Faculdade de Engenharia da Universidade do Porto

Sistema de análise e validação dos dados empresariais em vários organismos de registo a nível mundial

Rui Filipe Fernandes Santos

Dissertação realizada no âmbito do
Mestrado Integrado em Engenharia Electrotécnica e de Computadores
Major Automação

Orientador Interno: Prof. Luís Filipe Pinto de Almeida Teixeira
Orientador Externo: Victor Manuel Duarte Dias Coelho

Julho 2017

Resumo

A presente dissertação foi proposta pela empresa E-goi, cuja principal atividade é o fornecimento de soluções de gestão de clientes e marketing multicanal. A maioria das suas vendas é feita *online*, e muitas delas com empresas de países diferentes. Um dos grandes desafios das vendas *online* é a validação da identidade do cliente. Essa validação pode ser feita através do número de identificação fiscal (NIF) ou número equivalente, no caso de outros países. Para a E-goi, esta validação é importante para: (1) evitar a criação de perfis falsos de utilizador dos serviços E-goi; (2) cobrar o valor correto de IVA (dependendo do país, o valor de IVA a cobrar é diferente) e (3) validação dos dados fiscais para emissão de faturas. A criação de perfis falsos de utilizador é especialmente preocupante no caso da E-goi, uma vez que uma pequena parte dos seus clientes utiliza o seu serviço para fins maliciosos, como o envio de muitos SMS ou emails com conteúdo que visa roubar os dados de outras pessoas (*phishing*) ou para fazer SPAM.

Não existe um serviço unificado que permita a validação do NIF ou número equivalente para vários países, numa mesma plataforma. Desta limitação, surge o tema desta dissertação: a criação de um serviço unificado para validação dos dados de uma empresa cliente ou cliente particular, a partir do NIF ou número equivalente para qualquer país.

No seguimento deste problema, foi desenvolvida uma API REST que permite a troca de dados com pedidos HTTP utilizando estruturas de dados no formato JSON. A aplicação foi desenvolvida em PHP com Zend Framework 3, utiliza MariaDB como software de base de dados relacional e NGINX para servir a aplicação. A documentação da API segue a especificação OpenAPI Specification e utiliza Swagger UI para gerar o ficheiro de documentação interativa. A aplicação desenvolvida permite validar o NIF ou número equivalente, e a partir deste permite também validar nome do cliente, assim como morada ou data de nascimento. Esta validação é feita recorrendo a serviços de validação externos ou através de algoritmos de validação do formato do NIF ou número equivalente. A aplicação permite a validação para países da União Europeia e Brasil, países com os quais a E-goi efetua mais transações comerciais. O trabalho desenvolvido consiste numa aplicação funcional e robusta que cumpre todos os requisitos impostos pela E-goi.

(Página intencionalmente deixada em branco)

Abstract

This dissertation was proposed by E-goi, whose core business activity is related with customer management and multichannel marketing solutions. Most of their sales are online, and many of them with companies from different countries. The biggest challenge of online sales is validating the customer identity. One of the methods to validate customer identity is through the tax identification number (NIF, in Portugal) or equivalent number in other countries, which is important to: (1) avoid fake E-goi user profiles creation; (2) collect the correct VAT percentage (the VAT percentage is different depending on the client country) and (3) validation of tax data for issuing invoices. For E-goi, the creation of fake user profiles is a concern, since a small number of its customers might use their services for malicious purposes, such as phishing or SPAM.

There is not a unified service that allows NIF or equivalent number validation for different countries, on the same platform. Thus, the theme of this dissertation has the purpose to fix this limitation: the creation of a unified service for client data validation using NIF or equivalent number for any country.

To solve this problem, a REST API was developed to exchange data with HTTP requests using JSON data structures. The application was developed in PHP with Zend Framework 3, it uses MariaDB as relational database software and NGINX to serve the application. The API documentation follows the OpenAPI Specification and uses Swagger UI to generate the interactive documentation file. The application developed allows validating the NIF or equivalent number, as well as the client name, address or date of birth. This validation is performed combining the usage of external validation services or NIF format algorithms. The application allows validation for European Union countries and Brazil, which are the countries E-goi has more transactions with.

The work developed consists of a functional and high performance web application that meets all the E-goi requirements.

(Página intencionalmente deixada em branco)

Agradecimentos

Em primeiro lugar, gostaria de agradecer à empresa E-goi, por me ter dado a oportunidade de desenvolver esta dissertação em ambiente empresarial e me ter dado todas as condições necessárias para o bom desenvolvimento deste projeto. Em especial, gostaria de agradecer ao Ivo Pereira, Vítor Tavares, Victor Coelho e Rui Mendes por me terem dado feedback, aconselhamento e orientação ao longo de todo o projeto.

Gostaria também de agradecer ao meu orientador na Faculdade de Engenharia da Universidade do Porto, Prof. Dr. Luís Teixeira, pelos seus conselhos.

Para finalizar, um grande agradecimento para a Sara e para a minha família por me ter acompanhado e ajudado ao longo desta jornada.

(Página intencionalmente deixada em branco)

Índice

Lista de figuras

Lista de tabelas

Listagens

Abreviaturas e Símbolos

Lista de abreviaturas

API	Application Programming Interface
CND	Content Delivery Network
CE	Comissão Europeia
CNPJ	Cadastro Nacional da Pessoa Jurídica
CPF	Cadastro de Pessoas Físicas
CRUD	Create, Read, Update, and Delete
EIN	Employer Identification Number
EUA	Estados Unidos da América
FEIN	Federal Employer Identification Number
HATEOAS	Hypermedia as the Engine of Application State
HTTP	Hypertext Transfer Protocol
IVA	Imposto sobre o Valor Acrescentado
IRS	Internal Revenue Service
LATAM	Latin American
MVC	Model-View-Controller
NIF	Número de Identificação Fiscal
NIPC	Número de Identificação de Pessoa Coletiva
OAS	OpenAPI Specification
ORM	Object Relational Mapper
PHP	Hypertext Preprocessor
PHP/FI	Personal Home Page/Forms Interpreter
REST	REpresentational State Transfer
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
TIN	Tax Identification Number
UE	União Europeia
URI	Uniform Resource Identifier

VAT	Value-added Tax
VIES	VAT Information Exchange System
XML-RPC	Extensible Markup Language – Remote Procedure Call
ZF	Zend Framework

Capítulo 1

Introdução

Esta dissertação foi realizada em parceria com a empresa E-goi. A sua principal atividade é o fornecimento de soluções de gestão de clientes e marketing multicanal. A empresa tem vindo a registar um aumento no número das trocas comerciais com empresas de países diferentes. Um dos grandes desafios nas vendas *online* é a validação da identidade do cliente.

A validação dos dados dos clientes é importante para prevenir a criação de contas de utilizador falsas. Além disso, para manter a legalidade fiscal das transações, a E-goi deve tentar certificar-se que as faturas emitidas correspondem ao cliente que realmente fez a compra. Torna-se então necessária a validação dos dados da empresa cliente, nomeadamente número de identificação fiscal (NIF) ou número equivalente, nome e morada. Este procedimento é bastante laborioso, uma vez que diferentes países utilizam diferentes métodos para identificar os contribuintes.

Atualmente, a avaliação da veracidade das informações disponibilizadas pelos clientes é feita recorrendo a várias plataformas de validação, não existindo um serviço unificado que compreenda validar qualquer NIF ou número equivalente a nível mundial.

Esta dissertação tem como objetivo apresentar uma solução unificada que permita validar as informações de qualquer empresa a nível mundial.

Na presente secção será feita uma introdução à empresa parceira, serão apresentados o enquadramento da dissertação: apresentação do problema e a sua relevância para a E-goi, e os objetivos e motivações que levam à procura da sua resolução. Por fim, será apresentada a estrutura da dissertação.

1.1 - Enquadramento

O presente trabalho foi desenvolvido em parceria com a empresa E-goi, uma empresa tecnológica fundada em 2008 por Miguel Gonçalves, atual CEO. A empresa localiza-se no concelho de Matosinhos e conta atualmente com mais de 50 trabalhadores.

A E-goi fornece soluções de *software* como serviço, ou seja, presta serviços a partir de *software* a outras empresas, nomeadamente serviços de gestão de clientes e de marketing multicanal. O marketing multicanal consiste na implementação de estratégias de marketing através de diferentes canais de comunicação, maximizando as oportunidades de interação com futuros clientes. Esses canais de comunicação podem ser email, SMS, chamadas de voz interativas, redes sociais, notificações *push*, fax, entre outras. A E-goi fornece uma solução que permite integrar todos estes canais.

O serviço de marketing por email fornecido pela E-goi permite às empresas o envio de (1) campanhas de emails aos seus clientes, como por exemplo a anunciar promoções, a desejar boas festas, ou a publicitar um novo produto; (2) *newsletters*; (3) *autoresponders* que permitem enviar mensagens automáticas aos clientes quando estes se registam ou efetuam uma compra, por exemplo; entre outros. Para além disso, fornece também formulários de registo que as empresas podem utilizar para captar o contacto dos seus clientes e juntá-lo à sua base de dados. A adicionar, disponibilizam serviços de marketing por SMS, chamadas de voz, notificações *push* e possibilidade de integração com as redes sociais.

O mercado dos canais de marketing e da gestão de contactos é bastante competitivo e requer constante inovação para a empresa se manter na vanguarda. O grande fator de diferenciação da E-goi é o facto de interligar SMS, email, voz e redes sociais, tudo na mesma plataforma. Isto significa que é possível que as empresas enviem aos seus clientes, por exemplo, um SMS que quando lido, desencadeia o envio de um email, que posteriormente desencadeia o envio de uma chamada de voz. Todos estes canais conjugados na mesma plataforma permitem poupar custos e maximizar o retorno das estratégias de marketing.

O serviço fornecido pela plataforma da E-goi é usado por algumas das maiores empresas e instituições em Portugal como por exemplo: Continente, Jumbo, Pingo Doce, Banco de Portugal, Ikea, Worten, Sephora, Boticário, entre muitas outras. A E-goi não se foca apenas no mercado nacional, tendo clientes por todo o mundo, a sua grande maioria em Espanha, Brasil, e outros países latino-americanos. A empresa tem suporte e atendimento ao cliente em Português, Português do Brasil, Espanhol e Inglês. A estratégia da empresa tem dado frutos, tendo registado um aumento de vendas de 50% em 2016 [1].

Como parte da sua estratégia de inovação, a empresa faz frequentemente parcerias com estudantes, que com uma mente mais "fresca" têm novas ideias para resolução de problemas de forma mais criativa. O tema proposto pela E-goi que deu origem a esta dissertação consiste na criação de um sistema centralizado de validação do formato do NIF das empresas clientes (ou um número equivalente noutros países), a existência do NIF (ou número equivalente) e a veracidade dos dados associados ao NIF (ou número equivalente), como nome, morada e data de nascimento.

1.2 - Motivação e Objetivos

A presente dissertação foi desenvolvida em ambiente empresarial, na empresa E-goi. Os seus clientes encontram-se em todas as partes do mundo, uma vez que fornece um serviço digital que pode ser utilizado por qualquer empresa onde quer que esteja sediada.

Com o aumento das trocas comerciais entre empresas de países diferentes, surge o desafio de validar a veracidade dos dados de uma empresa como o NIF (ou número equivalente), nome, morada e data de nascimento. Cada país adota um formato específico de código de validação de empresas ou clientes particulares, tornando a validação uma tarefa árdua. Muitas vezes esta validação é feita de forma manual com recurso a várias organizações, com sistemas diferentes, sendo este um processo ineficiente.

A validação dos dados é um passo importante durante o processo de compra de determinados serviços. Neste caso, a validação dos dados permite que a empresa mantenha toda a legalidade a nível fiscal no que respeita às transações, certificando-se que o NIF inserido pelos clientes existe, é válido, e que é cobrada a devida taxa de IVA – cada país tem um valor de taxa de IVA diferente.

Outro dos interesses em desenvolver este serviço de validação dos dados das empresas clientes prende-se com o tipo de serviço que a E-goi disponibiliza. A empresa fornece serviços de email marketing, SMS marketing e automação de marketing tendo capacidade de enviar grandes volumes de mensagens para um elevado número de contactos num curto espaço de tempo. Com isto surge um grave problema, uma pequena parte dos clientes que utilizam o serviço E-goi são mal-intencionadas e querem tirar partido do serviço para enviar muitas SMS ou muitos emails com conteúdo que visa roubar os dados de outras pessoas (*phishing*) ou para fazer SPAM. O processo de verificar se o nome do cliente e a morada corresponde ao NIF introduzido é um passo extra que permite à E-goi rejeitar automaticamente, durante o processo de registo, clientes que tentem criar contas com dados falsos.

Ao propor esta dissertação, a empresa E-goi pretende o desenvolvimento de um sistema de validação dos dados de uma empresa de qualquer parte do mundo, dando prioridade aos países com que efetua mais trocas comercias: países da união europeia (UE), Estados Unidos da América (EUA), Brasil e países latino-americanos (LATAM). Existem soluções no mercado, mas que são limitadas, uma vez que permitem a validação só de países membros da UE, ou só do Brasil, etc, não existindo uma plataforma unificada. Há também interesse em saber a localização fiscal do cliente, para se cobrar o valor de IVA adequado durante as trocas comerciais.

A solução desenvolvida consiste num sistema que retorna a seguinte informação em resposta a um NIF ou número equivalente e respetivo país:

- **Validação da existência:** consulta um serviço externo para verificar se o número existe e corresponde às informações inseridas pelo cliente;

- **Validação do formato:** retorna se o formato do número é válido para o país em questão. Note-se que um NIF pode ser válido e não existir;
- **Consulta do valor do IVA:** retorna o valor do IVA para a troca comercial entre a E-goi e a empresa cliente de acordo com a localização fiscal do cliente;
- **Consulta de informação geral acerca do país:** essa informação inclui qual a designação dada ao número de identificação fiscal (por exemplo, em Portugal existe o NIF e o número de identificação de pessoa coletiva (NIPC)); se é possível validar esse número; que tipo de serviços de validação estão disponíveis para esse número.

Para além disso, a solução desenvolvida permite o acesso externo por parte de outras aplicações através de uma *Application Programming Interface* (API) e com o protocolo *Hypertext Transfer Protocol* (HTTP). Assim, a motivação consiste em criar um sistema centralizado que permita validar as informações do cliente através do NIF ou número equivalente.

1.3 - Estrutura da Dissertação

Esta dissertação está dividida em 5 capítulos. O Capítulo 1 introduz a empresa E-goi, o enquadramento do problema proposto, motivação e objetivos para o desenvolvimento desta dissertação.

O Capítulo 2 compreende o estado da arte das tecnologias necessárias para desenvolver o problema proposto. É feito um levantamento da arquitetura de vários serviços web (é dado especial destaque à arquitetura do tipo REST, pois é a que se vai utilizar para desenvolvimento da solução), *frameworks* PHP, bases de dados, servidores web e especificações para documentação de APIs. São apresentadas as suas características, vantagens e desvantagens e uma comparação entre as várias opções disponíveis. Também é feita uma introdução ao NIF ou número equivalente e as várias formas de validação, tendo em conta o país de origem.

No Capítulo 3 é feita uma descrição aprofundada do problema proposto, sendo também apresentada uma solução. É explicada a arquitetura da solução, são analisados os casos de uso da aplicação e são apresentados os métodos da API REST a implementar.

O Capítulo 4 foca-se na implementação da solução. No seguimento das tecnologias apresentadas no estado da arte e da solução proposta, é feita uma escolha entre as diversas tecnologias mencionadas e explicado porque são uma boa solução. É referido como foi implementado o projeto e são apresentados os resultados, tendo em conta os requisitos impostos pela empresa E-goi.

Por fim, o Capítulo 5 inclui uma avaliação do trabalho proposto em termos dos requisitos cumpridos e das eventuais limitações deste projeto. São também apresentadas várias ideias de trabalho futuro para expandir as funcionalidades deste projeto.

Capítulo 2

Estado da Arte

O mercado das aplicações web tem vindo a registar um crescimento significativo todos os anos, prova disso é o facto de a Amazon Web Services, uma empresa de distribuição de aplicações, ter registado o maior crescimento de sempre em 2016 [2]. A acompanhar este crescimento está o aparecimento e evolução de novas tecnologias que permitem melhorar o processo de criação e manutenção de aplicações.

