manage.py* do projeto:

```
python manage.py test Acceptgml
```

Este comando indica o número de testes executados bem como a duração de execução dos mesmos, tal como mostra a Figura 6.1. Se um teste não passar, é-nos mostrado detalhes sobre a falha do mesmo.

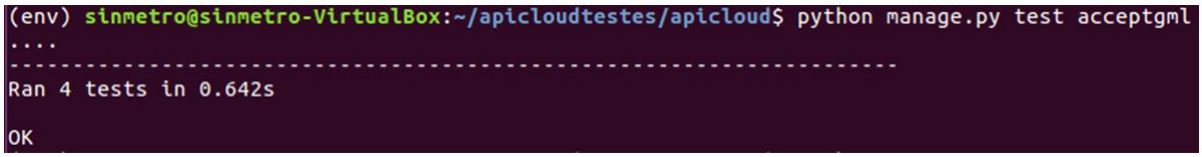

Figura 6.1: Exemplo de uma execução aos testes implementados

A realização de testes unitários aos métodos da *API* foi essencial no processo de desenvolvimento de *software* apresentando um conjunto de vantagens na realização dos mesmos.

Graças a este tipo de testes foi possível, de forma isolada, testar o comportamento do código sob os mais variados cenários de teste. Além disso, ao testar individualmente cada unidade, permite que o teste da soma dessas unidades tenha um maior grau de confiança e credibilidade.

Outra vantagem na utilização de testes unitários foi a realização dos mesmos à medida que os métodos da *API* foram sendo desenvolvidos. Desta forma, foi possível identificar falhas e consequentemente soluções para os métodos que não tinham o comportamento correto, evitando assim problemas futuros que podem por em causa a credibilidade do sistema *Accept*.

Além disso, os testes unitários permitem detetar se os métodos da *API* já existentes regrediram ou não com a implementação de novo código.

Capítulo 7

Conclusão

No decurso do projeto desenvolveu-se uma API REST que permitiu, de forma ágil e segura, integrar a aplicação *Web* que estava a ser paralelamente desenvolvida.

O trabalho desenvolvido durante o estágio permitiu apresentar uma solução bastante completa para a aplicação orientada ao desenvolvimento de serviços REST. Tendo em conta a solução apresentada e o objetivo do projeto na fase inicial, considero cumpridos os objetivos propostos, fazendo um balanço positivo do trabalho realizado.

Devido à arquitetura REST estar implementada na *framework Django* REST de forma nativa, permitiu-me desenvolver uma API de forma simples e rápida, através de uma solução onde as diferentes responsabilidades são claramente delimitados pelos diferentes elementos envolvidos.

Quanto à linguagem de programação, o *Python* permitiu implementar cálculos estatísticos de forma fácil e flexível, graças ao vasto conjunto de bibliotecas de estatísticas de elevada qualidade.

Em relação às dificuldades encontradas no desenvolvimento do projeto, considero que as mais desafiantes se deveram à falta de conhecimento específico sobre metrologia, nomeadamente sobre o controlo estatístico da quantidade de pré-embalados. O tratamento estatístico implementado recorreu a algumas fórmulas que aplicavam Legislação específica da metrologia que me eram totalmente desconhecidas, pelo que foi necessário um estudo prévio.

Por fim, outro fator que condicionou o desenvolvimento da API foi o planeamento inicial do projeto, uma vez que os requisitos, inicialmente, foram definidos de forma bastante genérica, sendo especificadas ao longo do desenvolvimento da API o que levou a constantes alterações a nível de programação.

Em termos de trabalho futuro destaca-se o alojamento da API REST em ambiente *cloud* e o desenvolvimento de novas funcionalidades bem como o melhoramento das já

existentes, com vista a melhorar a fiabilidade e a adaptação da aplicação às necessidades reais dos clientes do sistema Accept.

