INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




# Gestão e monitorização de API

**CATARINA CARDOSO DIAS DE SOUSA**
Outubro de 2020

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

# API Management & Monitoring

**Catarina Sousa**

**Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de Software**

**Orientador: Ricardo Almeida**

**Coorientador: Paulo Proença**

**Supervisor: Ivo Pereira**

Porto, outubro 2020

# Dedicatória

Dedico a toda as pessoas que me ajudaram na conclusão desta etapa da minha vida.

# Resumo

Durante o processo de criação de um produto existem várias etapas, nomeadamente, análise da viabilidade e utilidade, planeamento do desenvolvimento, implementação da solução e avaliação do produto final (tendo em conta padrões definidos).

No presente documento é descrito o processo de criação de uma ferramenta de monitorização e gestão da API, da empresa E-goi. Esta conta com vários produtos, contudo o principal é uma plataforma de automatização de marketing multicanal.

Neste documento o processo é apresentado através de uma introdução, contextualização, estado de arte, análise de valor, do design da solução, da implementação e da experimentação.

Na introdução foi descrito todo o contexto do problema e do projeto.

No caso da contextualização descreveu-se todo o conhecimento necessário para entender o lado técnico do problema, nomeadamente API, SDK, gestão e monitorização de API.

Com o estado de arte, pretendeu-se apresentar os critérios necessários para uma solução que resolva o problema e comparou-se a ferramentas no mercado. Neste caso foram analisados o OpenAPI Generator, APIMAtic e o Assertible.

Em relação à análise de valor foi analisada a solução, na perspetiva do valor que irá trazer à empresa onde foi concebida, através do *New Concept Developement* (NCD).

O design da solução descreveu todo o planeamento elaborado para a implementação da mesma, como os requisitos funcionais e não funcionais, domínio, arquitetura, implantação e serviços.

Na implementação detalhou-se toda a solução numa perspetiva técnica, nomeadamente a interface gráfica, serviços e pipeline de SDK.

E por fim, a experimentação, examinou a solução encontrada segundo um conjunto de testes de software e usabilidade.

Este trabalho deu origem a um artigo de investigação submetido ao simpósio de engenharia informática 20-21.

**Palavras-chave**: Monitorização, Gestão, API

# Abstract

During the process of creating a product, there are several steps, namely, feasibility and utility analysis, development planning, solution implementation and evaluation of the final product (considering defined standards).

This document describes the process of creating an API monitoring and management tool, by the company E-goi. The company has several products, however the main one is a multichannel marketing automation platform.

In this document, the process is presented through an introduction, contextualization, state of the art, value analysis, solution design, implementation, and experimentation.

In the introduction was described the entire context of the problem and the project.

In the case of contextualization, all the knowledge necessary to understand the technical side of the problem was described, namely API, SDK, API management and monitoring.

With the state of the art, it was intended to present the necessary criteria for a solution that solves the problem and compared it to tools on the market. In this case, OpenAPI Generator, APIMAtic and Assertible were the ones analyzed.

In relation to value analysis, the solution was analyzed, from the perspective of the value it will bring to the company where it was conceived, through the New Concept Developement (NCD).

The solution design described all the planning developed for its implementation, such as functional and non-functional requirements, domain, architecture, deployment, and services.

In the implementation, the entire solution was detailed from a technical perspective, namely the graphical interface, services, and SDK pipeline.

Finally, the experimentation examined the solution found according to a set of software and usability tests.

This work gave rise to a research article submitted to the computer engineering symposium 20-21.

# Agradecimentos

Primeiramente gostaria de agradecer aos professores Ricardo Almeida e Paulo Proença que sempre estiveram disponíveis para me orientar, acompanhar e permitiram que este documento tenha existido.

Em segundo gostaria de agradecer à empresa onde foi realizado o projeto descrito neste documento, a E-goi, que possibilitou a oportunidade e os meios para o mesmo.

Também queria agradecer a todos colaboradores da E-goi pela integração na empresa e disponibilidade para ajudar.

E por último à minha família e amigos que sempre me apoiaram e me fizeram chegar a onde cheguei.

# Índice

# Lista de Figuras

# Lista de Tabelas

# Lista de Acrónimos e Siglas

| | |
|---|---|
| **ABAC** | Attribute Based Access Control |
| **ACID** | Atomicity, Consistency, Isolation, Durability |
| **AMQP** | Advanced Message Queuing Protocol |
| **API** | Application Programming Interface |
| **CRM** | Customer Relationship Management |
| **CSV** | Comma-separated values |
| **cURL** | Client Uniform Resource Locator |
| **HTML** | HyperText Markup Language |
| **HTTP** | Hypertext Transfer Protocol |
| **IDE** | Integrated Development Environment |
| **JDBC** | Java Database Connectivity |
| **JSON** | JavaScript Object Notation |
| **PKI** | Public Key Infrastructure |
| **RBAC** | Role Based Access Control |
| **REST** | Representational State Transfer |
| **SDK** | Software Development Kits |
| **SMTP** | Simple Mail Transfer Protocol |
| **SO** | Sistema Operativo |
| **SOAP** | Simple Object Access Protocol |
| **SSL** | Secure Socket Layer |
| **SQL** | Structured Query Language |
| **SWOT** | Strengths Weaknesses Opportunities Threats |
| **UI** | User Interface |
| **URI** | Uniform Resource Identifier |
| **URL** | Uniform Resource Locator |
| **XML** | Extensible Markup Language |
| **XMPP** | Extensible Messaging and Presence Protocol |

# Capítulo 1

# Introdução

Nesta era digital, cada vez mais as empresas procuram soluções que respondam às suas necessidades tecnológicas. Segundo um estudo realizado pela Cloud Elements (empresa de serviços de apoio a API), cinquenta e cinco por cento dos inquiridos afirmaram que a integração com interfaces de programação de aplicações (API) é de importância vital para a estratégia empresarial e vinte e nove por cento afirmaram que é importante[1].

## 1.1 Contexto

Nos dias de hoje as empresas, para competir no mercado, necessitam de soluções tecnológicas. Uma das tecnologias que apresenta grande importância para uma empresa são as API [2].

Uma API é um conjunto de protocolos e definições que processa pedidos. É uma forma de trocar dados entre aplicações e dispositivos, facilitando a utilização do *software* de aplicações por outros sistemas [3] [4]. Sendo as *User Interfaces* (UI) a camada visual de uma tecnologia (*site*, aplicação, entre outras) que responde às ordens de um utilizador [5], é possível fazer uma comparação entre UI e API. Tal como as UI são um meio de comunicação entre um utilizador e *software*, as API são-no entre as máquinas e *software* [6]. No caso de empresas como a Google, Microsoft e Apple, o uso de API, possibilitou a criação de aplicações (software) compatíveis, entre sistemas operativos (de dispositivos), como Android, iOs e Windows [7]. Esta tecnologia, destaca-se pela capacidade de tornar sistemas mais responsivos, adaptáveis e simples [3].

Com o aumento da utilização de uma API, aumenta consequentemente a necessidade da monitorização e gestão da mesma. A monitorização avalia a atividade e alerta para problemas. Desta forma é possível garantir o bom desempenho da API [8]. A gestão é constituída por um conjunto de processos, nomeadamente distribuição, controlo e análise. Algumas das vantagens, da gestão, são a centralização de processos, controlo da documentação, otimização e a publicação fácil da API [9] [10].

## 1.2 Problema

A plataforma de automação de marketing multicanal da E-goi, líder em Portugal, envia milhares de campanhas todos os dias, repartidas pelos vários canais de comunicação.

Para tal, são criadas centenas de novas contas na plataforma da E-goi. Muitos dos utilizadores usufruem da possibilidade de integrar o produto E-goi com outras plataformas. Havendo atualmente centenas de integrações e milhões de pedidos diários, implica garantir o desempenho do produto, crítico para o negócio da empresa. Todo este tráfego é efetuado através de uma API de acesso público. Devido ao rápido crescimento da plataforma torna-se cada vez mais difícil detetar e solucionar problemas na mesma.

A monitorização permite saber o estado da API e consequentemente obter uma resposta rápida a problemas por parte dos *developers* [8]. A gestão permite facilitar, para a equipa de desenvolvimento, os processos relacionados com a API como a publicação, a segurança e documentação [10]. A eficiência na resolução de erros e execução de processos, proporciona diminuição do tempo de inatividade do produto. Esta solução, permite aos clientes, realizar todas as suas tarefas nos produtos sem correrem o risco de não as conseguirem terminar ou mesmo começar. As empresas obtêm assim satisfação dos clientes e evitam a perda dos mesmos.

Com a implementação da monitorização e da gestão é possível responder ao problema do aumento do uso da API.

## 1.3 Objetivos

O projeto descrito neste documento, tem como objetivo geral desenvolver uma ferramenta que ajude a melhorar a performance da API, e consequentemente a da plataforma E-goi. De forma a atingir o objetivo final serão realizadas as seguintes etapas:

1º) Análise da API publica da empresa E-goi, juntamente com a estrutura tecnológica

2º) Definição da solução e levantamento de critérios

3º) Experimentação de ferramentas de gestão e monitorização da API e geração de SDK

4º) Definição do design da solução

5º) Implementação da solução e seleção da ferramenta de geração de SDK

6º) Definição de testes e avaliação da solução

A definição de objetivos facilita a gestão do projeto, e consequentemente, simplifica e aumenta a eficiência do processo de desenvolvimento.

## 1.4 Apresentação da Empresa

A empresa onde foi realizado o projeto intitula-se E-goi, é uma empresa da área de marketing digital, sediada em Matosinhos. Criada em 2001, inicialmente intitulada

Maxideia, a empresa tem vindo a crescer ao longo dos anos e conta atualmente com uma equipa multigeracional e internacional de aproximadamente 100 colaboradores. A missão da E-goi é "*criar soluções de comunicação digital, acessíveis a todos os profissionais de marketing do mundo e que se divirtam a usar desafiando-nos todos os dias*" [1]. A visão da empresa é que "*as marcas as entendam cada vez mais as pessoas, oferecendo mais valor*" [1].

A E-goi disponibiliza aos seus clientes uma plataforma de marketing multicanal que permite aos seus clientes criar de listas de contactos, criar e enviar de newsletters, enviar de sms, criar notificações via web, monitorizar o percurso dos clientes num *site*, criar e gerir formulários. É possível também fazer integrações com o *Facebook*, *Shopify*, *Wordpress*, entre outras. A integração do produto E-goi com o Facebook conta com funcionalidades como o *Facebook Leads Ads*, o *Facebook Audiences* e o *Facebook Instant Articles*. O *Facebook Leads Ads* permite criar anúncios e contribui para aumentar a lista de contactos de um utilizador. O *Facebook Audiences* possibilita originar anúncios personalizados para os segmentos da lista de envios da conta E-goi. E o *Facebook Instant Articles* permite publicar formulários no *Facebook* [11]. Estas integrações pretendem apoiar os utilizadores do produto E-goi com outras plataformas.

## 1.5 Metodologia

O projeto descrito neste documento irá ser desenvolvido em ambiente empresarial. A sua realização estará divida nas seguintes etapas:

- **Análise de requisitos** – onde será feito o levantamento dos critérios necessários para a resolução do problema em causa.
- **Análise** – em que irá ser realizado uma pesquisa de soluções possíveis para o problema.
- **Design da solução** – no qual irá ser feito um planeamento de como irá ser concebida a solução.
- **Desenvolvimento** – fase de execução da solução, encontrada nas etapas anteriores.
- **Testes** – etapa de verificação e validação da solução [12].

Estas fases irão ser executadas segundo uma metodologia de trabalho mista, de cascata e *scrum*. A metodologia em cascata supõe um desenvolvimento linear, sequencial e por etapas [13]. S*crum* é uma metodologia, em que o desenvolvimento é feito de forma iterativa [14]. Durante a fase de identificação de requisitos, análise e design da solução irá ser aplicada a metodologia em cascata, pois permite ter os requisitos bem definidos

[1] Frase retirada de um PowerPoint de apresentação da empresa a novos colaboradores

e todas as fases são acompanhadas de documentação [15]. Nas fases de desenvolvimento e testes irá ser utilizada a metodologia *scrum*, com *sprints* de 3 semanas, porque é fácil adaptar a mudanças de requisitos e maximiza o tempo do processo de execução da solução [15].

Para além da metodologia de trabalho mista, foi utilizada durante o processo de pesquisa a metodologia de *Design Science*, de forma a construir um bom conhecimento para o desenvolvimento da solução e a garantir a relevância [16].

## 1.6 Estrutura do Documento

O presente documento encontra-se estruturado por capítulos. Por sua vez estes capítulos, apresentam subcapítulos, contudo estes não serão mencionados. Os capítulos abordam os seguintes assuntos:

- **Capítulo 1 - Introdução**: Este capítulo tem como propósito prefaciar o projeto desenvolvido. Aborda-se o enquadramento de forma resumida, o que levou ao nascimento da ideia, o que é pretendido do projeto, o incentivo, o ambiente de desenvolvimento e a metodologia utilizada;
- **Capítulo 2 - Contextualização:** Neste capítulo é apresentado o conhecimento necessário para a compreensão do projeto, como por exemplo API, monitorização e gestão**;**
- **Capítulo 3 - Estado da Arte**: Neste capítulo encontra-se a descrição do problema, a identificação dos componentes necessários para a solução e a apresentação de soluções que procuram resolver o problema identificado;
- **Capítulo 4 - Análise de Valor:** Neste capítulo é feita a avaliação da viabilidade do projeto;
- **Capítulo 5 – Design da solução:** Neste capítulo é descrita a solução do projeto numa perspetiva técnica;
- **Capítulo 6 – Implementação:** Neste capítulo é esmiuçado o processo de desenvolvimento;
- **Capítulo 7 – Experimentação e avaliação:** Neste capítulo é feita a avaliação do produto resultante;
- **Capítulo 8 – Conclusão:** No capítulo final são apresentadas de uma forma breve as informações principais do documento.

# Capítulo 2

# Contextualização

Com a apresentação do assunto do documento, torna-se necessário clarificar os conceitos específicos, relacionados com o tema. Neste capítulo irão ser explicados os conceitos de Interface de Programação de Aplicações (API), gestão e monitorização de API e *Software Development Kits* (SDK).

## 2.1 Application Programming Interface

Uma API especifica como sistemas e aplicações comunicam entre si, definindo um conjunto de regras para a ligação, o formato dos dados e os pontos de comunicação (*endpoints*)[7]. Esta permite que programas externos à empresa proprietária, interajam com programas internos da mesma sem acesso direto, como por exemplo a API do Google Maps possibilita interagir com o programa e biblioteca da Google sem aceder diretamente.

As interfaces de programação podem ser classificadas por diversas vertentes, nomeadamente a nível de acessibilidade, casos de uso e especificações.

### 2.1.1 Classificação de API - Acessibilidade

As API's a nível de acessibilidade, podem ser de três tipos: pública, de parceiros ou privada[17]. Começando pelas API públicas, tal como o nome sugere, qualquer pessoa consegue acedê-las. As de parceiros são disponibilizadas a parceiros de negócio da empresa proprietária, para integrações e consultas. E, por fim, as do tipo privado que são para uso interno da organização.

#### API Pública

Uma API pública, como a do *Google Maps*, é concebida com a finalidade de ser acessível sem qualquer tipo de restrições, não só internamente, como também por qualquer programador externo, que dela necessite. Esta acessibilidade permite a criação de novas ideias, que utilizam serviços da API, sem o investimento direto da organização. O êxito da publicação depende do alcance conseguido. Uma interface que seja bem concebida, documentada e intuitiva, ajuda a que a sua utilização e compreensão seja mais rápida e, consequentemente, a obter um maior alcance.

Contudo, as API públicas requerem trabalho acrescido, devido à sua necessidade de manutenção regular. Uma vez que é necessário manter a documentação atualizada [18].

A publicação da interface representa ainda um grande risco na segurança, pois torna os serviços acessíveis a qualquer intrusão e pode comprometer o produto associado. Daí ser necessário investir, neste cenário, em segurança acrescida [18].

**API Parceiros**

Relativamente às API de Parceiros, são disponibilizadas entre parceiros de negócio, de forma a permitem a integração de sistemas proprietários. As empresas ao permitirem aos parceiros acesso aos seus dados, pela API, conseguem obter rendimento extra com a venda do serviço.

**API Privada**

E por último, as API privadas são utilizadas internamente na sua organização. Empresas como a *Google*, *Facebook* ou *Twitter*, têm API pública, sites e aplicações móveis que vão buscar toda a sua informação a API privadas [18]. Estas interfaces são exclusivas para o uso dos seus colaboradores. Consequentemente não estão acessíveis para programadores externos, mas estão a conduzir o negócio da organização da API. Por vezes as interfaces privadas conseguem gerar mais benefícios do que as publicas. Devido a isso deve-se investir no desenvolvimento de uma boa API privada.

**Análise comparativa**

Na Tabela 1 é feita a comparação entre os três tipos de interfaces com base na acessibilidade.

Tabela 1 - Comparação entre tipos de API de acessibilidade [17]

| | Privada | Parceiros | Pública |
|---|---|---|---|
| **Acesso** | Uso interno na organização | Parceiros de negócio da empresa | Qualquer pessoa |
| **Finalidade** | Desenvolvimento de produtos na organização | Integrações com outros produtos | Incentivar a inovação |

Através da Tabela 1 é possível observar que a API privada é apenas para uso interno da organização e tem como finalidade o desenvolvimento de produtos na organização. No caso da API Parceiros o acesso é permitido a parceiros de negócio da empresa e tem como fim as integrações com outros produtos. E por fim, a API Pública é disponibilizada a qualquer pessoa e a sua finalidade é incentivar a inovação.

### 2.1.2 Classificação de API – Casos de Uso

Para além da classificação por acessibilidade, é possível classificar interfaces de programação pelos seus casos de uso. Relativamente a categorização existem 4 tipos: API de base de dados, API de sistemas operativos, API remota e API *web* [17]. Uma API de base de dados é quando a troca de informação é feita entre aplicação e sistema de gestão de base de dados. As API de sistemas operativos definem como as aplicações comunicam com sistemas operativos (SO) (por exemplo Windows e macOS). Quando a comunicação é feita entre aplicações que funcionam em máquinas diferentes, intitula-se API remota. E por fim a API *Web* acontece quando a troca de informação é feita entre sistemas à base da *web*.

#### API de Base de Dados

Começando pelas API de base de dados, permitem a ligação de aplicações com sistemas de gestão de base de dados. Os programadores, deste tipo de interfaces, utilizam linguagem especifica para consultar informações na base de dados(*queries*) e alteram tabelas de dados (*tables*) [17]. Um exemplo de uma API desta classificação é a API *Firebase*, pertencente à *Google*, permitindo aos seus programadores sincronizar e armazenar dados com facilidade e em tempo real [19].

#### API de Sistemas Operativos

Relativamente a interfaces de programação de sistemas operativos, estas definem como as aplicações utilizam os recursos e serviços dos SO [17]. Todos os sistemas operativos têm um conjunto de API. Um caso desta classificação é a API Windows, que possibilita o desenvolvimento de aplicações compatíveis com todas as versões do sistema operativo Windows [20].

#### API Remota

Uma API Remota, é uma interface que define como as aplicações, que estão em máquinas diferentes, interagem entre si. Ou seja, um sistema acede a recursos localizados noutra máquina. A conexão, uma vez que é feita entre equipamentos diferentes, realiza-se recorrendo a um protocolo de rede. Um exemplo desta categoria é a API *Java Database Connectivity* (JDBC), que permite aceder a qualquer fonte de dados, desde base de dados relacionais a simples arquivos [21].

#### API *Web*

Por último, API *web*, define a interação entre sistemas à base de *web* [17]. Estas interfaces fazem, principalmente, pedidos de aplicações *web* e respostas de servidos por HTTP. Um caso desta classificação é a API do Google Maps que possibilita adicionar um mapa com a localização de uma organização, num *site* ou aplicação.

**Análise comparativa**

Na Tabela 2 são comparadas as caraterísticas dos tipos de interfaces com base nos casos de uso.

Tabela 2 - Comparação entre tipos de API de casos de uso

| | API de Base de Dados | API de Sistemas Operativos | API Remota | API Web |
|---|---|---|---|---|
| **Ligação** | Aplicações e base de dados | Aplicações e sistemas operativos | Aplicações de máquinas diferentes | Sistemas à base da *web* |
| **Exemplos** | API Firebase | API Windows | API Java Database Connectivity | API Google Maps |

Segundo a Tabela 2 a API de Base de Dados efetua uma ligação com aplicações e base de dados e um exemplo desta é a API Firebase. No caso da API de Sistemas Operativos permite ligar aplicações e sistemas operativos, um exemplo é a API Windows. Em relação à API Remota a ligação é feita com aplicações de máquinas diferentes um caso é a API *Java Database Connectivity*. E por fim, a API Web efetua ligação entre sistemas à base da *web*, um exemplo é a API Google Maps.

### 2.1.3 Especificações

Nas classificações por caso de uso as API web, têm ainda padrões/especificações definidos para troca de dados entre serviços *web*.

Os termos serviços *web* e API são muitas vezes confundidos. Um serviço web é um recurso que permite a comunicação entre sistemas pela *internet*. API é uma interface que possibilita que sistemas e aplicações interajam, como explicado na secção 2.1. Todos os serviços *web* são API, uma vez que também os serviços acedem a informação de aplicações. Contudo nem todas as API são serviços *web*.

Existem duas grandes diferenças entre os conceitos. Uma delas é que os serviços apenas podem ser acedidos pela *internet*, ao contrário de uma interface que pode ser acedida com ou sem *internet*. E a outra diferença é que as interfaces de programação não estão restritas em termos de protocolo, ao contrário dos serviços que normalmente utilizam protocolo SOAP[22].

Uma API utilizar protocolos ou especificações diferentes, nomeadamente protocolo simples de acesso a objetos (SOAP), transferência representacional de estado (REST)[23], chamada de procedimento remoto (RPC)[24] ou *graphQL* [25].

**API SOAP**

Uma das especificações de uma API é utilizar o protocolo SOAP. Segundo a Microsoft, a empresa criadora, é um protocolo leve para a troca de informação estruturada num ambiente descentralizado e distribuído [26].
As mensagens são no formato XML e os pedidos são feitos pelo protocolo HTTP, SMTP ou XMPP [27]. O SOAP possui uma lógica de sucesso/nova tentativa integrada, para as mensagens, que assegura confiabilidade, mesmo quando é necessário utilizar intermediários SOAP [28]. A nível de base de dados, apresenta conformidade com as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade), que oferece a proteção da integridade da base de dados e reduz anomalias [29]. Em termos de segurança, suporta *WS-Security*, que é um protocolo que garante a integridade e a confidencialidade das mensagens [30], e SSL, que é um protocolo que oferece uma ligação, à *Internet,* segura e protegida [31]. Esta especificação é mais utilizada para sistemas que necessitam de ter grande segurança na troca de mensagens, sistemas de pagamentos, gestão de identidade e de gestão de relações com clientes (*Customer Relationship Management* ou CRM). Um exemplo de uma interface de programação com esta especificação é a API do Paypal.

**API REST**

Outra especificação que uma interface de programação pode ter é REST. Esta arquitetura obedece a um conjunto de seis restrições arquiteturais: cliente-servidor; interface uniforme; não tem estado; armazenável em cache; sistema por camadas e pedidos de código [32].

Qualquer API REST deve obedecer a uma arquitetura cliente-servidor, ou seja, existe um sistema servidor que tem a informação, e um sistema cliente que solicita essa informação. O sistema cliente pode aceder e manipular a informação do sistema servidor. Os dois são independentes um do outro [33]. O cliente não tem qualquer tipo de informação sobre a lógica de negócio e o servidor não tem informação sobre a Interface do utilizador (UI) [32].

Uma restrição importante que diferencia uma API REST de uma API não REST, é a interface ser uniforme. Esta sugere que o utilizador deve conseguir interagir, com um servidor, de forma uniforme independentemente do dispositivo ou aplicação. A uniformidade é conseguida através de um conjunto de indicações [34]. Sendo uma delas, o serviço ser identificado no pedido (por exemplo api/pagamentos). O cliente, através do serviço, conseguir alterar ou eliminar informações, desde que tenha permissões para tal. Cada pedido tem as informações necessárias para que a mensagem possa ser processada facilmente pelo servidor. E por fim, incluir *links* de outros pedidos que os clientes podem fazer, desta forma é fácil, para o cliente saber que outras ações podem fazer.

As interfaces de programação REST não têm estado. Ou seja, o servidor não guarda nenhuma informação sobre os pedidos do cliente. Cada pedido do cliente, ao servidor,

deve conter todas as informações necessárias para que possa ser entendido e não pode utilizar o armazenamento do servidor. O estado da sessão é, portanto, guardado no cliente. O cliente é responsável por armazenar e manipular todas as informações relacionadas ao estado da aplicação do seu lado [35]. A restrição de não ter estado, permite que o servidor tenha maior disponibilidade para guardar outras informações e separar as responsabilidades dos sistemas.

Armazenamento em cache é outra restrição de API REST. É possível, os clientes, armazenarem as respostas dos pedidos, desde que sejam definidas implicitamente ou explicitamente como armazenáveis em cache. A boa gestão deste atributo elimina certas interações entre o cliente e o servidor, melhorando escalabilidade e o desempenho da interface[34].

O REST permite ter um sistema em camadas, cada uma independente das outras. Na prática uma API pode fazer o *deploy* (enviar alterações) para o servidor A e autenticar-se no servidor B, por exemplo. O cliente não sabe a que camada está ligado e, esse facto, permite reforçar a segurança do sistema servidor [36].

Por último a restrição de pedidos de código, que é opcional em interfaces de programação REST. O comportamento normal é a resposta do servidor sejam informações. Contudo também é possível retornar excertos de código executável [34].

