

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



***API mashup in a collaborative logistics
platform***

Luís Filipe Fernandes da Costa Melo

MESTRADO INTEGRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES

Orientador: André Restivo

Co-orientador: Tiago Silva

25 de junho de 2017

API mashup in a collaborative logistics platform

Luís Filipe Fernandes da Costa Melo

**MESTRADO INTEGRADO EM ENGENHARIA ELETROTÉCNICA E DE
COMPUTADORES**

25 de junho de 2017

Resumo

Vivemos num mundo cada vez mais consumista onde se tende a efetuar progressivamente mais compras *online*. O mundo do *ecommerce* possui uma estrutura própria com muitos intervenientes e consequentemente processos logísticos complexos. Sendo a *HUUB* uma empresa de logística com base tecnológica que atua no setor da moda, o *ecommerce* é uma área importante para o seu negócio, cuja finalidade é encurtar a distância entre as marcas e os seus clientes. Sendo assim, a *HUUB* tenta-se posicionar como o centro de um ecossistema que conecte toda uma cadeia de abastecimento.

O objetivo desta dissertação é, então, o desenvolvimento de uma API que automatize e melhore alguns processos internos da empresa como o processo de *shipping* e a sincronização das principais plataformas de *ecommerce* com o ERP da empresa. Desta forma, foi desenvolvido uma *mashup* de *third party* APIs para integrar o sistema da *HUUB* com as diversas transportadoras e os vários serviços de *ecommerce*.

O desenvolvimento foi partido em três partes: automatização de todo o processo de *shipping*, que envolve recolher *rates* de transporte dos diferentes *carriers*, comprar e produzir a *label* de transporte e fazer *tracking* da encomenda; integração das principais plataformas de *ecommerce*, como o *Woocommerce* e o *Shopify* com o ERP da empresa de modo a que seja possível existir uma sincronização bidirecional da informação; e um sistema para notificação de alertas.

Esta dissertação marca o início da mudança do paradigma de desenvolvimento da *HUUB*, partindo do atual sistema monolítico para uma arquitetura baseada em micro serviços, tendo sempre em mente boas práticas de desenho e desenvolvimento de *software* orientado por objetos.

Abstract

We live in an increasingly consumerist world that tends to make more and more purchases online. The e-commerce world has its own structure with many stakeholders and consequently, complex logistical processes. *HUUB*, being a technological logistics company that operates in the fashion industry, considers e-commerce as an important area for its business as it aims at shortening the distance between brands and their customers by being the center of an ecosystem that connects an entire supply chain.

The goal of this dissertation is the development of an API that automates and improves some internal processes of the company as the process of shipping and the synchronization of the main ecommerce platforms with the company's ERP. In this way, a mashup of third party APIs was created to integrate the *HUUB*'s system with the various carriers and the various ecommerce services.

The development was divided into three parts: automation of the entire shipping process, which involves collecting transportation rates from different carriers, buying and generating the transport label and, the last one, tracking the order until the reception by the customer; integration of leading e-commerce platforms such as Woocommerce and Shopify with the company's ERP so that there is a two-way synchronization of information; and a system for notification of important alerts.

This dissertation marks the beginning of the change in *HUUB*'s development paradigm, starting from the current monolithic system and reaching a micro-service-based architecture, always having in mind good practices in design and development of object-oriented software.

Agradecimentos

Este espaço, ainda que pequeno, destina-se a agradecer a todas as pessoas que de uma maneira ou de outra, contribuíram para o meu sucesso académico e me acompanharam durante estes últimos 5 anos.

Em primeiro lugar, gostaria de agradecer ao Professor Doutor André Restivo, por me ter guiado durante o todo o desenvolvimento e por se mostrar sempre disposto a receber-me.

A todos os colaboradores da *HUUB*, principalmente ao Engenheiro Tiago Silva e restante equipa de IT e BI, pelo excelente ambiente de trabalho, entajuda e constante partilha de conhecimento.

Aos meus amigos de sempre, pelos convívios de fim de semana, pelos grandes momentos passados e por me acompanharem durante grande parte da minha vida.

A todas as amizades que fiz durante a faculdade, pelo espírito de cooperação sempre presente e por terem caminhado ao meu lado durante todo o percurso.

Aos meus pais, por serem a minha pedra angular, pelo apoio incondicional, e pela educação que sempre me proporcionaram, e à minha irmã, pela enorme paciência e por estar sempre presente.

A toda a minha família, por terem sempre a porta aberta e comida na mesa para me receber, principalmente ao meu avô Bernardino, pela amizade e pelos valores que sempre me transmitiu, ao meu avô Francisco, pela inigualável boa disposição e pelo interesse que continuamente demonstra e à minha avó Emília, pelo amor e preocupação que sempre teve por mim.

A todos, um enorme obrigado.

Luís Melo.

*“Any fool can write code that a computer can understand.
Good programmers write code that humans can understand.”*

Martin Fowler (2008)

Conteúdo

1	Análise Introdutória	1
1.1	Enquadramento	1
1.2	Motivação	3
1.3	Objetivos	6
1.4	Estrutura do documento	7
1.5	Conclusão	8
2	Estado da Arte	9
2.1	O ecossistema das lojas de <i>ecommerce</i>	9
2.2	<i>Mashup</i> de <i>third party</i> API	10
2.3	Plataformas existentes para integração com <i>carriers</i>	12
2.4	Plataformas existentes para integração com as lojas de <i>ecommerce</i>	15
2.4.1	<i>Mashup</i> de APIs de <i>ecommerce</i>	16
2.4.2	Lojas de <i>ecommerce</i> que disponibilizam API	17
2.5	Plataformas existentes para notificação por <i>email</i>	19
2.6	Conclusão	21
3	Caracterização do Problema	23
3.1	O processo da <i>HUUB</i>	23
3.2	Automatização do processo de <i>shipping</i>	25
3.3	Integração com lojas de <i>ecommerce</i>	26
3.4	Notificação de intervenientes	27
3.5	Requisitos	27
3.6	Conclusão	29
4	Solução proposta	31
4.1	Automatização do processo de <i>shipping</i>	31
4.2	Integração com lojas de <i>ecommerce</i>	33
4.3	Notificação de intervenientes	35
4.4	Arquitetura e tecnologia a usar	35
4.5	Conclusão	36
5	Implementação	37
5.1	Arquitetura e tecnologia a usar	37
5.1.1	Arquitetura	37
5.1.2	Esquema da Base de Dados	38
5.1.3	Segurança	39
5.1.4	GUI para gestão administrativa e tecnológica	39

5.2	Automatização do processo de <i>shipping</i>	42
5.2.1	Envio de <i>orders</i>	42
5.2.2	<i>Tracking</i> de <i>orders</i>	51
5.3	Integração com lojas de <i>ecommerce</i>	52
5.3.1	<i>Products</i>	54
5.3.2	<i>Orders</i>	57
5.3.3	<i>Customers</i>	59
5.3.4	<i>Webhook management</i>	59
5.3.5	<i>Webhooks</i>	59
5.3.5.1	Atualização de um produto	61
5.3.5.2	Atualização de uma <i>order</i>	62
5.4	Notificação de intervenientes	64
5.4.1	Envio de emails	65
5.4.2	Log de emails	67
5.5	Conclusão	68
6	Conclusões e trabalho futuro	69
6.1	Satisfação dos objetivos	69
6.2	Conclusões e trabalho futuro	69
	Referências	71
A	Normalização de classes	75
B	Documentação gerada	81

Lista de Figuras

1.1	Página inicial do <i>Spoke</i>	2
1.2	Posição da <i>HUUB</i> no mercado.	3
1.3	Diferença entre uma arquitetura monolítica e baseada em micro serviços.	5
1.4	Crescimento do número de APIs públicas disponibilizadas.	6
2.1	Exemplo do ecossistema de um negócio <i>online</i>	9
3.1	Estratégia de venda e distribuição dos produtos da nova <i>season</i> de um cliente da <i>HUUB</i>	24
4.1	Distribuição do número de envios e custo total feitos através de cada <i>carrier</i>	32
4.2	Distribuição do número de <i>HUUB clients</i> da <i>HUUB</i> por loja <i>online</i>	33
4.3	Diagrama das classes responsáveis pela integração de <i>ecommerce</i>	34
4.4	Arquitetura proposta para a <i>HUUB</i>	36
5.1	Arquitetura implementada para a <i>HUUB</i>	38
5.2	Esquema do <i>schema audit</i> da base de dados.	39
5.3	Esquema do <i>schema integrations</i> da base de dados.	40
5.4	Esquema do <i>schema public</i> da base de dados.	41
5.5	GUI para gestão administrativa. Vista dos <i>API Users</i>	41
5.6	Classe <i>Shipping</i> que trata de todas as operações do envio de <i>orders</i>	42
5.7	Diagrama de sequência para obtenção de <i>rates</i> para um transporte.	47
5.8	<i>Label</i> de transporte criada através do <i>Easypost</i>	50
5.9	Diagrama de sequência para criação de uma <i>label</i> de transporte.	51
5.10	Classe <i>Tracking</i> que trata de todas as operações relacionadas com o <i>tracking</i> das <i>orders</i>	51
5.11	Diagramas de classes da estrutura montada para as classes das lojas <i>ecommerce</i>	53
5.12	Diagramas de classes da estrutura montada para as classes dos produtos.	55
5.13	Diagramas de classes da estrutura montada para as classes das <i>orders</i>	58
5.14	Diagramas de classes da estrutura montada para as classes de cada <i>order item</i>	58
5.15	Diagramas de classes da estrutura montada para as classes de <i>customer</i>	60
5.16	Diagramas de classes da estrutura montada para as classes de <i>address</i>	60
5.17	<i>Flowchart</i> do processo de atualização de <i>orders</i> via <i>webhook</i>	63
5.18	Classe <i>Mail</i> que trata do envio de <i>emails</i>	64
5.19	<i>Email</i> enviado para notificação do término de um <i>job</i>	65
5.20	<i>Email</i> enviado para notificação de itens de uma <i>order</i> que não estão em sistema.	66
5.21	<i>Email</i> enviado para notificar <i>updates</i> ao <i>tracking</i> de uma <i>order</i>	67
5.22	Dashboard de <i>emails</i> criado para resolução de anomalias e exceções.	68

Lista de Tabelas

2.1	Análise comparativa das diferentes opções para <i>mashup</i> das plataformas de <i>shipping</i> .	15
2.2	Análise comparativa das diferentes opções para <i>mashup</i> de lojas <i>online</i> de <i>ecommerce</i>	17
2.3	Análise comparativa das APIs das diferentes lojas <i>online</i> de <i>ecommerce</i>	19
2.4	Análise comparativa das diferentes opções para envio de <i>emails</i>	21
3.1	Requisitos Funcionais.	28
3.2	Requisitos não funcionais.	29
4.1	<i>Endpoints</i> para <i>shipping</i> propostos para o cumprimento dos requisitos.	32
4.2	<i>Endpoints</i> para <i>ecommerce</i> propostos para o cumprimento dos requisitos.	34
4.3	<i>Endpoints</i> para o envio de notificações propostos para o cumprimento dos requisitos.	35
5.1	<i>Endpoints</i> criados para o processo de <i>shipping</i>	42
5.3	<i>Endpoints</i> criados para consumo das lojas <i>online</i> no processo de <i>ecommerce</i>	54
5.2	<i>Endpoints</i> criados para consumo interno da <i>HUUB</i> no processo de <i>ecommerce</i>	56
5.4	<i>Endpoints</i> criados para o processo de notificação dos intervenientes.	65

Siglas

AI Artificial Intelligence. 3

AM Account Manager. 3, 24, 28, 61, 62, 65, 66, 69, 70

API Application Programming Interface. 1, 4–6, 8, 10–21, 26, 28, 29, 35, 39, 41, 45, 46, 52, 54, 62, 65, 70, 81

AWS Amazon Web Services. 38

B2B Business to Business. 2

B2C Business to Customer. 2, 3

BD Base de dados. 4, 36, 38, 59, 65

BI Business Intelligence. 3, 24, 52, 62, 66

CC Carbon Copy. 64

CRUD Create, Read, Update and Delete. 11, 28, 39

CTO Chief Technology Officer. 4

DevOps Development Operations. 4

ERP Enterprise Resource Planning. 1, 6, 10, 23

GUI Graphical User Interface. 39

HTTP Hyper Text Transfer Protocol. 10, 11, 16, 61

HTTPS Hyper Text Transfer Protocol Secure. 10

IT Information Technology. 3

JSON JavaScript Object Notation. 10, 43, 46, 47, 49, 61

LTS Long Term Support. 38

MDND Mozilla Developer Network. 10

OOP Object Oriented Programming. 33, 37, 69

ORM Object-Relational Mapping. 38

PHP Hypertext Preprocessor. 13, 20, 38, 43

PO Purchase Order. 2, 24

REST Representational State Transfer. 6, 10, 12, 13, 17, 18, 39

SDK Software Development Kit. 11, 13

SKU Stock Keeping Unit. 56, 59

SO Sales Order. 2, 3, 24, 25, 28, 47, 62

SOA Service-oriented architecture. 3, 8, 36, 38, 69

SOAP Simple Object Access Protocol. 10, 18

SSH Secure Shell. 38

TTL Time To Live. 36, 43, 46, 56, 57

URL Uniform Resource Locator. 59, 67

WMS Warehouse Management System. 10

XML eXtensible Markup Language. 10, 11

Glossário

Account Manager é o responsável e a ponte entre os clientes e a *HUUB*. xv, 3

Cron é o agendamento de um *script* que é executado periodicamente, sendo esse período configurável. 26, 64

Design Pattern (em português, padrão de desenho) é uma solução típica para um problema considerado recorrente. 52–54, 57, 64, 75

Development Operations é um termo que surgiu numa conferência de metodologias *Agile* no desenvolvimento de *software*. *DevOps* é a função de preparar e manter toda a infraestrutura necessária para o software. São normalmente associados à manutenção de servidores e responsáveis por fazer chegar à equipa de desenvolvimento métodos de *Continuous Delivery* e *Continuous Integration*. xv, 4

Forecast é a previsão do custo de transporte de uma dada mercadoria. Como o forecast é realizado cerca de um mês antes do envio, o valor resultante pode não ser igual ao que vai custar no momento do envio. 24, 25, 43

Freemium é um modelo de negócio que providencia ao utilizador uma amostra dos seus serviços gratuitamente no entanto, para uma utilização mais regular, é necessário pagar por um serviço *premium*. 12

Fulfilment é o processo de expedição de parte ou da totalidade dos produtos de uma *order*. 2, 18, 19, 59, 70

Mashup é um serviço que integra numa só aplicação, diferentes serviços externos. 5, 11, 12, 15, 16, 21, 33, 70

Micro Serviço está normalmente associado a uma arquitetura orientada aos serviços, pressupondo que a lógica de uma aplicação complexa está dividida em serviços mais simples e independentes permitindo um desenvolvimento mais modular da aplicação. 3–5, 39

Object-Relational Mapping é uma técnica que permite gerir dados provenientes de uma base de dados, numa lógica semelhante à programação orientada por objetos. xvi, 38

Onboard é a primeira fase no processo de lançamento de uma nova *season*. Esta fase engloba a sincronização dos novos produtos e informação de *customers* para o Spoke, seguindo-se a preparação das SOs para envio. 24

Sistema Monolítico é, como o próprio nome indica, um sistema gigante, massivo. Em software, é quando todas as funcionalidades de um sistema grande estão centralizadas numa só aplicação, tornando-se, geralmente, pouco flexível e difícil de escalar. 4, 5, 35

Stock Keeping Unit é um código único que identifica um certo produto. xvi, 56

Text Mining é um processo de análise de texto. 61

Time To Live é o tempo que um registo vai existir em *cache*. Ao fim desse tempo, o registo é automaticamente eliminado. xvi, 36

Waybill é o *tracking number*. É único dentro do mesmo *carrier* e serve para identificar o *shipment*. 59

Webhook é um pedido HTTP normalmente definido pelo utilizador e que é disparado aquando a ocorrência de um certo evento. 16–19, 28, 35, 39, 42, 52, 54, 56, 59, 62, 65, 67, 70

Capítulo 1

Análise Introdutória

Neste capítulo estará expresso o contexto e a motivação para o desenvolvimento desta dissertação bem como os objetivos que se esperam alcançar com a mesma.

1.1 Enquadramento

Esta dissertação foi desenvolvida em ambiente empresarial, tendo como proponente externo a *HUUB* e como objetivo o desenvolvimento de uma aplicação de integração de *Application Programming Interfaces (APIs)* de terceiros com o *Enterprise Resource Planning (ERP)* desenvolvido pela empresa.

A *HUUB* é uma *startup* com aproximadamente trinta funcionários que atua principalmente no setor da moda. Foi criada em 2015, tem cerca de 100 000 produtos em *stock* e já conta com mais de 7 000 envios para mais de 60 países em todo o mundo.

O seu foco é encurtar a distância entre as marcas e os seus clientes sendo o centro de um ecossistema que conecta toda uma cadeia de abastecimento. O *core business* da empresa é o armazenamento de produtos dos seus clientes, a sua distribuição e a disponibilização de uma plataforma, denominada *Spoke*, de gestão e logística em tempo real de todo esse sistema.

O *Spoke*, Figura 1.1, é uma aplicação que reúne, no mesmo lugar, toda a informação necessária para uma boa gestão, desde as encomendas efetuadas até às ordens de expedição e transporte, oferecendo às marcas uma forma intuitiva de gerir toda a sua cadeia de abastecimento, desde os fornecedores até ao cliente final.

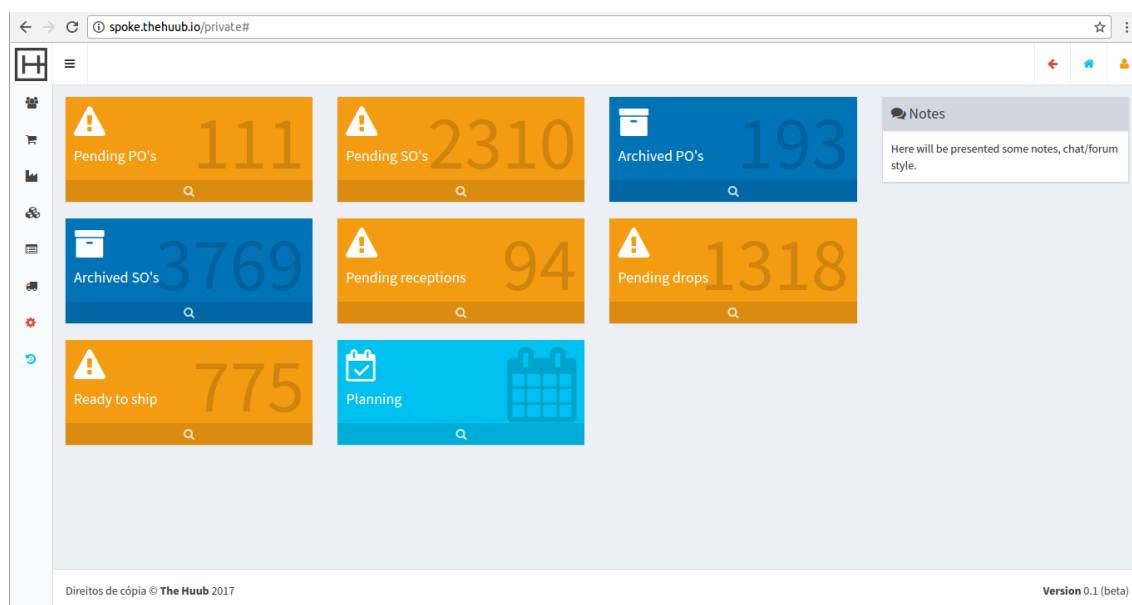


Figura 1.1: Página inicial do *Spoke*.

Ao longo desta dissertação serão mencionados termos usados internamente pela *HUUB*. Tratando-se de uma empresa de logística que atua no setor da moda, a *HUUB* é responsável pela gestão da cadeia de abastecimento dos seus clientes. Esses clientes, marcas de roupa e/ou acessórios, serão chamados de *HUUB clients*. Estas marcas vendem os seus produtos aos seus clientes, sendo estes designados como *customers* (cliente final) na visão da *HUUB*. Os fornecedores dos *HUUB clients* são designados de *suppliers*.

No que concerne às *orders* existem dois tipos: as *Purchase Orders* (POs), que são as encomendas dos *HUUB clients* aos *suppliers* e que dão entrada no armazém da *HUUB*; e as *Sales Orders* (SOs), que são encomendas feitas pelos *customers* aos *HUUB clients* e que dão saída do armazém da *HUUB*. As POs podem ser divididas em *receptions* por questões de otimização dos envios. Da mesma maneira, uma SO pode ser dividida em uma ou em várias *drops*. Quando uma *drop* é enviada, existe *fulfilment* dessa *drop*. Caso o número de *drops* enviadas (*fulfilled*), associadas a uma SO, seja inferior ao número de *drops* total em que essa SO foi dividida, existe *fulfilment* parcial da SO. O *fulfilment* completo de uma SO é feito quando todas as *drops* associadas a essa SO forem enviadas.

A *HUUB* possui clientes que atuam principalmente em dois mercados o *Business to Business* (B2B) e o *Business to Customer* (B2C).

O primeiro trata-se da venda em retalho. Aqui, os *customers* são os diversos retalhistas que vendem os produtos do *HUUB client* e o objetivo é fornecer uma plataforma em que os *suppliers* e as marcas (*HUUB clients*) possuam acesso à mesma informação. Com a inclusão dos *carriers* (empresas transportadoras) é criado todo um ecossistema em que o *HUUB client* se abstrai de todo o complexo processo logístico.

Já o segundo, B2C, é composto principalmente pelo mercado de *ecommerce*. Neste tipo de venda, o *customer* é o cliente final. Aqui também é importante manter toda a ligação entre *supplier* e *HUUB client* bem como a integração com os *carriers* para que as compras feitas nas lojas *online* dos *HUUB clients* (SOs) sejam tratadas, processadas e enviadas rapidamente e com o mínimo de intervenção humana.

A *HUUB* conta então com uma equipa de *Account Managers* (AMs), que são também utilizadores regulares do *Spoke* pois estão em contacto permanente com os *HUUB clients*, sendo responsáveis por gerir as suas informações e com uma equipa financeira, que gere todo o fluxo monetário e pagamentos. A *HUUB* possui também uma equipa de *Information Technology* (IT), que trata do desenvolvimento do *Spoke* e de integrações com serviços externos, uma equipa de *Business Intelligence* (BI) e *Artificial Intelligence* (AI), que desenvolvem algoritmos e estrutura para otimizar processos ou mesmo o negócio em si. Por último, mas não menos importante, tem-se a equipa de operação, formada por cerca de quinze trabalhadores cujo objetivo é garantir o funcionamento de todo o processo logístico.

A grande missão da *HUUB* é, portanto, contribuir para o crescimento de todos os intervenientes da cadeia de abastecimento, centralizando a informação e fazendo uso de algoritmos e técnicas de otimização da logística inerente a este tipo de mercados (Figura 1.2), deixando os seus clientes apenas com a preocupação do *design* dos seus produtos e das vendas, servindo mesmo de aceleradora para o seu crescimento.

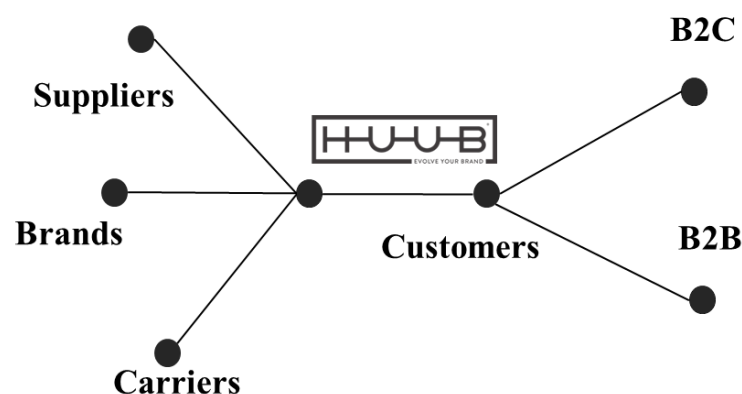


Figura 1.2: Posição da *HUUB* no mercado.

1.2 Motivação

A programação com base em *micro serviços* segue um paradigma *Service-oriented architecture* (SOA) cujo mote é, nas palavras de Juval Lowy, pioneiro dos *micro serviços*, "*Every class as a Service*" [1].

Este paradigma é associado por muitos autores ao termo Web 2.0 [2] que põem como pilares da Web, atualmente, as integrações, as APIs e os *web services*. Nas palavras de Lori MacVittie, *"It's further becoming a requirement of Web 2.0 sites that they provide some sort of API through which developers can write add-on applications"* [3], ou seja, tem por base uma arquitetura distribuída em módulos independentes. As grandes vantagens que este tipo de desenvolvimento oferece são, segundo Trevor Richardson [4]:

flexibilidade como são desenvolvidos *micro serviços* independentes e de pequena dimensão, todo o processo de desenvolvimento torna-se flexível, podendo optar-se por diferentes linguagens e tecnologias entre diferentes *micro serviços*, bem como descentralização do alojamento da informação (Base de dados (BD));

escalabilidade como cada *micro serviço* age independentemente dos outros, quando uma certa função de uma aplicação precisa de ser alterada ou escalada, esta pode ser feita sem que seja necessário alterar outros *micro serviços*;

modularidade dada esta independência, a falha de um *micro serviço* não implica uma falha da aplicação, sendo assim mais fácil identificar e corrigir essa falha, resultando, também, numa diminuição do *down time* da funcionalidade.

No entanto, este tipo de desenvolvimento também traz desvantagens, sendo alguma delas enunciadas por Benjamin Wootton, Chief Technology Officer (CTO) da *Contino*, uma consultora de Development Operations (DevOps) situada em Londres [5]:

- complexidade em desenhar a arquitetura e manter um sistema distribuído, pois este é bastante labiríntico, crescendo essa complexidade com o aumento do número de serviços;
- duplicação de esforço, isto é, irão existir serviços bastante semelhantes entre si mas que têm finalidades diferentes e, por essa razão, estão em serviços separados;
- aumento substancial de DevOps pois cada *micro serviço* tem um ambiente próprio sendo que, em muitos casos, cada *micro serviço* vive em máquinas separadas e usa tecnologia diferente (por exemplo, serviços que usam uma BD não relacional e outros que usam BD relacional).

Ao invés, um *sistema monolítico* é um sistema completamente centralizado, cujas funcionalidades são construídas em cima da aplicação, tornando-se assim uma aplicação pesada e de difícil compreensão. Este conceito já é utilizado há bastante tempo pela comunidade, tendo sido descrito no livro *The Art of Unix Programming* como um sistema que fica demasiado grande, monstruoso [6]. No entanto, este tipo de arquitetura também tem as suas vantagens, como por exemplo, o facto de estar toda com a mesma linguagem e tecnologias, não sendo necessário ser proficiente em diversas áreas, a facilidade de integração e a similaridade entre as várias peças que constituem esse *sistema monolítico*.

Na Figura 1.3 pode verificar-se as diferenças entre estes dois tipos de arquitetura.

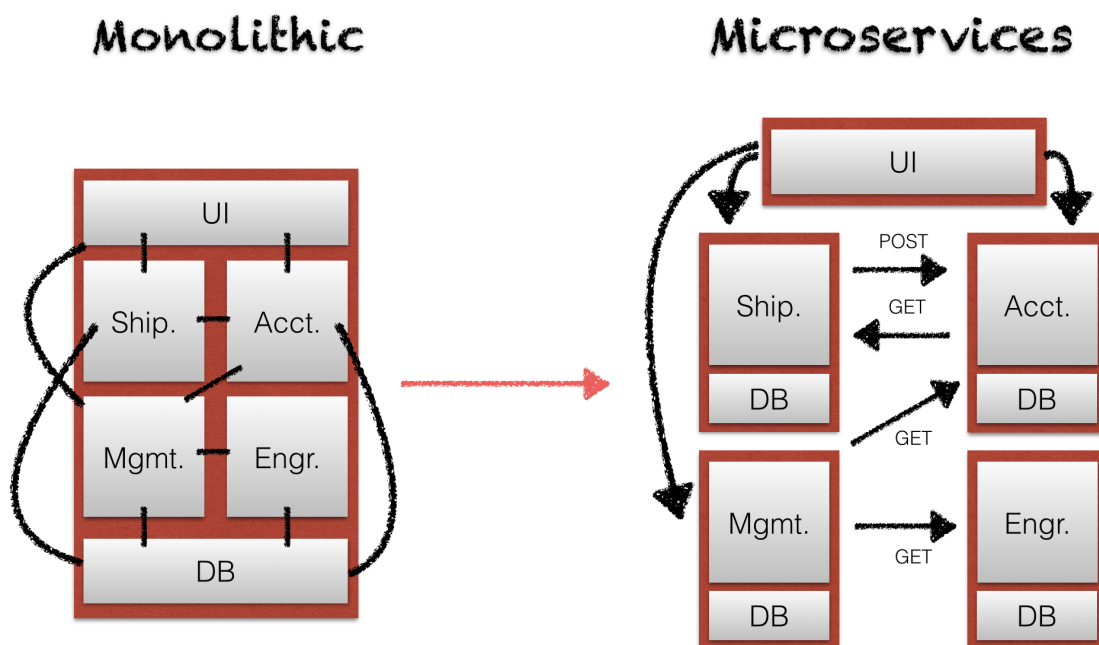


Figura 1.3: Diferença entre uma arquitetura monolítica e baseada em micro serviços [4].

O *Spoke* foi desenvolvido há relativamente pouco tempo em *Python* numa *framework* chamada *web2py*. É um sistema monolítico pois, aquando da criação, era necessário construir uma aplicação eficiente e em pouco tempo. Atualmente, esta solução não está de todo otimizada e a *HUUB* pretende refazê-la.

Com isto, e numa perspetiva de preparar o futuro, surge a necessidade de expandir esta plataforma, criando uma nova aplicação que providencie uma série de *web-services* e *micro serviços*, permitindo que a integração de *third party API*, como as plataformas de *ecommerce* e as transportadoras, seja efetuada de forma transparente, aumentando assim a qualidade do serviço prestado.

Aqui, surge um novo conceito, *mashup* que, segundo Darlene Fichter, "*is a web application that uses content from more than one source to create a single new service [...]*." [7]. Este tipo de aplicação começou a existir porque as grandes empresas como Amazon, Google, Netflix, etc começaram a disponibilizar os seus dados ao público através de *APIs*. No gráfico da Figura 1.4 pode constatar-se que, desde 2005, o número de *APIs* públicas disponibilizadas está a sofrer um crescimento substancial, existindo atualmente mais de 16 mil contabilizadas.

É neste conceito que surge o tema desta dissertação, começar o desenvolvimento de *micro serviços* na *HUUB* enquanto que se automatizam processos, como o processo de *shipping* e a sincronização com as lojas de *ecommerce*. Tudo isto permitirá agilizar e simplificar todo o processo desde a gestão das encomendas e dos *customers* por parte das marcas até à própria comodidade dos *customers*, sendo a *HUUB* o centro de todo este ecossistema.

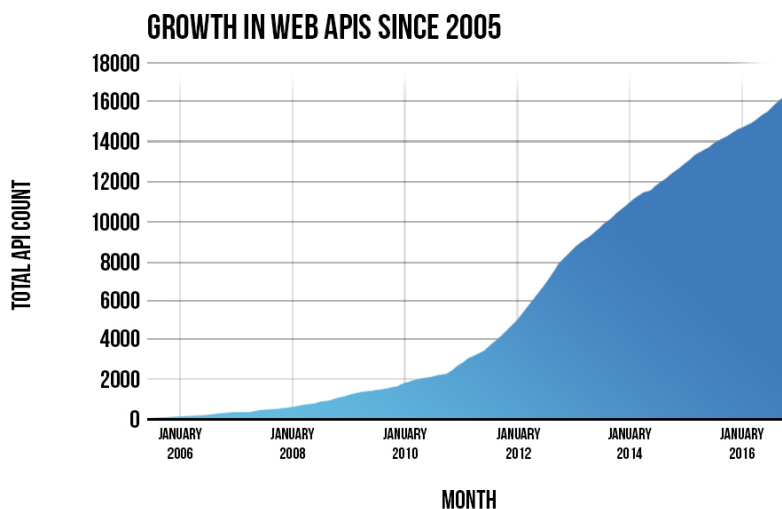
 ProgrammableWeb

Figura 1.4: Crescimento do número de APIs públicas disponibilizadas ¹.