Bibliografia

[1] Stuckenberg, S., E. Fielt e T. Loser. 2011. The Impact Of Software-As-A-Service On Business Models Of Leading Software Vendors: Experiences From Three Exploratory Case Studies. Proceedings of the 15th Pacific Asia Conference on Information Systems (PACIS 2011).

[2] Fielding, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000.

[3] 9.2. math - Mathematical functions. `https://docs.python.org/2/library/math.html`. Online. Acedido a 23 de janeiro de 2018.

[4] 9.7. statistics - Mathematical statistics functions. `https://docs.python.org/3/library/statistics.html#module-statistics`. Online. Acedido a 23 de janeiro de 2018.

[5] AFERYMED - Aferição e Medidas. `http://www.aferymed.pt/`. Online. Acedido a 20 de dezembro de 2016.

[6] AFERYMED - Aferição e Medidas - O que são. `http://www.aferymed.pt/index.php?option=com_content&view=article&id=78&Itemid=233`. Online. Acedido a 20 de dezembro de 2016.

[7] API Rest e os verbos HTTP. `https://blog.mbeck.com.br/api-rest-e-os-verbos-http-46e189085e21`. Online. Acedido a 20 de dezembro de 2016.

[8] Balança FPVO-Waage FKTF. `https://www.kern-sohn.com/shop/pt/balancas-industriais/balancas-para-o-controlo-de-prod-pre-embalados/FKTF/`. Online. Acedido a 20 de dezembro de 2016.

[9] Controlo de Qualidade. `http://www.bessegato.com.br/UFJF/resources/table_of_control_chart_constants_old.pdf`. Online. Acedido a 23 de janeiro de 2018.

[10] Controlo de Qualidade. `https://edisciplinas.usp.br/pluginfile.php/4124750/mod_resource/content/1/Controle%20de%20Qualidade.pdf`. Online. Acedido a 23 de janeiro de 2018.

[11] Declaração de Retificação 71/2008 de 5 de dezembro. http://publicos.pt/documento/id440501/declaracao-de-retificacao-71/2008. Online. Acedido a 20 de dezembro de 2016.

[12] Decreto-Lei 199/2008 de 8 de outubro. http://publicos.pt/documento/id452915/decreto-lei-199/2008. Online. Acedido a 20 de dezembro de 2016.

[13] Decreto-Lei 291/90 de 20 de setembro. http://publicos.pt/documento/id557469/decreto-lei-291/90. Online. Acedido a 20 de dezembro de 2016.

[14] Decreto Lei n.°199/2008 de 8 de outubro. http://publicos.pt/documento/id452915/decreto-lei-199/2008. Online. Acedido a 29 de dezembro de 2016.

[15] Despacho 18853/2008. http://www.aferymed.pt/images/legislacao/Despacho_18853.2008_Taxa_Desloca%C3%A7%C3%A3o.pdf. Online. Acedido a 20 de dezembro de 2016.

[16] Despacho 8146/2004 (2a série). http://www.aferymed.pt/images/legislacao/Despacho_8146_2004.pdf. Online. Acedido a 20 de dezembro de 2016.

[17] Directiva 2007/45/CE do Parlamento Europeu e do Conselho de 5 de setembro de 2007. http://eur-lex.europa.eu/legal-content/PT/TXT/PDF/?uri=CELEX:32007L0045&from=PT. Online. Acedido a 20 de dezembro de 2016.

[18] Django - The web framework for perfectionistics with deadlines. https://www.djangoproject.com/. Online. Acedido a 8 de fevereiro de 2017.

[19] Django Documentation - File storage API. https://docs.djangoproject.com/en/2.0/ref/files/storage/. Online. Acedido a 23 de janeiro de 2018.

[20] Django Documentation - Making queries. https://docs.djangoproject.com/en/2.0/topics/db/queries/. Online. Acedido a 23 de janeiro de 2018.

[21] Django Documentation - Models. https://docs.djangoproject.com/en/2.0/topics/db/models/. Online. Acedido a 23 de janeiro de 2018.

