INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




API Economy para Comunicações Transacionais

JOÃO MANUEL ARÊDE FERREIRA MARTINS CAMELO
Outubro de 2020

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

API Economy para Comunicações Transacionais

João Manuel Arêde Ferreira Martins Camelo

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas Computacionais

Orientador: Paulo Gandra de Sousa

Supervisor: Ivo Pereira

Porto, Outubro 2020

Dedicatória

Dedicado a toda a equipa da E-goi, em especial o Carlos, o António e o Henrique, e ao professor Paulo Gandra de Sousa.

Resumo

A E-goi apresenta-se como uma plataforma de marketing multicanal, que evoluiu para possibilitar o envio de mensagens transacionais. O produto que envia este tipo de mensagens denomina-se Slingshot, que, apesar de funcional, cria confusão e dificuldade de integração por parte de quem a usa. Perante as dificuldades sentidas, surgiu a necessidade de desenhar uma nova versão da Slingshot, de forma a facilitar a sua disponibilização e aceitação por parte de clientes novos e antigos.

Neste documento está presente uma introdução ao tópico e contexto do problema, e respetivos objetivos a atingir com o projeto. Segue-se um estudo ao estado da arte, analisando os conceitos relacionados com Slingshot e seus concorrentes, dentro do contexto de *API Economy*. Após, é feita uma análise de valor sobre o projeto, de forma a validar a pertinência do projeto e seu desenvolvimento.

Após a definição do contexto e da relevância do projeto, o documento apresenta a análise à plataforma existente, de forma a complementar a informação necessária para o desenho de uma solução. Depois, este desenho é apresentado e justificado, de acordo com a pesquisa efetuada.

A solução desenhada traduziu-se no desenvolvimento de uma nova versão da Slingshot, dependente desta e de outros serviços disponibilizados pela E-goi. Para complementar a solução, foi desenvolvida documentação pertinente e completa, e disponibilizados excertos de código que os utilizadores poderão utilizar ao integrar a plataforma.

A solução foi testada em duas partes. A primeira do ponto de visto técnico, onde se analisou o comportamento dos serviços face diferentes tipos de pedidos, e se mediu o tempo de resposta. A segunda do ponto de vista de utilização, através da análise de respostas dos desenvolvedores da E-goi a um inquérito, onde as duas versões foram comparadas do ponto de vista da documentação. Os resultados destes testes permitiram tirar conclusões sobre o desenvolvimento da solução e futuro da mesma dentro da E-goi.

Palavras-chave: API, API Economy, Mensagens Transacionais

Abstract

E-goi presents itself as a platform for multichannel marketing, that evolved to include transactional messages. The platform that sends this type of messages is called Slingshot which, although functional, creates confusion and difficulties to those who use it. Towards the hardships encountered, the need to draw a new version of the Slingshot platform arisen, in order to ease its availability and acceptance by old and new clients.

In this document there is present an introduction to the topic and context of the problem, and respective objectives to achieve with the project. A study to the state of the art follows, analyzing the concepts related with Slingshot and its competitors, all within the context of API Economy. After, a value analysis is made about the project, in order to validate the pertinence of the project and its development.

Once the context and relevance of the project are established, the document presents an analysis of the existent platform, in order to add to the information necessary to the design of a solution. After that, said solution is presented and justified according to the information gathered.

The designed solution was translated as a new version of Slingshot, dependent of it, as well as other services made available by E-goi. To complement the solution, a new documentation with complete and relevant information was developed, as well as code samples that the users can make use when integrating the platform.

The solution was tested in two parts. The first focus on the technical aspect, where an analysis of the behavior of the services was made, given different types of requests, and the time it took to respond. The second focus on the comprehension of the users, using the answers of other E-goi developers to an inquiry, where the documentation of both versions is compared. The results of these tests allowed conclusions to be drawn about the development of the solution and its future within E-goi.

Keywords: API, API Economy, Transactional Messaging

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

2FA	*Two Factor Authentication*
AHP	*Analytic hierarchy process*
API	*Application Programming Interface*
B2D	*Business to Developer*
CRUD	*Create, Read, Update, Delete*
HATEOAS	*Hypermedia as the Engine of Application State*
HTTP	*HyperText Transfer Protocol*
REST	*Representational State Transfer*
SaaS	*Software as a Service*
SDK	*Software Development Kit*
SMS	*Short Message Service*
SMTP	*Simple Mail Transfer Protocol*
SWOT	*Strengths, Weaknesses, Opportunities, Threats*
URL	*Uniform Resource Locator*
V1	Versão atual da Slingshot
V2	Nova versão da API

Lista de Símbolos

λ	Valor de Eigen (aplicado ao método AHP)

1 Introdução

Neste capítulo é realizada uma descrição do problema, o contexto e os objetivos do projeto. Também é realizada a apresentação do processo de desenvolvimento do projeto e da estrutura do documento.

1.1 Contexto

Esta dissertação descreve o projeto desenvolvido na empresa E-goi, referente a unidade curricular Tese de Mestrado, do Instituto Superior de Engenharia do Porto.

A E-goi é uma plataforma *Software as a Service* (SaaS) orientada para a automação de marketing multicanal. Ela oferece vários canais para o efeito, como email, SMS, notificações push, e recentemente publicações em redes sociais como Facebook e Twitter. Para complementar o envio de mensagens através desses canais, a E-goi permite a editar o corpo das mensagens a enviar, gerir destinatários, definir fluxos de comportamento conforme o estado das mensagens, entre outras funcionalidades. O objetivo principal da plataforma é ajudar os seus clientes a atingir os seus objetivos profissionais, facilitando a interação entre estes e os seus contactos.

Para além da vertente de marketing para o qual a plataforma é orientada, surgiu a necessidade de desenvolver uma plataforma de envio de mensagens transacionais. Estas mensagens possuem objetivos e comportamentos distintos daquelas mencionada no parágrafo anterior. Mensagens transacionais é um meio de comunicação individual entre um destinatário e um remetente, com o objetivo de passar informações relevantes e temporárias ao destinatário.

Esta plataforma, denominada Slingshot, permite enviar mensagens para contactos individuais, com conteúdo próprio e distinto das mensagens enviadas pela plataforma E-goi. Para além do envio de mensagens, a Slingshot permite configurar e despoletar alguns casos de uso próprios, como fluxos de mensagens multicanal e registo de mensagens. Com esta oferta, a E-goi complementa a sua oferta e abre oportunidade a novas fontes de rendimento. A aplicação, em Setembro de 2020, é utilizada por mais de 130 clientes E-goi, enviando acima de 1.500.000 emails e 2.000.000 SMS's por mês.

1.2 Problema

Em 2014, por ocasião dos Jogos Olímpicos no Rio de Janeiro, a E-goi foi desafiada a produzir uma ferramenta de envio de mensagens transacionais. O produto final deste desafio foi a plataforma Slingshot, que desde então a integra como meio de envio deste tipo de mensagens.

Apesar de funcional, a plataforma é vítima de algumas limitações técnicas, fruto do seu curto espaço de desenvolvimento. Desde desenho até disponibilização da mesma, a Slingshot foi desenvolvida por uma equipa pequena em apenas cerca de 3 meses. Esta limitação temporal levou a algumas decisões de desenho que, não estando incorretas, dificultam a compreensão por parte dos clientes Slingshot.

Alguns dos problemas da plataforma incluem métodos redundantes, estrutura de caminhos complexa, autenticação inconsistente e documentação deficiente. Estes fatores, conjugados com as limitações técnicas de alguns clientes, diminuem a confiança na plataforma, que pode levar a não aceitação da mesma.

Outro desafio que a plataforma apresenta é a grande quantidade de métodos descontinuados. A descontinuação de funcionalidades é uma prática controversa, que coloca em risco certas integrações dos clientes. Garantir que um método poderá ser removido sem consequências é muito difícil, uma vez que 90% dos clientes de uma API não tem reação perante serviços descontinuados (Sawant, Robbes, & Bacchelli, 2019). Por este motivo, qualquer alteração à API deve ser encarada com apreensão.

Até a data da escrita do presente documento, a plataforma tem sido sujeita a trabalhos de manutenção, quer através de correções a plataforma, quer através do desenvolvimento de novas funcionalidades e casos de uso. No entanto, o facto da Slingshot estar a ser ativamente utilizada pelos clientes E-goi levanta questões na alteração do contrato da API, que coloca em risco a compatibilidade com clientes atuais. Desta forma, devem ser estudadas opções de correção da plataforma, tendo em conta os clientes atuais e as necessidades futuras da Slingshot.

1.3 Objetivos

Os principais objetivos que o projeto deve atingir distinguem-se em três vertentes.

A primeira é a reestruturação do contrato da API atual. Será necessário redesenhar a API existente segundo boas práticas de engenharia, de forma a ser possível corresponder às exigências dos clientes que a usam e garantir a correta escalabilidade consoante novas funcionalidades vão sendo desenvolvidas. Será tido em conta o plano de desenvolvimento da Slingshot, de forma a acompanhar a evolução da mesma.

A segunda é a diminuição da dificuldade de compreensão da API atual. Com o presente projeto pretende-se reduzir a taxa de insatisfação dos utilizadores e minimizar os casos de não-conversão de vendas após avaliação técnica da API existente e sua documentação. A E-goi tem testemunhos de clientes que criticam a documentação pela sua falta de clareza e pouca informação. Para este objetivo será desenvolvida documentação com qualidade, que acompanhará o contrato definido no objetivo anterior.

A terceira é aumentar os meios de integração da Slingshot. Pretende-se estudar o conceito de API Economy, explorando a capacidade dos seus atores, e explorar soluções que facilitem o uso da API por parte dos clientes.

Espera-se que esta nova API suporte novos casos de uso e aumente a satisfação geral dos clientes da Slingshot, novos e atuais. Todos os objetivos deverão ser atingidos até Setembro de 2020.

1.4 Abordagem e processo de desenvolvimento

O processo de desenvolvimento do projeto será feito em quatro etapas.

Na primeira etapa será feito um estudo sobre o tema, explorando o conceito de API *Economy*, de forma a compreender como uma API se integra com outras plataformas, e como poderá evoluir. Seguidamente, será feito um estudo as práticas aconselhadas pelo mercado, as práticas dos concorrentes da Slingshot, e padrões adequados ao contexto.

Na segunda etapa será feito o levantamento de requisitos, de forma a desenhar a solução ideal para os objetivos. Aqui inclui-se a escolha da tecnologia, arquitetura da solução, tendo em conta o trabalho de pesquisa feita na primeira etapa.

Na terceira etapa a solução será implementada. Terá tido em conta o desenho da solução da segunda etapa, bem como boas práticas relacionadas com as tecnologias utilizadas.

Na quarta etapa, a aplicação será avaliada. Serão desenhados testes que não só avaliarão a qualidade dos serviços e da aplicação desenvolvida, como a documentação e outras ferramentas que possam ser desenvolvidas. As conclusões serão recolhidas e analisadas no presente documento, assim como a documentação de todas as anteriores etapas.

1.5 Estrutura do documento

A estrutura do documento é a seguinte:

1. **Introdução**, onde será apresentada o contexto inicial do projeto;

2. **Estado da Arte**, onde serão estudados os conceitos necessários a realização do projeto;

3. **Análise de Valor**, onde será feito um estudo a pertinência do projeto e da potencial solução;

4. **Análise à Plataforma**, onde será feita uma análise à Slingshot, tanto do ponto de vista técnico como de negócio;

5. **Design da Solução**, onde serão levantadas as necessidades e requisitos da solução, e estudadas possíveis soluções e alternativas;

6. **Implementação da Solução**, onde será apresentada a solução a ser desenvolvida, devidamente fundamentada e documentada.

7. **Avaliação da Solução**, onde será estudado os resultados da aplicação perante as partes interessadas.

8. **Conclusão**, onde se fará o resumo dos resultados do projeto.

2 Estado da Arte

Neste capítulo é descrito com o contexto do problema. É feito um estudo sobre API Economy, o que é e como se integra o conceito no projeto. É apresentada uma análise à plataforma da Slingshot, e comparada com produtos semelhantes de organizações concorrentes.

2.1 API

Application Programing Interface (API) descreve uma funcionalidade ou conjunto de funcionalidades que poderão ser utilizadas para facilitar a execução de várias tarefas (Robillard, What Makes APIs Hard to Learn? Answers from Developers, 2009). Estas funcionalidades são disponibilizadas por organizações, geralmente através de recursos na internet, aos quais os seus utilizadores podem aceder. Na Figura 1 está apresentado como uma API permite várias integrações. No contexto deste documento, considera-se que uma API é disponibilizada através de uma chamada a um recurso disponível através da internet.

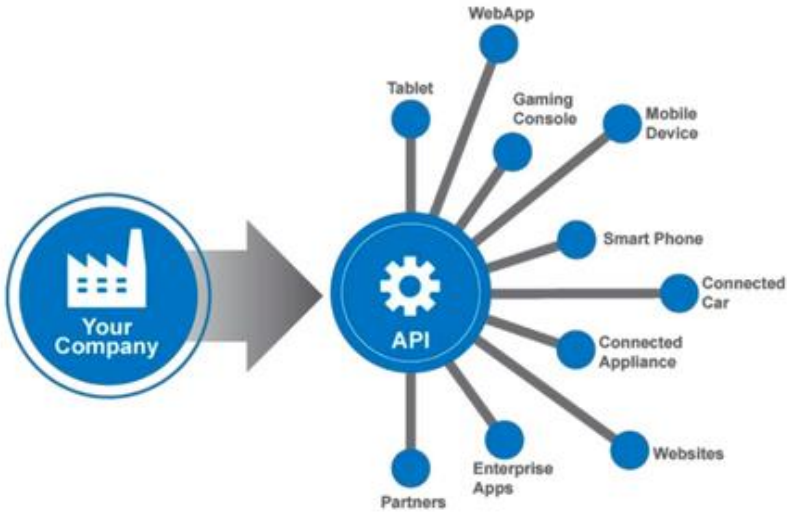

Figura 1 - Disponibilização de uma API (Glickenhouse, 2018)

As integrações apresentadas na Figura 1 devem ter por base uma determinada forma de interação que a API disponibilize, de forma a manter um padrão de utilização independente do utilizador. Isto porque, tal como na Figura 1, os utilizadores podem ser humanos ou as mais variadas aplicações e dispositivos.

Para conseguir definir não só o âmbito da API como a forma de interação é necessário estabelecer quais as principais propriedades da API. Algumas dessas propriedades são (Wodehouse, 2016):

1. **Segurança** – Existem duas formas que uma API pode ser disponibilizada, de acordo com o tipo de utilizadores que seja o mercado alvo: Públicas ou Privadas. A primeira é aberta a qualquer utilizador, enquanto que a segunda cria restrições de acesso. Dependendo do tipo de recursos que a API lide, ou como a API é comercializada, pode ser necessário restringir os utilizadores;

2. **Contrato** – Uma API deve ser encarada como um contrato entre a organização que a gere e os seus utilizadores. Qualquer método de uma API deve corresponder exatamente aquilo que é documentado, a fim de diminuir o risco de mau uso da API. Por esse motivo é necessário muito cuidado a qualquer alteração do mesmo, como a descontinuação algum método, sob pena de quebrar o contrato com o utilizador;

3. **Homogeneidade** – A utilização de API's permite que várias aplicações partilhem informação entre elas. Este tipo de comunicação obriga a que os seus serviços sejam idempotentes, garantindo o cumprimento do contrato e partilha de recursos;

4. **Integração** – Uma API pode ser um recurso flexível, adaptando-se a vários tipos de mensagem, e facilmente integrando-se com outras aplicações. Desta forma promove-se a partilha de recursos e reutilização de código.

5. **Universalidade** – A conjugação entre o contrato e a integração da API deve ser suficiente para um utilizador conseguir usufruir das funcionalidades dela. Mesmo com limitações como segurança ou limite de pedidos, estes devem estar claramente definidos, de modo a conseguir criar um padrão universal de utilização da API.

6. **Filtragem** – API's devem possuir opções de filtragem na sua utilização. Quando conjugado com a componente de segurança, a API pode limitar os seus utilizadores através de limitação de pedidos e/ou limitação de funcionalidades. Outro tipo de filtro que pode ser estabelecido prende-se com a informação que é devolvida ao utilizador. A construção de uma API deve oferecer opções de filtragem de informação, de forma a reduzir o tráfego de dados e aumentar o tempo de resposta.

2.1.1 Definição de uma API

A necessidade de desenvolvimento de uma API deve surgir como um passo natural na evolução de uma organização. Definir qual o objetivo da API é importante para definir qual será o contrato a ser implementado, de forma a promover a consistência de funcionalidade aos

utilizadores da API. Por este motivo é importante dividir as componentes de uma API, de forma a que fiquem bem definidas e com funcionalidades próprias.

Tendo definido qual o âmbito e contrato da API, deve ter-se em conta a integração da API. Para tal, devem ser tomadas algumas decisões que, independentemente do objetivo da API e dos utilizadores da mesma, mantém a coerência para quem integra.

De forma a definir uma boa API devem ser utilizadas convenções aceites na indústria, como (Tauberer, 2014):

1. Informação granular e Filtragem– a informação que o utilizador quer aceder dever ser facilmente acessível, e incluir mecanismos de filtragem, de forma a reduzir a informação aquilo que é necessário, e evitar desperdícios de transporte e processamento;

2. Valores "tipados" – tanto os pedidos como as respostas devem ter o seu tipo de dados claros e definidos, de forma a evitar problemas de compatibilidades entre o servidor e cliente;

3. RESTful – A API deve cumprir com princípios REST na medida do possível;

4. Variedade de formatos de resposta – Vários formatos de resposta equivalem a mais opções para o utilizador utilizar o que lhe é confortável. Desta forma o cliente pode utilizar o formato que deseja;

5. Mensagens de erros claras – As mensagens de erro devem informar o cliente sobre o comportamento que levou ao erro, e orientá-lo para tentar corrigi-lo;

6. Simplificar métodos de acesso frequentes – Existem recursos que serão mais acedidos que outros, pelo que o acesso a esses recursos estar o mais simplificado possível, sem comprometer a funcionalidade do serviço;

7. Bibliotecas para o cliente – Se possível devem ser disponibilizadas ao cliente opções de integração da API, como bibliotecas de integração ou excertos de código;

8. Alto desempenho – A API deve ser competitiva no tempo de resposta aos pedidos que recebe, de forma a evitar esperas prolongadas do cliente e perder o seu interesse;

9. Relação com os utilizadores – A API deve possuir uma boa comunicação com os seus clientes, quer através de canais individuais como emails, ou públicos, como publicações em blogs e documentação. Desta forma facilita-se a adaptação dos clientes a alterações que ocorram na API.

Tendo definido as boas práticas, pode dar-se início ao processo de escolha das componentes que compõem a API.

2.1.1.1 Autenticação, Autorização e Limitações

As API's dividem-se em dois géneros em relação à sua distribuição. Estas podem ser públicas, ou seja, abertas a qualquer utilizador, ou privadas, limitadas a alguns utilizadores.