Os consumidores estão cada vez mais exigentes, e esperam aplicações cada vez mais eficientes, rápidas e que estejam sempre disponíveis. Para tal, torna-se necessária: (1) a utilização de arquiteturas de troca de comunicação mais eficazes; (2) servidores web que estejam constantemente a correr e com capacidade para muitas comunicações simultâneas; (3) *frameworks* de programação que aceleram o processo de desenvolvimento de *software* e simplificam a manutenção dos serviços e (4) utilização de bases de dados escaláveis que suportem elevado volume de registos. É ainda importante salientar que na criação de serviços web é necessário recorrer a ferramentas de documentação que expliquem o modo de funcionamento dos serviços. No caso desta dissertação, é necessária a utilização de especificações para documentar *Application Programming Interfaces* (APIs).

Este capítulo integra a pesquisa desenvolvida para resolução do problema proposto para esta dissertação: criar uma solução unificada para validação dos dados de uma empresa de qualquer parte do mundo através do NIF ou número equivalente. Nesta secção serão abordadas as tecnologias existentes para criação da aplicação que soluciona o problema proposto, assim como o funcionamento e especificações do NIF pelo mundo.

2.1 - Serviços Web

A evolução da tecnologia e das formas de comunicação levou muitas empresas a criarem serviços web, isto é, serviços que estão conectados à Internet e que permitem a troca de informação de forma eficiente entre dois computadores. Um serviço web é uma tecnologia

que permite que diferentes sistemas comuniquem entre si. Existe um conjunto de operações criadas num formato *standard* que faz com que estes consigam perceber os comandos e as estruturas de dados trocados entre eles [3]. A utilização de um serviço web bem desenvolvido, idealmente, é independente da plataforma em que foi desenvolvido e linguagem de programação que vai interagir com o mesmo.

De uma forma simplificada, num serviço web existem duas partes: um servidor e um cliente. O servidor é o responsável por conter um conjunto de operações acessíveis, na qual o cliente pode interagir para fazer pedidos como ilustra a Figura 2.1. Desta interação surgiram várias formas de trocar informação. Nesta secção serão abordados três métodos mais relevantes para criação de serviços web no contexto desta dissertação: REST, XML-RPC e SOAP.

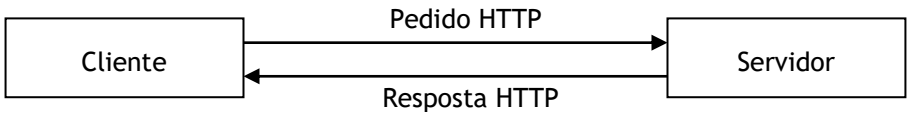

Figura 2.1 – Comunicação cliente-servidor.

2.1.1 – REST

O conceito de arquitetura de *software* REST (*REpresentational State Transfer*) foi definido em 2000 por Roy Thomas Fielding na sua dissertação "*Architectural Styles and the Design of Network-based Software Architectures*" [4] e foi concebido para desenvolvimento de serviços web. Um serviço web desenvolvido com a arquitetura REST é independente da plataforma e da linguagem de programação. Isto faz com que sistemas possam aceder e alterar recursos e informação utilizando um conjunto de operações predefinidas que são independentes do sistema. Num serviço REST, os pedidos são feitos a um *Uniform Resource Identifier* (URI). Tanto o pedido como a resposta podem ser dados em diferentes formatos: XML, HTML, JSON, etc. A resposta retorna a informação requisitada, podendo também devolver informação adicional sobre o processo de comunicação. Este processo de pedido-resposta utiliza o protocolo HTTP e utiliza, entre outras, as seguintes operações: GET, POST, PUT, DELETE.

Propriedades da arquitetura

A arquitetura REST apresenta sete propriedades que surgem a partir do conjunto de restrições impostas a um serviço REST [4].

- **Desempenho**: segundo Roy Thomas Fielding "o desempenho de uma aplicação de rede depende em primeiro lugar, dos requisitos dessa aplicação, depois do estilo de interação escolhido, seguido da arquitetura utilizada e por fim da implementação de cada componente" [4]. Independentemente da qualidade da

arquitetura implementada numa aplicação, se uma componente tiver sido mal implementada, irá afetar todo o desempenho do sistema;

- **Escalabilidade**: a escalabilidade da arquitetura refere-se à capacidade de esta suportar um elevado número de componentes. A escalabilidade pode ser atingida simplificando-se individualmente cada componente e/ou distribuindo as componentes por vários serviços, ou ainda através do controlo da interação das componentes;
- **Simplicidade**: deve-se separar e simplificar cada componente de forma a que cada uma se torne fácil de compreender, utilizar, manter e implementar;
- **Modificabilidade**: refere-se à facilidade com que alterações podem ser feitas à arquitetura da aplicação. Uma aplicação está em constante alteração, sendo os seus requisitos modificados ao longo do tempo. Torna-se por isso importante que a aplicação possa ser melhorada, configurada, customizada ou reutilizada sem influenciar as componentes que foram desenvolvidas no sistema até a esse momento;
- **Visibilidade**: capacidade de componentes monitorizaram as interações entre duas quaisquer componentes na aplicação;
- **Portabilidade**: a portabilidade refere-se à capacidade do *software* ser executável em diferentes ambientes;
- **Fiabilidade**: qualquer arquitetura é sujeita a falhas nos seus componentes. Para manter um sistema fiável é importante avaliar os pontos comuns de falha, utilizar redundância, métodos de recuperação de ações ou utilizar sistemas que monitorizem as interações;

Restrições da arquitetura REST

Existem seis restrições que definem um sistema REST [4]. Estas restrições determinam a forma como um servidor processa os pedidos e responde ao cliente.

- **Cliente-servidor:** criar uma aplicação no estilo cliente-servidor faz com que a interface de utilização possa ser separada da componente responsável pelo armazenamento de dados. Isto faz com que a aplicação tenha capacidade de correr em várias plataformas, simplificando as componentes do servidor e permitindo escalabilidade do sistema.
- ***Stateless*:** a interação cliente-servidor não deve guardar informação de sessões entre pedidos distintos. Cada pedido que o cliente faz ao servidor deve conter toda a informação necessária para este conseguir dar uma resposta. O estado da sessão deve ser guardado a nível de cliente e não de servidor.

- ***Cacheable***: para melhorar a performance da aplicação, certos pedidos cliente-servidor podem ser armazenados em *cache,* pois a resposta que o servidor dá ao cliente contém a mesma informação de pedidos anteriores.
- **Interface uniforme**: uma aplicação REST com uma interface uniforme deve respeitar as seguintes características:
    - em cada pedido os recursos são identificados com uma URI;
    - se um cliente fizer um pedido a um recurso com toda a representação de dados necessária, pode interagir com o recurso;
    - as mensagens enviadas pelo cliente devem conter as informações necessárias para o devido tratamento pelo servidor.
    - *Hypermedia as the Engine of Application State* (HATEOAS): numa API REST e partindo de um determinado recurso, deve ser possível navegar e descobrir quais os recursos e ações que lhe estão associados.
- **Sistema em camadas:** na criação de um serviço pode ser importante proteger alguns dos recursos. Ao desenvolver um sistema em camadas pode-se limitar o cliente de aceder a certos recursos.

REST aplicado a serviços web

Uma API num serviço web que apresenta todas as propriedades de uma arquitetura e segue as restrições de uma arquitetura REST é chamado de RESTful API [5]. Existem ainda outros conceitos que permitem pôr em prática um serviço REST [6], nomeadamente fazer pedidos bem estruturados com os métodos HTTP *standard* como: GET, PUT, POST, DELETE, OPTIONS, PATCH e HEAD [7]. Numa REST API, a troca de dados deve ser feita em JSON ou XML [8].

2.1.2 – XML-RPC

O XML-RPC (Extensible Markup Language – Remote Procedure Call) foi criado em 1998 por Dave Winer [9] para executar procedimentos entre computadores através da Internet independentemente do sistema de desenvolvimento. XML-RPC é uma *remote procedure call* (RPC) que utiliza a formatação XML em pedidos HTTP para transportar, processar e retornar estruturas de dados. RPC, como o seu nome indica, é um mecanismo que permite executar procedimentos ou funções a um computador disponível remotamente.

XML-RPC parte do seguinte princípio: os procedimentos de um sistema são conhecidos e estão bem descritos, logo existe um mecanismo que permite o controlo e troca de informação entre computadores [10]. Esta informação é frequentemente trocada a partir de uma rede com o protocolo HTTP. Quando um procedimento é invocado remotamente, o computador que fez o pedido fica suspenso, e passa os parâmetros para o computador que vai executar os comandos. Quando o procedimento termina e produz o resultado, a resposta é passada para o

computador que fez esse pedido em primeiro lugar. Ao contrário da arquitetura REST, em que se interage com recursos, XML-RPC foi concebido para chamar métodos ou executar funções.

2.1.3 – SOAP

O Simple Object Access Protocol (SOAP) é um protocolo que permite a troca de estrutura de dados entre diferentes aplicações utilizando o formato XML, os pedidos são frequentemente feitos por HTTP [11]. A versão mais recente do protocolo SOAP é 1.2 [12]. SOAP permite a criação de serviços web descentralizados e distribuídos em diferentes sistemas, com o intuito de facilitar a troca de informações entre um cliente e um serviço. O protocolo SOAP tem as seguintes funcionalidades: é independente da plataforma, da linguagem de programação ou do sistema operativo [13]. O protocolo SOAP fornece uma camada de transporte de mensagens para utilização em serviços web que consiste em três partes [3]:

1. Existe um envelope que descreve os detalhes da mensagem que está a ser passada para o serviço e que contém informações em como processar essa mesma mensagem;
2. Existe um conjunto de regras que determinam os tipos de dados que estão a ser trocados entre as aplicações;
3. Existe uma convenção para representar chamadas aos procedimentos e respetivas repostas.

O protocolo SOAP surgiu em Março de 1998 por Dave Winer, Don Box, Bob Atkinson e Mohsen Al-Ghosein da Microsoft, como uma evolução de XML-RPC, daí a ter muitas similaridades na utilização e lógica de funcionamento [14].

2.2 - Servidor Web

Um servidor web é um sistema que processa pedidos feitos por clientes, e de acordo com a informação contida nesse pedido vai disponibilizar páginas que estão armazenadas no sistema local e entregar essas mesmas páginas web a clientes [15]. Esta troca de informações é feita através do protocolo de comunicação HTTP. Na realidade um servidor web tem mais funcionalidades, sendo capaz de: entregar páginas web dinâmicas, correr código no servidor, armazenar e/ou receber conteúdo que é submetido pelos clientes, etc. No entanto existem alguns desafios em implementar servidores web:

- **Múltiplos clientes**: um servidor web deve ser capaz de dar resposta a vários clientes em simultâneo sem causar impacto na performance do servidor. Isto faz com o que servidor seja escalável e utilizável em aplicações complexas;

- **Rapidez de resposta**: um servidor web deve ser capaz de dar respostas o mais rápido possível, com o intuito de evitar o bloqueio de recursos que outros pedidos necessitam;
- **Preparado para falhas**: deve ser capaz de recuperar após uma falha sem perda de informações. Além disso, o servidor web deve estar sempre disponível e em caso de erro, dar informações que identifiquem a falha.

Tendo em conta estes requisitos, existem várias soluções de servidores web de alta performance que vão ser analisados de seguida: Apache2, NGINX e Zend Server.

2.2.1 – Apache

Apache é um *software open source* e multiplataforma, que é desenvolvido e mantido pela comunidade. De acordo com um inquérito realizado em Julho de 2016, 46,4% de todos os *websites* ativos utilizam Apache, sendo este o servidor web mais popular e utilizado [16]. Apache foi criado para poder ser altamente customizado e facilmente disponibilizado para funcionar independentemente do sistema operativo. Com a versão de Apache 2 foi introduzido o conceito de módulos que permitem expandir as funcionalidades principais e que alarga a integração com linguagens de programação como PHP, Python, Ruby, Perl, etc .

2.2.2 – NGINX

O NGINX surgiu com o intuito de ser uma alternativa superior ao Apache, porque apresenta uma arquitetura *event-driven*, isto é, NGINX utiliza um sistema assíncrono para responder a pedidos, o que permite prever mais facilmente a performance de um sistema quando está a receber muito tráfego. NGINX é o segundo *software* de servidor web mais popular, logo a seguir ao Apache, e estima-se que cerca de 21,8% dos sites ativos utilizam NGINX [16]. NGINX sem qualquer configuração adicional, consegue responder a quase 4 vezes mais pedidos por segundo do que o Apache, quando se tratam de ficheiros estáticos [17].

Na Tabela 2.1 apresenta-se uma comparação dos servidores *web* Apache e NGINX. Apache surgiu em 1995, sendo o mais popular, quando comparado com NGINX que surgiu em 2005. Ambos os servidores web são de fácil utilização. No que diz respeito ao tempo de resposta, NGINX tem menores tempos de resposta e maior capacidade para processamento de pedidos simultâneos.

Tabela 2.1 — Comparação quantitativa das características dos servidores web. A quantificação sobre popularidade, facilidade de utilização, tempo de resposta e processamento de pedidos simultâneos foram baseadas na opinião da comunidade de programadores, inquéritos e testes de performance. Relativamente à popularidade, ++ refere-se ao mais popular e + ao menos popular dos servidores web em análise. Em relação à facilidade de utilização, ++ indica que ambos são de simples utilização. No que diz respeito ao tempo de resposta e capacidade de pedidos simultâneos, ++ refere-se ao que tem melhor performance para responder aos clientes.

Servidor Web	**Apache**	**NGINX**
Data de lançamento	1995	2005
Popularidade	++	+
Open source	Sim	Sim
Facilidade de utilização	++	++
Tempo de resposta	+	++
Processamento de pedidos simultâneos	+	++

2.2.3 – Zend Server

Zend Server foi criado pela empresa Zend Technologies e é uma solução comercial de servidores web preparada para hospedar aplicações PHP de alta performance [18]. Zend Server utiliza funcionalidades avançadas que permitem otimizar as aplicações web a nível de performance, segurança escalabilidade e robustez para ambiente empresarial. Zend Server traz 80 extensões de PHP que suportam diferentes tipos de servidores web, bases de dados ou *frameworks* PHP. Zend Server é um produto que está preparado para correr qualquer tipo de aplicação PHP independentemente do servidor web que se pretende configurar ou *framework* PHP que se está a utilizar [18].

2.3 - Frameworks PHP

PHP é uma linguagem de *scripting* que corre no lado do servidor. Em 1994, Rasmus Lerdorf criou *Personal Home Page/Forms Interpreter* (PHP/FI) com o intuito de simplificar a criação de páginas web dinâmicas. Para tal, escreveu em programação C um conjunto de operações que lhe permitiam analisar quem visitava o seu site pessoal [19]. Em junho de 1995, para acelerar o processo de desenvolvimento, Rasmus apercebeu-se que se disponibilizasse publicamente o código que desenvolveu, iria incentivar à correção de erros, implementação de melhorarias e criação de novas funcionalidades por parte da comunidade. Nesse primeiro lançamento, passou a chamar o PHP/FI de Personal Home Page Tools (PHP Tools). Este lançamento foi tão bem aceite pela comunidade, que rapidamente se desenvolveram novas funcionalidades como variáveis, interpretação de formulários, comunicação com bases de dados, etc. Após vários anos de desenvolvimento, em 1997 com o

lançamento de PHP 3, é que este se passou a chamar *Hypertext Preprocessor* (PHP) e foi a primeira versão que incluiu a maioria das funcionalidades conhecidas no PHP atual.

Com o desenvolvimento extenso de serviços web, muitas empresas sentiram a necessidade de criar aplicações que pudessem ser implementadas rapidamente e de forma eficiente. No entanto, é importante seguir boas arquiteturas e sistemas bem estruturados para facilitar a manutenção das aplicações no futuro. As *frameworks* PHP criam aplicações estáveis, que conseguem ser facilmente escaláveis e reutilizadas. Em oposição, o método tradicional é demasiado simplificado e traz imensas limitações. Os programadores criavam as suas próprias formas de programar, mesmo dentro da mesma empresa ou mesma equipa, o que fazia com que outros programadores não conseguissem perceber o código ou estrutura criada. Seguir os padrões das *frameworks* garante que qualquer pessoa consiga adaptar-se ao desenvolvimento de quaisquer aplicações.

PHP possui uma vasta comunidade de pessoas que desenvolvem e criam *frameworks* que tornam a programação de grandes aplicações mais rápida. A maioria das *frameworks* PHP seguem a arquitetura Model View Controller (MVC) [20]. A arquitetura MVC permite combinar a conexão à base de dados (*Model)*, a apresentação da informação numa página web (*View*) e o controlador (*Controller*) que é responsável por processar as entradas e devolver as saídas. As *frameworks* permitem acelerar o desenvolvimento de aplicações e abstrair o programador de processos repetitivos.