[22] Django Documentation - URL dispatcher. https://docs.djangoproject.com/es/1.9/topics/http/urls/. Online. Acedido a 23 de janeiro de 2018.

[23] Django REST framework. http://www.django-rest-framework.org/. Online. Acedido a 8 de fevereiro de 2017.

[24] Django REST framework - Class-based Views. http://www.django-rest-framework.org/api-guide/views/. Online. Acedido a 23 de janeiro de 2018.

[25] Django REST framework - Model Serializer. http://www.django-rest-framework.org/api-guide/serializers/#modelserializer. Online. Acedido a 23 de janeiro de 2018.

[26] Django REST framework - Serializers. http://www.django-rest-framework.org/api-guide/serializers/. Online. Acedido a 23 de janeiro de 2018.

[27] Django Rest Framework - Testing (API Test cases. http://www.django-rest-framework.org/api-guide/testing/#api-test-cases. Online. Acedido a 24 de maio de 2017.

[28] Django Rest Framework - Testing (APIClient. http://www.django-rest-framework.org/api-guide/testing/#apiclient. Online. Acedido a 24 de maio de 2017.

[29] Escola Superior de Tecnologia e Gestão. http://www.ipleiria.pt/estg/. Online. Acedido a 20 de dezembro de 2016.

[30] Evolutionary Prototyping. http://www.teach-ict.com/as_a2_ict_new/ocr/A2_G063/331_systems_cycle/prototyping_RAD/miniweb/pg3.htm. Online. Acedido a 8 de fevereiro de 2017.

[31] HTTP Status Codes. https://restpatterns.mindtouch.us/HTTP_Status_Codes. Online. Acedido a 23 de janeiro de 2018.

[32] Incremental Prototyping. http://www.sqa.org.uk/e-learning/IMAuthoring01CD/page_09.htm. Online. Acedido a 8 de fevereiro de 2017.

[33] Instituto Politécnico de Leiria. http://www.ipleiria.pt/. Online. Acedido a 20 de dezembro de 2016.

[34] Instituto Português da Qualidade - Controlo Metrológico. http://www1.ipq.pt/pt/metrologia/scontrolometrologico/Pages/CM.aspx. Online. Acedido a 29 de dezembro de 2016.

[35] Instituto Português da Qualidade - o que é o controlo metrológico legal? http://www1.ipq.pt/pt/site/faq/Pages/FAQConteudo.aspx?TemaId=19&FaqId=ec9a0c91-f894-e511-ab7b-0050569e001a. Online. Acedido a 29 de dezembro de 2016.

[36] Instituto Português da Qualidade - Quais as entidades competentes no Controlo Metrológico Legal? http://www1.ipq.pt/pt/site/faq/Pages/FAQConteudo.aspx?TemaId=19&FaqId=bb843acb-f894-e511-ab7b-0050569e001a. Online. Acedido a 5 de janeiro de 2017.

[37] matplotlib - pyplot tutorial. https://matplotlib.org/users/pyplot_tutorial.html. Online. Acedido a 23 de janeiro de 2018.

[38] NumPy. http://www.numpy.org/. Online. Acedido a 23 de janeiro de 2018.

[39] Object-relational mappers (ORMs). https://www.fullstackpython.com/object-relational-mappers-orms.html. Online. Acedido a 8 de fevereiro de 2017.

[40] Oracle MySQL. https://www.oracle.com/br/mysql/index.html. Online. Acedido a 8 de fevereiro de 2017.

[41] ORM - Object Relational Mapper. https://www.devmedia.com.br/orm-object-relational-mapper/19056/. Online. Acedido a 8 de fevereiro de 2017.

[42] os path - Common pathname manipulations. https://docs.python.org/2/library/os.path.html. Online. Acedido a 23 de janeiro de 2018.

[43] Portaria 1198/91 de 18 de dezembro. http://publicos.pt/documento/id371272/portaria-1198/91. Online. Acedido a 20 de dezembro de 2016.