API's públicas podem ser utilizadas por qualquer utilizador. No universo de APIs são conhecidas como OpenAPIs, onde qualquer utilizador ou cliente se pode ligar a elas e partilhar recursos com outros utilizadores. Em alguns casos a utilização das mesmas pode ser complementada com um dos métodos de autenticação mencionados, de modo a facilitar a utilização da mesma e ajustar a funcionalidade ao que o cliente espera.

API's privadas, por outro lado, restringem os seus recursos a utilizadores específicos e autenticados, utilizando um método de autenticação. Organizações escolhem este tipo de abertura por vários motivos, entre os quais monetização da API, autorização de certos componentes, acompanhamento de métricas, entre outros motivos.

A forma de autenticação pode dar-se de várias formas, entre as quais (Sandoval, 2018):

1. Apikey – é um conjunto de carateres que identifica o utilizador perante a API. Este conjunto de carateres identifica o cliente dentro da API, garantindo as funcionalidades associadas a ele. A Apikey deve ser exclusiva para cada utilizador, e não deve ser partilhada, evitando situações de mau uso e consequências para o utilizador;
2. OAuth – coloca um intermediário entre o cliente e o servidor, e garante a integridade do cliente e servidor entre os dois. Este intermediário ser dado por uma entidade imparcial e estabelecida no mercado, sendo que o exemplo mais recorrente seá o das redes sociais. A autenticação é estabelecida através da associação com essa entidade, que gera um token e o partilha entre cliente e servidor, estabelecendo uma relação entre ambas;
3. Login – o cliente envia alguns dados pessoais (geralmente um identificador e password) ao servidor, iniciando uma nova sessão de utilização da API, em que o cliente está autenticado. Esta sessão é temporária, expirando ao fim de um determinado tempo (pré-definido pelo servidor). Após expirar o utilizador necessita de fazer novamente o processo de Login.

2.1.1.2 Erros de utilização

Apesar da API se esforçar por ser o mais acessível possível, a sua utilização está sujeita a erros de comunicação ou processamento entre cliente e servidor. Existem vários fatores onde o erro humano ou técnico pode surgir. O cliente pode fazer um pedido mal construído, e a API não conseguir processá-lo, ou o servidor pode ter uma alteração de comportamento que o cliente não espera.

Saber quais as condições que poderão causar problemas de interação, e saber quais as potenciais respostas permite ao desenvolvedor adaptar a sua aplicação, podendo desenvolver fluxos alternativos, e reduzindo a insatisfação do utilizador final (Neumann, Laranjeiro, & Bernardino, 2018).

Na Tabela 1 estão apresentadas as principais causas de erros numa API.

Tabela 1 - Causas de falhas na utilização de APIs (Aué, Aniche, Lobbezoo, & van Deursen, 2018)

Parte interessada	**Causa**	**Explicação**
Utilizador Final	Informação inserida inválida	O utilizador inseriu informação de forma incorreta, ou a informação não é válida no contexto da aplicação.
Utilizador Final	Informação em falta	O utilizador não inseriu informação suficiente para processar o pedido.
Utilizador Final	Validade de informação expirada	O utilizador inseriu informação fora do limite temporal em que seria válida.
Consumidor da API	Informação inserida inválida	Ocorreu um problema devido a informação inválida pelo consumidor da API
Consumidor da API	Informação em falta	O consumidor da API não inseriu informação suficiente para processar o pedido.
Consumidor da API	Permissões insuficientes	O consumidor da API não tem as permissões necessárias para processar o pedido.
Consumidor da API	Processamento duplo	O pedido inserido pelo consumidor da API é duplicado.
Consumidor da API	Configuração	As configurações do consumidor da API não estão alinhadas com a API.
Consumidor da API	Informação da API em falta	A API não possui os recursos que o consumidor da API pretende.
Fornecedor da API	Interno	Ocorreu um problema no processamento do pedido por parte do fornecedor da API.
Terceiros	Terceiros	Existe um problema com a aplicação de terceiros.

A análise da Tabela 1 mostra que a principal fonte de erros provém do lado do cliente. Uma vez que o cliente não tem conhecimento de como o seu pedido é processado, é necessário que lhe seja comunicado qual o problema encontrado e, se possível, como resolvê-lo.

Por outro lado, também a própria API pode levantar alguns casos de erros, por exemplo, devido a um *bug* no código. Neste caso, não existe muito que o utilizador poderá fazer para solucionar o problema, pelo que a mensagem de erro pode ter informação menos precisa.

Finalmente, as aplicações que a API utiliza poderá provocar algumas falhas no processamento de pedidos. Neste caso, ambas as partes devem ser notificadas, de forma a resolver o problema rapidamente. Para o cliente, esta situação é semelhante à falha da própria API, pelo que a mensagem de erro pode ser igualmente simplificada.

2.1.1.3 Documentação

A documentação de uma API deve acompanhar a sua disponibilização, uma vez que a sua consulta é o principal contacto que os desenvolvedores estabelecem (Meng, Steinhardt, & Schubert, 2018). Uma documentação de API deficiente dificulta a sua utilização (Zhong & Su, 2013), devendo por isso não só alinhar-se a mesma com as capacidades da API como com capacidade do utilizador de implementar um cliente para a API.

Outra função importante da documentação de uma API é estabelecer o contrato entre o cliente e o servidor. Todas as funcionalidades e modos de interação possível e válidos devem estar contemplados na documentação. Assim, o cliente poderá mais facilmente harmonizar-se com o servidor, aumentando a eficiência do processamento e diminuindo a probabilidade de erros de utilização.

2.1.2 Evolução de API

Nesta seção será explorada as condições que obrigam a evoluir uma API, quais as condições, e como fazer a evolução. Uma vez que a evolução de uma API é inevitável, através de situações como a evolução do negócio, torna-se importante manter um padrão das caraterísticas da evolução.

2.1.2.1 Motivos

Existem vários motivos que obrigam uma API a evoluir, como (Brito, Hora, & Valente, 2019):

1. **Novas funcionalidades** – São desenvolvidas novas funcionalidades a API;
2. **Simplificação** – Os métodos da API podem ser simplificados;
3. **Manutenção** – A estrutura da API necessita de ser alterada;
4. **Correção** – A API necessita de correções que colocam em risco o contrato;

Estes motivos devem ocorrer naturalmente no processo evolutivo de uma API, quer através da aplicação de correções, quer através do desenvolvimento de novas funcionalidades. No entanto, é importante considerar que a evolução imprudente pode colocar em causa tanto a estabilidade da API como o seu processamento.

2.1.2.2 Versionamento

De forma a identificar qual a versão da API a ser utilizada, deve ser definido como será feita a contagem das versões. Atualmente, a estrutura de versionamento mais utilizada será aquela proposta por Tom Preston-Werner (Raemaekers, van Deursen, & Visser, 2014):

$$[major].[minor].[patch]$$

Para cada elemento da estrutura corresponde a um determinado impacto dentro da API. Alterações do tipo ‘major’ correspondem a implementação de um conjunto de alterações na API, envolvendo alterações radicais de funcionalidades e conceitos, e, por norma, implicam quebra de funcionalidade com versões anteriores. Alterações do tipo ‘minor’ envolvem alterações e acrescento de novas funcionalidades, mas que não implicam quebras de compatibilidades com versões anteriores. Finalmente, alterações do tipo ‘patch’ envolvem melhorias e pequenas alterações nos métodos, sem quebra de funcionalidades.

Para que o cliente possa identificar a versão que necessita pode faze-lo de várias maneiras (REST API Versioning Guide, 2018):

- URL - neste caso, a versão da api é parte do endereço de chamada. A versão é integrada numa partição do endereço.
    - “https://www.api.com/v2/resource”
- Server - Uma das formas mais simples de versionar a api será a de implementar a API num servidor com um endereço próprio, que identifique qual a versão que é utilizada.
    - “https://www.apiv2.com/resource”
- Query - neste caso, a versão da api a ser utilizada estará disponível como uma variável query no endereço do pedido.
    - “https://www.api.com/resource?version=2”
- Request Header - Uma alternativa não intrusiva será a de utilizar cabeçalhos, quer através de cabeçalhos específicos (como Accept), ou cabeçalhos próprios para cada API.
    - “Accept-version:2”

A utilização de qualquer uma destas formas de identificação de API deve estar de acordo com a documentação da API, de forma a garantir que o cliente compreende como identificar a versão que pretende utilizar.

2.2 API Economy

Com a revolução tecnológica dos últimos 100 anos, criou-se um conflito entre os modelos de negócio antes e após. Esta revolução é atualmente derivada da entrada do computador no paradigma das organizações. A automação de certos processos, aliado a uma gestão de informação dinâmica, levou as organizações a desenvolverem as infraestruturas necessárias para acomodar esta evolução. A utilização de computadores e componentes associados, como bases de dados e servidores, permitiu as organizações acompanhar e até redefinir o mercado, permitindo partilha de recursos de forma mais simples e eficaz.

A introdução da internet permitiu as organizações a partilha de informações dentro dela própria, estabelecendo um padrão de gestão dos processos. Mais importante ainda, a internet gerou um novo conceito de negócios, disponível em qualquer altura para o mercado que pudesse aceder. Este modelo de negócios é uma consequência da intrusão da internet na rotina diária da sociedade moderna, onde a informação deve estar disponível o mais rapidamente possível. Desta forma, considera-se a internet uma plataforma imparcial, onde a primeira organização a disponibilizar um serviço de qualidade consegue frequentemente estabelecer-se como marco do mercado. Exemplos atuais deste tipo de mercados incluem a Google e Amazon (Hernández, 2018).

Nesta altura existe a necessidade de esclarecer dois conceitos de negócios. O primeiro é o conceito de Pipeline, onde a produção de valor para uma organização é centrada no produto.

O segundo, é o conceito de Plataforma, onde o valor é produzido através da partilha e alteração de serviços e recursos. A Figura 2 apresenta os dois tipos de modelos.

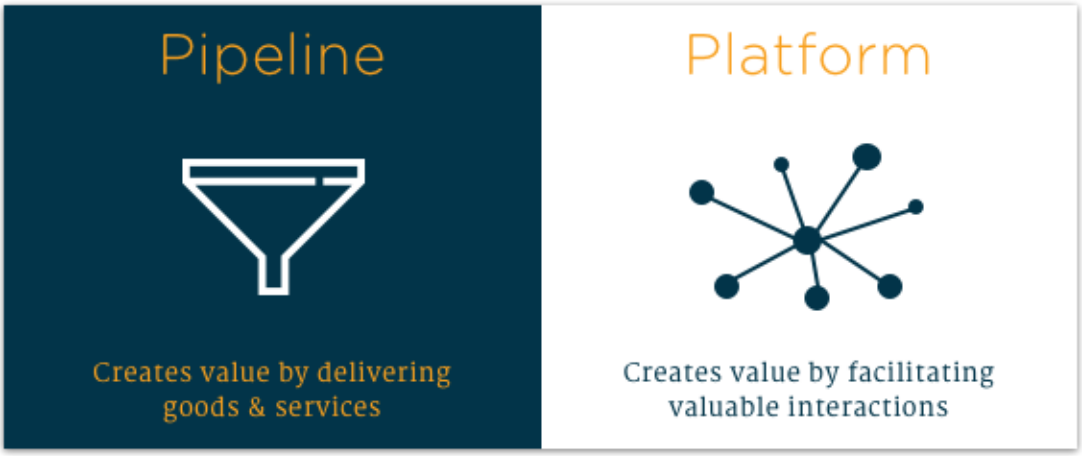

Figura 2 - Modelos de negócios Pipeline e Plataforma (Minds, 2017)

É percetível pela Figura 2 que o modelo de Plataforma promove interação e integração entre vários produtos, sendo que, por norma, um se destaca como o foco da oferta. Um exemplo deste tipo de modelo é a indústria de videojogos, onde primeiro se vende a máquina própria para tal, e se acrescenta valor através da venda de títulos para jogar. Este modelo contrasta com os modelos de negócio Pipeline, que são focados no desenvolvimento vertical do produto. Com desenvolvimento vertical, um produto era desenvolvido de forma linear desde o desenho até disponibilização, gerando valor no processo (Van Alstyne, Parker, & Choudary, 2016). Tal é o caso da indústria eletrodoméstica, onde o valor vem da venda do produto em si.

Durante a evolução do computador surgiu a dificuldade na gestão de compatibilidades entre máquinas e aplicações indústria (Zachariadis & Ozcan, 2017). Em resposta a estas dificuldades, certas empresas como Intel e Microsoft iniciaram o processo de criação de uma proposta de valor base, disponível para todos, sobre o qual se acrescenta funcionalidades complementares (Zachariadis & Ozcan, 2017). Desta forma, verifica-se que o modelo de Plataforma se tornou a norma na indústria.

Uma das ferramentas que complementa este modelo é a internet. Dada sua penetração no mundo, com acima de 50% da população mundial ser utilizador (ITU, 2019), esta deve ser encarada como uma ferramenta viável na distribuição de componentes do modelo de negócio das organizações. Uma forma das organizações explorarem este meio é através de API's. Esta forma de disponibilização permite disponibilizar funcionalidades por necessidade, quer da organização que a disponibiliza, quer dos utilizadores fora dela.

Na Figura 3 é apresentada o papel de uma API para uma organização. De forma a aumentar a sua oferta, a API pode ser usada como ferramenta fundamental de integração, permitindo gerar valor fora do alcance orgânico da organização.

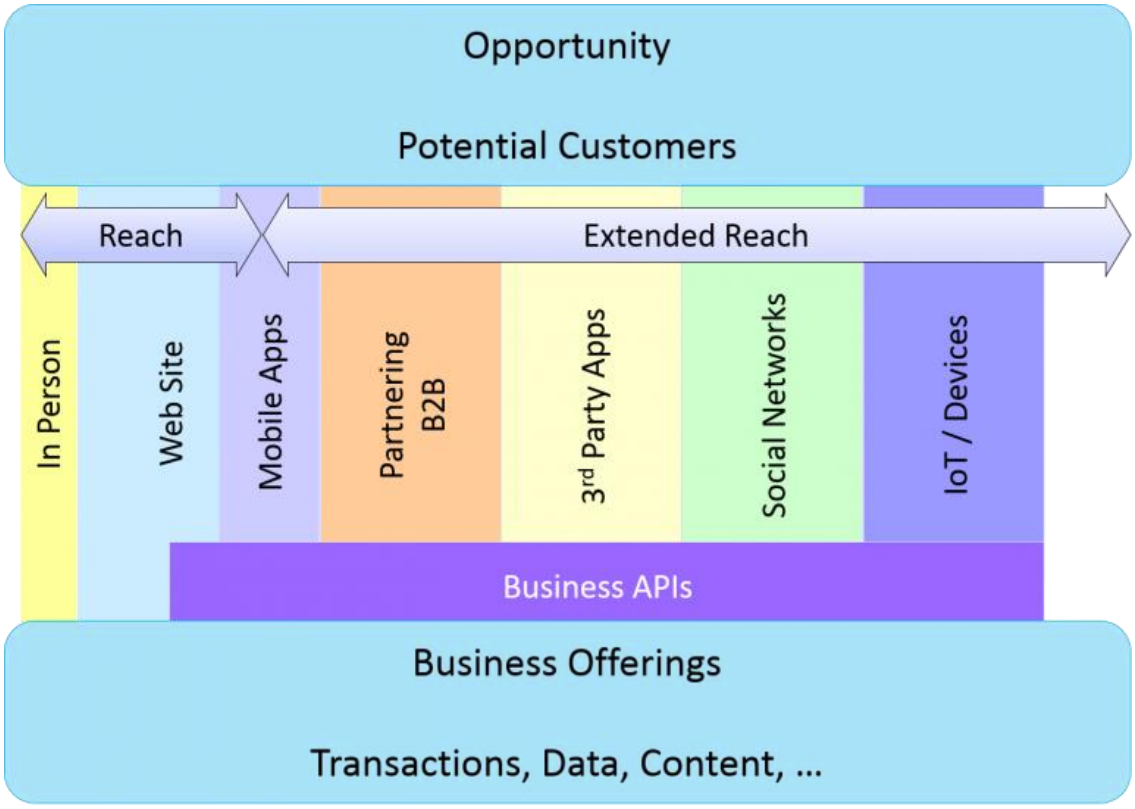

Figura 3 - Oferta de um Negócio (Glickenhouse, 2017)

A Figura 3 mostra como as parcerias entre organizações permitem aumentar o valor de ambas. Sem qualquer tipo de parceria, a organização depende apenas da sua oferta para gerar valor, e o seu crescimento é limitado pelo número de clientes diretos que a organização possui. Mas se a organização disponibilizar um meio de interação com as suas funcionalidades (por exemplo, através de uma API), o seu alcance irá incluir os clientes das organizações que integrem esse meio. Um exemplo deste tipo de interação é a disponibilização de máquinas multibanco (Zachariadis & Ozcan, 2017). A disponibilização das funcionalidades por parte dos bancos cria uma situação de simbiose entre estes, a organização responsável pelas máquinas, e o utilizador final. Disponibilizar máquinas que aceitem vários cartões bancários aumenta a oferta para o utilizador, que poderá aceder a sua conta mais comodamente. Por sua vez, o utilizador executa operações na sua conta do banco, gerando valor para todas as partes envolvidas no processo.

O conceito de API Economy descreve a expansão da oferta de uma organização através do uso de API's, de forma a permitir a integração do negócio com outras aplicações e organizações (Bonardi, Brioschi, Fuggetta, Verga, & Zuccalà, 2016). Algumas das maiores forças de negócio que potenciam a API Economy são as seguintes (Glickenhouse, 2017):

1. **Velocidade de mercado** - API's que sejam as primeiras a disponibilizar uma funcionalidade não possuem concorrência, pelo que poderão ganhar algum tempo e criar uma base de utilizadores e reputação no mercado;

2. **Aumento do número de utilizadores** - criar novas formas de interação com o software como API's permite aumentar o público interessado, não só o diretamente (programadores e aplicações) como indiretamente (utilizadores das aplicações);

3. **Inovação** - API´s permitem desenvolvimento incremental de funcionalidades, sem grande risco de perda de recursos. Por outro lado, permite procurar e criar novas vertentes de negócio, através de alterações no modelo, software e hardware atual, monetização, entre outros;

4. **Partilha de recursos** - aplicações que consomem a mesma API podem partilhar dados entre elas e diminuir a barreira de utilização das mesmas (Ex: login através de redes sociais).