2.3.1 –Utilização de *frameworks* PHP

As *frameworks* PHP são cada vez mais utilizadas. De seguida apresentam-se as principais razões que tornam as *frameworks* PHP tão populares [21]:

- **Rapidez de desenvolvimento**: *frameworks* PHP permitem reduzir significativamente a quantidade de código utilizado, uma vez que possuem muitas das funcionalidades já desenvolvidas. Por exemplo: uma *framework* PHP facilita o desenvolvimento de aplicações que requerem acesso à base de dados, porque já trazem operações CRUD (Create, Read, Update, and Delete) implementadas.
- **Escalabilidade**: as *frameworks* PHP permitem a criação de sistemas escaláveis, uma vez que as componentes de código estão otimizadas e permitem que os programadores evitem erros comuns que tornam as aplicações instáveis ou ineficientes.
- **Reutilização**: um código que utiliza *framework* PHP pode ser reutilizado, isto é, permite a criação de novas aplicações recorrendo a ligeiras alterações em código existente. Isto traz vantagem para empresas que escrevem aplicações similares com funcionalidades diferentes.

- **Manutenção**: as *frameworks* PHP forçam os programadores a organizarem o seu código de uma determinada forma, o que faz com que, a longo prazo, o código seja mais fácil de manter e desenvolver.
- **Segurança**: a utilização de *frameworks* PHP faz com que os programadores evitem erros comuns nas suas aplicações. Por exemplo, as *frameworks* trazem ferramentas que permitem acesso simples e seguro às bases de dados.

2.3.2 – Arquitetura Model-View-Controller (MVC)

A maioria das *frameworks* PHP segue a arquitetura Model-View-Controller (MVC) que é uma combinação de leitura e escrita de dados (*Model*), da apresentação da informação (*View*) e do controlador (*Controller*) que faz a ligação entre o modelo e a apresentação [20]. A arquitetura MVC separa a lógica de negócio da apresentação dessa informação num ambiente gráfico. Existem três componentes que definem a arquitetura MVC como ilustrado na Figura 2.2:

- **Modelo** (*Model*): os modelos são responsáveis por armazenar informação, geralmente de forma permanente fora da aplicação numa base de dados. Isto é, os modelos servem para inserir, consultar, apagar e alterar a informação e também para aplicar a lógica de negócio.
- **Apresentação** (*View*): representação visual de um modelo para um determinado contexto, que geralmente é disponibilizado num *browser* com o formato HTML ou JSON.
- **Controlador** (*Controller*): o controlador faz a ligação entre o modelo e apresentação da informação. Isto é, de acordo com a entrada, decide o modelo que deve carregar e que página deve apresentar.

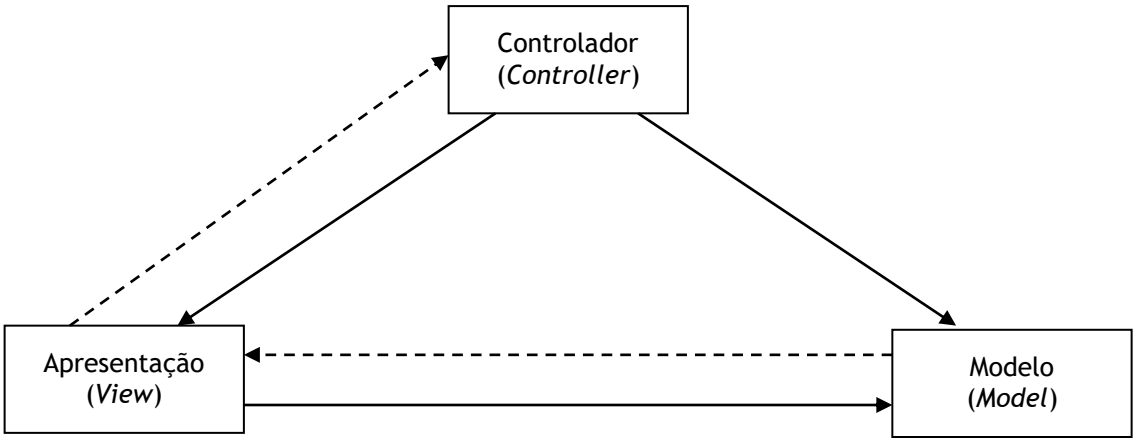

Figura 2.2 – Arquitetura MVC.

As três *frameworks* PHP que se apresentam de seguida utilizam a arquitetura MVC.

2.3.3 – Zend Framework

Zend Framework (ZF) é *open source*, utiliza a arquitetura MVC e foi inicialmente criada por dois contribuidores principais de PHP, Andi Gutmans e Zeev Suraski [19]. A versão mais recente é Zend Framework 3. ZF é orientada a objetos e é um conjunto de mais de 60 pacotes de PHP para criação de aplicações de alto desempenho preparado para ser utilizado em ambiente *enterprise* [22]. Zend Framework é considerada uma framework de difícil aprendizagem e implementação, devido à vasta quantidade de funcionalidades, módulos e padrões de programação que disponibiliza. Assim, é necessário analisar quais são as componentes ideais para o desenvolvimento de um determinado projeto.

Uma das vantagens da utilização de Zend Framework prende-se no conceito de modularização, isto é, ZF segue o padrão de modularizar as aplicações de forma a que cada módulo seja uma aplicação sem dependências. Isto é, cada módulo pode conectar a outro sem afetar o funcionamento da aplicação, o que é fundamental para criação de projetos que conectam a outros sistemas ou que podem ser reutilizados noutros projetos.

2.3.4 – Laravel Framework

Laravel Framework é uma *framework* PHP *open source* criada em 2011 por Taylor Otwell e é a mais popular neste momento [23]. Tem um vasto suporte e muitas componentes desenvolvidas que permitem a criação de aplicações complexas de forma rápida. Foi desenvolvida com a arquitetura MVC. Permite simplificar a implementação de funcionalidades que a maioria das aplicações web necessita sempre que se começa um novo projeto: autenticação, interação com a base de dados, sessões, criação de rotas, *cache*, etc…

2.3.5 – Yii Framework

Yii Framework é uma *framework* PHP criada por Qiang Xue e a primeira versão estável foi lançada em Dezembro de 2008 [24]. É uma *framework* considerada segura, rápida e de alta desempenho para criação de aplicações web. Tem uma funcionalidade que a torna uma das *frameworks* PHP mais rápida devido à técnica que utiliza de *lazy loading*. Yii framework é orientada a objetos e baseada na arquitetura MVC. Tem um conjunto de componentes fundamentais para utilizar em aplicações de grande escala: integração de módulos e extensões, ferramentas de *debugging* e validação, *widgets* para estender a utilização da aplicação atual [25].

2.3.6 – Comparação de *frameworks* PHP

A Tabela 2.2 apresenta uma comparação entre as *frameworks* abordadas anteriormente, nomeadamente a nível de data de lançamento, tipo de arquitetura, se são *open source*,

facilidade de utilização, popularidade e escalabilidade [26]. Todas as *frameworks* apresentadas seguem arquitetura MVC e são *open source*. Laravel Framework é considerada a de mais fácil utilização, sendo também a mais popular. Por outro lado, Zend Framework apresenta melhor escalabilidade.

Tabela 2.2 — Comparação das *frameworks* PHP. A quantificação sobre facilidade de utilização, popularidade e escalabilidade foram baseadas na opinião da comunidade de programadores. Em relação à facilidade de utilização, +++ refere-se à mais simples de aprender e desenvolver e + à mais difícil. Relativamente à popularidade, +++ refere-se à mais popular e + à menos popular das *frameworks* em análise. No que diz respeito à escalabilidade, +++ refere-se à que tem mais robustez para aplicação em grande escala.

***Framework* PHP**	**Zend Framework**	**Laravel Framework**	**Yii Framework**
Data de lançamento	2006	2011	2008
Arquitetura MVC	Sim	Sim	Sim
Open source	Sim	Sim	Sim
Facilidade de utilização	+	+++	++
Popularidade	+	+++	++
Escalabilidade	+++	++	+

2.4 - Bases de Dados Relacionais

Bases de dados relacionais permitem gerir grandes quantidades de informação de forma eficiente e de forma simples. Bases de dados relacionais são baseadas num modelo relacional de dados proposto em 1970 por Edgar Frank Codd no laboratório de investigação do IBM em San Jose [27]. Praticamente todas as bases de dados relacionais utilizam a linguagem de computador Structured Query Language (SQL) que surgiu em 1974 para facilitar a interação com a informação armazenada. SQL é independente do sistema em que está a correr e permite consultar, inserir, apagar e atualizar dados que estão armazenados em bases de dados relacionais.

2.4.1 – MySQL

MySQL é um *software* de gestão de base de dados relacionais *open source* que foi desenvolvido nas linguagens de programação C e C++ [28]. A primeira versão do servidor MySQL foi desenvolvida por Michael Widenius e David Axmark e foi lançada em 1995. MySQL é a base de dados *open source* mais popular [29]. É conhecida por ser rápida e fácil de utilizar, compatível e integrável em praticamente qualquer linguagem de programação, uma vez que tem bibliotecas que permitem utilizar MySQL de forma intuitiva. MySQL pode ser usada em pequenos e grandes projetos.

2.4.2 – MariaDB

Em 2009 MySQL foi adquirida pela Oracle, apesar da Oracle manter MySQL *open source*, a comunidade decidiu continuar o projeto fazendo um *fork,* devido à preocupação desta aquisição. Desta forma, garantiram a continuação do desenvolvimento, mantendo-o *open source*. A este novo projeto chamou-se de MariaDB [30]. Até à versão 5.5, todas as funcionalidades de MySQL eram suportadas por MariaDB. MariaDB passou diretamente de versão 5.5 para 10, para reforçar as diferenças de compatibilidade com MySQL. Apesar de muitas das funcionalidades serem compatíveis, muitas outras deixaram de o ser.

2.4.3 – PostgreSQL

PostgreSQL é um sistema de gestão de bases de dados relacional *open source*, criado em 1996 [31]. É reconhecido na indústria como uma base de dados com arquitetura robusta. É compatível com todos os principais sistemas operativos e as principais linguagens de programação, tem bibliotecas que torna a utilização e integração de PostgreSQL simples. PostgreSQL é reconhecida por cumprir todos os requisitos de SQL na sua implementação.

2.4.4 – Doctrine ORM

Doctrine é uma *framework* de mapeamento de objetos relacionais - *Object Relational Mapper* (ORM) [32]. Quando se programa com uma linguagem de programação ou *framework* que é orientada a objetos e, simultaneamente, se utiliza bases de dados relacionais, é possível criar tabelas nas bases de dados que estão associadas às classes do código. A isto chamam-se modelos, como foi analisado anteriormente na secção da arquitetura MVC.

Doctrine ORM permite criar uma ponte entre o modelo e a base de dados, utilizando uma camada de abstração que elimina a necessidade de utilização de comandos SQL, como esquematizado na Figura 2.3. Com a camada de abstração que Doctrine proporciona, o *software* de base de dados utilizado é indiferente. O que quer dizer que, com ligeiras alterações no ficheiro de configuração, este é facilmente compatível com MySQL, MariaDB ou PostgreSQL. Assim, se for necessário migrar a aplicação para outra base de dados, não é necessário alterar código.

2.5 - Documentação de APIs

A arquitetura REST é atualmente a mais utilizada para a implementação de serviços web. No entanto, não existem *standards* obrigatórios a seguir para documentar e descrever APIs REST. Independentemente do objetivo ou propósito de uma determinada API REST, ou seja, se a API foi criada para uso interno ou externo numa organização, uma API REST bem

documentada é fundamental para a sua fácil utilização, manutenção e compreensão por parte de quem a desenvolveu ou a vai utilizar [33].

Criar uma documentação para uma API REST permite definir interfaces de fácil compreensão tanto para programadores como para computadores. Assim, os programadores ao acederem à documentação podem perceber as funcionalidades da API REST e como se utiliza sem ser necessário ler ou analisar código.

Um dos desafios no desenvolvimento de APIs é aquando de modificações, por exemplo no lançamento de uma nova versão. As modificações efetuadas devem ser documentadas e devidamente atualizadas para que outros utilizadores ou serviços consigam utilizar as novas funcionalidades ou adaptar-se às alterações. Para facilitar todo este processo de documentar, testar e manter a documentação de APIs REST, surgiu um conjunto de especificações como Open API Specification, API Blueprint e RAML que facilitam este trabalho.

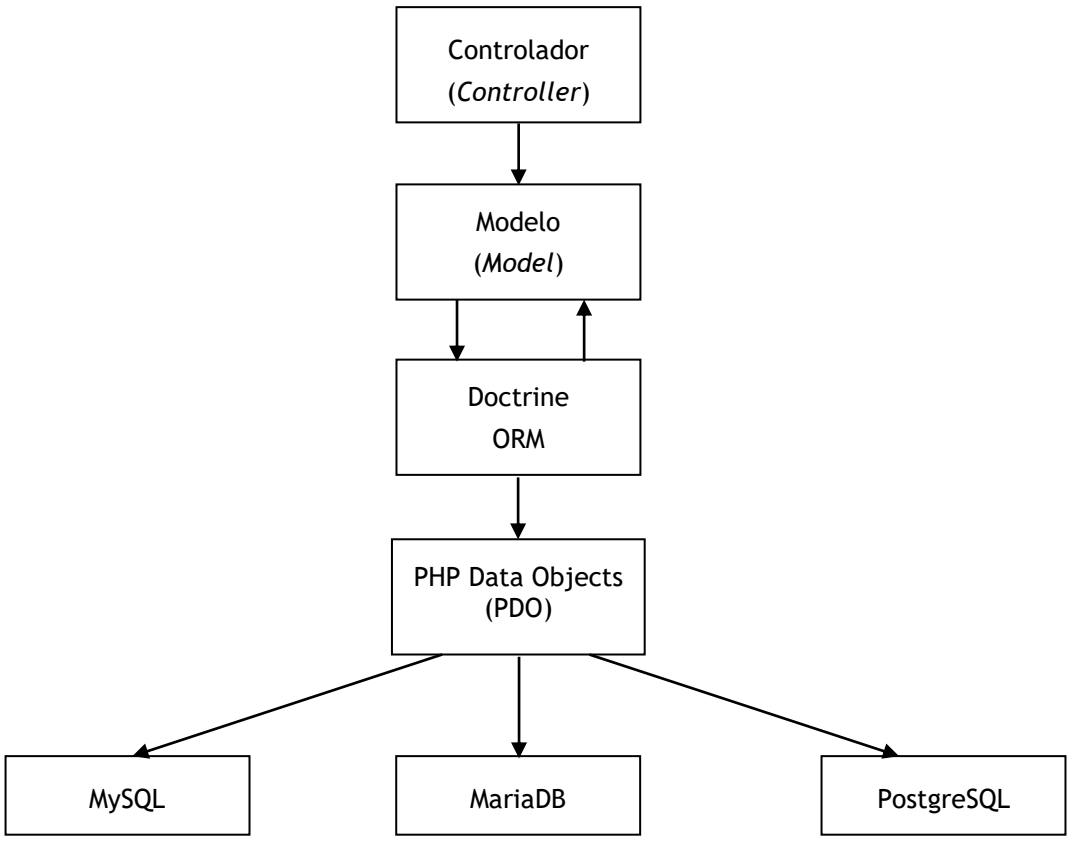

Figura 2.3 – Relação entre o controlador, modelo, Doctrine ORM, PDO e as bases de dados relacionais.

2.5.1 – Open API Specification (OAS)

Existe um conjunto de *standards* chamado de OpenAPI Specification (OAS) que se deve seguir para documentar um serviço de API RESTful [34]. Inicialmente, existia o Swagger Specification, que foi criado em 2011 com o objetivo de descrever, consumir e visualizar serviços web REST. No entanto, em 2015 com o apoio da fundação Linux e um conjunto de

programadores experientes nesta área, decidiu-se que Swagger Specification passaria a chamar-se OpenAPI Specification (OAS). Isto também permitiu evitar um pouco da confusão, porque OpenAPI Specification é apenas um conjunto de regras que utilizam o formato JSON ou YAML que se deve seguir para fazer uma documentação de uma API. Mas Swagger é um conjunto de ferramentas que realmente aplicam esse conjunto de especificações definidas no OAS. Swagger permite melhorar o desenvolvimento das APIs, desde o design, manutenção de versões, testes, produção de serviços etc [35]. Swagger também tem uma componente extra chamada de Swagger UI [36] que permite gerar automaticamente uma representação visual da documentação a partir de um ficheiro OAS.