As API REST utilizam Identificador Uniforme de Recurso (URI), que é um conjunto de caracteres que identificam um recurso na *internet* [37], e protocolo HTTP. O uso do protocolo HTTP, confere simplicidade a API deste tipo [28]. As mensagens podem ser nos formatos texto simples, HTML, XML e JSON. O facto de utilizar o formato JSON torna compatível com *browsers* (como por exemplo Firefox, Google Chrome, Internet Explorer)[28]. Com os pedidos, o cliente, pretende obter informações ou efetuar alterações provenientes de recursos/serviços, que têm métodos: *get* (consultar), *put* (alterar), *post* (criar), *delete* (eliminar)[23]. Em termos de segurança, suporta SSL e HTTPS (versão segura do protocolo HTTP)[28].

As interfaces de programação com a especificação REST são a escolha mais comum para a criação de API publicas [17].

**API RPC**

No que diz respeito a API RCP, foi das primeiras especificações a ser criada, tornando-a das mais simples também [38]. O protocolo RPC tem como objetivo executar um pedaço de código noutro servidor. As interfaces RPC seguem uma arquitetura cliente-servidor. Os pedidos, desta especificação, utilizam URI e são feitos por HTTP ou *Advanced Message Queuing Protocol* (AMQP) [39]. As mensagens podem ser do formato XML ou JSON [40] [41]. Estas interfaces utilizam pedidos com método e argumentos e sua utilização é comparada à chamada de uma função em linguagens de programação [38]. Os pedidos utilizam métodos *get* ou *post*, podendo ser armazenados em cache [41].

### API GraphQL

E por último, uma interface de programação pode ter a especificação GraphQL. Esta é uma linguagem concebida para que sistemas cliente consultem bases de dados.

A nível de *backend* (camada onde os dados são armazenados e manipulados), é especificado como a API deve apresentar os dados ao cliente. Um servidor GraphQL disponibiliza, ao cliente, um modelo de dados(*schema*) que lhe pode ser pedido, dessa forma define-se como podem ser acedidos os dados. O cliente define que dados quer receber no formato JSON, segundo o *schema*. Um *schema* tem uma componente que é o tipo, que descreve a categoria do objeto e que campos tem (por exemplo *String*, *Character*). O *schema* define que consultas podem ser feitas, que tipos de dados podem ser acedidos e as relações entre esses tipos [42].

O GraphQL permite três operações: query (consultar dados), *mutation* (escrita de dados) e *subscription* (receber dados automaticamente em tempo real)[42].

### Análise comparativa

A Tabela 3 apresenta uma comparação entre as especificações abordadas, segundo os parâmetros de avaliação: estilo, formato dos dados, segurança, cache dos dados e conformidade com propriedades ACID.

Tabela 3 - Comparação entre especificações de API

| | API SOAP | API REST | API RPC | API GraphQL |
|---|---|---|---|---|
| **Estilo** | Protocolo | Arquitetura | Protocolo | Linguagem |
| **Formato dos dados** | XML | Texto simples, HTML, XML ou JSON | XML ou JSON | JSON |
| **Segurança** | Suporta WS-Security e SSL | Suporta SSL e HTTPS | --- | --- |
| **Cache de dados** | Não | Sim | Sim | Não |
| **Conformidade com as propriedades ACID** | Sim | Não | --- | --- |

Através da Tabela 3 é possível observar que a API SOAP utiliza o estilo protocolo, o formato dos dados é XML, em termos de segurança suporta WS-Security e SSL, não faz cache de dados e está em conformidade com as propriedades ACID. Em relação a API

REST usa o estilo arquitetura, o formato dos dados é variado (Texto simples, HTML, XML ou JSON), em termos de segurança suporta SSL e HTTPS, faz cache de dados e não está em conformidade com as propriedades ACID.A API RPC tem como estilo protocolo, o formato dos dados pode ser XML ou JSON e faz cache de dados. E por último a API GraphQL tem como estilo linguagem, o formato dos dados é JSON e não faz cache de dados.

### 2.1.4 Segurança

Concluída a descrição das especificações das interfaces de programação, torna-se necessário descrever os procedimentos a tomar a nível de segurança de uma API. Uma vez que as interfaces lidam com informações valiosas, é crucial que esta implemente fortes requisitos de segurança para proteger a informação disponibilizada.

A segurança de uma API é um conjunto de processos, sistemas e escolhas de design que fazem com que uma API web responda a pedidos, processe dados com segurança e funcione conforme o esperado. Existem múltiplas abordagens para garantir a segurança de uma interface de programação, nomeadamente, autenticação, autorização, segurança no transporte e escolhas de design.

A autenticação é o ato de confirmar a identidade de um cliente[43]. Para a confirmação existem vários métodos por onde escolher, sendo eles:

- **Autenticação por chave –** como o nome diz a autenticação é feita através de uma chave que é única para cada utilizador. Contudo os utilizadores conseguem gerar novas chaves, a partir de um ambiente seguro. Isso permite um acesso e acompanhamento mais detalhado nos ambientes de desenvolvimento, teste e produção. Em alguns casos é ainda possível os utilizadores cancelarem a chave (quando é roubada) e expirar. Este método é aconselhado quando se pretende aceder aos recursos de uma API ou trabalhar em plataformas que conseguem manter a chave segura. Alguns exemplos de interfaces que usam esta autenticação são a API IBM *Cloud* e API do Google *Cloud*.
- **Autenticação básica –** este tipo é o mais convencional. Trata-se de passar um utilizador e uma password no cabeçalho de um pedido. Deve ser utilizada com ligações HTTPS. A autenticação básica é indicada para interfaces que se misturam com autenticações de outras API, e desta forma é criado um sistema mais seguro. Um caso de uma interface que usa esta autenticação é a API Jira *Cloud*.
- **Autenticação por certificado do cliente –** neste caso são emitidos certificados *Public Key Infrastructure* (PKI) que são passados aos clientes para se autenticarem. Existem certificados de cliente e servidor, que são gerados com o mesmo conjunto de chaves e necessitam de estar sincronizados. Quando uma ligação é estabelecida, pela primeira vez, existem uma serie de validações, nomeadamente, se os certificados expiraram ou são inválidos ou têm informações incompatíveis. Falhando em alguma das validações a ligação não

irá ser feita. Este método de autenticação é aconselhado quando se tem um número limitado de clientes importantes e recursos suficientes para os assistir. Um exemplo de uma interface com esta autenticação é a API *Salesforce*.

- ***Open Auth* (OAuth) 2.0** – o OAuth concebido para ser um protocolo de autenticação/autorização. Utiliza SSL para garantir a segurança das transmissões e necessita de ligações HTTPS. Este tipo de autenticação é aconselhado em casos em que a interface tem um serviço principal em que os programadores estão a desenvolver aplicações. Exemplos de API com esta autenticação são o Github e o Facebook.

Em relação à autorização, é o processo de dar a alguém permissão para usufruir de um produto [44]. O processo oferece vantagens como não dar demasiadas permissões a utilizadores e que padrões não seguros persistam. Para este método são possíveis duas abordagens:

- ***Role Based Access Control* (RBAC) –** neste caso são criadas entidades, para serem associadas aos utilizadores, com certas permissões (por exemplo admin, autor)[45]. Um exemplo de uma interface com esta autorização é a API do *Wordpress*.
- ***Attribute Based Access Control* (ABAC) -** neste modelo, o importante é o contexto do pedido e o contexto do que está a ser pedido (por exemplo ip do cliente, sensibilidade do pedido ou hora do dia)[46].

No que diz respeito à segurança do transporte, este refere-se a formas de assegurar que a informação é enviada de um sistema para outro, sem que seja perdida ou que terceiros a consigam aceder. Este tipo de segurança é conseguido pela utilização de protocolos como o HTTPS e TLS/SSL. O protocolo HTTPS(versão mais segura do HTTP) garante que todos os pedidos são encriptados [47]. E o protocolo TLS/SSL estabelece um *link* encriptado entre um cliente e um servidor, permitindo uma ligação segura[48].

E por fim, as escolhas de *design*, na programação, podem ajudar a garantir a segurança de uma interface de programação. Uma das opções é garantir que os clientes têm acesso apenas ao que precisam e não mais. Incluir apenas as informações necessárias nas respostas, é outra opção. E por fim, minimizar a quantidade de dados que são solicitados e guardados [49].

### 2.1.5 Testes

A segurança na qualidade do funcionamento de uma interface de programação é garantida, também, por testes. Esta garantia passa por fazer pedidos à API e validar a resposta, através da exatidão dos dados, formato dos dados, códigos de *status* HTTP e códigos de erro.

Uma estratégia eficiente para desenvolver testes automáticos num projeto é seguindo a pirâmide de automatização de testes. O conceito sustenta que os testes devem ser

feitos na sequência *unit*, serviços e UI. A camada de testes *unit* representa os testes ao código que são realizados à medida que os programadores programam. A camada de serviços é quando é testada a API. E, por fim, a camada de testes UI, que são os que são efetuados após estar finalizada a parte do sistema que interage com o utilizador(*frontend*)[50]. Na Figura 1 encontra-se representado, de forma esquemática, o conceito de pirâmide de automatização de testes.

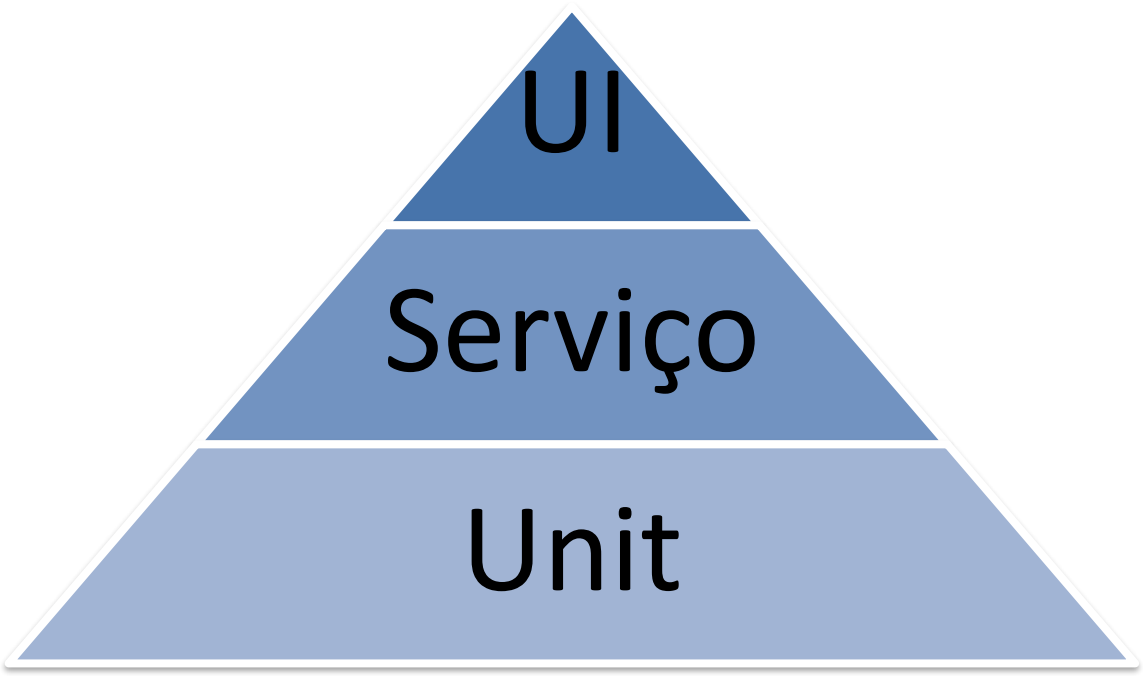


Figura 1 - Pirâmide de Testes automáticos

Sem os testes realizados à interface de programação (na camada de testes de serviço), podem acontecer erros no servidor ou na unidade. Uma vez que os testes UI não conseguem cobrir todos os aspetos relacionados com a parte do sistema onde são implementadas as regras de negócio (*back-end*)[50], como os serviços e caminhos de uma interface.

Os testes de API podem ser classificados em quatro categorias:

- **Funcionais –** são analisadas as várias funcionalidades, na perspetiva se retornam o resultado esperado e como reagem em sequência de erros. Com a sua implementação é conferida confiabilidade ao código. Um exemplo desta categoria são os testes positivos/negativos. Os testes negativos verificam respostas a pedidos feitos com informação inconsistente. E os positivos verificam a resposta em sequência de pedidos coerentes.
- **Integração e Confiabilidade –** neste caso, é analisada a ligação entre serviços internos e de terceiros. Em relação aos de integração é validada a comunicação entre interfaces de programação. E os de confiabilidade a ligação com dispositivos.

- **Desempenho –** neste contexto, é validada a funcionalidade e o desempenho, através da simulação ou criação de chamadas à interface. Alguns exemplos, são os testes de carga, onde é medido o desempenho da API sob grande carga, e os testes de *stress*, onde é aumentada a quantidade de utilizadores gradualmente e analisado o comportamento da interface.
- **Segurança –** são analisadas vulnerabilidades na interface. Alguns exemplos são os testes de segurança, no qual são validados os requisitos de segurança (como autenticação e permissões), e testes de penetração, em que são atacadas funções, recursos e processos.

### 2.1.6 Prós e contras de API

Tendo em conta as classificações, a segurança e os testes, as API's têm prós e contras com a sua implementação. Entre os prós, destacam-se:

- Ligação entre aplicações ou sistemas distintos, com linguagens de programação diferentes
- Proteção e segurança dos dados, obtida pela implementação de mecanismos durante o processo de desenvolvimento, como autenticação e autorização [51]
- Publicar de forma eficiente novos conteúdos em diferentes canais automaticamente [52]
- Facilidade de se adaptar a alterações do sistema, como por exemplo a migração de dados, processo habitualmente dispendioso

Relativamente aos contras, em termos de segurança, representa mais um ponto de ataque e, se atacado, deixa vulnerável a estrutura do sistema [53]. Também a manutenção frequente, fornecer e atualizar documentação.

## 2.2 Gestão da API

Com o crescimento da utilização de uma API, torna-se mais difícil detetar e solucionar problemas e, consequentemente, aumenta a necessidade de gerir a mesma. A gestão, representa um conjunto de processos de análise, controlo e distribuição.

As ferramentas de gestão, da API, apresentam em comum as funcionalidades: portal destinado a *developers*, *gateway*, *deployment*, analises de desempenho e gestão dos processos.

O portal de *developers* permite testar e utilizar a API [9]. O *gateway* tem como objetivo controlar o tráfego da API. O *deployment* é a publicação e disponibilização da API para o publico. Com as análises é possível inferir a média do tempo de resposta da API, as tendências do tráfego, o máximo e o mínimo tempo de resposta, média da troca de

dados e a taxa de erro. E por fim a gestão de processos, pretende apresentar o estado dos processos e geri-los.

Segundo um estudo realizado pelo Zion Market Research, em 2017 a gestão de API representa um mercado no valor de 70 milhões de dólares e é esperado que, este cresça em 33,4% entre 2017 e 2022 [10].

A gestão permite obter confiabilidade, qualidade e rapidez da API [9]. As análises e outras funcionalidades da gestão conferem a redução do tempo de inatividade da API, consequentemente aumenta a confiabilidade e maior rapidez de resposta. A consistência em conjunto com as expetativas dos *developers* permite determinar a qualidade de uma API e com a gestão a qualidade tende a aumentar consideravelmente.

## 2.3 Monitorização da API

Diminuir o tempo de inatividade da API, e consequentemente aumentar a confiabilidade na mesma, é o objetivo da monitorização. Este objetivo é atingido através de uma avaliação de chamadas, feitas à tecnologia em questão, em caso da não concordância com os padrões de aceitação é apresentada notificação de erro [8]. Esta função pode ser executada de duas formas, através da monitorização básica ou de múltiplas fases. A monitorização básica é feita através de uma única chamada e as análises são feitas com base na mesma. A monitorização de múltiplas fases faz uma análise em torno de todas as chamadas possíveis [54].

## 2.4 Software Development Kits

Um *Software Development Kits* (SDK) serve como uma ferramenta facilitadora do desenvolvimento de aplicações. É constituído por métodos e classes gerados a partir das especificações de uma API, normalmente estão acompanhados da referente documentação para a utilização dos mesmos [55]. É uma forma de outros *developers* utilizarem funcionalidades especificas de outros serviços, sem terem de fazer desde o início, no desenvolvimento do seu produto.

# Capítulo 3

# Estado da Arte

De forma a aprofundar a compreensão da área do problema torna-se necessário fazer um estudo do estado da arte, aplicando os conhecimentos obtidos no capítulo 2. O estado de arte é um estudo das ferramentas e tecnologias, existentes no mercado, académicas ou comerciais com o objetivo de suportar as decisões a serem tomadas no desenvolvimento da aplicação. Neste capítulo é apresentado o estudo, por meio da análise da estrutura tecnológica do produto E-goi, da descrição da solução para o problema da dissertação e da avaliação de possíveis soluções no mercado.

O produto E-goi exibe uma variedade de opções, que têm como propósito auxiliar as empresas no processo de comunicação com os seus consumidores. Algumas das possibilidades, do produto, são a criação de listas de contactos, de campanhas e de formulários, a elaboração de análises do comportamento dos consumidores e do sucesso das campanhas, edição de *newsletters*, envio de emails com as campanhas, envio de sms com links e promoções, criação e envio de notificações *web push* e *push mobile* [56]. Para além destas opções existem ainda integrações (ligação do E-goi com outras plataformas e vice-versa), com WordPress, Shopify, Facebook, Vtex, Google, entre outras [57]. A integração com o Shopify permite sincronizar contactos, analisar o comportamento dos clientes (da empresa), sincronização de campanhas de sms com a loja Shopify, automatização de marketing multicanal e comunicação personalizada da empresa com os clientes [58]. Toda esta atividade passa pela API pública da E-goi.

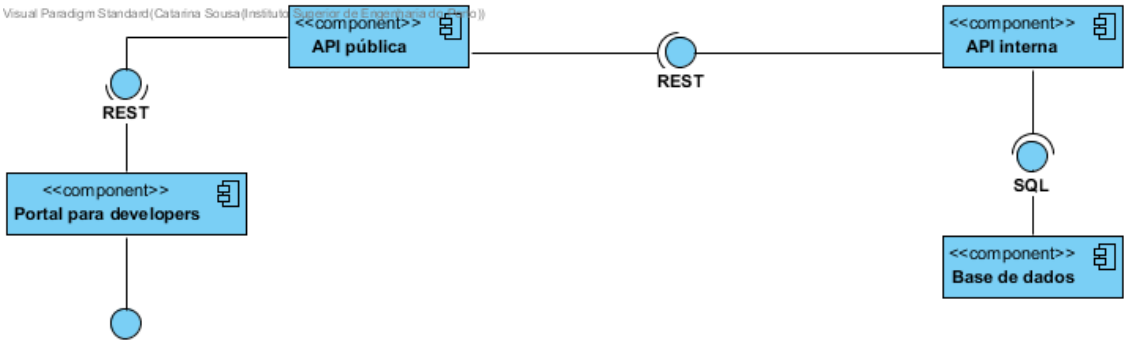


Figura 2 - Estrutura tecnológica da E-goi

A Figura 2 apresenta a arquitetura da estrutura tecnológica da empresa. A API interna comunica com a base de dados por SQL. A API pública obtém a informação através de pedidos REST à API interna. Existe ainda um portal para *developers* que recebe toda a sua informação por pedidos REST à API publica.

O portal para *developers* fornece toda a informação necessária sobre como a utilizar a API publica da E-goi. Este expõe a documentação sobre a API pública da E-goi, em que são apresentados todos os pedidos possíveis e a sua sintaxe. A informação fornecida, pelo portal, tem como objetivo facilitar integrações de outras plataformas com o produto E-goi. Todos os pedidos são guardados no MongoDb [59], plataforma de armazenamento de dados.

O Jenkins é uma ferramenta de *Continuos Integration*, ou seja o processo de integração de código num projeto é automatizado [60]. A E-goi utiliza o Jenkins para todo o processo de *deploy*.

## 3.1 Critérios da Solução

Tendo em conta toda a estrutura tecnológica da empresa E-goi existe um claro problema. Todos os dias são criadas centenas de novas contas, no produto E-goi, e enviadas milhares de campanhas, por diferentes canais de comunicação. Para além disso, até ao momento, existem centenas de integrações com o produto. Uma vez que todo este tráfego passa pela API (como foi mencionado na secção 3) e a plataforma está a crescer, a resolução e deteção de problemas está a tornar-se cada vez mais um desafio que exige uma resolução com alguma brevidade.

O problema encontrado, pode levar ao aumento do tempo de resposta e a períodos de inatividade nos serviços. O que provoca o descontentamento dos clientes, uma vez que, com o aumento do tempo de resposta, irão demorar mais a executar as suas tarefas e, com a inatividade, não irão conseguir utilizar os serviços. Por causa destas circunstâncias a empresa pode vir a ter um grande prejuízo, irá perder clientes e gastar recursos. Segundo um estudo feito sobre o impacto da inatividade, as empresas que estão dependentes da tecnologia, como é o caso da E-goi, podem ter um prejuízo elevado.

Uma solução, para o presente dilema, seria utilizar uma ferramenta de gestão e monitorização de API, dado que a gestão possibilita controlar processos (*deploy*, documentação, entre outros) [10], e a monitorização, permite saber o estado da tecnologia [8]. Para a resolução do problema é imperativo que a ferramenta tenha as seguintes funcionalidades:

- Elaborar análises feitas por pedido. Desta forma é possível saber quais os pedidos mais problemáticos e resolver os erros encontrados.
- Permitir a monitorização e a gestão de várias API. No futuro será necessário que a API interna também possa ser analisada, apesar de não ser uma prioridade no momento.

A empresa também estipulou os seguintes requisitos:

- Ser uma ferramenta gratuita.

- Ter geração de *Software Development Kits* (SDK) com a respetiva documentação.
- Elaborar análises por cliente. Deste modo conseguem saber os clientes que mais utilizam a plataforma e os que estão a criar transtorno.
- Implementar monitorização básica.

As ferramentas que permitem, de alguma forma, responder aos requisitos identificados e que seriam concorrentes a resolver o problema em causa são: OpenAPI Generator, APIMatic e Assertible.

### 3.1.1 OpenAPI Generator

Em 2012, a empresa SmartBear[61], criadora de soluções que compilam, testam e monitorizam *software*, sentiu que a sua ferramenta de desenvolvimento de API, Swagger Codegen, estava a divergir do seu propósito [62]. E assim foi criada, a partir do Swagger Codegen, a ferramenta OpenAPI Generator, *open-source*, gratuita e escrita em Java. Tem como objetivo gerar SDK a partir de uma API. Pode ser utilizada, nos sistemas operativos, Windows, macOs e Linux. Oferece gerações em 114 linguagens de programação, entre elas java, php e C# e permite ainda personalizar as gerações de código esqueleto [63]. Contudo as gerações não incluem a implementação de testes unitários, apenas são gerados os cabeçalhos.

Tabela 4 - Customizações de java [64]

| Opção | Descrição | Valor por omissão |
|---|---|---|
| **apiPackage** | *package* para classes geradas da API | org.openapitools.client.api |
| **artifactDescription** | Descrição do artefacto no pom.xml gerado | OpenAPI Java |
| **artifactId** | Id do artefacto no ficheiro pom.xml. Também se torna parte do nome do ficheiro da biblioteca gerada | openapi-java-client |
| **artifactVersion** | Versão do artefacto no ficheiro pom.xml. Também se torna parte do nome do ficheiro da biblioteca gerada | 1.0.0 |
| **licenseName** | O nome da licença | Sem licença |

Na Tabela 5 é possível observar um exemplo das possibilidades de personalizar a geração de Java, entre elas alterar o nome da pasta gerada e a versão. Esta ferramenta tem uma comunidade ativa e disponibiliza atualizações frequentes.

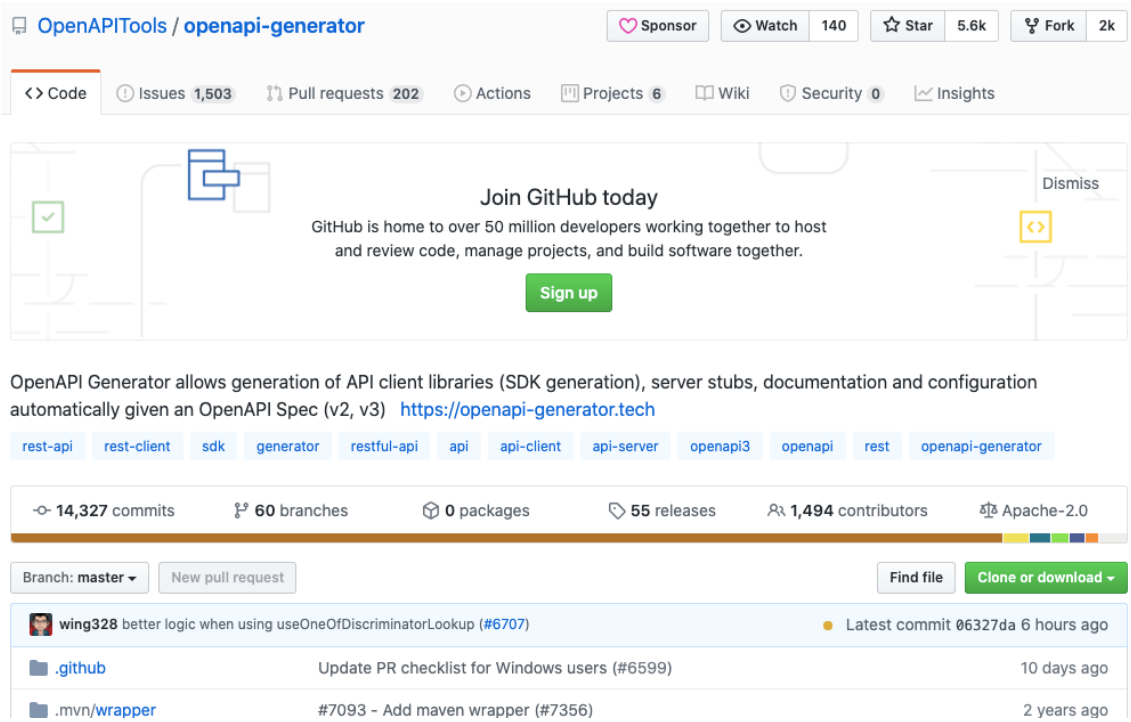


Figura 3 - Página do GitHub do *OpenAPI Generator*[62]

Como é possível observar na Figura 3, atualmente, a comunidade conta com 1494 colaboradores, 1503 problemas com a utilização (*issues*) e 60 versões do código que representam distintos desenvolvimentos (ramos).