Com o desenvolvimento desta aplicação, a *HUUB* terá um ganho operacional enorme visto que vai automatizar tarefas que demoram bastante tempo, como a criação de ordens de *shipment* nas diversas transportadoras e criação da *label* correspondente, tarefa que atualmente é feita através da inserção manual de todos os dados. Esta aplicação também permitirá, no futuro, a automatização de outros processos e tarefas que permitam que a empresa cresça a nível tecnológico.

1.3 Objetivos

Esta dissertação tem como fio condutor o desenvolvimento de uma API usando a arquitetura *Representational State Transfer (REST)* que consiga interagir com o ERP da *HUUB* servindo de camada de abstração para integrar com diversas APIs de terceiros. No início do período de estágio, foram identificados, em conjunto com a empresa, os objetivos que guiaram todo o desenvolvimento sendo eles divididos em três grandes partes:

Automatização do processo de *shipping*, através da integração com as principais plataformas de transporte. Aqui estão incluídas transportadoras como a *UPS* e *TNT* ou um serviço já existente de integração, como o *Shiphawk* ou o *EasyPost*;

Integração com as lojas de *ecommerce*, sendo importante assegurar a integração com as plataformas mais usadas pelos *HUUB clients*, como *Woocommerce* e *Shopify*, garantindo um

¹Imagem retirada de: <https://www.programmableweb.com>.

código limpo e com abstração que permita a inclusão de novas lojas de *ecommerce* com relativa facilidade;

Notificação de intervenientes, através do desenvolvimento de um sistema de notificação via *email* e/ou sms automático, consoante certas mudanças na base de dados como encomendas expedidas ou *tracking* de encomendas.

Foi estabelecido como objetivo secundário a utilização de um mecanismo de *caching* para melhorar a performance.

Estes objetivos são objetivos de alto nível estando, mais à frente, definidos os requisitos do projeto.

1.4 Estrutura do documento

Este documento está dividido em cinco partes:

Capítulo 1: Introdução, onde é feito um panorama do projeto, da *HUUB* e de como vai funcionar o desenvolvimento;

Capítulo 2: Estado da arte, onde é referido o que já existe em termos tecnológicos semelhante ao que vai ser desenvolvido;

Capítulo 3: Caracterização do problema, onde será abordado qual o problema que se pretende resolver, os requisitos levantados, bem como uma caracterização um pouco mais detalhada de cada uma das partes em que o projeto se desenrola;

Capítulo 4: Solução proposta, onde estará evidenciada a solução proposta para resolver o problema de cada uma dessas três partes bem como uma proposta de solução a nível tecnológico;

Capítulo 5: Implementação, onde será descrito como foi solucionado o problema a nível tecnológico, bem como detalhe do desenvolvimento de cada uma das três partes em que este projeto se divide;

Capítulo 6: Conclusões, onde serão abordadas as conclusões a que se chegou, que requisitos foram validados e trabalho futuro.

No final de cada capítulo estará presente uma breve conclusão desse mesmo capítulo, bem como uma curta abordagem do capítulo seguinte.

1.5 Conclusão

Este capítulo serviu para introduzir diversos conceitos que serão importantes para uma melhor compreensão desta dissertação, começando-se por enquadrar o problema no contexto atual da *HUUB*. De seguida explicou-se qual a motivação para o desenvolvimento deste projeto, abordando-se a arquitetura *SOA* e o crescimento da disponibilização e uso de *third party APIs*. Foram também referidos os objetivos acordados com a *HUUB*, finalizando com a descrição de como o presente documento está estruturado.

No próximo capítulo será evidenciado algum do estudo prévio sobre como o mundo do *ecommerce* funciona, bem como as plataformas existentes para este tipo de aplicação.

Capítulo 2

Estado da Arte

2.1 O ecossistema das lojas de *ecommerce*

O mundo do *ecommerce* tem à sua volta todo um ecossistema complexo composto por plataformas e sistemas que, interagindo, põem em funcionamento um negócio de venda de produtos ou serviços *online*.

Dentro deste mundo existem vários tipos de negócios *online*, uns com mais complexidade do que outros. No caso mais simples ter-se-ia apenas um *website* com uma montra de produtos para venda. No entanto, um negócio de *ecommerce* complexo pode ter uma arquitetura semelhante à apresentada na Figura 2.1.

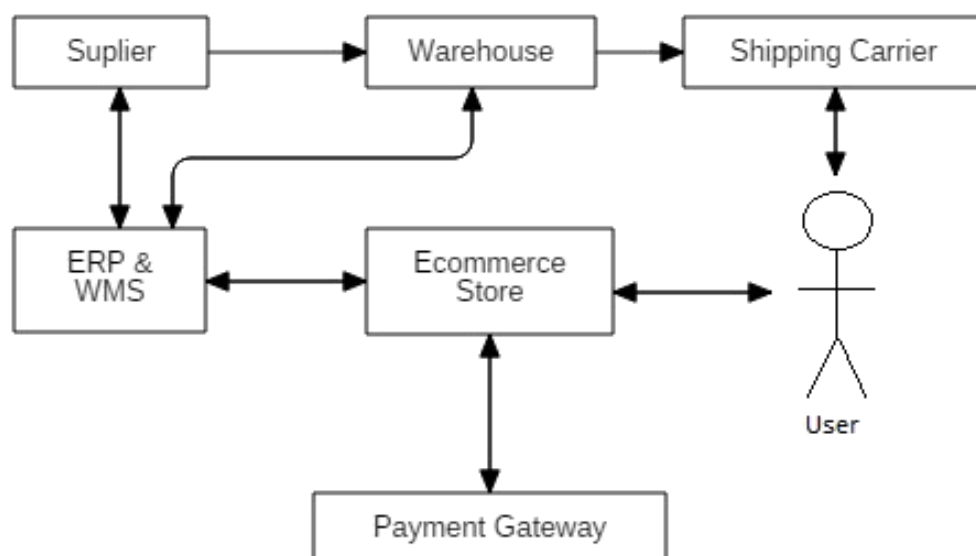


Figura 2.1: Exemplo do ecossistema de um negócio *online*.

Nesta estrutura de negócio, o utilizador pode efetuar uma compra em qualquer parte do mundo (desde que tenha acesso à Internet). O pagamento é efetuado usando um *payment gateway* onde pode ser incorporado o *Paypal*, o *Stripe*, a rede de transação *Mastercard*, etc. O ERP será notificado quando a encomenda for paga e avisa o armazém (*warehouse*) para prepará-la. Quando esta estiver pronta, uma empresa de transporte (*UPS*, *FedEx*, *TNT*, etc) recolhe a encomenda no armazém e entrega-a ao cliente final. Todo o processo de *stock* é controlado pelo *Warehouse Management System* (WMS) que contacta o fornecedor (*supplier*) para que este, por sua vez, efetue a entrega no armazém.

O exemplo aqui identificado não é de todo generalista e passível de ser aplicado em todos os casos, aliás, em bastantes casos, a proposta de valor está mesmo no próprio desenho de todo este ecossistema e na maneira como cada ator interage entre si.

Todo este sistema acresce em complexidade quando existe mais do que um canal de vendas, como por exemplo, *ecommerce* e venda por grosso (*wholesale*).

Sendo uma área que gera muito dinheiro, existem diversas empresas que tentam aglomerar alguns destes intervenientes num só, tentando ganhar espaço no mercado. Exemplo disso são as lojas de *ecommerce* como o *Woocommerce*, *Shopify* e *Magento* que disponibilizam uma loja *online* fácil de montar, incorporando já os métodos de pagamento e mesmo a *Amazon* que já controla grande parte destes intervenientes.

A *HUUB* pretende então entrar em cena, possuindo um armazém próprio, um ERP para gerir o negócio e um WMS para a gestão do *warehouse*, querendo providenciar integração com as transportadoras (*shipping carriers*) e com as lojas *online*.

2.2 Mashup de *third party* API

Segundo a Mozilla Developer Network (MDND), *third party* API são "*APIs provided by third parties — generally companies such as Facebook, Twitter, or Google — to allow you to access their functionality [...] and use it*"[8], isto é, permite a uma certa aplicação ter acesso a informação de terceiros sem que precise de saber detalhes sobre a sua implementação.

Em termos genéricos, uma aplicação faz um pedido *Hyper Text Transfer Protocol* (HTTP) ou *Hyper Text Transfer Protocol Secure* (HTTPS) a uma *third party* API de modo a obter informação ou prestação de um serviço. A este pedido, a aplicação irá receber uma resposta que poderá vir em formato *eXtensible Markup Language* (XML), uma *markup language* cujo principal intuito é dar significado semântico a texto em forma de árvore, ou *JavaScript Object Notation* (JSON), que tem como principal propósito representar estruturas de dados [9].

Estes pedidos podem ser efetuados com base no protocolo *Simple Object Access Protocol* (SOAP) ou na arquitetura *REST*. O primeiro, embora mais seguro, apresenta o inconveniente de apenas

permitir que os dados sejam enviados em XML, já o segundo, embora não incorpore mecanismos de autenticação é muito mais simples de usar, tendo sido primeiramente enunciado por Roy Thomas Fielding na sua tese de doutoramento intitulada "*Architectural Styles and the Design of Network-based Software Architectures*"[10]. Nessa Dissertação, Roy T. Fielding introduz este tipo de serviço como:

- sendo baseado numa arquitetura do tipo cliente/servidor;
- sem estado, o que aumenta a confiabilidade e escalabilidade do sistema;
- sem restrições a nível da *cache*, eliminando interações e aumentando a eficiência, escalabilidade e *performance*;
- uma interface uniforme, usando para isso os comandos já definidos pelo protocolo HTTP;
- um sistema baseado em camadas.

Este tipo de arquitetura pretende aliar os métodos HTTP para mapear operações Create, Read, Update and Delete (CRUD), isto é, fornecer recursos aos utilizadores que lhes permitam efetuar pedidos usando as funções HTTP: *Get*, *Put*, *Post* e *Delete* [11].

Como se disse no capítulo anterior, uma *mashup* de APIs é uma aplicação *web* que usa diferentes serviços de diferentes fontes, combinando-os num só serviço.

Este é um termo relativamente recente e existem já empresas que oferecem este tipo de serviços. Um exemplo deste tipo de serviço é a *Stormpath*, empresa criada em 2011 que providencia uma camada de abstração das APIs fornecidas por multinacionais no ramo das redes sociais de modo a que os seus clientes possam, usando uma única API, gerir a autenticação pelo *Facebook*, *Google* e até mesmo pelo *Github*, por exemplo. Tudo isto significa que o cliente não precisará de desenvolver código para as mais diversas plataformas, precisando-se apenas de preocupar com a plataforma que as integra, neste caso o *Stormpath*, poupando tempo e trabalho.

Outro exemplo desse tipo de integrações é o disponibilizado pela *Samepage* [12], cujo foco é o desenvolvimento de uma *mashup* que integra APIs nas mais diversas áreas, centralizando assim toda a informação. Esta plataforma integra então na áreas de *cloud storage* (*Drive*, *Dropbox*, etc), visualização de vídeos (*Vimeo*, *Youtube*, etc), aplicações de produtividade (*Slack*, *Zapier*, etc), entre outras. Isto é conseguido através da API fornecida pela *Cloudrail* [13], outra empresa focada na integração de *third party* APIs, permitindo integrar todas estas plataformas e ainda outras como de email (*Mailjet*, *Sendgrid*, etc), pagamento online (*Paypal*, *Stripe*), de mapas (*Google Places*, *Foursquare*, *Yelp*) e também SMS (*Twilio*, *Nexmo*). Este serviço providencia um *Software Development Kit* (SDK) para diversas linguagens para um desenvolvimento mais rápido.

Outro serviço que faz *mashup* de APIs é o *Tapiriik* [14]. Este serviço é totalmente *open source* e desenvolvido em *Python* com a *framework* Django e é destinado a desportistas que possuam conta em diversas aplicações designadas de redes sociais para atletas como o *Strava*, *Runkeeper*

ou *Garmin*. Ao contrário das restantes, este serviço não providencia uma *API*, sendo que o valor que dá ao utilizador é o facto de sincronizar automaticamente com todas as suas contas nessas aplicações desportistas. Assim, quando uma pessoa vai correr usando a aplicação *Strava*, os dados dessa corrida serão sincronizados com o *Runkeeper* e também com a *Dropbox*, através de um ficheiro com a extensão *.fit*.

2.3 Plataformas existentes para integração com *carriers*

De modo a integrar os sistemas, é importante analisar as *APIs* fornecidas pelos *carriers* mais usados pela *HUUB* como os *CTT*, *UPS*, *TNT*, *DHL* e *FedEx*.

Aqui é importante garantir que: se possa obter *rates* de transporte (custo do transporte) com os diversos *carriers* e seus serviços; se consiga comprar *label* de transporte para que a transportadora possa fazer levantar a encomenda das instalações da *HUUB*; e fazer o *tracking* da encomenda.

Depois de efetuada a pesquisa e análise, chegou-se à conclusão que a grande maioria destas transportadoras não fornece *API REST* pública. No entanto, existem opções que podem ser estudadas que oferecem uma *API* que permite esse serviço de integração com transportadoras. A utilização de um serviço deste tipo é preferível pois o tempo de desenvolvimento é muito mais curto. Isso deve-se ao facto do número de transportadoras é considerável.

De seguida, serão apresentadas serviços *mashup* para integração com transportadoras que consigam satisfazer uma ou mais fases do processo de *shipping*.

TrackingMore

O *TrackingMore* [15] é um serviço *freemium* que permite efetuar o *tracking* de encomendas de mais de 380 *carriers*, entre elas os *CTT*, a *Fedex*, a *TNT*, a *DHL* e a *UPS*.

Dispõe de uma *API REST*, com exemplos para várias linguagens de programação e tem integrado, opcionalmente, um sistema de notificação por email. Caso o cliente pretenda, também pode incluir um serviço de notificações por SMS, no entanto tem um custo acrescido por cada SMS, dependente do país em questão.

O preço deste serviço varia consoante o número máximo de encomendas por cada mês ou pode ser comprado em pacotes. Na primeira opção, para um volume a rondar as 700 encomendas por mês¹, teria um custo mensal de \$16. Já a segunda opção, que seria mais vantajosa, teria um preço de \$40 para 10 000 encomendas, ou seja, \$0.004 por encomenda.

¹Número estimado de encomendas que a *HUUB* processa por mês.

Aftership

O *Aftership* [16] é um serviço em tudo igual ao *TrackingMore*, permitindo efetuar *tracking* de encomendas em 382 *carriers*, entre elas os *CTT*, a *Fedex*, a *DHL* e a *UPS*.

Também dispõe de uma *API REST*, com exemplos ilustrativos, bem como um *SDK* para *Hypertext Preprocessor* (PHP) para uma maior facilidade de acesso aos *endpoints* da *API*, disponível em formato *open-source* [17].

Esta plataforma permite também um sistema de alertas por *email* ilimitado bem como alertas por *sms*, também com custo acrescido. O custo desta plataforma, para um volume de encomendas de aproximadamente 700 por mês é maior, chegando aos \$40 por mês.

Postmen

O *Postmen* [18] é um serviço gratuito, pelo menos por agora, que permite integração com 46 transportadoras, entre elas a *Fedex*, *UPS* e *DHL*, não tendo, contudo, integração com os *CTT*.

Fornece também uma *API* para criar encomendas, seus manifestos e *labels*, e também dá a possibilidade de cancelamento das mesmas se for necessário. Tem, além disso, recursos para calcular *rates* de transporte, permitindo assim, verificar qual a transportadora que oferece o serviço mais barato.

No entanto, não possui a capacidade de fazer *tracking* das encomendas.

Circle

Circle [19] é uma *startup* cujo objetivo vai ao encontro das necessidades. Aqui, os serviços disponibilizados gratuitamente são o processamento de encomendas, criação de *labels* e *tracking*.

Das transportadoras usadas pela *HUUB*, este serviço permite efetuar encomendas apenas com a *UPS* e a *Fedex*, no entanto está em constante crescimento.

Dispõe de uma *API*, que permite validar moradas, gerir *orders*, criar ou cancelar *labels* e também efetuar *tracking* de cada encomenda.

Shippo

Shippo [20] é uma solução que se apresenta bem documentada e cujo principal objetivo é simplificar todo o processo de *shipping*.

Com recurso à sua *API*, é-se capaz de criar encomendas, manifestos e *labels*, fazer *tracking* das mesmas e validar endereços de morada. Tem ainda opções para notificação consoante mudanças nos estados.

Este serviço permite efetuar encomendas globalmente usando a *UPS*, *Fedex* e *DHL*.

A nível de custos, este serviço é gratuito para chamadas à *API* (validar endereços, fazer *tracking*, consultar *rates* das diversas transportadoras, etc) e tem um custo de \$0.05 por *label* criada, acrescida ao valor da encomenda.

A *Shippo* serve então de intermediário, no entanto, toda e qualquer responsabilidade (fraudes, etc) fica do lado da *HUUB* e a comunicação é feita em exclusivo com a empresa, cabendo à *HUUB* transmitir aos seus clientes a informação que achar necessária.

Shiphawk

Shiphawk [21] é uma empresa que vende software de otimização e *tracking* de encomendas cujo grande objetivo é a automatização de todo o processo de transporte.

Esta plataforma é responsável pela escolha da melhor transportadora no que ao preço diz respeito, seguindo-se todo um processo de criação de *labels*, empacotamento, transporte e *tracking* das encomendas, sendo potenciada pelo seu *dashboard* que contém toda essa informação agrupada.

É uma plataforma que possui também integração com algumas lojas de *ecommerce* usadas, como o *Magento* e o *Shopify*, e possui uma *API* que entre as diversas funcionalidades estão a criação de *orders* e o *tracking* das mesmas.

A utilização da *API* do *Shiphawk* é feita de forma gratuita no entanto, para aceder à *sandbox* e a funcionalidades de otimização, é necessário ter conta *premium* o que acarreta custos não discriminados.

Easypost

A *Easypost* [22] é uma solução focada também em simplificar todo o processo de *shipping*, desde a sua criação até à receção, sendo especialmente focada no desenvolvimento de uma *API* que ajude outros programadores.

É então possível automatizar a criação de novas encomendas e suas *labels*, verificar moradas de todo o mundo, comparar *rates* e também fazer seguros às próprias encomendas.

De momento, a integração é possível de ser feita com quase 80 *Carriers*, entre eles a *Fedex*, *UPS*, *TNT* e *DHL* contudo, a integração com os *CTT* não está disponível.

Este serviço tem também um custo associado de \$0.03 por encomenda (incluindo todo o processo da mesma) ou de \$0.01 se o objetivo for fazer apenas *tracking* de encomendas que não foram processadas pela *EasyPost*.

Análise comparativa

A partir da Tabela 2.1 é possível ver uma análise comparativa das plataformas descritas. De notar que a *FedEx* não foi mencionada. Isto deve-se ao facto de ser impossível integrar com essa transportadora pois a Rangel, empresa subsidiária da *Fedex* em Portugal, não tem acesso ao contrato estabelecido entre a *FedEx* e a *HUUB*. Desta forma, os preços que seriam obtidos não eram os acordados entre as duas empresas.

Tabela 2.1: Análise comparativa das diferentes opções para *mashup* das plataformas de *shipping*.

	Carriers Suportados	Rates	Shipping	Tracking	Custo
TrackingMore	Todos	Não	Não	Sim	\$0.004/tracking \$16 ²
Aftership	Todos	Não	Não	Sim	\$40 ²
Postmen	Todas menos <i>CTT</i>	Sim	Sim	Não	Grátis ³
Circle	<i>UPS</i>	Sim	Sim	S/I ⁴	Grátis
Shippo	<i>UPS</i> <i>DHL</i>	Sim	Sim	Sim	\$0.05/label ⁵
Shiphawk	N/A ⁶	Sim	Sim	S/I ⁴	Grátis ⁷
Easypost	Todas menos <i>CTT</i>	Sim	Sim	Sim	\$0.01/tracking ou \$0.03/shipping

2.4 Plataformas existentes para integração com as lojas de *ecommerce*

Para que seja possível integrar com lojas de *ecommerce*, uma solução passa por se recorrer a uma plataforma *mashup* que já integre com as diversas lojas *online*, de modo a facilitar o desenvolvi-

²Preço de referência para 700 encomendas/mês - varia consoante este número.

³Em promoção para angariação de clientes. Pode, a qualquer momento, deixar de ser gratuito. Sem indicação de qual será o preço.

⁴Sem Informação.

⁵Gratuito fazer GET's à API. Envio de encomendas e *labeling* pagos.

⁶Shiphawk decide qual *carrier* usa (escolhe o mais barato).

⁷Gratuito aceder à API. Possibilidade de pagar para outras funcionalidades.

mento. Outra opção poderá ser desenvolver um sistema agnóstico que integre diretamente com essas mesmas lojas. Serão explorados, de seguida, diferentes alternativas *mashup*, bem como as APIs de algumas das principais lojas de *ecommerce* caso se pretenda implementar uma *mashup* própria. Aqui, é importante garantir que se possa ter acesso à informação sobre produtos, *orders* e *customers* sendo de valor a existência de *webhooks*, que são pedidos HTTP disparados pelas lojas *online*, em tempo real, mediante certos *triggers*, como o pagamento ou cancelamento de uma *order*.

2.4.1 *Mashup* de APIs de *ecommerce*

Ecomdash

O *Ecomdash* [23] é uma plataforma *mashup* que integra diversas lojas *online* e *marketplaces* como *Shopify*, *Magento*, *Woocommerce*, *Amazon*, *Ebay*, entre outros, bem como algumas plataformas de *shipping*.

Apresenta-se como um serviço bastante completo cujo objetivo é automatizar todo o processo de vendas *online* por diversos canais, tendo como proposta de valor o controlo dado ao utilizador sobre toda a cadeia de abastecimento e a reunião de toda a informação num único sítio, o que permite uma gestão mais eficaz.

Disponibilizam uma API para integração com outros sistemas, no entanto, não têm *webhooks*.

Tem também um custo associado que varia consoante o número de encomendas mensais. Considerando 700 encomendas mensais, o custo seria de \$125 por mês.

ChannelApe

O *ChannelApe* [24] é uma plataforma *mashup* muito semelhante ao *Ecomdash*, no entanto, é mais limitado a nível de integrações, sendo que as principais lojas de *ecommerce* que integra são o *Shopify* e o *Magento*.

É um serviço em constante evolução e que já tem prevista a integração com uma série de novas plataformas. Oferece também uma API e, da mesma forma que o anterior, não tem *webhooks*.

O custo associado é de \$199 por mês.

Api2Cart

O *Api2Cart* [25] é a última plataforma *mashup* estudada e é um pouco diferente das duas anteriores pois foca-se inteiramente no processo de *ecommerce*. Integra com praticamente todas as grandes

webshops, possuindo um quadro onde disponibiliza os métodos passíveis de serem acedidos por cada loja.

No entanto, esta plataforma tem um custo mínimo de \$500 por mês, limitando ainda o número de pedidos à API feitos em simultâneo e o número de diferentes lojas associadas. Também não está contemplada a hipótese de disparo de *webhooks*.

Análise comparativa

A partir da Tabela 2.2 é possível ver uma análise comparativa das plataformas descritas.

Tabela 2.2: Análise comparativa das diferentes opções para *mashup* de lojas online de *ecommerce*.

	API REST	Lojas online que integra	Webhooks	Mensalidade
<i>Ecomdash</i>	Sim	<i>Woocommerce</i>	Não	\$125
		<i>Shopify</i>		
		<i>Magento</i>		
<i>ChannelApe</i>	Sim	<i>Shopify</i>	Não	\$199
		<i>Magento</i>		
		<i>Woocommerce</i>		
<i>Api2Cart</i>	Sim	<i>Shopify</i>	Não	\$500
		<i>Magento</i>		
		<i>Prestashop</i>		

2.4.2 Lojas de *ecommerce* que disponibilizam API

Woocommerce

Esta plataforma *open-source* nasceu em 2008 e desde aí faz parte dos *plugins* para o *wordpress* [26].

É uma empresa internacional com colaboradores de 19 países e cobre cerca de 39% das lojas de comércio online.

Analisando a documentação da API fornecida, verifica-se que se consegue extrair muita informação como *orders*, *customers*, produtos, moradas, etc. O esquema de dados do *woocommerce* é um pouco complexo, existindo distinção entre produtos simples e compostos, bem como a existência de variantes de produtos.

Esta plataforma também disponibiliza *webhooks* mediante alteração nas informações de *orders*, produtos e *customers*. No entanto, não tem incorporada uma opção para realização do processo de *fulfilment*.

Shopify

O *Shopify* [27] é um dos líderes neste mercado do *ecommerce*, tendo começado em 2006 como uma simples loja *online* de venda de equipamento de *snowboard*.

Depois de averiguar a documentação da *API* fornecida, verifica-se que o *Shopify* é outra loja de *ecommerce* da qual também se pode extrair muita informação. Esta loja possui a capacidade de fazer *fulfilment* parcial e total e também dá a possibilidade de configurar *webhooks*.

O *Shopify* é uma plataforma extremamente completa e bem organizada no seu esquema de dados, daí que seja também uma plataforma paga.

Magento

O *Magento* [28], fundado em 2008, anuncia-se como a plataforma número um de comércio a nível mundial.

Observando-se o *Developer Guide*, verifica-se que, tal como o *Shopify* e o *WooCommerce*, o *Magento* providencia uma *API* completa. Também oferece a possibilidade de configurar *webhooks* e de fazer *fulfilment*.

Esta plataforma tem duas versões, cada qual com uma *API* própria e, para cada versão, existe a distinção entre *API REST* e *SOAP*, isto é, apresenta no total quatro versões. Cada cliente, tendo uma loja, terá que escolher uma destas versões, não sendo as mesmas compatíveis umas com as outras.

Prestashop

O *Prestashop* é também uma opção válida para muitas marcas que pretendem vender os seus produtos na *web*. Destaca-se por ser totalmente gratuito e *open-source* tendo, neste momento, mais de 300 contribuidores.

Examinando a documentação, facilmente se percebe que esta não está tão bem documentada como as anteriores. No entanto, também disponibiliza uma série de *endpoints* que permitem a obtenção de informação, bem como a possibilidade de disparo de *webhooks*. Contudo, tal como o *WooCommerce*, não tem opção para fazer *fulfilment*.

Tictail

O *Tictail* é uma loja recente, fundada em 2013, que funciona como uma loja comunitária onde os vendedores disponibilizam os seus produtos para venda, sendo esses produtos vendidos num espaço comum, ou, da maneira convencional, em que o cliente cria a sua própria loja *online*.

Examinando a documentação, facilmente se percebe que se está a falar de uma loja recente, no entanto, cativa *developers* pela simplicidade na maneira como está organizado o seu esquema de dados, providenciando *endpoints* para se obter a informação mais importante para o comércio online.

Tem também incluído o conceito de *fulfilment* no entanto não existe *fulfilment* parcial de *orders*. O disparo de *webhooks* é algo muito embrionário e para se ter acesso à funcionalidade é necessário contactar diretamente o *Tictail*.

Análise comparativa

A partir da Tabela 2.3 é possível ver uma análise comparativa das plataformas descritas.

Tabela 2.3: Análise comparativa das APIs das diferentes lojas *online* de *ecommerce*.

	<i>Open source</i>	<i>Hosting service</i>	<i>API REST</i>	<i>Product info</i>	<i>Customers info</i>	<i>Stock update</i>	<i>Fulfilment</i>	<i>Webhooks</i>
Woocommerce	Sim	<i>Self-hosted</i>	Sim	Sim	Sim	Sim	Não	Sim
Shopify	Não	<i>Cloud</i>	Sim	Sim	Sim	Sim	Sim	Sim
Magento	Sim	<i>Self-hosted</i>	Sim	Sim	Sim	Sim	Sim	Sim
Prestashop	Sim	<i>Self-hosted</i>	Sim	Sim	Sim	Sim	Não	Sim
Tictail	Não	<i>Cloud</i>	Sim	Sim	Sim	Sim	Sim ⁸	Sim ⁹

2.5 Plataformas existentes para notificação por email**MailChimp**

O *Mailchimp* [29] é a plataforma líder a nível mundial em *marketing por email*, tendo já ao seu dispor, bastantes integrações com serviços populares, como é o caso das lojas de *ecommerce* e do *Facebook*. Tem também mecanismos para automatizar o envio e também um bom sistema de *analytics*.

⁸Apenas existe o conceito de *fulfilment* total, não existe *fulfilment* parcial.

⁹Estado muito embrionário.

O *MailChimp* oferece três modalidades: uma gratuita, com 12 000 *emails* por mês e possibilidade de ter até 2 000 *subscribers*; uma versão para *startups*, com a possibilidade de ter até 500 *subscribers* e *emails* ilimitados bem como a possibilidade de automatizar o envio de *emails*; e uma última versão Pro que possui *features* indicadas para profissionais de *marketing*.

Para o envio de *emails*, o *MailChimp* disponibiliza uma *API* que permite criar *templates* e um *workflow* da automatização. Tem também integração com o *Shopify*, o *Magento* e o *Woocommerce*.

E-Goi

A *E-Goi* [30] é uma empresa portuguesa nascida em 2006 e especializada na automatização do marketing digital, usando *email*, sms, voz e redes sociais.

Este serviço dispõe de três modalidades: uma de mensagens pré-pagas, com *emails* a 0.0024€; uma de envios mensais, tendo o modelo mais barato um custo de 62.50€, dando para 50 000 *emails* e 500 sms; e uma versão ilimitada, com um custo a partir de 18.00€, dando para enviar *emails* ilimitados mas apenas 50 sms por mês.

A *E-Goi* permite integração com *Magento* e *Shopify* e disponibiliza uma *API* para integração com sistemas externos.

MailGun

O *MailGun* [31] é mais uma plataforma para envio de *emails* e é usado por grandes marcas como o *Github*, o *Slack* e o *Heroku*.

Esta plataforma permite, de forma gratuita o envio de 10 000 *emails* por mês e é orientada para *email* um para um, um requisito imposto pela *HUUB*. Este serviço tem ainda a vantagem de ser suportado *out of the box* por *Laravel*, a *framework* PHP que vai ser usada.

A sua documentação é excelente, isto porque a *API* foi construída como um serviço e não para um serviço.

Disponibiliza também um bom *dashboard* que permite analisar os estados dos emails enviados bem como outras métricas de interesse.

SparkPost

A *SparkPost* [32] é outra das plataformas recomendadas pelo *Laravel* e é também usada por empresas conhecidas como o *LinkedIn*, o *Twitter* e o *Paypal*.

Este serviço permite o envio de 100 000 *emails* mensais, suportada pela sua *Transmission API* que possibilita, de uma maneira relativamente simples, enviar *emails* direcionados para o cliente ou consumidor em questão, possuindo também um *dashboard* bastante completo.

Análise comparativa

A partir da Tabela 2.4 é possível ver uma análise comparativa das plataformas descritas.

Tabela 2.4: Análise comparativa das diferentes opções para envio de *emails*.

	Features	Out of the Box Integration com Laravel	Preço (5 000 emails/mês)
MailChimp	Integração com lojas <i>ecom</i>	Não	\$199 ¹⁰
E-Goi	Integração com lojas <i>ecom</i> e <i>facebook</i> Possibilidade de usar SMS	Não	€ 18 (emails ilimitados)
Mailgun	API muito boa e bem documentada <i>Batch Sending</i> <i>Analytics</i>	Sim	Grátis (até 10 000)
Sparkpost	<i>Reliability</i> <i>Analytics</i>	Sim	Grátis (até 100 000)

2.6 Conclusão

Durante o presente capítulo foi abordado o ecossistema das lojas de *ecommerce* e a simbiose entre as diversas entidades que interagem entre si. Foi também referido e explicado o conceito de *mashup* de *third party APIs*.

De seguida, foi realizada uma análise das diversas plataformas de *mashup* de *APIs* para integração com os *carriers*, concluindo com um balanço comparativo das diferentes soluções.