2.5.2 - API Blueprint

API Blueprint é uma linguagem utilizada para documentar APIs RESTful que utiliza a formatação Markdown [37]. Seguindo a especificação API Blueprint e criando um ficheiro do tipo Markdown é possível gerar automaticamente a documentação do serviço REST.

2.5.3 - RAML

RESTful API Modeling Language (RAML) é uma linguagem baseada na formatação YAML que serve para descrever APIs RESTful [38]. RAML traz todas a funcionalidades necessárias para descrever uma API RESTful de forma a que a sua documentação possa ser facilmente utilizada, tanto por computadores como por utilizadores. Criando um ficheiro do tipo YAML que segue a especificação RAML, podem-se utilizar geradores de código que criam automaticamente a documentação e até o código que os clientes podem utilizar para consumir a API.

2.5.4 – Comparação de especificações para documentação de APIs

Na Tabela 2.3 é apresentada uma comparação das especificações para documentação de APIs abordadas. Neste caso, a OpenAPI Specification, por ser mais antiga, foi adotada na indústria mais cedo, o que a torna mais compatível e conhecida pelos programadores. Também o facto de poder utilizar formato YAML ou JSON faz com que seja mais versátil, comparativamente com as restantes.

Tabela 2.3 — Comparação de especificações de documentações de APIs.

Especificação	**Data lançamento**	**Formato**	***Open source***
OpenAPI Specification	2011	YAML ou JSON	Sim
API Blueprint	2013	Markdown	Sim
RAML	2013	YAML	Sim

2.6 – Validação de Número de Identificação Fiscal

Um dos principais objetivos do trabalho desenvolvido ao longo desta dissertação é a validação dos números de identificação fiscal ou número equivalente. A validação de tal número é importante para que a E-goi possa confirmar os dados inseridos pelas suas empresas clientes e manter a legalidade fiscal das transações ao certificar-se que todas as faturas emitidas correspondem ao cliente que realmente fez a compra [39]. Além disso, este processo de validação previne a criação de contas falsas, uma vez que o cliente tem de introduzir um número de contribuinte válido. Para tal, a E-goi tem que utilizar várias plataformas de validação para avaliar a veracidade das informações introduzidas no ato do pagamento e de registo. Neste momento, não existe um procedimento unificado que valide automaticamente qualquer número contribuinte a nível mundial.

2.6.1 - Número de Identificação Fiscal

O número de contribuinte ou número de identificação fiscal, em Portugal, é um número único de nove dígitos que identifica um contribuinte, ou seja, aquele que por lei deve efetuar pagamentos ao fisco [40]. No caso das empresas, o termo mais correto para se referir ao número de identificação fiscal é NIPC que quer dizer "Número de Identificação Fiscal de Pessoa Coletiva". O NIF/NIPC é uma forma de identificar, de forma inequívoca, uma entidade em transações financeiras [41]. Deste modo, uma das formas que a E-goi tem para validar a veracidade das informações fornecidas pelas empresas clientes é através do seu NIF/NIPC ou número equivalente para outros países.

A maioria dos países da União Europeia usa o número de identificação fiscal (Tax Identification Number (TIN), em inglês) para identificar os contribuintes e facilitar a regulação e administração de impostos [42]. O número de identificação fiscal de empresas está também associado ao nome e morada da sede da instituição [41].

No caso dos EUA as empresas possuem um Employer Identification Number (EIN) também conhecido como Federal Employer Identification Number (FEIN) ou Federal Tax Identification Number. O EIN é um número único de nove dígitos fornecido pelo Internal Revenue Service (IRS) para identificação de empresas a operar nos Estados Unidos. O acesso ao EIN é algo que não é facilmente disponível a todos, só as empresas públicas facultam livremente o EIN e existem bases de dados com as mesmas. No entanto, empresas privadas nos EUA ocultam o EIN e não o facultam durante uma transação por ser considerado um número confidencial [43].

No Brasil, para identificação do contribuinte, existe o Cadastro de Pessoas Físicas (CPF) e o Cadastro Nacional da Pessoa Jurídica (CNPJ) [44].

Diferentes países adotam formas diferentes de identificar os contribuintes, dando diferentes designações a esses números. Uma compilação da informação recolhida a nível de

NIF para diferentes países pode ser consultada no Anexo A. O NIF pode ser validado comparando as informações introduzidas pelas empresas com bases de dados oficiais do seu país de registo, ou na sua inexistência, pode ser validado o formato. A validação do formato do NIF não confirma a sua existência, ou seja, um determinado número de contribuinte pode ser válido e não existir. No entanto, no caso da inexistência de bases de dados de acesso público oficiais, a confirmação do formato é já um bom passo, na validação de informação.

2.6.2 - Algoritmo de identificação de número de contribuinte

Os números de contribuinte funcionam de forma distinta para cada país e a Comissão Europeia (CE) decide não facultar esses mesmo algoritmos. No entanto existe uma tabela que nos permite identificar a estrutura dos números nos vários estados-membros da UE [39]. Os números de contribuinte podem ser validados pelo seu formato. O número de dígitos difere de país para país e devem obedecer a regras de formato. Por exemplo, em Portugal, as regras para validação do NIF são as seguintes:

- O NIF deve ter nove dígitos;
- O primeiro dígito deve ser maior do que zero;
- O último dígito é o dígito de controlo. Para obter este número:
    - Calcular o valor de S, através da seguinte operação, em que N(1) se refere ao primeiro dígito, N(2) ao segundo e assim sucessivamente.

$$S = 9\times N(1)+8\times N(2)+7\times N(3)+6\times N(4)+5\times N(5)+4\times N(6)+3\times N(7)+2\times N(8) \quad (2.1)$$

    - Calcular o resto da divisão inteira de S por 11. O resultado desta operação deve corresponder ao dígito de controlo, que é o último algarismo do NIF. No caso de o resultado desta operação ser 0 ou 1, o dígito de controlo deve ser 0.

Se o NIF obedecer a estas regras, apresenta um formato válido. A título de exemplo, considere-se o seguinte número, que é um NIF com formato válido mas inexistente: 123456789. Este número obedece a todas as regras:

- Tem nove algarismos
- O primeiro algarismo é maior do que zero
- Para calcular o valor de S:

$$S=9\times1+8\times2+7\times3+6\times4+5\times5+4\times6+3\times7+2\times8=159, \quad (2.2)$$

Sabendo o valor de S, pode calcular-se o valor do dígito de controlo:

O resto da divisão inteira de 165 por 11 é 9. O algarismo 9 corresponde ao número de controlo. Portanto o NIF 123456789 é um número válido.

2.6.3 - Validação de existência

A validação de existência do NIF ou número equivalente tem de ser feita por comparação com bases de dados externas. Existem várias soluções comerciais que fazem validação de NIF. A validação do NIF para qualquer parte do mundo não é uma tarefa simples, uma vez que nem todos os países disponibilizam bases de dados oficiais, e por vezes, também não existem bases de dados não oficiais. Há também outros países em que não é comum a partilha do NIF ou equivalente, como é o caso dos EUA, como referido anteriormente. O Anexo A apresenta uma compilação de vários serviços para validação do NIF ou equivalente em vários países.

Para comparação da informação introduzia pelo utilizador com a informação nas bases de dados, é necessário recorrer a algoritmos de comparação para analisar a similaridade. Por exemplo: na base de dados a morada de uma empresa pode estar como "Rua da empresa" e o utilizador introduzir "R. empresa". A informação introduzida pelo utilizador é verdadeira, no entanto não é igual à da base de dados. Têm de se utilizar algoritmos ou funções de comparação adequados, que validem a informação mesmo quando esta não é exatamente igual. Ao utilizar algoritmos de similaridade, estes vão retornar o quão semelhante as informações introduzidas pelo utilizador são com a base de dados. Mais tarde, terá de se definir uma percentagem de similaridade a partir da qual, se considera a informação válida.

Na linguagem de programação PHP existem várias funções que permitem a comparação de duas variáveis de texto [45], o que será útil na implementação da solução.

2.6.4 - Taxa de IVA

Outro interesse das empresas em avaliar o NIF do cliente, prende-se com o facto de terem de aplicar taxas de IVA diferentes consoante o país.

No caso de trocas comerciais entre empresas da UE deve-se isentar o IVA durante a troca comercial, quando o cliente indica o país e o respetivo NIF válido. Se a troca comercial for entre a E-goi e um cliente particular, o IVA deve ser cobrado de acordo com o país do consumidor [46].

Todas as trocas comerciais de produtos e serviços entre clientes fora da UE estão isentas de IVA, sendo o cliente responsável por pagar os seus impostos respetivos [46].

Capítulo 3

Problema e Solução Proposta

O capítulo anterior aborda as tecnologias necessárias para o desenvolvimento de uma aplicação e sistemas de validação do NIF ou número equivalente. No seguimento desse estudo, o presente capítulo foca-se na análise detalhada do problema e sua solução, tendo em conta os temas mencionados no capítulo anterior.

3.1 – Problema

Esta dissertação tem como objetivo a criação de um serviço unificado para validação de números de identificação fiscal ou números equivalentes. Pretende-se efetuar a validação tanto a nível de existência como de formato, a nível mundial. Para desenvolver este tipo de aplicação, é necessário considerar os requisitos propostos, as limitações tecnológicas e analisar tecnicamente a solução a desenvolver. Para tal:

- A aplicação deve seguir as propriedades e restrições da arquitetura REST;
- Deve ser utilizada uma *framework* PHP que contenha as funcionalidades adequadas;
- É necessário fazer pesquisa de serviços externos de validação que verifiquem a existência do NIF ou equivalente e dos dados correspondentes inseridos pelo cliente, e posterior integração;
- Deve ser feita pesquisa e implementação de algoritmos de validação do formato do NIF ou equivalente;
- É preciso construir uma base de dados com informação geral acerca de cada país e os respetivos valores de IVA a ser aplicados aquando de uma transação;
- A aplicação deve ser robusta, testada, e estar sempre disponível;
- Por fim, é fundamental recorrer a uma especificação para documentar a API REST.

Numa primeira fase, o público alvo da aplicação a desenvolver é a empresa E-goi. Esta pretende utilizar a API REST durante o processo de registo de novos clientes para determinar de modo automático se o cliente é válido. Por exemplo, se o cliente apresentar dados cuja validação não é possível, ou apresentar dados inválidos pode acontecer uma das seguintes situações: (1) o sistema pode impedir que o cliente crie uma conta; (2) a conta pode ser bloqueada e o cliente impedido de utilizar o serviço E-goi ou (3) a conta pode ficar num estado pendente, aguardando validação manual.

Este passo extra de validação durante o registo de novos clientes permite à E-goi prevenir o registo de contas falsas. Um dos exemplos seria alguém tentar fazer o registo com o nome de uma entidade que não lhe pertence. Um exemplo muito comum, é alguém tentar fazer-se passar por uma entidade bancária para recolher os dados pessoais de clientes [47]. A título de exemplo, se o cliente introduzir o nome da empresa como "Banco X", mas número de contribuinte 123456789, automaticamente a conta ficaria bloqueada. Isto porque 123456789 é um NIF que não existe. No entanto, mesmo que existisse, uma vez que não corresponde ao nome da empresa introduzida, poderia acontecer uma das 3 situações descritas no parágrafo anterior, impedindo o cliente de utilizar os serviços E-goi. Assim, este passo previne a utilização de contas criadas com o intuito de fazer SPAM e ataques de *phishing*, isto é, tentativa de alguém se fazer passar por uma entidade para capturar dados de outras pessoas.

Numa fase posterior, a empresa E-goi mostrou interesse em disponibilizar este serviço de forma gratuita. Assim, outras empresas com negócios online (e não só) poderiam validar os dados dos clientes durante o processo de registo ou consultar valores de IVA. É importante salientar, que este serviço poderá também ser útil para validar os dados de clientes aquando da emissão das faturas associadas às transações.

3.2 – Solução Proposta

A solução para resolução do problema proposto consistirá numa aplicação que será um micro serviço com arquitetura REST, com casos de uso adequados aos requisitos e métodos para uma API REST para interagir com a aplicação. Nesta secção abordar-se-ão essas temáticas, analisando-se porque são boas alternativas para construção desta solução.

3.2.1 - Micro serviço

A arquitetura micro serviço divide um grande sistema em pequenos serviços, micro serviços. Neste caso, o grande sistema é a E-goi, e o micro serviço é a aplicação que vai ser desenvolvida durante esta dissertação. O objetivo é fazer com que micro serviços comuniquem com o sistema principal invocando métodos de APIs.

A arquitetura micro serviço surge com o intuito de criar uma aplicação que contém um conjunto de serviços que é totalmente independente e modulariza a criação de sistemas complexos.

Vantagens de arquitetura micro serviço

A utilização de micro serviços traz várias vantagens ao desenvolvimento e ciclo de vida das aplicações, nomeadamente a nível de [48]:

- **Falhas**: com aplicações complexas é comum existirem falhas e erros. No entanto, com um micro serviço bem estruturado, qualquer falha fica isolada e só ocorre na componente onde está presente, sendo mais fácil identificar o ponto de erro e corrigi-lo;
- **Compreensão**: novos programadores ou novos trabalhadores que entram para uma empresa e necessitam de utilizar funções que estão implementadas num micro serviço conseguem mais rapidamente compreender o funcionamento e utilização do código;
- ***Deployments***: durante o *deployment* de um código para produção, em vez de ser necessário alterar todo o sistema, pode ser feito apenas o *deploy* do micro serviço. No entanto, isto também pode ser visto como uma desvantagem, como vai ser analisado na próxima secção;
- **Tecnologias**: a criação de micro serviços permite a utilização de diferentes tecnologias no sistema. Cada micro serviço é um projeto isolado e como os diferentes micro serviços comunicam entre si por uma API, a linguagem de programação ou *framework* entre projetos torna-se indiferente. Isto aumenta a flexibilidade das equipas em testar novas tecnologias e não criar dependências de *frameworks*, do *hardware* em que o projeto está a correr ou até de linguagens de programação;
- **Ambiente de programação**: uma vez que a aplicação é mais "leve", os ambientes de programação utilizados vão ser mais rápidos e responsivos para abrir, pesquisar e compilar código, por exemplo;
- **Escalabilidade**: micro serviços podem facilmente ser reutilizadas noutros projetos e permitem criar aplicações que podem ser replicadas por múltiplos servidores, tornando esta arquitetura escalável.

Desvantagens de arquitetura micro serviço

Apesar de existirem razões que tornam a arquitetura de micro serviços uma solução atrativa, também existem desvantagens que são precisas considerar [48]:

- **Mau funcionamento**: em caso de interrupção de um micro serviço, as outras componentes de código devem ter condições que permitam lidar com esses casos

de exceção. Isto leva à necessidade de escrever código adicional no sistema principal que considere eventuais falhas nos micro serviços;

- **Utilização de múltiplas bases de dados**: a criação de bases de dados para cada micro serviço pode tornar a manutenção e criação de *backups* de redundância uma tarefa árdua;
- ***Deployments***: se uma empresa utilizar a arquitetura de micro serviços para todas as suas componentes, os *deployments* tornam-se um desafio. Apesar de cada serviço ser independente a nível de código e funcionalidades, em ambiente de produção todos os micro serviços estão dependentes uns dos outros, tornando-se uma tarefa complexa nos *deployments* para garantir o bom funcionamento sem quebras na interrupção de serviços;
- **Definição de *standards***: é fundamental que haja boa comunicação entre as equipas de programadores. Apesar dos sistemas micro serviços serem independentes, devem seguir boas práticas e estar preparados para integrar com outros sistemas, devendo seguir um conjunto de *standards* predefinidos pela equipa ou empresa.

Utilização de arquitetura micro serviço

Para esta dissertação vai ser utilizado o conceito de micro serviço para desenvolver a solução para o problema proposto. Esta aplicação está fora das funcionalidades do sistema principal da E-goi, logo pode ser considerado um serviço totalmente à parte, que é consultado pelo sistema principal. Além disso, numa fase posterior pretende-se que a aplicação desenvolvida possa integrar com qualquer serviço, mesmo em outras empresas. Tendo em conta este cenário, a utilização de arquitetura micro serviço é uma boa alternativa para o desenvolvimento da aplicação.