[44] POSTMAN. https://www.getpostman.com/. Online. Acedido a 24 de maio de 2017.

[45] Pycharm: PYCHARM FEATURES. https://www.jetbrains.com/pycharm/features/. Online. Acedido a 8 de fevereiro de 2017.

[46] Python. https://wiki.python.org/moin/BeginnersGuide/Overview. Online. Acedido a 8 de fevereiro de 2017.

[47] Python - Built-in Functions. https://docs.python.org/3/library/functions.html. Online. Acedido a 23 de janeiro de 2018.

[48] python - pdfkit 0.6.1. https://pypi.python.org/pypi/pdfkit. Online. Acedido a 23 de janeiro de 2018.

[49] Python - simplejson 3.13.2. https://pypi.python.org/pypi/simplejson/. Online. Acedido a 23 de janeiro de 2018.

[50] Python - The History of Python: Introduction and Overview. http://python-history.blogspot.pt/2009/01/introduction-and-overview.html. Online. Acedido a 8 de fevereiro de 2017.

[51] Reshaping IT Project Delivery Through Extreme Prototyping. http://archive.oreilly.com/pub/a/onjava/2006/11/15/

reshaping-it-project-delivery-through-extreme-prototyping.html. Online. Acedido a 8 de fevereiro de 2017.

[52] Reviewing Django REST Framework By Alex Holmes. https://www.isotoma.com/blog/2014/03/25/reviewing-django-rest-framework/. Online. Acedido a 8 de fevereiro de 2017.

[53] Richardson Maturity Model. https://martinfowler.com/articles/richardsonMaturityModel.html. Online. Acedido a 20 de dezembro de 2016.

[54] SciPy - numpy.std. https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.std.html. Online. Acedido a 23 de janeiro de 2018.

[55] SciPy.org. https://www.scipy.org/. Online. Acedido a 23 de janeiro de 2018.

[56] SciPy.org - scipy.stats.t. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.t.html. Online. Acedido a 23 de janeiro de 2018.

[57] SDLC - Software Prototype Model. https://www.tutorialspoint.com/sdlc/sdlc_software_prototyping.htm. Online. Acedido a 8 de fevereiro de 2017.

[58] simplejson - JSON encoder and decoder. https://simplejson.readthedocs.io/en/latest/. Online. Acedido a 23 de janeiro de 2018.

[59] SINMETRO. http://web.sinmetro.pt. Online. Acedido a 23 de agosto de 2017.

[60] SINMETRO - Sistema ACCEPT. http://web.sinmetro.pt/accept/. Online. Acedido a 23 de agosto de 2017.

[61] SOAP VS REST - A NordicAPIs infographic. https://nordicapis.com/rest-vs-soap-nordic-apis-infographic-comparison/. Online. Acedido a 20 de dezembro de 2016.

[62] Suporte ao Minitab 18 - Limites de especificação na análise de capacidade. https://support.minitab.com/pt-br/minitab/18/help-and-how-to/quality-and-process-improvement/capability-analysis/supporting-topics/basics/specification-limits/. Online. Acedido a 23 de janeiro de 2018.

[63] Suporte ao Minitab 18 - O que são limites de controlo? https://support.minitab.com/pt-br/minitab/18/help-and-how-to/quality-and-process-improvement/control-charts/supporting-topics/basics/what-are-control-limits/. Online. Acedido a 23 de janeiro de 2018.

[64] The Django Book - The Model-View-Controller Design Pattern. http://djangobook.com/model-view-controller-design-pattern/. Online. Acedido a 8 de fevereiro de 2017.

[65] 'Throw-away' Prototyping. http://www.teach-ict.com/as_a2_ict_new/ocr/A2_G063/331_systems_cycle/prototyping_RAD/miniweb/pg4.htm. Online. Acedido a 8 de fevereiro de 2017.