Da mesma maneira foi realizada uma investigação das plataformas de *mashup* de *APIs* para integração com as lojas de *ecommerce*, seguindo-se uma análise comparativa das diversas soluções. Seguiu-se uma análise das *APIs* das próprias lojas *online* finalizando da mesma forma, com uma tabela comparativa.

Por último e com o mesmo formato, foram averiguadas as diferentes plataformas de envio de *email*, havendo também uma comparação de soluções no final.

¹⁰Caso se pretenda funcionalidade *email* um para um.

No próximo capítulo será exposto um caso específico sobre a *HUUB* para contextualização, os problemas que se pretendem resolver e os requisitos do projeto.

Capítulo 3

Caracterização do Problema

Neste capítulo será apresentado o problema identificado pela *HUUB*, assim como uma exposição detalhada de como o processo era desenrolado e qual a relevância do projeto para os diversos intervenientes.

Este capítulo começará com a exposição de um caso real da *HUUB* para que se possa analisar alguns dos processos, sendo, de seguida, dividido em três grandes partes. A primeira, automatização do processo de *shipping*, retrata a integração do ERP da *HUUB*, o *Spoke*, com as transportadoras usadas e tem como grande objetivo agilizar todo o processo de envio de encomendas.

A segunda parte, integração com lojas de *ecommerce*, tem como propósito estimular a troca de informação entre as lojas *online* dos clientes e o *Spoke* para que exista sincronização da informação entre as duas entidades.

A última parte, notificação dos intervenientes, é a mais curta e estará presente nas duas anteriores. Servirá para enviar, automaticamente, notificações e alertas para os participantes do processo logístico e da cadeia de abastecimento, permitindo que todos estejam informados sobre acontecimentos importantes e de possíveis exceções que ocorram, atuando mais rapidamente sobre essas exceções.

3.1 O processo da *HUUB*

O grande objetivo da *HUUB*, como referido em capítulos anteriores, passa por ligar toda a cadeia de abastecimento, criando um ecossistema que conecte todos estes intervenientes. Esta tarefa não é fácil e está longe de ser concluída. Atualmente, e porque a equipa de desenvolvimento é pequena (apenas três pessoas), grande parte dos processos ainda são manuais, estando a iniciar-se uma política de automatização desses processos, enquanto que, paralelamente, são desenvolvidas novas *features* que acrescentem ainda mais valor à empresa.

Existe um *HUUB client* que vende roupa de criança e tem dois canais de venda, *ecommerce*, com recurso à própria loja *online* (*Woocommerce*), e *wholesale*, cujo *flow* do processo de lançamento de uma nova *season* está evidenciado na Figura 3.1. Este *HUUB client* possui uma série de comerciantes e retalhistas, *customers*, que vendem os seus produtos. Assim, antes do lançamento de uma nova *season*, o *HUUB client* faz acordos com esses retalhistas sobre a quantidade de produtos que serão enviados para cada um.

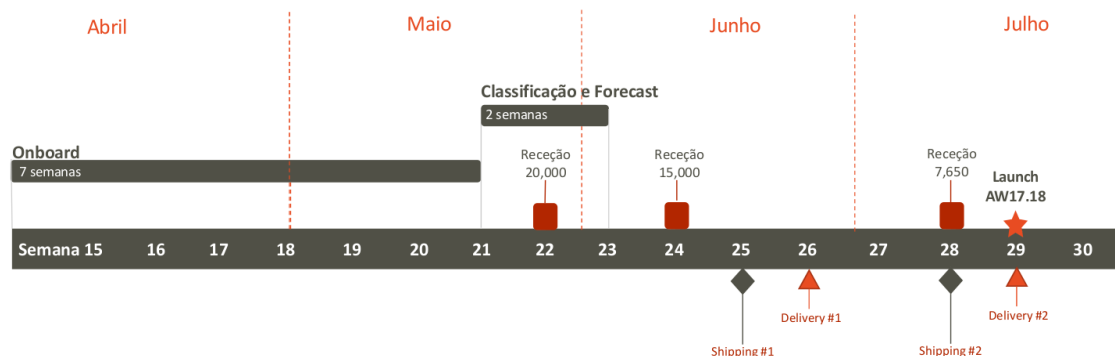


Figura 3.1: Estratégia de venda e distribuição dos produtos da nova *season* de um cliente da HUUB.

Também é possível ver que o início do processo da nova *season* está marcado para março com o *onboard* da marca que demorará cerca de 7 semanas. Este *onboard* é realizado pela equipa de AMs recorrendo a ficheiros partilhados e incorpora as seguintes ações:

- Sincronização dos produtos do *HUUB client* para o sistema da *HUUB*;
- Sincronização da informação dos diferentes *customers*;
- Preparação das *SOs* a enviar para os *customers*.

De seguida, é realizada a categorização dos produtos¹ e o *forecast* para previsão do custo de todos os envios que serão realizados para cada *customer*. Este processo demorará cerca de 2 semanas, no entanto, num futuro próximo, será praticamente instantâneo, pois a equipa de BI encontra-se a desenvolver um algoritmo de classificação e otimização de caixas e, todo o processo de *forecast* será automatizado no decorrer deste projeto de dissertação.

Entretanto, o armazém dará entrada de 2 das 3 *receptions*² dos produtos do *HUUB client*. Na terceira semana de junho, serão enviadas as *SO* para os principais *customers* que as receberão passada uma semana. Para a realização deste envio é necessário proceder à compra da *label* de transporte, operação que, atualmente, demora entre 15 a 20 minutos e que se pretende automatizar, também no contexto desta dissertação. Cerca de 1 mês depois, serão enviadas as restantes *SOs*,

¹ Este processo envolve também a estimativa do número de *packs* a enviar para cada *customer*. É apenas uma estimativa porque a *HUUB* ainda não recebeu nenhum produto, sendo que o volume de cada produto é apenas estimado.

² Uma *PO* pode ser dividida em várias *receptions*.

bem como recebida a última *reception*. Na terceira semana de julho, todos os *HUUB clients* já terão em *stock* os produtos desse *HUUB client* e será realizado o *launch* da nova *season*, isto é, os produtos da nova *season* começarão a ser vendidos na loja *online* do próprio cliente da *HUUB* e nos diversos retalhistas.

3.2 Automatização do processo de *shipping*

Nesta parte, o foco será a automatização do processo de *shipping*, *end to end*. Este processo realiza-se em duas fases distintas, o envio da *order*, que envolve *forecast* e criação da *label*, e o *tracking* da encomenda.

O processo de *forecast* é iniciado cerca de 1 mês antes do envio da *order* e trata-se de uma estimativa do custo de transporte (consulta de *rates*) seguida por uma negociação com o *HUUB client*. Este processo é feito com recurso a um ficheiro *xls* que contém uma tabela fixa dos preços negociados com os diferentes *carriers*, algo que se pretende deixar cair para que esta previsão de custos seja feita de forma automática e precisa, entrando em direto contacto com os *carriers*, obtendo-se assim, informação em *real-time*.

Já o processo de criação da *label* finaliza a fase de *shipping* e é despoletado depois dos itens da *Sales Order* (SO) serem picados, embalados e pesados. Nesta fase, decorre uma nova consulta de *rates* dos diversos *carriers*, seguindo-se a seleção do serviço, com base numa negociação prévia e registo do transporte no sistema da transportadora. Esta fase tem como *output* a *label* de transporte gerada pela transportadora, que é obrigatória existir e diferente entre *carriers* e a documentação necessária. Essa *label* é colada na embalagem, estando esta pronta para que seja feito o *pickup* pelo *carrier*. Como foi referido, esta *label* demora entre 15 a 20 minutos a ser feita manualmente. Isto deve-se ao facto de ser necessário introduzir manualmente toda a informação do *customer* no *website* da transportadora, que normalmente tem bastante burocracia para lidar com *tax and duties* e garantir que está tudo correto através de várias validações. Estando a informação toda no sistema da *HUUB*, é possível automatizar este processo, trazendo, assim, vantagens a nível operacional.

Após a encomenda sair das instalações da *HUUB*, o processo de *shipping* entra na terceira e última fase, o *tracking* da mesma. Neste momento, esta fase encontra-se do lado das transportadoras, no entanto, quer seja para trazer os *HUUB clients* e *customers* para o ecossistema da *HUUB* como também para ter um maior controlo sobre que informação é passada e a quem é passada, é objetivo da *HUUB* ser responsável pelo *tracking* das diversas encomendas que são expedidas todos os dias bem como das notificações resultantes da mudança do estado das mesmas.

Uma automatização de parte ou da totalidade dos envios reduz os erros do processo, traz ganho operacional para a *HUUB* e para os seus *HUUB clients* nas duas primeiras fases do processo, *forecast* e *shipping*, e traz conveniência para os *customers* na terceira e última fase, *tracking*.

3.3 Integração com lojas de *ecommerce*

Com a chegada de *HUUB clients* que possuem canais de vendas *online* em lojas de *ecommerce*, surge a necessidade de automatizar processos nesta área, sendo importante integrar diversas lojas de *ecommerce*.

Atualmente, a *HUUB* tem quatro *HUUB clients* que utilizam a plataforma *Shopify* e um *HUUB client* que possui loja *online* com o *Woocommerce*. Por esse motivo, estas duas plataformas são prioridade. No entanto, e com a rápida evolução do mercado *ecommerce* é importante pensar na integração de outros serviços como *Magento*, *Tictail*, etc.

Nesta fase, é interessante:

- obter informação dos produtos;
- obter informação dos *customers*;
- receber, em tempo real, atualizações de uma *order*;
- atualizar o estado de cada *order* (processo de *fulfilment*);
- sincronizar o *stock*.

Com a automatização de parte ou da totalidade destas ações, é esperado o decréscimo de erros que até então têm sido sistemáticos, devido à transação de ficheiros *xls* e *gdocs* entre *HUUB* e *HUUB clients* para sincronização da informação, bem como devido à intervenção humana, pois existem operários constantemente a atualizar informações nas lojas *online* dos *HUUB clients*.

Outro processo que se pretende melhorar é a sincronização das *orders*. Neste momento, existe um *cron* que, de hora a hora, percorre a lista de *HUUB clients* com loja *online*, faz o pedido de todas as *orders* feitas na última hora, processa-as e guarda-as no *Spoke*. No entanto, alguns problemas surgem devido a:

- limitações de número de pedidos por segundo das *webshops*;
- existir um elevado volume de informação a ser transacionada, o que poderia originar *timeout* por parte do servidor;
- existir uma elevada carga no servidor da *HUUB*, pois é necessário processar todas as *orders* de todos os *HUUB clients* que, em caso de pico³ de vendas, congestionam bastante o servidor;
- existir um limite máximo do número de *orders* que a *API* das lojas *online* permite retornar, isto é, caso tenham sido feitas 105 *orders* na última hora, como está pré-definido que apenas se poderá buscar 100 *orders*, 5 *orders* não são sincronizadas.

³No comércio (principalmente indústria do têxtil), existem alturas de elevado pico de vendas. São elas o início de *season*, início de época de saldos e eventos populares como a *black friday*.

Esta última situação é extremamente crítica pois as lojas *online* têm que dar uma resposta a encomendas pagas num prazo máximo de três dias. Neste caso, e como já aconteceu, algumas *orders* não foram sincronizadas resultando num enorme atraso pois essas encomendas só foram enviadas quando o erro foi detetado.

Tudo isto, mantendo sempre uma camada de abstração para que, caso surja necessidade de integrar com outros serviços, o desenvolvimento seja feito independentemente dos restantes serviços integrados e com relativa facilidade.

3.4 Notificação de intervenientes

O último dos objetivos desta dissertação será o envio de notificações, por *email*, quando acontecem exceções para que se consiga colmatar eventuais falhas no sistema de forma rápida e organizada.

Um exemplo disso será o *tracking* de encomendas, em que o *customer* não vai, pro-ativamente, consultar o estado da encomenda. Para isso, é necessário uma plataforma que integre tudo isso e notifique o consumidor sobre mudanças no estado da encomenda, como por exemplo se esta já foi expedida, se está retida na alfândega, etc.

Tudo isto traria valor para o consumidor que estaria sempre a par da informação mais recente sem precisar de consultar a plataforma da transportadora ou o *Spoke*. Para que isto seja possível, foram encontrados quatro serviços, *MailChimp*, *MailGun*, *E-Goi* e *Sparkpost*, analisados no capítulo anterior.

3.5 Requisitos

Para que se possa validar todo o projeto foram definidos requisitos no início do projeto. No entanto, devido à volatilidade de uma *startup* e à tentativa da implementação de metodologias de desenvolvimento de *software* ágeis, estes requisitos foram sofrendo alterações à medida do tempo.

Estes requisitos podem ser classificados como funcionais e não funcionais. Os requisitos funcionais definem as funcionalidades do sistema e a maneira como o sistema atua, estando evidenciados na Tabela 3.1.

Tabela 3.1: Requisitos Funcionais.

A	Requisitos funcionais
1	Obtenção de <i>rates</i> de transporte em <i>real time</i>
2	Comprar <i>label</i> de transporte
3	Atualização do <i>tracking</i> de uma <i>order</i> de forma passiva
4	Atualização do <i>stock</i> da loja <i>online</i>
5	Possibilidade de fazer <i>fulfil</i> parcial ou total
6	Gerir <i>webhooks</i>
7	Processar uma <i>order</i> de <i>ecommerce</i> mal ela seja comprada
8	Notificar <i>updates</i> de <i>tracking</i>
9	Notificar <i>orders</i> mal inseridas

Os primeiros dois requisitos dizem respeito ao processo pré-envio da encomenda. Para que a automatização do processo de *shipping* seja consumada, é necessário obter as *rates* de transporte em tempo real, de modo a que os AMs não precisem de o fazer manualmente, e comprar a *label* de transporte para que a encomenda seja preparada e levantada no final do dia pela transportadora em questão. O terceiro requisito, atualização do *tracking* de uma *order* de forma passiva também faz parte do processo de *shipping*, no entanto, numa fase pós-envio. Esta é uma funcionalidade importante pois será a HUUB a tratar das notificações dos diferentes *customers*, algo que era levado a cabo pela própria transportadora, aumentando assim o tráfego do *website* e contribuindo para a internacionalização da empresa.

No que toca à integração com as lojas de *ecommerce*, existem quatro requisitos impostos. O primeiro é permitir a alteração do *stock* da loja, pois o verdadeiro *stock* está sempre do lado da HUUB. É também importante conseguir fazer *fulfilment* parcial ou total das SOs de modo a dar a conhecer ao *customer* e ao HUUB *client* que certa encomenda foi expedida. Este requisito está dependente das APIs das lojas de *ecommerce*. Devido à utilização de *webhooks*, é necessário também conseguir geri-los (operações CRUD), daí o requisito número 6. Por último, é importante dar resposta rápida às *orders* que são feitas nas lojas *online*. Desta forma, através do disparo do *webhook* pela *webshop* quando uma *order* é paga, a aplicação terá de criar tudo o que for necessário para que essa *order* possa ser preparada. É importante também garantir um mecanismo de cancelamento da *order* caso exista ordem de reembolso.

Os últimos dois requisitos dizem respeito ao sistema de alertas. Neste momento foram definidos que deverão existir alertas para notificar atualizações no *tracking* de uma encomenda quando: ela é expedida, entra num estado de exceção, sai de um estado de exceção e em *last mile*⁴.

Os requisitos não funcionais, por sua vez, definem como as funcionalidades do sistema deverão ser implementadas e estão representados na Tabela 3.2.

⁴Este estado é ativado quando a encomenda está na última estação de controlo e prestes a ser recebida pelo destinatário.

Tabela 3.2: Requisitos não funcionais.

B	Requisitos não funcionais
1	Segurança nos pedidos
2	Boa documentação da API
3	Plataforma para gestão administrativa
4	Obedecer às normas legais
5	Facilidade de integração com novos serviços

Estes requisitos, por sua vez, incorporam toda a segurança dos pedidos à API, boa documentação da mesma para que seja fácil utiliza-la, uma plataforma para gerir informação relevante como utilizadores e *roles*, obedecer às normas legais que regulam o comércio *online* e o envio de mercadoria e que se consiga garantir uma abstração no código de modo a ser fácil implementar integrações com novos serviços semelhantes.

3.6 Conclusão

Neste capítulo foram caracterizados os problemas propostos pela HUUB, sendo eles: automatização do processo de *shipping*, integração com as lojas *online* e notificação de intervenientes. Foram também evidenciados os requisitos para o projeto.

No próximo capítulo serão realçadas as soluções propostas para mitigar os problemas aqui descritos, bem como a arquitetura que se propõe implementar.

Capítulo 4

Solução proposta

Neste capítulo é apresentada a solução pensada para o problema especificado no Capítulo 3.

Numa primeira instância, o capítulo irá seguir uma estrutura semelhante à do capítulo anterior, contendo as três partes já abordadas, automatização do processo de *shipping*, integração com lojas de *ecommerce* e notificação dos intervenientes. De seguida, será abordada a solução pretendida em termos tecnológicos como arquitetura e tecnologias a usar.

4.1 Automatização do processo de *shipping*

Para a escolha do serviço a usar, o ideal será fazer uma análise às encomendas anteriores efetuadas pela *HUUB* presentes na Figura 4.1. Pode verificar-se que os *carriers* com maior número de encomendas foram os *CTT* e a *UPS*, cobrindo quase 70% dos transportes efetuados. No entanto, em termos de volume de negócio, a *UPS* encontra-se bastante destacada.

É também importante salientar o crescimento do número de encomendas feitas pela *TNT* e o recente acordo com a *DHL*, que irão ser *carriers* cada vez mais usadas pela *HUUB*, assim como o decréscimo da utilização da *DPD* e a impossibilidade de integração com *Nacex* e *Fema* devido à falta de disponibilização de recursos da parte das duas transportadoras. Assim sendo, as plataformas mais importantes para realizar integração seriam: *UPS*, *TNT*, *Fedex*, *DHL* e *CTT*.

Para que os requisitos sejam cumpridos foi pensado um conjunto de *endpoints*, presentes na Tabela 4.1, que são agnósticos e que não dependem das plataformas escolhidas.

Consultando a Tabela 2.1, é possível fazer uma análise comparativa das diferentes plataformas enunciadas no Capítulo 2.3.

Da tabela, é possível concluir que a melhor alternativa, para o imediato, seria o uso do *Postmen* integrado com o *TrackingMore*, pois permitiria todas as funcionalidades para quatro das grandes transportadoras que a *HUUB* usa, a um preço acessível.

	name character varying(512)	numero numeric	custo numeric
1	DHL	0.22	5694.04
2	UPS	35.47	49052.89
3	TNTExpress	2.60	4908.14
4	FedEx	19.27	17206.80
5	CTT Correios	33.59	2939.24
6	Fema	3.07	190.82
7	DPD	4.48	1812.11
8	Nacex	1.28	370.26

Figura 4.1: Distribuição do número de envios e custo total feitos através de cada *carrier* ¹.

Esta opção no entanto, não é a ideal para o futuro pois poderá acarretar custos imprevisíveis uma vez que o serviço do *Postmen* terá um custo indefinido. Nesse caso, o uso da plataforma *Shippo* ou *EasyPost* parecem ser as que melhor vão de encontro às necessidades, havendo preferência pela *EasyPost* pois é mais barata e integra uma maior panóplia de *carriers*, que poderão, no futuro, trabalhar em conjunto com a *HUUB*.

Foi também decidido que se usaria o *TrackingMore* em conjunto com o *Easypost* pois o mesmo oferece um serviço de *tracking* mais completo que o *EasyPost* pois faz uma melhor identificação do estado da encomenda e consegue obter informações sobre todos os *carriers* por um preço aceitável.

Tabela 4.1: *Endpoints* para *shipping* propostos para o cumprimento dos requisitos.

Método	Url	Descrição
POST	/api/shipping/new	Coletar rates para um certo <i>shipment</i>
POST	/api/shipping/buy	Comprar <i>shipment</i> e respetiva <i>label</i>
POST	/api/shipping/track	Cria <i>tracking</i> de um certo <i>shipment</i>
GET	/api/shipping/track	Devolve todas as informações de <i>tracking</i> de um <i>shipment</i>
POST	/api/shipping/update	<i>Webhook</i> disparado quando existe atualização de <i>tracking</i> de um <i>shipment</i>

¹ Análise feita para 3191 envios realizados desde outubro de 2016 até abril de 2017, inclusive.

4.2 Integração com lojas de ecommerce

Depois de ter sido efetuada uma análise às diferentes opções para a integração com as lojas de *ecommerce*, chegou-se à conclusão, em conjunto com a *HUUB*, que o melhor seria montar uma *mashup* que integre diretamente com as lojas ao invés de usar uma já existente. Como, para já, são apenas duas lojas, ambas bem documentadas, optou-se pela integração direta com essas lojas, conseguindo-se adaptar a solução às necessidades reais da *HUUB*. Noutras palavras, a solução proposta passa por construir uma *mashup* que permita à *HUUB* ter maior controlo sobre as especificidades pretendidas.

A grande maioria dos *HUUB clients* trabalha para retalhistas, vendendo os seus produtos a diversas lojas espalhadas pelo globo. Por este motivo, existem poucos *HUUB clients* com loja *online*. Assim, esta solução irá ser implementada tendo em vista a integração com as duas lojas *online* que os *HUUB clients* utilizam, *Shopify* e *Woocommerce*, como pode ser visto na Figura 4.2. No entanto, está prevista a entrada de clientes que possuem *Magento*, *Prestashop* e *Tictail*, por isso, a solução será pensada com vista na escalabilidade para que a integração com essas ou outras loja *online*, no futuro, seja feita de uma forma rápida e eficaz.

	ecommerce character varying(250)	number_of_clients bigint
1	shopify	4
2	woocommerce	2

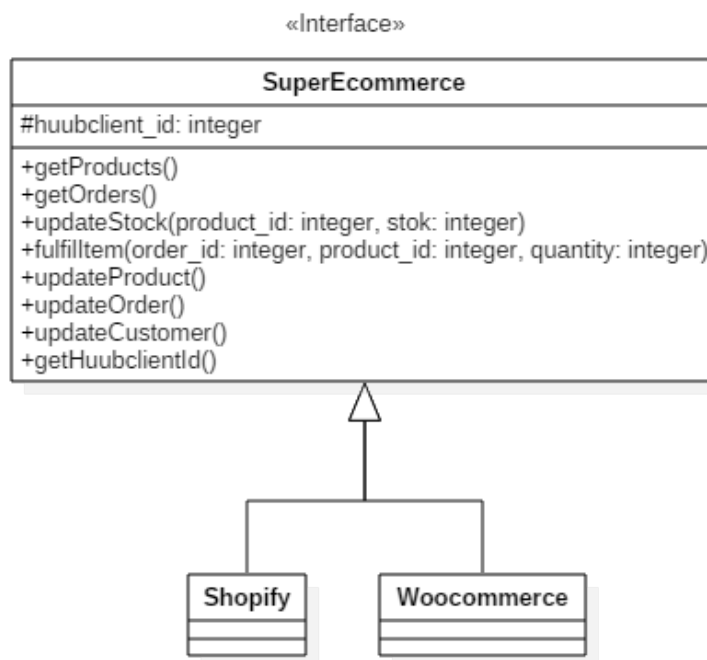
Figura 4.2: Distribuição do número de *HUUB clients* da *HUUB* por loja *online*.

Para o cumprimento de todos os requisitos, as rotas pensadas estão expostas na Tabela 4.2.

Tendo em mente uma arquitetura baseada em Object Oriented Programming (OOP), foram usadas as grandes vantagens deste tipo de programação: herança, encapsulamento, polimorfismo e abstração. Assim, o objetivo será implementar uma interface, *SuperEcommerce*, com todas as operações necessárias e depois, criar uma classe por cada loja *ecommerce* que implemente essa interface. Deste modo, quando for preciso integrar uma nova loja *online* terá de ser criada uma classe que implemente a interface mencionada. Com esta abordagem, Figura 4.3, o sistema torna-se flexível à integração com novos serviços, providenciando uma espécie de *road map* daquilo que é necessário implementar para que a integração fique completa.

Tabela 4.2: *Endpoints* para *ecommerce* propostos para o cumprimento dos requisitos.

Método	Url	Descrição
GET	<i>/api/ecommerce/products/get</i>	Coleta informação dos produtos de um dado <i>HUUB client</i>
POST	<i>/api/ecommerce/products/update</i>	<i>Webhook</i> disparado quando existe atualização de um produto
POST	<i>/api/ecommerce/stock/update</i>	Atualiza o stock de um certo produto na <i>webshop</i>
GET	<i>/api/ecommerce/orders/get</i>	Coleta informação das orders em trânsito de um dado <i>HUUB client</i>
POST	<i>/api/ecommerce/orders/update</i>	<i>Webhook</i> disparado quando existe atualização de uma <i>order</i>
POST	<i>/api/ecommerce/orders/fulfil</i>	<i>Fulfil</i> um certo produto de uma <i>order</i>
POST	<i>/api/ecommerce/customer/update</i>	<i>Webhook</i> disparado quando existe atualização de um <i>customer</i>

Figura 4.3: Diagrama das classes responsáveis pela integração de *ecommerce*.

4.3 Notificação de intervenientes

Para que se tome a decisão de que plataforma de envio de *emails* usar é preciso, primeiramente, definir que tipo de *endpoints* são necessários para que os requisitos sejam satisfeitos. Neste caso é bastante simples, estando representado na Tabela 4.3 e sendo que apenas precisamos de um *endpoint* para enviar *emails* e, opcionalmente, um *endpoint* para fazer *log* do estado do envio dos *emails* em situações de exceção, como *bouncing*, falha ou *email dropped*.

Para auxiliar a escolha da plataforma foi elaborada a Tabela 2.4. Daqui retirou-se que as opções mais viáveis para implementar seria o *Mailgun* e o *Sparkpost* pois estão já integrados com a *framework* que se usou e, para o volume de emails necessário, no máximo 2 000 por mês, são ambos gratuitos.

Acabou por se optar pelo *Mailgun* pois está muito bem documentado, satisfaz todos os requisitos e ainda disponibiliza *webhooks* e uma *API* para monitorização e *analytics*.

Para além do envio de *emails* será também implementado um sistema simples de notificações via *Slack* visto que toda a empresa usa esta plataforma de comunicação. Assim, quer seja através de canais para temas específicos ou através de mensagem direta, existirá um *bot* que tratará de notificar em tempo real sobre as situações mais críticas.

4.4 Arquitetura e tecnologia a usar

A parte fulcral em qualquer projeto é a idealização e conceptualização da arquitetura e como os componentes funcionam e comunicam entre si. Atualmente, a *HUUB* possui um sistema monolítico que está a tornar complicada a sua escalabilidade. Assim, é objetivo da empresa adotar um desenvolvimento baseado em micro serviços.

Desta forma, acha-se que a melhor estrutura seria a enunciada na Figura 4.4. Aqui, o *middleware* seria responsável por receber e tratar toda a informação proveniente da aplicação, usando, sempre que necessário, as *APIs* aí conectadas que se ligam com as diversas aplicações *third party*, sendo, portanto, o *backend* da aplicação.

Tabela 4.3: *Endpoints* para o envio de notificações propostos para o cumprimento dos requisitos.

Método	Url	Descrição
POST	/api/mail/send	Envia email
POST	/api/mail/log	Webhook disparado quando ocorre problemas no envio do email

²Apenas estão representadas as API cujo desenvolvimento irá ser abordado neste documento.

Com uma arquitetura deste tipo, a *HUUB* ganharia uma enorme flexibilidade pois, a qualquer momento, poderá substituir a utilização de um serviço externo com bastante facilidade, ganhando todas as vantagens que uma arquitetura *SOA* oferece.

Este *middleware* seria também o único responsável por escrever na *BD*.

Para otimizar alguns processos está pensado usar *Redis* que é uma tecnologia de *caching* que funciona com base num par *key value*, isto é, a cada chave (*key*) está associado um valor (*value*). A este par pode estar associado um *Time To Live (TTL)* que é o tempo de vida desse mesmo par, isto é, se uma *key* tem um *TTL* de 10 minutos, ao final desses 10 minutos, a *key* é automaticamente eliminada.

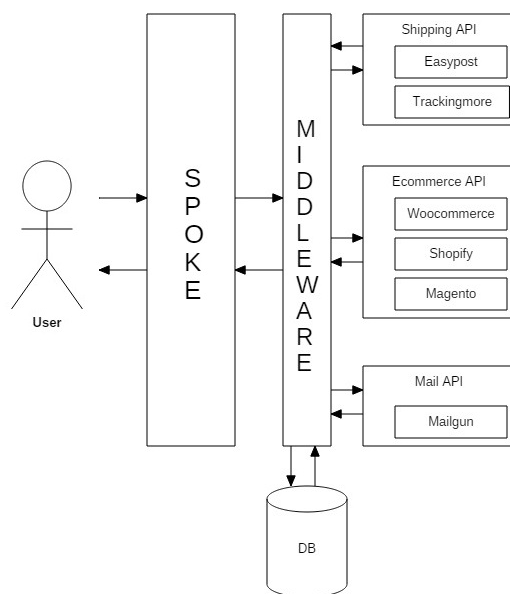


Figura 4.4: Arquitetura proposta para a *HUUB* ².

4.5 Conclusão

Este Capítulo 4 serviu para expor o modelo de solução dos problemas que foram propostos, criando-se já protótipos das rotas a serem implementadas, das tecnologias e da arquitetura que se pretende desenvolver.

No capítulo seguinte será tratado de forma mais exaustiva a implementação da solução a nível tecnológico.

Capítulo 5

Implementação

A execução do projeto sofreu algumas alterações comparativamente com aquilo que estava inicialmente pensado fazer e que foi exposto no Capítulo 4. Assim, neste capítulo será apresentada a implementação que foi realizada.

Primeiramente, o capítulo aborda o projeto a nível tecnológico e mais generalista. De seguida, a estrutura será concordante à dos capítulos anteriores, onde estarão as 3 partes já abordadas: automatização do processo de *shipping*, integração com lojas de *ecommerce* e notificação dos intervenientes.

Aqui, a abordagem será mais detalhada, no entanto, para qualquer dúvida deverá ser consultada a documentação que se encontra no anexo B.

5.1 Arquitetura e tecnologia a usar

5.1.1 Arquitetura

A arquitetura exposta na Secção 4.4 é platónica para a atual realidade da *HUUB*, pois a empresa é recente e tem uma equipa de desenvolvimento pequena para reconstruir o *Spoke*. Assim, é impossível esta arquitetura ser implementada no contexto desta dissertação pois envolveria mexer com a estrutura atual do *Spoke*, uma estrutura complexa e desconhecida, sendo necessário alocar recursos para efetuar essa transição. Deste modo, a arquitetura proposta, Figura 5.1, é um pouco diferente, no entanto, seguindo um desenvolvimento baseado no paradigma de OOP, a implementação será modular e facilmente adaptável à arquitetura que se acha ideal quando a empresa possuir meios e disponibilidade para tal.

Como foi referido, a *HUUB* pretende alterar a arquitetura do *Spoke* e uma dessas mudanças passa pela alteração da linguagem de programação usada. Até agora, era usado uma *framework* de

Python, *Web2Py*. Devido ao grande volume de dados transacionado e com o elevado crescimento da empresa, a mudança passa por começar a usar preferencialmente *PHP* com a *framework Laravel*.

Devido a *PHP* ser uma linguagem orientada por objetos e aliando o facto de *Laravel* ser uma *framework* madura com uma excelente comunidade e um *Object-Relational Mapping (ORM)* de excelência, *Eloquent*, a escolha da linguagem de programação recaiu também nesta linguagem ainda que existissem diversas opções pois essa é uma das grandes vantagens de seguir uma arquitetura *SOA*.

Para versionamento do código foi usado o *git* e o *gitlab*, pois é o que já está em uso na *HUUB*.

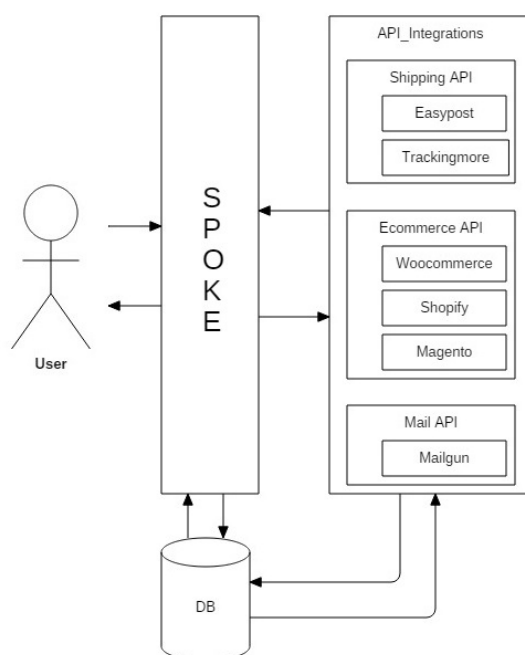


Figura 5.1: Arquitetura implementada para a *HUUB* ¹.