### 3.1.2 APIMatic

No ano de 2014, um grupo de estudantes apercebeu-se de um problema na indústria tecnológica relacionado com os SDK [65]. Notaram que o desenvolvimento de SDK, com a respetiva documentação, era difícil e dispendioso, contudo permitia às empresas alcançar maior número de consumidores. Deste problema nasceu o APIMatic [66], uma plataforma focada no desenvolvimento de API, desde a sua definição à geração de SDK.

Quando o utilizador faz o seu primeiro login, na plataforma, é lhe pedido para importar ou criar uma API, de seguida é validada e é gerado um portal para a mesma (Figura 4).

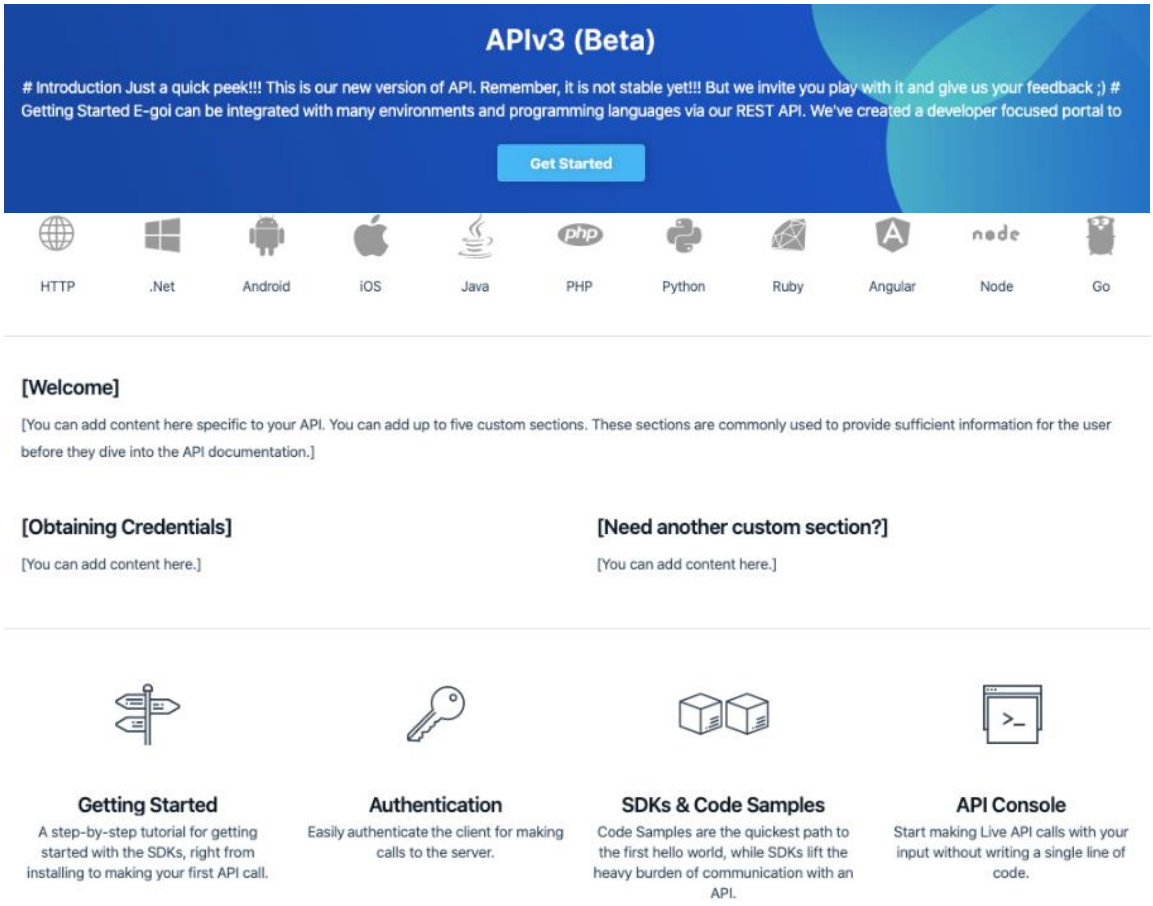


Figura 4 - Exemplo de um portal gerado

O portal apresentado na Figura 4 é editável e disponibiliza 11 SDK's de diferentes linguagens de programação, como Go e Java. Cada SDK tem a respetiva documentação, que explica como os utilizar com os IDE recomendados, e código para testes unitários pré-feito.

Para além do portal, a ferramenta permite gerar SDK individualmente, nas mesmas 11 linguagens, fazer o download, publicar e fazer o *deploy* para o Github diretamente. Antes da geração, é sempre validada a API e apresentados os erros (Figura 5).

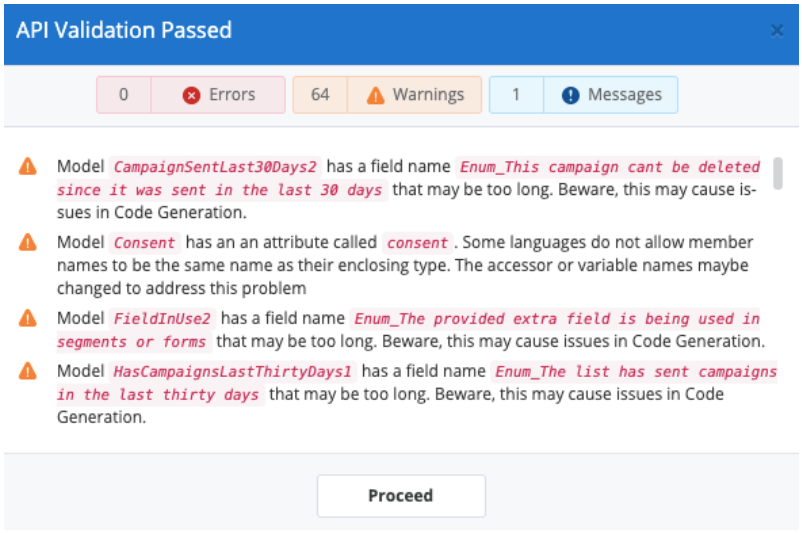


Figura 5 - Log de erros da validação da API

As opções da ferramenta não são apenas o portal e os SDK. É possível editar uma API, exportar, ver os logs e as chaves de integração, substituir a versão e copiar (*fork*). A plataforma oferece suporte para mais do que uma API e todas as ações, abordadas anteriormente, podem ser efetuadas para cada uma delas, como é possível ver na Figura 6.

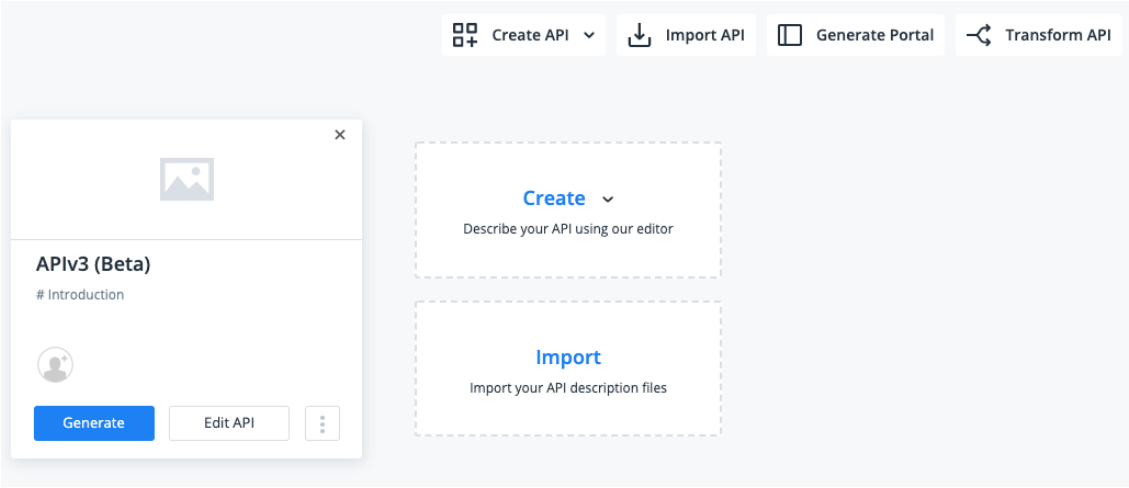


Figura 6 - Página inicial do APIMatic

Segundo o site *Featured Customers*, o APIMatic tem uma avaliação, constituída pela opinião de 560 utilizadores, de 4,7 em 5 [67]. Sendo assim uma ferramenta popular e de eleição relativamente à gestão de API. Contudo a sua utilização é limitada ao plano da conta, os preços podem variar entre 15 a 115 dólares, por mês, e o plano mais caro ser ilimitado [68]. Apesar de ser uma ferramenta paga, houve necessidade de a necessidade de incluir nesta análise uma vez que é uma referência em gestão de API e é simples de utilizar.

### 3.1.3 Assertible

Criada por Christopher e Cody Reichert, em 2015, a ferramenta Assertible [69] foi concebida com o intuito de garantir a disponibilidade de API. Esta plataforma proporciona uma variedade de ações de monitorização, incluídas na documentação, e ainda assistência por *chat*.

De forma a começar a usufruir das ações da plataforma, existe necessidade de definir uma API para utilizar. Esta pode ser importada por URL, especificação de Swagger/OpenAPI, coleção do Postman (conjunto de pedidos [70]) ou por comando para conectar com um URL (cURL) [71] (Figura 7).

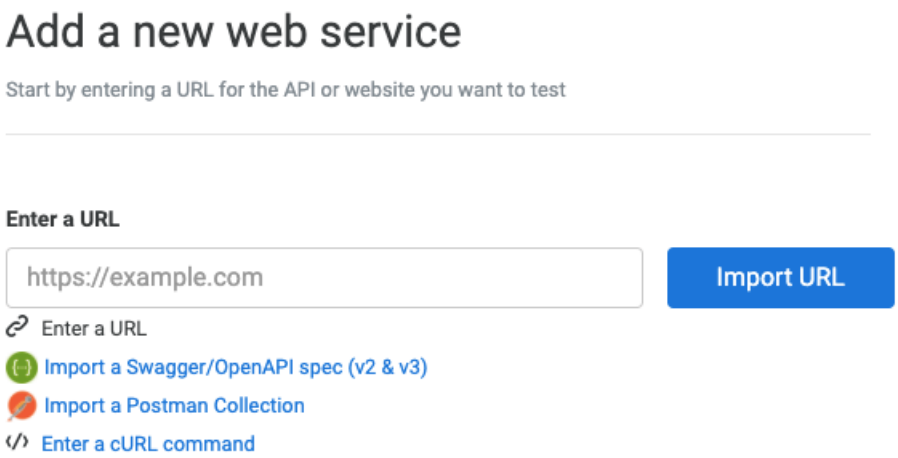


Figura 7 - Página para adicionar uma API

Em seguida é necessário definir como irá ser feita a monitorização. Neste caso é possível fazer a mais do que um pedido (monitorização de múltiplas fases) ou apenas a um pedido em específico (monitorização básica). Para isso são definidos os pedidos a testar e os parâmetros de aceitação de cada um, também é possível importar e sincronizar testes do *postman* ou especificação Swagger/OpenAPI. A Figura 8 apresenta como são criados testes na ferramenta.

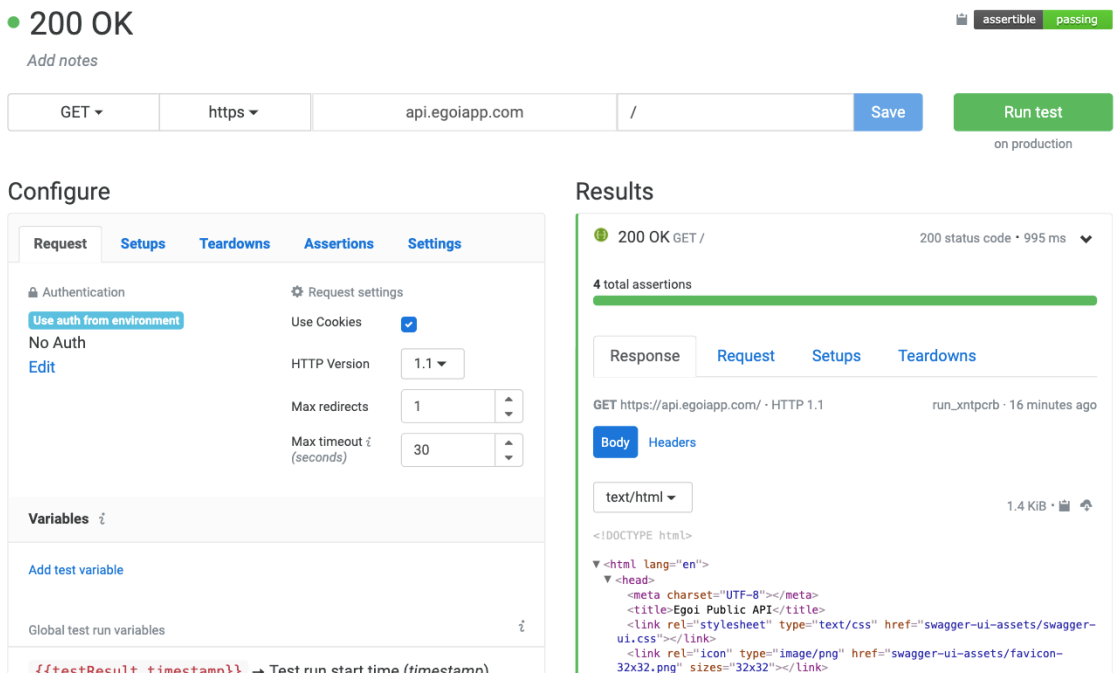


Figura 8 - Interface de criação de testes

O processo de criação é bastante simples e muito completo, é possível adicionar variáveis, corpo e *header* do pedido, entre outros. Após a execução é apresentada a resposta do teste e quanto tempo demorou. Reunindo esta informação, são geradas análises da performance e dos testes com os seguintes dados: tempo de resposta, testes que falharam e o momento em que falharam (Figura 9).

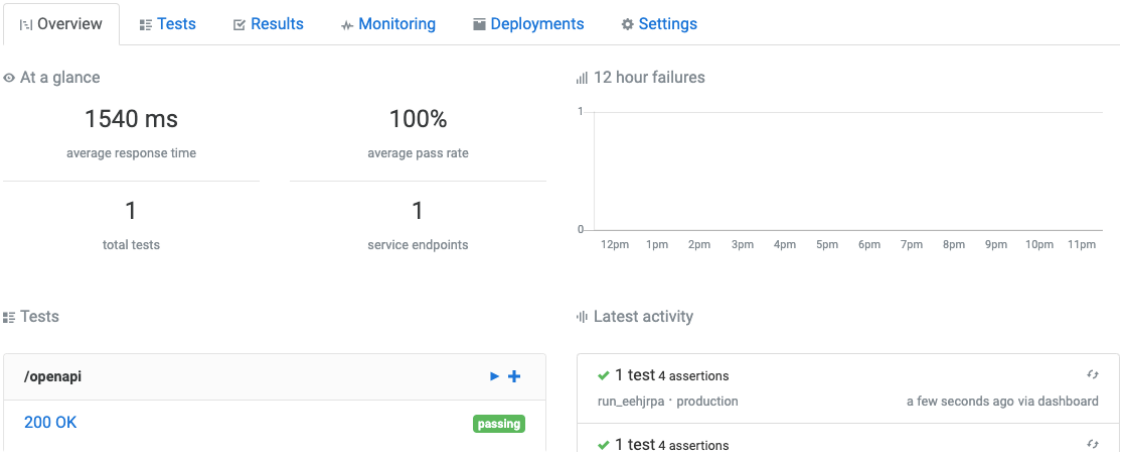


Figura 9 - Análises Assertible

É possível ainda agendar as monitorizações receber alertas de erro por email, slack, entre outros.

Para além das funcionalidades abordadas anteriormente, o Assertible permite conectar com um repositório no GitHub e oferece um comando em que, quando utilizado, a plataforma é notificada e corre os testes (normalmente é utilizado em eventos de *deploy*). Todas estas opções podem ser realizadas para múltiplas API, uma vez que a

plataforma disponibiliza apoio para mais do que uma. O uso da plataforma está condicionado aos planos de pagamento, em que podem oscilar entre 0 e 100 dólares [72].

## 3.2 Análise comparativa

A tabela seguinte oferece de uma forma visual e sumaria os cumprimentos e incumprimentos dos requisitos necessários.

Tabela 5 - Tabela comparativa das soluções segundo os requisitos

| Critérios | OpenAPI Generator | APIMatic | Assertible |
|---|---|---|---|
| **Geração de SDK com documentação** | Sim | Sim, automática | Não |
| **Análises relativas a clientes** | Não | Não | Não |
| **Análises por pedido** | Não | Não | Não |
| **Suporta várias API** | Sim | Sim | Sim |
| **Monitorização básica** | Não | Não | Sim |
| **Ferramenta gratuita** | Sim | Não | Tem plano gratuito, mas é limitado |

Segundo a Tabela 1, as ferramentas que tem mais pontos em concordância com os necessários é o *Assertible* e o *OpenApi Generator*. O *OpenApi Generator* está de acordo com os requisitos: geração automática com documentação associada, suporta várias API e é uma ferramenta gratuita. O *Assertible* satisfaz os requisitos de suportar várias API, tem monitorização por múltiplas fases e tem um plano gratuito, contudo é limitado. O *APIMatic* está de acordo com o critério de geração de SDK, a geração é automática e

suporta várias API. Porém é de realçar que nenhuma das ferramentas satisfaz todos os requisitos necessários para resolver o problema.

Relativamente ao problema em causa, as ferramentas abordadas satisfazem alguns requisitos, mas nenhuma satisfaz todos. Devido a esse facto as opções não são soluções para o problema. Contudo todas as ferramentas mencionadas são referências na área do problema.

# Capítulo 4

# Análise de Valor

Todos os dias as empresas combatem os desafios constantes de num mercado competitivo, como é o que temos atualmente. Tendo estas que saber gerir com eficiência o seu capital e investir com precaução. Tornando-se assim uma análise de valor essencial para avaliar a viabilidade da demanda e a capacidade de reduzir custos de um investimento [73].

## 4.1 Processo de Inovação

A palavra inovação, refere-se a uma ideia, método ou objeto que não se enquadra nos padrões anteriores. O processo de inovação é a descoberta, criação e desenvolvimento de ideias, juntamente com a sua utilização para lucrar e aumentar a eficiência. Este procedimento divide-se em três fases: *Fuzzy Front End* (FFE), *New Development Product* (NDP) e comercialização. O processo requer que as fases sejam executadas segundo a sua ordem: FFE, NDP e comercialização, pois estão dependentes das conclusões das anteriores.

A etapa de *Fuzzy Front End*, representa um papel importante para o sucesso do processo de inovação. Esta etapa é o ponto de partida para o desenvolvimento. São criadas, analisadas e validadas oportunidades, com o intuito de gerar uma ideia que irá ser implementada.

Segue-se a fase de *New Development Product*, onde é executada a ideia resultante da fase anterior. É concebido o produto, analisado o mercado, testado perante uma amostra de pessoas e alterado em função dos resultados.

Por último, a comercialização representa a etapa em que o produto é recebido no mercado. Nesta fase, o produto, já foi concebido, validado e testado, sendo que, para finalizar, será produzido e vendido.

Das três fases abordadas, apenas o conceito FFE suscita controvérsia relativamente à sua definição. Um estudo realizado por colaboradores de oito empresas diferentes, provou que o conceito de FFE é ambíguo e incomparável entre empresas. E contrariamente o termo *New Concept Development* provou ser claro e comum entre as empresas [74].

## 4.2 New Concept Development

O conceito de *New Concept Development* (NCD), surgiu como uma forma de se obter uma linguagem comum relativamente ao *Front End*. Este termo encontra-se dividido em três partes: motor, conceitos-chave e fatores de influência (Figura 10).

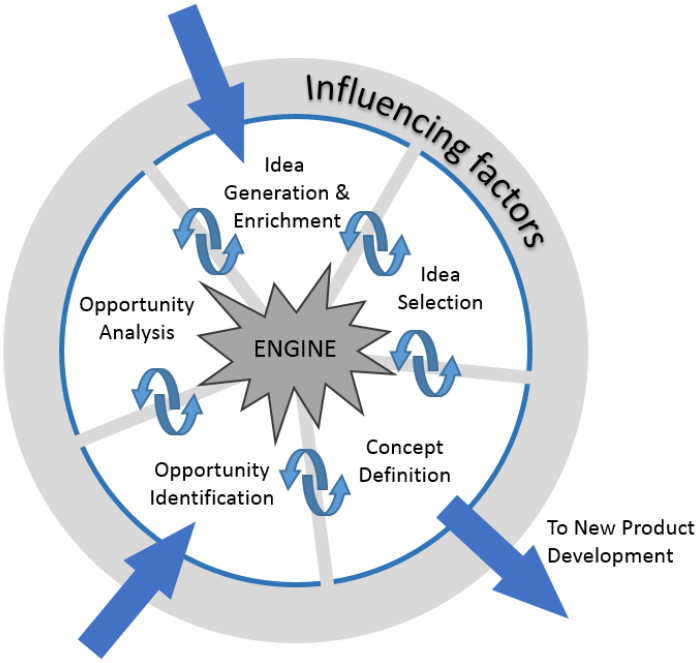


Figura 10 - Conceito de NCD e as suas parte[75]

A figura representa a constituição do termo NCD. O motor representa a liderança, a cultura e a estratégia de negócios da empresa que impulsiona os conceitos-chave. Os conceitos-chave são: identificação da oportunidade, análise da oportunidade, geração de ideias, seleção da ideia, definição do conceito. Os fatores de influência afetam as decisões tomadas no motor e nos conceitos. Estes são as capacidades da empresa, fatores externos (leis, canais de distribuição, clientes, concorrência, entre outros) e a ciência envolvente [76].

O processo descrito é considerado um fator determinante para o sucesso do produto resultante [77].

### 4.2.1 Identificação da Oportunidade

As oportunidades de inovar podem aparecer de várias formas, inesperadamente, por necessidade, incongruência entre outras [78, p. 7]. Estas chances são uma forma de resolver um problema, obter lucros e aumentar a competitividade da empresa no mercado.

O problema identificado na secção 1.3 do Capítulo 1 representa uma oportunidade para melhorar a performance da API interna da E-goi, através de uma ferramenta de gestão e monitorização da mesma.

Contudo, as oportunidades de inovação podem resultar tanto em algo positivo para a empresa como negativo. De forma a averiguar que tipo de resultado irá proporcionar esta oportunidade, será feita uma análise da oportunidade.

### 4.2.2 Análise de Oportunidade

Com a análise de oportunidade pretende-se traduzir a oportunidade encontrada numa oportunidade de negócio para a empresa. Para fazer essa tradução irá ser feita uma análise SWOT.

A análise SWOT é uma ferramenta que apresenta, de uma forma esquemática, as forças, fraquezas, oportunidades e ameaças, permitindo assim fazer uma avaliação estratégica de um produto ou de uma empresa. As forças apresentam os pontos positivos, relativos à empresa perante a concorrência. As fraquezas demostram os pontos negativos, na mesma perspetiva das forças. As oportunidades representam os aspetos externos positivos que podem aumentar a competitividade da empresa. E por fim, as ameaças, são os pontos negativos que podem colocar em risco a posição da empresa no mercado [79].

Tabela 6 - Análise SWOT da oportunidade

| **Forças** | **Fraquezas** |
|---|---|
| Manutenção, gestão de API juntamente com geração de SDK | Custos de produção |
| Análises dos pedidos por clientes | |
| Análises por pedido | |
| Simplicidade da ferramenta | |
| **Oportunidades** | **Ameaças** |
| Melhoramento do desempenho da plataforma E-goi | A concorrência especifica em gestão, monitorização ou SDK |
| Acompanhamento do crescimento da plataforma | Novos concorrentes |

Com a análise da Tabela 7 pretende-se avaliar a viabilidade da oportunidade de melhorar a performance da API.

No que se refere às forças, foram encontradas quatro, com necessidade de serem referidas. A primeira, é o facto de haver a possibilidade de fazer a manutenção, gestão de API e a geração de SDK. Após uma pesquisa de mercado concluiu-se que a maioria das ferramentas capazes de solucionar o problema não reúnem as três funcionalidades. Normalmente as ferramentas apresentam uma ou duas das opções.

Outra força são as análises dos pedidos por cliente. Esta análise tem como objetivo saber quais os clientes que mais utilizam a plataforma e as funcionalidades mais populares. Na pesquisa de mercado não foi encontrada nenhuma ferramenta com esta funcionalidade, atribuindo assim uma caraterística que oferece distinção perante o mercado.

Plataformas concorrentes fazem testes aos pedidos de uma API. No entanto o produto em causa oferece análises tendo em conta os pedidos que foram feitos à API. Distinguindo-se, novamente, perante as atuais soluções e atribuindo uma vantagem à empresa.

Por última força reconhece-se que o produto em questão se destaca pela simplicidade. No mercado de plataformas de gestão e monitorização, poucas são fáceis de utilizar, as que não obedecem à regra são o APIMatic e o Assertible.