2.2.1 Business to Developer

Apesar de uma API poder ser utilizada por utilizadores humanos, a grande maioria dos pedidos recebidos são automatizados por aplicações que consomem esses recursos. No entanto, para chegar as aplicações, os desenvolvedores da mesma terão de ter conhecimento sobre qual API se ligar. Este é um desafio para quem disponibiliza a API, uma vez que o seu público alvo é um segmento discreto no mundo de negócios. A necessidade de alinhar os objetivos da organização com as capacidades do produto recai sobre os seus desenvolvedores, que analisam a sua aplicação e integrações do ponto de vista do líder da organização (Quartz+Co, 2007). Como o desenvolvedor é normalmente a pessoa que tem os conhecimentos técnicos, consegue discernir quais as parcerias e ferramentas que trarão os maiores benefícios para ele e a aplicação que desenvolve. Por este motivo, será ele quem decide qual a ferramenta a integrar na adoção de um determinado componente nas suas aplicações.

Desta necessidade nasce um novo conceito de negócio, denominado *Business to Developer* (B2D). Este conceito provoca uma alteração na oferta do produto, levando as organizações fornecedoras a desenharem os seus serviços de forma a apelar e facilitar a aceitação por parte dos desenvolvedores.

De forma a aumentar a sua utilização, as organizações procuram forma de chegar não apenas aos seus clientes, mas aos indivíduos que necessitam de consumir a API para atingir os seus objetivos. Uma API é tão mais apelativa quanto mais aplicações a utilizarem e/ou pedidos receberem (Doerrfeld, Wood, Anthony, Sandoval, & Lauret, 2016). De facto, o crescimento na base de utilizadores de uma aplicação é exponencial, levando ao interesse de quem não conhecia, e contribuindo para o crescimento (Doerrfeld, Wood, Anthony, Sandoval, & Lauret, 2016).

Na Figura 4 é apresentada a relação entre os diversos componentes e atores na produção e utilização de uma aplicação. Note-se que esta figura apresenta apenas um exemplo no modelo de desenvolvimento, não representando necessariamente o modelo aplicado na indústria.

Figura 4 - Cadeia de integração de uma aplicação (Wodehouse, 2016)

Observando a Figura 4, é possível compreender o papel da API como intermediário entre o programador e os recursos que necessita. Ela faz a gestão entre os ativos da organização e os desenvolvedores que as integraram nas suas aplicações. Da mesma forma, o desenvolvedor tem um papel semelhante, atuando como intermediário entre a sua aplicação e a API. Dependendo da situação, pode ocorrer que o desenvolvedor seja a figura mais poderosa nesta cadeia, uma vez que, não só terá opção de escolha sobre qual API integrar, como a popularidade da sua aplicação pode alavancar o seu poder de negociação na escolha desta.

É relevante chamar a atenção que, excetuando alguns casos, todas as relações entre os atores e componentes ocultam outras relações, ou seja, não transmitem a complexidade da relação para quem originou a interação. Por exemplo, no momento em que o utilizador final necessita de consultar alguma informação, este não tem conhecimento dos passos necessários para retornar a informação que deseja.

Na Tabela 2 é apresentada a comparação do foco entre um utilizador final e um desenvolvedor.

Tabela 2 - Transferência de experiência do utilizador para o desenvolvedor (Fagerholm & Münch, 2012)

Foco	**Perspetiva do Utilizador**	**Perspetiva do Desenvolvedor**
Experiência positiva + uso apropriado + uso eficiente	Experiência do utilizador	Experiência do desenvolvedor
Uso apropriado, adequação ao propósito + uso eficiente	Desenho centrado no utilizador	Compreensão da relação processo-produto num determinado contexto
Eficiência e facilidade de uso	Usabilidade	Modelação de processos descritivos, modelação de processos adaptativos
Evitar defeitos da utilização, aumentar robustez, segurança	Desenho da interface do utilizador	Modelação de processos prescritivos
Objetivo Final	Usar o produto/serviço	Criar o produto/serviço

A conclusão que se tira da Tabela 2 é a diferença no foco de cada perspetiva. O utilizador apresenta uma perspetiva de produto, enquanto que a perspetiva do desenvolvedor é de um processo para criar um produto. O utilizador espera utilizar a aplicação, e esta é desenhada a volta do denominador comum de utilizadores, ou seja, do utilizador com pouco ou nenhum conhecimento sobre a aplicação. No caso do desenvolvedor, que se espera que tenha algum conhecimento técnico, isto não se aplica, pelo grau de complexidade da sua aplicação e integrações depende exclusivamente de si. De notar que, apesar da complexidade mencionada, a utilização direta do desenvolvedor num produto/serviço será menor que um utilizador comum, sendo que a utilização indireta é feita por parte da aplicação que desenvolve.

2.3 Marketing vs. Transacional

A necessidade da existência da plataforma transacional da Slingshot levanta a questão da necessidade da sua existência. A E-goi foca-se principalmente no envio de campanhas de marketing, de um remetente para vários destinatários simultaneamente. A principal dificuldade neste modelo de negócios é a complexidade em automatizar o envio de mensagens transacionais para um utilizador. Esta dificuldade levanta a necessidade de esclarecer a diferença entre estes tipos de mensagens, tanto para definição dos conceitos no modelo de negócio, como desenvolvimento de funcionalidades.

Marketing é definido como o conjunto de atividades e processos de uma organização para criação e comunicação de ofertas que possuem valor para os seus clientes, parceiros e até sociedade em que se insere (American Marketing Association, 2017). É uma das componentes que requer mais atenção numa organização, principalmente naquelas com fins lucrativos (Rao & Klein, 1994).

As estratégias de comunicação de oferta de uma organização através de tecnologias como a Internet denominam-se marketing digital. Este tipo de marketing apresenta algumas vantagens em relação a estratégias tradicionais, tais como custos mais baixos, melhor aproveitamento de oportunidades temporárias, e adaptação de conteúdo ao cliente (Todor, 2016). Alguns exemplos deste tipo de marketing incluem campanhas de email temáticas (por exemplo, relacionadas com a época natalícia), anúncios em secções de websites e publicações em blogs. É nesta vertente que a E-goi apresenta a sua principal oferta.

No entanto, existem comunicações necessárias entre organizações e seus clientes que não caem no âmbito de marketing. Casos específicos a cada um dos clientes, como confirmação de informação e envio de códigos privados devem ser tratados de forma distinta. Desta necessidade surgiu o conceito de comunicações transacionais.

Transacional, no contexto de comunicações, vem do termo transação, que descreve uma interação entre intervenientes para troca de recursos. Esta definição implica que este tipo de comunicações é fruto de uma interação prévia entre ambas as partes ou da intenção prévia em realizar essa interação. Exemplos deste tipo de mensagens incluem envio de recibos (após compra *online*, por exemplo) e notificações de eventos. É neste contexto que surge a Slingshot como oferta complementar da E-goi.

Na Tabela 3 estão descritas algumas das principais diferenças entre comunicações de Marketing e Transacionais. A principal diferença entre as duas é o objetivo que alcançam. Mensagens de marketing procuram explorar o interesse dos destinatários na sua oferta, e tentar converter esse interesse em vendas e lucro para a organização. Mensagens transacionais procuram

aumentar a confiança do destinatário na organização e na respetiva oferta, de forma a fidelizar o cliente após uma interação positiva entre ambos.

Tabela 3 - Principais diferenças entre Marketing e Transacional (Claessens, 2018)

	Marketing	**Transacional**
Foco da Organização	Vendas	Retenção, Confiança
Natureza dos Destinatários	Geral	Particular
Timing da Comunicação	Pró-ativo	Reativo
Foco da Oferta	Caraterísticas	Benefícios
Frequência da Interação	Regular	Pontual
Relação com o Cliente	Limitada, geral	Personalizada, exclusiva
Preocupação com Qualidade	Organização	Organização, Cliente

As diferenças entre os dois tipos de mensagens na Tabela 3 refletem o objetivo de cada. Existe uma transferência na importância da oferta da organização, onde esta deixa de vender as características do produto no momento da venda para vender os benefícios aos clientes após a venda.

Esta distinção origina resultados próprios para cada tipo de mensagem. Mensagens transacionais, para além de possuírem melhor taxas de abertura, com 65.5% (SparkPost, 2018). Também são consideradas mensagens relativamente importantes, em que 82% de destinatários consideram mensagens transacionais como relevantes (SparkPost, 2018). Por contraste, uma campanha de email marketing possui uma taxa de abertura de mensagens entre os 17% e os 36% (Sign-Up.to, 2015).

Apesar de terem objetivos diferentes, a natureza de algumas mensagens coloca-a numa zona cinzenta entre estes dois tipos. Dando o exemplo de um desconto de aniversário numa loja, esta mensagem contém características de marketing (objetivo de criar receita) e de transacional (mensagem exclusiva a um consumidor). Situações como estas devem ser analisadas com cuidado, uma vez que existe uma relação entre o servidor que faz o envio e a taxa de abertura das mensagens que envia (Rao & Reiley, 2012). Esta separação deve ficar a cargo da organização que disponibiliza o envio deste tipo de mensagens, de forma a não comprometer a infraestrutura e os resultados.

2.4 *REpresentational State Transfer* (REST)

REpresentational State Transfer (REST) é um estilo de arquitetura de software que descreve um conjunto de restrições para interações com serviços web. Foi proposto por Roy Fielding em 2000 como padrão de utilização dos recursos HTTP disponíveis (Massé, 2012). Estas restrições foram aceites por grande parte da comunidade de programadores, que as adotou no desenvolvimento das suas aplicações.

As restrições que fazem parte do protocolo REST são as seguintes (Neumann, Laranjeiro, & Bernardino, 2018):

1. **Interface Uniforme** - a estrutura como os recursos são acedidos deverá ser bem definida e clara para quem acede aos recursos através das seguintes normas:

    - Identificação de recursos - o recurso disponível num determinado endereço URL deve ser único e representativo do conceito que pretende representar;

    - Manipulação da representação de recursos - estabelece a diferença entre recursos e representações do mesmo. Um recurso poderá ter várias representações (como mostra a Figura 5), e um cliente interage apenas com as representações do recurso, e não com o recurso em si;

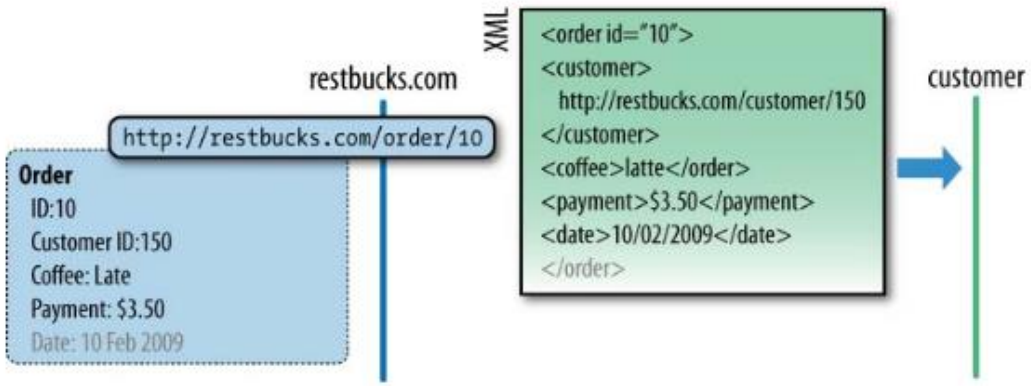

Figura 5 – Exemplo de multiplas representações de dados (Webber, Parastatidis, & RObinson, 2010)

    - Mensagens auto descritivas - todos os pedidos devem ter informação sobre como processar as mesmas, de forma a facilitar o esforço de processamento do servidor;

    - *Hypermedia as the engine of application state* (HATEOAS) - Na resposta a um pedido o servidor deve anexar links relacionados com o pedido, de forma a expor recursos que o cliente poderá ter interesse;

2. **Relação Cliente-Servidor** - estabelece que, na relação entre uma API e uma aplicação, a API terá o papel de servidor, e o componente o papel de cliente. O servidor é quem possui os recursos, e o cliente pretende fazer uso desses recursos. A relação é puramente transacional, ao ponto de ambos poderem evoluir independentemente (sem quebrar compatibilidade), sem dependências tecnológicas e sem necessidade de configuração extraordinária. No exemplo da Figura 6, o servidor possui acesso a base de dados, e os clientes necessitam de informação contida na base de dados;

3. ***Stateless*** - a interação entre cliente e servidor deve ser puramente transacional. Nem o cliente nem o servidor deverão armazenar informação sobre o outro, exceto aquela estritamente necessária para fazer ou responder a pedidos;

4. **Armazenáveis em Cache** - de forma a reduzir o número de interações entre cliente e servidor, as respostas devem indicar as condições em que poderá ser temporariamente armazenada em cache. Enquanto as condições forem válidas, o cliente não deve fazer novos pedidos ao servidor, e assim, aumenta a disponibilidade do servidor para outros clientes;

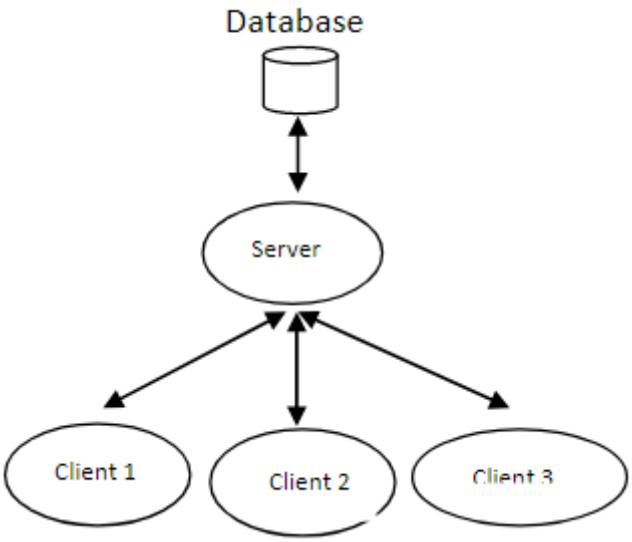

Figura 6 - Modelo Cliente-Servidor (Oluwatosin, 2014)

5. **Arquitetura em camadas** - estabelece a interação entre as várias componentes envolvidas numa transação. Um cliente não deve ter conhecimento de qualquer outra comunicação necessária por parte do servidor para gerar uma resposta. Esta restrição estabelece uma propriedade de abstração, simplificando a relação entre as várias componentes. Voltando a Figura 6, os clientes não sabem necessariamente que o servidor comunica com uma base de dados, estando apenas interessados na informação que o este retorna;

6. **Código sob Demanda** - esta é uma restrição opcional, que indica que, se necessário, o servidor pode retornar código para o cliente executar.

Estas restrições permitem limitar as formas de interação entre clientes e servidores. Desta forma é possível antecipar quais os tipos de pedidos que um servidor pode esperar receber, e qual a natureza das respostas que um cliente pode receber. Outra vantagem da adoção de REST é o uso dos recursos HTTP disponíveis no envio de cada pedido. Desta forma diminui-se o tamanho do pedido feito, e diminui-se a arga de processamento do pedido por parte do servidor.

Uma vez que pode ser complexa a adoção de todas estas restrições, foi proposto por Leonard Richardson a definição de três passos simplificados para cumprir as restrições propostas no protocolo Rest (Fowler, 2010). Na Figura 7 - Niveis de adoção Rest Figura 7 encontram-se os três passos divididos em quatro níveis de adoção. Estes níveis estão ordenados por grau de complexidade de adoção, pelo que, para uma API estar num determinado nível de adoção, deve cumprir os requisitos dos níveis anteriores.

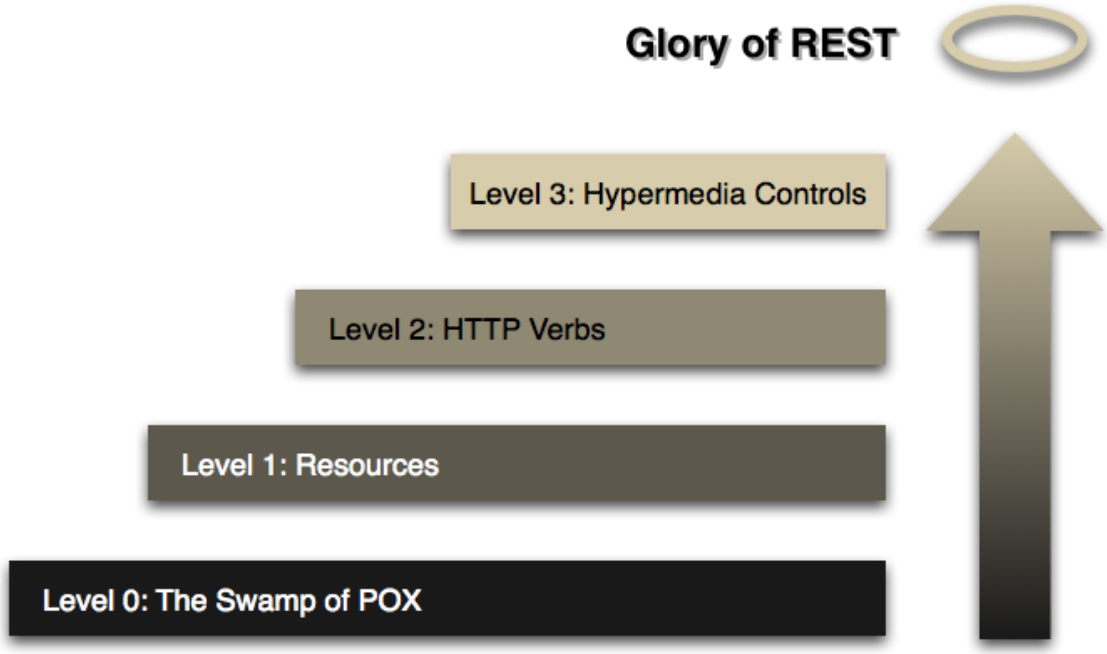

Figura 7 - Niveis de adoção Rest (Fowler, 2010)

De acordo com a Figura 7 - Niveis de adoção Rest Figura 7, uma API pode estar nos seguintes níveis de utilização das restrições REST:

- Nível 0 representa a utilização do protocolo HTTP, sem qualquer tipo de definição ou utilização das restrições REST. Este tipo de interações também é conhecido como *Remote Procedure Call* (RPC), uma vez existe apenas um recurso e uma forma de acedê-lo, independentemente do conteúdo do pedido;
- Nível 1 representa a divisão da informação em recursos. Dividir a informação em recursos permite abstrair e separar conceitos dentro da API, permitindo diferir sobre qual a informação que se deseja aceder. A informação é melhor organizada, e começa a existir o conceito de objetos;
- Nivel 2 representa o uso dos verbos HTTP corretos. É esperado que um determinado método HTTP tenha não só o comportamento esperado que é sugerido pelo seu nome, como que o mesmo seja consistente para vários serviços. É possível alinhar os métodos HTTP (*Get*, *Post*, *Put*, *Delete*) com os métodos de gestão de informação (*Create, Read, Update, Delete* (CRUD));
- Nível 3 representa o uso de *Hypermedia as the Engine of Application State* (HATEOAS). HATEOAS permite fazer ligações entre recursos. Ao fazer um pedido, a resposta a ele deve conter outros recursos relevantes ao pedido, de forma a orientar o cliente sobre a sua próxima ação na API.