O desenvolvimento foi feito localmente com um ambiente semelhante ao ambiente do servidor de produção. Foi também montado num servidor físico um ambiente de teste para que o *software* seja testado, montado na *Amazon Web Services (AWS)*, o servidor de produção com sistema operativo *Ubuntu Server 16.04 Long Term Support (LTS)* servido por *Apache2*. Toda a manutenção foi efetuada usando o protocolo *Secure Shell (SSH)*.

5.1.2 Esquema da Base de Dados

Para este projeto houve a necessidade de consumir muita informação já existente na *BD* da *HUUB*, assim como gerar nova informação. Nas Figuras 5.2, 5.3 e 5.4 encontram-se os esquemas das

tabelas da base de dados que foram usadas. De notar que a cinzento se encontram as tabelas que tiveram de ser criadas durante o desenvolvimento.

5.1.3 Segurança

Existem diversas formas de garantir segurança em API, como o protocolo OAuth2 [33], JWT [34], *Basic Authentication* ou através de *API keys*. Aqui foi preciso ter em atenção que, sendo a implementação baseada na arquitetura REST, não deverá haver sessão e também o facto de ser necessária alguma coisa simples que permita implementar do lado dos diferentes serviços externos de modo a reconhecer sempre os pedidos efetuados por esses serviços (*webhooks*). Assim, escolheu-se efetuar a autenticação através de uma *API key*.

Para garantir a segurança das chaves de cada utilizador da API foi efetuada encriptação sha256 com um *salt* único para cada utilizador, garantindo assim chaves únicas e difíceis de serem descobertas.

Para complementar, foram definidos *roles* que, em conjunto com os *middlewares*, limitam e restringem o acesso aos *endpoints* dependendo do nível de permissão da *API key*.

5.1.4 GUI para gestão administrativa e tecnológica

Foi construída uma Graphical User Interface (GUI) para gestão por parte dos administradores. Está protegida com *login*, sendo apenas autorizada a utilizadores do *Spoke* com *role* de *admin*.

Nesta página é possível efetuar operações CRUD sobre diversas tabelas de gestão, como os *roles* existentes, os diversos utilizadores e gerir estados de *tracking*, bem como poder aceder aos *logs* de *email* e à documentação de toda a API. Estima-se que esta página será de extrema importância à medida que se forem implementados mais *micro serviços*, pois será um sítio único onde os vários *developers* poderão consultar documentações e efetuar a gestão de uma maneira mais simples e intuitiva.

Audit Schema		
log_info	log_jobs	log_emails
{documentation = Log of important actions and webhooks}		
*id integer	*id integer	*id integer
°title varchar	°job_id integer	°event_type varchar
°identifier varchar	°description varchar	°recipient varchar
°description varchar	°request_status varchar	°timestamp timestamp
°status varchar	°job_status varchar	°error_description varchar
°json text	°deleted_at timestamp	°resolved boolean
°created_at timestamp	°created_at timestamp	°deleted_at timestamp
°updated_at timestamp	°updated_at timestamp	°created_at timestamp
		°updated_at timestamp

Figura 5.2: Esquema do *schema audit* da base de dados.

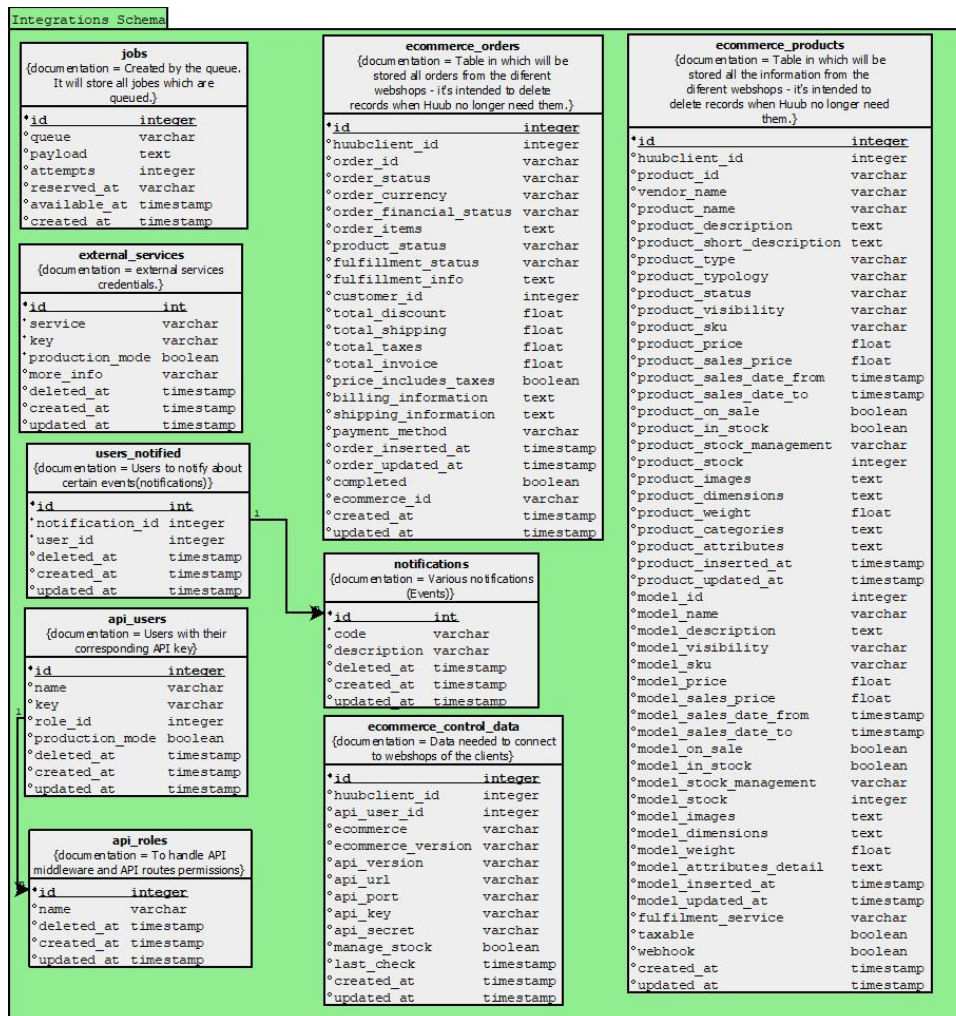


Figura 5.3: Esquema do *schema integrations* da base de dados.

Na Figura 5.5 é possível ver um exemplo de uma das páginas de gestão, a que gere os utilizadores da API. Aqui é possível observar que não existe maneira de editar as *keys* dos *users*, sendo que a única maneira será gerar uma nova chave aleatória baseada no algoritmo apresentado na Secção 5.1.3. É possível também ver que existe um modo de não produção que permitirá testar os diversos *endpoints* sem efetuar nenhuma compra. As APIs *key* de produção estão censuradas por motivos óbvios.

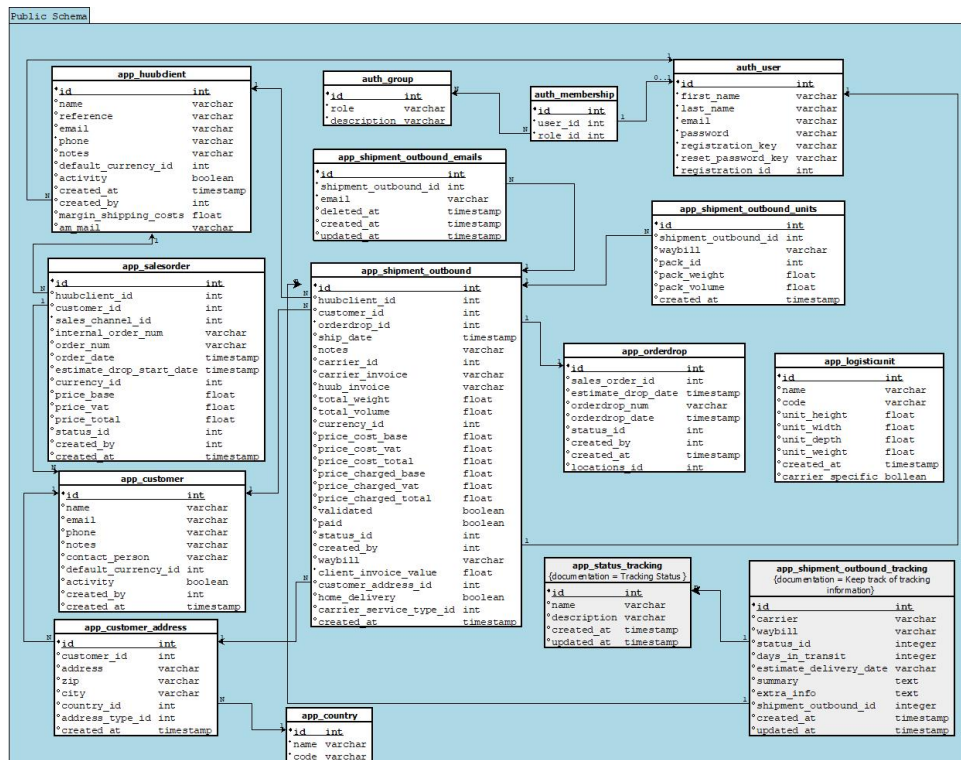


Figura 5.4: Esquema do *schema public* da base de dados.

Name	Key	Role	Production_mode	Action
easypost	[Redacted]	easypost	✓	[Icons]
huub	[Redacted]	admin	✓	[Icons]
Huub - Woocommerce test	[Redacted]	huubclient_ecommerce	✓	[Icons]
mailgun	[Redacted]	mailgun	✓	[Icons]
Maison Pixel - Shopify	[Redacted]	huubclient_ecommerce	✓	[Icons]
trackingmore	[Redacted]	webhook	✓	[Icons]
Twenty Five - Shopify	[Redacted]	huubclient_ecommerce	✓	[Icons]
Wolf & Rita - Woocommerce	[Redacted]	huubclient_ecommerce	✓	[Icons]

Figura 5.5: GUI para gestão administrativa. Vista dos *API Users*.

5.2 Automatização do processo de *shipping*

A automatização do processo de *shipping* pode ser dividida, como já foi falado, em duas grandes fases: envio de *orders* e *tracking* das mesmas.

Para este processo foram criados seis *endpoints*, evidenciados na Tabela 5.1. Será apresentada, de seguida informação mais detalhada, sendo que para melhor compreensão é recomendável a consulta da documentação no anexo B.

Tabela 5.1: *Endpoints* criados para o processo de *shipping* ².

Método	Endpoint	Descrição
POST	new/key={api_key}	Get Rates
POST	buy/key={api_key}	Buy an Order
POST	track/key={api_key}	Track a new Shipment
GET	track/{carrier_code}/{tracking_number}	Get Tracking Details
POST	track/update/key={api_key} ³	Update Tracking Details
GET	track/fetch/key={api_key}	Fetch Tracking Details

5.2.1 Envio de *orders*

Para esta fase foi desenvolvida uma classe, Figura 5.6, que trata de todas as operações relacionadas com o envio das *orders*, sendo este o único ponto de contacto com a plataforma do *Easypost*. Esta plataforma tem dois *endpoints* importantes, *New Order*, que serve para preparar a *order* e tem como retorno as *rates*, e *Buy Order*, que serve para efetivar a compra do transporte e tem como retorno uma *label* e a sua documentação.

Shipping
-dropbox -redisTTL -redisClient
+getRates(req) +buyShipment(req) +newOrder(req) +storeInDropbox(client, so, files) +getRedisKey(req)

Figura 5.6: Classe *Shipping* que trata de todas as operações do envio de *orders*.

²O prefixo deste grupo de *endpoints* é: *stem.api.thehuub.io/api/shipping/*

³Endpoint para consumo do trackingmore, *webhook*.

Quando um objeto desta classe é instanciado, é criada uma conexão ao *Redis*⁴, cujas *keys* têm um TTL de um dia, e instanciado um objeto do *EasyPost*⁵ que permitirá a utilização dos métodos oferecidos por este serviço.

Esta classe tem duas operações importantes, que são o *forecast* e a criação da *label*.

Para o *forecast*, o objetivo é obter os custos de envio de uma *order* para cada *carrier* e para cada serviço, sendo que é obrigatório enviar no pedido:

- morada de envio;
- morada de destino e se se trata de uma morada residencial ou não, pois em alguns casos poderá ter taxas acrescidas;
- tipologia⁶ dos *packs* a enviar e peso dos mesmo;
- informações de faturação.

De seguida, caso as *rates* já existam em *cache* são logo retornadas. Caso contrário, é feito um pedido de *New Order* ao *Easypost* que tem como resultado as diversas rates de transporte.

Para guardar as *rates* em *cache* é usado um par *key value*. Neste caso, a *key* é construída seguindo a lógica abaixo apresentada e o *value* é uma string em formato **JSON** com as diversas rates para esse transporte.

```
rate : [cidade_origem] : [zip_pais_origem] : [codigo_pais_origem] : [cidade_destino] : [zip_pais_destino] :
[codigo_pais_destino] : [tipologia_pack_1] : [peso_pack_1] : [tipologia_pack_2] : [peso_pack_2] :
... : [faturacao] : [home_delivery?]
```

Por exemplo, um pedido de rates das instalações da *HUUB* (Maia, 4470-777, Portugal) para Connecticut (zip: 06877) nos Estados Unidos da América, contendo um *pack*, CBox 60x40x40⁷ de 5 kg, com dois artigos, cada um com 2 kg e cujos encargos de taxas são afetos ao destinatário (*customer*), entregue numa morada comercial (não residencial) e com *incoterm* do tipo DAP (*Delivered at Place*)⁸, proveria num pedido cujo *body* do pedido seria:

```
1 {
  "from_address": {
    "name": "Pedro Santos",
    "street1": "Rua Joaquim Dias Salgueiro",
```

⁴Utilizando Predis, um *package* PHP de um *HUUB client* Redis que agiliza o desenvolvimento.

⁵Utilizando um *package* PHP oficial.

⁶Nome do *pack* caso sejam específicos de um *carrier* (Exemplo: DHLFlyer, UPSEXPRESSBox, Pak,...) ou, em caso contrário, as dimensões do *pack*.

⁷Caixa logística da *HUUB* com dimensões 60x40x40 cm.

⁸Incoterm são normas definidas no envio de mercadorias. *Incoterm* do tipo DAP atira toda a responsabilidade sobre o transporte para o destinatário sendo que o remetente apenas precisa de tratar das formalidades padrão.

```

    "street2": "130",
    "city": "Maia",
    "state": "PO",
    "zip": "4470-777",
    "country": "PT",
    "company": "The Huub",
    "phone": "(+351) 223 290 967",
    "email": ""
  },
  "to_address": {
    "name": "Bruce Wayne",
    "street1": "South Indian Summer Driv",
    "street2": "895",
    "city": "Ridgefield",
    "state": "CT",
    "zip": "06877",
    "country": "US",
    "company": "Wayne Enterprises",
    "phone": "415-123-4567",
    "email": "bruce@wayneenterprises.com",
    "residential": "false"
  },
  "shipments": [
    {
      "parcel": {
        "typology": "CBox 60x40x40",
        "weight": 5
      },
      "customs_info": {
        "eel_pfc": "NOEEI 30.37(a)",
        "contents_type": "merchandise",
        "contents_explanation": "",
        "restriction_type": "none",
        "non_delivery_option": "return",
        "customs_items": [
          {
            "description": "Black cape",
            "quantity": 2,
            "weight": 2,
            "value": 100,
            "hs_tariff_number": "123456",
            "origin_country": "PT",
            "currency": "EUR"
          }
        ]
      }
    ]
  },
  "billing": {

```

```
56      "receiver": true,  
        "bill_account": "",  
        "bill_country": "",  
        "bill_zip": "",  
        "incoterm": "DAP"  
    }  
}
```

O resultado da chamada à API seria:

```
5  
10  
15  
20  
25  
30  
35  
{  
  "code": 200,  
  "message": "Success.",  
  "error_messages": [],  
  "rates": [  
    {  
      "carrier": "TNTExpress",  
      "service": "1200Express",  
      "rate": "100.92",  
      "currency": "EUR"  
    },  
    {  
      "carrier": "TNTExpress",  
      "service": "GlobalExpress",  
      "rate": "78.12",  
      "currency": "EUR"  
    },  
    {  
      "carrier": "TNTExpress",  
      "service": "EconomyExpress",  
      "rate": "73.33",  
      "currency": "EUR"  
    },  
    {  
      "carrier": "DHLExpress",  
      "service": "MedicalExpressNonDoc",  
      "rate": "303.18",  
      "currency": "EUR"  
    },  
    {  
      "carrier": "DHLExpress",  
      "service": "Express1200NonDoc",  
      "rate": "302.55",  
      "currency": "EUR"  
    },  
    {  
      "carrier": "DHLExpress",  
      "service": "ExpressWorldwideNonDoc",  
      "rate": "297.55",
```

```

40      "currency": "EUR"
    },
    {
      "carrier": "UPS",
      "service": "Express",
45      "rate": "364.38",
      "currency": "EUR"
    },
    {
      "carrier": "UPS",
      "service": "Expedited",
50      "rate": "279.23",
      "currency": "EUR"
    },
    {
      "carrier": "UPS",
      "service": "UPSSaver",
55      "rate": "343.47",
      "currency": "EUR"
    },
    {
      "carrier": "UPS",
      "service": "ExpressPlus",
60      "rate": "422.23",
      "currency": "EUR"
    }
  ]
}

```

Este mesmo pedido iria resultar na seguinte *redis key*:

rate : Maia4470777PT : Connecticut06877USA : CBox60x40x60 : 5 : customer : nothome

Caso não exista nenhum registo em *cache*, a *API* vai tratar a informação e fazer um pedido de *rates* ao *Easypost*. Quando recebe a resposta, guarda em *cache* e retorna em formato *JSON*.

Devido ao processo de obtenção de *rates* fazer parte da negociação de preços com o *HUUB client*, são feitos bastantes pedidos iguais. Cada um destes pedidos, que são feitos ao *Easypost*, demoram em média 5 a 6 segundos. Por esse motivo foi implementado este mecanismo de *caching* de forma a otimizar este processo. Desta forma, apenas o primeiro pedido terá essa duração, sendo que todos os outros pedidos iguais que se seguirem demoram em média 100 a 200 milissegundos, sendo que cada *redis key* tem um *TTL* de 24 horas. O valor associado à *key* é a própria resposta da *API*.

O processo referido anteriormente para a obtenção de *rates* pode ser visto no diagrama de sequência da Figura 5.7.

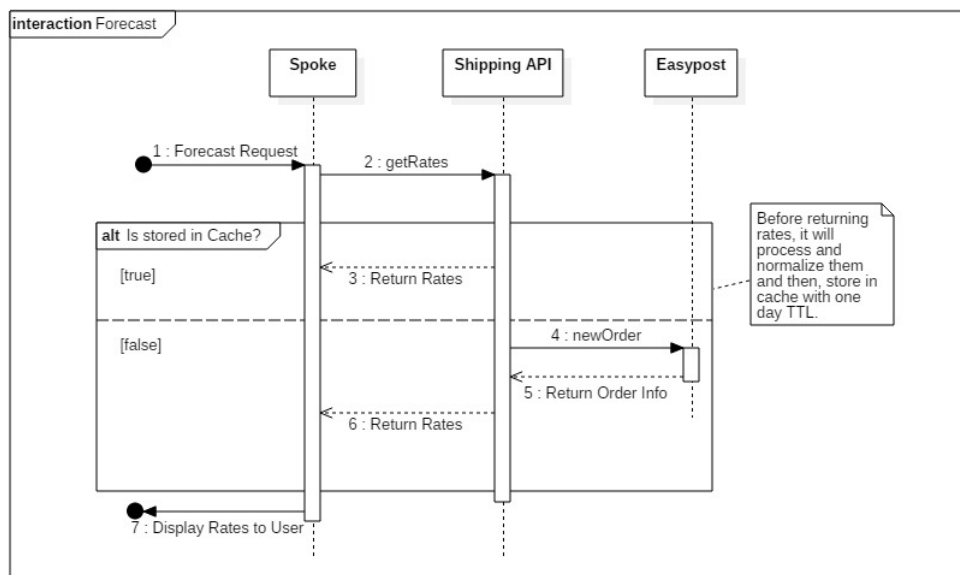


Figura 5.7: Diagrama de sequência para obtenção de *rates* para um transporte.

A outra operação importante é a criação da *label*, que vai despoletar o envio da *order* e iniciar a fase seguinte, o *tracking*

Para a criação da *label*, é necessário que seja enviado, para além do que foi atrás exposto, o nome do *HUUB client*, o número da *SO*, o *shipment_outbound_id*, para identificar o *shipment*, e a *rate* escolhida (conjunto *carrier* e *serviço*⁹).

O processo começa então com um pedido de *rates* igual ao exposto anteriormente para verificar que o par *carrier-serviço* existe e está disponível para o envio que se pretende realizar.

Depois, para o transporte atrás mencionado, escolhendo-se o serviço *UPSSaver* da *UPS*, é necessário ainda enviar, para além do requerido no pedido de *rates*, a identificação do *shipment_outbound_id*, assim como identificação das diversas *logistic units* com o *shipment_outbound_unit_id* e o nome do *HUUB client* a quem pertence a *SO*. Opcionalmente, podem enviar-se duas referências para colocar na *label*. Determinou-se que essas referências seriam, em primeiro lugar, o nome do *HUUB client* e número da ordem de compra e, em segundo lugar o número interno da *SO*.

Assim, o *body* do pedido será o seguinte JSON:

```

3 {
  "rate": {
    "price": "652.66",
    "carrier": "UPS",
    "service": "UPSSaver"
  },

```

⁹Exemplo: No caso da *UPS UPSSaver*, *UPS* é o *carrier*, *UPSSaver* é o serviço.

```

    "shipment_outbound_id": 3406,
8  "order": {
    "ref1": "Order From HUUB client 1 (#23675)",
    "ref2": "SO#2017-52789",
    "from_address": {
13      "city": "Maia",
      "state": "PO",
      "name": "Pedro Santos",
      "zip": "4470-777",
      "country": "PT",
      "street1": "Rua Joaquim Dias Salgueiro",
18      "street2": "130",
      "company": "The Huub",
      "phone": "(+351) 223 290 967",
      "email": ""
    },
23  "to_address": {
    "name": "Bruce Wayne",
    "street1": "South Indian Summer Driv",
    "street2": "895",
    "city": "Ridgefield",
28    "state": "CT",
    "zip": "06877",
    "country": "US",
    "company": "Wayne Enterprises",
    "phone": "415-123-4567",
33    "email": "bruce@wayneenterprises.com",
    "residential": "false"
  },
  "shipments": [{
38    "shipment_outbound_unit_id": "3143",
    "parcel": {
      "typology": "CBox 60x40x40",
      "weight": 5
    },
43    "customs_info": {
      "eel_pfc": "NOEEI 30.37(a)",
      "contents_type": "merchandise",
      "contents_explanation": "",
      "restriction_type": "none",
      "non_delivery_option": "return",
48    "customs_items": [
      {
53        "description": "Black cape",
        "quantity": 2,
        "weight": 2,
        "value": 100,
        "hs_tariff_number": "123456",
        "origin_country": "PT",

```

```
        "currency": "EUR"
      }
    ]
  },
  "billing": {
    "bill_country": "",
    "incoterm": "DAP",
    "bill_zip": "",
    "bill_account": "",
    "receiver": true
  },
  "sales_order": "SO#2017-52789",
  "client": "HUUB client 1"
}
```

Este *endpoint* retorna o *waybill* para que se possa fazer *tracking* da *order*, assim como a *label* de transporte, Figura 5.8 e a *invoice*, que lista os itens de cada pacote bem como o seu preço para controlo alfandegário. A resposta demora cerca 13 segundos e pode ser observada, em formato JSON de seguida:

```
{
  "code": 200,
  "message": "Success.",
  "error_messages": [],
  "shipment_outbound_id": 3406,
  "shipments": [
    {
      "easypost_id": "shp_6803a288763c417ab3d02e2a44bc86a2",
      "shipment_outbound_unit_id": "3143",
      "parcel": {
        "length": 40,
        "width": 60,
        "height": 40,
        "weight": 5
      },
      "label": "https://easypost-files.s3-us-west-2.amazonaws.com/files/postage_label/20170610/a4d6c88635cf4d3a8350b4acf2be8605.pdf",
      "waybill": "1ZA9V2340496676178",
      "forms": [
        "https://easypost-files.s3-us-west-2.amazonaws.com/files/form/20170610/01bf84125c714df9bfed339d229b14e2.pdf"
      ]
    },
    {
      "carrier_waybill": "1ZA9V2340496676178",
      "selected_rate": {
        "carrier": "UPS",
```

29

```

    "service": "UPSSaver",
    "price": "343.47",
    "currency": "EUR"
  }
}

```

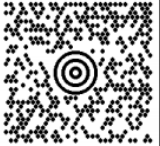
PEDRO SANTOS 351223290967 THE HUUB RUA JOAQUIM DIAS SALGUEIRO 4470777 MAIA PO PORTUGAL		5 KG 1 OF 1
SHIP TO: BRUCE WAYNE 4151234567 WAYNE ENTERPRISES SOUTH INDIAN SUMMER DRIV 895 RIDGEFIELD CT 06877 UNITED STATES		SHP#: A9V2 34V9 4BJ SHP WT: 5 KG SHP DWT: 19.5 KG DATE: 10 JUN 2017 DWT: 60,40,40
	CT 068 0-02 	
UPS SAVER TRACKING #: 1Z A9V 234 04 9667 6178		1P
		
BILLING: P/P DESC: Black cape		INV
Reference No.1: Order From Wolf & Rita (#23675) Reference No.2: SO#2017-52789		
Shipper agrees to the UPS Terms found at www.ups.com and UPS service centers. For international air carriage, the Warsaw Convention as amended or Montreal Convention may apply and limits UPS's liability for loss or damage to cargo. International carriage by road may be subject to the Convention on the Contract for the International Carriage of Goods by Road. Except as otherwise governed by international conventions or other mandatory law, the UPS Terms limit UPS's liability for damage, loss or delay of this shipment. There are no stopping places agreed upon at the time of tender of the shipment and UPS reserves the right to route the shipment in any way it deems appropriate. Where allowed by law, shipper authorizes UPS to act as forwarding agent for export control and customs purposes. If exported from the US, shipper certifies that the commodities, technology or software were exported from the US in accordance with the Export Administration Regulations. Diversion contrary to law is prohibited.		

Figura 5.8: *Label* de transporte criada através do *Easypost*.

Este *endpoint* também guarda estes documentos e a *label* na Dropbox para serem impressos na seguinte diretoria:

./Pasta da equipe Huub/Orders Docs (Integrations)/{nome do HUUB client}/{número da SO}

Todo este processo pode ser visto no diagrama de sequências da Figura 5.9.

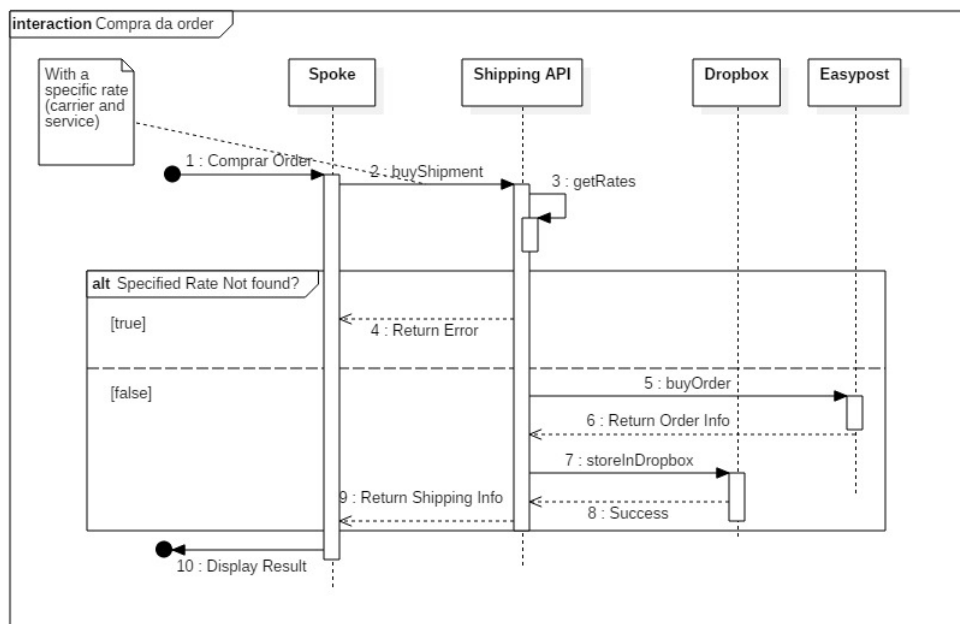


Figura 5.9: Diagrama de sequência para criação de uma *label* de transporte.

5.2.2 Tracking de orders

O processo de *shipping* culmina nesta fase, o *tracking* da *order*. Foi implementada também uma classe, Figura 5.10, que trata de todas as operações realizadas com o *TrackingMore*, serviço externo usado para *tracking*.

Quando um objeto desta classe é instanciado, é criada uma instância de um objeto *TrackingMore* que auxilia o pedido de informações ao serviço com o mesmo nome.

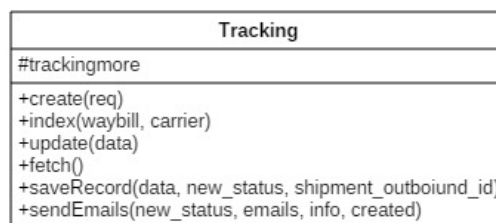


Figura 5.10: Classe *Tracking* que trata de todas as operações relacionadas com o *tracking* das *orders*.

Esta classe permite uma escrita bidirecional entre o *Spoke* e o *TrackingMore* e começa após a compra da *label*¹⁰ com a função *create*, sendo necessário enviar o *tracking number*, nome do *carrier* e o *shipment_outbound_id* para identificar o *shipment*, sendo possível enviar também informações adicionais como endereço de *emails* dos *customers* para envio de notificações, entre outras. Nesta fase, irá ser criado um registo na tabela *app_shipment_outbound_tracking* com toda a informação relevante.

Depois do *shipment* estar registado na plataforma, e até ele ser entregue, sempre que surgir uma nova atualização irá ser disparado um *webhook* por parte do *TrackingMore* que a *API* vai processar. Aqui, será recebido, entre outras informações, o estado atual da encomenda e o detalhe.

Durante o desenvolvimento, notou-se que existiam alguns casos que o *TrackingMore* não considerava como exceção mas precisariam de ser considerados como tal. Para tal ser possível, desenvolveu-se à parte uma *API* que, recebendo o detalhe da última atualização do envio, determina o *status* da *order* com base em técnicas de *textmining*. Estas alterações resultam num *update* da tabela *app_shipment_outbound_tracking*.

Foram também criadas duas rotas, associadas a dois métodos desta classe que podem ser interessantes. Um desses métodos, *index()*, tem por objetivo retornar a informação atualizada da *order* sem que para isso seja feito um pedido ao *TrackingMore*, sendo relativamente útil quando a *HUUB* disponibilizar uma página própria de *tracking* das *orders* dos seus *HUUB clients*. O outro método, *fetch()*, foi criado com o propósito de auxiliar a equipa de *BI* a recolher toda a informação presente no *TrackingMore* para serem criados *datasets* de treino.

5.3 Integração com lojas de *ecommerce*

Como foi dito na Secção 3.3, durante o projeto, foram integradas duas lojas *online*, o *Shopify* e o *Woocommerce*, no entanto, é de extrema importância que o *software* tenha sido desenvolvido para se inserir uma nova loja *online* nesta integração.