[66] Trello: como esta ferramenta pode ajudar vocÃª a organizar a sua vida. https://www.tecmundo.com.br/organizacao/75128-trello-ferramenta-ajudar-voce-organizar-vida.htm. Online. Acedido a 8 de fevereiro de 2017.

[67] URL regular expressions. https://www.webforefront.com/django/regexpdjangourls.html. Online. Acedido a 23 de janeiro de 2018.

[68] URL (Uniform Resource Locator - definition.

[69] Using HTTP Methods for RESTful Services. http://www.restapitutorial.com/lessons/httpmethods.html. Online. Acedido a 20 de dezembro de 2016.

[70] WHAT IS SCRUM? https://www.scrum.org/resources/what-is-scrum. Online. Acedido a 8 de fevereiro de 2017.

[71] (2004). International Recommendation - OIML R 87. https://www.oiml.org/en/files/pdf_r/r087-e04.pdf. International Organization of Legal Metrology.

[72] (2012). Centro Integrado de Capacitação em Metrologia e Avaliação da Conformidade (CICMAC) - Inmetro: Conceito de Metrologia. http://www.inmetro.rs.gov.br/cicmac/material_didatico/polig_conceito_metrologia.pdf. Porto Alegre, Brasil.

[73] (2012). Vocabulário International de Metrologia (VIM) - Conceitos Fundamentais e Gerais e Termos Associados. http://www1.ipq.pt/PT/Metrologia/Documents/VIM_IPQ_INMETRO_2012.pdf. 1.ª Edição Luso Brasileira, Caparica, Portugal.

[74] (2014). Controlo Estatístico de Produtos Pré-embalados. http://www1.ipq.pt/PT/IPQ/historico_eventos/Documents/Metrologia%20Setor%20Alimentar%202014-10-30/Apresentacao%20Aferymed-30%20OUT.pdf. IPQ - Seminário: Metrologia no Setor Alimentar.

[75] Machado, F. P. Controlo Metrológico de Pré-embalados. http://www.qualidademadeira.com.pt/ficheiros/documentos/Seminarios/filipe_pinto_machado.pdf. Online. Acedido a 20 de dezembro de 2016.

[76] Montgomey, D. (2013). *Introduction to Statistical Quality Control* (7 ed.). EUA: John Wiley & Sons, Inc.

Anexos

Anexo A - Exemplo de Relatório em PDF gerado pela API que contém o estudo de capacidade efetuado a um processo.