3 Análise de Valor

Neste capítulo é realizada uma análise de valor as soluções que serão propostas neste capítulo. Será descrita a proposta de valor para a E-goi e clientes do Slingshot. Será feita uma análise à organização da E-goi, fazendo uso dos modelos CANVAS, SWOT e cadeia de comando de Porter. Também será feita uma análise sobre qual estratégia deve ser escolhida para orientar o desenho da solução, sendo utilizado o método AHP para a mesma.

3.1 Cadeia de Valor de Porter

De forma a compreender o modelo de negócios que a nova versão da API se insere é importante definir como se integra no contexto da E-goi. Para tal, é utilizada a Cadeia de Valor de Porter, que divide a organização em setores de atividades. Estes setores estão divididos em dois conjuntos diferentes: Atividades Primárias e Atividades de Suporte. A demonstração desta divisão está presente na Figura 8.

Atividades de suporte é horizontal a qualquer atividade de criação de ofertas por parte das organizações. Estas atividades dividem-se em:

1. **Infraestrutura** – atividades que suportam as restantes atividades como contabilidade, fiscal, entre outros;

2. **Gestão de Recursos Humanos** – atividades relacionadas com a envolvência dos colaboradores na organização, como contratações, formações e gestão de ausências;

3. **Desenvolvimento Tecnológico** – atividades relacionadas com manutenção das ferramentas necessárias, como compra de equipamento e instalação de software;

4. **Aquisição e Compras** – atividades relacionadas com a utilização de serviços externos a organização, como limpeza das infraestruturas e confeção de refeições adquiridas a empresas especializadas.

Figura 8 - Cadeia de Valor de Porter (TECNICON, 2019)

O outro conjunto de atividades são as primárias. Estes tipos de atividades são aquelas que geram valor a organização, através da disponibilização de produtos ou serviços aos mercados que necessitem. As atividades dividem-se em:

1. **Logística de Entrada** – atividades relacionadas com receção de recursos necessários a organização. Neste projeto são exemplos desse tipo de atividades a recolha de informação necessária a configuração e envio de mensagens;

2. **Operações** – atividades relacionadas com transformação de recursos. No caso do projeto estas incluem a geração e envio de mensagens, ativação de um caso de uso como o Verify, e atualização de informação;

3. **Logística de Saída** – atividades relacionadas com armazenamento de produto e distribuição do mesmo. Neste projeto englobam atividades de consulta de dados como estado de uma mensagem e informações armazenadas (como *Templates*);

4. **Marketing e Vendas** – atividades relacionadas com a promoção e entrega do produto ao cliente. Neste projeto estas atividades são a produtização da API através da disponibilização da documentação, publicações no blog da E-goi, e *cross-selling* da API para clientes E-goi;

5. **Serviços** – atividades relacionadas com manutenção do produto após a venda. Exemplos destas atividades relacionadas com o projeto incluem melhorias da aplicação, criação de novas funcionalidades, e suporte ao cliente.

3.2 Seleção de Solução (Modelo AHP)

De forma a determinar a qual a solução a desenhar e desenvolver, deve-se primeiro explorar as principais ideias e soluções propostas, de forma a determinar qual a melhor. Neste caso, a melhor ideia será aquela que satisfaz melhor as necessidades da E-goi, tendo em conta aquilo que são os seus objetivos e capacidades.

Após reflexão sobre o problema, foram propostas três ideias:

- Substituir a Aplicação por uma nova Aplicação, desenvolvida de raiz. Traz garantias de criação de uma aplicação bem-desenvolvida e que vai ao encontro das necessidades existentes. Corre o risco de quebrar compatibilidades atuais;

- Corrigir a Aplicação existente, colmatando falhas na documentação e corrigindo métodos. Pode quebrar compatibilidades em métodos que necessitem de correção, mas não necessita de suporte extra nem necessita de muitos recursos a desenvolver;

- Adicionar uma nova versão da API, mantendo a versão atual e desenvolvendo uma nova. Garante compatibilidade e correção, mas cria problemas de versionamento e gestão de aplicações.

Cada ideia acima descrita possui valor próprio para a Slingshot. No entanto, também levantam outras questões que necessitam de ser solucionadas. De forma a avaliar qual a solução mais adequada, utilizou-se o método de *Analytic Hierarchy Process* (AHP). Este método permite a quem o aplica navegar entre decisões complexas utilizando racionalidade e matemática, ajudando não só a tomar uma decisão como dar confiança a quem a toma que é a melhor decisão.

O método AHP está dividido em três passos. O primeiro é a Divisão Hierárquica, onde se define os critérios e prioridades a analisar. O segundo passo é a Definição de Prioridades, onde se conjuga os critérios com as soluções propostas, com fim de calcular a solução mais adequada ao problema. Finalmente, o terceiro passo é a Consistência, onde se avalia a integridade dos critérios da avaliação a fim de validar a escolha da solução. Nas secções seguintes serão explorados os passos do modelo no contexto do projeto.

3.2.1 Divisão hierárquica

A fim de determinar a melhor solução é necessário estabelecer os critérios e as soluções que farão parte do método. Assim sendo, foram definidos os três seguintes critérios relevantes:

- A – Tempo de desenvolvimento;

- B – Dificuldade de Manutenção;

- C – Compatibilidade com clientes atuais.

Com estes critérios pretende-se avaliar não só a dificuldade do processo de desenvolvimento do projeto como a minimização das repercussões negativas no negócio da E-goi. O critério A prevê o investimento na solução na fase de desenho e desenvolvimento. Após o lançamento e disponibilização da solução, avalia-se a mesma do ponto de vista da sua manutenção. Esta manutenção inclui atualização de tecnologias, correções necessárias e desenvolvimento de novas funcionalidades. Este ponto é representado pelo critério B. Finalmente, o critério C prevê que os clientes atuais da Slingshot sofram o mínimo de inconveniência por parte da nova aplicação. É dos interesses da E-goi que os seus clientes não encontrem dificuldades na utilização dos seus serviços, pelo que deve ter-se em atenção que a solução reduza esse risco.

As ideias acima propostas foram adaptadas as seguintes propostas:

- X – Substituir a Aplicação;
- Y – Corrigir a Aplicação;
- Z – Desenvolver uma nova versão da API.

Na Figura 9 encontra-se o diagrama AHP desenhado com os critérios e soluções mencionadas. No topo encontra-se o principal objetivo com a solução, que no caso é o de melhoramento da API. Por baixo estão os critérios de avaliação da solução. Por baixo deles, as soluções propostas. Note-se que todos os critérios serão aplicados de igual forma sobre todas as soluções propostas, de forma a avaliá-las de forma igualitária.

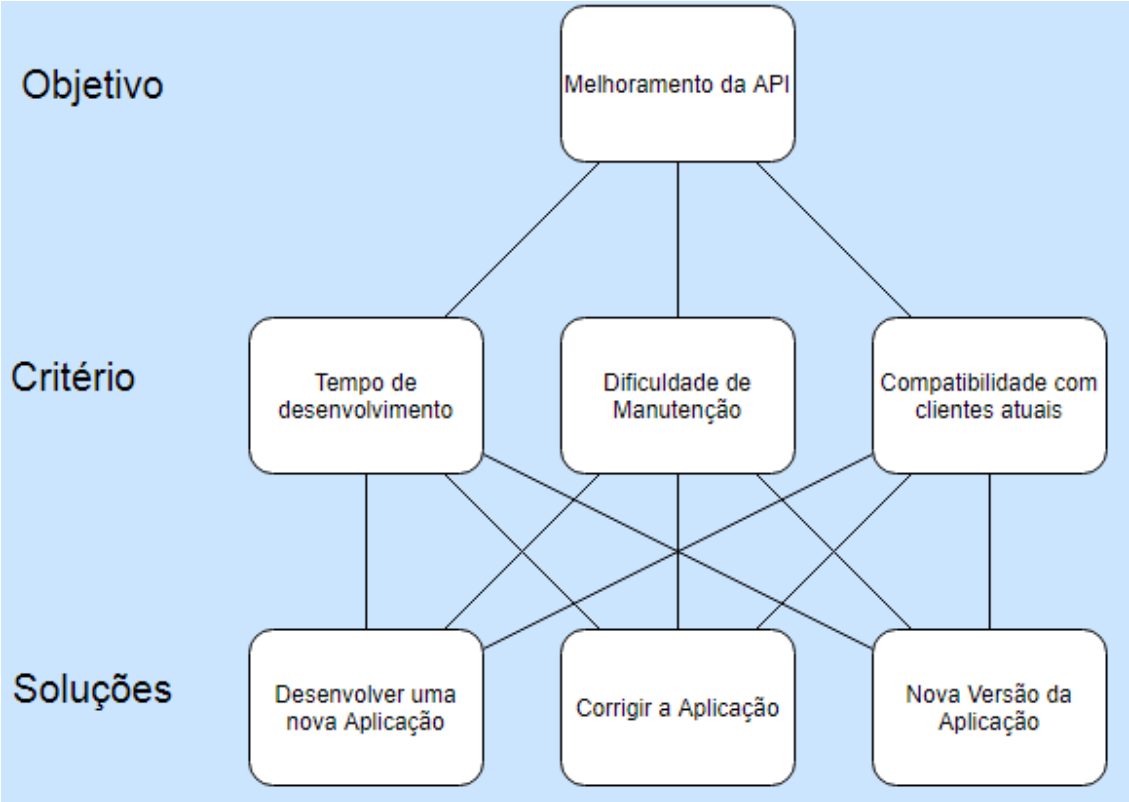

Figura 9 - Estrutura Hierárquica do AHP

Definido o diagrama da Figura 9, reúnem-se as condições para dar início a aplicação do método AHP.

3.2.2 Definição de Prioridades

Definidas os critérios e soluções alternativas, pode dar-se início ao processo de priorização dos critérios. Este processo foi feito com ajuda da equipa responsável pela Slingshot, de forma a adequar os critérios as soluções propostas.

Primeiro faz-se a priorização das prioridades entre si. Este passo será útil para determinar quão mais importante um critério é em relação a outro. Os resultados desta comparação estão disponíveis na Tabela 4. A escala utilizada foi a de Saaty, onde o valor 1 representa igualdade de prioridade e o valor 9 representa a maior grau de prioridade (Saaty, 2008).

Tabela 4 - Cálculo da prioridade relativa dos critérios

	A	B	C	Prioridade Relativa
A	1	0.2	0.14	0.07
B	5	1	0.33	0.28
C	7	3	1	0.64

Os dados que se encontram na Tabela 4 mostram que o critério C é o mais prioritário, e o critério A o menos prioritário. Uma vez que a aplicação funciona atualmente, não existe incentivo em arriscar quebrar compatibilidade com os clientes atuais. Por outro lado, as condições de estágio curricular do projeto são propícias a experimentação, pelo que o investimento de recursos no desenvolvimento do projeto não é relevante para a E-goi.

Após definidas as prioridades, compraram-se cada uma das soluções em cada um dos critérios. A Tabela 5 apresenta o calculo para o critério A, a Tabela 6 para o critério B, e a Tabela 7 para o critério C.

Tabela 5 - Cálculo da prioridade relativa de cada solução dado critério A

	X	**Y**	**Z**	Prioridade Relativa
X	1	0.33	0.2	0.12
Y	5	1	4	0.65
Z	3	0.25	1	0.23

A Tabela 5 mostra que, em termos de tempo de desenvolvimento, a solução Y é a preferível. Este resultado tem sentido, uma vez que a aplicação já existe e é funcional, pelo que o investimento de recursos é menor para atingir os objetivos e, portanto, preferível.

Tabela 6 - Cálculo da prioridade relativa de cada solução dado critério B

	X	**Y**	**Z**	Prioridade Relativa
X	1	1	0.33	0.23
Y	1	1	7	0.50
Z	3	0.14	1	0.26

A Tabela 6 mostra que, dada a manutenção da aplicação, continua a ser preferível a solução Y. Tal como no critério A, a aplicação é funcional, e o trabalho de manutenção da mesma faz parte da rotina da equipa do Slingshot e da E-goi.

Tabela 7 – Cálculo da prioridade relativa de cada solução dado citério C

	X	**Y**	**Z**	Prioridade Relativa
X	1	0.2	0.14	0.07
Y	5	1	0.11	0.20
Z	7	7	1	0.73

A Tabela 7 mostra que, em termos de compatibilidade, a solução Z é preferível. Este resultado justifica-se pelo risco que existe em quebrar compatibilidade nas soluções X e Y.

Depois de calculadas as prioridades relativas de cada solução face os critérios, os resultados são agregados na Tabela 8.

Tabela 8 - Prioridade relativa de cada alterativa para cada critério

	A	B	C
X	0.12	0.23	0.07
Y	0.65	0.50	0.20
Z	0.23	0.26	0.73

Para terminar a etapa de definição de prioridades, é necessário atribuir uma prioridade geral a cada uma das soluções, face a prioridade em cada critério e o peso geral do mesmo. Desta forma, multiplica-se a matriz presente na Tabela 8 pela matriz presente na Tabela 9.

Tabela 9 - Prioridade Relativa de cada critério

	Prioridade Relativa
A	0.07
B	0.28
C	0.64

O resultado está presente na Tabela 10.

Tabela 10 - Peso Global de cada solução

	Peso Global
X	0.12
Y	0.31
Z	0.56

Os resultados presentes na Tabela 10 mostram que a solução Z é a ideal, dadas as soluções e os critérios que foram tidos em conta. Assim sendo, a solução ideal para o problema deverá ser a do desenvolvimento de uma nova versão da API da Slingshot.

3.2.3 Consistência

Para validar a decisão tomada é necessário garantir que as prioridades definidas são válidas. Para tal deve garantir-se que o rácio de consistência é menor que 0,1. Primeiro, normalizam-se os valores presentes na Tabela 4, de forma a que a soma das colunas seja igual a um, e somam-se os valores das linhas (que devolve o valor de Eigen). Depois deste passo, multiplica-se a matriz normalizada pela matriz composta pelos valores de Eigen. A matriz resultante encontra-se na Tabela 11.

Tabela 11 - Cálculo do λ para cada critério

Critério	λ
A	0.21
B	0.84
C	1.97

Após calcular o Lambda, é necessário relacionar com a prioridade dada a cada um dos critérios. Assim sendo, calcula-se a média do valor de lambda sobre a prioridade global.

$$\lambda max = \frac{\frac{0.21}{0.07} + \frac{0.84}{0.28} + \frac{1.97}{0.64}}{3} \approx 3.026$$

Definido o λmax, pode calcular-se o índice de consistência, utilizando a seguinte formula:

$$IC = \frac{\lambda max \ - \ n}{n - 1}$$

Substituindo as variáveis da equação pelos respetivos valores, têm-se o seguinte resultado:

$$IC = \frac{3.026 - 3}{3 - 1} \approx 0.013$$

Uma vez que o índice de consistência é menor que 0.1, temos condições para continuar e calcular o rácio da consistência (RC), utilizando a seguinte equação:

$$RC = \frac{IC}{IA}$$

O valor de IA, ou índice aleatório, é um valor predefinido por Saaty, onde ao número de critérios se atribui um valor.

Tabela 12 - Índice Aleatório de Saaty

n	1	2	3	4	5	6	7	8	9	10
IA	0	0	0.58	0.9	1.1	1.24	1.32	1.41	1.45	1.49

Como existem 3 critérios, têm-se que n = 3, e, substituindo as variáveis pelos respetivos valores, obtêm-se o seguinte resultado:

$$RC = \frac{0.013}{0.58} \approx 0.0224$$

Como 0.0224 < 0.1, consideram-se os dados consistentes, e o resultado é valido. Desta forma, valida-se que o desenvolvimento de uma nova versão para a Slingshot será a solução ideal.

3.3 Business Model Canvas

Para enquadrar a solução encontrada no contexto de negócio da E-goi, foi desenhado o Business Model Canvas. Este modelo permite reduzir a componente do negócio relacionado com a nova versão da Slingshot a um único diagrama.

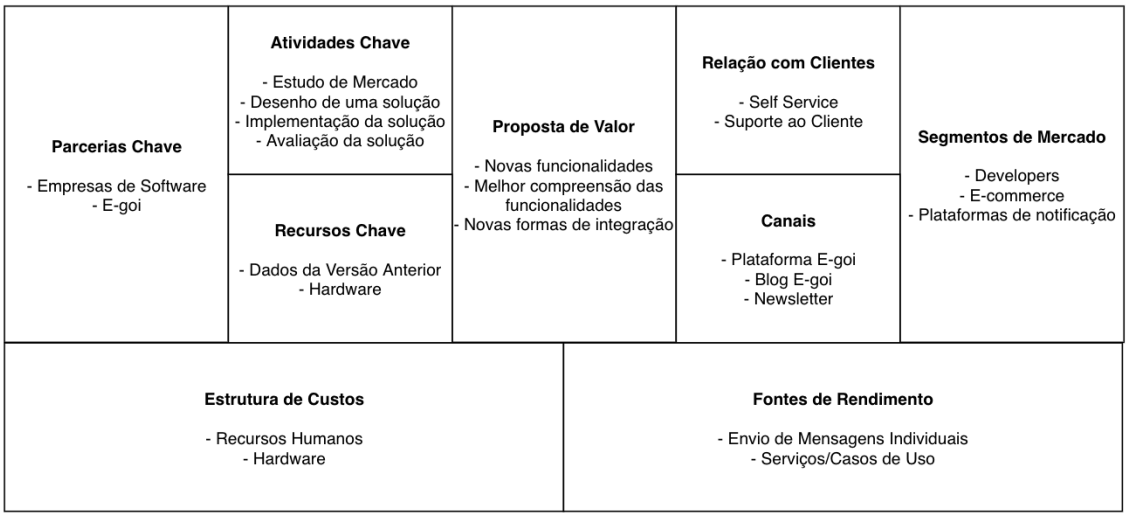

Figura 10 - Business Model Canvas da nova versão da Slingshot

As parcerias chaves são a própria E-goi e as empresas que mantenham a tecnologia que a nova versão irá utilizar. Da mesma forma que a Slingshot é um produto a cargo da E-goi, usufruindo da infraestrutura e dados da empresa, a nova versão terá as mesmas regalias. Outro tipo de

parcerias será feito com as empresas de *software* que disponibilizam as componentes que a nova versão irá utilizar.

Muitas das componentes do Canvas são semelhantes aquilo que é a Slingshot, como o segmento de mercado, os canais de comunicação, a estrutura de custos, as fontes de rendimento e a relação com o cliente. Estas observações mostram que a nova versão deverá encaixar sem dificuldades no mesmo contexto da Slingshot.

A definição da proposta de valor será melhor descrita na secção 3.4.