Desta maneira, achou-se que o mais vantajoso seria usar o *Factory*, *design pattern* do tipo criacional cujo grande intuito é colmatar a falta de conhecimento prévio de qual subclasse criar [35].

Este *design pattern* foi fortemente utilizado e ajudou na obtenção de um código mais limpo e fácil de perceber. Na Figura 5.11 pode ser encontrado o diagrama de classes.

¹⁰A compra da *label* pode ser feita via *Easypost*, caso exista integração (*UPS*, *TNT* e *DHL*) ou manualmente pois o *TrackingMore* possui um grande número de *carriers* suportados, sendo apenas preciso ter o *tracking number*.

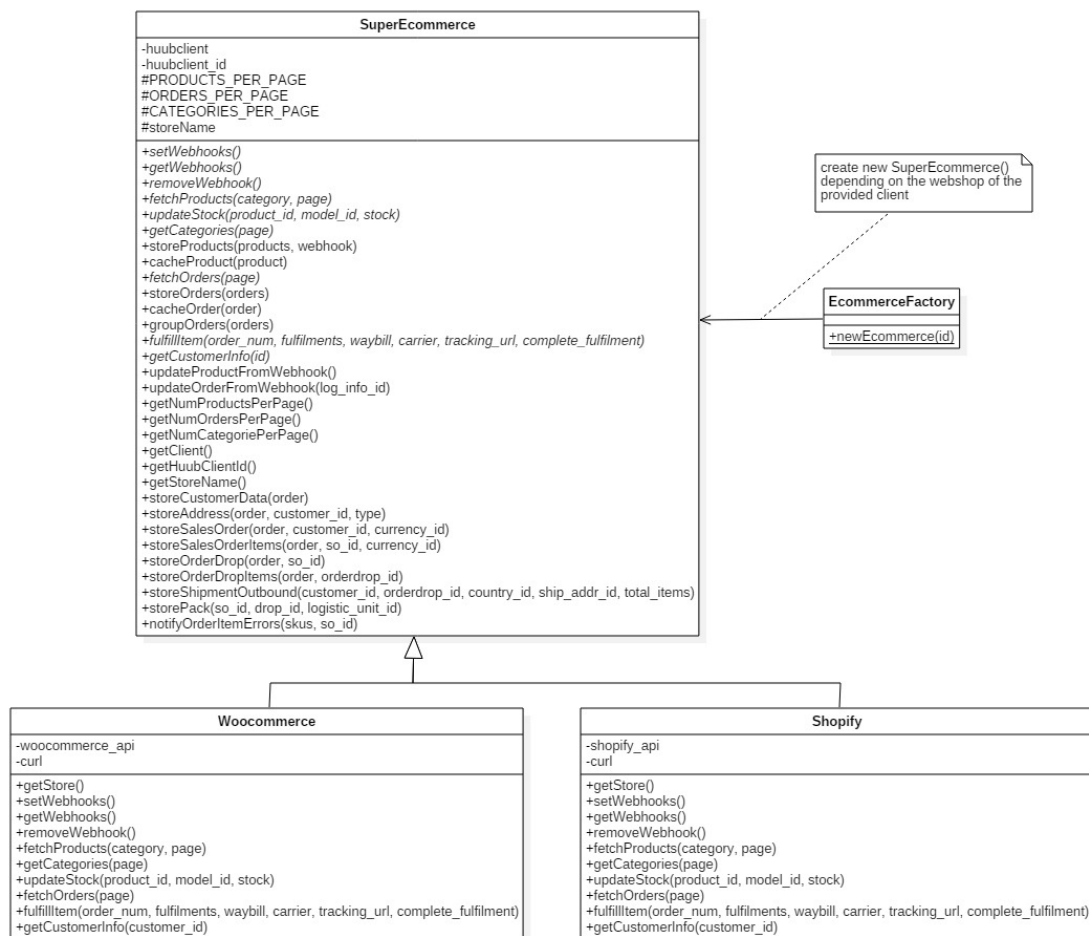


Figura 5.11: Diagramas de classes da estrutura montada para as classes das lojas *ecommerce*.

Neste diagrama podemos observar as vantagens que o *design pattern* traz. A classe *EcommerceFactory* tem o conhecimento inerente de que subclasse de *SuperEcommerce* necessita de criar, consoante o *id* do cliente em questão, tirando toda essa lógica do lado da aplicação. Desta forma o código torna-se mais limpo e organizado. Esta técnica, assim como outro *design pattern*, *Adapter*, foi também utilizada para outros objetos como *Addresses*, *Customers*, *Products*, *Orders* e *OrderItems*.

Este *design pattern*, *adapter*, é do tipo estrutural e faz com que as diversas subclasses adaptem a estrutura da classe da qual descendem, usando os métodos abstratos da maneira que melhor lhes convier [35].

Como cada loja *online* de *ecommerce* tem o seu esquema de dados, lógica de negócio e métodos diferentes ou que usam diferentes recursos, foi criada uma estrutura base para as lojas de *ecommerce* com vista a normalizar todo este processo. Assim, por cada uma dessas lojas *online*, é criada uma subclasse que implementa os métodos abstratos da classe mãe. Esta normalização é muito importante, criando abstração em tudo o que é diferente e generalização no que é comum.

Para o processo de *ecommerce*, os *endpoints* criados podem dividir-se em duas categorias: aqueles usados internamente pela *HUUB*, apresentados na Tabela 5.2 e, por outro lado, um conjunto de *endpoints* que são consumidos pelas lojas online, sendo disparados consoante certos eventos, que são chamados *webhooks* e estão apresentados na Tabela 5.3.

Tabela 5.3: *Endpoints* criados para consumo das lojas *online* no processo de *ecommerce* ¹².

Método	Endpoint	Descrição
POST	/products/update?key={api_key}&client_id={id}	<i>Webhook</i> para atualização de produtos
POST	/orders/update?key={api_key}&client_id={id}	<i>Webhook</i> para atualização de <i>orders</i>

Neste processo, as funcionalidades podem ser divididas em 5 grupos. São eles:

- *Products*;
- *Order*;
- *Customers*;
- *Webhook Management*;
- *Webhooks*.

5.3.1 *Products*

Aqui estão inseridas as primeiras 4 rotas. A primeira, com sufixo do *endpoint* */products/fetch*, é especialmente importante para o *onboard* de novos *HUUB clients* ou para a sincronização dos produtos no início da *season* para que, os produtos registados no *Spoke* coincidam com os produtos da loja *online*.

Existindo a possibilidade de alguns *HUUB clients* terem centenas ou até milhares de produtos, torna-se inviável um pedido buscar a informação de todos os produtos ou de um conjunto de produtos (por categoria). Desta forma, criou-se um sistema de *queue*. Assim, quando for necessário fazer *fetch* de todos os produtos ou dos produtos de certas categorias, o servidor irá criar um *job* e adiciona-lo à *queue*. A função desta tarefa é fazer um pedido à API da loja *online* de um número fixo de produtos¹³ e processar a informação recebida. Uma notificação será lançada quando o *fetch* de todos os produtos for completado, assim como um relatório de erros, caso ocorram.

Os produtos recolhidos com estes *jobs* são normalizados, usando a classe *Products*, sendo implementados os *design patterns* *Factory* e *Adapter*, como exemplificado na Figura 5.12.

¹¹O prefixo deste grupo de *endpoints* é: *stem.api.thehuub.io/api/ecommerce/{HUUBclient_id}*.

¹²O prefixo deste grupo de *endpoints* é: *stem.api.thehuub.io/api/webhooks/ecommerce*.

¹³O número *default* é 25.

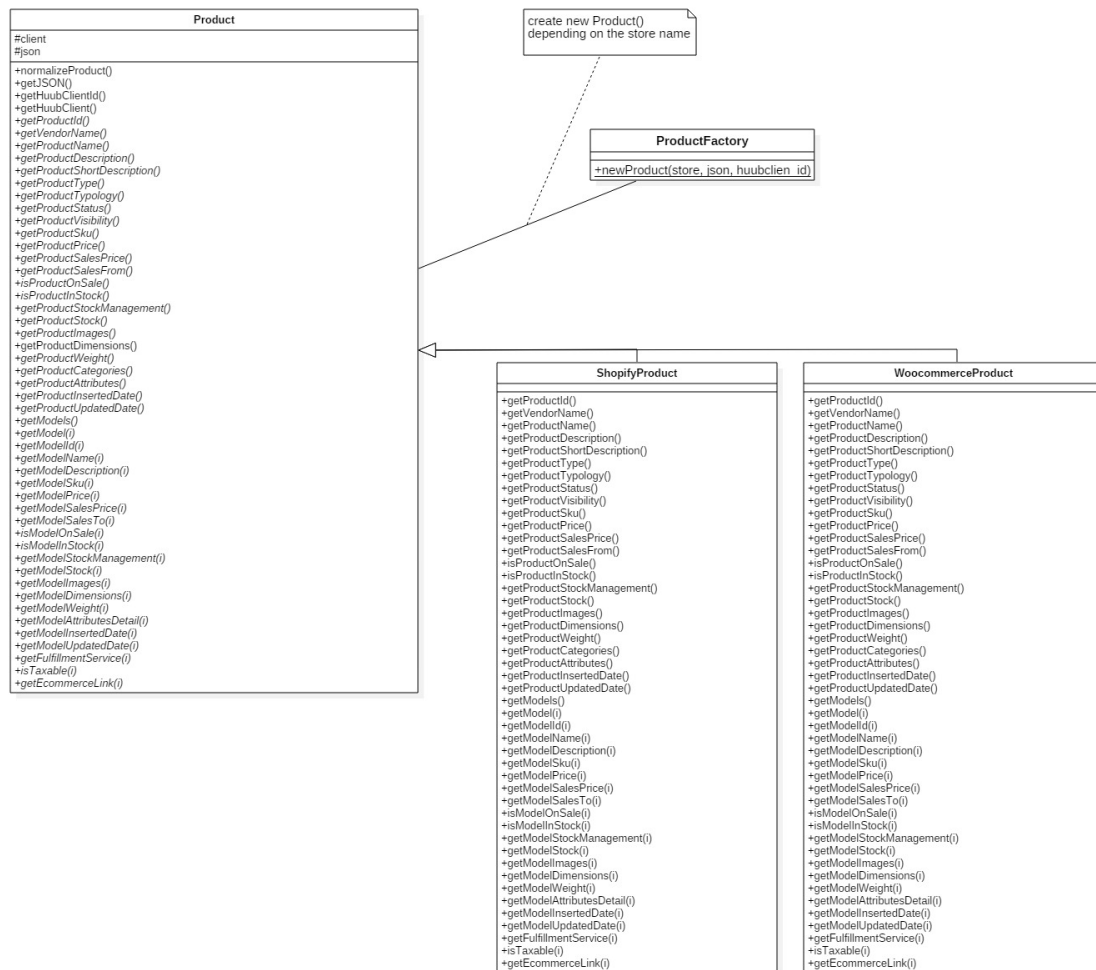


Figura 5.12: Diagramas de classes da estrutura montada para as classes dos produtos.

Depois dos produtos de cada loja *online* serem normalizados para coincidirem com a estrutura de dados pretendida, são guardados na base de dados, numa tabela criada para o efeito, *ecommerce_products*. Esta tabela é bastante extensa ¹⁴ e serve como *staging* para operações futuras. Os produtos normalizados também são guardados em *cache* com um TTL de 24 horas e seguindo a norma

product : [*huubclient_id*] : [*product_sku*]

Por exemplo, um produto com Stock Keeping Unit (SKU) *woo-single-2*, pertencente ao cliente

Tabela 5.2: *Endpoints* criados para consumo interno da HUUB no processo de *ecommerce* ¹¹.

Método	Endpoint	Descrição
GET	/products/fetch?categories={catg_ids}	Coletar informação dos produtos de um dado cliente (todos ou de certas categorias).
GET	/products/get?key={api_key}	Coletar informação dos produtos de um dado cliente que estão em <i>cache</i> .
GET	/products/categories?key={api_key}	Coletar categorias de produtos de um dado cliente.
POST	/products/stock?key={api_key}	Atualizar <i>stock</i> de um certo produto.
GET	/orders/fetch?key={api_key}	Coletar informação das <i>orders</i> em trânsito de um dado cliente.
GET	/orders/get?key={api_key}	Coletar informação das <i>orders</i> de um dado cliente que estão em <i>cache</i> .
POST	/orders/fulfil?key={api_key}	Fulfil Order de um dado cliente
GET	/webhooks?key={api_key}	Coletar informação sobre <i>webhooks</i> associados a um dado cliente.
POST	/webhooks?key={api_key}	Criar conjunto de <i>webhooks</i> para um cliente.
DELETE	/webhooks?key={api_key}	Remover um ou mais <i>webhooks</i> de um dado cliente

com *id* 1 iria resultar na seguinte *redis key*:

product : 1 : *woo – single – 2*

Outro *endpoint* criado foi o que tem sufixo */products/get*. Este *endpoint* complementa o anterior e serve para recolher todos os produtos guardados em *cache* de um certo cliente. Pressupõe sempre que o *endpoint* anterior, *fetch*, seja chamado à posteriori.

De seguida, foi implementado um *endpoint* para coletar todas as categorias de produtos de um dado cliente. Normalmente, os *HUUB clients* quando lançam uma nova *season* adicionam-na às

¹⁴Tem atualmente 51 colunas

categorias. Assim, sabendo os *ids* das categorias pretendidas, quando for necessário sincronizar os produtos de apenas uma *season*, faz-se o *fetch* dos produtos das categorias que se quiser.

De notar que a sincronização de produtos está numa fase embrionária. Como existem muitas diferenças no modo de atuar de cada *HUUB client* e aliando o facto de as diversas lojas *online* possuírem estruturas diferentes, é bastante complicado, e fora do contexto desta dissertação, categorizar os produtos. Desta forma, foi guardada toda a informação dos produtos, normalizando grande parte dessa informação.

Por último, tem-se um *endpoint* para atualizar o *stock*. Sendo a *HUUB* o único operador logístico dos seus clientes, ou seja, a única entidade a gerir *stock*, é um dado adquirido que o *stock* correto está sempre do lado do *Spoke*, pois os produtos disponíveis estão todos em armazém. Assim, quando existir alguma mudança no *stock* de algum produto, este *endpoint* é chamado para atualizar a loja *online*, de modo a não existirem problemas de rutura de *stock*.

5.3.2 Orders

Neste grupo, estão inseridos mais três *endpoints*. Os dois primeiros, *fetch* e *get* são semelhantes aos seus homónimos na secção anterior. O primeiro adiciona à *queue* um *job* cujo intuito é coletar informação de um número fixo de *orders*¹⁵ que estejam em trânsito. Estas *orders* são também normalizados, usando a classe *Orders*, sendo implementados os *design patterns* *Factory* e *Adapter*, como exemplificado na Figura 5.13.

As *orders* serão também armazenadas numa estrutura de dados comum, na base de dados, *ecommerce_orders*. Esta tabela serve também como *staging* para operações futuras. As *orders* normalizadas são guardados em *cache* com um TTL de 24 horas e seguindo a norma:

order : [*huubclient_id*] : [*order_number*]

Por exemplo, uma *order* identificada com #446, pertencente ao cliente com *id* 1 iria resultar na seguinte *redis key*:

order : 1 : #446

Cada *order* é composta por vários *order items*. Desta maneira, efetuou-se, da mesma forma, o processo de normalização dos itens de cada *order*, Figura 5.14.

¹⁵O número *default* é 25.

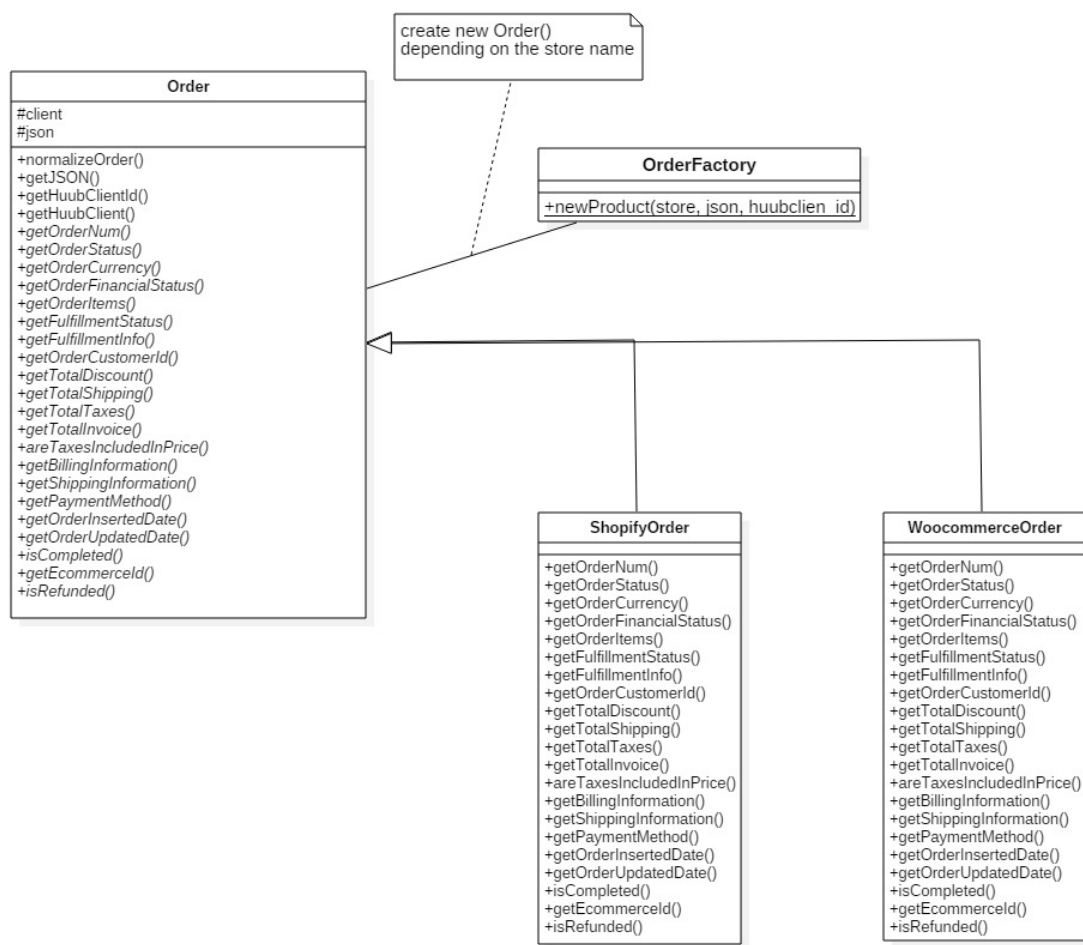


Figura 5.13: Diagramas de classes da estrutura montada para as classes das *orders*.

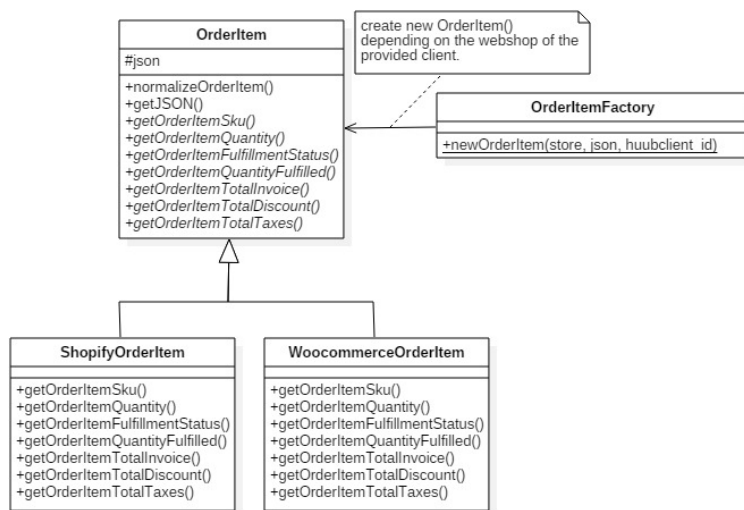


Figura 5.14: Diagramas de classes da estrutura montada para as classes de cada *order item*.

Junto de cada *order* é também enviado os dados do *customer* associado bem como as moradas de *billing* e *shipping*. Assim, efetuou-se um processo semelhante aos atrás mencionados, com a normalização destas classes, Figuras 5.15 e 5.16.

Um exemplo de como é efetuada essa normalização é possível ser visto no Anexo A.

O último *endpoint* neste grupo é o *Fulfilment*. Este processo apenas existe em algumas lojas *ecommerce*. No caso do *Shopify*, o processo de *fulfilment* é bastante completo, sendo possível realizá-lo parcialmente, enviando o *SKU* e quantidade dos produtos expedidos, assim como informações sobre o *tracking* como Uniform Resource Locator (URL) e *waybill*. Já no *Woocommerce*, não existindo o conceito de *fulfilment* parcial, apenas pode ser feito *fulfilment* completo de uma *order*, arquivando-a na *webshop*.

5.3.3 Customers

Neste grupo apenas existe um *endpoint* cujo intuito é obter informação sobre um determinado *customer*, fornecendo, para isso, o id da loja *online*.

O grande objetivo deste *endpoint* será a obtenção de mais informação do que aquela que está guardada na BD da *HUUB*.

5.3.4 Webhook management

Para gerir os *webhooks* de um certo cliente é necessário efetuar 3 operações: escrita, leitura e eliminação. Assim, é providenciado um *endpoint* de escrita, com o objetivo de criar todos os *webhooks* necessários para que a integração funcione da melhor maneira, sendo eles disparados quando existe alteração numa *order* ou num produto. Existe também um *endpoint* de leitura dos *webhooks* ativos bem como os seus identificadores e um *endpoint* para eliminar um ou mais *webhooks*.

Desta maneira, é possível gerir os *webhooks* de todos os *HUUB clients* de forma eficiente e completa.

5.3.5 Webhooks

Os *webhooks* são o meio para as lojas *online* escreverem no *Spoke* consoante o disparo de certos eventos, neste caso, a atualização de um produto ou de uma *order*.

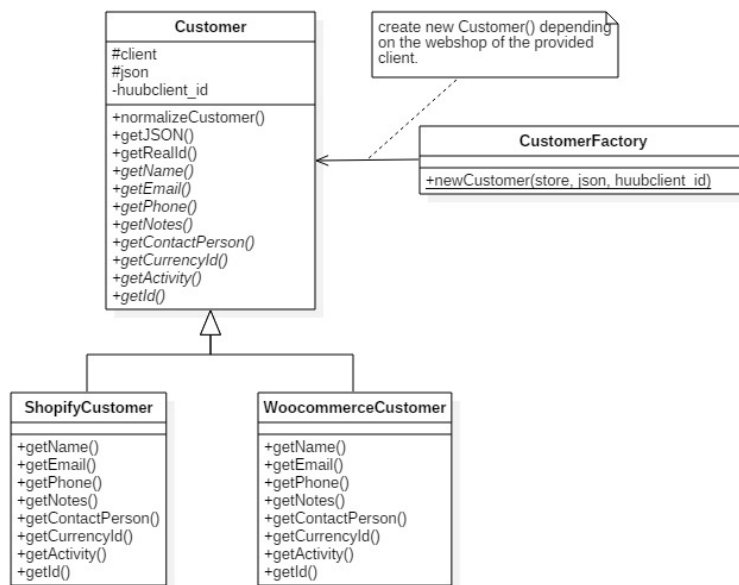


Figura 5.15: Diagramas de classes da estrutura montada para as classes de *customer*.

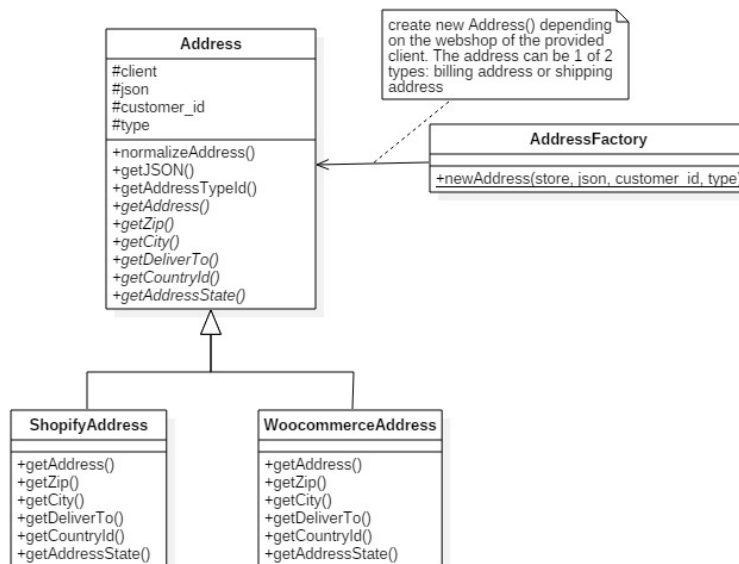


Figura 5.16: Diagramas de classes da estrutura montada para as classes de *address*.

5.3.5.1 Atualização de um produto

No momento em que um produto é criado ou alterado, a loja *online* em questão faz pedido HTTP a um *endpoint* criado para o efeito que irá normalizar todos os campos recebidos das diversas lojas *online*, de modo a ter como *output* uma estrutura de dados igual, independentemente da loja *online* de que proveio. De seguida, irá guardar toda essa informação na tabela *ecommerce_products*, tabela de *staging*. Como referido na Secção 5.3.1, esta tabela tem muitas colunas e é bastante pesada. Isto deveu-se ao facto da empresa querer armazenar toda a informação possível de modo a que, no futuro, possa usar esses dados para novas *features* e para *data analytics*.

Um problema encontrado, foi a dificuldade em escrever nas tabelas da *HUUB* a informação dos produtos recebidos pois existem bastantes campos de preenchimento obrigatório que não são encontrados nas lojas *online* como o *supplier*, a *season* e a família de produtos. Por outro lado, ainda existem outros campos que estão "escondidos", isto é, campos como o tipo de material do produto e a composição do produto, são por muitas vezes colocados na descrição do produto sem regras definidas.

Outro exemplo de problemas encontrados foi nas categorias. Cada cliente tem a sua nomenclatura, havendo *HUUB clients* que chegam a criar categorias como: *girl, girls, woman, female*, mulher, rapariga, etc, palavras que significam exatamente o mesmo. Já outros, optam por ter categorias como: casaco, calças, *bottom parts*, etc. Neste último caso, estas categorias eram redundantes pois já existe um campo na base de dados da *HUUB* para o tipo de produto, no entanto, com um nome ligeiramente diferente, não havendo correspondência.

Desta forma, optou-se por normalizar o máximo de campos possível, sendo que outros, como atributos¹⁶ e categorias foram guardados em formato JSON. Foi implementado desta maneira para que os AMs possam verificar a informação e complementar o que falta.

Como trabalho futuro seria interessante implementar algoritmos de *text mining* e algoritmos de análise lexical como a distância de *Levenshtein* [36] e *soundex* [37], de modo a tentar criar um *match* entre o que está nas lojas *online* e o *Spoke*, ao mesmo tempo que se agregaria categorias com nomes semelhantes numa só.

Este processo está pronto a ser posto em produção, no entanto, o desenvolvimento de infraestrutura para completar o processo do lado do *Spoke* está em falta.

Este processo agilizará o trabalho dos *HUUB clients* pois, neste momento, os produtos são importados através de ficheiros *xls*, sendo trabalhoso pois é necessário escrever todos os dados de todos os produtos no formato aceite pelo *Spoke*, o que resulta em bastantes erros, como incompatibilidade ou mesmo falta de campos críticos e até mesmo de *timeout* do servidor pois o ficheiro pode conter muitos produtos para importar.

¹⁶Os atributos podem ser cor, tamanho, textura, etc.

A tabela *ecommerce_products* foi construída num *schema* à parte para que não comprometa o desempenho do *Spoke*.

5.3.5.2 Atualização de uma *order*

No caso das *orders*, quando existe uma atualização, a *webshop* correspondente dispara um *webhook* para notificar a API. Aqui, a *order* só é considerada se já tiver sido paga ou se for cancelada¹⁷. De seguida, é normalizada para ficar numa estrutura conhecida e agnóstica e é guardada na tabela *ecommerce_orders*, construída num *schema* à parte da aplicação principal, à semelhança da tabela *ecommerce_products*. Depois de normalizada existem dois processos possíveis, a criação da mesma e de tudo o que dela provém ou o seu cancelamento.

No primeiro caso, começa-se por guardar o *customer* na base de dados, se este ainda não existir. De seguida, são guardadas, ou atualizadas se já existirem, as moradas de entrega e faturação, *shipping* e *billing addresses* respetivamente.

Sempre que é criada uma *order* é necessário que seja criada uma *SO* e respetivos *SO items*. No caso de *ecommerce* é preciso criar uma *order drop*¹⁸ e respetivos *drop items*. Desta forma, estas operações foram aglomeradas numa *data base transaction*, sendo feito um *rollback* caso ocorra algum erro na inserção desta informação.

Por último, é necessário criar o *shipment* e respetivos *packs*. Aqui é consultada uma tabela de referência já criada para determinar qual a transportadora e serviço a utilizar, dependendo do cliente, do país de destino e dos itens a enviar no transporte. Futuramente, esta tabela irá ser substituída por um algoritmo de otimização e categorização de produtos que está a ser desenvolvido pela equipa de BI. Para os *packs*, é necessário criar uma referência de identificação no formato EAN-13.

Existe também a possibilidade do estado da *order* ser *refunding*. Neste caso, devido ao processo ser um pouco mais exigente a nível da base de dados, é criado um *job* que será adicionado à *queue*. Esse *job* irá verificar se a *order* já foi expedida. Em caso afirmativo, deverá alertar um *AM* pois o processo logístico da *HUUB* está completo e é necessário entrar em contacto com o cliente para saber que ação deverá ser tomada. Também é necessário contactar um *AM* caso a *order* não tenha sido expedido mas já esteja tudo preparado para a expedição, isto é *label* e carta de porte criadas, ou caso já existam itens da *order* "*pickados*"¹⁹. No caso de nenhum item ter sido "*pickado*", é necessário cancelar a *SO*, a *drop* e o *shipment*, seguindo-se uma notificação de sucesso.

Todo este processo pode ser visto através do *flowchart* da Figura 5.17.

¹⁷Quando é feito um pedido de *refunding*.

¹⁸Uma *drop* (ou *order drop*) é quando uma *SO* é repartida em vários envios. No caso de *ecommerce* existe apenas uma *drop* pois os produtos existem todos em *stock*. No caso de *wholesale*, é comum uma *SO* ser repartida em duas ou três *drops*.

¹⁹Para que uma *order* seja processada, o primeiro passo é buscar os produtos às estantes pretendidas e "*picka-los*". Este processo é o que despoleta a saída de *stock* de um produto.