Relativamente às fraquezas existe os custos de produção. O desenvolvimento de uma nova ferramenta de ajuda à melhoria da performance de uma API tem custos associados, uma vez que é necessário fazer uma análise dos critérios a avaliar, fazer uma ferramenta que apresente um relatório da performance e como melhorar, desenvolver e testar. Esses custos podem não justificar o lucro que irá ser gerado.

As forças e as fraquezas apresentam caraterísticas internas relativas à empresa e ao mercado em que está inserida, que por sua vez são controláveis e possíveis de alterar e melhorar. Pelo contrário, as oportunidades e as ameaças representam atributos externos à empresa, e consequentemente impossíveis de controlar.

As oportunidades encontradas foram duas. Um cenário possível de acontecer ao implementar a ferramenta de ajuda à melhoria do desempenho da API, é a melhoria da performance da plataforma da E-goi. Também é possível que haja melhor compatibilidade com o crescimento da plataforma e aumento da demanda.

Por outro lado, as ameaças também existem. A concorrência é sempre um fator que tem grande impacto nos negócios de uma empresa e nos seus investimentos. Tanto novos concorrentes que podem surgir com uma ferramenta com as mesmas caraterísticas, bem como ferramentas especializadas num dos processos (gestão, monitorização de API ou geração de SDK), representam ameaças para o bom desempenho do investimento.

Tendo em conta todos os pontos apresentados da análise SWOT é possível concluir que a oportunidade é viável e irá trazer benefícios para a empresa. Alguns deles são, a nível externo, oferece a melhoria da performance da plataforma E-goi e, a nível interno, tem as funcionalidades que marcam a diferença no mercado. De seguida irá ser feita a geração de ideias para o desenvolvimento da solução.

### 4.2.3 Geração de Ideias

O terceiro passo do *New Concept Development* é a geração de ideias, uma fase em que são feitas sugestões de possíveis soluções para que sejam escolhidas as melhores entre elas. Resultando um conjunto de ideias que poderão ser executadas.

Para o caso em questão foi feita uma pesquisa de possíveis soluções que cumprissem os critérios da oportunidade, da análise SWOT. Após uma discussão interna concluiu-se que existem três hipóteses de solução:

- Implementar uma nova ferramenta de gestão e monitorização de API, juntamente com a criação de um novo gerador de SDK
- Implementar uma ferramenta de gestão e monitorização de API, que utiliza um gerador *open-source* de SDK e usa as tecnologias da empresa
- Implementar uma ferramenta de gestão e monitorização de API, que utiliza um gerador *open-source* de SDK e não usa as tecnologias da empresa

A empresa utiliza tecnologias que poderão ser uteis à monitorização e gestão da API, nomeadamente os registos dos pedidos efetuados à API publica são todos guardados numa base de dados no MongoDB e os processos de *deploy* tem associado uma *pipeline* no *Jenkins* (secção 3.1 capítulo 3). A utilização desta informação permite a integração com recursos já existentes na empresa.

Com as hipóteses de solução enumeradas, selecionadas e validadas com base nos critérios de resolução do problema, o próximo passo é a seleção das ideias.

### 4.2.4 Seleção de Ideias

Com a identificação da oportunidade, a análise de oportunidade e a geração de ideias concluídas e validadas, segue-se a seleção de ideias. Com esta análise pretende-se escolher quais as ideias que irão trazer mais valor para o consumidor e o negócio. O modelo utilizado para fazer a seleção das ideias foi o *Analytic Hierarchy Process* (AHP).

#### Divisão Hierárquica

O *Analytic Hierarchy Process* (AHP) foi criado em 1970, por Thomas L. Saaty. É um modelo utilizado para a tomada de decisões que apresentam muitas variáveis e cenários complexos. Para utilizar o modelo é necessário definir um objetivo, critérios e alternativas [80].

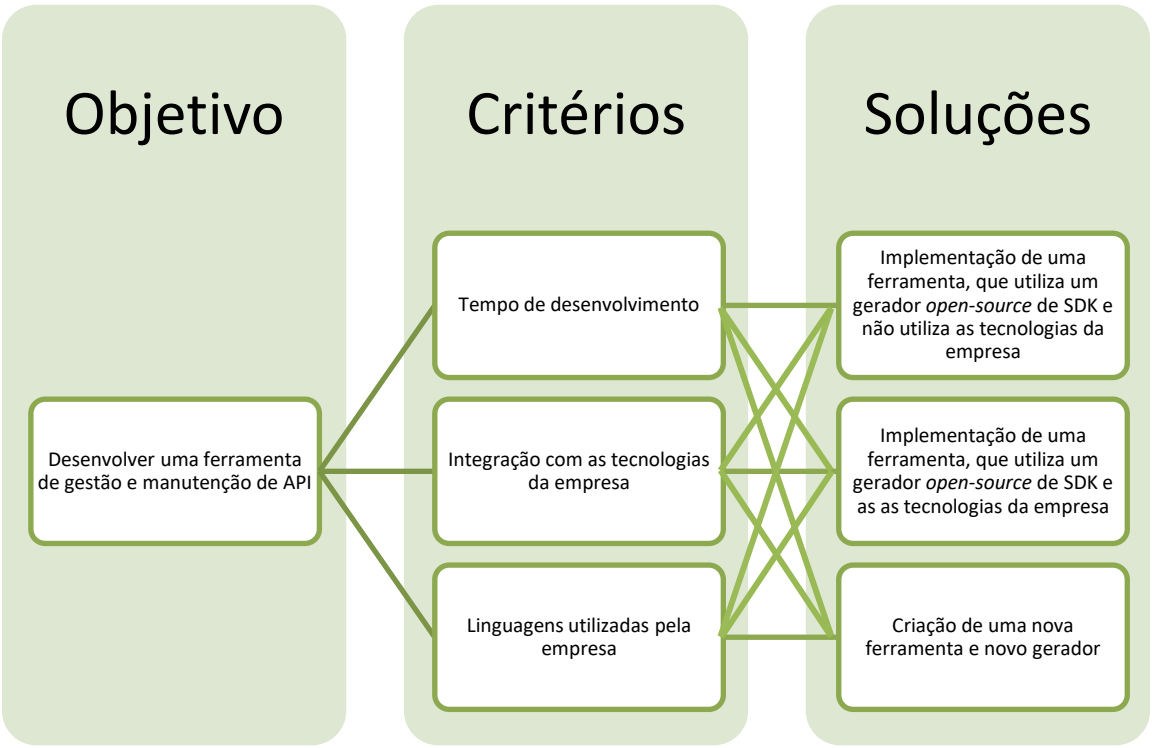


Figura 11 – Esquematização Hierárquica de AHP

No âmbito do problema em questão o objetivo definido foi desenvolver uma ferramenta de gestão e manutenção de API. Os critérios são o tempo de desenvolvimento, a facilidade de integração com as tecnologias da empresa e as linguagens utilizadas. Relativamente ao tempo de desenvolvimento considera-se que quanto menor for, melhor é para a empresa. A integração das tecnologias considera que com quantas mais tecnologias da empresa forem utilizadas, mais vantajoso é. E por fim, as linguagens utilizadas pela empresa tem em conta as linguagens aplicadas no desenvolvimento da ferramenta e se são usadas pela empresa. As soluções são as que foram definidas na secção 4.2.3.

Tendo em conta todas as variáveis e cenários do problema, é necessário definir as prioridades da decisão.

**Definição de Prioridades dos Critérios**

O primeiro passo para a definição de prioridades é a comparação dos critérios. A comparação entre dois elementos, pelo AHP, pode ser feita de múltiplas formas, contudo a mais comum é pela escala de importância (escala de Saaty). Os valores, da escala, estão compreendidos entre 1 e 9, e representam a importância de um elemento relativamente a outro.

| Scale | Numerical Rating |
|---|---|
| **Extremely Preferred** | **9** |
| Very strong to extremely | 8 |
| **Very strongly preferred** | **7** |
| Strongly to very strongly | 6 |
| **Strongly preferred** | **5** |
| Moderately to strongly | 4 |
| **Moderately preferred** | **3** |
| Equally to moderately | 2 |
| **Equally preferred** | **1** |

Figura 12 - Escala de Saaty [80]

Como é possível observar na Figura 12, o valor mínimo, 1 representa igualmente preferíveis e, o valor máximo, 9 equivale a extremamente preferível. Tendo em conta a escala e os critérios obteve-se os seguintes valores:

Tabela 7 - Comparação entre os critérios

| | | | Importância |
|---|---|---|---|
| Integração com as tecnologias da empresa | relativamente | Linguagens utilizadas pela empresa | 3 |
| Tempo de desenvolvimento | | Linguagens utilizadas pela empresa | 2 |
| Integração com as tecnologias da empresa | | Tempo de desenvolvimento | 3 |

Pela tabela de comparação é possível notar que a integração com as tecnologias de empresa comparado com as linguagens utilizadas pela empresa é moderadamente preferível. O tempo de desenvolvimento confrontado com as linguagens utilizadas pela empresa é igual a moderadamente importante. E por fim a integração com as

tecnologias da empresa comparado com o tempo de desenvolvimento é moderadamente preferível.

Com a tabela de comparação dos critérios definida é necessário colocar os valores numa matriz (matriz de comparação).

Considerando:

- A - Tempo de desenvolvimento
- B - Integração com as tecnologias da empresa
- C - Linguagens utilizadas pela empresa

Tabela 8 - Matriz de comparação dos critérios

| | $A$ | $B$ | $C$ |
|---|---|---|---|
| $A$ | 1 | $\frac{1}{3}$ | 2 |
| $B$ | 3 | 1 | 3 |
| $C$ | $\frac{1}{2}$ | $\frac{1}{3}$ | 1 |

O passo seguinte é normalizar os valores da matriz.

Tabela 9 - Matriz de comparação dos critérios normalizada

| | $A$ | $B$ | $C$ |
|---|---|---|---|
| $A$ | 0,22222 | 0,2 | 0,33333 |
| $B$ | 0,66667 | 0,6 | 0,5 |
| $C$ | 0,11111 | 0,2 | 0,16667 |

Por fim efetuou-se o cálculo do *EigenValue* (Prioridade relativa), onde foi efetuada a média de cada uma das linhas da matriz de comparação normalizada.

Tabela 10 - Tabela de importância dos critérios

| | *Prioridade relativa* | *Importância* |
|---|---|---|
| $A$ | 0,25185 | Menor |
| $B$ | 0,58889 | Maior |
| $C$ | 0,15926 | Intermédio |

Através da tabela 11 é possível concluir que o critério tempo de desenvolvimento é o de menor importância, com uma percentagem de 15,89%, o critério de integração com as tecnologias da empresa é o de maior importância, com uma percentagem de 58,97% e o critério linguagens utilizadas pela empresa é de importância intermédia, com uma percentagem de 25,15%. Com os cálculos das prioridades, dos critérios, finalizados, é essencial verificar a veracidade dos valores.

**Consistência dos Critérios**

Uma vez que os valores atribuídos na comparação são subjetivos e baseados em opiniões, é necessário verificar a sua coerência. Esta é feita por uma análise de consistência da matriz apresentada na Tabela 11. Os cálculos seguiram a seguinte sequência [81]:

1) Soma dos valores de cada coluna e multiplicar pela respetiva prioridade relativa*, obtendo assim o* $\lambda_{máx}$ *;*
2) Cálculo do Índice de Consistência (IC), da seguinte forma $\frac{\lambda_{máx}-n}{n-1}$ no qual o n representa o número de linhas;
3) Cálculo da Relação de Consistência (RC), da seguinte forma $\frac{IC}{IAM}$ .

O valor de Inconsistência Aleatória Média (IAM) é um valor pré-definido que está dependente da dimensão da matriz em questão (Tabela 10). No caso em questão a nossa matriz tem de dimensão 3, logo o valor do IAM é 0,58. A consistência é validada se o valor final for inferior ou igual 0,1.

Tabela 11 - Índice de Consistência Aleatória

| **Dimensão da matriz** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| **IAM** | 0,00 | 0,00 | 0,58 | 0,90 | 1,12 | 1,24 | 1,32 | 1,41 | 1,45 | 1,49 |

Posto isto, foi executado o cálculo do primeiro ponto da sequência:

$$\begin{bmatrix} 1 & \frac{1}{3} & 2 \\ 3 & 1 & 3 \\ \frac{1}{2} & \frac{1}{3} & 1 \end{bmatrix}$$

$$SC1 \quad SC2 \quad SC3$$

Em que SCn é a soma da coluna e o n o número da coluna.

Obtendo os seguintes valores:

- SC1 = 4,5
- SC2 ≈ 1,66667
- S3 = 6

Com todos os valores para o $\lambda_{máx}$ , foi feito o seu cálculo:

$\lambda_{máx}$= 4,5*0,25185+1,66667*0,58889+6*0,15926 = 3,07036

Seguiu-se o Índice de Consistência:

$$IC = \frac{3,07036-3}{3-1} = 0,03518$$

E a Relação de Consistência:

$$RC=\frac{0,03518}{0,58} = 0,06066$$

Em conclusão como o RC é menor que 0,1 significa que a matriz utilizada para a comparação dos critérios é valida e não é necessário realizar os cálculos novamente.

**Definição de prioridades das Soluções**

Após a definição de prioridade dos critérios e da verificação da sua consistência, é necessário fazer a definição das prioridades das soluções, relativamente aos critérios. Para a definição de prioridades das soluções, seguiu-se os seguintes passos:

1) Comparou-se as soluções, traduziu-se a informação para uma matriz de comparação, normalizou-se e calculou-se a prioridade relativa, para cada um dos critérios.
2) As prioridades relativas foram colocadas numa matriz e multiplicou-se pelos valores das prioridades relativas dos critérios.

Considerando:

- Nova ferramenta - Criação de uma nova ferramenta e novo gerador
- Open-source com tecnologias - Implementação de uma ferramenta, que utiliza um gerador *open-source* de SDK e as tecnologias da empresa
- Open-source sem tecnologias- Implementação de uma ferramenta, que utiliza um gerador *open-source* de SDK e não utiliza as tecnologias da empresa

Foi feita a comparação entre as soluções, relativamente ao critério tempo de desenvolvimento (critério A).

Tabela 12 - Comparação das soluções

| | | | Importância |
|---|---|---|---|
| Open-source sem tecnologias | relativamente | Nova ferramenta | 3 |
| Open-source com tecnologias | | Nova ferramenta | 6 |
| Open-source com tecnologias | | Open-source sem tecnologias | 2 |

Resultando na matriz de comparação seguinte:

Tabela 13 - Matriz de comparação, relativamente ao critério A

| | *Nova ferramenta* | *Open-source com tecnologias* | *Open-source sem tecnologias* |
|---|---|---|---|
| *Nova ferramenta* | 1 | $\frac{1}{6}$ | $\frac{1}{3}$ |
| *Open-source com tecnologias* | 6 | 1 | 2 |

| | | | |
|---|---|---|---|
| *Open-source sem tecnologias* | 3 | $\frac{1}{2}$ | 1 |

De seguida, normalizou-se os seus valores e calculou-se a prioridade relativa:

Tabela 14 - Matriz de comparação normalizada e prioridade relativa

| | *Nova ferramenta* | *Open-source com tecnologias* | *Open-source sem tecnologias* | *Prioridade Relativa* |
|---|---|---|---|---|
| *Nova ferramenta* | 0,1 | 0,1 | 0,1 | 0,1 |
| *Open-source com tecnologias* | 0,6 | 0,6 | 0,6 | 0,6 |
| *Open-source sem tecnologias* | 0,3 | 0,3 | 0,3 | 0,3 |

Relativamente ao critério integração com as tecnologias da empresa (critério B) resultou a seguinte comparação:

Tabela 15 - Comparação entre os critérios

| | | | Importância |
|---|---|---|---|
| Open-source com tecnologias | relativamente | Nova ferramenta | 8 |
| Open-source sem tecnologias | | Nova ferramenta | 4 |
| Open-source com tecnologias | | Open-source sem tecnologias | 2 |

E da comparação surgiu a seguinte matriz de comparação:

Tabela 16 - Matriz de comparação, relativamente ao critério B

| | *Nova ferramenta* | *Open-source com tecnologias* | *Open-source sem tecnologias* |
|---|---|---|---|
| *Nova ferramenta* | 1 | $\frac{1}{8}$ | $\frac{1}{4}$ |
| *Open-source com tecnologias* | 8 | 1 | 2 |
| *Open-source sem tecnologias* | 4 | $\frac{1}{2}$ | 1 |

Sendo o passo seguinte normalizá-la e calcular a prioridade relativa com base nos valores da normalização.

Tabela 17 - Matriz de comparação normalizada e prioridade relativa

| | *Nova ferramenta* | *Open-source com tecnologias* | *Open-source sem tecnologias* | *Prioridade Relativa* |
|---|---|---|---|---|
| *Nova ferramenta* | 0,07692 | 0,07692 | 0,07692 | 0,07692 |
| *Open-source com tecnologias* | 0,61538 | 0,61538 | 0,61538 | 0,61538 |
| *Open-source sem tecnologias* | 0,30769 | 0,30769 | 0,30769 | 0,30769 |

No que se refere à comparação das soluções, tendo em conta o critério linguagens utilizadas pela empresa (critério C), obteve-se os seguintes valores:

Tabela 18 - Comparação dos critérios das soluções

| | | | Importância |
|---|---|---|---|
| Open-source sem tecnologias | relativamente | Nova ferramenta | $\frac{1}{5}$ |
| Open-source com tecnologias | | Nova ferramenta | $\frac{1}{5}$ |
| Open-source com tecnologias | | Open-source sem tecnologias | 1 |

Resultando a seguinte matriz de comparação:

Tabela 19 - Matriz de comparação, relativamente ao critério C

| | *Nova ferramenta* | *Open-source com tecnologias* | *Open-source sem tecnologias* |
|---|---|---|---|
| *Nova ferramenta* | 1 | 5 | 5 |
| *Open-source com tecnologias* | $\frac{1}{5}$ | 1 | 1 |
| *Open-source sem tecnologias* | $\frac{1}{5}$ | 1 | 1 |

De seguida, normalizou-se a matriz e calculou-se a prioridade relativa.

Tabela 20 – Matriz comparação normalizada e a prioridade relativa

| | *Nova ferramenta* | *Open-source com tecnologias* | *Open-source sem tecnologias* | *Prioridade Relativa* |
|---|---|---|---|---|
| *Nova ferramenta* | 0,71429 | 0,71429 | 0,71429 | 0,71429 |

| | | | | |
|---|---|---|---|---|
| *Open-source com tecnologias* | 0,14286 | 0,14286 | 0,14286 | 0,14286 |
| *Open-source sem tecnologias* | 0,14286 | 0,14286 | 0,14286 | 0,14286 |

A matriz das prioridades relativas das soluções, relativamente aos critérios, é a seguinte:

Tabela 21 - Prioridades relativas das soluções

| | *Critério A* | *Critério B* | *Critério C* |
|---|---|---|---|
| *Nova ferramenta* | 0,1 | 0,07692 | 0,71429 |
| *Open-source com tecnologias* | 0,6 | 0,61538 | 0,14286 |
| *Open-source sem tecnologias* | 0,3 | 0,30769 | 0,14286 |

E a matriz das prioridades relativas dos critérios, é a representada na Tabela 23.

Tabela 22 - Matriz das prioridades relativas dos critérios

| | *Prioridades relativas* |
|---|---|
| *Critério A* | 0,25185 |
| *Critério B* | 0,58889 |
| *Critério C* | 0,15926 |

Tendo em conta a matriz das prioridades relativas, em relação aos critérios e a matriz dos critérios, efetuou-se o cálculo das prioridades relativas das soluções (descrito anteriormente):

$$\begin{pmatrix} 0{,}1 & 0{,}07692 & 0{,}71429 \\ 0{,}6 & 0{,}61538 & 0{,}14286 \\ 0{,}3 & 0{,}30769 & 0{,}14286 \end{pmatrix} * \begin{pmatrix} 0{,}25185 \\ 0{,}58889 \\ 0{,}15926 \end{pmatrix}$$

Do cálculo efetuado obteve-se as seguintes prioridades relativas:

Tabela 23 – Matriz de Pesos Globais das soluções

| | *Peso Global* |
|---|---|
| *Nova ferramenta* | 0,18424 |
| Open-source *com tecnologias* | 0,53625 |
| Open-source *sem tecnologias* | 0,27950 |

**Consistência das Soluções**

Com a definição das soluções concluída, é necessário validar a consistência dos valores. Foram efetuadas as analises de consistência para todas as matrizes de comparação das soluções com os critérios (Tabela 14, 17 e 20). A validação segue o mesmo procedimento do cálculo da consistência dos critérios.

Obteve-se os seguintes valores de IC e RC das matrizes:

Tabela 24 - Valores de IC e RC para as matrizes

| *Matriz* | *IC* | *RC* |
|---|---|---|
| *Critério A* | 0 | 0 |
| *Critério B* | 0,00003 | 0,00005 |
| *Critério C* | 0,00002 | 0,00004 |

Como é possível observar, todos os valores de RC são inferiores a 0,1, logo é possível concluir que os valores das matrizes são consistentes.

**Conclusões**

Com a consistência dos cálculos comprovada é possível considerar os valores obtidos. Os resultados das soluções são os seguintes:

- Nova ferramenta – 18,42%
- *Open-source* com tecnologias – 53,63%
- *Open-source* sem tecnologias – 27,95%

Provando que a solução de implementar uma ferramenta, que utiliza um gerador *open-source* de SDK e as tecnologias da empresa, a melhor para o problema em questão.

### 4.2.5 Definição de Conceito

Com a seleção da ideia concluída é necessário definir o conceito do que irá ser implementado. A solução proposta é uma ferramenta, que utiliza um gerador *open-source* de SDK e as tecnologias da empresa, nomeadamente *Jenkins* e a base de dados do *MongoDB*. É pretendido que a sua utilização seja intuitiva e que a solução cumpra os critérios impostos pela empresa.

## 4.3 Valor da Solução

### 4.3.1 Valor, Valor para o Cliente, Valor Percecionado

De forma a compreender a importância de um produto, é necessário analisar o seu valor, valor para o cliente e o valor percecionado.

O valor de um produto é a importância de um produto ou serviço na perspetiva de um cliente. É um conceito fundamental para o desenvolvimento de um produto e na definição de um preço de mercado [82]. Alguns exemplos de ações que aumentam o valor de um produto são novas funcionalidades, a confiabilidade, fácil usabilidade e a eficiência.

O valor para o cliente é o valor da satisfação dos clientes, perante um produto ou serviço, sendo este subjetivo e uma avaliação pessoal [83]. O valor da satisfação de um cliente varia consoante a sua utilização e a sua experiência, com o mesmo. Na perspetiva do projeto, referenciado neste documento, com a facilidade de obtenção de informação relativa à API, através de uma interface intuitiva, é esperado que o cliente apresente um valor de satisfação elevado.

O valor percecionado é uma comparação entre os benefícios e os sacrifícios de um produto ou serviço, pelo cliente [84]. Relativamente ao projeto, os benefícios da solução, para o cliente, são o aumento da produtividade, redução de custos, a simplificação de processos e a centralização de informação. No entanto a adaptação para com a utilização da ferramenta pode ser considerada um sacrifício para o cliente. É de salvaguardar que o valor percecionado é variável consoante a opinião pessoal de cada cliente.

### 4.3.2 Proposta de Valor

A proposta de valor expõe, de uma forma clara, objetiva e transparente, a relevância da utilização de um serviço ou da compra de um produto, para o cliente [85]. Representa assim uma parte fulcral de um negócio, uma vez que sem um conceito bem executado é difícil obter sucesso.

Aplicando este conceito ao projeto, a proposta de valor traduz-se numa ferramenta de monitorização e gestão de API, aumentando a produtividade da equipa de desenvolvimento quando comparada com o processo atual de notificação de erros e resolução dos mesmos. Para além disso existe ainda redução de custos, uma vez que irão ser necessários menos recursos para detetar erros e desempenhar processos. E ainda a centralização de informação relativa à API e aos seus processos.

## 4.4 Modelo de Negócio Canvas

O modelo de negócio Canvas é uma ferramenta vantajosa para a definição do modelo de negócio de uma empresa ou de um projeto. Permite uma assistência visual para a compreensão do funcionamento de um negócio [86]. Responde a questões como:

- Quem irão ser os principais parceiros e fornecedores?
- Quais são as atividades mais importantes?
- Quais são os recursos que o negócio necessita?
- Quais são os custos?
- Qual é o problema a ser resolvido?
- Quem são os clientes?
- Qual é a relação a estabelecer com os clientes?
- Como conseguimos alcançá-los?

Este modelo é constituído por parceiros, atividades e recursos chave, proposta de valor, relação com os clientes, canais, segmentos de clientes, estrutura de custos e fontes de receita [86].

A Figura 13 ilustra o modelo de negócio Canvas elaborado para o projeto desta tese.