3.2.2 - Arquitetura do projeto

Atualmente a maioria dos serviços web só suportam a arquitetura REST para desenvolvimento de APIs e trocam informação com estruturas de dados no formato JSON com o protocolo de comunicação HTTP. Isto deve-se a alguns fatores, nomeadamente a indústria ter adotado este tipo de arquitetura como *standard*. Assim, a maioria dos serviços que possivelmente podem integrar ou interagir como a aplicação a desenvolver nesta dissertação partem do principio que uma API REST vai ser disponibilizada. Neste momento, a E-goi possui todos os seus métodos definidos por APIs REST.

O projeto a desenvolver segue a arquitetura apresentada na Figura 3.1, sendo que esta é dividida em três componentes distintos:

1. **Cliente**: uma aplicação interna ou externa ou um *browser* faz um pedido HTTP a um servidor utilizando uma estrutura de dados em formato JSON;

2. **Servidor**: o servidor que possui uma API REST *gateway* que recebe o pedido, analisa a estrutura de dados e processa esse pedido de acordo com a lógica de negócio aplicacional. De acordo com resultado obtido durante o processamento desse pedido, o servidor acede à camada de informação, se o pedido tiver sido bem estruturado. Caso contrário, devolve uma resposta de erro ao cliente.
3. **Camada de informação**: assumindo que o cliente fez um pedido bem estruturado ao servidor, é feita uma consulta à base de dados interna utilizando Doctrine ORM, ou é feito um pedido a um serviço externo utilizando o protocolo HTTP. No fim deste processo, a informação obtida durante o acesso à camada de informação é devolvida ao servidor, que vai processar essa informação e devolver ao cliente uma resposta HTTP em formato JSON.

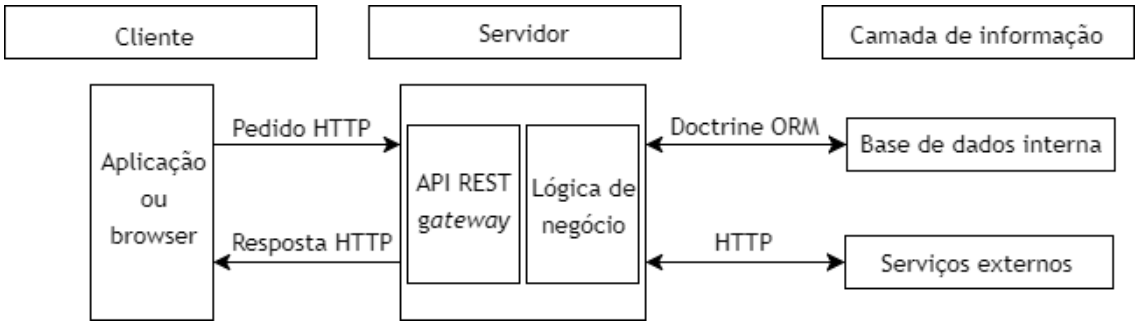

Figura 3.1 – Arquitetura REST e relação entre cliente, servidor e camada de informação.

As aplicações que seguem arquitetura REST são conhecidas pela sua facilidade de integração entre projetos internos ou externos, robustez e escalabilidade [49]. Portanto, a utilização deste tipo de arquitetura na aplicação a desenvolver é uma boa opção.

3.2.3 – Casos de uso

A aplicação a desenvolver nesta dissertação tem vários casos de uso. Os casos de uso vão ser traduzidos na secção seguinte como métodos API REST que são necessários desenvolver para dar resposta a este problema.

Na Figura 3.2 estão ilustrados os diferentes casos de uso para a API REST. O cliente (o ator) que pode ser um *browser* ou qualquer outra aplicação web pode validar o NIF ou número equivalente, nome, morada, data de nascimento ou consultar informações dos países.

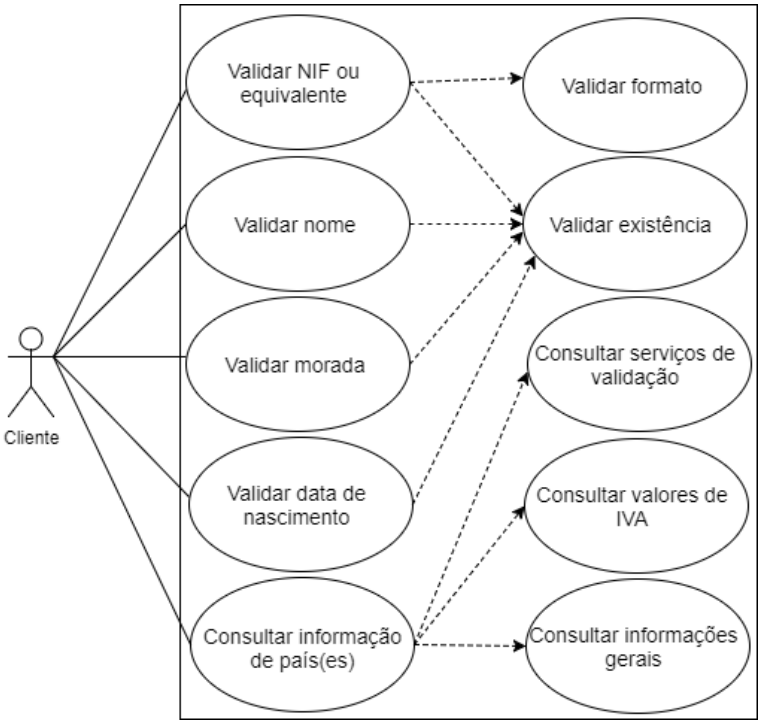

Figura 3.2 – Casos de uso para a API REST.

A validação do NIF ou número equivalente pode ser feita com recurso ao serviço interno de validação de formato, isto é, recebe um NIF como parâmetro e recorrendo a um algoritmo de validação, retorna se esse número é valido ou inválido a nível de formato.

O serviço de validação de existência pode validar quatro parâmetros distintos dependendo do país e dos tipos de métodos de validação disponíveis. Para todos os países da União Europeia e Brasil foram encontrados serviços para validação do NIF ou número equivalente, do nome do cliente e da sede fiscal. No caso específico do Brasil, é ainda possível validar a data de nascimento de um cliente particular.

Existe também uma componente que permite consultar informações dos países: serviços de validação disponíveis, valores de IVA e informações gerais. No caso dos serviços de validação disponíveis, estes permitem verificar que serviços de validação estão disponíveis para cada país. Por exemplo, há casos em que só é possível validar formato, e outros é possível validar formato, existência e sede fiscal. Assim, ao consultar este tipo de informação, o utilizador tem acesso a uma espécie de documentação dinâmica que lhe mostra como deve fazer um pedido bem estruturado à API para um determinado país.

Ainda dentro das informações dos países, também é possível consultar os valores de IVA associados a um determinado país para diferentes tipos de transações: o valor de IVA pode ter o tipo *standard*, reduzido ou zero como vai ser explicado posteriormente. Por fim, é possível consultar informações gerais sobre os países, como por exemplo os códigos de cada país de acordo com o ISO 3166-1 alpha-2, a sua moeda, continente e designação dada ao NIF.

3.2.4 - Métodos API REST

Nesta dissertação é pretendido implementar na API REST do micro serviço seis diferentes métodos. Esses métodos vão ser descritos e exemplificados nesta secção. Também será feita uma comparação entre métodos HTTP POST e HTTP GET.

POST e GET

No Capítulo 2 foram analisados os critérios de uma boa arquitetura API REST. De acordo com esses critérios, é possível concluir que quando se trata de um pedido de validação ou de criação de um novo recurso, é utilizado um pedido HTTP do tipo POST. Por outro lado, utiliza-se um pedido HTTP do tipo GET quando se pretende consultar recursos, quer seja uma coleção ou um item.

Validação de existência

Para validação de existência do NIF ou equivalente, nome, morada e data de nascimento (no caso do Brasil), será desenvolvido o método **POST /validation**. Para validar essas informações é necessário passar um parâmetro *country* que corresponde ao código do país de acordo com a ISO 3166-1 alpha-2 e um outro parâmetro *vatNumber* que corresponde ao NIF ou número equivalente.

Nesse mesmo pedido podem ser passados parâmetros para validação de nome do cliente, morada e ou data de nascimento de acordo com uma percentagem de correspondência entre as informações submetidas com as disponibilizadas em serviços externos. As estruturas de dados que podem ser passadas para validação de um NIF estão ilustradas na Listagem 3.1 e na Listagem 3.2. Note-se que os pedidos são feitos no *body raw* do pedido POST.

```
{
 "country": "PT",
 "vatNumber": "123456789",
 "settings": {
  "name": {
   "enable": true,
   "value": "Cliente exemplo",
   "matchPercentage": 0.6
  },
  "address": {
   "enable": true,
   "value": "Rua do cliente exemplo N 123",
   "matchPercentage": 0.4
 }}}
```

Listagem 3.1 - Estrutura de dados para validar NIF, nome e morada com o pedido POST /validation.

No caso do número de identificação fiscal para pessoas brasileiras (CPF) é possível validar o nome do cliente e a sua data de nascimento. Para essa situação, considera-se o exemplo descrito na Listagem 3.2.

```
{
 "country": "BR",
 "vatNumber": "88182947090",
 "settings": {
  "name": {
   "enable": true,
   "value": "Cliente exemplo",
   "matchPercentage": 0.6
  },
  "dateOfBirth": {
   "enable": true,
   "value": "31/12/1990",
   "matchPercentage": "1"
  }
 }
}
```

Listagem 3.2 - Estrutura de dados para validar CPF, nome e data de nascimento com o pedido POST /validation.

De acordo com o pedido feito e com a lógica de negócio implementada no micro serviço, existem vários cenários de resposta. Se o NIF ou número equivalente existir e corresponder aos parâmetros inseridos de acordo com o grau comparação exigido (*matchPercentage*) a API deve retornar uma resposta verdadeira e dar feedback do resultado, ver Listagem 3.3.

```
{
 "result": true,
 "data": {
  "info": "vatNumber is valid."
 }
}
```

Listagem 3.3 - Resposta de sucesso para pedido POST /validation.

Se o código do país passado como parâmetro for inválido, a API deverá responder com o exemplificado na Listagem 3.4.

```
{
 "result": false,
 "data": {
  "info": "Invalid country code."
 }
}
```

Listagem 3.4 - Resposta de erro para pedido POST /validation quando o código do país é inválido.

Quando é feito um pedido de validação para um NIF ou número equivalente de um país que não tem esse tipo de serviço disponível, deve ser dada uma resposta de erro como ilustrado na Listagem 3.5.

```
{
 "result": false,
 "data": {
  "info": "This validation service method is not available for this country."
 }
}
```

Listagem 3.5 - Resposta de erro para pedido POST /validation quando o método de validação não está disponível.

Se o serviço de validação de existência não conseguir encontrar informações relativas a esse país, retorna um resultado falso e indica que o número não foi encontrado, como mostra na Listagem 3.6.

```
{
 "result": false,
 "data": {
  "info": "vatNumber not found."
 }
}
```

Listagem 3.6 - Resposta de erro para pedido POST /validation quando o NIF ou equivalente não é encontrado no serviço externo.

Validação de formato

Outro método a desenvolver é o **POST /validation/format** que tem como objetivo ser uma forma imediata para validação de números de NIF ou equivalente, uma vez, que recebe como entrada apenas o código do país e o NIF ou número equivalente. Para validação do formato são utilizados algoritmos, e portanto, o pedido de validação de formato não está dependente de serviços externos. Por isso, a resposta da API será mais rápida do que o método anterior, POST /validation. Para muitas utilizações a verificação do formato é o suficiente. A estrutura de dados também deverá ser passada no *raw body* do pedido POST, como exemplificado na Listagem 3.7.

```
{
 "country": "BR",
 "vatNumber": "88182947090",
}
```

Listagem 3.7 - Estrutura de dados para validar o formato de um CPF (POST /validation/format).

Se o pedido de validação de formato for bem executado e o NIF ou número equivalente tiver um formato válido retorna uma resposta como na Listagem 3.8.

```
{
 "result": true,
 "data": {
  "info": "vatNumber has valid format/syntax."
 }
}
```

Listagem 3.8 - Resposta de sucesso para pedido POST /validation/format.

No entanto, se o código do país for inválido retorna Listagem 3.4, se o serviço não estiver disponível para o país retorna a Listagem 3.5 e se o número for inválido retorna a resposta presente na Listagem 3.9.

```
{
 "result": false,
 "data": {
  "info": "vatNumber has invalid format/syntax."
 }
}
```

Listagem 3.9 - Resposta de erro para pedido POST /validation/format quando o NIF ou equivalente é inválido.

Consultas de país

É também necessário desenvolver dois métodos GET para consultar informações gerais para todos os países ou um país em específico. No pedido **GET /countries** deve ser possível consultar uma coleção que retorna todas as informações armazenadas relativamente a todos os países. E com o método **GET /countries/<country>** será possível consultar todas as informações armazenadas para um determinado país. Neste caso, a Listagem 3.10 ilustra o pedido GET /countries/PT e contém a estrutura de dados retornada para Portugal.

```
{
 "result": true,
 "dataLastUpdated": "2017-05-17",
 "disclaimer": "This data was compiled from official sources to be as
accurate as possible.",
 "data": {
  "country": [{
   "name": "Portugal",
   "countryCode": {
    "codeAlpha2": "PT",
    "codeAlpha3": "PRT",
    "numeric": "620"
   },
   "continent": {
    "name": "PT",
    "codeAlpha2": "PRT"
   },
   "currency": {
    "codeAlpha3": "EUR",
```

```
    "name": "Euro",
    "numeric": 978
   },
   "vatRate": [{
    "value": 23,
    "type": "standard"
   }, {
    "value": 13,
    "type": "reduced"
   }, {
    "value": 6,
    "type": "reduced"
   }, {
    "value": 0,
    "type": "zero"
   } ],
   "vatNumber": [{
    "acronym": "NIF",
    "format": "PT + 9 digits - example PT199999999",
     "validationSettings": {
      "name": true,
      "address": true,
      "dateOfBirth": false
     },
     "manualValidation":
"http://ec.europa.eu/taxation_customs/vies/vatResponse.html",
     "service": {
      "format": true,
      "exists": true
     },
    }, {
    "acronym": "NIPC",
    "format": "PT + 9 digits - example PT199999999",
    "validationSettings": {
     "name": true,
     "address": true,
     "dateOfBirth": false
    },
    "manualValidation":
"http://ec.europa.eu/taxation_customs/vies/vatResponse.html",
    "service": {
     "format": true,
     "exists": true
    }
   }]
  }]
 }
}
```


Listagem 3.10 - Resposta de sucesso para pedido GET /countries/PT

Se o pedido GET /countries/<country> for inválido retorna uma resposta de erro como está ilustrada na Listagem 3.4.

Consulta de IVA

Para consultar informações para os valores de IVA de todos os países ou de um país em específico, são necessários dois métodos GET. O pedido **GET /vatRates** deverá consultar uma coleção que retorna todos os valores de IVA para todos os países. E o método **GET /vatRates/<country>** deverá consultar todas as informações armazenadas para um país em específico. A Listagem 3.11 ilustra o pedido GET /vatRates/PT e contém a estrutura de dados que deverão ser retornados para Portugal. Se o pedido GET /vatRates/<country> for inválido retorna uma resposta de erro como ilustrado anteriormente na Listagem 3.4.

```
{
 "result": true,
 "dataLastUpdated": "2017-05-17",
 "disclaimer": "This data was compiled from official sources to be as
accurate as possible.",
 "data": {
  "countryCode": "PT",
  "vatRate": [{
   "value": 23,
   "type": "standard"
  }, {
   "value": 13,
   "type": "reduced"
  }, {
   "value": 6,
   "type": "reduced"
  }, {
   "value": 0,
   "type": "zero"
  }]
 }
}
```

Listagem 3.11 - Resposta de sucesso para pedido GET /vatRates/PT.

Capítulo 4

Implementação e Resultados

De acordo com o problema analisado no capítulo anterior e tendo em conta a solução proposta, neste capítulo será analisada a implementação da solução e respetivos resultados. Para isso serão discutidas as tecnologias utilizadas, a estrutura de um projeto em Zend Framework 3 e os resultados da solução apresentada.

4.1 – Tecnologias a Utilizar

Após análise do problema e da solução proposta é possível escolher as tecnologias que permitem desenvolver a aplicação que cumpra as funcionalidades requisitadas. É importante ter em consideração que as escolhas seguintes foram devidamente aconselhadas pela empresa E-goi e estavam contidas nos requisitos. A empresa tinha como objetivo desenvolver uma aplicação funcional e que simultaneamente fosse fácil de manter pela sua equipam de programadores numa fase posterior.