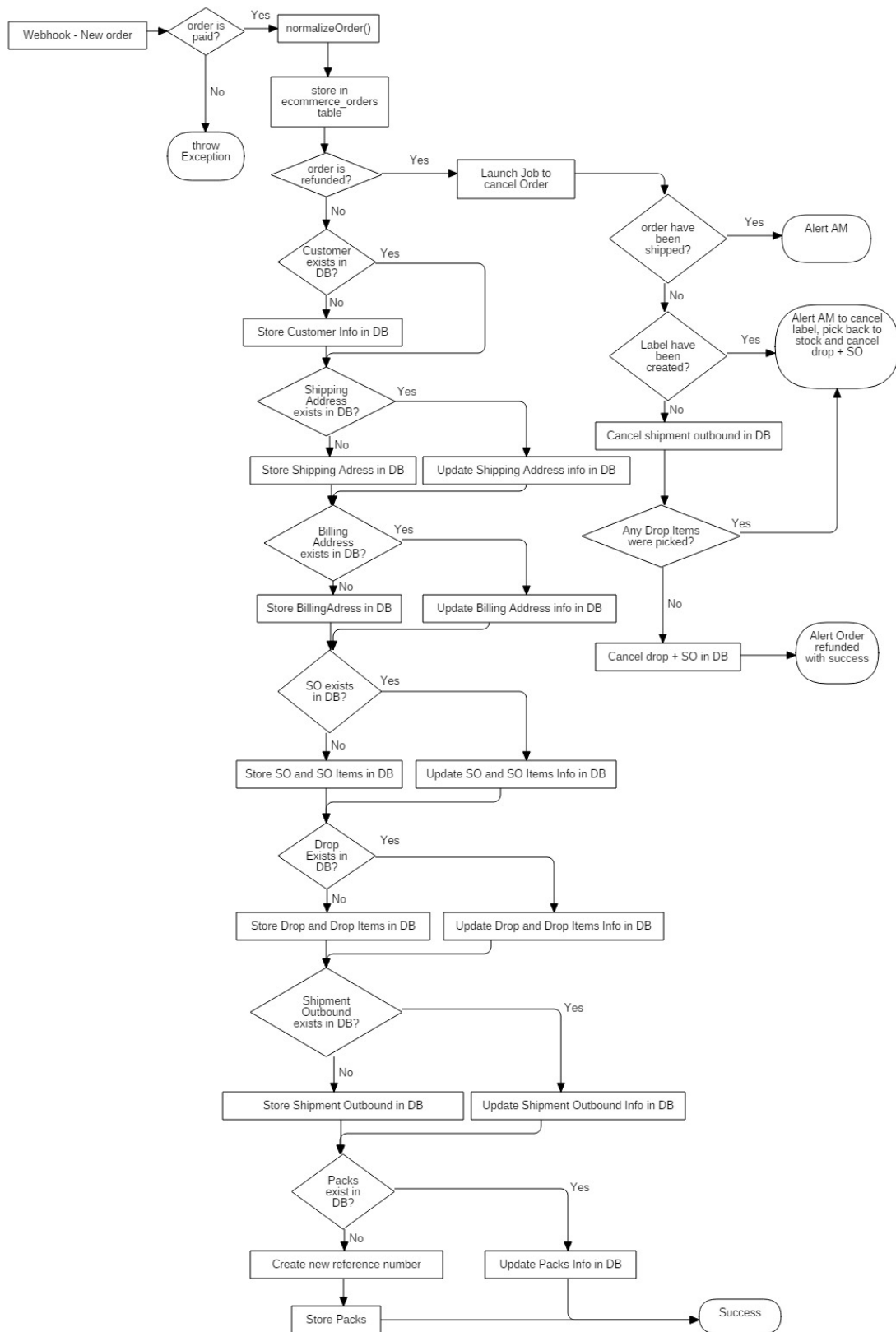


Figura 5.17: Flowchart do processo de atualização de orders via webhook.

Este processo é de extrema importância pois as *orders* em *ecommerce* precisam de ser tratadas rapidamente²⁰. Até ao momento, este processo, como já foi referido, era feito através da utilização de um *cron* que resultava em diversos problemas como perda de informação, elevada sobrecarga do servidor, entre outros. Desta maneira, o processo deixa de ser ativo e passa a ser passivo, sendo realizado em tempo real. Este processo já está também pronto a ser posto em produção, faltando apenas infraestrutura do lado do *Spoke* e ultimar os testes que estão a ser feitos para que tudo corra sem problemas e a integração seja plena.

5.4 Notificação de intervenientes

É importante garantir que alguém seja notificado aquando da ocorrência de certos eventos. Como já foi referido, a plataforma escolhida foi o *Mailgun* pela sua fácil integração com *Laravel* e o *Slack*, pois toda a equipa na *HUUB* o usa no seu dia-a-dia.

Para o *Slack* foi usado um *package* de *composer*. Já para o envio de *emails*, a lógica é semelhante à usada nas classes anteriores onde foram aplicado os *design patterns* *Factory* e *Adapter*. Assim, para cada tipo de *email* a ser enviado foi criada uma classe que descende de outra classe *SuperMailable*, que por sua vez descende de outra classe *Mailable* que já faz parte da estrutura da *framework Laravel*. A estrutura pode ser visível na Figura 5.18:

Na classe *SuperMailable* foram implementados métodos gerais do *email* com funções para escrever quais os endereços de *email* de destino, endereços de *email* em *Carbon Copy* (CC), ficheiros a anexar e endereço de *email* para onde será enviada a resposta.

Já as classes que descendem da *SuperMailable* apenas implementam um método, *sendMail*²¹. Este método é responsável por escolher qual a *view* do *email* a enviar, o assunto do *email* e as variáveis que são passadas.

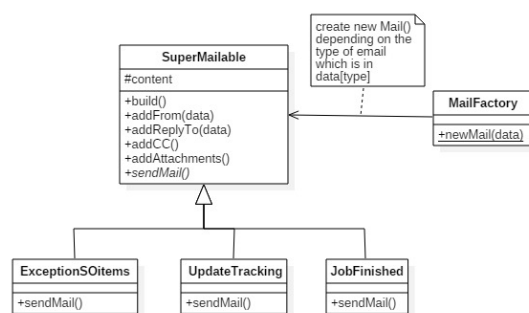


Figura 5.18: Classe *Mail* que trata do envio de *emails*.

²⁰Todas as *orders* feitas antes das 12h serão enviadas no próprio dia. Quando as *orders* são feitas depois das 12h, são enviadas no dia seguinte.

²¹Todas as classes que descendem de *Mailable*, classe existente na *framework* precisam de implementar o método *sendMail*.

Para este processo foram criados dois *endpoints*, um para enviar email e outro para receber *logs* de *email* que falharam por algum motivo, provenientes de um *webhook* disparado pelo *Mailgun*. Estes *endpoints* estão apresentados na Tabela 5.4.

Tabela 5.4: *Endpoints* criados para o processo de notificação dos intervenientes.

Método	Endpoint	Descrição
POST	/api/mail/send	Envia <i>email</i>
POST	/api/webhooks/mail/log	<i>Webhook</i> disparado quando o envio de um <i>email</i> falha

Como o envio de *email* pode ser um processo que demore algum tempo e que, consequentemente, afete o tempo de resposta da *API*, decidiu implementar-se uma lógica de *queue*. Assim, sempre que se pretende enviar um *email*, é criado um *job* que vai, assincronamente, tratar do seu envio.

A gestão da *queue* é feita com a *framework Laravel* que já possui *helpers* para auxiliar e usando uma tabela da *BD*, *log_jobs*, para controlo.

5.4.1 Envio de emails

Como foi referido, foram criados 3 *templates* de *email*, cada um com o seu propósito.

Primeiramente foi criado um email para notificar os *AMs* do término de uma tarefa assíncrona criada por eles, Figura 5.19, como o *import* de todos os produtos de uma loja *online*. Este email é interessante devido ao elevado tempo que algumas destas tarefas demoram até serem concluídas.

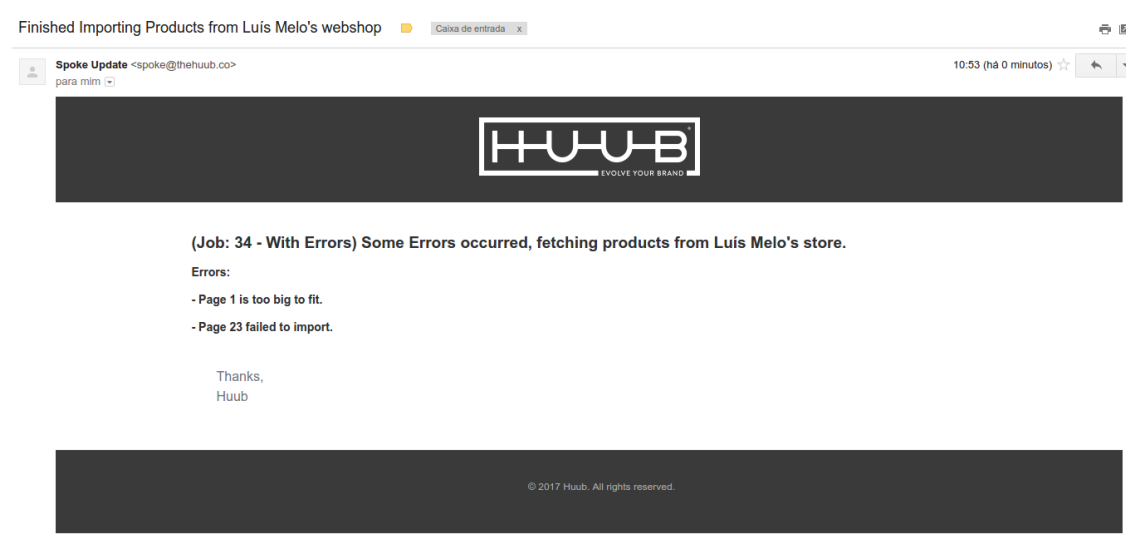


Figura 5.19: *Email* enviado para notificação do término de um *job*.

Foi desenvolvido ainda um *email* específico para notificar também os AM de produtos adicionados a uma *order* que ainda não estão em sistema, Figura 5.20. Como a HUUB trata da logística de todos os produtos dos seus HUUB clients, esses mesmos produtos terão de estar em sistema para que possam ser identificados no armazém. No entanto, como alguns HUUB client possuem lojas *online*, podem haver produtos que tenham sido criados na loja *online* sem que a HUUB tenha sido notificada²². Desta forma, os AMs são notificados mal a *order* é paga, logo, conseguem agir sempre em tempo útil.

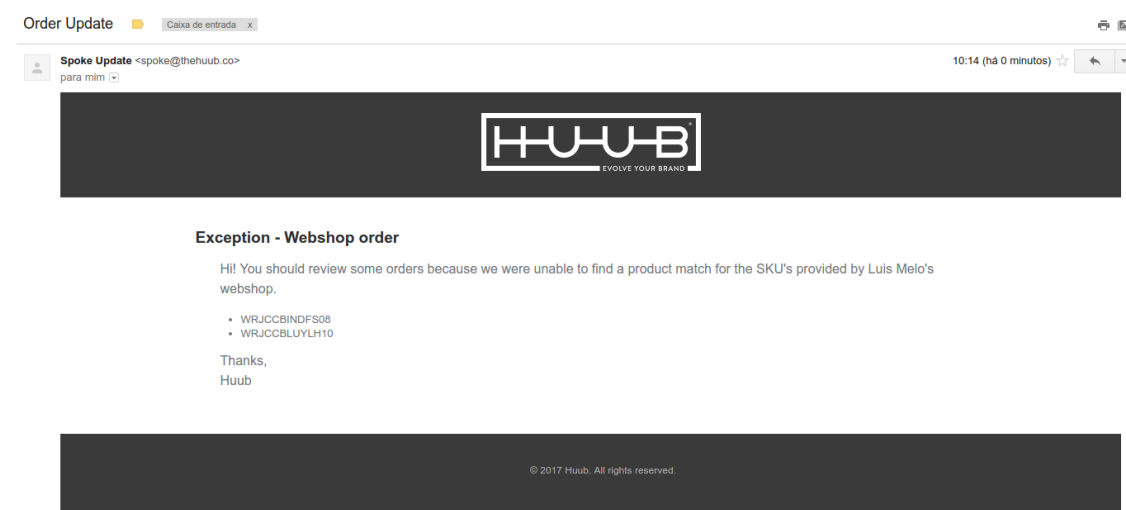


Figura 5.20: Email enviado para notificação de itens de uma *order* que não estão em sistema.

O último *template* criado, e talvez o mais importante, é o de atualização do *status* do *tracking* de uma *order*. Como foi abordado em capítulos anteriores, a HUUB pretende ter o controlo do processo mesmo depois da *order* ser expedida. Desta forma, e aliando os *updates* do *tracking-more*, os estados do *tracking* das *orders* é feito recorrendo a um algoritmo de *machine learning* construído pela equipa de BI. Isto deve-se ao facto de existirem situações que devem ser tratadas como exceções mas que o *trackingmore* não considerava existência de qualquer anomalia. Assim, o *customer* recebe notificação quando a sua *order* sai das instalações da HUUB, no caso de entrar em exceção²³, quando sai do estado de exceção e quando está no estado de *last mile*²⁴, Figura 5.21.

Como já foi referido, construi-se toda a lógica numa perspetiva de escalabilidade, isto é, caso seja necessário acrescentar um novo tipo de email, basta criar um *template* do email a enviar, e uma classe que descenda de *SuperMailable* que implemente um método *sendMail()* que trate das variáveis a usar no *template*.

²²Este fenómeno costuma acontecer no início de *season* quando alguns produtos não são carregados para o *Spoke*

²³Pode ser, por exemplo, falta de documentos, falha na entrega da *order* por o estabelecimento estar fechado, etc.

²⁴O estado de *last mile* é o último estado antes da *order* ser entregue

Desta forma, para qualquer *email* que se pretenda enviar terá de ser feito um pedido a uma só rota, em que o *body* deverá conter o destinatário (*mail_to*, o tipo de email a enviar (*type*) e um *array* de variáveis que serão necessárias para construir o *email* (*options*).

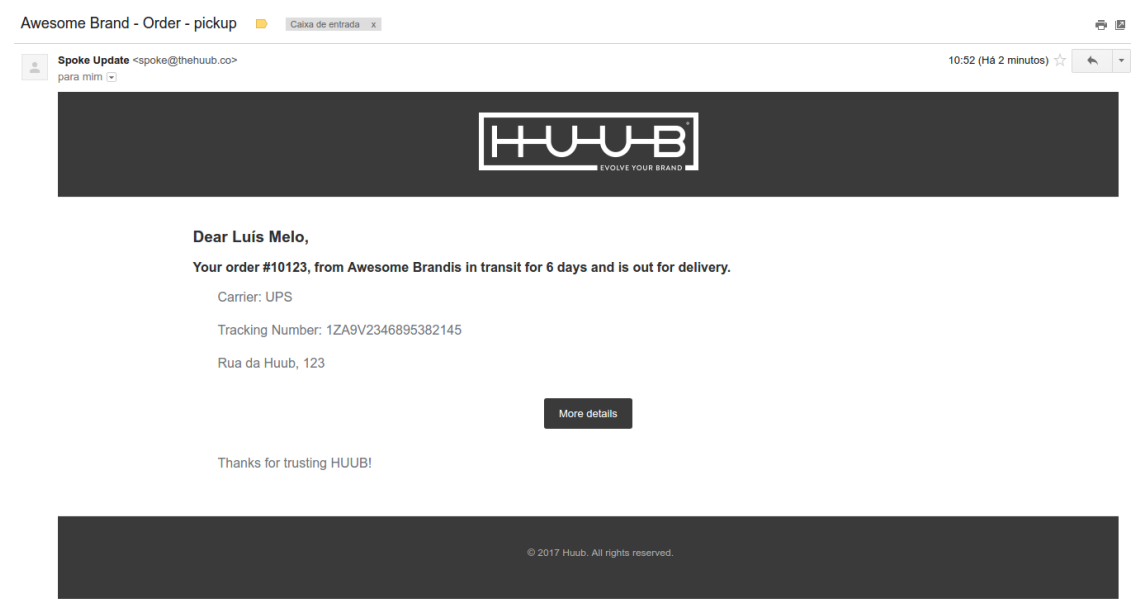
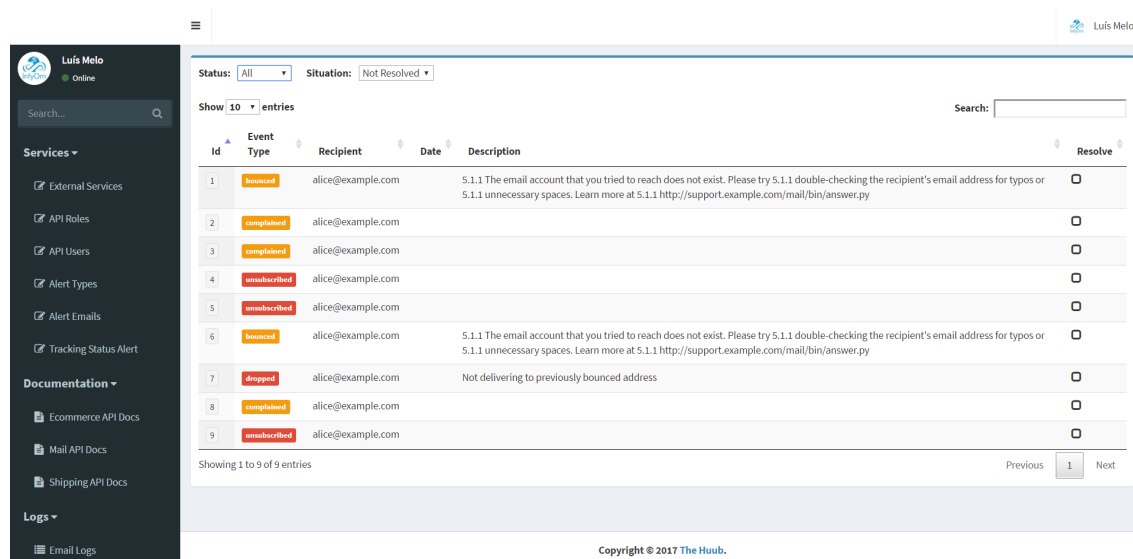


Figura 5.21: *Email* enviado para notificar *updates* ao *tracking* de uma *order*.

5.4.2 Log de emails

Para que os *emails* sejam de facto enviados, é necessário toda uma configuração no lado do *Mailgun*. Depois de estar configurado, é possível ver as estatísticas dos envios de email e criar *webhooks* para cada estado do email. Os estados do *mailgun* são: *delivered*, *dropped*, *bounced*, *spam complaints*, *unsubscribed*, *clicked* e *opened*.

Para estes estados, associou-se um *URL* para notificação via *webhook*, aos estados mais críticos sendo eles *dropped*, *bounced*, queixas de *spam* e *unsubscribed*. Assim, sempre que algo de anormal acontece no envio do *email* que faça com que o estado do mesmo seja um dos anteriores, essa informação é recebida e guardada na tabela *email_logs* para posterior análise. Essa tabela é possível consultar, de forma visual, na plataforma de gestão administrativa montada para o efeito, Figura 5.22.



Id	Event Type	Recipient	Date	Description	Resolve
1	bounced	alice@example.com		5.1.1 The email account that you tried to reach does not exist. Please try 5.1.1 double-checking the recipient's email address for typos or 5.1.1 unnecessary spaces. Learn more at 5.1.1 http://support.example.com/mail/bin/answer.py	☐
2	complained	alice@example.com			☐
3	complained	alice@example.com			☐
4	unsubscribed	alice@example.com			☐
5	unsubscribed	alice@example.com			☐
6	bounced	alice@example.com		5.1.1 The email account that you tried to reach does not exist. Please try 5.1.1 double-checking the recipient's email address for typos or 5.1.1 unnecessary spaces. Learn more at 5.1.1 http://support.example.com/mail/bin/answer.py	☐
7	dropped	alice@example.com		Not delivering to previously bounced address	☐
8	complained	alice@example.com			☐
9	unsubscribed	alice@example.com			☐

Showing 1 to 9 of 9 entries

Previous 1 Next

Figura 5.22: Dashboard de *emails* criado para resolução de anomalias e exceções.

5.5 Conclusão

Durante o presente capítulo, entrou-se ao detalhe de como se implementou toda a solução. Aqui estão presentes os resultados obtidos que vão de encontro aos requisitos impostos no início do projeto. Estão também apresentados diagramas de sequência para identificar alguns processos, um *flow chart*, bem como diversos diagramas de classes para expor a arquitetura desenvolvida. Para mais informação, e como foi referido anteriormente, a documentação está disponível no anexo B.

No Capítulo 6 será feita uma reflexão de tudo aquilo que foi desenvolvido e qual vai ser o trabalho futuro.

Capítulo 6

Conclusões e trabalho futuro

6.1 Satisfação dos objetivos

Com o fim do projeto, e em jeito de retrospectiva, é possível avaliar que a solução implementada, para cada uma das três partes, é uma solução robusta, funcional e, acima de tudo, modular. A validação do *software* foi feita usando o *Postman* [38] para testar os *endpoints* criados e, mais tarde, com a equipa de *AMs* para garantir que todo o processo estava correto. Foram também realizados testes de integração com a ferramenta *phpunit* e *xdebug*, no entanto, e segundo a documentação, o *branch cover* ainda não está disponível, "*The Opcode Coverage, Branch Coverage, and Path Coverage software metrics are not yet supported by PHP_CodeCoverage*".

Do projeto desenvolvido, a parte de automatização do *shipping* e de notificação dos intervenientes está em modo produção e a ser usado recorrentemente na *HUUB* desde maio. A integração com as lojas de *ecommerce* encontra-se neste momento num servidor de *staging*, estando a ser testado para garantir que está totalmente operacional.

Todos os objetivos e requisitos impostos, presentes nas Tabelas 3.1 e 3.2, foram cumpridos.

6.2 Conclusões e trabalho futuro

Todo este projeto contribuiu para dar início a uma nova abordagem de desenvolvimento de *software* na *HUUB*, *SOA*, enquanto que se automatizaram algumas tarefas internas da empresa, tendo sempre em vista boas práticas de engenharia de *software* e um paradigma de *OOP*.

Foi então desenvolvido um módulo de automatização do processo de *shipping* que visa cobrir de ponta a ponta todo este processo, isto é, obter *rates* de transporte, comprar a *label* de transporte, usando a plataforma *Easypost*, e ter, do lado da *HUUB*, informação do *tracking* da encomenda em tempo real, usando a plataforma *Trackingmore*. As duas primeiras fases deste sistema englobam

integração com a *UPS*, *TNT* e *DHL*. Já a parte do *tracking* integra com todas os *carriers* excepto *Nacex* e *Fema*. Este processo já está em produção e em uso pelos *AMs* estando, por isso, concluída esta parte do projeto.

A segunda parte foi a integração com as lojas *online* de *ecommerce*. Pela pesquisa efetuada, as *mashup* de *APIs* existentes não eram vantajosas para serem implementadas, quer seja pela falta de *features* consideradas essenciais, como os *webhooks*, mas também pelo preço. Desta forma, implementou-se uma solução *mashup* que integra com o *Woocommerce* e com o *Shopify*, da qual se consegue obter informação sobre produtos, *orders* e *customers*, gerir *webhooks*, atualizar *stock* de um produto na loja *online*, fazer *fulfilment* de uma *order* e receber, via *webhook* atualizações de *orders*, isto é, quando uma *order* é paga ou cancelada. Desta forma, e tendo este processo sido desenvolvido de forma a ser possível a inclusão de uma nova loja *online* com relativa facilidade, chega-se à conclusão que todos os objetivos foram cumpridos.

Como trabalho futuro, está prevista a inclusão de mais lojas *online*, como a *Tictail* e o *Magento*, de novos clientes. Adicionalmente, seria interessante explorar um pouco mais a sincronização dos produtos (semelhante com a que foi implementada para as *orders*), pois, atualmente, a estrutura da *HUUB* e as grandes diferenças no esquema de dados das diferentes lojas *online* tornam complicado que isso possa acontecer.

A terceira e última parte, notificação dos intervenientes foi realizada com recurso ao *Mailgun* para o envio de *emails* e a uma biblioteca que integra com o *Slack*. O grande objetivo era, portanto, poder notificar certos indivíduos dependendo de mudanças que ocorram. Até ao momento foram criados três tipos de alertas: notificação do fim de um *job*, notificação de criação de uma *order online* com itens que não estão em sistema e notificação do *tracking* de uma encomenda. Foi também implementado um sistema de *logs* para quando os *emails* falham. Dito isto, conclui-se que os objetivos foram todos alcançados, estando já em produção. Como trabalho futuro, está a inclusão de outros tipos de alerta que se achem necessários. Esta inclusão deverá ser realizada com o mínimo de trabalho possível pois o *software* foi montado tendo sempre esse objetivo em mente.

Para além de tudo isto, foi desenvolvido um painel de controlo administrativo para gerir *API keys*, bem como uma documentação que permita a qualquer *developer* da *HUUB* usar os recursos desenvolvidos, estando implementado de tal forma que se consiga acrescentar novas *APIs* no futuro.

Referências

- [1] Channel 9. Juval Lowy on Microservices [online]. URL: <https://channel9.msdn.com/Blogs/raw-tech/Juval-Lowy-On-Microservices>.
- [2] Christoph Schroth e Till Janner. Web 2.0 and SOA: Converging Concepts Enabling the Internet of Services. *IT Professional*, 9(3):36–41, 2007. URL: <https://www.alexandria.unisg.ch/37270/1/036-041.pdf>.
- [3] Lori MacVittie. Web 2.0: Integration, APIs, and Scalability [online]. 2008. Acedido a: 2017-01-30. URL: <https://devcentral.f5.com/articles/web-20-integration-apis-and-scalability>.
- [4] Trevor Richardson. Microservices in Practice: The Positive – Source Allies Blog [online]. 2015. Acedido a: 2017-03-14. URL: <http://blogs.sourceallies.com/2015/11/microservices-in-practice-the-positive/>.
- [5] Benjamin Wootton. Microservices - Not a free lunch! - High Scalability - [online]. 2014. Acedido a: 2017-05-13. URL: <http://highscalability.com/blog/2014/4/8/microservices-not-a-free-lunch.html>.
- [6] James Lewis e Martin Fowler. Microservices [online]. 2014. Acedido a: 2017-03-14. URL: <https://martinfowler.com/articles/microservices.html>.
- [7] Darlene Fichter. *What is a Mashup*. Nicole C. Engard, 2010. URL: <http://scholar.google.com/scholar?hl=en{%&}btnG=Search{%&}q=intitle:What+Is+a+Mashup+?{#}1>.
- [8] Chris; Mills. Third party APIs - Learn web development | MDN [online]. 2017. Acedido a: 2017-06-20. URL: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side{ }web{ }APIs/Third{ }party{ }APIs>.
- [9] Mariell Längle. XML vs. JSON [online]. 2015. Acedido a: 2017-01-24. URL: <http://mobdok.de/2015/05/xml-vs-json/>.
- [10] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Tese de doutoramento, UNIVERSITY OF CALIFORNIA, IRVINE, 2000. URL: <http://www.ics.uci.edu/{~}fielding/pubs/dissertation/top.htm>, doi:10.1.1.91.2433.
- [11] Terry Hunt. Understanding REST [online]. 1990. Acedido a: 2017-01-24. URL: <https://spring.io/understanding/REST>.
- [12] Samepage. Team Collaboration & Messaging [online]. Acedido a: 2017-01-31. URL: <https://www.samepage.io/>.

- [13] licobo GmbH. CloudRail - API Integration Solution [online]. 2016. Acedido a: 2017-01-31. URL: <https://cloudrail.com/>.
- [14] Collin Fair. Tapiriik - Keeps your fitness in sync [online]. Acedido a: 2017-03-15. URL: <https://tapiriik.com/>.
- [15] TrackingMore. TrackingMore [online]. Acedido a: 2017-01-26. URL: <http://www.trackingmore.com/>.
- [16] AfterShip. AfterShip: Shipment Tracking Platform [online]. Acedido a: 2017-01-26. URL: <https://www.aftership.com/>.
- [17] Aftership. Aftership SDK - PHP [online]. Acedido a: 2017-01-31. URL: <https://github.com/AfterShip/aftership-sdk-php>.
- [18] Postmen. Postmen [online]. Acedido a: 2017-01-26. URL: <https://www.postmen.com/>.
- [19] Circle. Circle [online]. Acedido a: 2017-01-26. URL: <https://www.circle.us/>.
- [20] Shippo. Shippo: The Shipping API For Ecommerce [online]. 2013. Acedido a: 2017-01-26. URL: <https://goshippo.com/>.
- [21] ShipHawk. ShipHawk [online]. Acedido a: 2016-11-20. URL: <https://shiphawk.com/>.
- [22] EasyPost. EasyPost: The Simple Shipping API [online]. Acedido a: 2017-01-26. URL: <https://www.easypost.com/>.
- [23] Ecomdash. Ecomdash [online]. Acedido a: 2017-02-18. URL: <https://www.ecomdash.com/>.
- [24] ChannelApe. ChannelApe [online]. Acedido a: 2017-02-18. URL: <https://www.channelape.com/>.
- [25] MagneticOne. API2Cart [online]. Acedido a: 2017-01-25. URL: <https://www.api2cart.com/>.
- [26] WordPress Foundation. WordPress [online]. 2003. Acedido a: 2017-01-25. URL: <https://wordpress.org/>.
- [27] Shopify. Shopify [online]. Acedido a: 2016-11-18. URL: <https://www.shopify.com/>.
- [28] Inc. Varian. Magento [online]. Acedido a: 2016-11-18. URL: <https://magento.com/>.
- [29] The Rocket Science Group. MailChimp [online]. Acedido a: 2016-11-25. URL: <https://mailchimp.com/>.
- [30] E-goi Portugal. E-goi [online]. Acedido a: 2016-11-25. URL: <https://www.e-goi.pt/pt/>.
- [31] Rackspace. Mailgun [online]. Acedido a: 2017-01-26. URL: <https://www.mailgun.com/>.
- [32] SparkPost. SparkPost: Email Delivery Services for Developers [online]. Acedido a: 2017-01-27. URL: <https://www.sparkpost.com/>.
- [33] Auth0 e Okta. OAuth Community Site [online]. Acedido a: 2017-03-25. URL: <https://oauth.net/>.

- [34] Auth0. JSON Web Token Introduction - jwt.io [online]. Acedido a: 2017-03-25. URL: <https://jwt.io/introduction/>.
- [35] Erich Gamma, Richard Helm, Ralph Johnson, e John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 21 edição, 2000.
- [36] Rosetta Code. Levenshtein distance - Rosetta Code [online]. Acedido a: 2017-05-30. URL: <http://rosettacode.org/wiki/Levenshtein{ }distance>.
- [37] PostgreSQL. PostgreSQL: Documentation: 9.1: fuzzystmatch [online]. Acedido a: 2017-05-30. URL: <https://www.postgresql.org/docs/9.1/static/fuzzystmatch.html>.
- [38] Postdot Technologies. Postman | Supercharge your API workflow [online]. Acedido a: 2017-06-10. URL: <https://www.getpostman.com/>.