3.3.1 Análise SWOT

Para auxiliar na identificação das forças e fraquezas da nova versão foi efetuada a análise SWOT, que analisa as forças (*Strengths*), fraquezas (*Weaknesses*), Oportunidades (*Opportunities*) e ameaças (*Threats*). Este tipo de análise divide a força e fraqueza no contexto interno e externo a organização. O contexto interno relaciona-se com a dinâmica dentro da organização, enquanto que o contexto externo se relaciona com o ambiente em que a organização se insere. Para este último contexto pode ser realizada uma análise PESTEL, que analisa 6 áreas do ambiente (Política, Económica, Social, Tecnológica, Ecológica, Legal).

Uma vez que se têm uma proposta de solução definida, pode fazer-se a análise SWOT ao desenvolvimento de uma nova versão da Slingshot, face ao contexto onde se irá inserir. Esta análise é apresentada na Tabela 13.

Tabela 13 - Análise SWOT da nova versão da API

	Vantagens	Desvantagens
Interna	**Forças** • Integração com a plataforma E-goi; • Desenvolvimento de novas funcionalidades; • Disponibilização potencial de novas formas de interação com a plataforma; • Utilização da base de dados da versão anterior; • Diminuição da curva de aprendizagem da API por parte do utilizador. • Duas versões oferece versatilidade e redundância.	**Fraquezas** • Dependência na versão anterior da API; • Gestão de API's exige esforço na manutenção e consistência entre versões; • Manutenção da comunicação entre as duas versões;
Externa	**Oportunidades** • Necessidade de envio de mensagens transacionais por parte das organizações; • Venda da ferramenta em mercado nacional e similares; • Estudo de novos canais e casos de uso;	**Ameaças** • Desenvolvimento de funcionalidades da concorrência; • Alterações a convenções de programação; • Ameaças de segurança (DoS, Injection...);

A análise a Tabela 13 permite verificar que a nova versão possui várias vantagens. As principais vantagens da nova versão são fruto da associação com a E-goi, que diminui o custo de integração da ferramenta na sua oferta. No entanto, existem algumas desvantagens, uma vez que comporta maiores custos de manutenção. Em relação ao contexto externo, verifica-se que a análise também se aplica a Slingshot atualmente, uma vez que, sendo uma nova versão, serão produtos semelhantes.

3.4 Proposta de Valor

Para facilitar a definição da proposta de valor será utilizado o modelo *Value Proposition Canvas*. Este modelo permite apontar as necessidades esperadas do cliente, e conjugá-la com a oferta que a organização pretende dar. Nasce como a relação isolada entre a proposta de valor e o segmento de mercado do *Business Model Canvas* (Osterwalder, Pigneur, Bernarda, & Smith, 2014). Assim, o modelo, presente na Figura 11, é dividido em duas áreas: o Perfil do Cliente, e a Proposta de Valor.

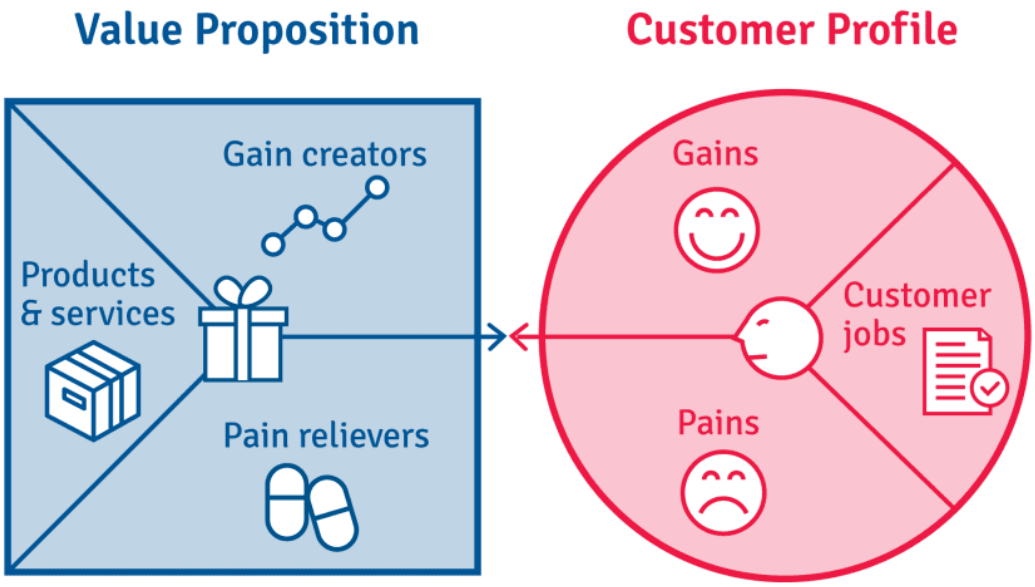

Figura 11 - Value Proposition Canvas (Moura, 2018)

O perfil do cliente descreve todas as caraterísticas e necessidades percetíveis pela organização do seu mercado alvo. Neste caso será tido em conta o perfil dos clientes atuais e futuros da Slingshot, uma vez que a nova versão da API é um produto que deverão ter interesse. Dentro do perfil do cliente surgem três perspetivas: Tarefas, Ganhos e Dores. Tarefas descrevem as ações que os clientes procuram fazer. Podem ser tarefas como aumentar o seu desempenho, atingir objetivos, realizar ações ou melhorar o seu estado de espírito. Dores descrevem as dificuldades sentidas pelos clientes. Aqui se inserem as componentes negativas dos clientes, em termos de custos, emoções, riscos, entre outros. Ganhos descrevem os benefícios que os clientes procuram atingir, onde se inserem todas as componentes positivas dos clientes, em termos de custos, emoções, entre outros. Com estes termos definidos, é apresentado como se adaptam a realidade da E-goi.

- Tarefas – Os clientes E-goi necessitam de enviar mensagens transacionais aos seus contactos. Também procuram formas de automatizar os seus processos, utilizando os casos de uso disponibilizados. Estas tarefas conjugam-se com o objetivo final de aumentar a confiança dos destinatários;
- Dores - Os clientes E-goi sentem alguma dificuldade em compreender e integrar a plataforma nas suas aplicações. Também a plataforma E-goi, por si, não permite enviar facilmente mensagens para apenas um destinatário. E sendo possível, os custos de tempo de envio e financeiros são relativamente altos. Também a análise da documentação disponível da plataforma Slingshot gera confusão. Finalmente, apenas existe uma forma da plataforma do cliente se ligar a API, que pode levar a problemas de comunicação e processamento;
- Ganhos - Os clientes da E-goi possuem a plataforma Slingshot, que lhes permite fazer o envio de mensagens transacionais. Este tipo de mensagens é mais simples e rápido que

a restante oferta da E-goi. Também o custo de envio das mensagens é menor, que diminui a despesa do cliente.

Com o perfil do cliente estabelecido, pode iniciar-se a definição da proposta de valor. Esta proposta é dividida em três partes: Produtos e Serviços, Aliviadores de Dores, e Criadores de Ganhos. Produtos e Serviços lista toda a oferta da organização que irá gerar o valor da proposta. São as ferramentas que permitem aos clientes atingir os seus objetivos. Aliviadores de dores descrevem como a oferta da organização ajuda o cliente a aliviar os seus problemas, diminuindo as emoções negativas sentidas pelos mesmos. Criadores de ganhos descrevem como a oferta cria vantagens e melhorias para o cliente, contribuindo para a satisfação na organização e na oferta. Com estes termos definidos, é apresentado como se adaptam a realidade da E-goi:

- Produtos e Serviços - Com a nova versão da Slingshot, os clientes terão opções no momento de integrar a plataforma com as suas aplicações, aumentando a perceção de valor da mesma (Le, 2014);
- Aliviadores de Dores - A inclusão da nova versão da Slingshot irá facilitar a compreensão da plataforma por parte dos clientes. Também irá reduzir o tempo de integração da mesma na plataforma do cliente, disponibilizando integrações para ligar a plataforma. Finalmente, a disponibilização de formas de integração reduz o risco de erros no uso da Slingshot, diminuindo perdas de lucro (Tricentis, 2017).
- Criadores de Ganhos - A nova versão da Slingshot será integrada com a base de dados da versão original da API, pelo que os clientes não perdem informação na transição. A utilização da nova API também permitirá que os pedidos feitos através dela não sejam perdidos em momentos de indisponibilidade da aplicação. Desta forma diminui-se a quantidade de não são processados (Celar, Mudnic, & Seremet, 2016).

4 Análise da Plataforma

Neste capítulo é apresentado o estado atual da Slingshot. Será feita uma breve descrição da sua história, seguida de uma análise as suas funcionalidades e conceitos. Após esta análise serão apresentadas as limitações da Slingshot, de forma a contextualizar e justificar o objetivo do presente projeto.

4.1 A plataforma Slingshot

Em preparações para os Jogos Olímpicos de 2016 no Rio de Janeiro, o Comité Organizador e a Atos, a sua fornecedora tecnológica, depararam-se com incapacidades técnicas por parte da ferramenta que utilizavam para enviar mensagens transacionais. Por este motivo decidiram abandonar a ferramenta, e desafiar a E-goi a desenvolver uma solução capaz de responder aos seus requisitos, quer de funcionalidade, quer de volume de processamento.

Com o prazo definido, a E-goi iniciou o desenvolvimento de uma plataforma de envio de mensagens transacionais, de raiz e com recursos limitados. O produto final deste desafio foi a plataforma Slingshot, que permite enviar mensagens transacionais através de Email e Sms. A plataforma cumpriu com os objetivos que foram propostos, quer em termos de tempo de desenvolvimento, quer em termos de funcionalidade. Os resultados da Slingshot no contexto dos Jogos Olímpicos são apresentados na Figura 12.

Figura 12 - Resultados da Slingshot durante os Jogos Olímpicos de 2016 (E-goi, 2016)

Os resultados apresentados na Figura 12 aliciaram a E-goi a disponibilizar a plataforma para os seus clientes, promovendo uma vertente alternativa de envio de mensagens de marketing aos seus clientes. A sua base ideal de clientes seria as que tivessem o seu negócio online, como *e-commerces*, visto que teriam maior necessidade de enviar mensagens do tipo transacional. Com esta base, o interesse e utilização da Slingshot foi aumentando com o tempo, expandindo para outros tipos de clientes, como centros de saúde interessados em lembrar os seus utentes das suas consultas.

Também a equipa que a desenvolvia foi crescendo, de forma a acompanhar a evolução do mercado. Neste momento a equipa é composta por 3 elementos, sendo que a equipa original não faz parte da constituição atual. A equipa é responsável pela manutenção das funcionalidades da Slingshot, acrescentando valor com novas funcionalidades, e corrigindo erros de processamento.

Após a apresentação da história da Slingshot, deve fazer-se uma análise das suas componentes técnicas. A estrutura atual da API está presente na Figura 13. Uma vez que a Slingshot foi desenvolvida como um produto à parte da base de negócio da E-goi, comprova-se que os componentes que a compõem estejam num contexto diferenciado. No entanto, como os clientes da Slingshot são clientes E-goi, é necessário estabelecer uma ligação entre os dois, para fins de configuração ou calculo de custos.

Uma conclusão que se pode tirar da Figura 13 é que, neste momento, a Slingshot permite o envio de mensagens através de notificações Push. Este tipo de mensagens é associada a uma aplicação, enviando notificações diretamente para o telemóvel dos seus destinatários.

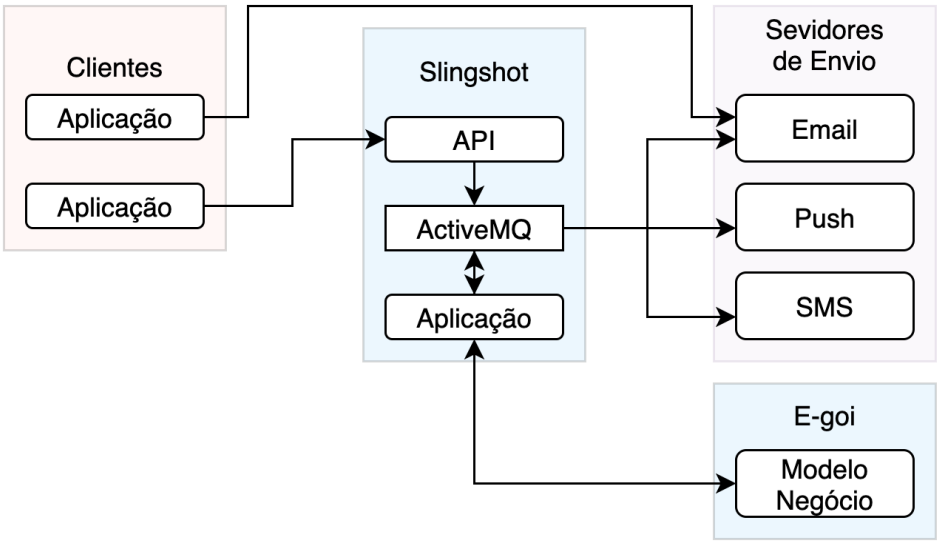

Figura 13 - Estrutura do Slingshot

De acordo com a Figura 13, a Slingshot atua como intermediário entre os seus utilizadores, a E-goi e os servidores de envio. Cada canal que é integrado na Slingshot possui o seu próprio servidor de envio, integrando a tecnologia necessária ao envio das mensagens. Desta forma, as mensagens que serão enviadas podem ser processadas pela E-goi, validando o seu conteúdo e aplicando o processamento necessário.

Finalmente, o servidor de envio de emails pode ser diretamente acedido através de *Simple Mail Transfer Protocol* (SMTP), permitindo o envio de mensagens mais rapidamente. No entanto, o seu envio poderá ser mais limitado, uma vez que a Slingshot não consegue implementar algumas funcionalidades como registo de dados.

Dentro das funcionalidades da Slingshot podem discernir-se algumas áreas e conjuntos de funcionalidades. Estas estão apresentadas no mapa de conceitos da Figura 14.

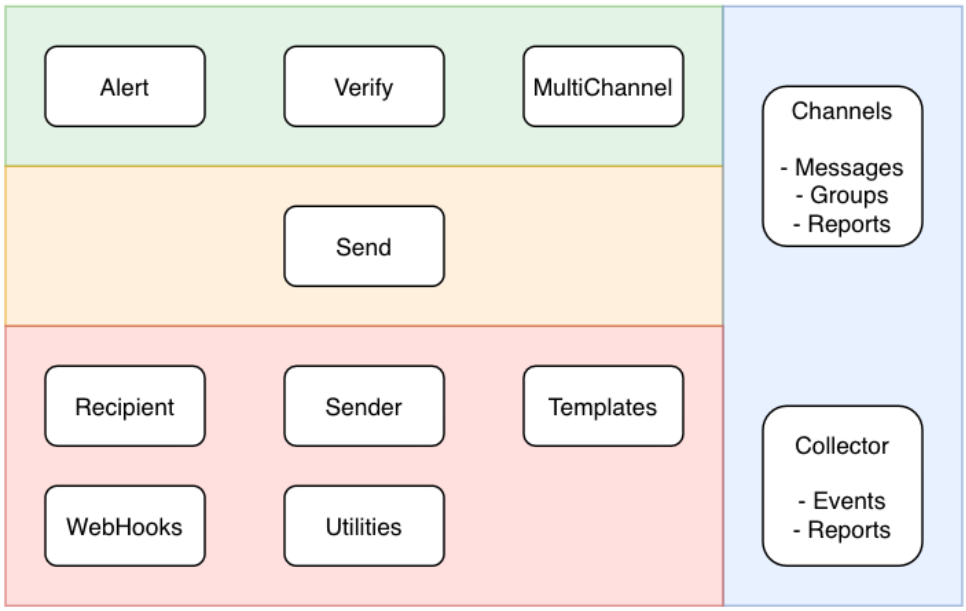

Figura 14 - Mapa de conceitos da Slingshot

Os conceitos da Slingshot dividem-se em quatro grandes agrupamentos:

1. Casos de Uso (a verde) – processos de envio de mensagens transacionais, com estruturas bem definidas e automatização do seu envio;
2. Envio de mensagens (a amarelo) – o caso de uso que é a base da Slingshot, permite enviar mensagens de forma pontual;
3. Configuração (a vermelho) – conjunto de funcionalidades de gestão de informação definida previamente ao envio de mensagens;
4. Resultados (a azul) – Agregação de informação relativa as mensagens que foram enviadas. São auxiliadas pela presença de um Collector, que recolhe dados e eventos sobre as mensagens que são enviadas pela plataforma.

As componentes que compõem os agrupamentos serão exploradas na seção 4.1.1 - Capacidades e Funcionalidades.

4.1.1 Capacidades e Funcionalidades

A Slingshot permite, através da sua API, a configuração e o envio de mensagens transacionais. A plataforma permite o envio de mensagens de email e SMS, com planos para integrar notificações Push para aplicações moveis.

4.1.1.1 Configuração

Antes do envio de qualquer mensagem, pode existir a necessidade de configurar opções como remetentes ou domínios. Estas configurações serão depois aplicadas as mensagens enviadas através da Slingshot, para que o cliente possa adaptar a mensagem as suas necessidades e capacidades.

Estas configurações serão validadas por parte da Slingshot, garantindo não só a qualidade da informação inserida como a integridade do que ela representa. Por exemplo, ao definir um remetente de email, este é validado utilizando validações como *Sender Policy Framework* e *Domain-based Message Authentication, Reporting & Conformance*.

A Slingshot permite ainda fazer a gestão de *Templates* de mensagens para cada canal que disponibiliza. Estes Templates serão depois referenciados no envio da mensagem, de forma a substituir as configurações da mensagem pelas suas. Assim o utilizador não necessita de definir cada mensagem para cada envio, podendo apenas escolher o *Template* que deseja e associar a informação necessária para adaptar a mensagem.

Outra funcionalidade que poderá ser configurada é o acompanhamento das mensagens através de *webhooks*. Estes são definidos como funções definidas pelo utilizador que são chamadas via HTTP (Atlassian, 2019). Na Slingshot, *webhooks* permitem enviar notificações via HTTP para o servidor do utilizador. Estas notificações são despoletadas quando a mensagem atinge um determinado ponto no seu ciclo de vida.

4.1.1.2 Envio de mensagens

A plataforma permite enviar mensagens através do recurso na API de cada canal. A API também permite enviar fazer vários envios num só pedido, encarando uma chamada a esse método como várias chamadas para cada mensagem individual. O utilizador pode ainda escolher se as mensagens são enviadas sequencialmente ou não.

Outro fator no envio que afeta o processamento de mensagens é o Registered. Esta funcionalidade permite que uma mensagem e respetivo ciclo de vida seja armazenado de forma segura num servidor estanque e imparcial. Esta mensagem apenas pode ser consultada em condições especiais, podendo ser utilizada como prova de consulta em situações legais.

No caso de emails, a Slingshot permite ainda integrar o servidor SMTP nas aplicações dos seus utilizadores. O servidor permite enviar diretamente o email da E-goi para o destinatário. Neste caso, o uso do Registered não é possível, uma vez que o processamento necessário para tal não se encontra nos servidores de envio.

4.1.1.3 Casos de uso

Para simplificar a adoção da Slingshot, foram criados alguns casos de uso que os clientes sintam necessidade de automatizar. A Slingshot disponibiliza três casos de uso para este efeito: Verify, Alerts e Multi-channel.