4.1.1 – Zend Framework 3

Um dos requisitos a cumprir é a criação de uma aplicação com recurso à linguagem de programação PHP. Em ambiente empresarial não é utilizado PHP puro, mas sim *frameworks* de alto nível que aceleram o processo de desenvolvimento. As possíveis escolhas de *frameworks* foram descritas no Capítulo 2.

Neste projeto utilizou-se Zend Framework 3. Apesar de Laravel Framework poder ter sido também uma boa alternativa, Zend Framework tem provado ser mais estável em termos suporte e atualizações ao longo dos anos, fazendo com que esta se torne uma das melhores *frameworks* para aplicações em ambiente *enterprise* [50]. A E-goi tem utilizado a Laravel Framework em alguns dos seus projetos mais recentes. No entanto, a sua plataforma principal continua a utilizar ZF. Para facilidade de integração e futura manutenção, foi

escolhida a ZF 3 porque tem todas as componentes necessárias para desenvolver uma API REST com os requisitos propostos [51].

4.1.2 – NGINX

Para servidor web utilizou-se NGINX. NGINX foi escolhido pois é um servidor de alto desempenho para aplicações web, e também porque a E-goi já utiliza esse mesmo servidor web na sua plataforma, o que facilita a implementação e manutenção da aplicação.

A utilização do servidor web é importante a nível de performance e rapidez de resposta da API REST, por isso é necessário configurar devidamente o servidor web. No entanto, a nível do desenvolvimento da aplicação, a escolha do servidor web é indiferente, porque o projeto é compatível com qualquer servidor web capaz de disponibilizar aplicações PHP, sendo possível migrar a aplicação para outro servidor web.

4.1.3 – MariaDB

A aplicação tem uma base de dados relacional para armazenar os países disponíveis para consulta, definições que permitem estruturar um bom pedido à API REST, os diferentes tipos de NIF ou número equivalente e os serviços que estão ativos ou inativos para validação de acordo com cada país. Na aplicação só foram implementadas funcionalidades de leitura da base dados e *queries* simples para interagir com a mesma. Optou-se pela utilização de Doctrine ORM e anotações nos modelos para comunicar com a base de dados relacional. MariaDB cumpre os requisitos necessários aplicacionais e de desempenho, e é a base de dados que a empresa já utiliza noutros projetos, facilitando o processo de manutenção e gestão de *backups*.

4.1.4 – OAS e Swagger

OAS é a especificação para documentação de APIs mais utilizada e "universal". OAS é vantajoso na medida em que é de fácil utilização e muitos programadores que pretendam integrar com a API REST da aplicação já estão familiarizados com este tipo de documentação.

Também de notar que Swagger tem ferramentas que permitem utilizar o ficheiro OAS da especificação da API para gerar código de cliente para várias linguagens de programação como PHP, Python, Perl, Java, etc [52].

4.1.5 – Análise das tecnologias escolhidas

Tendo em conta os requisitos e funcionalidades que a aplicação deve ter, as escolhas tecnológicas permitem a criação de uma aplicação que vai de encontro aos requisitos, já que é robusta e garante uma bom desempenho de serviço. Os detalhes da implementação e discussão de resultados vão ser analisados de seguida.

4.2 – Estrutura do Projeto

Um projeto Zend Framework 3 bem estruturado tem diferentes componentes são descritas nesta secção. Também é descrito como foi preparado um diretório para um projeto ZF 3, o diagrama de classes utilizado na implementação e a estrutura da base de dados relacional.

4.2.1 – Projeto Zend Framework 3

Uma aplicação desenvolvida com Zend Framework 3 segue uma arquitetura específica que está descrita na documentação oficial [51]. Numa aplicação ZF 3, existem diversos componentes e cada um desempenha uma funcionalidade distinta. Esta estrutura é fundamental porque permite organizar código de tal forma que facilita a compreensão e manutenção do mesmo por outros elementos da mesma equipa ou empresa.

Uma aplicação ou cliente num *browser* ativa o funcionamento do projeto quando acede a um recurso da API REST com um pedido HTTP. O *router* recebe esse pedido, a entrada, que depois desencadeia o funcionamento da aplicação, ou seja, o pedido é passado para as outras componentes como ilustrado na Figura 4.1.

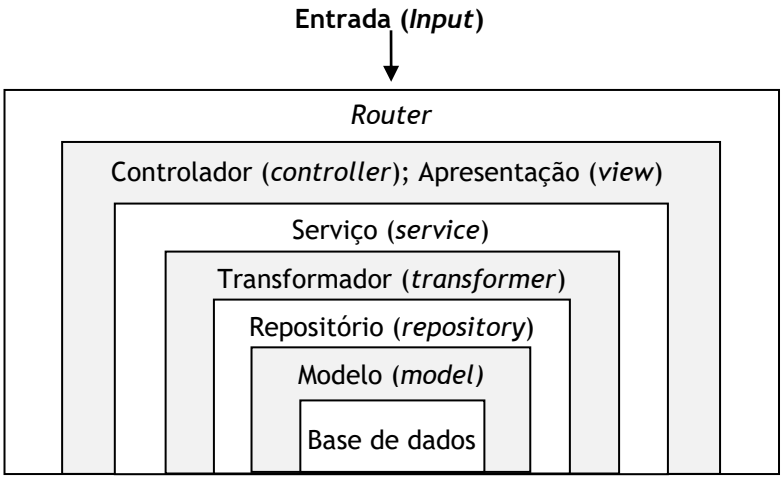

Figura 4.1 – Componentes de um projeto Zend Framework 3.

Router

O router é responsável por determinar o recurso (URI) que está a ser requisitado e associá-lo ao controlador que vai ser invocado para processar esse pedido. Se o recurso for válido, o controlador correspondente é instanciado e invocado.

Controlador (controller)

Nesta arquitetura, o controlador não possui lógica de negócio e é responsável por invocar o serviço adequado e fazer a ligação entre a camada de informação (transformador,

repositório, modelo e base de dados) e a apresentação. Assim que o controlador recebe um pedido estruturado do *router*, instancia o serviço apropriado que vai processá-lo.

Serviço (*service*)

O serviço é a componente que contém a lógica de negócio. De acordo com o pedido recebido pelo controlador, o serviço vai aceder à camada de informação (base de dados interna e/ou serviços de validação externos) para processar o pedido e devolver a resposta de sucesso ou erro ao controlador. O serviço invoca o repositório e passa um objeto ao transformador para ser transformado numa resposta JSON que a API REST retorna ao cliente.

Transformador (*transformer*)

Um transformador converte o objeto PHP devolvido pelo repositório numa estrutura de dados complexa no formato JSON que vai ser apresentada ao cliente [53]. Os transformadores são frequentemente utilizados em APIs REST para sistematizar a formatação da informação em JSON. Com os transformadores evita-se: (1) a repetição de código; (2) o acesso à informação dos objetos por propriedades; (3) a utilização de ciclos *foreach* dentro de ciclos *foreach*. Além disso, permite a associação de modelos complexos com recurso a inclusão.

Repositório (*repository*)

Um repositório contém métodos que permitem retornar os modelos que estão associados às tabelas armazenadas na base de dados. Por isso, quando é feita a requisição de um repositório, o modelo associado que está mapeado à base de dados retorna todas informações armazenadas e quaisquer associações entre tabelas (um para um, um para muitos, etc).

Modelo (*model*)

Cada modelo corresponde a uma tabela da base de dados. Os modelos usam anotações com Doctrine ORM que permitem associar cada modelo a uma tabela e cada variável a uma coluna da base de dados [54].

Base de dados

A base de dados relacional armazena a informação relativamente aos países, NIF ou número equivalente, serviços de validação disponíveis, etc. A estrutura da base de dados será analisada em mais detalhe mais à frente neste capítulo.

Apresentação (view)

Assim que o serviço (*service*) devolve uma resposta estruturada ao controlador, este último é responsável por passar a resposta de sucesso ou erro para a apresentação (*view*), que a retorna ao cliente que fez o pedido.

4.2.2 – Estrutura do diretório

Para manter o código organizado e separar as componentes analisadas na secção anterior, é necessário estruturar devidamente o diretório do projeto ZF 3 [55]. Um diretório de um projeto ZF 3 tem a estrutura apresentada na Figura 4.2.

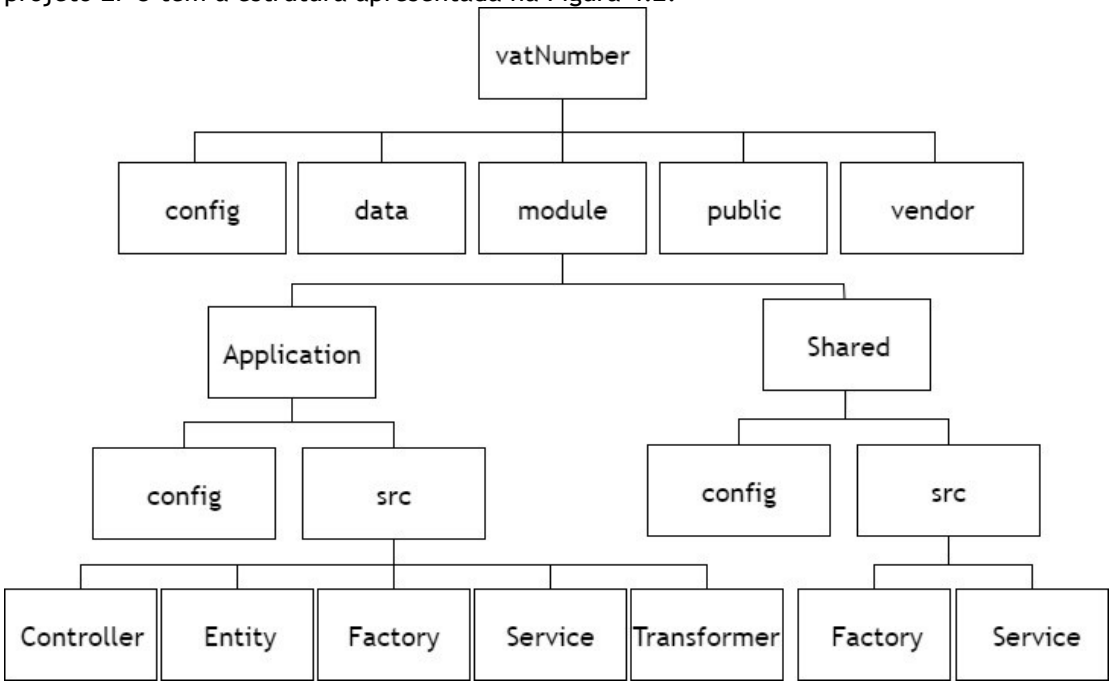

Figura 4.2 – Estrutura do diretório de um projeto Zend Framework 3.

O diretório principal tem o nome da aplicação, neste caso, *vatNumber*. Dentro do diretório *vatNumber* existem cinco pastas distintas:

- **config**: armazena as configurações gerais da aplicação. É utilizada para guardar as credenciais que estabelecem a comunicação com a base de dados (endereço de IP, utilizador e palavra-chave). Também carrega os diferentes módulos que o projeto utiliza, nomeadamente: DoctrineModule, Zend\Router, Zend\Validator, Application, Shared, ZF\ApiProblem, entre outros;
- **data:** contém um ficheiro do tipo SQL com a estrutura da criação da base de dados. Pode ser também utilizada para guardar ficheiros temporários ou de conteúdo volátil, como *cache*, *logs*, sessões, diferentes versões da estrutura da base de dados, etc;
- **module:** esta pasta contém todos os módulos desenvolvidos para uma aplicação ZF 3 em particular. Neste caso, foram implementados dois módulos *Application* e *Shared*. Para cada módulo existe um conjunto de pastas que permite armazenar as configurações do módulo. Também é necessário um conjunto de pastas para armazenar cada componente de código descrito na secção anterior;

- **public**: utilizado para armazenar ficheiros públicos da aplicação como: ficheiros de estilos do tipo CSS, documentação da API, imagens e quaisquer outros ficheiros que devem estar acessíveis publicamente;
- **vendor**: este diretório é utilizado para instalar as bibliotecas e módulos dos quais a aplicação depende.

4.2.3 – Diagrama de classes

Um diagrama de classes é uma forma de representar a relação entre as diferentes classes de um sistema, a Figura 4.3 ilustra o diagrama de classes da aplicação implementada. Nesta implementação, todos os controladores fazem *extend* à classe *AbstractRestfulJsonController* que contém os métodos para processar os pedidos HTTP GET e HTTP POST à API REST e as respostas de erro da API. Neste tipo de arquitetura, os controladores são só responsáveis por invocar os serviços, como mostra a Figura 4.3.

Quando se utiliza o método POST /validation da API REST, a classe *ValidationService* é invocada e posteriormente invoca os serviços de validação de existência que fazem pedidos a serviços, *ValidationExistsService*. Por outro lado, a classe *CountryService* e *VatRateService* invocam os transformadores dos países, que por sua vez acedem aos modelos que filtram e retornam a informação adequada de acordo com o pedido feito à API REST. Exemplo dessa utilização seria retornar os valores de IVA para um país específico.

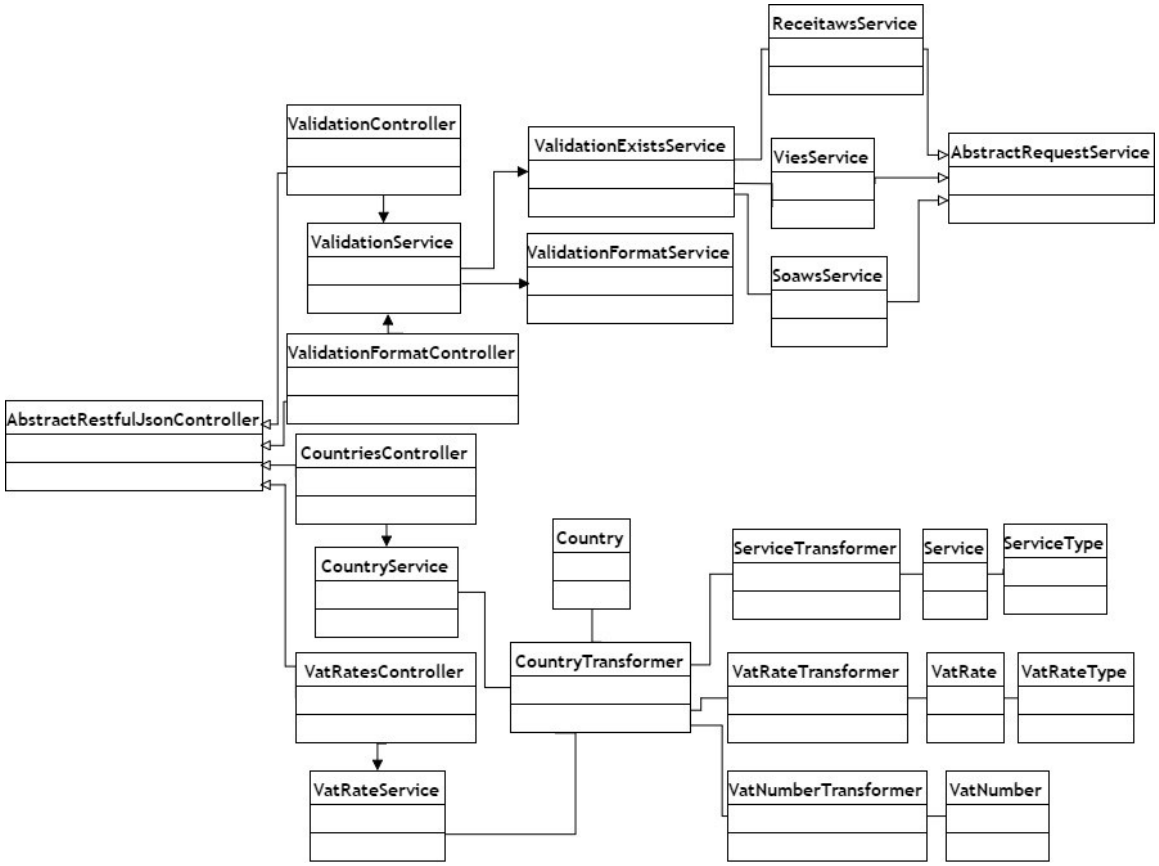

Figura 4.3 – Diagrama de classes.

4.2.4 – Modelo da base de dados

O projeto da dissertação utiliza uma base de dados relacional MariaDB e na qual foram criadas seis tabelas (ver Figura 4.4) que estão diretamente relacionadas com os modelos descritos no diagrama de classes da Figura 4.3.