Anexo A

Normalização de classes

Em primeiro lugar, temos a aplicação do *design pattern Factory* para que, em qualquer parte da aplicação seja criado um filho da classe *Address*, dependendo da loja *online* do cliente:

```
<?php
namespace App\Classes\Addresses;

class AddressFactory
5 {
    // $type = Billing / Shipping
    public static function newAddress($store, $json, $customer_id,
        $type)
    {
        switch ( $store) {
10         case 'woocommerce': return new WoocommerceAddress($json,
            $customer_id, $type);
            case 'shopify': return new ShopifyAddress($json,
                $customer_id, $type);
        }
        return null;
15 }
?>
```

A classe *parent* tem um conjunto de métodos abstratos que terão de ser implementados nos seus filhos bem como um método de normalização que chama essas funções abstratas.

```
<?php
namespace App\Classes\Addresses;

4 use App\Models\AppAddressType;
```

```
abstract class Address
{
    protected $client;
    protected $json;
    protected $customer_id;
    protected $type;

    function __construct($json, $customer_id, $type){
        $this->json      = (array)json_decode($json);
        $this->customer_id = $customer_id;
        $this->type       = $type;
    }

    public function normalizeAddress(){
        $result = array(
            'customer_id'    => $this->customer_id,
            'address'        => $this->getAddress(),
            'zip'            => $this->getZip(),
            'city'           => $this->getCity(),
            'country_id'     => $this->getCountryId(),
            'address_type_id' => $this->getAddressTypeId(),
            'deliver_to'     => $this->getDeliverTo(),
            'address_state'  => $this->getAddressState(),
        );

        // delete all fields which are null
        $result = array_filter($result, function($val){
            return !is_null($val);
        });
        return $result;
    }

    public function getJSON(){
        return $this->json;
    }

    public function getAddressTypeId(){
        return AppAddressType::where('name', $this->type)->pluck('id')
            [0];
    }

    public abstract function getAddress();
    public abstract function getZip();
    public abstract function getCity();
    public abstract function getCountryId();
}
```

```

    public abstract function getDeliverTo();
    public abstract function getAddressState();
}
?>

```

Já as classes filhas, são bastante semelhantes. O seu trabalho é implementar os métodos abstratos do *parent*. Esses métodos têm a função de recolher a informação pretendida dependendo da estrutura de cada loja *online*.

```

1  <?php
    namespace App\Classes\Addresses;

    use App\Models\AppCountry;

6  class ShopifyAddress extends Address
    {
        public function getAddress(){
            $shipAddr = '';
            if ( key_exists('address1', $this->json))
11             $shipAddr .= $this->json['address1'];
            if ( !is_null($this->json['address2']) )
                $shipAddr .= ',_' . $this->json['address2' ];

            return $shipAddr;
16     }

        public function getZip(){
            if ( key_exists('zip', $this->json))
                return $this->json['zip'];
21
            throw new \Exception('Zip_code_is_super_important._Please,_
                provide_one. ');
        }

        public function getCity(){
26             return ( key_exists('city', $this->json) && $this->json['city']
                != null ) ?
                $this->json['city'] :
                null;
        }

31     public function getCountryId(){
            $countryId = AppCountry::where('code', $this->json['
                country_code' ]);
            // 250 is the default for Unknown

```

```

        return ($countryId->count() == 1) ?
            $countryId->pluck('id')[0] :
36         250;
    }

    public function getDeliverTo(){
        // If complete name is available
41         if ( key_exists('name', $this->json))
            return $this->json['name'];

        // If it's not available
        $name = '';
46         if ( key_exists('first_name', $this->json))
            $name .= $this->json['first_name'] . '_';
        if ( key_exists('last_name', $this->json))
            $name .= $this->json['last_name'];

51         return $name;
    }

    public function getAddressState(){
        if ( key_exists('province', $this->json) && $this->json['
            province'] != null)
56         return $this->json['province'];

        if ( key_exists('province_code', $this->json) && $this->json['
            province_code'] != null)
            return $this->json['province_code'];

61         return null;
    }
}
?>

```

```

1  <?php
    namespace App\Classes\Addresses;

    use App\Models\AppCountry;

6  class WoocommerceAddress extends Address
    {
        public function getAddress(){
            $shipAddr = '';
            if ( key_exists('address_1', $this->json))
11         $shipAddr .= $this->json['address_1'];
    }
}

```

```
        if ( $this->json['address_2'] != null )
            $shipAddr .= ',_' . $this->json['address_2'];

        return $shipAddr;
16    }

    public function getZip(){
        if ( key_exists('postcode', $this->json))
            return $this->json['postcode'];
21    throw new \Exception('Zip_code_is_super_important._Please,_
        provide_one.');
```

```
    }

    public function getCity(){
        return ( key_exists('city', $this->json) && $this->json['city']
26            != null ) ?
            $this->json['city'] :
            null;
    }

    public function getCountryId(){
31    $countryId = AppCountry::where('code', $this->json['country']);
        // 250 is the default for Unknown
        return ($countryId->count() == 1) ?
            $countryId->pluck('id')[0] :
            250;
36    }

    public function getDeliverTo(){
        // If complete name is available
        if ( key_exists('name', $this->json))
41    return $this->json['name'];

        $name = '';
        if ( key_exists('first_name', $this->json))
            $name .= $this->json['first_name'] . '_';
46    if ( key_exists('last_name', $this->json))
            $name .= $this->json['last_name'];

        return $name;
51    }

    public function getAddressState() {
        return ( key_exists('state', $this->json) && $this->json['state
        ''] != null) ?
```

```
56         $this->json['state'] :  
            null;  
    }  
?>
```

Anexo B

Documentação gerada

De seguida será anexada a documentação produzida para a API desenvolvida. Esta documentação foi feita no formato *markdown* (ficheiro com extensão .md), sendo posteriormente convertida em pdf para ser aqui inserida. Por esta razão, existe alguma má formatação, fruto do compilador usado.

A mesma documentação está também disponível na Internet, no *website*:

<https://paginas.fe.up.pt/~ee12103/dissertacao>

Shipping and Tracking API

1. [Create New Order](#)
2. [Buy an Order](#)
3. [Track a new Shipment](#)
4. [Get Tracking Details](#)
5. [Update Tracking Details](#)
6. [Fetch Tracking Details](#)
7. [Objects](#)

Create New Order

POST `/api/shipping/new/key={api_key}`

This endpoint has the purpose of create a new order (for one or more shipments) and to retrieve the rates which will be use to make a forecast. It takes a JSON object containing sender, receiver, shipment and billing information. Each element of the shipment array is a package and contains its dimensions, weight, and detail about the items that make it up.

- **Users allowed**

- Huub

- **Data Params**

Parameter	Function	Variable	Required
from_address	Sender's addresses (Huub)	Address Object	yes
to_address	Receiver's address	Address Object	yes
shipments	Information about shipment and packs	array of Shipments Objects	yes
billing	Information about billing	Billing Object	yes

- **Sample Request Body:**

```
{
  "from_address": {
    "name": "Pedro Santos",
    "street1": "Rua Joaquim Dias Salgueiro",
    "street2": "130",
    "city": "Maia",
    "state": "PO",
    "zip": "4470-777",
    "country": "PT",
    "company": "The Huub",
    "phone": "(+351) 223 290 967",
    "email": ""
  },
  "to_address": {
    "name": "",
    "street1": "Plaza de la cebada",
    "street2": "5",
    "city": "Madrid",
    "state": "",
    "zip": "28005",
    "country": "ES",
    "company": "estanco de la latina",
    "phone": "415-123-4567",
    "email": "",
    "residential": "false"
  },
  "shipments": [
    {
      "parcel": {
        "typology": "CBox 60x40x40",
        "weight": 20
      },
      "customs_info": {
        "eel_pfc": "NOEEI 30.37(a)",
        "contents_type": "merchandise",
        "contents_explanation": "",
        "restriction_type": "none",
        "non_delivery_option": "return",
        "customs_items": [
          {
            "description": "Yellow shirts",
            "quantity": 40,
            "weight": 5,
            "value": 200,
```

```

        "hs_tariff_number": "654322",
        "origin_country": "PT",
        "currency": "EUR"
    }
]
}
},
{
    "parcel": {
        "typology": "CBox 60x40x40",
        "weight": 10
    },
    "customs_info": {
        "eel_pfc": "NOEEI 30.37(a)",
        "contents_type": "merchandise",
        "contents_explanation": "",
        "restriction_type": "none",
        "non_delivery_option": "return",
        "customs_items": [
            {
                "description": "Green shirts",
                "quantity": 30,
                "weight": 7,
                "value": 100,
                "hs_tariff_number": "123456",
                "origin_country": "PT",
                "currency": "EUR"
            }
        ]
    }
}
],
"billing": {
    "receiver": true,
    "bill_account": "",
    "bill_country": "",
    "bill_zip": "",
    "incoterm": "DAP"
}
}

```

- **Success Response:**

```

{

```

```

"code": 200,
"message": "Success.",
"error_messages": [
  {
    "carrier": "UPS",
    "type": "rate_message",
    "message": "UPS multi-shipment rate includes this shipment."
  }
],
"rates": [
  {
    "carrier": "TNTExpress",
    "service": "1200Express",
    "rate": "138.19",
    "currency": "EUR"
  },
  [...]
  {
    "carrier": "UPS",
    "service": "ExpressPlus",
    "rate": "610.76",
    "currency": "EUR"
  }
]
}

```

- **Error Responses:**

Code	Message
400	Bad Request. Please send both shipping and receiving addresses
400	Bad Request. Please specify shipment information
400	Bad Request. Please specify billing information
400	Bad Request. All shipments need to have dimensions and detailed items information.
400	Bad request. You need to specify either dimensions or package typology.
403	Forbidden. Error when creating order.
502	Bad Gateway. No Rates found in EasyPost.

Buy an Order

POST `/api/shipping/buy/key={api_key}`

This endpoint has the purpose of buy an order (consisting of one or more shipments). It takes a JSON object containing shipment info such as sender's and receiver's address and packages specifications. You need to specify in that JSON what service you want to use (you can get the services and corresponding rates using the endpoint above ([Create New Order](#))). It will return information about the shipment as well as documents such as labels and invoices. It will also store those documents in dropbox under **root/{client}/{sales_order}**

- **Users allowed**

- Huub

- **Data Params**

Parameter	Function	Variable	Required
shipment_outbound_id	identify the shipment	integer	yes
rate	selected rate	Rate Object	yes
client	client associated to this shipment	string	yes
sales_order	sales order code	string	yes
order	order to buy	Order Object	yes

- **Sample Request Body:**

```
{
  "shipment_outbound_id": 1,
  "rate": {
    "price": 80,
    "carrier": "TNTEexpress",
    "service": "EconomyExpress"
  }
}
```

```
},
"client": "Twenty Five",
"sales_order": "SO#2017-1000005",
"order": {
  "from_address": {
    "name": "Pedro Santos",
    "street1": "Rua Joaquim Dias Salgueiro",
    "street2": "130",
    "city": "Maia",
    "state": "PO",
    "zip": "4470-777",
    "country": "PT",
    "company": "The Huub",
    "phone": "(+351) 223 290 967",
    "email": ""
  },
  "to_address": {
    "name": "",
    "street1": "218 Wallabout street",
    "street2": "",
    "city": "Brooklyn",
    "state": "NY",
    "zip": "112065418",
    "country": "USA",
    "company": "Clara Childrens Boutique",
    "phone": "415-123-4567",
    "email": "",
    "residential": true
  },
  "shipments": [
    {
      "shipment_outbound_unit_id": 1,
      "parcel": {
        "typology": "CBox 60x40x40"
        "weight": 15
      },
      "customs_info": {
        "eel_pfc": "NOEEI 30.37(a)",
        "contents_type": "merchandise",
        "contents_explanation": "",
        "restriction_type": "none",
        "non_delivery_option": "return",
        "customs_items": [
          {
            "description": "Yellow shirts",
```

```

        "quantity": 40,
        "weight": 5,
        "value": 200,
        "hs_tariff_number": "654322",
        "origin_country": "PT",
        "currency": "EUR"
    }
]
}
},
{
    "shipment_outbound_unit_id": 2,
    "parcel": {
        "typology": "CBox 40x40x40"
        "weight": 10
    },
    "customs_info": {
        "eel_pfc": "NOEEI 30.37(a)",
        "contents_type": "merchandise",
        "contents_explanation": "",
        "restriction_type": "none",
        "non_delivery_option": "return",
        "customs_items": [
            {
                "description": "Green shirts",
                "quantity": 30,
                "weight": 7,
                "value": 100,
                "hs_tariff_number": "123456",
                "origin_country": "PT",
                "currency": "EUR"
            }
        ]
    }
}
],
"billing": {
    "receiver": true,
    "bill_account": "",
    "bill_country": "",
    "bill_zip": "",
    "incoterm": "DAP"
}
}
}

```

- **Success Response:**

```
{
  "code": 200,
  "message": "Success.",
  "shipments": [
    {
      "shipment_outbound_unit_id": 1,
      "parcel": {
        "length": 60,
        "width": 40,
        "height": 40,
        "weight": 15
      },
      "label": "https://easypost-files.s3-us-west-2.amazonaws.com/files/postage_label/20170306/5cdc5899fd3549cdbac66c20eb2d1121.pdf",
      "waybill": "GE337655089PT",
      "forms": [
        "https://easypost-files.s3-us-west-2.amazonaws.com/files/form/20170306/db653866f3fc4853b8585ffd09b872f3.pdf"
      ]
    },
    {
      "shipment_outbound_unit_id": 2,
      "parcel": {
        "length": 40,
        "width": 40,
        "height": 40,
        "weight": 10
      },
      "label": null,
      "waybill": "GE337655089PT",
      "forms": [
        "https://easypost-files.s3-us-west-2.amazonaws.com/files/form/20170306/db653866f3fc4853b8585ffd09b872f3.pdf"
      ]
    }
  ],
  "huub_waybill": "HUUB1004537193",
  "carrier_waybill": "GE337744994PT",
  "selected_rate": {
```

```
"carrier": "TNTExpress",  
"service": "1200Express",  
"price": "138.19",  
"currency": "EUR"  
}  
}
```

- **Error Responses:**

Code	Message
400	Bad Request. You need to specify the shipment (use shipment_outbound_id).
400	Bad Request. Please send both client name and SO number.
400	Bad Request. Please send shipment outbound unit id.
400	Bad Request. In order to buy an order you should give Carrier, Service and the price from forecast.
400	Bad Request. Please send both shipping and receiving addresses
400	Bad Request. Please specify shipment information
400	Bad Request. All shipments need to have dimensions and detailed items information.
403	Forbidden. Error when creating order.
406	Not Acceptable. Carrier and Service not found.
406	Not Acceptable. New price too high. Please, review rates and shipment info.
406	Not Acceptable. New price too low. Please, review rates and shipment info.
406	Not acceptable. Label could not be purchased.
406	Not acceptable. Tracking not inserted.

Track a new Shipment

POST /api/shipping/track/key={api_key}

This endpoint serves to register a new shipment and to track it.

- **Users allowed**

- Huub

- **Data Params**

Parameter	Function	Variable	Required
waybill	Tracking Code	string	yes
carrier	Carrier's Code	string	yes
shipment_outbound_id	Id in Huub's DB	integer	yes
title	Name to Identify the shipment	string	no
customer_name	For trackingmore automatic email	string	no
customer_email	For trackingmore automatic email	string	no
order_id	For trackingmore automatic email	string	no
lang	Language of the customer	string	no

- **Sample Request Body:**

```
{
  "waybill": "1ZA9V2340496274174",
  "carrier": "ups",
  "title": "test email",
  "customer_name": "Name of The customer",
  "customer_email": "tfsilva@huub.co",
  "order_id": 12345,
  "lang" : "PT",
  "shipment_outbound_id": 4
}
```

- **Success Response:**

```
{
  "code": 200,
  "message": "Successfully created new tracking"
}
```

- **Error Responses:**

Code	Message
400	Bad Request. Please send waybill and carrier's name.
400	Bad Request. Please send shipment outbound id
400	Bad Request. Please send a valid Carrier name.
400	Bad Request. Please send a Carrier which has TrackingMore integration.
403	Forbidden. Error creating tracking item on TrackingMore.
403	Forbidden. Error creating tracking item on DB.

Get Tracking Details

GET `/api/shipping/track/{carrier_code}/{tracking_number}`

This endpoint serves to get the data of the shipment registered under {tracking_number} waybill

- **Users allowed**

- No restrictions

- **URL Params**

Parameter	Function	Variable	Required
carrier_code	Code of Carrier (stored in Huub's DB)	string	yes
tracking_number	Waybill of the shipment	string	yes

- **Response**

/api/shipping/track/ups/1ZA9V2340496274174

```
{
  "code": 200,
  "message": "Success",
  "tracking_info": {
    "id": 434,
    "carrier": "ups",
    "waybill": "1ZA9V2340496274174",
    "shipment_outbound_id": 4,
    "status_id": 3,
    "days_in_transit": 4,
    "estimate_delivery_date": null,
    "summary": "{\\"origin_info\\":[{\\"Date\\":\\"2017-05-07
15:51\\",\\"StatusDescription\\":\\"ARRIVAL
SCAN\\",\\"Details\\":\\"BOTANY,AU\\"},{\\"Date\\":\\"2017-05-07
12:22\\",\\"StatusDescription\\":\\"THE PACKAGE IS AWAITING CLEARING
AGENCY REVIEW. \\\\/ YOUR PACKAGE WAS RELEASED BY THE CLEARING
AGENCY.\\",\\"Details\\":null},{\\"Date\\":\\"2017-05-07
12:22\\",\\"StatusDescription\\":\\"YOUR PACKAGE IS AT THE CLEARING
AGENCY AWAITING FINAL RELEASE. \\\\/ YOUR PACKAGE WAS RELEASED BY THE
CLEARING AGENCY.\\",\\"Details\\":null},{\\"Date\\":\\"2017-05-07
07:48\\",\\"StatusDescription\\":\\"YOUR PACKAGE IS AT THE CLEARING
AGENCY AWAITING FINAL RELEASE.\\",\\"Details\\":null},
{\\"Date\\":\\"2017-05-06 09:58\\",\\"StatusDescription\\":\\"DEPARTURE
SCAN\\",\\"Details\\":\\"HONOLULU,HI,US\\"},{\\"Date\\":\\"2017-05-06
07:39\\",\\"StatusDescription\\":\\"ARRIVAL
SCAN\\",\\"Details\\":\\"HONOLULU,HI,US\\"},{\\"Date\\":\\"2017-05-06
04:49\\",\\"StatusDescription\\":\\"DEPARTURE
SCAN\\",\\"Details\\":\\"LOUISVILLE,KY,US\\"},{\\"Date\\":\\"2017-05-05
12:00\\",\\"StatusDescription\\":\\"ARRIVAL
SCAN\\",\\"Details\\":\\"LOUISVILLE,KY,US\\"},{\\"Date\\":\\"2017-05-05
10:07\\",\\"StatusDescription\\":\\"DEPARTURE
SCAN\\",\\"Details\\":\\"PHILADELPHIA,PA,US\\"},{\\"Date\\":\\"2017-05-05
07:59\\",\\"StatusDescription\\":\\"ARRIVAL
SCAN\\",\\"Details\\":\\"PHILADELPHIA,PA,US\\"},{\\"Date\\":\\"2017-05-05
05:48\\",\\"StatusDescription\\":\\"DEPARTURE
SCAN\\",\\"Details\\":\\"KOELN (COLOGNE),DE\\"},{\\"Date\\":\\"2017-05-05
00:22\\",\\"StatusDescription\\":\\"ARRIVAL SCAN\\",\\"Details\\":\\"KOELN
(COLOGNE),DE\\"},{\\"Date\\":\\"2017-05-04
20:50\\",\\"StatusDescription\\":\\"DEPARTURE
SCAN\\",\\"Details\\":\\"MAIA,PT\\"},{\\"Date\\":\\"2017-05-04
18:53\\",\\"StatusDescription\\":\\"ORIGIN
```

```
SCAN\","Details\":"MAIA,PT\"},"Date\":"2017-05-04
17:46\","StatusDescription\":"PICKUP
SCAN\","Details\":"MAIA,PT\"},"Date\":"2017-05-04
06:17\","StatusDescription\":"BILLING INFORMATION
RECEIVED\","Details\":"PT\"]]}",
  "extraInfo": null,
  "created_at": "2017-05-05 15:25:37",
  "updated_at": "2017-05-07 12:44:38"
}
}
```

- **Error Responses:**

Code	Message
------	---------

400	Bad Request. Shipment not found.
-----	----------------------------------

Update Tracking Details

POST `/api/shipping/track/update/key={api_key}`

This endpoint serves to update shipment details about tracking. This is a webhook for easypost use.

- **Users allowed**

- TrackingMore

- **Sample Request Body:**

```
{
  "meta": {
    "code": 200,
    "type": "Success",
    "message": "Success"
  },
  "data": {
    "id": "92a3ef572e51aa5352bc5e3333ea45ec",
    "tracking_number": "RF290896544PT",
  }
}
```

```
"carrier_code": "ctt",
"status": "transit",
"created_at": "2017-05-05T01:48:27+08:00",
"updated_at": "2017-05-05T21:46:36+08:00",
"title": "",
"order_id": null,
"customer_name": null,
"customer_email": "",
"original_country": "Portugal",
"destination_country": "Italy",
"itemTimeLength": 10,
"origin_info": {
  "weblink": "http:\\\\www.ctt.pt\\/",
  "phone": null,
  "carrier_code": "ctt",
  "trackinfo": [{
    "Date": "2017-05-02 20:21",
    "StatusDescription": "International reception",
    "Details": "-,MILANO PESCHIERA BORROMEO,-"
  }, {
    "Date": "2017-04-26 18:02",
    "StatusDescription": "Acceptance by the Post
Services",
    "Details": "-,FREIXIEIRO,-"
  }]
},
"destination_info": {
  "weblink":
"http:\\\\www.poste.it\\online\\dovequando\\home.do",
  "phone": null,
  "carrier_code": "poste-italiane",
  "trackinfo": [{
    "Date": "2017-05-05 17:52",
    "StatusDescription": "In lavorazione presso il
Centro Operativo Postale",
    "Details": "PADOVA PD"
  }, {
    "Date": "2017-05-04 00:28",
    "StatusDescription": "In lavorazione presso il
Centro Operativo Postale",
    "Details": "PESCHIERA BORROMEO MI"
  }, {
    "Date": "2017-05-03 04:07",
    "StatusDescription": "In lavorazione presso il
Centro Operativo Postale",
```

```

        "Details": "Centro Scambi Internazionale"
      }, {
        "Date": "2017-05-03 02:18",
        "StatusDescription": "In lavorazione presso il
Centro Scambi Internazionale",
        "Details": "Centro Scambi Internazionale"
      }, {
        "Date": "2017-05-02 08:41",
        "StatusDescription": "Partito dal Centro Scambi
Internazionale",
        "Details": "PORTOGALLO"
      }, {
        "Date": "2017-04-27 00:02",
        "StatusDescription": "Preso in carico all'estero",
        "Details": "PORTOGALLO"
      }
    ]
  },
  "lastUpdateTime": "2017-05-05 17:52",
  "lastEvent": "In lavorazione presso il Centro Operativo
Postale,PADOVA PD,2017-05-05 17:52"
},
"verifyInfo": {
  "timeStr": 1493992223,
  "signature":
"a5de28b07ce5f5306067849af7e84878839329164d0674c0bf9cd490eb0992da"
}
}

```

- **Success Response:**

```

{
  "code": 200,
  "message": "Successfully updated tracker item with email to
client."
}

```

- **Error Responses:**

Code	Message
------	---------

400	Error saving record.
-----	----------------------

Fetch Tracking Details

GET `/api/shipping/track/fetch/key={api_key}`

This endpoint serves to get all the data of the all shipment registered in trackingmore. This is extremely useful for data analysis.

- **Response**

`/api/shipping/track/ups/1ZA9V2340496274174`

```
{
  "code": 200,
  "message": "Success. Fetched 1042 trackings."
}
```

Objects

Rate Object

Parameter	Function	Variable	Required
carrier	carrier name	string	yes
service	service of the carrier	string	yes
rate	price of the rate	string	yes
currency	currency of the rate (default: EUR)	string	no

Address Object

Parameter	Function	Variable	Required
name	-	string	no

Parameter	Function	Variable	Required
street2	-	string	no
city	-	string	yes
state	-	string	no
zip	-	string	yes
country	-	string	yes
company	-	string	no
phone	-	string	yes
email	-	string	no
residential	default: <i>false</i>	boolean	no

Parcel Object

Parameter	Function	Variable	Required
typology	Typology of the logistic unit (Huub DB)	string	yes *
width	Width of the box (Customized Box)	string	yes **
length	Length of the box (Customized Box)	string	yes **
height	Height of the box (Customized Box)	string	yes **
weight	in kilograms	float (to 1 decimal point)	yes

* only required if the box is registered in Spoke (in this case, width, length and height are not required)

* only required if it's for simulation of customized boxes (in this case, typology is not required)

Custom Item Object

Parameter	Function	Variable	Required
-----------	----------	----------	----------

description Parameter	description of the item Function	string Variable	no Required
quantity	quantity	float	no
weight	weight	float	no
value	monetary value	float	no
hs_tariff_number	HS tariff number	string	no
origin_country	origin country	string	no
currency	currency of the value (default: <i>EUR</i>)	string	no

Customs Info Object

Parameter	Function	Variable	Required
eel_pfc	EEL or PFC (usually: <i>NOEEI 30.37(a)</i>)	string	yes
contents_type	<i>documents, gift, merchandise, returned_goods, sample, or other</i>	string	no
contents_explanation	Human readable description of content. Required for certain carriers and always required if contents_type is <i>other</i>	string	yes
restriction_type	<i>none, other, quarantine, or sanitary_phytosanitary_inspection</i>	string	no
customs_item	items in that shipment	array of Customs Items Objects	yes

Shipment Object

Parameter	Function	Variable	Required
shipment_outbound_unit_id	ID of the shipment (Huub DB)	integer	yes *
parcel	dimensions of the parcel	Parcel Objects	yes

Parameter	Function	Variable	Required
customs_info	Information about the parcel	Customs Info Object	yes

** only required when [buying an Order](#)*

Order Object

Parameter	Function	Variable	Required
from_address	sender address	Address Object	yes
to_address	receiver address	Address Object	yes
shipments	information about the shipment	array of Shipment Objects	yes
billing	information about billing (if it's not Huub paying)	Billing Object	no
reference	Huub's internal Reference	String	no *

Billing Object

Parameter	Function	Variable	Required
receiver	True if it's the receiver paying. False if it's Huub Client paying	boolean	yes
bill_account	Bill Account Number	string	yes
bill_country	Bill Country	string	yes
bill_zip	Bill Postal Code	String	yes *

** only required when it's the receiver paying (receiver = true)*

Ecommerce API

1. [Products](#)
 - 1.1. [Fetch Products](#)
 - 1.2. [Get Products](#)
 - 1.3. [Get all categories](#)
 - 1.4. [Update Stock](#)
 2. [Order](#)
 - 2.1. [Fetch Orders](#)
 - 2.2. [Get Orders](#)
 - 2.3. [Fulfil Order](#)
 3. [Customers](#)
 - 3.1. [Get Customer](#)
 4. [Webhook Management](#)
 - 4.1. [Set Webhooks](#)
 - 4.2. [Get Webhooks](#)
 - 4.3. [Delete Webhooks](#)
 5. [Webhooks](#)
 - 5.1. [Product Update](#)
 - 5.2. [Order Update](#)
 6. [Objects](#)
-

Products

Fetch Products

`GET /api/ecommerce/{huubclient_id}/products/fetch?key={api_key}&categories={categories_ids}`

This endpoint has the purpose of fetching all products associated with an huub client. It will fire a job that will push the information, normalize it and store it in a big table and in cache (redis) for 1 day. When the job finished, a slack notification will be fired.

This method is extremely important for the onboard of new clients (initial synchronization) and is intended to end with the transaction of xls files which are the source of multiple errors.

It is possible to only fetch products of certain categories. In this case, the url should contain a *categories* parameter. This parameter is all the categories ids needed separated by commas. To see all categories and the corresponding ids, please see [Get all categories](#).

Redis key example for a product with sku example-shirt-s and owned by the client with id 2:
product:2:example-shirt-s

- **Users allowed**

- Huub

- **Success Response:**

```
{
  "code": 202,
  "message": "Fetching Products."
}
```

- **Error Responses:**

Code	Message
400	Bad Request. Client's Id not recognized.
400	Bad Request. This client hasn't a valid ecommerce store.

Get Products

GET `/api/ecommerce/{huubclient_id}/products/get?key={api_key}`

This endpoint has the purpose of get all products stored in cache and associated with an huub client.

This method is intended to be called after fetching products for that client. For more information see: [Fetch Products](#).

The result will contain an array of [Normalized Product Objects](#).

- **Users allowed**

- Huub

- **Success Response:**

```
{
  "code": 202,
  "message": "Success.",
  "products": [
    {...},
    {...}
  ]
}
```

- **Error Responses:**

Code	Message
400	Bad Request. Client's Id not recognized.
400	Bad Request. This client hasn't a valid ecommerce store.

Get all categories

GET `/api/ecommerce/{huubclient_id}/products/categories?key={api_key}`

This endpoint has the purpose of getting all categories of products of one specific Huub client. The result will contain the number of categories found as well as an array of [Category Objects](#).

- **Users allowed**

- Huub

- **Success Response:**

```
{
  "code": 200,
  "message": "Success.",
  "number_of_categories": 195,
  "categories": [
    {...},
  ]
}
```

```
{...}
]
```

- **Error Responses:**

Code	Message
400	Bad Request. Client's Id not recognized.
400	Bad Request. This client hasn't a valid ecommerce store.
400	Bad Request. Error fetching categories from web shop.

Update Stock

POST `/api/ecommerce/{huubclient_id}/products/stock`

This endpoint has the purpose of create a new order (for one or more shipments) and to retrieve the rates which will be use to make a forecast. It takes a JSON object containing sender, receiver, shipment and billing information. Each element of the shipment array is a package and contains its dimensions, weight, and detail about the items that make it up.

- **Users allowed**

- Huub

- **Data Params**

Parameter	Function	Variable	Required
sku	Product SKU	String	yes
stock	Stock for that product	Integer	yes

- **Sample Request Body:**

```
{
  "sku": "woo-single-2",
  "stock": 100
}
```

```
}
```

- **Success Response:**

```
{  
  "code": 200,  
  "message": "Success."  
}
```

- **Error Responses:**

Code	Message
400	Bad Request. You need to send sku and stock.
400	Bad Request. Client's Id not recognized.
400	Bad Request. This client hasn't a valid ecommerce store.
403	Forbidden. Error Updating Stock. No products were found with this SKU.
403	Forbidden. Error Updating Stock. No match for ecommerce store were found for this products.

Orders

Fetch Orders

GET `/api/ecommerce/{huubclient_id}/orders/fetch?key={api_key}`

This endpoint has the purpose of fetching all ongoing orders associated with an Huub client. It will fire a job that will push the information, normalize it and store it in an big table and in cache (redis) for 1 day. When the job finished, a slack notification will be fired.

This method is important for the onboard of new clients (initial synchronization) when they

have ongoing orders that want to keep track.

Redis key example for an order with reference #54749 and owned by the client with id 2:

order:2:#54749

- **Users allowed**

- Huub

- **Success Response:**

```
{
  "code": 202,
  "message": "Fetching Orders."
}
```

- **Error Responses:**

Code	Message
400	Bad Request. Client's Id not recognized.
400	Bad Request. This client hasn't a valid ecommerce store.

Get Products

GET `/api/ecommerce/{huubclient_id}/orders/get?key={api_key}`

This endpoint has the purpose of get all orders stored in cache and associated with an huub client.

This method is intended to be called after fetching orders for that client. For more information see: [Fetch Orders](#).

The result will contain an array of [Normalized Order Objects](#).

- **Users allowed**

- Huub

- **Success Response:**

```
{
  "code": 202,
  "message": "Success.",
  "orders": [
    {...},
    {...}
  ]
}
```

- **Error Responses:**

Code	Message
400	Bad Request. Client's Id not recognized.
400	Bad Request. This client hasn't a valid ecommerce store.

Fulfil Order

POST `/api/ecommerce/{huubclient_id}/orders/fulfil?key={api_key}`

This endpoint has the purpose of partially or complete fulfil an order on the webshop of a certain Huub client

- **Users allowed**

- Huub

- **Data Params**

Parameter	Function	Variable	Required
order_num	Order number	String	yes
waybill	Tracking number	String	yes
carrier	Carrier Name	String	yes
tracking_url	Tracking Number to follow the shipment	String	yes

fulfilments Parameter	Quantities to be fulfilled Function	Array of Fulfil Variable Object	yes Required
complete_fulfilment	Stock for that product	Boolean	yes

- **Sample Request Body:**

```
{
  "order_num": "#1017",
  "waybill": "1ZA9V2340492089248",
  "carrier": "ups",
  "tracking_url":
  "https://thehub.trackingmore.com/en/1ZA9V2340492089248",
  "fulfilments": [
    {
      "sku": "sku3",
      "qnt": 1
    }
  ],
  "complete_fulfilment" : false
}
```

- **Success Response:**

```
{
  "code": 200,
  "message": "Success."
}
```

- **Error Responses:**

Code	Message
400	Bad Request. You need to send fulfilments and Order number.
400	Bad Request. Fulfilments must be an array.
400	Bad Request. You need to send waybill and carrier name.
400	Bad Request. Client's Id not recognized.
400	Bad Request. This client hasn't a valid ecommerce store.

400 Code	Message
400	Bad Request. For Woocommerce Clients, you need to make complete fulfilment.
400	Bad Request. Order #1017 not found.
400	Bad Request. Product sku-3 has an error. Fulfilable quantity is less than the real quantity fulfilled.
403	Forbidden. Some Error occurred with fulfilment.
403	Forbidden. Some Error occurred with fulfilment.

Webhook Management

Set Webhooks

POST `/api/ecommerce/{huubclient_id}/webhooks?key={api_key}`

This endpoint has the purpose of automatically set the webhooks on the web stores in order to expedite the onboard of new clients. This way, Huub can get the more recent information in real time since day one.

- **Users allowed**

- Huub

- **Success Response:**

```
{
  "code": 200,
  "message": "Webhooks created with success."
}
```

- **Error Responses:**

Code	Message
400	Bad Request. Client's Id not recognized.
400	Bad Request. This client hasn't a valid ecommerce store.
400	Bad Request. This User has not api key.
400	Bad Request. Error saving webhooks for Order Updates
400	Bad Request. Error saving webhooks for Product Updates.
400	Bad Request. Error saving webhooks for Client Updates.
400	Bad Request. This client hasn't a valid ecommerce store.

Get Webhooks

GET `/api/ecommerce/{huubclient_id}/webhooks?key={api_key}`

This endpoint has the purpose of get all current webhooks associated with an huub client.