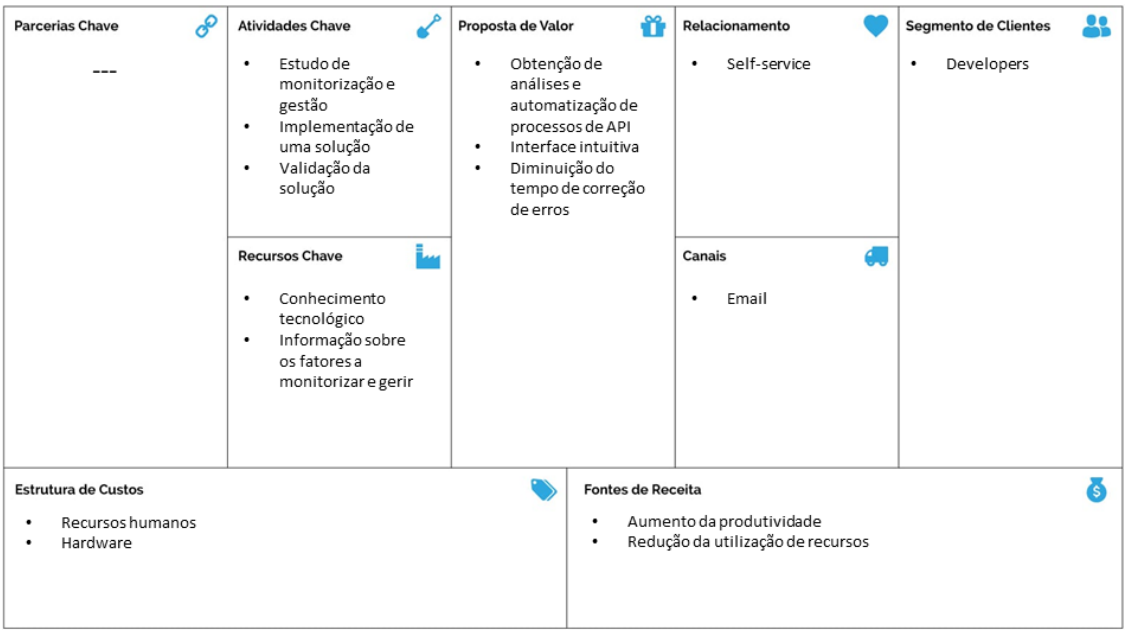


Figura 13 – Modelo de negócio Canvas do projeto

Como é possível observar na figura cada quadrado corresponde a uma área do negócio, de seguida irão ser explicados, cada um deles, com maior detalhe:

- **Parceiras Chave** – nesta área é pretendido que se enumere as alianças estratégicas com fornecedores ou com outras empresas, que irão possibilitar ajuda para o desenvolvimento do produto. Uma vez que o produto é uma ferramenta interna, ou seja, vai ser apenas utilizada pela equipa de desenvolvimento da E-goi e mantida pela empresa, não existem parcerias chave.
- **Atividades Chave –** neste campo são enumerados os processos fundamentais para a eficácia do modelo de negócio. No caso do presente projeto, as atividades são o estudo da monitorização e gestão de API, de modo a perceber os pontos essenciais a analisar e melhor forma de executar o desenvolvimento, implementação de uma solução e a validação da solução perante a correspondência com o que foi pedido pela empresa e o seu funcionamento.
- **Recursos Chave –** este campo visa perceber quais as necessidades a nível de recursos que o negócio exige. Os recursos podem ser de caratér humano, físico, intelectual e financeiro. No âmbito do projeto é necessário o conhecimento tecnológico para implementar a ferramenta e a informação sobre fatores a monitorizar e gerir de modo a conceber uma ferramenta que se adeque à situação.
- **Proposta de Valor –** nesta área é apresentado o que irá ser oferecido aos clientes. É favorável que a oferta se destaque pela diferença, relativamente à

concorrência. Tendo em conta o projeto as ofertas são a obtenção de informação relacionada com a API através de uma interface intuitiva, uma vez que outras ferramentas semelhantes são complexas e difíceis de usar. Com a facilidade de obter a informação é possível diminuir o tempo de resposta a problemas e detetá-los mais cedo.

- **Relacionamento –** este campo pretende expor como irá ser formada a relação com os clientes. No âmbito deste projeto é pretendido que os clientes sejam independentes na utilização da ferramenta, no entanto irá ser dada assistência quando estritamente necessário.
- **Canais –** pretende-se, com este ponto, saber como irão ser alcançados os clientes. No âmbito do projeto, descrito neste documento, irão ser enviados emails, com uma pequena introdução à ferramenta, aos colaboradores da equipa de desenvolvimento do produto.
- **Segmento de Clientes –** esta área tem como objetivo enumerar quem são os clientes que irão usufruir do produto. No caso deste projeto irão ser a equipa de desenvolvimento, uma vez que são os que possuem o conhecimento e aptidões necessárias para agir conforme as informações obtidas.
- **Estrutura de Custos –** com este ponto é pretendido que se identifique os custos de maior importância inerentes ao negócio. No projeto em questão são os recursos humanos (ou mão de obra) e o hardware quer para a implementação quer para a guardar a solução.
- **Fontes de Receita –** nesta área é esperado que se enumere as formas como irá ser conseguido o retorno do investimento no negócio. No âmbito do projeto é obtido através do aumento da produtividade da equipa de desenvolvimento, uma vez que irão ser simplificadas tarefas, e a redução da quantidade de recursos utilizados, para desempenhar as tarefas que a ferramenta simplifica.

## 4.5 Cadeia de Valor de Porter

O modelo de cadeia de valor, criado por Michael Porter, permite analisar, como uma empresa, cria valor e compete no mercado. A forma como se relacionam os elementos, do modelo, entre si determinam os custos e prejudica os lucros [87].


Figura 14 - Cadeia de Valor de Porter

A cadeia de valor divide-se em dois tipos de atividades, primárias e de apoio. As atividades primárias estão relacionadas com as fases a criação, venda, manutenção e suporte de um produto ou serviço. Estas dividem-se em cinco áreas, nomeadamente:

- Logística de Entrada – contempla os processos de preparação para a criação do produto (inventário, transporte, entre outros). No âmbito do projeto, uma vez que não se trata de um produto físico, os processos serão o estudo dos critérios a gerir e monitorizar.
- Operações – é pretendido que se enumere as atividades que irão transformar matéria prima(inputs) no produto final. No caso do projeto a implementação de uma ferramenta de manutenção e gerenciamento de API, será a operação efetuada.
- Logística de Saída – diz respeito às atividades necessárias para, após a criação do produto, fazer chegar ao cliente. Relativamente ao projeto, as suas atividades de logística de saída são a manutenção e a atualização do produto.
- Marketing e Vendas – diz respeito aos processo utilizados para persuadir o cliente a comprar e utilizar o produto ou serviço. No âmbito do projeto, irão ser enviados emails com uma introdução da ferramenta.
- Serviços – contempla as atividades de acompanhamento do cliente após a aquisição do produto ou serviço, nomeadamente apoio ao cliente, formação, serviços de reparação, entre outras. Relativamente à solução, deste documento, irá ser feito algum apoio ao cliente, mas é esperado que o cliente seja autónomo.

Relativamente às atividades de apoio, estas são as ações responsáveis por auxiliar, direta ou indiretamente, as atividades primárias. Estas dividem-se em quatro áreas:

- Infraestrutura – representa os sistemas necessários às operações diárias de uma organização, como gestão administrativa, legal, financeira, contabilística e qualidade.
- Gestão de Recursos Humanos – contempla as atividades de gestão de recursos humanos da empresa, como recrutamento, desenvolvimento de conhecimentos, retenção e compensação de funcionários. Uma vez que os colaboradores representam uma grande fonte de valor para a empresa, é vantajoso utilizar boas práticas de gestão. Alguns exemplos são estágios com instituições de ensino, formações, avaliações, progressão de carreira e prémios de incentivo ao bom desempenho.
- Desenvolvimento Tecnológico – representa as ações relacionadas com a parte tecnológica, como investigação e desenvolvimento, automação de processos e design.
- Aquisição e Compras – constituem os processos de aquisição de recursos necessários ao funcionamento da empresa. Nomeadamente a aquisição de hardware e software para o desempenho de funções.

Um outro elemento do modelo de negócio é a margem, que representa a diferença entre o valor percebido e custo de todo o processo de criação do produto ou serviço.

Através da análise dos elementos descritos, do modelo de negócio de Porter, é possível aumentar a qualidade da prestação de serviços, do valor do produto e da satisfação do cliente [69].

# Capítulo 5

# Design da solução

Com o valor da solução comprovado, o passo seguinte é fazer um planeamento para a implementação. Este tem como finalidade definir como a solução irá responder ao problema enunciado na secção 1.2. Para elaborar o planeamento é necessário fazer um levantamento dos requisitos e produzir artefactos que auxiliem o desenvolvimento da solução.

## 5.1 Análise

Com esta secção pretende-se expor o processo de identificação dos requisitos, funcionais e não funcionais, e a elaboração do diagrama de fluxo da ferramenta, onde são apresentados os seus processos.

### 5.1.1 Requisitos funcionais

Nesta secção são elencados os requisitos funcionais necessários à ferramenta, que resultaram de uma análise das necessidades da empresa e das opiniões dos clientes. Os requisitos funcionais representam o que a ferramenta necessita de disponibilizar aos utilizadores. Estes são:

1) **Criar registo –** o utilizador que não está registado cria conta na plataforma;
2) **Consultar análises por pedido –** o utilizador e o administrador conseguem visualizar as análises dos pedidos da API;
3) **Consultar análises de clientes –** o utilizador e o administrador conseguem ver as análises de pedidos que os clientes fizeram;
4) **Adicionar API –** o administrador pode adicionar outras API para monitorizar e gerir;
5) **Criar teste –** o utilizador pode criar testes;
6) **Consultar testes –** o utilizador pode consultar testes passados;

O diagrama de casos de uso da ferramenta, tendo em conta os requisitos, é o seguinte:

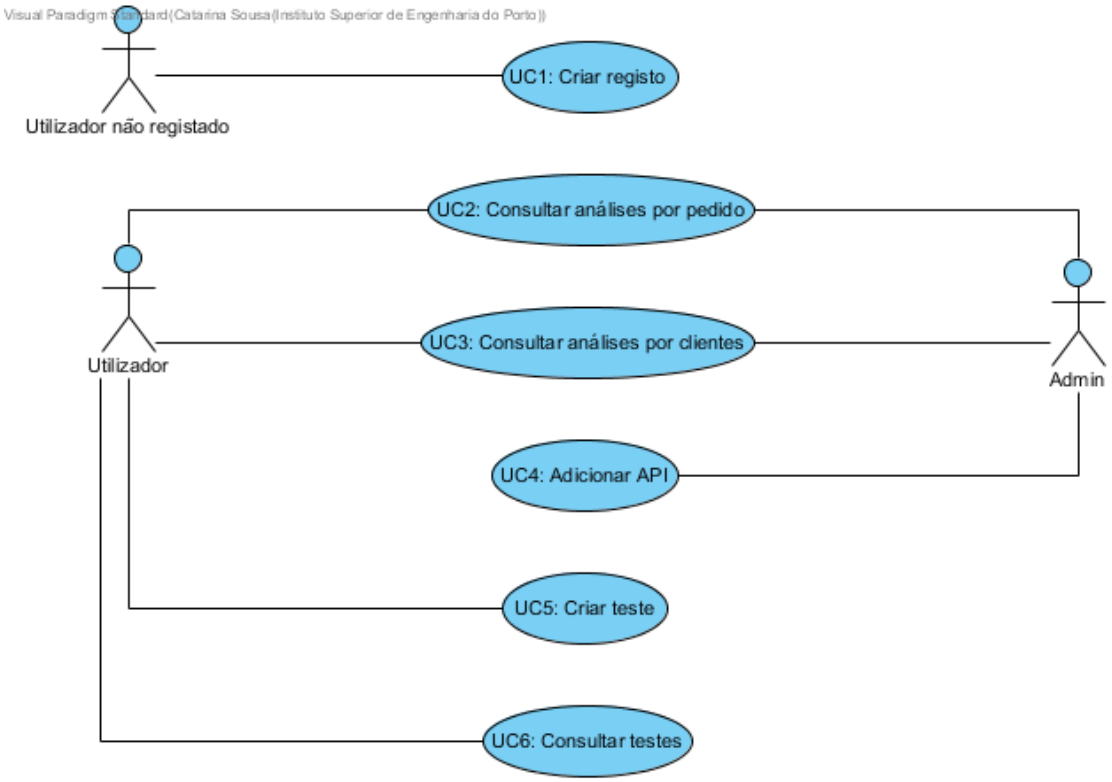


Figura 15 - Diagrama de casos de uso

O diagrama da Figura 15 tem como objetivo demonstrar as funcionalidades da ferramenta e que atores as executam. Relativamente ao utilizador não registado a única ação que ele pode fazer na ferramenta é criar registo. O ator admin pode adicionar API, consultar analises por pedido e por clientes. E por fim, o utilizador consultar análises por pedido e por clientes, criar teste e consultar testes.

Os casos de uso descrevem com os utilizadores interagem com o sistema. Com recurso da notação UML, é possível desenvolver representações das ações descritas nos casos de uso, denominados diagramas de sequência de sistema (SSD).

O primeiro caso de uso, **criar registo,** começa por o utilizador não registado, iniciar, no sistema, a operação de criação de nova conta. O sistema solicita os dados. O utilizador não registado introduz os dados solicitados. O sistema valida os dados e informa do sucesso da operação. O SSD relativo a esta troca encontra-se na Figura 16.

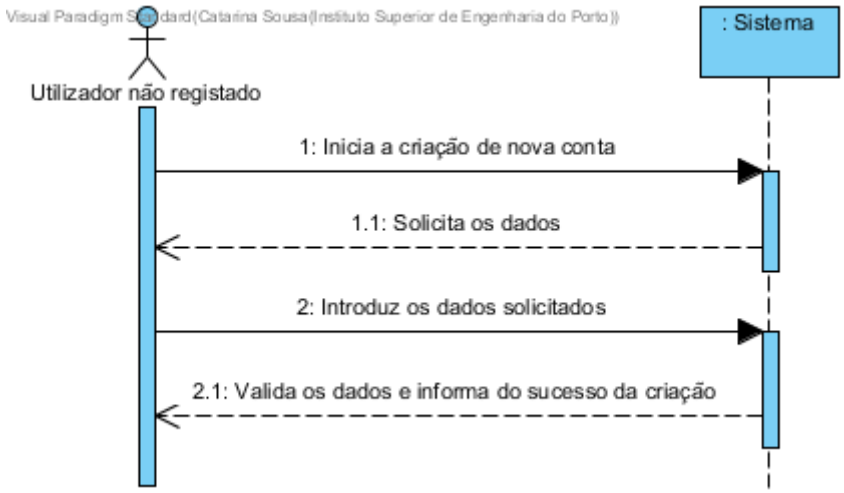


Figura 16 - SSD do caso de uso criar registo

O caso de uso **consultar análises por pedido** é efetuado, também, pelo utilizador e admin. Os atores iniciam a consulta das análises por pedido, no sistema. O sistema finaliza o fluxo ao apresentar as análises. O SSD do fluxo do caso de uso está representado na Figura 17.

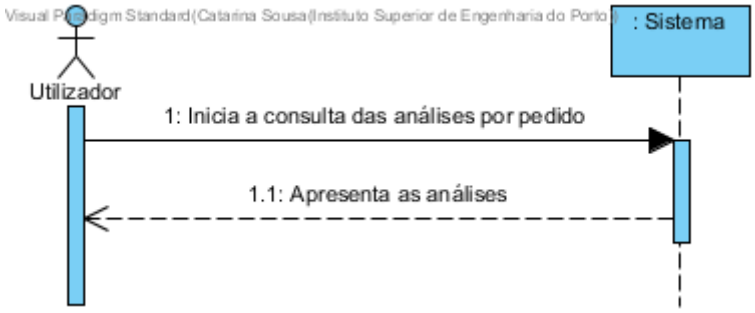


Figura 17 - SSD do caso de uso consultar análises por pedido

Relativamente ao caso de uso **consulta das análises por cliente** os atores são o utilizador e o admin, mais uma vez. Os atores iniciam a consulta das análises por cliente, no sistema. E, por fim, o sistema apresenta as análises. O diagrama que espelha esta interação encontra-se na Figura 18.

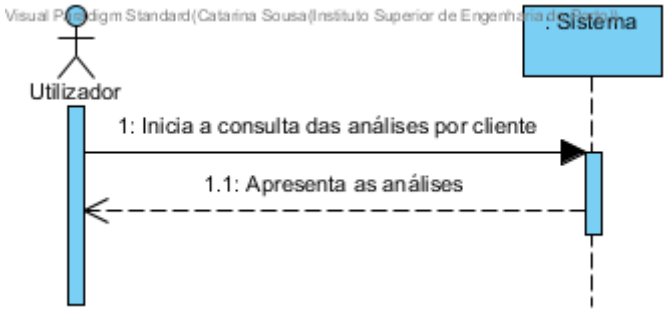


Figura 18 - SSD do caso de uso consultar análises por cliente

O caso de uso **adicionar API** começa pelo admin iniciar a operação de adicionar API, no sistema. O sistema solicita os dados. O admin introduz os dados solicitados. E o sistema informa do sucesso da adição. O SSD do fluxo do caso de uso está retratado na Figura 19.

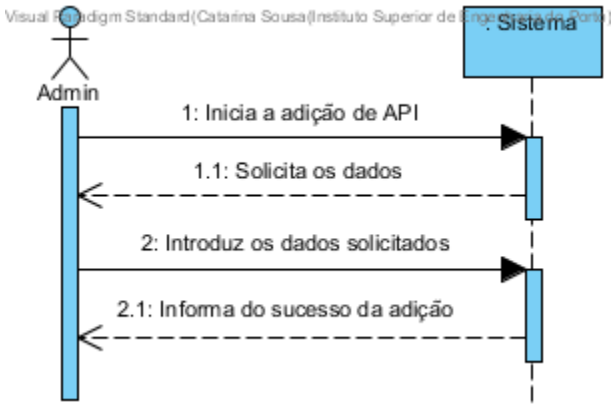


Figura 19 - SSD do caso de uso adicionar API

No caso de uso **criação de teste** o utilizador inicia, no sistema, a criação de teste. O sistema solicita os dados. O utilizador introduz os dados solicitados. O sistema valida os dados e informa do sucesso da criação. A Figura 20 apresenta a interação descrita.

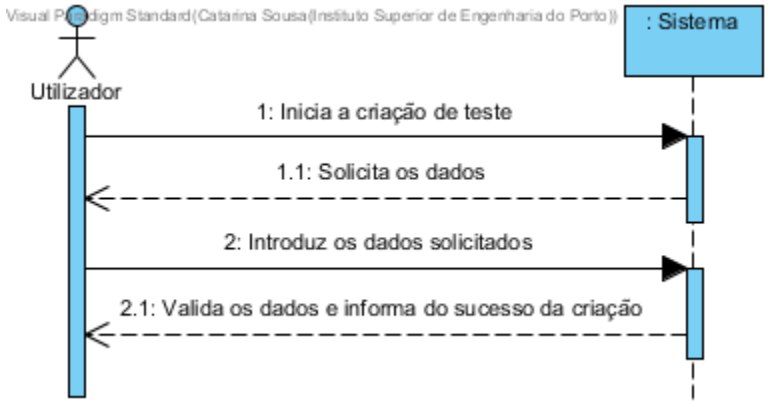


Figura 20 - SSD do caso de uso criar teste

E, por último, no caso de uso **consultar testes** o utilizador inicia a consulta dos testes, no sistema. E o sistema apresenta a lista de testes. A representação desta interação encontra-se na Figura 21.

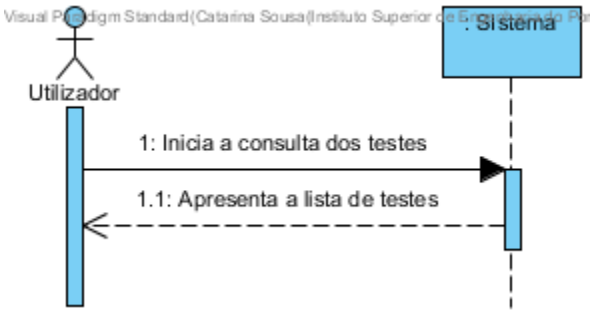


Figura 21 - SSD do caso de uso consultar testes

### 5.1.2 Requisitos não funcionais

Com os requisitos funcionais enunciados, falta mencionar os requisitos não funcionais. Estes requisitos correspondem aos atributos necessários para garantir a qualidade da solução, com este intuito é utilizado o modelo de FURPS+. A Tabela 26 apresenta os requisitos não funcionais identificados associando-os à correspondente categoria FURPS.

Tabela 25 – Requisitos não funcionais identificados

| Classificação FURPS+ | Requisitos |
|---|---|
| **Funcionalidade** | A interface deve ser adaptável aos diversos tipos de utilizadores. |
| **Usabilidade** | A interface deve ser intuitiva, de modo a garantir a facilidade na utilização da ferramenta. |
| **Suportabilidade** | Suporta diferentes browsers (como por exemplo *Chrome*, *Edge*).<br>O código é escalável e adaptável a alterações e adições. |
| **Implementação** | Linguagens de desenvolvimento a adotar para a ferramenta: Angular, Node.js, HTML e CSS.<br>Os SDK devem ser gerados nas linguagens Java, Php e Python. |
| **Segurança** | Os utilizadores têm de fazer login e registo na ferramenta. E a palavra-passe é encriptada na base de dados. |
| **Design** | Utilizar uma base de dados relacional. |

## 5.2 Domínio

O domínio de um projeto são o conjunto dos conceitos principais para a resolução de um problema. Usando notação UML, novamente, é possível desenvolver um modelo de domínio (diagrama) que represente o domínio do problema, enunciada na secção 3.2.

No âmbito do problema, identificado na secção 1.2, a proposta de modelo de domínio encontra-se representada na Figura 22. As quatro entidades identificadas para o domínio são: utilizador (da aplicação), teste, pedido e API. A entidade teste está ligada a utilizador, pois um utilizador pode criar vários testes. A entidade pedido está ligada a teste, uma vez que o teste é efetuado a um pedido. E a entidade API está ligada com o pedido, pois os pedidos são feitos à API.

Com esta proposta é possível começar o desenvolvimento da ferramenta com fim a resolver a adversidade reconhecida.

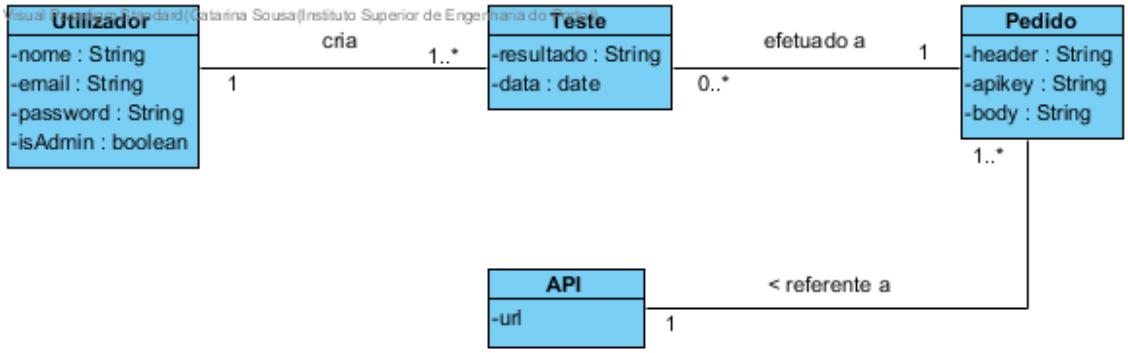


Figura 22 - Modelo de domínio da ferramenta de monitorização e gestão de API

## 5.3 Arquitetura

A arquitetura tem como finalidade desenhar a estrutura do projeto, com foco no comportamento dos componentes, descartando limitações de implementação ou tecnologias. Foi utilizada, mais uma vez, a notação UML, para elaborar o diagrama de componentes, representado na Figura 23.

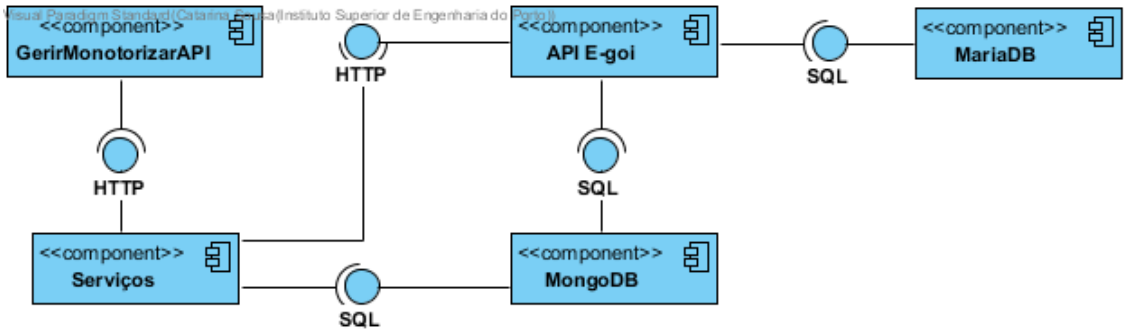


Figura 23 - Diagrama de Componentes da ferramenta de monitorização e gestão de API

Como é possível observar pelo diagrama da Figura 23, o componente GerirMonitorizarAPI comunica com os Serviços, por protocolo *Hyper Text Transfer Protocol* (HTTP). Estes comunicam com as componentes MongoDB e API E-goi. E por fim a componente API E-goi comunica com a base de dados da E-goi e com MongoDB.

A componente GerirMonitorizarAPI, desenvolvida na linguagem de programação Angular [89], é onde se encontra a lógica da interface, que permite aos seus utilizadores criar e visualizar testes à API e consultar análises por pedido ou por cliente. A Figura 24 apresenta a arquitetura desta componente.