Verify é a versão da E-goi de *Two Factor Authentication* (2FA). Com esta funcionalidade a E-goi permite atuar como intermediário imparcial entre dois utilizadores. Na Figura 15 é apresentado um diagrama de sequência de um caso de uso da funcionalidade. Neste caso, a Slingshot atua como intermediário entre a aplicação do cliente e o destinatário a validar. Uma vez que a aplicação não conhece o código que foi gerado, também não sabe como validá-lo, pelo que deposita a sua confiança na Slingshot para o fazer.

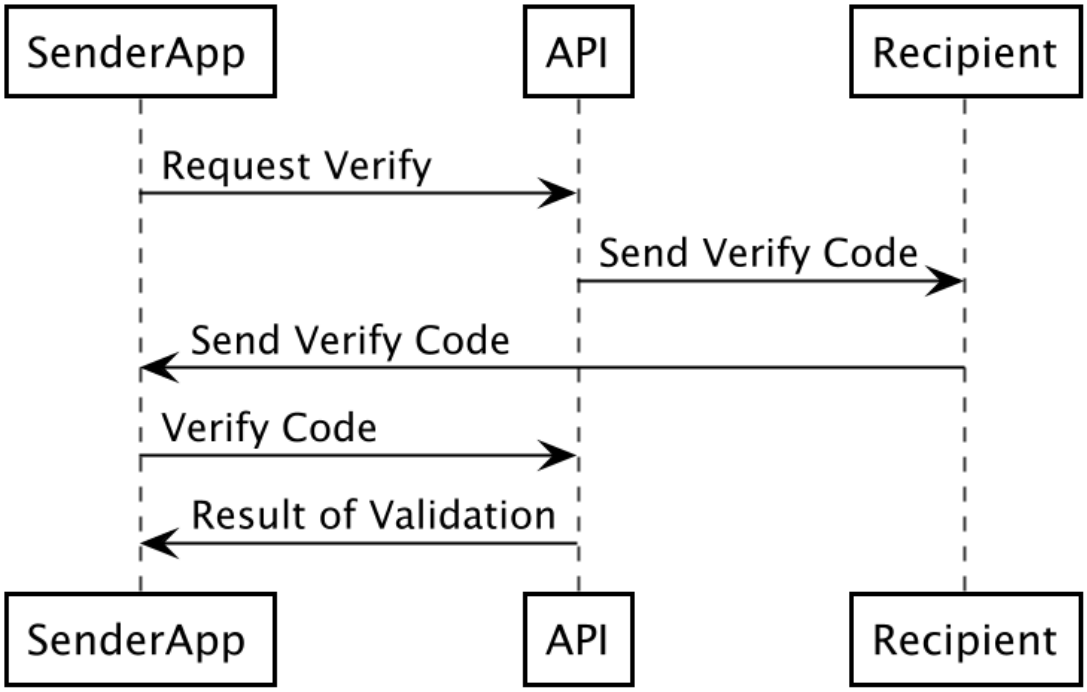

Figura 15 - Diagrama de Sequência de um pedido 2FA

Os objetivos que a aplicação de 2FA se propõe são os mesmos da funcionalidade do Verify: acrescentar uma camada de validação, em que é exigido do utilizador duas formas de autenticação. Desta forma evitam-se abusos de utilização por parte dos clientes que a integram, e até evitando ataques do tipo *phishing*.

Alerts consistem no envio periódico de uma mensagem a um ou vários destinatários. A mensagem possui um link que, uma vez aberto, quebra o ciclo de envio. O ciclo também poderá ser quebrado atingindo o número máximo de envios, como mostra a Figura 16. O cenário ideal é apresentado com a cor verde, a azul estão apresentadas as alternativas de processamento, e a vermelho o pior cenário.

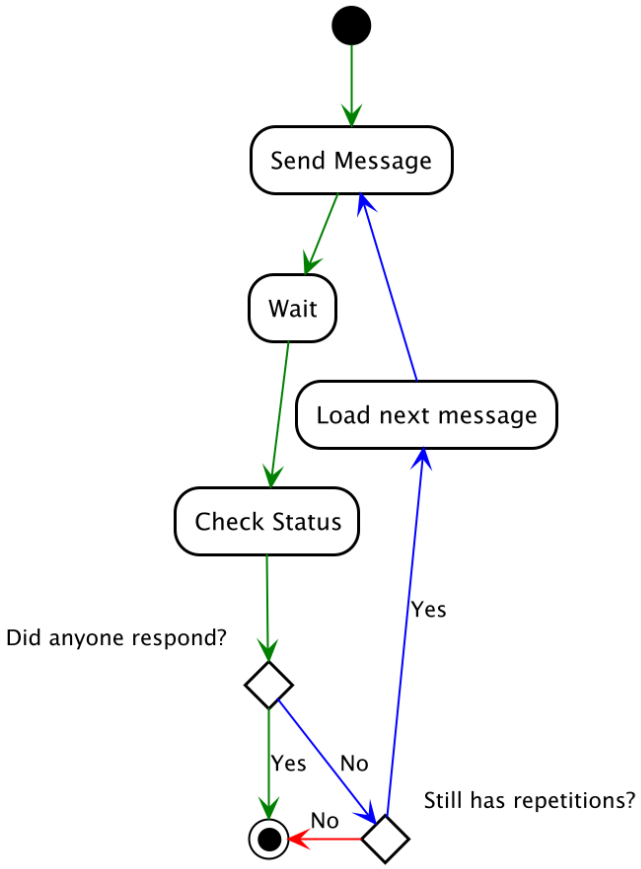

Figura 16 - Fluxo de processamento de um Alert

O passo de "Check Status" apresentado na Figura 16 avalia se o Alert foi interrompido. O resultado deste passo pode terminar o Alert se verificado que o link foi aberto. Caso contrário, verifica se tem ainda iterações para enviar mensagens. Por cada iteração diminui o número destas, até que se esgotem. Neste caso, o fluxo é interrompido.

Multi-channel permite ao utilizador predefinir uma sequência condicional de mensagens, onde o estado da anterior num determinado espaço de tempo despoleta o envio da seguinte. Esta sequência permite enviar mensagens de qualquer canal de comunicação disponível na Slingshot. Uma vez definida, a sua ativação poderá ser despoletada através do envio do destinatário e da informação necessária para apropriar a mensagem ao mesmo. Desta forma, uma sequência poderá ser reutilizada na mesma situação para vários destinatários.

O processamento de um Multi-channel é apresentado na Figura 17. Tal com na Figura 16, a verde é apresentado o melhor cenário e a azul as alternativas. Neste caso, não se considera o pior cenário possível, uma vez o Multi-channel oferece flexibilidade para se ajustar aos casos que os seus utilizadores pretendam implementar.

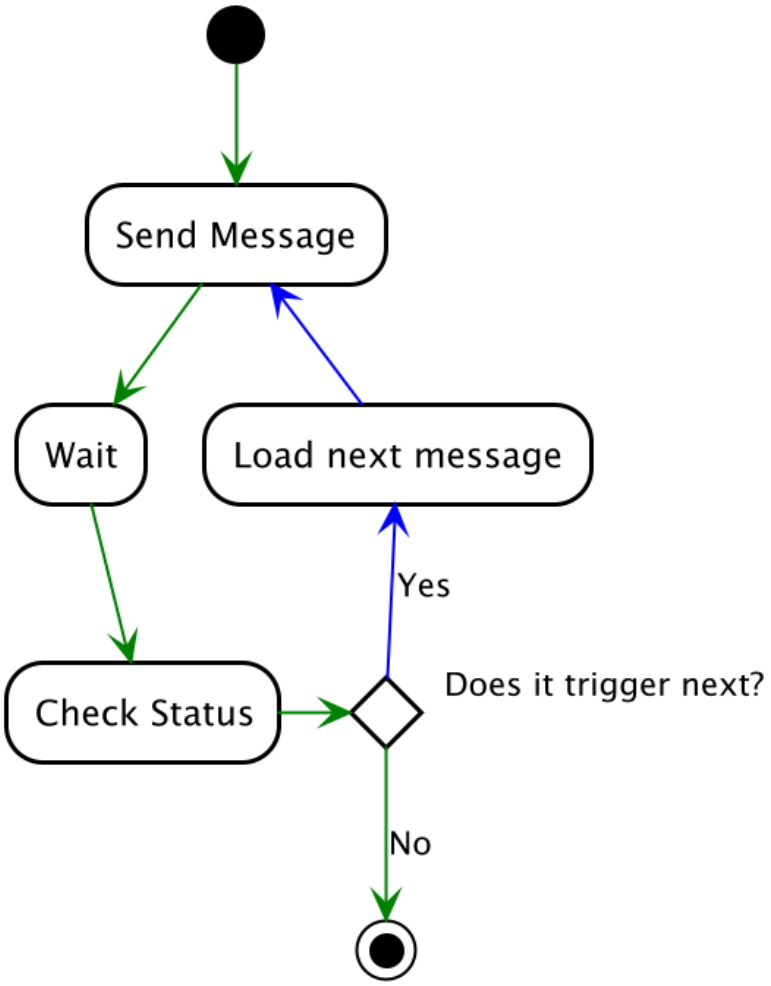

Figura 17 - Fluxo de um Multi-channel

A Figura 17 mostra como a sequência de mensagens é processada. Por cada iteração é enviada uma mensagem para um destinatário. Após o tempo definido, é avaliado o estado da mensagem. Se o estado permitir enviar a mensagem seguinte, então é enviada a mensagem e repete-se o processamento. Caso contrario, o Multi-channel é interrompido.

4.1.1.4 Resultados

Após o envio das mensagens, é possível ao utilizador gerar relatórios sobre as estatísticas de envio das suas mensagens, desde percentagem de envios com sucesso até taxa de abertura e interação.

A aplicação também permite a consulta do estado de mensagens especificas. O utilizador poderá consultar se uma mensagem foi enviada com sucesso, quanto tempo demorou a ser aberta, e qual a informação que foi enviada.

4.1.2 Limitações

De forma a desenhar a nova versão da API é necessário fazer uma análise sobre o estado da API atual, a fim de descobrir como melhorar. Para tal, será estudada principalmente a

documentação da API, de onde poderemos analisar as caraterísticas de cada método, e verificar-se são pontos de dificuldade.

Um dos pontos mais recorrentes durante a análise das limitações da plataforma prende-se com inconsistências. Os métodos e a própria documentação dos mesmos não são completamente coerentes entre si. Apesar de, neste caso, o impacto delas ser relativamente baixo, a presença de inconsistências pode ter fortes efeitos negativos (Mahato, Dudhal, Revagade, & Bhargava, 2019), pelo que deve ser questão de princípio identificá-las para corrigi-las no futuro.

4.1.2.1 Agrupamento de métodos

Um dos pontos que levanta muitas questões nos clientes é o agrupamento de serviços. Deve ser esperado que todos os serviços dentro de um grupo estejam relacionados com o título do grupo. No entanto, um serviço que realiza uma ação pode não corresponder ao conceito do grupo em que se insere. Uma das vertentes deste ponto pode assinalar-se quando um grupo é demasiado abrangente. Aqui fica o exemplo do conceito de *Domain*, importante para definição de domínios de email. Apesar da sua importância, todos os métodos (6 ao todo) inserem-se no grupo Mail, apresentado na Figura 19. Esta situação cria um grupo de serviços demasiado extenso e diverso, que leva a confusão por parte do utilizador.

Outra vertente deste problema é quando o conceito do grupo não descreve o conceito dos métodos que o compõem. Exemplo disso é o grupo *Stats*, que contem apenas um método que retorna a lista de grupos de um determinado canal de comunicação. O conceito de estatística implicado pelo nome do grupo não corresponde aos grupos devolvidos pelo método, levantando confusão para o utilizador.

Figura 18 - Grupo *Stats* da API

Levando esta problema ao extremo, tem-se casos em que a representação do conceito do grupo não corresponder a realidade da aplicação. Por exemplo, o conceito de *Campaign* existe na E-goi como uma mensagem que é enviada para vários contactos. Em contexto da transacional, este conceito não existe, pelo que a nomenclatura do grupo não faz sentido.

Finalmente, a ordenação dos grupos não é coerente com o grau de investimento do utilizador. Por exemplo, o grupo *Client* está relacionado com a definição da relação entre o utilizador e a API. Neste grupo existem apenas penas possui dois métodos. O primeiro permite ao utilizador ativar a sua Apikey para comunicar com a API. O segundo permite consultar qual o estado da conta. Tanto o primeiro como o segundo terão mais utilização numa primeira fase de utilização da API, pelo que deveriam ter um destaque diferente na documentação da API, podendo estar apresentado mais acima na hierarquia do grupo, ou referenciando na introdução inicial da documentação. No entanto, este grupo surge nomeio de outros grupos, sem critério de ordenação.

4.1.2.2 Serviços

Também alguns serviços em si dificultam a compreensão do utilizador. Existem vários problemas de coerência e compreensão que são levantados, principalmente a nível de documentação. Na Figura 19 é apresentado o grupo *Mail*.

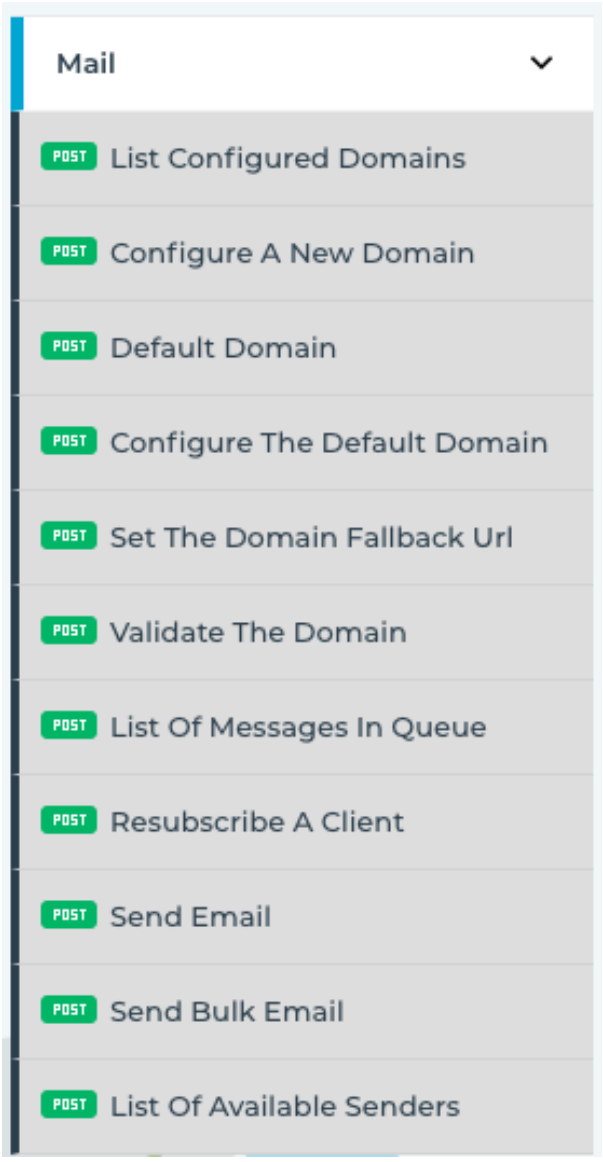

Figura 19 – Serviços do Grupo Mail

Um dos principais problemas prende-se com o método HTTP que é utilizado ao chamar o serviço. Aquando o seu desenvolvimento, todos os métodos da Slingshot foram desenvolvidos utilizando o verbo *Post*. Esta opção não é incorreta, uma vez que, inicialmente, a interpretação do problema foi que todos os pedidos seriam um pedido com dados associados. Não só isso, mas também se assumiu que todos os pedidos levariam a processamento de dados de forma que não retornasse necessariamente informação de consulta. No entanto, para implementação de uma API RESTful, esta acaba por ficar no nível 1, de acordo com a Figura 7. Este nível é considerado relativamente baixo, e não vai de encontro com as normas atualmente aplicadas na indústria.

Analisando os métodos na Figura 19, é possível verificar que, em alguns casos, o método HTTP não corresponde ao método utilizado na descrição do método. Por exemplo, o método "List of Messages in Queue" pressupõe a consulta das mensagens que estão fila de espera para envio. No entanto, o verbo HTTP associado é o Post, que sugere envio de informação e processamento da mesma.

Apesar disto, os serviços acrescentados após o lançamento da Slingshot apresentam os métodos HTTP ideais para o serviço a que estão associados. Exemplo disso é o grupo dos Templates apresentado na Figura 19, onde os métodos HTTP correspondem à ação de gestão de informação esperado.

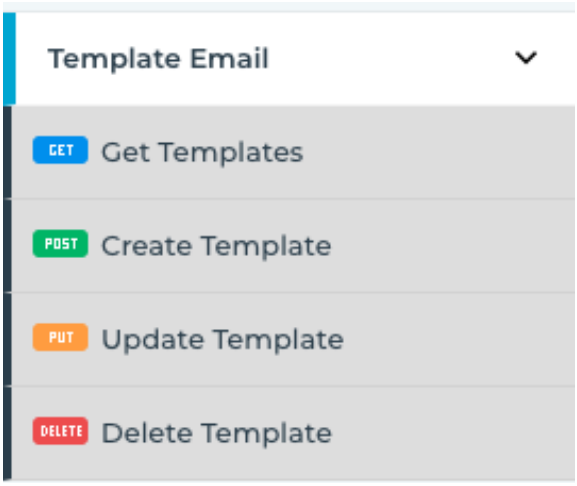

Figura 20 – Métodos do grupo Template Email

De forma a passar do panorama da Figura 19 para a Figura 20, iniciou-se o processo de criação de novos métodos que conformem com os princípios REST, e descontinuando os anteriores, como mostra a Figura 21.

Figura 21 - Exemplo de um método descontinuado, e seu substituto

Apesar de indicado na Figura 21, não existem garantias de que um utilizador adote um serviço em vez do outro. Num caso extremo, a disponibilização dos dois métodos pode levar o utilizador a integrar o método que lhe for mais comodo.

Por outro lado, apesar da correta utilização de mensagens de descontinuação poder acelerar o processo de adoção do novo método (Brito G. , Hora, Valente, & Robbes, 2016), estudos indicam que existe uma grande inercia no que toca a atualização de métodos (McDonnel, Ray, & Kim, 2013). Por este motivo o método não pode ser automaticamente removido, correndo o risco de quebrar compatibilidades com clientes que o utilizem.

Fazendo uma análise do contrato definido, guardou-se uma cópia no início do desenvolvimento do projeto. Ao inserir a definição do contrato num leitor próprio, verificou-se que existiam alguns modelos que não eram utilizados, mas que poderiam estar relacionados com os serviços existentes. Um destes casos é o modelo InternalRemoveInfo, apresentado na Figura 22.