O título de cada caixa contém o nome da tabela, neste caso *country*, *vatNumber*, *service*, *serviceType*, *vatRate* e *vatRateType*. Dentro de cada tabela existe uma lista de propriedades que indica as diferentes colunas de cada tabela e respetivos tipos de variáveis. Por exemplo, a tabela *service* contém: uma chave primária *id* que é do tipo inteiro; uma coluna *vatNumberId* do tipo inteiro que é uma chave estrangeira da tabela *vatNumber*; um *serviceTypeId* do tipo inteiro; e uma coluna booleana *active*. As linhas que unem as diferentes tabelas indicam as chaves estrangeiras das relações entre elas. Este conjunto de seis tabelas permite armazenar e consultar todas as informações necessárias para o funcionamento da aplicação.

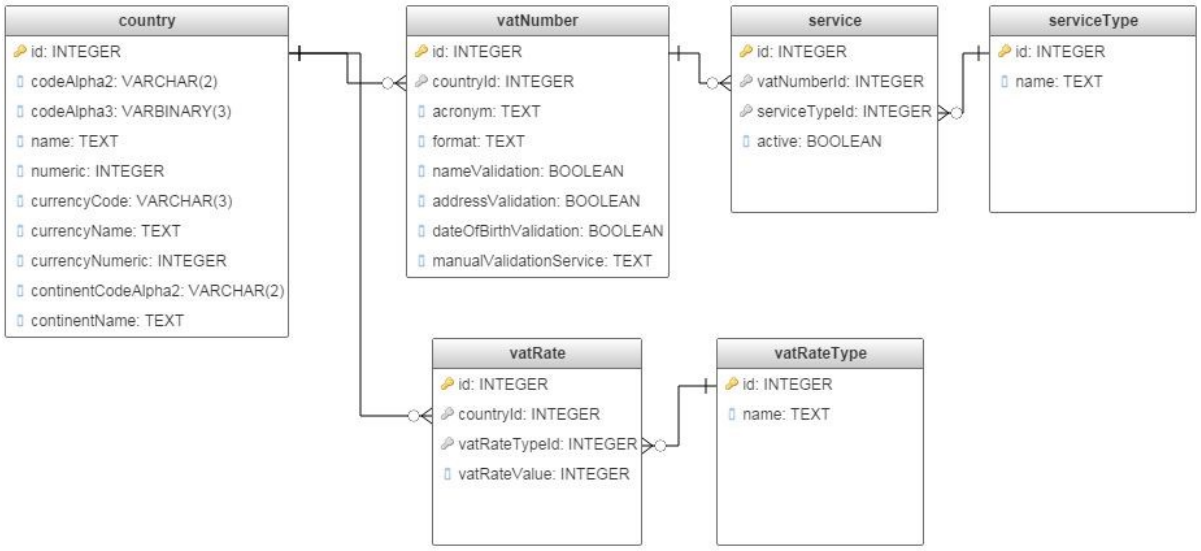

Figura 4.4 – Modelo da base de dados.

4.3 – Resultados

Implementada a solução, o passo seguinte é interpretar os resultados obtidos e avaliar se estes vão de encontro com o pretendido. Tendo em conta os métodos da API REST a implementar descritos no Capítulo 3, é possível verificar que todas as funcionalidades contidas nos requisitos propostos foram implementadas com sucesso. Essas funcionalidades estão devidamente ilustradas na Figura 4.5 que mostra a documentação simplificada da API com Swagger UI.

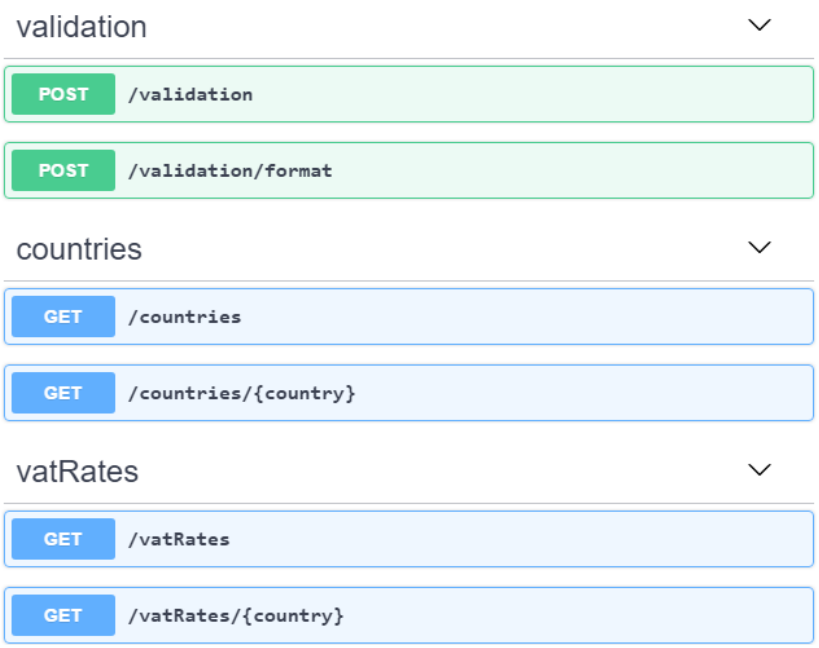

Figura 4.5 – Documentação da API REST com Swagger UI.

4.3.1 – Testes de funcionamento

Para cada método da API, foi criado um conjunto de pedidos teste para avaliar o funcionamento da aplicação, de acordo com o abordado no Capítulo 3, secção 3.2.4 "Métodos da API REST". Os pedidos teste foram efetuados recorrendo ao *software* Postman [56]. Os testes efetuados ao método POST /validation estão descritos na Tabela 4.1, onde se apresentam os tipos de teste e respetivo tipo de resposta (se é uma resposta de sucesso ou de erro), resposta esperada e a resposta obtida através da aplicação. O conjunto completo de testes efetuado a todos os métodos pode ser consultado no Anexo B.

Tabela 4.1 — Conjunto de testes e respetivos resultados para o método POST /validation da API REST.

Tipo de teste	Tipo de resposta	Resposta esperada	Resposta obtida
Validação de NIF válido	sucesso	true	true
Validação de NIF válido com nome e morada corretos	sucesso	true	true
Validação de NIF válido com nome incorreto	erro	false	false
Validação de NIF válido com nome incorreto e morada correta	erro	false	false
Validação de NIF inválido	erro	false	false
Validação de NIF a um país que não tem serviço de validação ativo	erro	false	false

Tipo de teste	Tipo de resposta	Resposta esperada	Resposta obtida
Validação de NIF com código de país inválido	erro	false	false
Validação de CPF válido com nome e data de nascimento	sucesso	true	true
Validação de CPF inválido	erro	false	false
Validação de CPF válido com data de nascimento inválida	erro	false	false
Validação de CPF inválido	erro	false	false
Validação de CNPJ válido com nome e morada	sucesso	true	true

Após testar esse conjunto de testes para cada método da API, verificou-se que a estrutura do pedido à API REST e as respostas de sucesso e erro foram implementas de acordo com a solução proposta.

4.3.2 – Tempo de resposta

O tempo de resposta da aplicação foi testado num computador com Linux instalado, distribuição Ubuntu 16.04. Note-se que o servidor que futuramente vai hospedar a versão final da aplicação dispõe de *hardware* mais rápido e uma melhor conexão à Internet.

Apesar das limitações de *hardware* durante a fase de testes, verificou-se que todos os pedidos que utilizavam apenas a lógica interna aplicação davam a resposta entre 20 a 30 milissegundos – o tempo de resposta da API foi testado utilizando o *software* Postman. Como já foi dito anteriormente, o único método implementado que utiliza serviços externos para validação da existência de NIF ou números equivalentes é o POST /validation. Este método está dependente do tempo que esses serviços demoram a responder à aplicação. Neste caso foram obtidos tempos de resposta da API entre 430 e 1100 milissegundos.

Após discussão com os elementos da E-goi, considerou-se que os tempos de resposta obtidos são satisfatórios. Além disso, estes serão inferiores em ambiente de produção. Tudo isto vai de encontro aos requisitos impostos pela empresa E-goi.

4.3.3 – Comparação de mercado

Comparativamente à solução implementada e ao que existe no mercado pode-se verificar que atualmente não existe nenhuma solução que permita validação de existência e formato de NIF ou equivalente para todos os países da União Europeia e Brasil no mesmo serviço.

No futuro, seria interessante encontrar mais serviços de validação para outros países e integrá-los com a solução desenvolvida nesta dissertação, o que tornaria esta aplicação uma

boa solução e ainda mais competitiva neste mercado, podendo até ser utilizada como uma solução comercial.

Capítulo 5

Conclusões

Nesta secção apresenta-se uma avaliação do trabalho desenvolvido para o problema proposto para esta dissertação: construir um sistema unificado de validação dos dados de um cliente ou empresa de qualquer parte do mundo a partir do NIF ou número equivalente. Faz-se também uma análise às limitações do trabalho desenvolvido, assim como sugestões de trabalhos futuros.

5.1 – Avaliação do Trabalho Desenvolvido

Para o desenvolvimento da API REST implementada nesta dissertação foi usado Zend Framework 3. No entanto, outras *frameworks* PHP poderiam ter sido uma melhor opção, uma vez que ZF 3 é demasiado complexo para o tipo de aplicação. Por exemplo, Laravel Framework proporcionava o mesmo resultado, mas com uma implementação mais simples e rápida. A escolha de ZF 3 era um dos requisitos do projeto por parte da E-goi, para facilitar a manutenção da aplicação por parte da sua equipa de programadores, uma vez que é a *framework* utilizada pela empresa.

Após análise dos resultados da aplicação implementada, descrito no Capítulo 4, os métodos da API REST desenvolvidos retornam as respostas de sucesso e erro esperadas. A rapidez de resposta da API satisfaz os requisitos impostos pela E-goi.

O projeto desenvolvido utiliza a base de dados relacional MariaDB, servidor web NGINX e a documentação da API REST segue a especificação OAS com Swagger UI. Todas estas tecnologias são robustas, apresenta bom desempenho, são escaláveis e facilitam a manutenção em ambiente de produção.

Por fim, é possível concluir que a API foi desenvolvida com sucesso: é funcional e permite validar NIF ou número equivalente a nível de existência e formato para qualquer país da UE e Brasil. É ainda possível validar o nome, morada e data de nascimento.

5.2 – Limitações do Trabalho Desenvolvido

Uma das limitações do serviço desenvolvido é estar dependente de serviços de consulta externos. Esse era o objetivo da dissertação, isto é, integrar com serviços externos. No entanto, a aplicação desenvolvida fica dependente de vários serviços que estão sujeitos a falhas, atrasos na resposta, alterações do serviço, podendo mesmo deixar de funcionar, sem que se possa ter qualquer controlo sobre eles. Tudo isto limita o desempenho da aplicação a nível de rapidez de resposta e questiona sobre problemas inesperados por falhas de serviços externos. Para contornar esta situação terá de haver uma manutenção mais frequente para controlar eventuais falhas ou alterações dos serviços externos.

Um dos desafios e limitações do trabalho desenvolvido está relacionado com a falta de serviços de validação para certos países. Após uma longa pesquisa conclui-se que a maioria dos países não disponibiliza um serviço para validação de existência de NIF ou número equivalente. Além disso, apesar de alguns países disponibilizarem serviços de consulta manual, muitos destes não têm a possibilidade de integração. No entanto, nesta dissertação o objetivo era integrar com todos os países da UE e Brasil, tarefa que foi concluída com sucesso.

5.3 – Sugestões de Trabalho Futuro

Todas as sugestões apresentadas como trabalho futuro são abordagens que não foram solicitadas para a finalidade desta dissertação. No entanto, as sugestões que se apresentam de seguida, foram devidamente discutidas na empresa, concluindo-se que são boas ideias para melhorar o serviço desenvolvido e as suas funcionalidades.

- **Disponibilizar o serviço gratuitamente para o exterior**: o CEO da empresa E-goi, Miguel Gonçalves, desde o início demonstrou vontade de desenvolver esta solução para ser disponibilizada *online* gratuitamente para outras empresas poderem usufruir do serviço. O serviço desenvolvido apresenta um grau de abstração suficiente para poder ser facilmente integrado noutros projetos. Apesar de tudo, esta ideia ficou por desenvolver por razões de tempo e calendário relativamente à E-goi. Para disponibilizar o serviço para o exterior seria necessário a criação de um website que explicasse o funcionamento e integração do mesmo para outros possíveis utilizadores;
- **Adicionar um mecanismo de criação de contas**: caso esta aplicação fosse disponibilizada *online*, a nível de funcionamento da aplicação, seria interessante introduzir um mecanismo de criação de contas para controlar os pedidos feitos à API para evitar o uso excessivo de pedidos;
- **Melhorar tempo de resposta**: para melhorar o tempo de resposta da aplicação, poderia ser utilizado um serviço de *Content Delivery Network* (CDN) que

distribuísse a aplicação por vários países. Assim, esta poderia estar hospedada no mesmo país que os serviços externos para diminuir o tempo de resposta;

- **Adicionar mais serviços ou algoritmos de validação de formato**: poderia ter sido feita uma pesquisa mais extensa a serviços de existência pagos de outros países, para integrar e tornar a solução mais universal;
- **Desenhar uma interface gráfica para alterar informações da base de dados e adicionar novos serviços**: uma vez que a aplicação desenvolvida é modular faz com que a sua integração com novos serviços seja fácil. Assim, poderia ser implementada uma interface gráfica que permita alterar informações da base de dados e adicionar novos serviços de validação sem recorrer a alterações no código.

Referências

[1] Público, “Empresas e clientes sempre ligados? A E-goi faz a magia - PÚBLICO.” [Online]. Available: https://www.publico.pt/2017/05/22/conteudo-patrocinado/noticia/empresas-e-clientes-sempre-ligados-a-egoi-faz-a-magia-1772460. [Accessed: 30-May-2017].

[2] “AMAZON.COM ANNOUNCES FOURTH QUARTER SALES UP 22% TO $43.7 BILLION.” [Online]. Available: https://www.sec.gov/Archives/edgar/data/1018724/000101872417000003/amzn-20161231xex991.htm. [Accessed: 31-May-2017].

[3] M. Tsenov, “Web Services Example with PHP/SOAP,” 2006.

[4] R. T. Fielding, “UNIVERSITY OF CALIFORNIA, IRVINE Architectural Styles and the Design of Network-based Software Architectures,” 2000.

[5] “What is REST.” [Online]. Available: http://restfulapi.net/. [Accessed: 29-May-2017].

[6] P. Adamczyk, P. H. Smith, R. E. Johnson, and M. Hafiz, “REST: From Research to Practice,” 2011.

[7] P. J. Leach, T. Berners-Lee, J. C. Mogul, L. Masinter, R. T. Fielding, and J. Gettys, “Hypertext Transfer Protocol -- HTTP/1.1.”

[8] C. Rodr *et al.*, “REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices,” vol. 9671, pp. 21–39, 2016.

[9] “What is XML-RPC.” [Online]. Available: http://xmlrpc.scripting.com/. [Accessed: 29-May-2017].

[10] P. G. Soares, “On Remote Procedure Call *.”

[11] M. Allman, “An evaluation of XML-RPC,” *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 4, pp. 2–11, 2003.

[12] “SOAP Version 1.2 Part 1: Messaging Framework (Second Edition).” [Online]. Available: https://www.w3.org/TR/soap12/. [Accessed: 29-May-2017].

[13] K. A. Kadouh and K. A. Albashiri, “Improvement of Data Transfer over Simple Object Access Protocol (SOAP),” *Int. J. Comput. Inf. Sci. Eng.*, vol. 8, no. 2, pp. 16–19, 2014.

[14] Derek Ferguson, “Exclusive .NET Developer’s Journal "Indigo" Interview with Microsoft’s Don Box | Microsoft Cloud.” [Online]. Available: http://dotnet.sys-con.com/node/45908. [Accessed: 31-May-2017].

[15] S. Marlow, “Developing a High-Performance Web Server in Concurrent Haskell.”

[16] Netcraft, “July 2016 Web Server Survey | Netcraft.” [Online]. Available: https://news.netcraft.com/archives/2016/07/19/july-2016-web-server-survey.html. [Accessed: 30-May-2017].

[17] DreamHost, “Web server performance comparison – DreamHost.” [Online]. Available: https://help.dreamhost.com/hc/en-us/articles/215945987-Web-server-performance-comparison. [Accessed: 30-May-2017].