- **Users allowed**
 - Huub
- **Success Response:**

```
{
  "code": 200,
  "message": "Success.",
  "Webhooks": [
    {
      "id": 353362629,
      "topic": "orders/updated",
      "url": "http://demo-spoke.thehuub.co:8080/api/webhooks/ecommerce/orders/update?key=.&client_id=2"
    },
    {
      "id": 353362693,
```

```
    "topic": "products/create",
    "url": "http://demo-spoke.thehuub.co:8080/api/webhooks/ecommerce/products/update?key=.&client_id=2"
  },
  {
    "id": 353362757,
    "topic": "products/update",
    "url": "http://demo-spoke.thehuub.co:8080/api/webhooks/ecommerce/products/update?key=.&client_id=2"
  }
]
```

- **Error Responses:**

Code	Message
400	Bad Request. Client's Id not recognized.
400	Bad Request. This client hasn't a valid ecommerce store.
400	Bad Request. Error getting webhooks.

Delete Webhooks

DELETE /api/ecommerce/{huubclient_id}/webhooks?key={api_key}

This endpoint has the purpose of delete webhooks that are not necessary anymore.

- **Users allowed**

- Huub

- **Data Params**

Parameter	Function	Variable	Required
sku	Product SKU	String	yes

stock	Stock for that product	Integer	Yes
Parameter	Function	Variable	Required

- **Sample Request Body:**

```
{
  "webhooks": [353362629, 353362693]
}
```

- **Success Response:**

```
{
  "code": 200,
  "message": "Successfully deleted webhooks."
}
```

- **Error Responses:**

Code	Message
400	Bad Request. Client's Id not recognized.
400	Bad Request. This client hasn't a valid ecommerce store.
400	Bad Request. Error deleting webhook 353362629. Please Delete Manually.

Webhooks

Product Update

POST `/api/webhooks/ecommerce/products/update/client_id={huubclient_id}&key={api_key}`

This endpoint will receive product information of the webhop. It will normalize and store it in a big table and in cache

- **Users allowed**

- ecommerce stores

- **Sample Request Body:**

- For woocommerce

```
{
  "product": {
    "title": "Julieta Black",
    "id": 52708,
    "created_at": "2017-02-01T04:16:46Z",
    "updated_at": "2017-05-10T08:22:13Z",
    "type": "variable",
    "status": "publish",
    "downloadable": false,
    "virtual": false,
    "permalink": "https://www.wolfandrita.com/shop/oh-a-la-bourgeois/julietta-black/",
    "sku": "WRWSS17JUBLK",
    "price": "75.00",
    "regular_price": null,
    "sale_price": null,
    "price_html": "<ins class='h2'><span class='woocommerce-Price-amount amount'><span class='woocommerce-Price-currencySymbol'>&euro;75.00</ins>",
    "taxable": true,
    "tax_status": "taxable",
    "tax_class": null,
    "managing_stock": true,
    "stock_quantity": 0,
    "in_stock": true,
    "backorders_allowed": false,
    "backordered": false,
    "sold_individually": false,
    "purchaseable": true,
    "featured": false,
    "visible": true,
    "catalog_visibility": "visible",
    "on_sale": false,
    "product_url": null,
    "button_text": null,
    "weight": null,
    "dimensions": {
      "length": null,
      "width": null,
      "height": null,
      "unit": "cm"
    },
    "shipping_required": true,
    "shipping_taxable": true,
    "shipping_class": null,
    "shipping_class_id": null,
    "description": null,
    "short_description": "<p>SS Blouse with Raw Edges Hem and Sleeves</p>\n<p>100% CLY</p>\n<h5>SOFT IRON<br />\nMACHINE WASH UP TO 30\u00ba<br />\nDO NOT DRY CLEAN<br />\nNO TUMBLE DRY</h5>",
    "reviews_allowed": false,
    "average_rating": "0.00",
    "rating_count": 0,
    "related_ids": [52844, 52752, 52808, 52732, 52856],
    "upsell_ids": [],
    "cross_sell_ids": [],
    "parent_id": 0,
    "categories": ["Adult", "Oh A La Bourgeois", "Woman"],
    "tags": [],
    "images": [
      {
        "id": 53435,
        "created_at": "2017-02-03T09:29:11Z",
        "updated_at": "2017-02-03T09:29:11Z",
        "src": "https://www.wolfandrita.com/wp-content/uploads/2017/02/julietta_black-1.jpg",
        "title": "julietta_black",
        "alt": null,
        "position": 0
      },
      {
        "id": 53436,
        "created_at": "2017-02-03T09:29:52Z",
        "updated_at": "2017-02-03T09:29:52Z",
        "src": "https://www.wolfandrita.com/wp-content/uploads/2017/02/lifestyle_adult07-
```

```
1.jpg","title":"lifestyle_adult07","alt":null,"position":
1},{ "id":52354,"created_at":"2017-02-
01T03:06:50Z","updated_at":"2017-02-
01T04:16:46Z","src":"https://www.wolfandrita.com/wp-
content/uploads/2017/02/lifestyle_adult05.jpg","title
":"lifestyle_adult05","alt":null,"position":2}], "featured
_src":"https://www.wolfandrita.com/wp-
content/uploads/2017/02/julieta_black-
1.jpg","attributes":[{"name":"Size","slug":"size","positi
on":0,"visible":true,"variation":true,"options":["L","M",
"S"]}],"downloads":[],"download_limit":0,"download_expiry
":0,"download_type":null,"purchase_note":null,"total_sale
s":0,"variations":[{"id":52709,"created_at":"2017-02-
01T04:16:46Z","updated_at":"2017-05-
04T11:23:50Z","downloadable":false,"virtual":false,"perma
link":"https://www.wolfandrita.com/shop/oh-a-la-
bourgeois/julieta-black/?
attribute_pa_size=s","sku":"WRWSS17ELBLKS","price":"75.00
","regular_price":"75.00","sale_price":null,"taxable":tru
e,"tax_status":"taxable","tax_class":null,"managing_stock
":true,"stock_quantity":9,"in_stock":true,"backorders_all
owed":false,"backordered":false,"purchaseable":true,"visi
ble":true,"on_sale":false,"weight":null,"dimensions":{"le
ngth":null,"width":null,"height":null,"unit":"cm"},"shipp
ing_class":null,"shipping_class_id":null,"image":[{"id":5
3435,"created_at":"2017-02-
03T09:29:11Z","updated_at":"2017-02-
03T09:29:11Z","src":"https://www.wolfandrita.com/wp-
content/uploads/2017/02/julieta_black-
1.jpg","title":"julieta_black","alt":null,"position":0}],
"attributes":[{"name":"Size","slug":"size","option":"s"}]
,"downloads":[],"download_limit":0,"download_expiry":0},
{"id":52710,"created_at":"2017-02-
01T04:16:46Z","updated_at":"2017-05-
10T08:22:12Z","downloadable":false,"virtual":false,"perma
link":"https://www.wolfandrita.com/shop/oh-a-la-
bourgeois/julieta-black/?
attribute_pa_size=m","sku":"WRWSS17ELBLKM","price":"75.00
","regular_price":"75.00","sale_price":null,"taxable":tru
e,"tax_status":"taxable","tax_class":null,"managing_stock
":true,"stock_quantity":8,"in_stock":true,"backorders_all
owed":false,"backordered":false,"purchaseable":true,"visi
ble":true,"on_sale":false,"weight":null,"dimensions":{"le
ngth":null,"width":null,"height":null,"unit":"cm"},"shipp
ing_class":null,"shipping_class_id":null,"image":[{"id":5
```

```

3435,"created_at":"2017-02-
03T09:29:11Z","updated_at":"2017-02-
03T09:29:11Z","src":"https://www.wolfandrita.com/wp-
content/uploads/2017/02/julieta_black-
1.jpg","title":"julieta_black","alt":null,"position":0}},
"attributes":[{"name":"Size","slug":"size","option":"m"}],
"downloads":[],"download_limit":0,"download_expiry":0},
{"id":52711,"created_at":"2017-02-
01T04:16:46Z","updated_at":"2017-05-
10T08:22:11Z","downloadable":false,"virtual":false,"perma
link":"https://www.wolfandrita.com/shop/oh-a-la-
bourgeois/julieta-black/?
attribute_pa_size=l","sku":"WRWSS17ELBLKL","price":"75.00
","regular_price":"75.00","sale_price":null,"taxable":tru
e,"tax_status":"taxable","tax_class":null,"managing_stock
":true,"stock_quantity":8,"in_stock":true,"backorders_all
owed":false,"backordered":false,"purchaseable":true,"visi
ble":true,"on_sale":false,"weight":null,"dimensions":{"le
ngth":null,"width":null,"height":null,"unit":"cm"},"shipp
ing_class":null,"shipping_class_id":null,"image":[{"id":5
3435,"created_at":"2017-02-
03T09:29:11Z","updated_at":"2017-02-
03T09:29:11Z","src":"https://www.wolfandrita.com/wp-
content/uploads/2017/02/julieta_black-
1.jpg","title":"julieta_black","alt":null,"position":0}},
"attributes":[{"name":"Size","slug":"size","option":"l"}],
"downloads":[],"download_limit":0,"download_expiry":0}},
"parent":[],"grouped_products":[],"menu_order":0},"api_us
er_id":6,"key":"woo_wolf","client_id":"9"}

```

- For shopify

```

{"id":9254893073,"title":"Nice Shoes","body_html":"
<p>\u00a0Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Vivamus id purus et nisl elementum
ornare sit amet non mi. Phasellus euismod sapien et
sapien mollis auctor. Suspendisse interdum erat ut metus
euismod sed elementum massa euismod. Donec eget magna
orci, at fermentum metus. Sed rhoncus, nibh nec dapibus
adipiscing, felis felis convallis arcu, quis semper justo
mauris sed magna. Nulla facilisi. Aliquam ultricies diam
non turpis tristique porta. Sed nec ligula id turpis
iaculis adipiscing. Phasellus viverra urna sed purus
porta rutrum. Nam ultrices molestie tellus, ac vulputate

```

```
odio condimentum in.
</p>","vendor":"shopify","product_type":"kicks","created
_at":"2017-04-12T11:17:29+01:00","handle":"nice-
shoes","updated_at":"2017-04-
13T16:08:00+01:00","published_at":"2017-04-
12T11:17:29+01:00","template_suffix":null,"published_scop
e":"global","tags":null,"variants":[{"id":"41534041926","pr
oduct_id":"10347474950","title":"Default
Title","price":"120000.00","sku":"sku9","position":1,"gra
ms":0,"inventory_policy":"deny","compare_at_price":null,"
fulfillment_service":"manual","inventory_management":"sho
pify","option1":"Default
Title","option2":null,"option3":null,"created_at":"2017-
04-12T06:17:29-04:00","updated_at":"2017-04-13T11:08:00-
04:00","taxable":true,"barcode":null,"image_id":null,"inv
entory_quantity":6,"weight":0,"weight_unit":"kg","old_inv
entory_quantity":6,"requires_shipping":true}], "options":[
{"id":"12523231366","product_id":"10347474950","name":"Title"
,"position":1,"values":["Default
Title"]}]], "images":[{"id":"25259958918","product_id":"103474
74950","position":1,"created_at":"2017-04-
12T11:17:29+01:00","updated_at":"2017-04-
12T11:17:29+01:00","src":"https://cdn.shopify.com/s/f
iles/1/1919/6391/products/ADG20065_thum1_177_2152_20
721.jpg?
v=1491992249","variant_ids":[]}], "image":{"id":"2525995891
8","product_id":"10347474950","position":1,"created_at":"201
7-04-12T11:17:29+01:00","updated_at":"2017-04-
12T11:17:29+01:00","src":"https://cdn.shopify.com/s/f
iles/1/1919/6391/products/ADG20065_thum1_177_2152_20
721.jpg?v=1491992249","variant_ids":[]}]}
```

- **Success Response:**

```
{
  "code": 200,
  "message": "Product Updated successfully."
}
```

- **Error Responses:**

Code	Message
------	---------

Code	Message
202	Bad Request. Client's Id not recognized.
202	Bad Request. This client hasn't a valid ecommerce store.
202	(Product id: 356) Error storing Product in DB.

Note: The Code is 202 because if the code was different from 2xx, the webhook would be inactive

Order Update

POST `/api/webhooks/ecommerce/orders/update/client_id={huubclient_id}&key={api_key}`

This endpoint will receive order information of the webhop when an order is created or updated. It will normalize and store it in a big table and in cache.

If the order is paid, this will trigger all the process of shipment outbound. First it will create the customer and his addresses(billing and shipping) in Huub's DB, then, will create and store Sales Order, Order drop and their corresponding items. Finally, it will create and store shipment outbound information as well as the packs.

- **Users allowed**
 - ecommerce stores
- **Sample Request Body:**
 - For woocommerce

```
{
  "id": 526,
  "parent_id": 0,
  "number": "526",
  "order_key": "wc_order_58fa1c89a244e",
  "created_via": "checkout",
  "version": "3.0.3",
  "status": "processing",
  "currency": "EUR",
  "date_created": "2017-04-21T14:51:53",
  "date_created_gmt": "2017-04-21T14:51:53",
  "date_modified": "2017-04-21T14:51:54",
  "date_modified_gmt": "2017-04-21T14:51:54",
  "discount_total": "0.00",
  "discount_tax": "0.00",
  "shipping_total": "0.00",
  "shipping_tax": "0.00",
  "cart_tax": "0.00",
  "total": "2.00",
  "total_tax": "0.00",
  "prices_include_tax": false,
  "customer_id": 1,
  "customer_ip_address": "192.168.1.150",
  "customer_user_agent": "mozilla/5.0 (x11; linux x86_64) applewebkit/537.36 (khtml, like gecko) chrome/58.0.3029.81"
}
```

```
safari/537.36", "customer_note": "", "billing": {"first_name": "Luis", "last_name": "Melo", "company": "", "address_1": "hfhvg", "address_2": "", "city": "vhvchvhg", "state": "", "postcode": "4770-252", "country": "PT", "email": "lfmelo@thehuub.co", "phone": "123456789"}, "shipping": {"first_name": "", "last_name": "", "company": "", "address_1": "", "address_2": "", "city": "", "state": "", "postcode": "", "country": ""}, "payment_method": "cod", "payment_method_title": "Cash on delivery", "transaction_id": "", "date_paid": "2017-04-21T14:51:54", "date_paid_gmt": "2017-04-21T14:51:54", "date_completed": null, "date_completed_gmt": null, "cart_hash": "ed07fd2c8806cba4b5c18794a2b540dc", "meta_data": [{"id": 4779, "key": "_download_permissions_granted", "value": "yes"}], "line_items": [{"id": 5, "name": "Woo Single #2", "product_id": 441, "variation_id": 0, "quantity": 1, "tax_class": "", "subtotal": "2.00", "subtotal_tax": "0.00", "total": "2.00", "total_tax": "0.00", "taxes": [], "meta_data": [], "sku": "woo-single-2", "price": 2}], "tax_lines": [], "shipping_lines": [], "fee_lines": [], "coupon_lines": [], "refunds": []}
```

- For shopify

```
{ "id": 4557432645, "email": null, "closed_at": null, "created_at": "2017-05-11T11:59:40+01:00", "updated_at": "2017-05-11T11:59:40+01:00", "number": 18, "note": null, "token": "54d3b95b376abdb6a8510e2a623d6d57", "gateway": "manual", "test": false, "total_price": "120.00", "subtotal_price": "120.00", "total_weight": 0, "total_tax": "22.44", "taxes_included": true, "currency": "EUR", "financial_status": "paid", "confirmed": true, "total_discounts": "0.00", "total_line_items_price": "120.00", "cart_token": null, "buyer_accepts_marketing": false, "name": "#1018", "referring_site": null, "landing_site": null, "cancelled_at": null, "cancel_reason": null, "total_price_usd": "130.41", "checkout_token": null, "reference": null, "user_id": 106164997, "location_id": 24492997, "source_identifier": null, "source_url": null, "processed_at": "2017-05-11T11:59:40+01:00", "device_id": null, "phone": null, "browser_ip": null, "landing_site_ref": null, "order_number": 1018, "discount_codes": [], "note_attributes": [], "payment_gateway_names": [ "manual" ], "processing_method": "manual", "checkout_id": null, "source_name": "shopify_draft_order", "fulfillment_status": null, "tax_lines": [ { "title": "VAT", "price": "22.44", "
```

```
rate":0.23}]], "tags":null, "contact_email":null, "order_status_url":null, "line_items":[{"id":8273335557, "variant_id":28802967173, "title":"Gnarly Shoes", "quantity":1, "price":"120.00", "grams":0, "sku":"sku5", "variant_title":null, "vendor":"shopify", "fulfillment_service":"manual", "product_id":7730957893, "requires_shipping":true, "taxable":true, "gift_card":false, "name":"Gnarly Shoes", "variant_inventory_management":null, "properties":[], "product_exists":true, "fulfillable_quantity":1, "total_discount":"0.00", "fulfillment_status":null, "tax_lines":[{"title":"VAT", "price":"22.44", "rate":0.23}]]], "shipping_lines":[], "fulfillments":[], "refunds":[], "api_user_id":9, "key":".", "client_id":"2"}
```

- **Success Response:**

```
{
  "code": 200,
  "message": "Order Updated successfully."
}
```

- **Error Responses:**

Code	Message
202	Bad Request. Client's Id not recognized.
202	Bad Request. This client hasn't a valid ecommerce store.
202	(Order id: 356) Error storing Order in DB.
202	(Order id: 356) Error in Address Storing.
202	(Order id: 356) Error in Sales Order Storing.
202	(Order id: 356) Error in Order Drop Storing.
202	(Order id: 356) Error in Shipment/Pack Storing.

Note: The Code is 202 because if the code was different from 2xx, the webhook would be inactive

Objects

Address Object

Parameter	Function	Variable	Required
customer_id	Customer identifier	Integer	yes
address	Address	String	yes
zip	Postal code	String	yes
city	City	String	yes
country_id	Country identifier	Integer	yes
address_type_id	Type of address (billing/shipping)	Integer	yes
deliver_to	Name of the person to deliver the pack	String	no

Customer Object

Parameter	Function	Variable	Required
name	Customer Name	String	yes
email	Email address of the customer	String	yes
phone	Phone of the customer	String	yes
notes	Notes	String	no
contact_person	Contact Person	String	no
default_currency_id	Default Currency identifier	Integer	yes
activity	If the customer data is updated	Boolean	no
created_by	who created that record in DB	Integer	no
ecommerce_customer_id	Customer identifier in webshop	Integer	no

Normalized Product Object

Parameter	Function	Variable	Required
huubclient_id	Huub Client Identifier	Integer	yes
product_id	Product identifier	String	yes
vendor_name	Name of the Vendor	String	no
product_name	Name of the product	String	no
product_description	Description of the product	String	no
product_short_description	Short Description of the product	String	no
product_type	Type of the product	String	no
product_typology	Typology of the product (simple, variable,...)	String	no
product_status	Status of the product	String	no
product_visibility	Visibility of the product	Boolean	no
product_sku	SKU of the product	String	no
product_price	Price of the product	Float	no
product_sales_price	Price of the product in time of sales	Float	no
product_sales_from	Date of the beginning of sales	Date	no
product_sales_to	Date of ending of sales	Date	no
product_on_sale	If the product is on sales	Boolean	no
product_in_stock	If the product is in stock	Boolean	no
product_stock_management	If webshop keep track of stock	Boolean	no
product_stock	Quantity available of this product	Integer	no

Parameter	Collection of categories to which the product belongs	Variable Category Objects	Required
product_images	Collection of links to images of the product	Array of Strings	no
product_dimensions	Dimensions of the product	String	no
product_weight	Weight of the product	String	no
product_attributes	Collection of product attributes	Array of Objects	no
product_inserted_at	Date of product creation	Date	no
product_updated_at	Date of product update	Date	no
model_id	Model identifier	String	yes
model_name	Name of the model	String	no
model_description	Description of the model	String	no
model_visibility	Visibility of the model	Boolean	no
model_sku	SKU of the model	String	no
model_price	Price of the model	Float	no
model_sales_price	Price of the model in time of sales	Float	no
model_sales_from	Date of the beginning of sales	Date	no
model_sales_to	Date of ending of sales	Date	no
model_on_sale	If the model is on sales	Boolean	no
model_in_stock	If the model is in stock	Boolean	no
model_stock_management	If webshop keep track of stock	Boolean	no
model_stock	Quantity available of this model	Integer	no

Parameter	Function	Variable	Required
model_images	Collection of links to images of the model	Array of Strings	no
model_dimensions	Dimensions of the model	String	no
model_weight	Weight of the model	String	no
model_attributes_detail	Collection of the detail of the model attributes	Array of Objects	no
model_inserted_at	Date of model creation	Date	no
model_updated_at	Date of model update	Date	no

Note (1): Much of the parameters are not required. This is due to the extreme discrepancy between ecommerce stores.

Note (2): A Product may or may not have one or multiple models. Example: A T-shirt can have multiple attributes like color, size,... So, Models of that t shirt can be a green XL, green M, red M, red S,...

Normalized Order Object

Parameter	Function	Variable	Required
huubclient_id	Huub Client Identifier	Integer	yes
order_id	Order identifier	String	yes
order_status	Status of the order	String	no
order_currency	Currency of the order (default: EUR)	String	no
order_financial_status	Financial Status of the order	String	no
order_items	Collection of the products ordered	Array of Order Items	no
fulfillment_status	Status of the fulfilment	String	no
fulfillment_info	Collection of items fulfilled	JSON	no
customer_id	Customer identifier	Integer	no
total_discount	Total value of the discount	Float	no

Parameter	Function	Variable	Required
total_shipping	Total value of the shipping	Float	no
total_taxes	Total value of the taxes	Float	no
total_invoice	Total value of the invoice	Float	no
price_includes_taxes	If taxes are included in price	Boolean	no
billing_information	Address of billing	Address Object	yes
shipping_information	Address of Shipping	Address Object	yes
payment_method	Method of paying	String	no
order_inserted_at	Date of Order creation	Date	no
order_updated_at	Date of Order Update	Date	no
completed	If the order is completed or not	Boolean	no
ecommerce_id	Id of the order in webshop	String	no

Note: Much of the parameters are not required. This is due to the extreme discrepancy between ecommerce stores.

Order Items Object

Parameter	Function	Variable	Required
sku	Product Identifier	String	yes
quantity	Quantity ordered	Integer	yes
fulfillment_status	Status of the fulfilment	String	yes
quantity_fulfilled	Quantity fulfilled	Integer	yes
total_invoice	Total value of the invoice	Float	yes
total_discount	Total value of the discount	Float	yes
total_taxes	Total value of the taxes	Float	yes

Category Object

Parameter	Function	Variable	Required
id	Numeric identifier	Integer	yes
name	Name of the category	String	yes
slug	Alphabetic identifier	string	yes
description	Description of the category	String	no

Fulfilment Object

Parameter	Function	Variable	Required
sku	Product identifier	String	yes
qnt	Quantity to be fulfilled	Integer	yes

v1.1, 12-05-2017

Mail API

1. [Send an Email](#)
2. [Email Log](#)

Send an Email

POST `/api/mail/send/key={api_key}`

This endpoint has the purpose of sending emails in various situations such as updating customers on the status of their orders, updating forecast, etc. The email type is specified in the request body.

- **Users allowed**

- Huub

- **Data Params**

Parameter	Function	Variable	Required
<code>mail_to</code>	receivers email addresses	array of strings	yes
<code>type</code>	type of email to send	string	yes
<code>options</code>	information needed to fill the template email	object	yes
<code>mail_from</code>	senders email addresses	array of strings	no
<code>mail_replyto</code>	reply to email addresses	array of strings	no
<code>mail_cc</code>	CC email addresses	array of strings	no

mail_attachments
Parameter

attachments to email (local storage)
Function

array of
strings
Variable

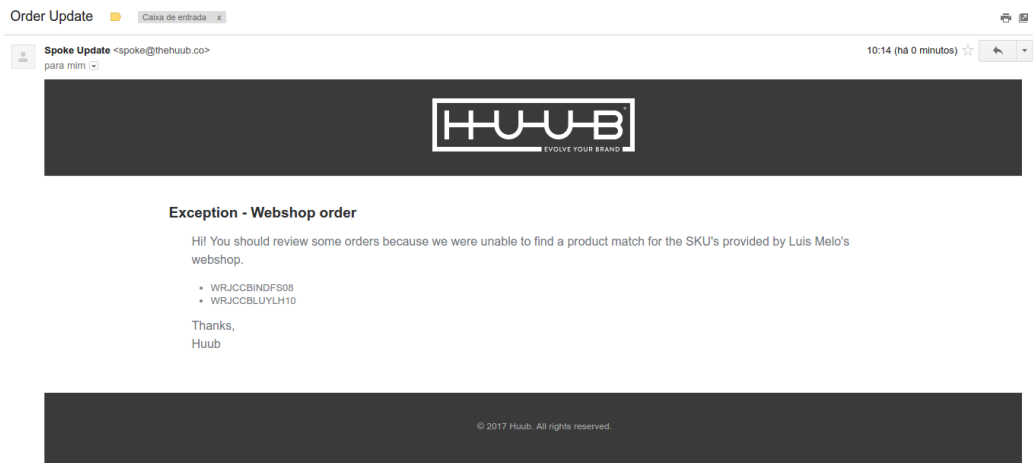
no
Required

- **Sample Request Body for different types of email:**

- **Exception SO items**

```
{
  "mail_to": "account1@thehuub.co",
  "mail_from": ["exception@thehuub.co"],
  "mail_replyto": ["account1@thehuub.co"],
  "mail_cc": ["account2@thehuub.co",
    "account3@thehuub.co"],
  "type": "exception_so_items",
  "options": {
    "client_name": "Luis Melo",
    "skus": ["WRJCCBINDFS08", "WRJCCBLUYLH10"]
  }
}
```

This type of email is generated automatically when a Sales Order with products that are not registered in Huub's DB is placed. It is intended to let account managers know and resolve this issue with the client.



- **Job Finished**

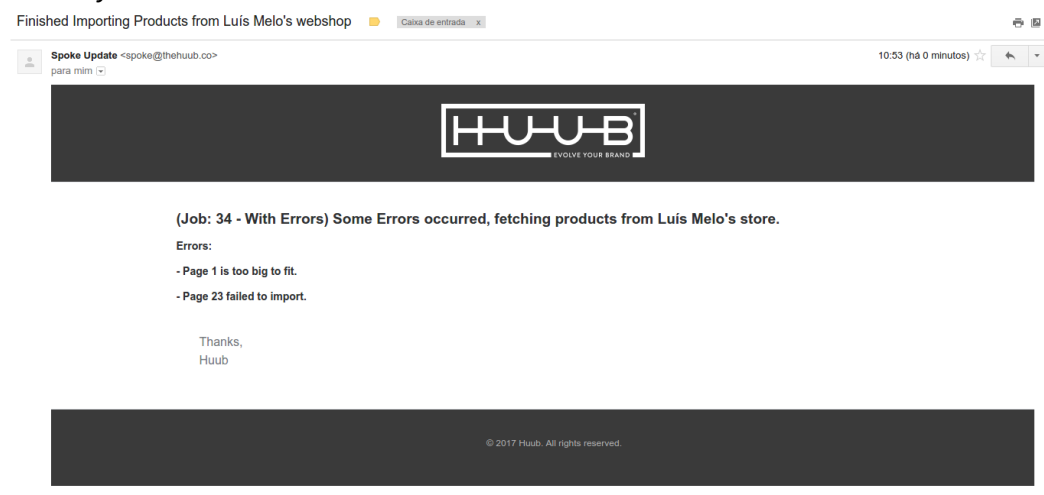
```
{
  "mail_to": "account1@email.com",
  "mail_from": ["spoke@thehuub.co"],
  "mail_replyto": ["account1@thehuub.co"],
  "mail_cc": ["account2@thehuub.co",
```

```

"account3@thehuub.co"],
  "type": "job_finished",
  "options": {
    "subject": "Finished Importing Products from Luís
Melo's webshop",
    "message": "(Job: 34 - With Errors) Some Errors
occurred, fetching products from Luís Melo's store.",
    "errors": [
      "Page 1 is too big to fit.",
      "Page 23 failed to import"
    ]
  }
}

```

This type of email is generated automatically when an Async Job finish. It's mere informative and will display errors if they occurred. It is intended to let account managers know that the job they fired either is complete or something failed along the way.



- Tracking Update

```

{
  "mail_to": "client1@email.com",
  "mail_from": ["delivery@thehuub.co"],
  "mail_replyto": ["account1@thehuub.co"],
  "mail_cc": ["account2@thehuub.co",
"account3@thehuub.co"],
  "type": "update_tracking",
  "options": {
    "status" : "pickup",
    "days_in_transit" : 6,
    "client" : "Awesome Brand",

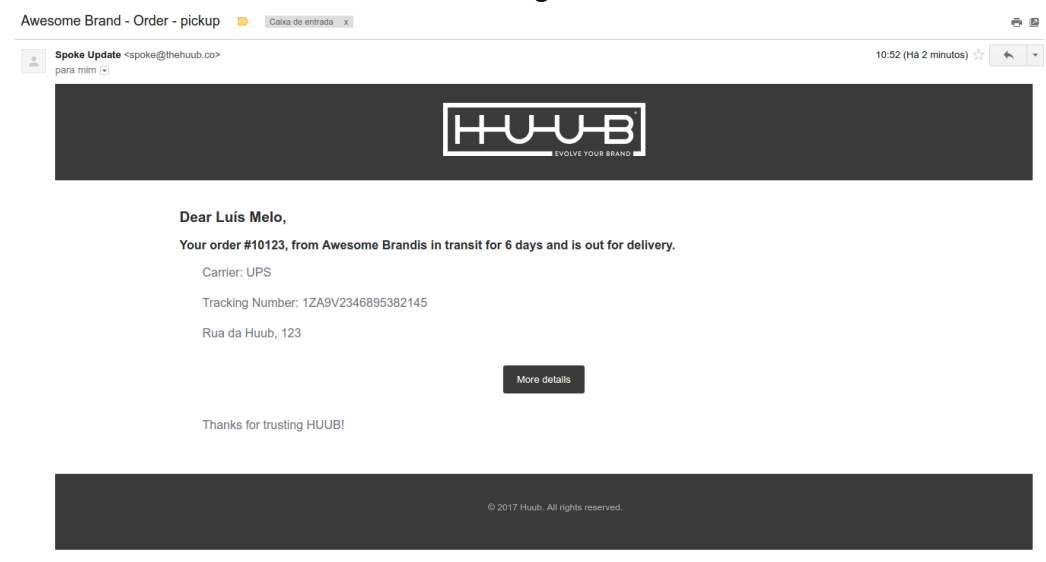
```

```

    "customer" : "Luís Melo",
    "order" : "#10123",
    "waybill" : "1ZA9V2346895382145",
    "carrier" : "UPS",
    "url" :
    "https://thehuub.trackingmore.com/en/1ZA9V2346895382145",
    "status_message" : " is in transit for 6 days and
is out for delivery.",
    "address" : "Rua da Huub, 123"
  }
}

```

This type of email is generated automatically when TrackingMore send update on a certain order. To fire the email, the order must be ready to ship, encounter an exception, exception was solved or is in the last mile (out for delivery). The email is sent to both customer and account manager.



- **Success Response:**

```

{
  code: 200
  message: Mail Sent Successfully
}

```

- **Error Responses:**

Code	Message
400	Bad Request. Please send the receiver email address

Code	Message
403	Forbidden. Wrong type of email provided

0	Error sending the email
---	-------------------------

Email Log

POSTT `/api/mail/log/key={api_key}`

This endpoint has the purpose of storing logs of unsuccessful email sent. This will log various situations such as dropped messages, hard bounces, spam complaints and unsubscribes. This email serves as webhook to mailgun.

- **Users allowed**

- Mailgun

- **Sample Request Body for bounced email:**

```
{
  "domain" : "thehuub.co",
  "recipient": "alice@example.com",
  "event": "bounced",
  "timestamp": 1489145076,
  "error": "Not delivering to previously bounced address"
}
```

- **Success Response:**

```
{
  code: 200
  message: Success.
}
```

- **Error Responses:**

Code	Message
------	---------

403	Forbidden. Error storing at DB.
Code	Message

v1.0, 06-05-2017