Figura 22 - Modelo InternalRemoveInfo

Este modelo, descrito na Figura 22, poderá representar uma estrutura de dados que descreve os subscritores que foram removidos da API. Isto poderá ser informação útil para o utilizador, uma vez que poderá, entre outras coisas, filtrar emails inválidos no futuro. Apesar da utilidade deste objeto, a especificação é composta por serviços diferenciados para cada tipo de remoção, como mostra a Figura 23.

Figura 23 - Métodos de retorno de remoções

Os serviços da Figura 23, para além de apresentarem o problema de serviços descontinuados mencionado acima, apresentam redundância no objeto de retorno. De facto, os métodos são bastante semelhantes na sua funcionalidade. No entanto, o aumento da especificidade implica um aumento do número de serviços disponibilizados. Uma vez que a funcionalidade é semelhante, e retornam o mesmo objeto, pode analisar-se a possibilidade de simplificar e juntar serviços semelhantes. Assim, condensa-se o contrato e ocultar a complexidade da informação retornada.

4.1.2.3 Autenticação e Autorização

Uma questão importante é a autenticação e autorização do utilizador dentro da API. Neste momento, esta é feita através de uma apikey, que combina as duas funcionalidades. No entanto, o envio da mesma na mensagem não é consistente. Nos pedidos aos métodos originais, a apikey é enviada no corpo do pedido, enquanto que nos métodos mais recentes, principalmente pedidos de consulta, a apikey é enviada na *query*, ou seja, nos parâmetros após o endereço URL. Este tipo de inconsistência pode levar a confusão sobre qual o método a aplicar, e pode levar a erros de autenticação. A Figura 24 mostra um exemplo desta inconsistência.

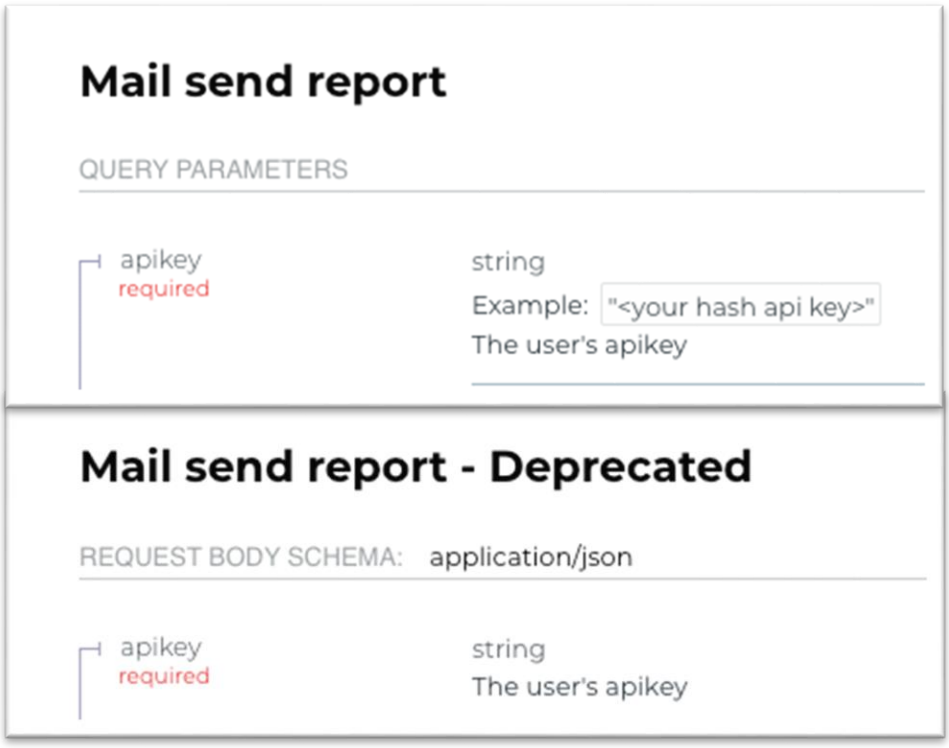

Figura 24 - Exemplos de formas de envio de Apikey

Apesar da Figura 24 mostrar versões diferentes do mesmo método, este é um exemplo da inconsistencia que existe nos métodos da API. Um cliente menos atento poderá não conseguir discernir onde colocar a ApiKey, impedindo-o de utilizar corretamente a API.

4.1.2.4 Exemplos de pedidos e respostas

A API está documentada utilizando o padrão OpenApi, que permite povoar os pedidos com exemplos sugestivos. A utilização de exemplos em vários contextos de uma aplicação permite orientar o utilizador até que este ganhe confiança para adaptar os exemplos ao seu contexto (Lee, 2018).

Começando pelos exemplos de respostas, alguns métodos não possuem estrutura de resposta apresentada na documentação, como mostra a Figura 25. Este tipo de situação é um problema, uma vez que a ausência de exemplos é uma das maioríssimas dificuldades sentidas por desenvolvedores ao utilizar uma API nova (Buse & Weimer, 2014).

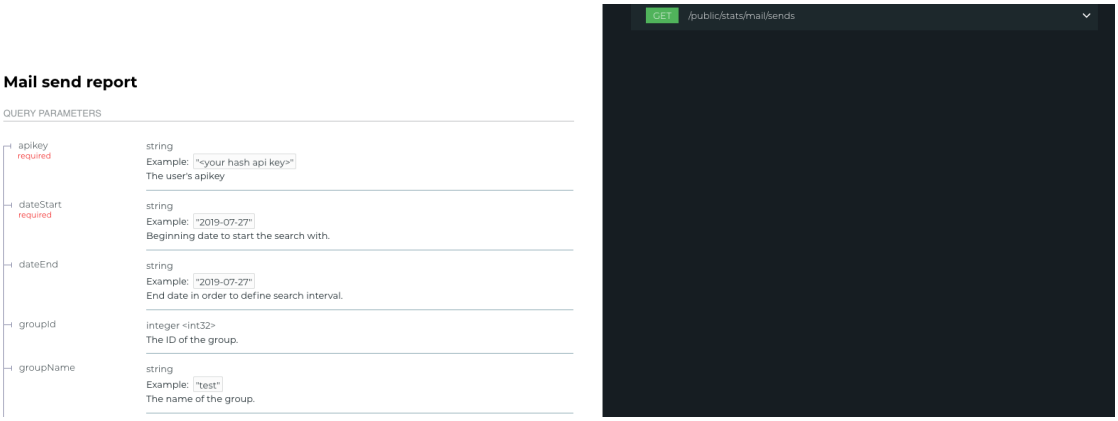

Figura 25 - Exemplo de método sem exemplos

A ausência de estrutura de resposta da Figura 25 levanta questões na altura de integração do método na aplicação do cliente. Uma vez que ele não sabe o que esperar, pode criar alguma insegurança no cliente, e obrigá-lo a experimentar antes de integrar o serviço. Por outro lado, no caso de existirem alterações na estrutura de resposta, o utilizador não tem como saber que a alteração foi aplicada, o que pode levar a erros de processamento por parte da aplicação do utilizador.

No entanto, é possível que a documentação apresente estrutura de resposta, como mostra a Figura 26.

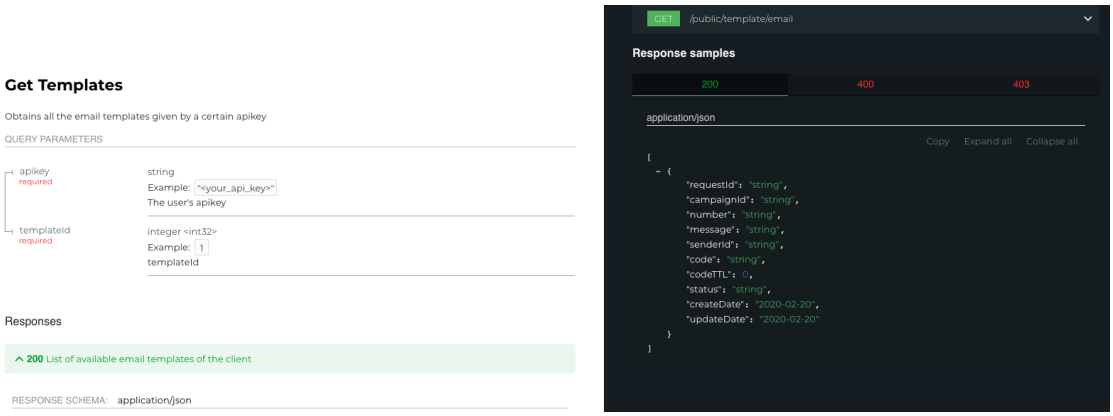

Figura 26 - Exemplo de método com estrutura de resposta

De facto, a Figura 26 incorpora tanto a estrutura de resposta como um pequeno exemplo do tipo de parâmetros. Este contraste com a Figura 25 não cria confiança para o cliente, que pode ter a ideia de que alguns métodos são privilegiados em relação a outros.

Outra falha na documentação da API é a documentação de possíveis erros durante a execução. Apesar da aplicação poder lançar erros no processamento de pedidos (como informação inválida), não existem sugestões de potenciais erros na documentação da API. Isto levanta alguns problemas para utilizadores da API, principalmente quando se torna necessário fazer processamento de erros (Aué, Aniche, Lobbezoo, & van Deursen, 2018). O utilizador terá

interesse em integrar na sua aplicação alternativas de processo em situações de erro, pelo que uma boa documentação, quer do código de erro, quer do tipo de erro, diminui a envolvência do mesmo na correção de situações indesejadas.

A Figura 27 mostra um método documentado da Slingshot que não apresenta respostas de erro.

Figura 27 - Exemplo de método sem respostas de erro

A documentação da Figura 27 não informa o utilizador sobre potenciais situações de erro no uso do método. Alguns dos erros que poderão ocorrer poderá ser a inexistência do recurso ou falta de autenticação. Em contraste, o serviço da Figura 28 apresenta alguns códigos de erro.

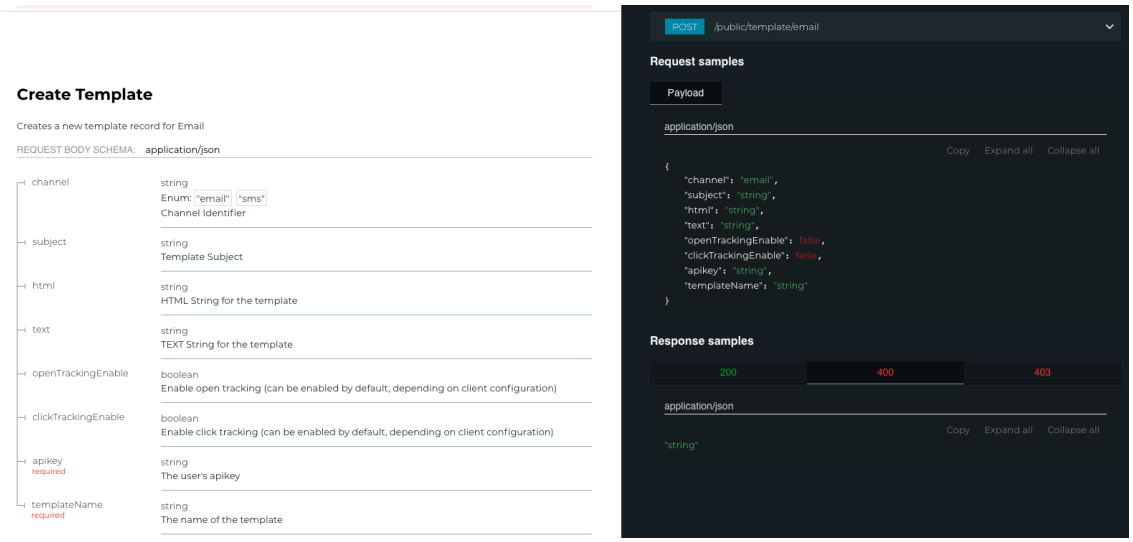

Figura 28 - Exemplo de método com respostas de erro

No entanto, os códigos presentes na Figura 28 não são satisfatórios por dois motivos. O primeiro motivo é porque não estão presentes todos os códigos que deveriam estar presentes, como o erro 500, indicando um problema com o servidor. O segundo motivo é a estrutura da mensagem de erro, que para além de não existir, não reflete a realidade de utilização da API.

Todas estas limitações técnicas levam o cliente a não conseguir prevenir potenciais situações de erro, aumentando a instabilidade da sua aplicação.

4.2 Mercado

Tendo definido o contexto da Slingshot, procura-se ferramentas e plataformas semelhantes no mercado. Assim torna-se possível verificar como a Slingshot se compara com ferramentas adversárias, identificando limitações de funcionalidade e como pode destacar-se no mercado.

4.2.1 Produtos e Serviços Concorrentes

Uma vez que a E-goi atua em várias áreas dentro do marketing digital, sujeita-se a concorrência em diferentes frentes de negócio. Por este motivo, algumas das organizações mencionadas nesta secção não terão o seu foco na oferta que será explorada. Em alguns casos, a oferta em si esta dividida num negócio a parte, havendo distinção entre a ferramenta e a organização que a detém. O foco em casos destes será na ferramenta e na documentação publicamente disponibilizada.

4.2.1.1 Mandrill (MailChimp)

A MailChimp é uma plataforma de automação de marketing, criada em 2001 nos Estados Unidos. Neste momento a MailChimp separa os seus produtos em duas categorias: Marketing e Transacional. O primeiro permanece sobre a alçada da MailChimp, enquanto que o segundo faz parte de um produto à parte, denominado Mandrill. Este último será o foco da análise desta plataforma.

Enquanto que o MailChimp oferece várias plataformas de marketing, o Mandrill apenas fornece comunicações transacionais via mensagens de email. Este é um serviço complementar do MailChimp, e obriga o utilizador a fazer a gestão de ambos os serviços.

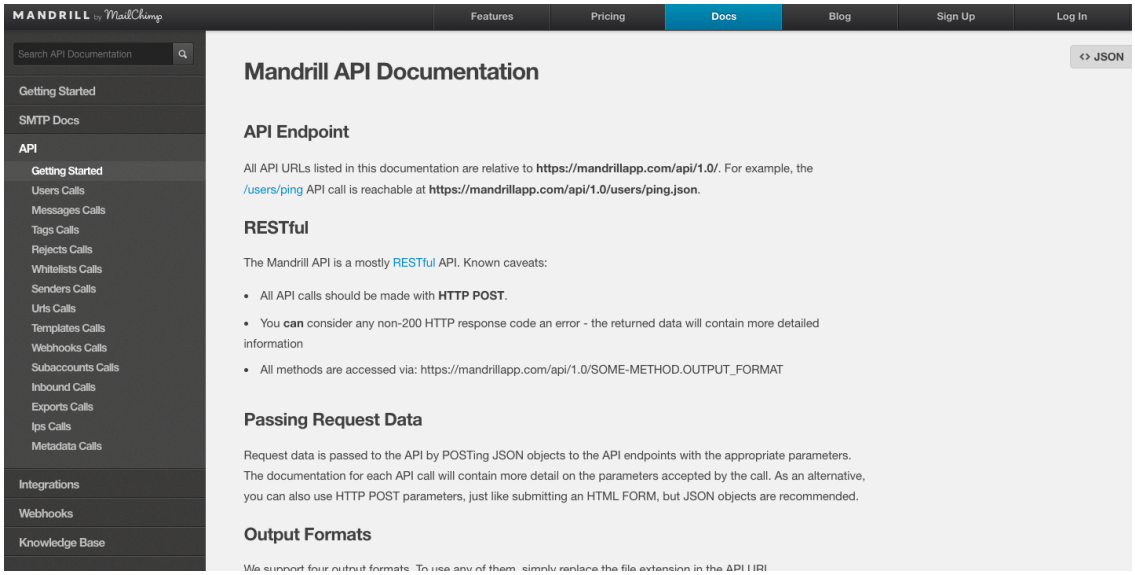

Figura 29 - Documentação da API da Mandrill

O Mandrill disponibiliza várias integrações complementares a API, como NodeJs ou PHP, para além de referenciar várias outras integrações desenvolvidas por terceiros, como Java ou .NET.

4.2.1.2 Mail Jet

Mail Jet é uma plataforma de envio de emails de marketing e transacionais, criada 2010 em França. As suas ferramentas de email permitem construir emails de forma concorrente com outros utilizadores. Também permite enviar SMS's transacionais, embora não seja o foco da plataforma. Foi adquirida em 2019 pela Mailgun, uma plataforma de envio de emails transacionais.

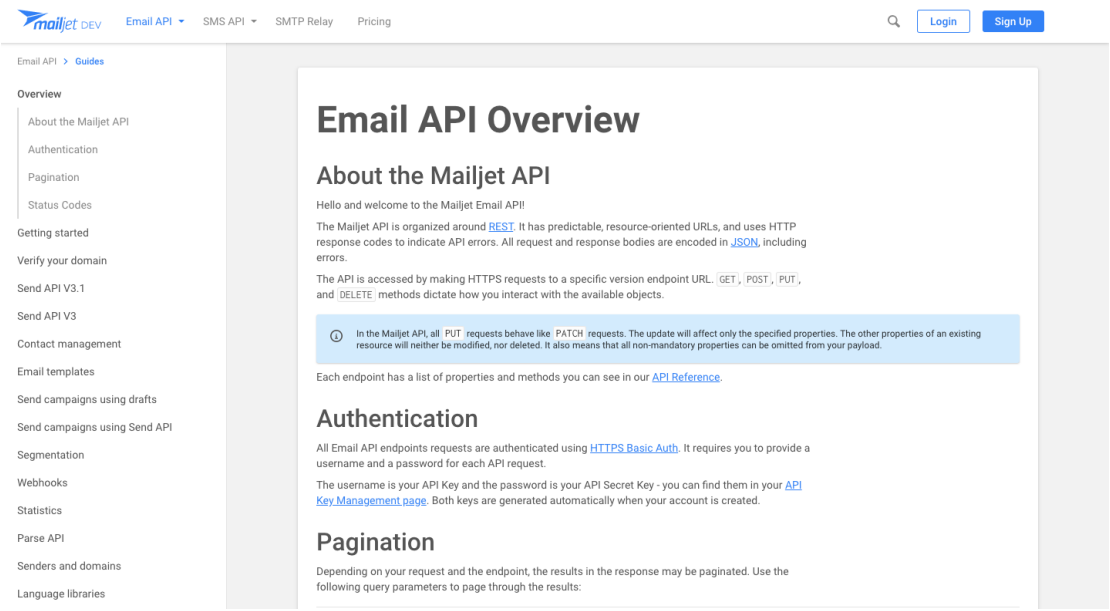

Figura 30 - Documentação da API da Mail Jet

4.2.1.3 Nexmo (Vonage)

Nexmo é uma plataforma que integra API's de vários canais de comunicação, como SMS's e Voz. Foi fundada em 2010, e é sediada nos Estados Unidos. Também possui algumas API's específicas a alguns casos de uso, como o caso do *Verify*, a versão da Nexmo de 2FA, e integrações com canais de comunicação como *Facebook Messenger*, *Whatsapp*, *Viber*, entre outros.