[18] “Zend Server - The fastest way to enterprise PHP.” [Online]. Available: http://www.zend.com/en/products/zend_server. [Accessed: 31-May-2017].

[19] “PHP: History of PHP - Manual.” [Online]. Available: http://php.net/manual/en/history.php.php. [Accessed: 29-May-2017].

[20] M. Kalelkar, P. Churi, and D. Kalelkar, “Implementation of Model-View-Controller

Architecture Pattern for Business Intelligence Architecture," *Int. J. Comput. Appl.*, vol. 102, no. 12, pp. 975–8887, 2014.

[21] "The great PHP MVC Framework Showdown of 2016 - (CakePHP 3 vs Symfony 2 vs Laravel 5 vs Zend 2) - zen of coding." [Online]. Available: http://zenofcoding.com/2015/11/16/the-great-php-mvc-framework-showdown-of-2016-cakephp-3-vs-symfony-2-vs-laravel-5-vs-zend-2/. [Accessed: 18-Jun-2017].

[22] a R. W. C. Zend, "Zend Framework Tutorials and Documentation." [Online]. Available: https://docs.zendframework.com/tutorials/. [Accessed: 31-May-2017].

[23] Fryan at Red Monk, "Language Framework Popularity: A Look at PHP – Charting Stacks." [Online]. Available: http://redmonk.com/fryan/2016/11/01/language-framework-popularity-a-look-at-php/. [Accessed: 31-May-2017].

[24] Yii Framework, "Yii 1.0.0 is released | News | Yii PHP Framework." [Online]. Available: http://www.yiiframework.com/news/3/yii-1-0-0-is-released. [Accessed: 31-May-2017].

[25] A. Jumppanen and H. Landvik, "WEB APPLICATION WITH YII FRAMEWORK WEB-SOVELLUS YII-SOVELLUSKEHYKSELLÄ."

[26] "Top 7 Best PHP Frameworks for 2017 | Archer Software." [Online]. Available: http://www.archer-soft.com/en/blog/top-7-best-php-frameworks-2017. [Accessed: 08-Jul-2017].

[27] P. Baxendale and E. F. Codd, "A Relational Model of Data Large Shared Data Banks."

[28] "MySQL." [Online]. Available: https://www.mysql.com/. [Accessed: 01-Jun-2017].

[29] "DB-Engines Ranking - popularity ranking of database management systems." [Online]. Available: https://db-engines.com/en/ranking. [Accessed: 15-May-2017].

[30] "About MariaDB - MariaDB.org." [Online]. Available: https://mariadb.org/about/. [Accessed: 01-Jun-2017].

[31] "PostgreSQL: About." [Online]. Available: https://www.postgresql.org/about/. [Accessed: 01-Jun-2017].

[32] "Doctrine Project." [Online]. Available: http://www.doctrine-project.org/. [Accessed: 03-Feb-2017].

[33] S. Judd, "Documenting REST APIs - a tooling review - OpenCredo." [Online]. Available: https://opencredo.com/rest-api-tooling-review/. [Accessed: 09-Jun-2017].

[34] "About | Open API Initiative." [Online]. Available: https://www.openapis.org/about. [Accessed: 31-May-2017].

[35] "Swagger – The World's Most Popular Framework for APIs." [Online]. Available: http://swagger.io/. [Accessed: 31-May-2017].

[36] "Swagger UI." [Online]. Available: http://swagger.io/swagger-ui/. [Accessed: 31-May-2017].

[37] "API Blueprint | API Blueprint." [Online]. Available: https://apiblueprint.org/. [Accessed: 31-May-2017].

[38] "RAML | The simplest way to design APIs." [Online]. Available: http://raml.org/. [Accessed: 31-May-2017].

[39] "VIES FAQ." [Online]. Available: http://ec.europa.eu/taxation_customs/vies/faq.html. [Accessed: 02-Feb-2017].

[40] "O que é um Contribuinte? - NIF." [Online]. Available: http://www.nif.pt/o-que-e-um-contribuinte/. [Accessed: 30-May-2017].

[41] "NIF das Empresas - NIF." [Online]. Available: http://www.nif.pt/nif-das-empresas/. [Accessed: 01-Feb-2017].

[42] "Tax identification numbers (TIN) - European Commission." [Online]. Available: https://ec.europa.eu/taxation_customs/business/tax-cooperation-control/administrative-cooperation/tax-identification-numbers-tin_en. [Accessed: 30-May-2017].

[43] "Employer ID Numbers - EIN." [Online]. Available: https://www.irs.gov/businesses/small-businesses-self-employed/employer-id-numbers-eins. [Accessed: 02-Feb-2017].

[44] "Comprovante de Situação Cadastral no CPF." [Online]. Available: https://www.receita.fazenda.gov.br/Aplicacoes/SSL/ATCTA/CPF/ConsultaSituacao/ConsultaPublica.asp. [Accessed: 02-Feb-2017].

[45] "PHP: String Functions - Manual." [Online]. Available:

http://php.net/manual/en/ref.strings.php. [Accessed: 01-May-2017].
[46] “Cross-border VAT rates in Europe - Your Europe - Business.” [Online]. Available: http://europa.eu/youreurope/business/vat-customs/cross-border/index_en.htm. [Accessed: 02-Feb-2017].
[47] “Expresso | Santander alerta para email falso.” [Online]. Available: http://expresso.sapo.pt/sociedade/2017-05-08-Santander-alerta-para-email-falso. [Accessed: 13-Jun-2017].
[48] N. Alshuqayran, N. Ali, and R. Evans, “A Systematic Mapping Study in Microservice Architecture.”
[49] C. Pautasso, O. Zimmermann, and F. Leymann, “RESTful Web Services vs. " Big " Web Services: Making the Right Architectural Decision,” 2008.
[50] “What PHP Framework Should I Choose? - WebiNerds.” [Online]. Available: https://webinerds.com/choose-php-application-framework/. [Accessed: 11-Jun-2017].
[51] “Zend Framework Components.” [Online]. Available: https://docs.zendframework.com/. [Accessed: 09-Jun-2017].
[52] “Swagger Codegen.” [Online]. Available: http://swagger.io/swagger-codegen/. [Accessed: 11-Jun-2017].
[53] “Fractal - Output complex, flexible, AJAX/RESTful data structures.” [Online]. Available: http://fractal.thephpleague.com/. [Accessed: 10-Jun-2017].
[54] “21. Annotations Reference — Doctrine 2 ORM 2 documentation.” [Online]. Available: http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/annotations-reference.html. [Accessed: 10-Jun-2017].
[55] “Introduction - zend-mvc.” [Online]. Available: https://docs.zendframework.com/zend-mvc/intro/. [Accessed: 11-Jun-2017].
[56] “Postman | Supercharge your API workflow.” [Online]. Available: https://www.getpostman.com/. [Accessed: 11-Jun-2017].
[57] “VIES Validation.” [Online]. Available: http://ec.europa.eu/taxation_customs/vies/vatRequest.html. [Accessed: 02-Feb-2017].
[58] “Emissão de Comprovante de Inscrição e de Situação Cadastral.” [Online]. Available: https://www.receita.fazenda.gov.br/pessoajuridica/cnpj/cnpjreva/cnpjreva_solicitacao.asp. [Accessed: 02-Feb-2017].
[59] “ReceitaWS.” [Online]. Available: https://www.receitaws.com.br/. [Accessed: 04-Feb-2017].
[60] “SOAWebServices, maior hub de webservices brasileiro.” [Online]. Available: https://www.soawebservices.com.br/. [Accessed: 15-Mar-2017].
[61] “Rutificador | Buscador de CUIT y DNI argentinos a través del nombre.” [Online]. Available: http://argentina.rutificador.com/. [Accessed: 15-Mar-2017].
[62] “Rutificador de nombres totalmente gratuito para RUT chilenos.” [Online]. Available: https://chile.rutificador.com/. [Accessed: 17-Mar-2017].
[63] “validanit.co , el sitio para verificar NIT.” [Online]. Available: http://validanit.co/. [Accessed: 15-Jun-2017].
[64] “Rutificador | Buscador de CI costarricense través del nombre.” [Online]. Available: http://costarica.rutificador.com/. [Accessed: 15-Mar-2017].
[65] “Rutificador | Buscador de TDI hondureños a través del nombre.” [Online]. Available: http://honduras.rutificador.com/. [Accessed: 15-Mar-2017].
[66] “Rutificador | Buscador de CIC paraguayos a través del nombre.” [Online]. Available: http://paraguay.rutificador.com/. [Accessed: 14-Mar-2017].
[67] “Consulta de RIF.” [Online]. Available: http://contribuyente.seniat.gob.ve/BuscaRif/BuscaRif.jsp. [Accessed: 20-Mar-2017].

Anexo A

NIF ou Número Equivalente

Tabela A.1 — Dados armazenados numa tabela da base de dados com informação relativa aos NIFs ou números equivalentes. Cada tipo de NIF tem um acrónimo e corresponde a um país. Permite verificar se existe serviço de validação e o que é possível validar (nome, morada ou data de nascimento).

Designação do NIF	Nome	Código de país	Validação				
			Serviço	Nome	Morada	Data de nascimento	Formato
USt	Austria	AT	[57]	SIM	SIM	NÃO	SIM
BTW - TVA - NWSt	Belgium	BE	[57]	SIM	SIM	NÃO	SIM
VAT	Bulgaria	BG	[57]	SIM	SIM	NÃO	SIM
PDV	Croatia	HR	[57]	SIM	SIM	NÃO	SIM
VAT	Cyprus	CY	[57]	SIM	SIM	NÃO	SIM
DPH	Czech Republic	CZ	[57]	SIM	SIM	NÃO	SIM
moms	Denmark	DK	[57]	SIM	SIM	NÃO	SIM
km	Estonia	EE	[57]	SIM	SIM	NÃO	SIM
ALV - Moms	Finland	FI	[57]	SIM	SIM	NÃO	SIM
TVA	France	FR	[57]	SIM	SIM	NÃO	SIM
MwSt. - Ust.	Germany	DE	[57]	SIM	SIM	NÃO	SIM
VAT	Greece	GR	[57]	SIM	SIM	NÃO	SIM
ANUM	Hungary	HU	[57]	SIM	SIM	NÃO	SIM
CBL - VAT	Ireland	IE	[57]	SIM	SIM	NÃO	SIM
CBL - VAT	Ireland	IE	[57]	SIM	SIM	NÃO	SIM
IVA	Italy	IT	[57]	SIM	SIM	NÃO	SIM
PVN	Latvia	LV	[57]	SIM	SIM	NÃO	SIM
PVM	Lithuania	LT	[57]	SIM	SIM	NÃO	SIM

Designação do NIF	Nome	Código de país	Validação				
			Serviço	Nome	Morada	Data de nascimento	Formato
TVA	Luxembourg	LU	[57]	SIM	SIM	NÃO	SIM
VAT	Malta	MT	[57]	SIM	SIM	NÃO	SIM
BTW - TVA - NWSt	Netherlands	NL	[57]	SIM	SIM	NÃO	SIM
PTU - VAT	Poland	PL	[57]	SIM	SIM	NÃO	SIM
NIF	Portugal	PT	[57]	SIM	SIM	NÃO	SIM
NIPC	Portugal	PT	[57]	SIM	SIM	NÃO	SIM
CIF	Romania	RO	[57]	SIM	SIM	NÃO	SIM
DPH	Slovakia	SK	[57]	SIM	SIM	NÃO	SIM
DPH	Slovenia	SI	[57]	SIM	SIM	NÃO	SIM
NIF (CIF)	Spain	ES	[57]	SIM	SIM	NÃO	SIM
Moms	Sweden	SE	[57]	SIM	SIM	NÃO	SIM
VAT Reg No	United Kingdom and Isle of Man	GB	[57]	SIM	SIM	NÃO	SIM
CNPJ	Brazil	BR	[58] [59]	SIM	SIM	NÃO	SIM
CPF	Brazil	BR	[44] [60]	SIM	NÃO	SIM	SIM
NIPT	Albania	AL		NÃO	NÃO	NÃO	NÃO
ABN	Australia	AU		NÃO	NÃO	NÃO	NÃO
UNP	Belarus	BY		NÃO	NÃO	NÃO	NÃO
BN / NE	Canada	CA		NÃO	NÃO	NÃO	NÃO
VSK / VASK	Iceland	IS		NÃO	NÃO	NÃO	NÃO
VAT TIN / CST TIN	India	IN		NÃO	NÃO	NÃO	NÃO
NPWP	Indonesia	ID		NÃO	NÃO	NÃO	NÃO
Orgnr	Norway	NO		NÃO	NÃO	NÃO	NÃO
TIN	Philippines	PH		NÃO	NÃO	NÃO	NÃO
INN	Russia	RU		NÃO	NÃO	NÃO	NÃO
C.O.E.	San Marino	SM		NÃO	NÃO	NÃO	NÃO
PIB	Serbia	RS		NÃO	NÃO	NÃO	NÃO
MWST/TVA/IVA	Switzerland	CH		NÃO	NÃO	NÃO	NÃO
INPP	Ukraine	UA		NÃO	NÃO	NÃO	NÃO
Stir	Uzbekistan	UZ		NÃO	NÃO	NÃO	NÃO
CUIT	Argentina	AR	[61]	NÃO	NÃO	NÃO	NÃO
NIT	Bolivia	BO		NÃO	NÃO	NÃO	NÃO

Designação do NIF	Nome	Código de país	Validação				
			Serviço	Nome	Morada	Data de nascimento	Formato
RUT	Chile	CL	[62]	NÃO	NÃO	NÃO	NÃO
NIT	Colombia	CO	[63]	NÃO	NÃO	NÃO	NÃO
CI	Costa Rica	CR	[64]	NÃO	NÃO	NÃO	NÃO
RUC	Ecuador	EC		NÃO	NÃO	NÃO	NÃO
NIT	Guatemala	GT		NÃO	NÃO	NÃO	NÃO
TDI	Honduras	HN	[65]	NÃO	NÃO	NÃO	NÃO
RFC	Mexico	MX		NÃO	NÃO	NÃO	NÃO
RUC	Nicaragua	NI		NÃO	NÃO	NÃO	NÃO
RUC	Panama	PA		NÃO	NÃO	NÃO	NÃO
CIC	Paraguay	PY	[66]	NÃO	NÃO	NÃO	NÃO
DNI	Peru	PE		NÃO	NÃO	NÃO	NÃO
RNC	Dominican Republic	DO		NÃO	NÃO	NÃO	NÃO
RUT	Uruguay	UY		NÃO	NÃO	NÃO	NÃO
RIF	Venezuela	VE	[67]	NÃO	NÃO	NÃO	NÃO

Anexo B

Conjunto de Testes à API REST

Tabela B.1 — Conjunto de testes e respetivos resultados para o método POST /validation/format da API REST.

Tipo de teste	Tipo de resposta	Resposta esperada	Resposta obtida
Validação de NIF válido	sucesso	true	true
Validação de NIF inválido	erro	false	false
Validação de NIF a um país sem serviço de validação ativo	erro	false	false
Validação de NIF com código de país inválido	erro	false	false
Validação de CPF válido	sucesso	true	true
Validação de CPF inválido	erro	false	false
Validação de CNPJ válido	sucesso	true	true
Validação de CNPJ inválido	erro	false	false

Tabela B.2 — Conjunto de testes e respetivos resultados para os métodos GET /countries e GET /countries/<country> da API REST.

Tipo de teste	Tipo de resposta	Resposta esperada	Resposta obtida
Consulta da informação de todos os países com pedido GET /countries	sucesso	true	true
Consulta da informação de Portugal com pedido GET /countries/PT	sucesso	true	true
Consulta da informação do Brasil com	sucesso	true	true

Tipo de teste	Tipo de resposta	Resposta esperada	Resposta obtida
pedido GET /countries/BR			
Consulta da informação de um país com código de país inválido, exemplo: GET /countries/PO	erro	false	false

Tabela B.3 — Conjunto de testes e respetivos resultados para os métodos GET /vatRates e GET /vatRates/<country> da API REST.

Tipo de teste	Tipo de resposta	Resposta esperada	Resposta obtida
Consulta dos valores de IVA para todos os países com pedido GET /vatRates	sucesso	true	true
Consulta dos valores de IVA para Portugal com pedido GET /vatRates/PT	sucesso	true	true
Consulta dos valores de IVA para Espanha com pedido GET /vatRates/ES	sucesso	true	true
Consulta da informação de um país com código de país inválido, exemplo: GET /vatRates/PO	erro	false	false