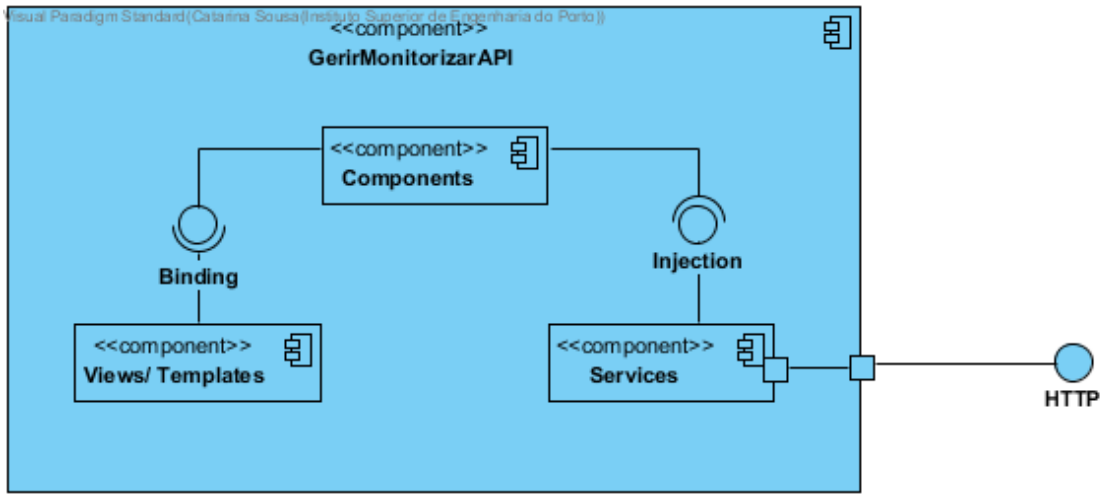


Figura 24 - Diagrama da componente GerirMonitorizarAPI

Como é possível observar pela Figura 24 a componente divide-se em três subcomponentes: *views/templates*, *components* e *services*. As *views/templates* são onde estão localizados os elementos que são expostos ao utilizador (por exemplo caixas de texto e botões)[90] e comunicam com as *componentes* por *binding* (mecanismo que coordena partes do *template* com as partes da *component*)[91]. As *components* controlam as ações e informação das *views,* utilizando os *services*[91]. E, por fim, os *services* são responsáveis pelo processamento da informação e comunicam com as componentes por *injection* (mecanismo que injeta os *services* nas *component*)[92].

Visto que a componente GerirMonitorizarAPI apenas inclui lógica para a interface, é necessária uma componente que tenha a lógica da persistência de dados, neste caso é Serviços. Esta trata-se de uma API REST, desenvolvida na linguagem de programação Node.js, que contém a lógica de negócio e do tratamento de dados. A componente Serviços comunica com a API da E-goi, por protocolo HTTP, e com o MongoDB, por *Structured Query Language* (SQL). A Figura 25 apresenta a arquitetura da componente Serviços.

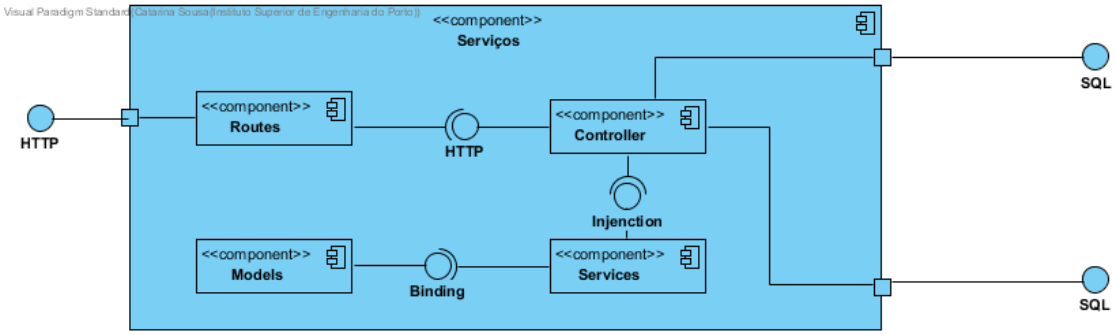


Figura 25 - Diagrama da componente Serviços

Pela Figura 25 é possível aferir que a componente Serviços apresenta as subcomponentes: *routes*, *controller*, *services* e *models*. As *routes* são o meio de comunicação entre o Serviços e GerirMonitorizarAPI, por protocolo HTTP, e mapea para os *controllers*. Estes lidam com toda a lógica relacionada com a validação de parâmetros de pedidos e envio de respostas, todo este processo usando *services,* por *injection*. Estes tratam das alterações e consultas à base de dados e lançam exceções. Os *models* representam os conceitos do domínio[93] e comunicam com os *services* por *binding*.

Pela análise da Figura 23 é ainda possível visualizar que a componente API E-goi comunica com duas componentes de bases de dados diferentes, MongoDB e MariaDB, por SQL. A componente MongoDB, é responsável por a persistência de dados da ferramenta e guarda os dados dos pedidos feitos à API E-goi. Utiliza a tecnologia MongoDB [59] que é um sistema de gestão de base de dados relacional. A componente MariaDB, é responsável pela persistência de dados do produto E-goi.

A forma como estão organizados os componentes facilita adicionar ou alterar funcionalidades à ferramenta, uma vez que se encontra repartida por diferentes componentes, oferecendo assim escalabilidade e adaptabilidade (recurso não funcional da Tabela 26, na secção 5.1.2).

## 5.4 Implantação

A implantação representa o processo de instalação de um projeto em máquinas, com o objetivo de conceder acesso aos utilizadores. Este processo é de grande importância, uma vez que o custo de instalação, em máquinas, é dispendioso. De forma a apresentar a implantação do projeto deste documento foi utilizada, novamente, notação UML, neste caso para elaborar um diagrama de implantação, representado na Figura 26.

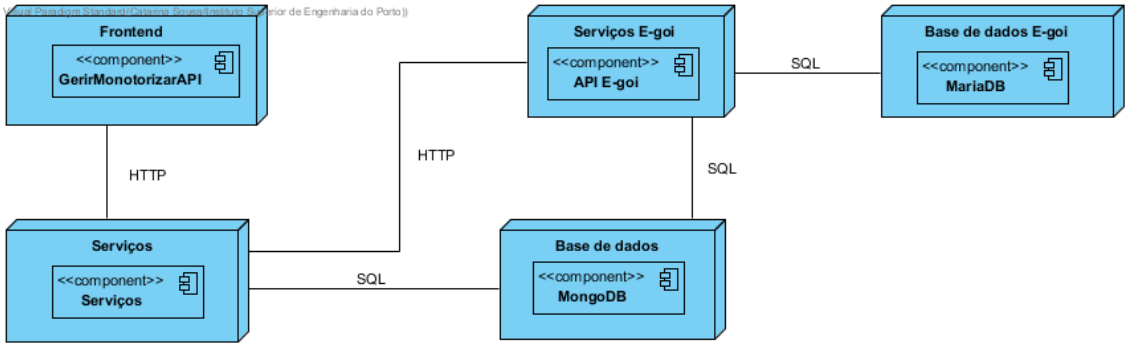


Figura 26 - Diagrama de Implantação da ferramenta de monitorização e gestão de API

Como é possível observar pela Figura 26 a base de dados MongoDB encontra-se num servidor autónomo. Esta decisão foi tomada para desacoplar responsabilidades comuns, entre os Serviços e a base de dados MariaDB. Para além disso também é possível reparar que a interface gráfica (GerirMonitorizarAPI) e os Serviços se encontram em máquinas diferentes, a tomada desta decisão derivou do facto de serem utilizadas tecnologias distintas (Angular e Node.js) nas componentes.

## 5.5 Serviços

De forma a que toda a informação disponibilizada e criada por um utilizador na interface de um sistema persista é fundamental que haja uma conexão entre a interface e o servidor que trata e guarda os dados. Como apresentado na secção 5.3., o componente Serviços contém os serviços que comunicam com a base de dados da ferramenta e faculta *endpoints* de ligação com os serviços.

Os serviços utilizados para a construção da ferramenta são apresentados na Tabela 27.

Tabela 26 - Listagem de serviços

| | | |
|---|---|---|
| POST | /Auth/SignUp | Criar uma nova conta |
| POST | /Auth/Login | Fazer *login* na ferramenta |
| GET | /APInterface | Obter a listagem de API |
| POST | /APInterface | Adicionar nova API na ferramenta |
| GET | /Test | Obter a listagem dos testes feitos pelo utilizador |
| POST | /Test | Executar novo teste |
| GET | /AnalysisPerClient | Obter listagem dos clientes que executam mais pedidos com erros |

| | | |
|---|---|---|
| GET | /AnalysisPerRequest | Obter listagem de pedidos com mais erros |

Os pedidos enunciados na tabela utilizam o protocolo *Hyper Transfer Protocol* (HTTP), que possibilita a utilização de *token* de autenticação nos *headers*, e consequentemente assegurar a segurança no acesso aos serviços.

# Capítulo 6

# Implementação

Concluída a fase de *design*, torna-se necessário descrever o processo de desenvolvimento. Neste capítulo irá ser abordada a metodologia de desenvolvimento, a implementação da ferramenta de gestão e monitorização da Interface de Programação de Aplicações (API) (intitulado de GesMo), com base no design do Capítulo 5, e as alterações dos serviços.

## 6.1 Metodologia de Desenvolvimento

Para a realização de um projeto em equipa, é necessário que sejam utilizadas metodologias de desenvolvimento iguais para todos os membros. A E-goi aplica um conjunto de técnicas para a colaboração, entre os mais de 30 programadores. Apesar deste projeto ter sido realizado individualmente foram utilizadas as práticas da empresa para desenvolvimento de *software*.

A ferramenta Git é uma das ferramentas usadas para controlo de versões dos projetos da E-goi, complementarmente com a aplicação GitLab. Esta permite armazenar os projetos no servidor interno da empresa. O projeto, descrito no documento, foi desenvolvido à parte dos outros projetos da E-goi, tendo assim só uma *branch,* a *master* (versão principal).

Para partilhar, com os clientes, atualizações e correções na plataforma E-goi, a empresa usa a ferramenta de *contínuos integration* Jenkins, abordada na secção 3. Esta encontra-se num servidor interno e possibilita gerir os projetos. Em conjunto com o GitLab, a ferramenta possibilita a compilação e automatização de testes. Assim quando é feita uma alteração no código e é enviada para o servidor interno, são despoletados um conjunto de processos, nomeadamente a compilação do projeto com a alteração, a geração e execução de testes unitários e o *deploy*. No caso deste projeto, o Jenkins foi utilizado para a geração e automatização de *Software Development Kits* (SDK), descrita na secção 6.4.

## 6.2 Interface Gráfica

O principal objetivo deste projeto é a implementação de uma ferramenta de gestão e monitorização de API, detalhado na secção 1.3. Na implementação da interface gráfica foi utilizada a linguagem de programação Angular, satisfazendo um dos requisitos não funcionais, apresentado na secção 5.1.2.

A página inicial da ferramenta encontra-se divida em 2 componentes: **HeaderComponent** e **HomePageComponent**. A HeaderComponent diz respeito à barra preta no topo da página (comum a todas as páginas) e a HomePageComponent a todo o resto da página. Na Figura 27 é apresentada a página inicial.

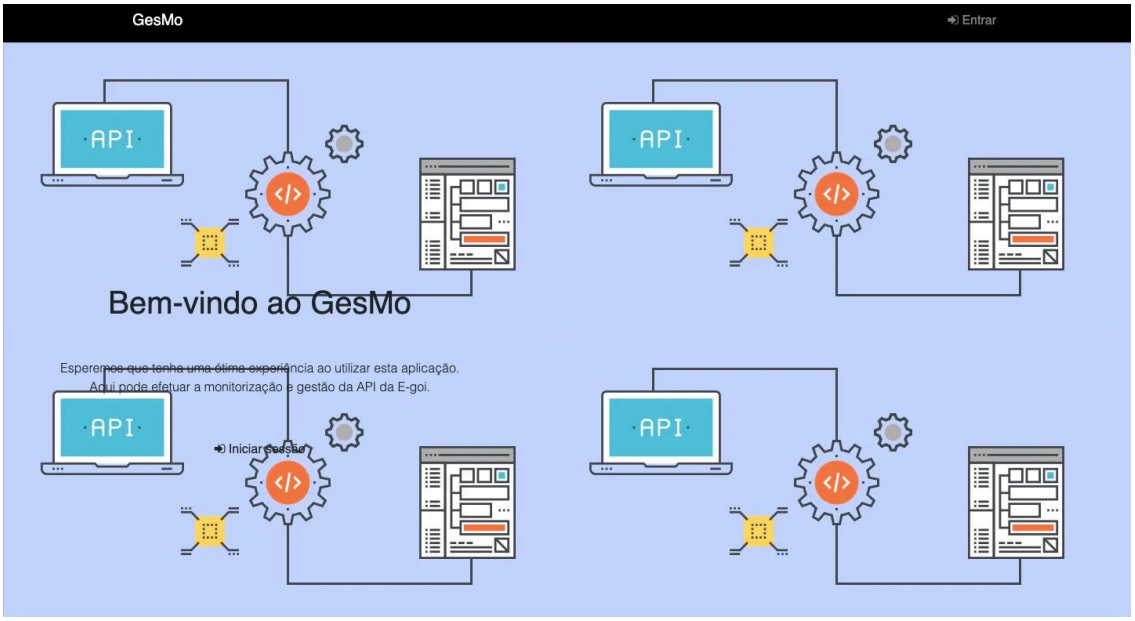


Figura 27 - Página inicial

Quando os utilizadores carregam no botão do lado superior direito, na página inicial, que diz "Entrar", são reencaminhados para a página de realização de *login*. Para além da componente **HeaderComponent**, é utilizada também a **LoginComponent** que contém toda a área branca. A Figura 28 apresenta o resultado da página de *login*.

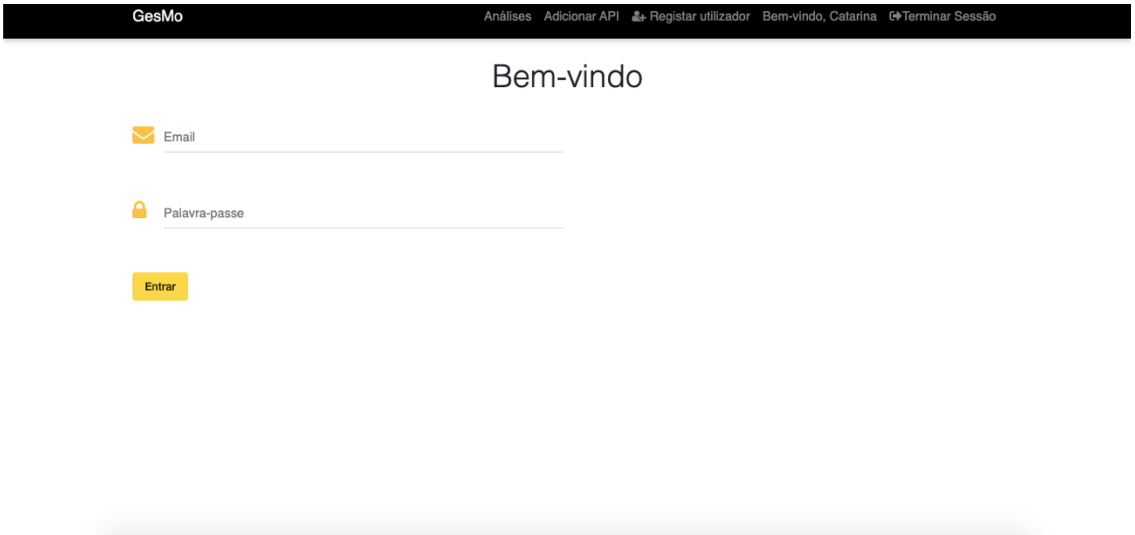


Figura 28 – Página de realização de *login*

De forma a perceber como interagem as componentes para a realização do *login*, foi usado um diagrama UML de sequência, apresentado na Figura 29.

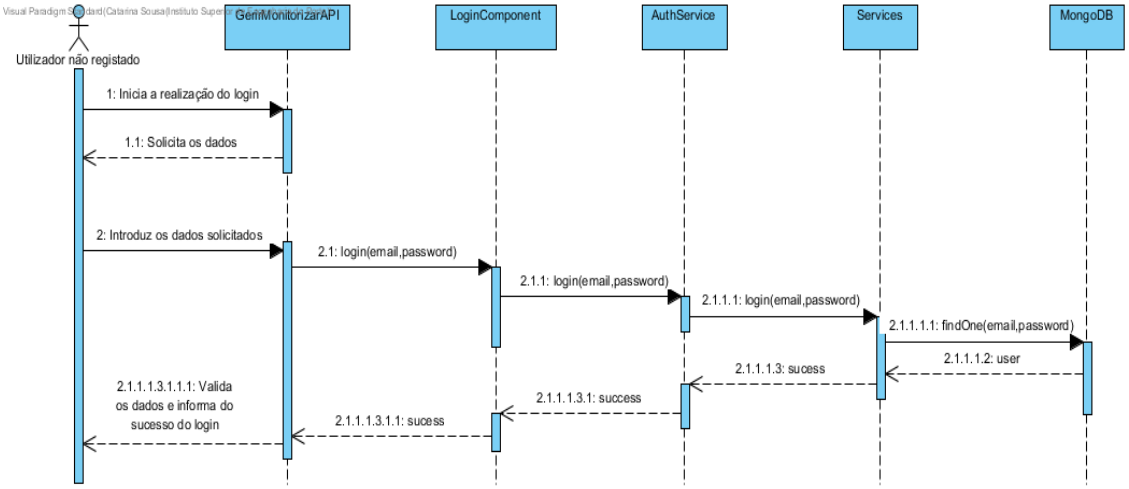


Figura 29 - Diagrama de sequência para o login

Pela análise da Figura 29 é possível ver que o utilizador não registado inicia a realização do login e a interface (**GerirMonitorizarAPI**) solicita os dados. O utilizador não registado introduz os dados solicitados e a interface chama o método login com os parâmetros email e password da **LoginComponent**, que consequentemente chama o método com o mesmo nome e os mesmo parâmetros do **AuthService** e da componente de *backend* **Services**. Esta componente chama o método findOne com os parâmetros email e password que retorna o utilizador da componente de base de dados **MongoDB** para os Services. Estes numa situação em que foi tudo validado e retorna sucesso, passa a mensagem de volta para o AuthService, LoginComponent e o GerirMonitorizarAPI. E por fim a interface informa do sucesso do login ao utilizador não registado.

No caso de utilizador não ter conta, é possível efetuar o seu registo. Para a página de registo, a componente utilizada foi **SignUpUsersComponent**, que contém toda a área branca. Na Figura 30 é apresentada a página de criação de registo.


Figura 30 -Página de criar registo

A interação das componentes para a criação do registo, está apresentada na Figura 31.

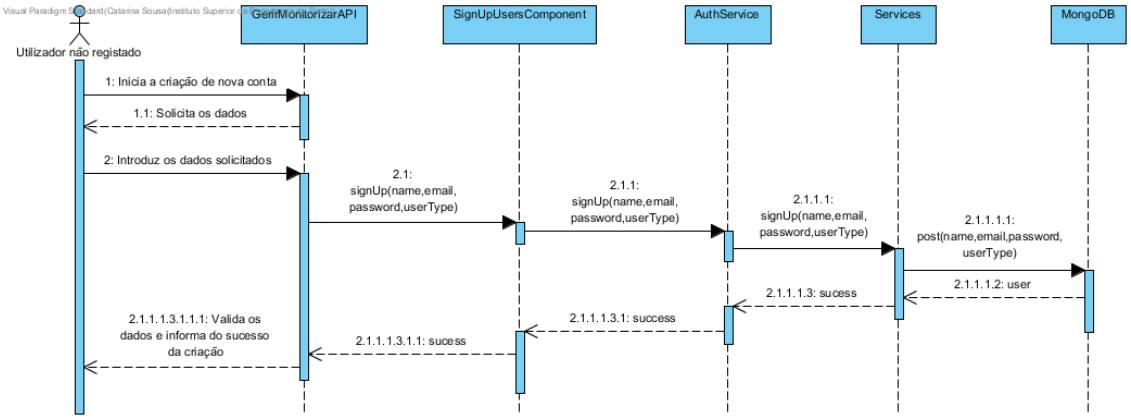


Figura 31 - Diagrama de sequência da criação de registo

Na Figura 31 é possível observar que o utilizador não registado inicia a criação da nova conta e a componente **GerirMonitorizarAPI** solicita os dados. O utilizador não registado introduz os dados solicitados e a interface chama o método signUp com os parâmetros name, email, password e userType (admin ou utilizador) da **signUpUsersComponent**, que consequentemente chama o método com o mesmo nome e os mesmo parâmetros do **AuthService** e da componente de *backend* **Services**. Esta componente chama o método post com os parâmetros mencionados no signUp que retorna o utilizador da componente de base de dados **MongoDB** para os Services. Estes numa situação em que foi tudo validado e retorna sucesso, passa a mensagem de volta para o AuthService, signUpUsersComponent e o GerirMonitorizarAPI. E por fim a interface informa do sucesso da criação do registo do utilizador não registado.

Um utilizador ou admin pode consultar as análises de pedido. Para esta página foi usada a componente **AnalysisRequestComponent**, para toda a área branca e a biblioteca pública **CanvasJS**. Esta biblioteca permite utilizar gráficos responsivos (adaptam-se a comandos, por exemplo ao colocar o rato sob uma parte do círculo são apresentados os detalhes) e tem suporte para múltiplos browsers (Chrome, Firefox, Safari e IE8+)[94]. Na Figura 32 é apresentada a página.

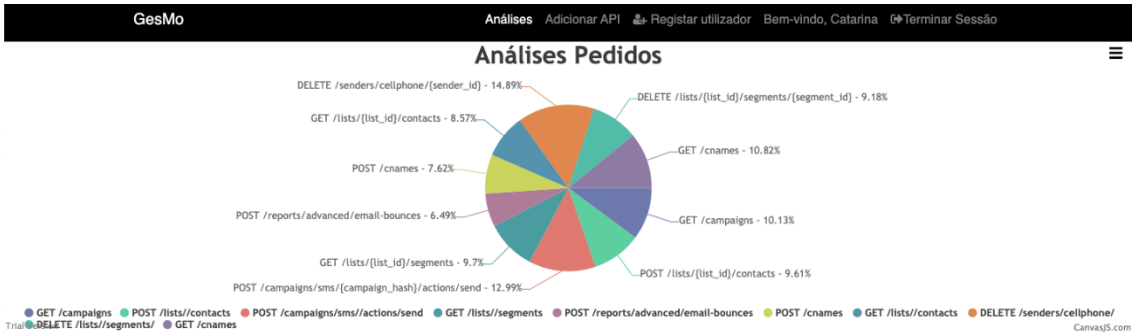


Figura 32 – Página de consulta de análises por pedido[2]

A comunicação das componentes para a consulta de análises por pedido, está apresentada na Figura 33.

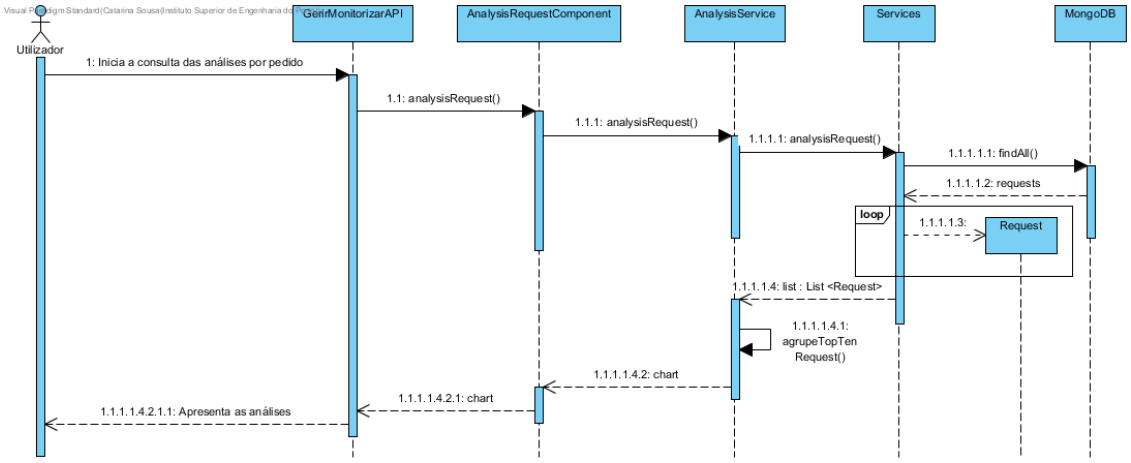


Figura 33 - Diagrama de sequência da consulta de análises por pedidos

Na Figura 33 é possível observar que o utilizador inicia a consulta das análises por pedido. A componente **GerirMonitorizarAPI** chama o método analysisRequest da

[2] Os dados da figura não são reais, por razões de segurança

**AnalysisRequestComponent**, que consequentemente chama o método com o mesmo nome do **AnalysisService** e da componente de *backend* **Services**. Esta componente chama o método findAll que retorna os pedidos da componente de base de dados **MongoDB** para os Services. Estes criam os objetos do tipo request e retorna-os numa lista para o AnalysisService, que chama o método agrupeTopTenRequest e retorna a lista para o AnalysisRequestComponent e o GerirMonitorizarAPI. E por fim a interface apresenta as análises.

Semelhante à página de consulta de análises por pedido, a consulta de análises por cliente pode ser acedida por um utilizador ou admin. Para esta página foi utilizada a componente **AnalysisClientComponent**, contém a área branca, e a biblioteca **CanvasJS**[94], para o gráfico. A Figura 34 exibe a página.

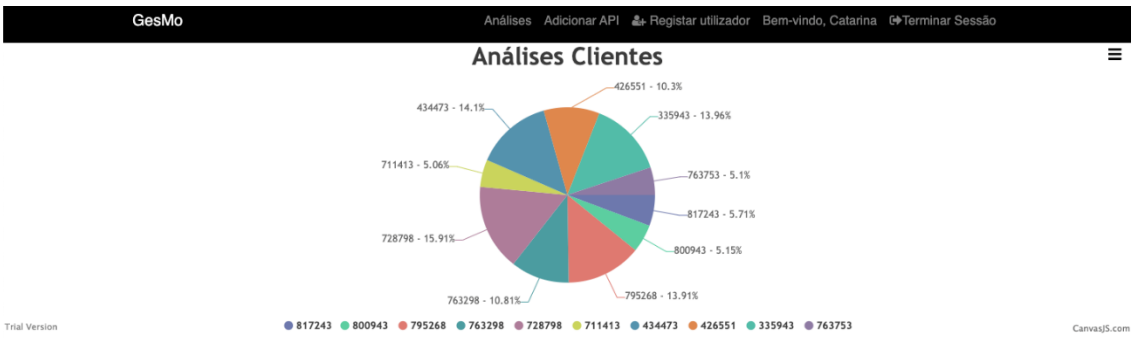


Figura 34 – Página de consulta de análises por clientes[3]

A interação das componentes para a análises por clientes, está apresentada na Figura 35.