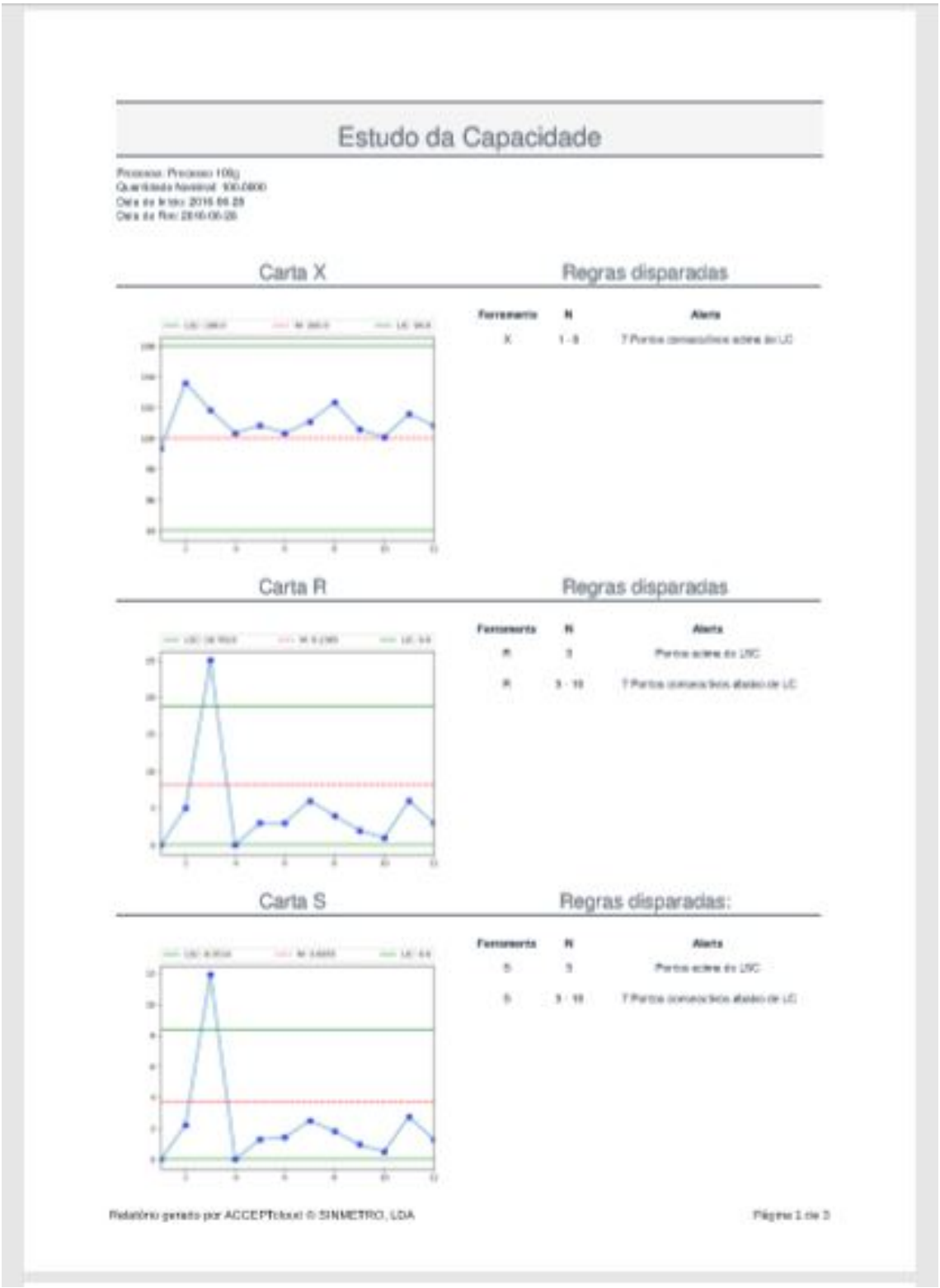

Figura 1: Exemplo do Relatório sobre o Estudo da Capacidade - Página 1

Histograma

Dados do Processo

CARTAS DE CONTROLO		DESCRIÇÃO		ESTATÍSTICA		CAPACIDADE	
CARTA X		n:	48	LIE:	94.00	Cp:	0.93
LSC:	100.00	# Amostras:	12	LSE:	106.00	Cpk:	0.70
LIC:	94.00	N:	4	n < LIE:	1	Cpk Superior:	0.70
LCC Média:	100.00	Amplitude:	25.80	n > LSE	2	Cpk Inferior:	1.09
		Amplitude Média:	8.83	LIE < n < LSE	45	Sigma	2.33
CARTA R		Máximo:	114.34	P% < LIE:	0.10 %	Desvio-Padrão:	3.47
LSC:	18.78	Mínimo:	88.54	P% > LSE:	1.70 %	Desvio-Padrão Médio:	2.22
LIC:	0.00	Média:	101.07	LIE < P% < LSE:	98.80 %		
LCC Média:	8.29						
CARTA S							
LSC:	8.56						
LIC:	0.00						
LCC Média:	3.69						

Controlo Metrológico

TL1:	95.00	n < TL1:	2	P% < TL1:	0.80 %
TL2:	91.00	n < TL2:	1	P% < TL2:	0.00 %

Observações

Relatório gerado por ACCEPTicloud © SINMETRO, LDA

Página 2 de 3

Figura 2: Exemplo do Relatório sobre o Estudo da Capacidade - Página 2