[3] Os dados da figura não são reais, por razões de segurança

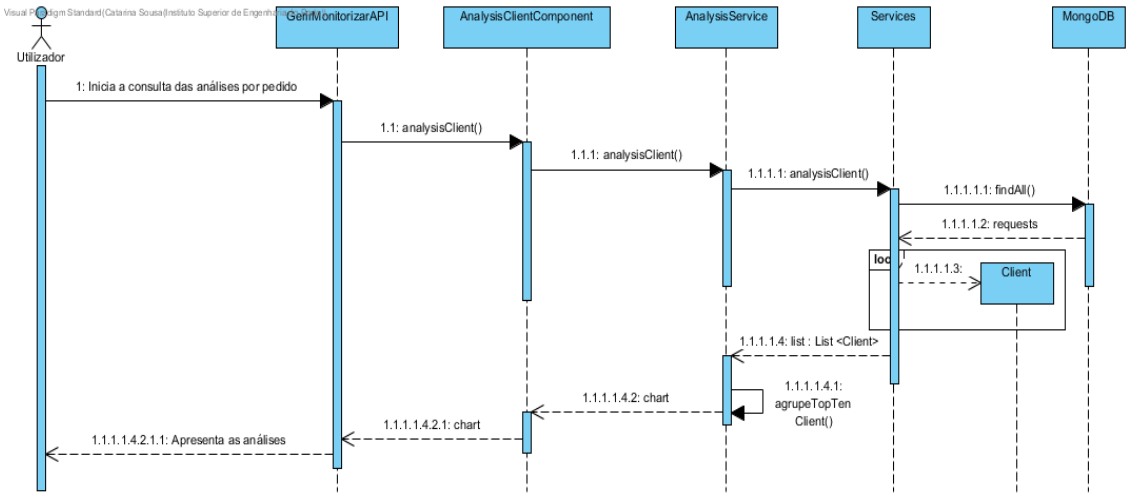


Figura 35 - Diagrama de sequência da consulta de análises por cliente

Através da Figura 35 é possível observar que o utilizador inicia a consulta das análises por cliente. A interface (**GerirMonitorizarAPI)** chama o método analysisClient da **AnalysisClientComponent**, que chama o método com o mesmo nome do **AnalysisService** e da componente de *backend* **Services**. Esta componente chama o método findAll que retorna os pedidos da componente de base de dados **MongoDB** para os Services. Estes criam os objetos do tipo client e retorna-os numa lista para o AnalysisService, que chama o método agrupeTopTenClient e retorna a lista para o AnalysisClientComponent e o GerirMonitorizarAPI. E por último a interface apresenta as análises.

Para utilizadores com o papel de *admin* é lhes permitido aceder à página de registo de API. A componente **AddApiComponent** foi usada para a área branca. A Figura 36 mostra a página em questão.


Figura 36 - Registar API

A interação das componentes para a criação de API, é retratada no diagrama da Figura 37.

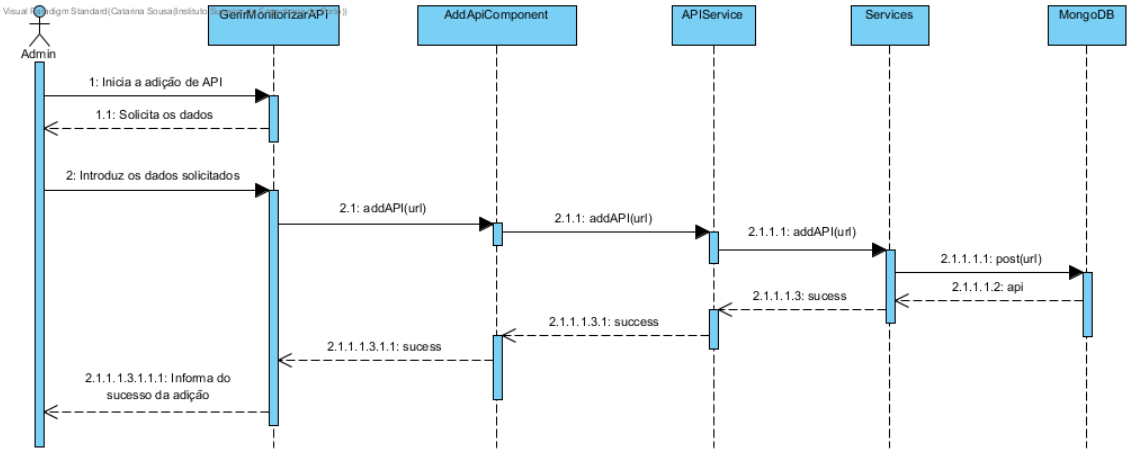


Figura 37 - Diagrama de sequência da criação de API

Na Figura 37 é possível observar que o admin inicia a criação de API e a componente **GerirMonitorizarAPI** solicita os dados. O admin introduz os dados solicitados e a interface chama o método addAPI com o parâmetro url da **AddAPIComponent**, que consequentemente chama o método com o mesmo nome e o mesmo parâmetro do **APIService** e da componente de *backend* **Services**. Esta componente chama o método post com os parâmetros mencionados no addAPI que retorna a api da componente **MongoDB** para os Services. Estes numa situação em que foi tudo validado e retorna sucesso, passa a mensagem de volta para o APIService, AddAPIComponent e o GerirMonitorizarAPI. E por fim a interface informa do sucesso da criação da API ao admin.

O utilizador pode aceder à página de criação de teste. Para essa página foi utilizada a componente **CreateTestComponent**, representada na área branca. A Figura 38 mostra a página em questão.


Figura 38 – Página de criação de teste

A comunicação das componentes para a criação de teste, está apresentada na Figura 39.

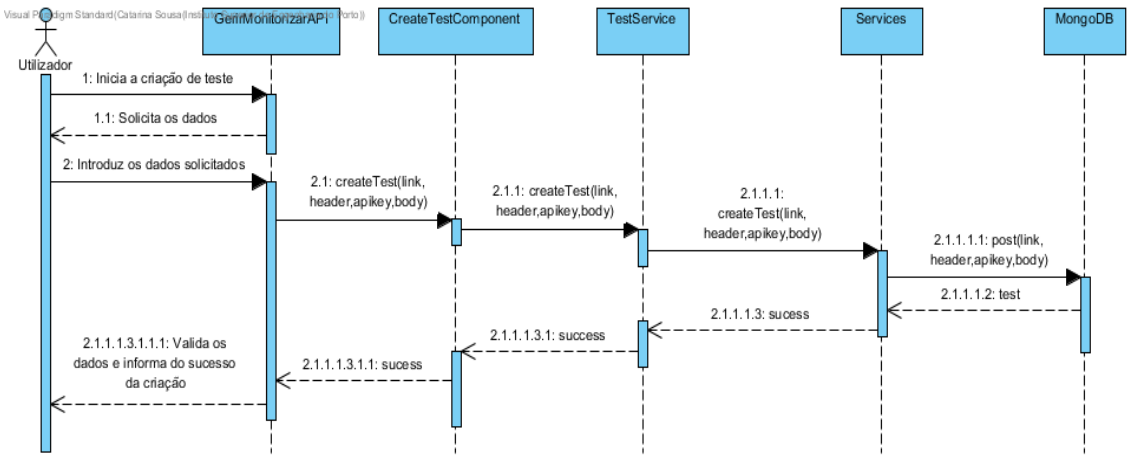


Figura 39 - Diagrama de sequência da criação de teste

Na Figura 39 é possível observar que o utilizador inicia a criação de teste e a componente **GerirMonitorizarAPI** solicita os dados. O utilizador introduz os dados solicitados e a interface chama o método createTest com os parâmetros link, header, apikey e body da **CreateTestComponent**, que consequentemente chama o método com o mesmo nome e os mesmos parâmetros do **TestService** e da componente **Services**. Esta componente chama o método post com os parâmetros mencionados no addAPI

que retorna o teste da componente **MongoDB** para os Services. Estes numa situação em que foi tudo validado e retorna sucesso, passa a mensagem de volta para o TestService, CreateTestComponent e o GerirMonitorizarAPI. E por último a interface informa do sucesso da criação ao utilizador.

Para além da página de criação de testes, o utilizador pode também consultar a listagem de testes criados. A componente **ListTestComponent** foi utilizada para a parte branca. A Figura 40 apresenta a página de listagem.

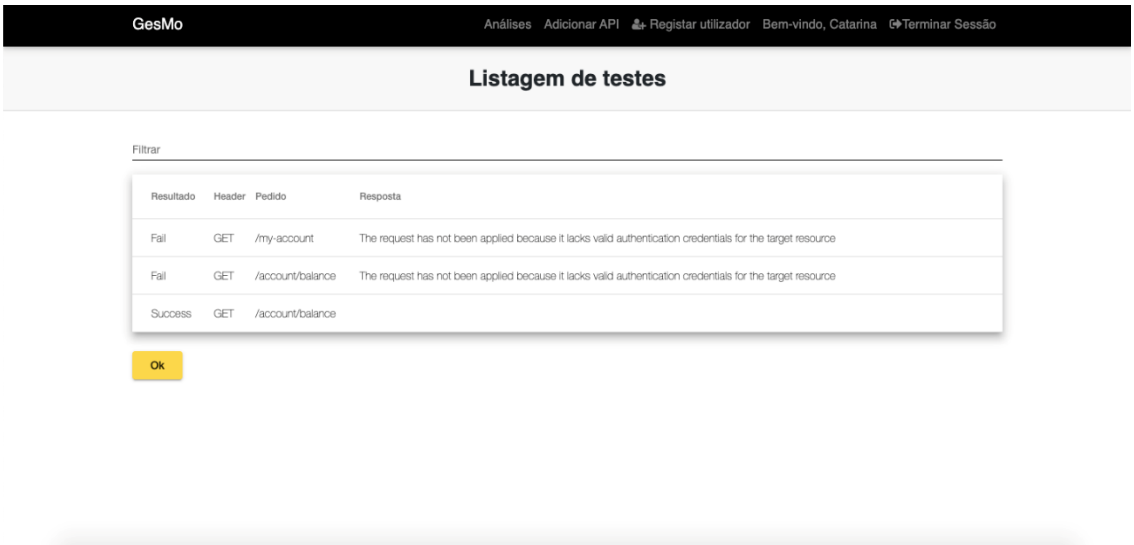


Figura 40 – Página de listagem de testes

A interação das componentes para a listagem de testes, é retratada no diagrama da Figura 41.

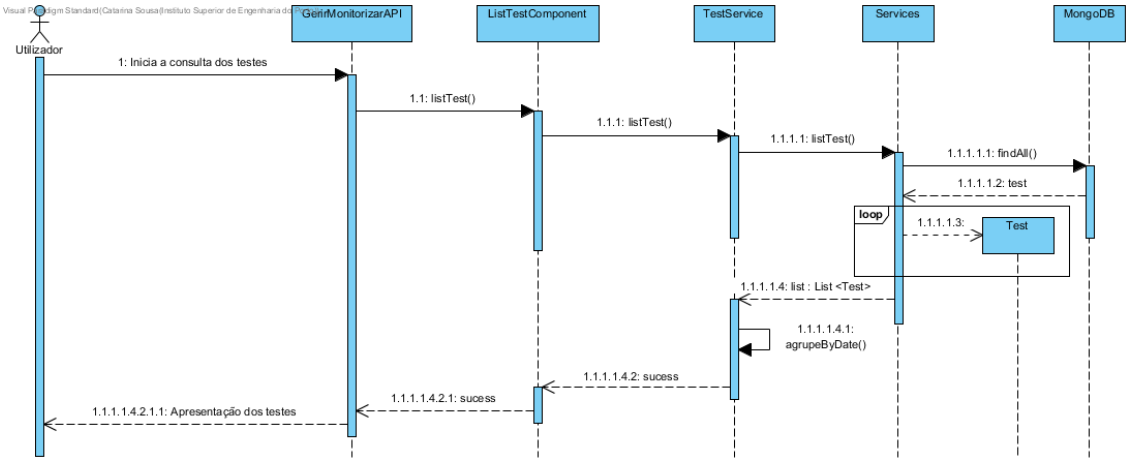


Figura 41 - Diagrama de sequência da listagem de testes

Através da Figura 41 é possível observar que o utilizador inicia a listagem dos testes. A componente (**GerirMonitorizarAPI)** chama o método listTest da **ListTestComponent**, que chama o método com o mesmo nome do **TestService** e da componente de *backend* **Services**. Esta componente chama o método findAll que retorna os testes da componente **MongoDB** para os Services. Estes criam os objetos do tipo test e retorna-

os numa lista para o TestService, que chama o método agrupeByDate e retorna a lista para o ListTestComponent e o GerirMonitorizarAPI. E por último a interface apresenta os testes.

## 6.3 Serviços

A componente Serviços consiste numa API REST, implementada na linguagem de programação Node.js. A comunicação entre esta componente e a interface gráfica é feita pelo protocolo Hypertext Transfer Protocol (HTTP), como referido na secção 5.3. Desta forma é possível que o cliente (interface gráfica) possa fazer pedidos.

Relativamente aos serviços de autenticação foram criados dois métodos POST um com recurso /Auth/SignUp e outro com /Auth/Login, como mencionado na secção 5.5. Para a criação de utilizador a informação necessária é apresentada na Tabela 28.

Tabela 27 - Dados para criação de utilizador

| Atributo | Descrição | Tipo de dados | Caraterísticas extra |
|---|---|---|---|
| **name** | Nome do utilizador | String | |
| **password** | Password do utilizador | String | |
| **email** | Email do utilizador | String | unique |
| **admin** | Definição de papel de admin | Boolean | default: false |
| **userType** | Papel do utilizador | String | |

Para o atributo password é utilizada a biblioteca **bcrypt**[95], que permite encriptar informações utilizando o método *hash*, satisfazendo desta forma um requisito não funcional abordado na secção 5.1.2.

No que diz respeito ao login de um utilizador, é necessário disponibilizar informação referente ao email e a password.

Em relação aos serviços de interfaces de programação foram usados os métodos POST e GET com o recurso /APInterface, referidos na secção 5.5. Para a criação de uma API indispensável disponibilizar os dados listados na Tabela 29.

Tabela 28 - Dados para a criação de API

| Atributo | Descrição | Tipo de dados | Caraterísticas extra |
|---|---|---|---|
| **Link** | Link da API | String | unique |

No que diz respeito a serviços de testes à API foram criados os métodos POST e GET com o recurso /Test, detalhados na secção 5.5. Para efetuar a criação de um teste é preciso disponibilizar os dados apresentados na Tabela 30.

Tabela 29 -Dados para a criação de teste

| Atributo | Descrição | Tipo de dados | Caraterísticas extra |
|---|---|---|---|
| **header** | Header do pedido do teste | String | enum:['GET','POST', 'PUT', 'DELETE']<br>default: 'GET' |
| **link** | Recurso do pedido do teste | String | |
| **api** | API do teste | APInterface | required |
| **apiKey** | Apikey de teste | String | required |
| **body** | Body do teste | String | |
| **result** | Resultado do teste | String | enum:['Sucess','Fail']<br>default: 'Fail' |
| **resultMessage** | Mensagem retornada do teste | String | |
| **error** | Código da resposta de erro do teste | String | |
| **errorMessage** | Mensagem de erro retornada do teste | String | |
| **creationDate** | Data de criação do teste | Date | default: Date.now |

Para obter os valores do resultado do teste (result, resultMessage, error e errorMessage) é utilizado o modulo **util.promisify**[96], que permite executar comandos e, consequentemente, efetuar testes aos pedidos da API.

E por último para os serviços de análises foram criados dois métodos GET, um com o recurso /AnalysisPerClient e /AnalysisPerRequest.

## 6.4 Pipeline de SDK

Um dos objetivos deste projeto era a automatização e geração de SDK, referido na secção 1.3. Para a sua realização foi criada uma pipeline no *Jenkins* por *Jenkinsfile*, intitulada SDK Configs, com 7 *stages*. A Figura 42 apresenta, de forma esquemática, como é executada a *pipeline*.

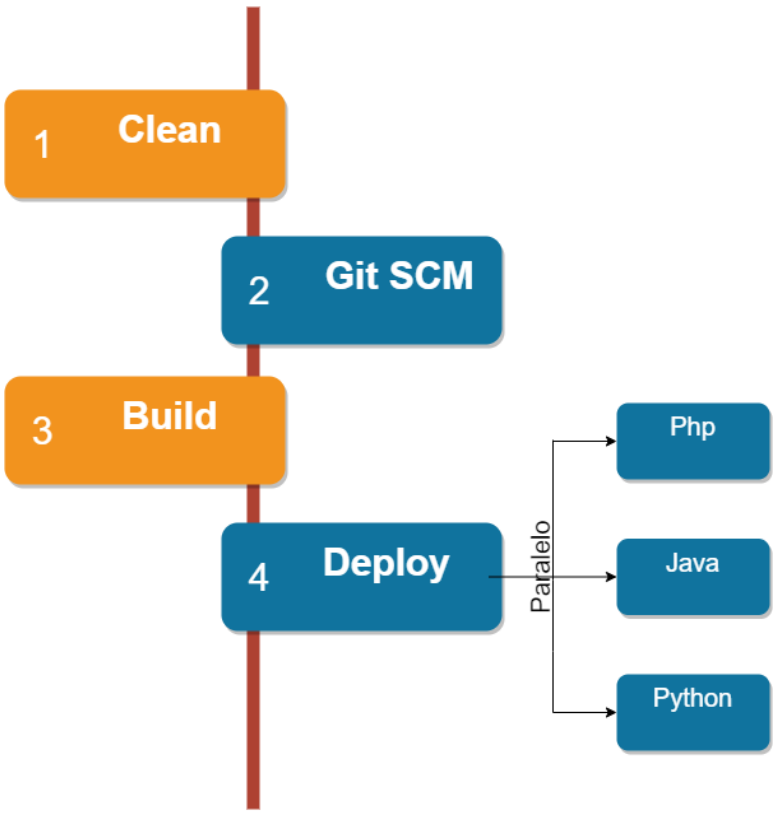


Figura 42 - Pipeline SDK Configs

O primeiro *stage* a ser realizado é o **Clean**. Este tem como função eliminar os artefactos guardados de execuções prévias da pipeline.

De seguida é executado o *stage* **Git SCM**. A finalidade deste é clonar o repositório do GitLab, onde se encontra o ficheiro Jenkinsfile com as instruções para a execução do SDK Configs, e faz atualização dos *submodules* do projeto.

Em terceiro é efetuado o **Build**, que vai buscar a última versão disponível no repositório e executa um script na linguagem de programação php. Este vai buscar os ficheiros com extensão ".json" e que o nome começa por config (por exemplo configJava.json), retira do nome o 'change' e troca a extensão para .build.

Por último, o *stage* **Deploy**, que executa 3 *stages* em paralelo, ou seja, executam de forma independente (se um encontrar um erro e parar a execução os outros não param). Os 3 (**Java**, **Php** e **Python**) efetuam os mesmo comandos, mas para a geração, de SDK, da linguagem de programação referente ao nome. Todos eles utilizam a ferramenta de geração de SDK, *OpenApi Generator*, referida na secção 3.1.1.

Os *stages* de geração iniciam com a verificação da existência de um ficheiro com o nome da linguagem de programação e extensão ".build" (resultante do Build). De seguida é lido o ficheiro ".json" do repositório(por exemplo configJava.json), que contém as caraterísticas da geração(como versão e nome da pasta gerada). Concluído isto vai ao repositório publico do SDK no GitHub, gera a pasta do SDK e envia as alterações (*commit*) com a tag da versão. E por fim, corre um script que gera um ficheiro readme com as atualizações feitas com a nova versão.

É importante denotar que houve necessidade de contribuir para o projeto *open-source*(permite que qualquer pessoa colabore para o desenvolvimento), do GitHub, do OpenApi Generator [62]. Devido ao facto que no gerador de Java não existia suporte para *oneOf* (especificação da API). Até ao momento aguarda-se pela aceitação da alteração (*merge request*).

# Capítulo 7

# Experimentação e Avaliação

Concluída a implementação, é necessário avaliar a solução como resolução do problema, referido na secção 1.2. Muitas vezes a solução encontrada para um problema pode não ser a que melhor o resolve, para isso é fundamental definir um conjunto de métricas pelas quais se possa analisar. Neste capítulo irão ser abordadas as experiências, os testes realizados e os seus resultados.

## 7.1 Experiências e Testes

Com esta secção pretende-se expor as grandezas para a avaliar os testes, as hipóteses e as metodologias de avaliação.

### 7.1.1 Grandezas a avaliar

O tipo de problemas identificados, na secção 1.2, são referentes a qualidade de *software* e usabilidade. Por esse facto as grandezas definidas foram os resultados dos testes de software e testes de usabilidade. Com esses dados é possível validar o cumprimento dos objetivos.

### 7.1.2 Hipóteses

Com a definição das grandezas, é fundamental indicar as hipóteses correspondentes.

De forma a validar a correção dos problemas de qualidade de *software*, definiu-se a hipótese**: se houver 100% de sucesso nos testes de *software* então é garantida a qualidade do código da solução, a fácil manutenção e a aplicação das boas práticas no desenvolvimento de *software***.

E para validar os problemas de usabilidade estabeleceu-se a hipótese: **se houver pelo menos 60% de satisfação dos utilizadores** então foram cumpridos os requisitos definidos.

### 7.1.3 Metodologias de avaliação

As metodologias de avaliação consistem nos métodos utilizados para avaliar projetos. Consoante as hipóteses e as grandezas, as metodologias variam. Dessa forma é essencial escolher as mais indicadas para cada situação.

A qualidade da implementação da solução é conferida por meio de testes de software, particularmente por teste unitários e de integração. Os testes unitários consistem em testes efetuados a unidades do código (funções, classes, entre outras) [97]. Desta forma é fácil identificar e resolver problemas. E os testes de integração têm como objetivo testar combinações de unidades de código [97]. Com isto é possível garantir a confiabilidade e a performance, satisfazendo desta forma um requisito não funcional abordado na secção 5.1.2. Ambos os métodos de testagem permitem garantir que quando são feitas alterações de código a solução continua a funcionar [98].

No que diz respeito à usabilidade, foram efetuados os testes alfa. Estes consistem na disponibilização de uma versão quase final da solução, a um grupo de pessoas. O grupo oferece a sua opinião sobre a situação da solução [99]. Deste modo é possível identificar problemas, que não tinham sido reconhecidos previamente e corrigir.

## 7.2 Resultados

Nesta secção irão ser apresentados os resultados obtidos pela implementação das metodologias de avaliação, nomeadamente pelos testes de software e testes de usabilidade.

### 7.2.1 Testes de software

Para os testes de software foi usada a *framework* Jasmine, a *framework* Mocha e a ferramenta Karma.

Na interface gráfica é utilizada a linguagem de programação Angular, referido na secção 6.2. O Angular Cli (linha de comandos) permite fazer download e instalar a *framework* Jasmine [97][98] que possibilita a criação de testes. Em conjunto com o Karma [102], que é uma ferramenta que proporciona um ambiente de testagem, permite obter um relatório sobre os testes.

As componentes criadas para a interface têm associado um ficheiro com extensão ".spec.ts" com os testes. No Jasmine, cada teste contém a sua definição, os valores de entrada e os valores de saída. Todos os testes seguem a mesma estrutura: criam a componente, importam as dependências no método *beforeEach*, instanciam e testam a componente. Desta forma é possível verificar se a componente está a funcionar como é pretendido. O relatório sobre os testes encontra-se na Figura 43.

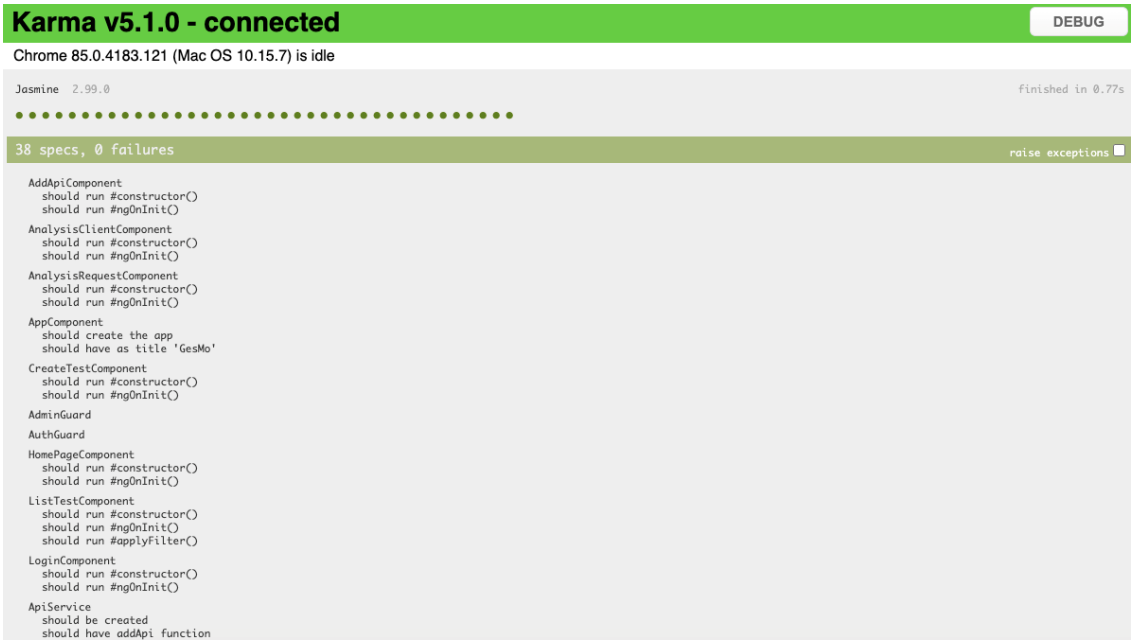


Figura 43 – Relatório dos testes

Como é possível observar pela Figura 43 foram feitos testes a todas as componentes e todos eles foram executados com sucesso. Por isso conclui-se que as componentes da interface gráfica estão a funcionar da forma esperada.

Relativamente aos serviços, estes utilizam a *framework* Node.js, referido na secção 6.3. Neste caso foi utilizada a *framework*, de desenvolvimento de testes, Mocha [103]. Os resultados da execução dos testes são apresentados pela linha de comandos. A Figura 44 exibe os resultados na linha de comandos.

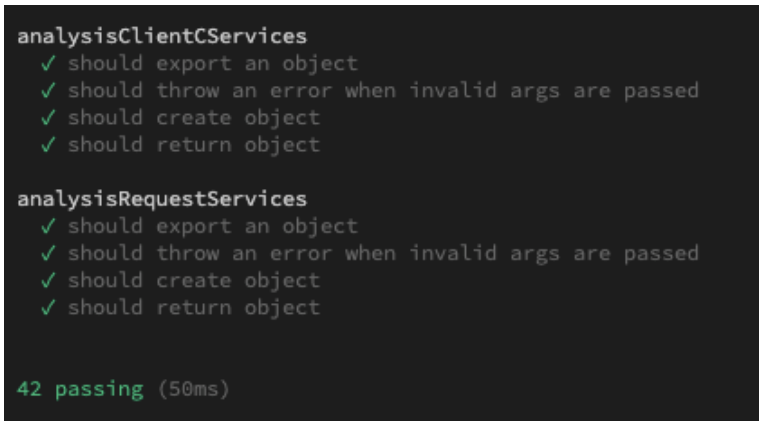


Figura 44 – Resultados dos aos Serviços

Na Figura 44 é possível observar que todos os testes foram executados com sucesso.

Os resultados obtidos pela execução dos testes, à interface gráfica e aos serviços, permitem afirmar que se conseguiu alcançar 100% de sucesso nos testes de *software*.

Comprovando assim a hipótese relativa à qualidade de software, mencionada na secção 7.1.2.

### 7.2.2 Testes de usabilidade

No que se refere às questões de usabilidade da solução, foram realizados os testes alfa. Estes consistem em disponibilizar uma versão quase final da solução, a um grupo de pessoas, e pedir *feedback*. As pessoas que formam o grupo são as que irão utilizar a solução quando estiver concluída. Neste caso os utilizadores finais da solução são os programadores da empresa E-goi, como mencionado na secção 1.2.

Para os testes alfa, a amostra consistiu em 4 utilizadores com experiência na área de informática entre 1 a 20 anos e com idades entre os 23 aos 40.

O *feedback* foi bastante positivo. Na generalidade as respostas obtidas foi que a plataforma, é simples de utilizar e bastante clara na apresentação da informação, e a pipeline está em concordância com o que foi pedido. Uma critica negativa apresentada foi o facto de as análises corresponderem apenas para a API pública, uma vez que no futuro seria desejável também fazer à API interna da empresa. Contudo isso era uma funcionalidade extra e algo que irá ser feito no futuro. Outra critica foi também o facto de no repositório dos SDK não existir um *changelog*, ou seja, um ficheiro com a informação das correções e adições da última versão do projeto. Algo que foi então implementado, após o *feedback*, por um *script*, mencionado na secção 6.4.

Para além do feedback foi pedido ao grupo para avaliarem a solução, de 0 a 10, e obteve-se os seguintes resultados: **7, 8, 6 e 7**. Alcançando assim 70% de satisfação dos utilizadores, com a solução encontrada. E consequentemente comprovando a hipótese relativa à usabilidade, mencionada na secção 7.1.2.

# Capítulo 8

# Conclusão

A metodologia seguida para este projeto, mencionada na secção 1.5, apresenta as seguintes etapas: Análise de requisitos, Análise, Design da solução, Desenvolvimento e Testes. Com a finalização de todas estas etapas, é possível perceber o êxito do projeto. Neste capítulo vai-se fazer um balanço de todo o projeto enunciando os objetivos atingidos e o trabalho que deve ser realizado no futuro.

## 8.1 Objetivos alcançados

Na secção 1.3. foram apresentados os objetivos que deveriam ser alcançados para o sucesso do projeto. Estes foram os seguintes:

1º) Análise da API publica da empresa E-goi, juntamente com a estrutura tecnológica

2º) Definição da solução e levantamento de critérios

3º) Experimentação de ferramentas de gestão e monitorização da API e geração de SDK

4º) Definição do design da solução

5º) Implementação da solução e seleção da ferramenta de geração de SDK

6º) Definição de testes e avaliação da solução

O primeiro e segundo objetivos foram alcançados nos capítulos 3 e 4. Através da análise da API publica e da estrutura tecnológica da empresa E-goi, foi possível encontrar os critérios essenciais para a solução, mencionados na secção 3.1. Estes são: elaborar análises feitas por pedido; permitir a monitorização e a gestão de várias API; geração de SDK com a respetiva documentação; análises por cliente; implementar monitorização básica. A definição da solução foi conseguida pela análise de valor do capítulo 4 que, através do *New Concept Development* (NCD), concluiu que o conceito da solução seria uma ferramenta, que utiliza um gerador *open-source* de SDK (OpenAPI Generator) e as tecnologias da empresa (como jenkins e mongoDB).

O terceiro objetivo, foi atingido também no capítulo 3. Foram experimentadas enumeras ferramentas de gestão e monitorização de API e geração de SDK. Contudo as mais importantes de mencionar são o OpenAPI Generator (secção 3.1.1.), APIMatic (secção 3.1.2.) e o Assertible (secção 3.1.3). Nenhuma destas satisfez todos requisitos

necessários para constituírem solução para o problema, como mencionado na secção 1.2.

No que diz respeito ao quarto objetivo, este foi conseguido no capítulo 5. Este apresenta todo o design da solução como os requisitos funcionais e não funcionais, o domínio, a arquitetura, quantas máquinas foram usadas e os serviços criados.

O quinto objetivo atingiu-se no capítulo 6. Este aborda as metodologias utilizadas no processo de implementação e detalhou-se a interface gráfica, os serviços e a pipeline de SDK. Também é mencionada a ferramenta *open-source* de geração de SDK usada, o OpenAPI Generator, e que foi necessário contribuir para o seu projeto.

O sexto e último objetivo conseguiu-se no capítulo 7. Neste são apresentadas as métricas utilizadas para a avaliar a solução, testes de software e usabilidade. Ambos atingiram os resultados pretendidos, validando as hipóteses.

Apesar de a solução ter alcançado todos os objetivos e ter passado nas hipóteses dos testes, não significa que esta tenha resolvido o problema. A solução apenas identifica os erros e não os corrige, será necessária intervenção humana para colmatar essa necessidade. Tal facto, só permite refletir resultados reais passado um longo período de tempo.

## 8.2 Trabalho Futuro

Independentemente de o projeto ter sido bem-sucedido, ainda é necessário trabalho para ficar concluído.

Um dos principais pontos, ainda a implementar, seria as análises para clientes e pedidos também poderem ser referentes à API privada da empresa, mencionado na secção 7.2.2. Apesar de não existir necessidade imediata, seria o próximo passo no projeto.

Como referido na secção 8.1, a ferramenta necessita de um uso perlongado para serem conseguidos resultados na API publica da empresa E-goi. E com este uso também obter melhor feedback dos utilizadores e colmatar problemas que irão aparecer com o tempo. Com este trabalho a solução ganhará mais valor e irá salvaguardar a sua manutenção.

# Referências


[1] C. Elements, «Study: 55 Percent Say API Integration Is “Critical” to Business Strategy», *GlobeNewswire News Room*, Ago. 04, 2019. http://www.globenewswire.com/news-release/2019/04/08/1798931/0/en/Study-55-Percent-Say-API-Integration-Is-Critical-to-Business-Strategy.html (acedido Jan. 11, 2020).

[2] «The seven make-or-break API challenges CIOs need to address | McKinsey». https://www.mckinsey.com/business-functions/mckinsey-digital/our-insights/the-seven-make-or-break-api-challenges-cios-need-to-address (acedido Jan. 11, 2020).

[3] «O que é uma API?» https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces (acedido Jan. 11, 2020).

[4] «Sabe o que é uma API (Application Programming Interface)?» https://pplware.sapo.pt/high-tech/sabe-o-que-e-uma-api-application-programming-interface/ (acedido Jan. 11, 2020).

[5] www.raffcom.com.br, «O que é UI - User Interface? Veja aqui- Blog Raffcom», *Raffcom - Agência Full Service*, Jun. 27, 2016. https://www.raffcom.com.br/blog/o-que-e-ui/ (acedido Jan. 25, 2020).

[6] «What is an API Integration? (for Non-Technical People)». https://tray.io/blog/what-is-an-api-integration-for-non-technical-people (acedido Jan. 25, 2020).

[7] P. Sagdeo, «Application Programming Interfaces and the Standardization-Value Appropriation Problem», vol. 32, n. 1, p. 29.

[8] «What Is API Monitoring | API Basics | SmartBear». https://smartbear.com/solutions/api-monitoring/ (acedido Jan. 11, 2020).

[9] «What is API management?» https://www.redhat.com/en/topics/api/what-is-api-management (acedido Jan. 11, 2020).

[10] «What is API Management? | MuleSoft». https://www.mulesoft.com/resources/api/what-is-api-management (acedido Jan. 11, 2020).

[11] «Marketing no Facebook: Integre com Marketing Automation», *E-goi*. https://www.e-goi.com/pt/marketing-facebook/ (acedido Jan. 18, 2020).

[12] «Teste de Software: Introdução, Conceitos Básicos e Tipos de Teste», *Blog One Day Testing - Powered by Sofist*, Mai. 10, 2015. https://blog.onedaytesting.com.br/teste-de-software/ (acedido Fev. 14, 2020).

[13] Andrew Powell-Morse, «Waterfall Model: What Is It and When Should You Use It?», *Airbrake Blog*, Dez. 08, 2016. https://airbrake.io/blog/sdlc/waterfall-model (acedido Jan. 11, 2020).

[14] «Scrum - framework de desenvolvimento de produtos, abordagem ágil». http://www.scrumportugal.pt/scrum/ (acedido Jan. 11, 2020).

[15] S. Balaji, «WATEERFALLVs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC», . *Vol.*, n. 1, p. 6, 2012.

[16] D. P. Lacerda, A. Dresch, A. Proença, e J. A. V. Antunes Júnior, «Design Science Research: método de pesquisa para a engenharia de produção», *Gest. Prod.*, vol. 20, n. 4, pp. 741–761, Nov. 2013, doi: 10.1590/S0104-530X2013005000014.

[17] «What is API: Definition, Types, Specifications, Documentation», *AltexSoft*. https://www.altexsoft.com/blog/engineering/what-is-api-definition-types-specifications-documentation/ (acedido Mai. 17, 2020).

[18] B. De, *API Management*. Berkeley, CA: Apress, 2017.

[19] «10 Most Popular Database APIs | ProgrammableWeb». https://www.programmableweb.com/news/10-most-popular-database-apis/brief/2019/05/06 (acedido Mai. 18, 2020).

[20] «Windows API Index - Win32 apps | Microsoft Docs». https://docs.microsoft.com/en-us/windows/win32/apiindex/windows-api-list (acedido Mai. 18, 2020).

[21] «Java JDBC API». https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/ (acedido Mai. 18, 2020).

[22] «Simple Object Access Protocol Overview». https://docs.oracle.com/cd/A97335_02/integrate.102/a90297/overview.htm (acedido Out. 15, 2020).

[23] «What is REST – Learn to create timeless REST APIs». https://restfulapi.net/ (acedido Mai. 23, 2020).

[24] «What is Remote Procedure Call (RPC)?» https://searchapparchitecture.techtarget.com/definition/Remote-Procedure-Call-RPC (acedido Out. 15, 2020).

[25] «GraphQL: A query language for APIs.» http://graphql.org/ (acedido Out. 15, 2020).

[26] «[MS-WUSP]: Glossary | Microsoft Docs». https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-wusp/5daaa9d9-26aa-42fc-a431-c011166dc58f (acedido Mai. 22, 2020).

[27] «Nordic APIs | Cases When Using SOAP Makes Sense». https://nordicapis.com/common-cases-when-using-soap-makes-sense/ (acedido Mai. 22, 2020).

[28] «SOAP vs. REST: A Look at Two Different API Styles». https://www.upwork.com/hiring/development/soap-vs-rest-comparing-two-apis/ (acedido Mai. 22, 2020).

[29] «O que é ACID? - Opensanca - Medium». https://medium.com/opensanca/o-que-%C3%A9-acid-59b11a81e2c6 (acedido Mai. 22, 2020).

[30] «IBM Knowledge Center». https://www.ibm.com/support/knowledgecenter/SSTTDS_11.0.0/com.ibm.etools.mft.doc/ac55630_.html (acedido Mai. 22, 2020).

[31] «What is SSL, TLS and HTTPS? | DigiCert». https://www.websecurity.digicert.com/security-topics/what-is-ssl-tls-https (acedido Mai. 22, 2020).

[32] «REST API Architectural Constraints», *GeeksforGeeks*, Dez. 09, 2018. https://www.geeksforgeeks.org/rest-api-architectural-constraints/ (acedido Mai. 22, 2020).

[33] «Client-server architecture | computer science», *Encyclopedia Britannica*. https://www.britannica.com/technology/client-server-architecture (acedido Mai. 23, 2020).

[34] «What is REST?» https://www.restapitutorial.com/lessons/whatisrest.html# (acedido Mai. 23, 2020).

[35] «REST – Statelessness – REST API Tutorial». https://restfulapi.net/statelessness/ (acedido Mai. 23, 2020).

[36] «REST Principles and Architectural Constraints – REST API Tutorial». https://restfulapi.net/rest-architectural-constraints/ (acedido Mai. 23, 2020).

[37] «URI», *MDN Web Docs*. https://developer.mozilla.org/pt-BR/docs/Glossario/URI (acedido Mai. 23, 2020).

[38] «Understanding RPC, REST and GraphQL». https://apisyouwonthate.com/blog/understanding-rpc-rest-and-graphql (acedido Mai. 23, 2020).

[39] «Home | AMQP». https://www.amqp.org/ (acedido Out. 15, 2020).

[40] «Apigility». https://apigility.org/documentation/api-primer/what-is-an-api (acedido Mai. 23, 2020).

[41] «Do you really know why you prefer REST over RPC? | API Handyman». http://apihandyman.io/do-you-really-know-why-you-prefer-rest-over-rpc/ (acedido Mai. 23, 2020).

[42] «GraphQL: Core Features, Architecture, Pros and Cons | AltexSoft». https://www.altexsoft.com/blog/engineering/graphql-core-features-architecture-pros-and-cons/ (acedido Mai. 24, 2020).

[43] «O que é autenticação?» https://www.tecmundo.com.br/seguranca/1971-o-que-e-autenticacao-.htm (acedido Mai. 29, 2020).

[44] «What is authorization? - Definition from WhatIs.com». https://searchsoftwarequality.techtarget.com/definition/authorization (acedido Mai. 29, 2020).

[45] «What is Role-Based Access Control (RBAC)? Examples, Benefits, and More», *Digital Guardian*, Ago. 20, 2018. https://digitalguardian.com/blog/what-role-based-access-control-rbac-examples-benefits-and-more (acedido Mai. 29, 2020).

[46] «API Security Best Practices MegaGuide». https://expeditedsecurity.com/api-security-best-practices-megaguide/#abac (acedido Mai. 29, 2020).

[47] «What is HTTPS? | Cloudflare». https://www.cloudflare.com/learning/ssl/what-is-https/ (acedido Mai. 29, 2020).

[48] «What Is SSL (Secure Sockets Layer)? | DigiCert.com», *DigiCert*. /ssl/ (acedido Mai. 29, 2020).

[49] «API Security Best Practices MegaGuide». https://expeditedsecurity.com/api-security-best-practices-megaguide/#secure-api-design (acedido Mai. 29, 2020).

[50] «O que é front-end e back-end? - Blog da TreinaWeb». https://www.treinaweb.com.br/blog/o-que-e-front-end-e-back-end/ (acedido Jun. 01, 2020).

[51] «Sabe o que é uma API (Application Programming Interface)?», *Pplware*, Fev. 09, 2019. https://pplware.sapo.pt/high-tech/sabe-o-que-e-uma-api-application-programming-interface/ (acedido Mai. 13, 2020).

[52] «5 benefits of API for developers | Application programming interface», *InsightsSuccess*, Nov. 15, 2019. https://www.insightssuccess.com/5-benefits-of-api-for-developers-application-programming-interface/ (acedido Mai. 13, 2020).

[53] «Advantages and Disadvantages of API for Business», *OpenVPN*, Jun. 04, 2019. https://openvpn.net/advantages-and-disadvantages-of-api-for-business/ (acedido Mai. 13, 2020).

[54] «What is API Monitoring? | Uptrends Glossary». https://www.uptrends.com/what-is/api-monitoring (acedido Jan. 19, 2020).

[55] «What is an SDK and an API?» https://www.skyhook.com/blog/what-is-an-sdk-and-an-api (acedido Nov. 06, 2019).

[56] «Funcionalidades», *E-goi*. https://www.e-goi.com/pt/funcionalidades/ (acedido Jan. 31, 2020).

[57] «Goidini». https://goidini.e-goi.com/index/home (acedido Jan. 31, 2020).

[58] «Shopify + E-goi: Email, SMS e Marketing Automation», *E-goi*. https://www.e-goi.com/pt/shopify/ (acedido Jan. 31, 2020).

[59] «The most popular database for modern apps», *MongoDB*. https://www.mongodb.com (acedido Fev. 02, 2020).

[60] «What is Jenkins? Continuous Integration (CI) Tool». https://www.guru99.com/jenkin-continuous-integration.html (acedido Fev. 02, 2020).

[61] «About SmartBear | Who is SmartBear Software?» https://smartbear.com/company/about-us/ (acedido Fev. 03, 2020).

[62] «OpenAPITools/openapi-generator», *GitHub*. https://github.com/OpenAPITools/openapi-generator (acedido Nov. 06, 2019).

[63] «Using Templates · OpenAPI Generator». https://openapi-generator.tech/ (acedido Nov. 06, 2019).

[64] «OpenAPI Generator · Generate clients, servers, and documentation from OpenAPI 2.0/3.x documents». https://openapi-generator.tech/docs/generators/java (acedido Out. 08, 2020).

[65] «About». https://www.apimatic.io/about/ (acedido Fev. 09, 2020).

[66] V. Schreibmann e P. Braun, «Model-driven Development of RESTful APIs»:, em *Proceedings of the 11th International Conference on Web Information Systems and Technologies*, Lisbon, Portugal, 2015, pp. 5–14, doi: 10.5220/0005411200050014.

[67] «15 APIMATIC Customer Reviews & References | FeaturedCustomers». https://www.featuredcustomers.com/vendor/apimatic (acedido Fev. 02, 2020).

[68] «Pricing». https://www.apimatic.io/pricing/ (acedido Fev. 14, 2020).

[69] Assertible, «About Us», *Assertible*. https://assertible.com/about (acedido Fev. 10, 2020).

[70] «Creating collections», *Postman Learning Center*. https://learning.getpostman.com (acedido Fev. 10, 2020).

[71] «O Que é Comando Curl e Como Usar?», *Hostinger Tutoriais*, Mar. 13, 2019. https://www.hostinger.com.br/tutoriais/comando-curl-linux/ (acedido Fev. 10, 2020).

[72] «Plans : Assertible». https://assertible.com/plans (acedido Fev. 14, 2020).

[73] «Conceito de Análise de Valor e sua Importância na Gestão de Projetos - JRM Coaching». https://www.jrmcoaching.com.br/blog/conceito-de-analise-de-valor-e-sua-importancia-na-gestao-de-projetos/ (acedido Fev. 16, 2020).

[74] P. Koen *et al.*, *Providing clarity and a common language to the "fuzzy front end". Research Technology Management 44: 46 – 55*. 2001.

[75] «Figure 13. The New Concept Development (NCD)-model. (Koen et al., 2001)», *ResearchGate*. https://www.researchgate.net/figure/The-New-Concept-Development-NCD-model-Koen-et-al-2001_fig12_324208591 (acedido Fev. 16, 2020).

[76] K. Dewulf, «Sustainable Product Innovation: The Importance of the Front- End Stage in the Innovation Process», *Advances in Industrial Design Engineering*, Mar. 2013, doi: 10.5772/52461.

[77] S. E. Reid e U. D. Brentani, «The Fuzzy Front End of New Product Development for Discontinuous Innovations: A Theoretical Model», *Journal of Product Innovation Management*, vol. 21, n. 3, pp. 170–184, 2004, doi: 10.1111/j.0737-6782.2004.00068.x.

[78] «As 7 Fontes de Oportunidade para Inovar», *Endeavor Brasil*, Mai. 09, 2014. https://endeavor.org.br/sem-categoria/as-7-fontes-de-oportunidade-para-inovar/ (acedido Fev. 16, 2020).

[79] «Significado de SWOT (O que é, Conceito e Definição) - Significados». https://www.significados.com.br/swot/ (acedido Fev. 16, 2020).

[80] «Using the analytic hierarchy process (ahp) to select and prioritize projects in a portfolio». https://www.pmi.org/learning/library/analytic-hierarchy-process-prioritize-projects-6608 (acedido Fev. 16, 2020).

[81] C. Marins, «O USO DO MÉTODO DE ANÁLISE HIERÁRQUICA (AHP) NA TOMADA DE DECISÕES GERENCIAIS – UM ESTUDO DE CASO», p. 11, 2009.

[82] «29 Examples of Product Value», *Simplicable*. https://simplicable.com/new/product-value (acedido Fev. 21, 2020).

[83] «Customer Value: What it Means and How to Create It [5+ Ideas]», *Tallyfy*, Mai. 25, 2017. https://tallyfy.com/customer-value/ (acedido Fev. 21, 2020).

[84] C. M. Kopp, «Understanding Perceived Value», *Investopedia*. https://www.investopedia.com/terms/p/perceived-value.asp (acedido Fev. 21, 2020).

[85] «Proposta de valor: o que é e como criar a proposta perfeita». https://rockcontent.com/blog/proposta-de-valor/ (acedido Fev. 22, 2020).

[86] «Canvas online: experimente novos modelos de negócio | Runrun.it», *Runrun.it Blog*, Nov. 22, 2019. https://blog.runrun.it/canvas-online/ (acedido Fev. 22, 2020).

[87] «O Modelo de Cadeia de Valor de Michael Porter - Portal Gestão». https://www.portal-gestao.com/artigos/6991-o-modelo-de-cadeia-de-valor-de-michael-porter.html (acedido Fev. 23, 2020).

[88] «Cadeia de valor de Michael Porter: Como funciona?» https://casadaconsultoria.com.br/cadeia-de-valor/ (acedido Fev. 23, 2020).

[89] «Angular». https://angular.io/ (acedido Jun. 12, 2020).

[90] «Angular - Introduction to Angular concepts». https://angular.io/guide/architecture#templates-directives-and-data-binding (acedido Jun. 19, 2020).

[91] «Angular - Introduction to components and templates». https://angular.io/guide/architecture-components (acedido Jun. 19, 2020).

[92] «Angular - Introduction to services and dependency injection». https://angular.io/guide/architecture-services (acedido Jun. 19, 2020).

[93] «Node.js - Route-Controller-Service structure for ExpressJS | node.js Tutorial». https://sodocumentation.net/node-js/topic/10785/route-controller-service-structure-for-expressjs (acedido Jun. 19, 2020).

[94] «Beautiful Angular Charts & Graphs | 10x Fast | CanvasJS». https://canvasjs.com/angular-charts/ (acedido Out. 04, 2020).

[95] «bcrypt», *npm*. https://www.npmjs.com/package/bcrypt (acedido Out. 04, 2020).

[96] «util.promisify», *npm*. https://www.npmjs.com/package/util.promisify (acedido Out. 04, 2020).

[97] «Teste de Integração na Prática». https://www.devmedia.com.br/teste-de-integracao-na-pratica/31877 (acedido Out. 08, 2020).

[98] «Entenda de uma vez por todas o que são testes unitários, para que servem e como fazê-los | by Dayvson Lima | Medium». https://medium.com/@dayvsonlima/entenda-de-uma-vez-por-todas-o-que-s%C3%A3o-testes-unit%C3%A1rios-para-que-servem-e-como-faz%C3%AA-los-2a6f645bab3 (acedido Out. 08, 2020).

[99] I. Carvalho, «Quais são as funções do Teste Alfa, Beta, e Regressão?», *Medium*, Abr. 01, 2019. https://medium.com/@ingrid.carvalho.mo/quais-s%C3%A3o-as-fun%C3%A7%C3%B5es-do-teste-alfa-beta-e-regress%C3%A3o-b0db1c3bf5d0 (acedido Out. 08, 2020).

[100] «Angular - Testing». https://angular.io/guide/testing (acedido Out. 09, 2020).

[101] «Jasmine Documentation». https://jasmine.github.io/ (acedido Out. 09, 2020).

[102] «Karma - Spectacular Test Runner for Javascript». https://karma-runner.github.io/latest/index.html (acedido Out. 09, 2020).

[103] «Mocha - the fun, simple, flexible JavaScript test framework». https://mochajs.org/ (acedido Out. 09, 2020).