Em 2016 foi adquirida pela Vonage, uma provedora norte americana de serviços de comunicação na *cloud*. Para aquilo que são as atuais funcionalidades a serem estudadas, apenas será focado o envio de mensagens SMS transacionais e casos de uso relacionados, como alarmes e 2FA.

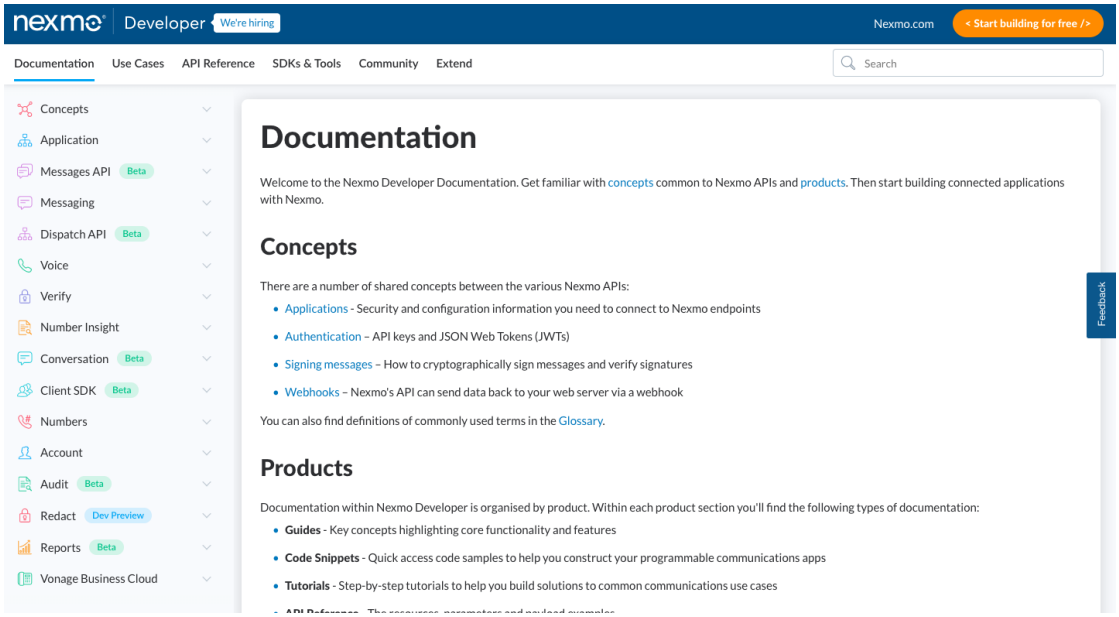

Figura 31 - Documentação da API da Nexmo

4.2.1.4 Infobip

Infobip é uma plataforma de envio de mensagens multicanal, como email, SMS, notificações push, entre outras. Foi fundada em 2006 na Croácia. Também possui ferramentas de integração com aplicações de comunicação como Whatsapp e Facebook Messenger.

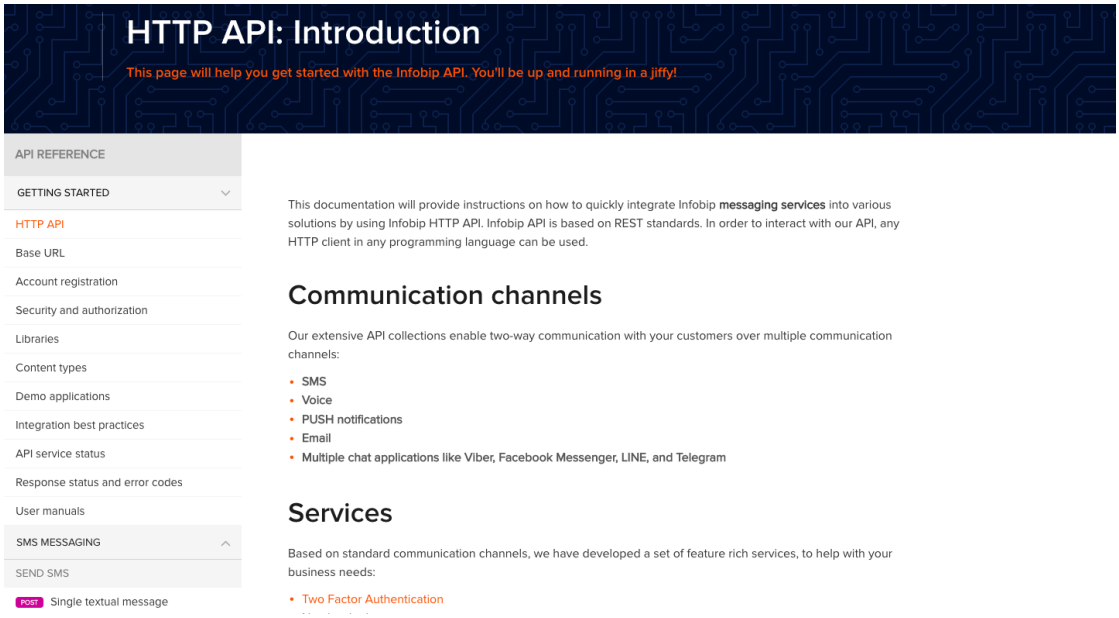

Figura 32 - Documentação da API da Infobip

4.2.1.5 SendGrid

SendGrid é uma plataforma de comunicações de marketing e transacionais, fundada em 2009 nos Estados Unidos. Neste momento suporta envio de mensagens de email, e começa a

suportar integração de publicidade com a Google, Facebook e Instagram. Foi adquirida em 2018 pela Twillio, uma plataforma de comunicação através da *cloud*.

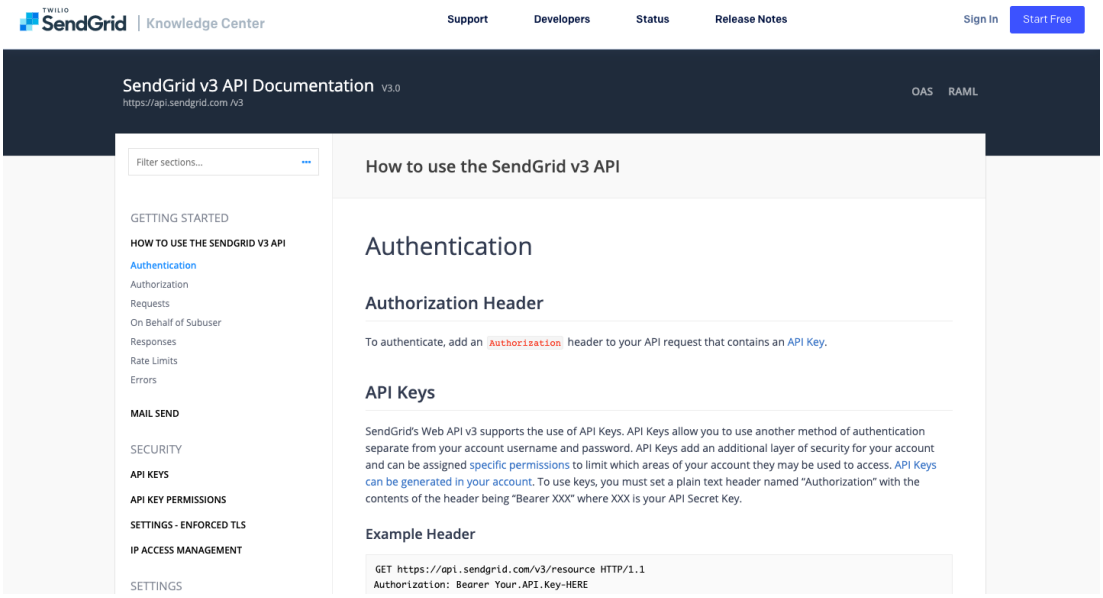

Figura 33 - Documentação da API da SendGrid

4.2.2 Comparação de Concorrentes

Tendo estabelecido os concorrentes, deve fazer-se uma análise ao que cada um oferece. Será feita uma análise as funcionalidades disponibilizadas, integrações existentes, e caraterísticas da documentação.

Os parâmetros a serem analisados serão baseados na oferta da própria API. Os resultados da comparação das funcionalidades de cada API estão apresentados na Tabela 14.

Tabela 14 - Matriz Competitiva (Funcionalidades da API)

	Send SMS	**Send Email**	**Other Channels**	**Webhooks**	**Templates**	**Verify (2FA)**	**Reports**
Slingshot	Y	Y	N	Y	Y	Y	Y
Mandrill	N	Y	N	Y	Y	N	N
Mailjet	Y	Y	N	Y	Y	N	Y
Nexmo	Y	N	Y	Y	N	Y	N
Infobip	Y	Y	Y	Y (tracking)	N	Y	Y
SendGrid	N	Y	N	Y	Y	N	Y

A análise a Tabela 14 mostra que não existe, neste momento, uma solução que tenha todas as funcionalidades na oferta de cada um dos concorrentes e da própria Slingshot. Cada uma das organizações, apesar de ter a mesma natureza transacional, possui focos diferentes. Por exemplo, a Nexmo possui um maior foco em comunicações através de dispositivos móveis, e é a única que não oferece envio de emails. A conclusão que se tira para análise a comparação de funcionalidades entre a Slingshot e os seus concorrentes é que a sua oferta não destoa no meio do mercado disponível.

De seguida faz-se uma análise às formas alternativas de integração da API de cada organização. Alguns dos concorrentes oferecem alternativas de integração da API, como o uso de SDKs. O uso de excertos de códigos também será considerado um exemplo de integração válida, uma vez que facilita a integração da API na aplicação do desenvolvedor. Finalmente, por uma questão de nomenclatura, será considerada a própria API como forma de integração. Na Tabela 15 estão apresentados os meios de disponibilização de cada concorrente.

Tabela 15 - Matriz competitiva dos meios de disponibilização de cada concorrente

	API	**SDK**	**Code Examples**
Slingshot	Y	N	N
Mandrill	Y	Y	Y
Mailjet	Y	Y	Y
Nexmo	Y	N	Y
Infobip	Y	Y	Y
SendGrid	Y	Y	Y

A análise a Tabela 15 mostra que vários dos concorrentes da Slingshot possuem mais do que uma forma de disponibilização das suas API's. Em comparação, a Slingshot apenas possui a disponibilização da própria API como forma de integração. Desta forma, vê-se como uma mais valia disponibilizar a API sob forma de SDK.

Antes de se iniciar a análise à documentação deve fazer-se um levantamento de algumas caraterísticas técnicas da API de cada organização. Componentes como o método de autenticação e versionamento, apesar de serem apresentados na documentação da API, são definidos na camada de desenvolvimento. Por outro lado, variações destes componentes são influenciadas pela evolução da API, pelo que não é descabido serem considerados para uma análise mais profunda. Na Tabela 16 estão apresentados os métodos que cada um dos concorrentes utiliza para autenticação e versionamento na API.

Tabela 16 - Métodos de autenticação e versionamento dos concorrentes

	Autenticação	Versionamento	Documentação
Slingshot	ApiKey	N.A.	(E-goi, 2020)
Mandril	ApiKey	Componente no url	(The Rocket Science Group, 2014)
Mailjet	HTTP Auth (ApiKey publica e privada)	Componente no url	(Mailjet, 2018)
Nexmo	ApiKey	Componente no url	(Nexmo, 2020)
Infobip	Servidor do url	Componente no url	(Infobip, 2020)
SendGrid	Auth Header	Componente no url	(SendGrid, 2020)

A primeira conclusão que se pode tirar da Tabela 16 é que a Slingshot não possui método de versionamento. O motivo para esta situação é que, neste momento, existe apenas uma única versão disponibilizada da API. Esta é implantada num único servidor público, pelo que não existem versões anteriores alternativas. Além disso, no servidor é disponibilizada apenas a versão mais recente da aplicação, pelo que o cliente utiliza automaticamente essa versão. Em contraste, o padrão de versionamento das restantes API's apresenta a versão como um componente do url (por exemplo, "http://<servidor>.com/<versão>/recurso"). Verifica-se que quase todos os concorrentes da Slingshot possuem pelo menos duas versões das suas API's, enquanto que outros, como o caso do Mandrill, já possuem as bases para o desenvolvimento de novas versões.

Em relação aos métodos de autenticação, existe uma tendência em utilizar apikeys como método de autenticação e autorização. Estas apikeys são geradas uma única vez, e ficam associadas ao cliente para o qual são geradas. Por contraste, a Mailjet utiliza um método em que conjuga a autenticação HTTP de identificador e password com apikey. Esta solução é mais complexa que usar apenas a apikey, mas é mais segura perante ataques de adivinha de passwords. A conjugação com apikey publicas e privadas esconde a identidade do utilizador no transporte da mensagem, garantindo o seu anonimato. Finalmente, a Infobip utiliza um sistema em que atribui a cada cliente o seu próprio endereço de servidor, que o associa a todos os pedidos feitos através desse endereço. De certa forma, pode encarar-se como autenticação através de apikey, que é enviada no endereço. A vantagem de utilizar esta solução é a conjugação entre servidor e autenticação, que diminui a carga de transporte de qualquer pedido. Tem a desvantagem de obrigar a identificar o DNS como sendo da Infobip, aumentando a complexidade de configuração e manutenção.

Finalmente, podem comparar-se as API'S em termos da sua documentação. Uma boa documentação deve informar o utilizador acerca das capacidades da API, apresentando todas as possibilidades de interações com a API, e oferecendo exemplos de ambientação do utilizador. Desta forma, foi feita uma análise a documentação, focando nos seguintes aspetos relativos a documentação:

1. Introdução de Conceitos – explica como utilizar a API, e quais os conceitos que trata?
2. Exemplo Inicial – apresenta um exemplo claro e pronto a testar da API?
3. Métodos HTTP – associa os métodos HTTP de forma lógica?
4. Exemplos Pedidos – apresenta exemplos lógicos e contextuais de pedidos?
5. Exemplos Respostas – apresenta exemplos lógicos e contextuais de respostas?
6. Erros de resposta – apresenta quais as potenciais mensagens de erro?

As respostas a estas questões estão apresentadas na Tabela 17.

Tabela 17 - Comparação da documentação

	Introdução Conceitos	Exemplo Inicial	Métodos HTTP	Exemplos Pedidos	Exemplos Resposta	Erros de resposta
Slingshot	Y/N	Y	Y/N	Y/N	Y/N	Y/N
Mandrill	N	N	N	Y	Y	Y
Mailjet	Y	Y	Y	Y	Y	N
Nexmo	Y	Y	Y	Y	Y	Y/N
Infobip	Y	Y	Y	Y	Y	Y/N
SendGrid	N	Y	Y	Y	Y	Y

Analisando a Tabela 17, é possível verificar que algumas das componentes estão marcadas com um 'Y/N'. Esta notação tem o significado de inconsistente ou incompleto. Por exemplo, no caso da E-goi, a utilização de métodos HTTP não é uniformemente implementada. Um ponto onde este símbolo também é notório prende-se nos erros de resposta. Algumas das documentações possuem uma secção separada dedicada aos códigos de erro. Devido a não estarem integradas com os métodos, ou aludirem a exemplos genéricos de erros, dá-se por insatisfeita a existência deste parâmetro.

Na questão dos exemplos, a Slingshot apresenta alguns, como mostra a Figura 28. No entanto, este tipo de exemplo é muito básico, apresentando em alguns casos o tipo de conteúdo do parâmetro, ao invés de exemplos com informação hipotética. Este tipo de informação facilita o cliente a compreender como utilizar a API (Robillard & DeLine, A field study of API learning obstacles, 2011), pelo que a Slingshot falha neste aspeto.

A principal conclusão a retirar da Tabela 17 deve ser a necessidade de uniformização da documentação da API da Slingshot. A manutenção evolutiva da API produziu estilos de documentação dispares, pelo que colocar os serviços mais antigos no mesmo estilo que os serviços mais recentes mantém a coerência de estrutura.

5 Design da Solução

Neste capítulo serão levantados os requisitos necessários para desenvolver a nova versão da Slingshot. Para tal será feita uma análise sobre os problemas da API atual, de forma a definir como a nova versão não incorra nesses problemas. Também se fará uma análise sobre boas práticas nos diversos. Depois, serão definidos os objetivos do projeto. Finalmente será feita a apresentação do desenho da solução.

5.1 Análise de Requisitos

Dadas as limitações descritas na secção 4.1.2 - Limitações, os principais requisitos passam por mitigar os problemas encontrados. Desta forma torna-se necessário redefinir a documentação do ponto de vista dos conceitos. O contrato da API terá de ser redefinido, de forma a melhor ir de encontro as normas identificadas no capítulo 2.

5.1.1 Requisitos funcionais

Os requisitos funcionais são as necessidades que serão respondidas através de funcionalidades do projeto. É nestes requisitos que serão definidos os componentes do modelo de negócio a serem desenvolvidos, de forma a que os seus utilizadores possam utilizar a solução desenvolvida com o mínimo de incómodo.

Neste projeto os requisitos funcionais passam por desenvolver um contrato de API que contenha as funcionalidades da Slingshot, e criando espaço no mesmo para acrescentar outras funcionalidades que complementem a pré-existentes.

Como tal, são definidos os seguintes requisitos gerais:

- Casos de uso – Os casos de uso existentes na Slingshot devem ser transportados para a nova versão;
- Envio de Mensagens – A solução deve ter a capacidade de enviar mensagens através dos canais que a Slingshot disponibiliza (Email, Sms, Push);

- Configuração – A solução deve poder fazer a configuração dos aspetos necessários ao envio de uma mensagem;
- Produção de resultados – A solução deve ser capaz de produzir dados que derivem das ações do utilizador, como o número de mensagens enviadas.

5.1.2 Requisitos não funcionais

Os requisitos não funcionais são as condições em que a solução deverá cumprir. Aqui se integra as condições de disponibilização da solução, estabilidade da mesma, entre outros. No contexto do projeto, aqui se inserem os requisitos que não estão diretamente ligados às funcionalidades da API.

A melhoria da documentação será um dos requisitos. Deve ser produzida informação relevante que acompanhará a disponibilização da API, estudando e apresentando os conceitos relacionados com as secções em que se inserem.

Outro requisito a ser implementado será o de aumentar as opções de integração da nova versão. Devem ser estudadas formas de integração, como SDK's e excertos de código, de forma a diminuir aos utilizadores da API o esforço de integrar a nova versão nas suas aplicações.

Para complementar a listagem dos requisitos, fez-se uso do modelo FURPS+, que mede a qualidade de um software. Assim sendo, os requisitos não funcionais do projeto são os seguintes:

- Funcionalidade – a solução deverá fazer um registo dos pedidos, de forma a monitorizar a aplicação, e facilitar a descoberta de erros de processamento; Todos os pedidos recebidos serão validados antes de serem processados;
- Usabilidade – As respostas aos pedidos devem ser consistentes na sua estrutura, quer em casos de sucesso ou erro; A estrutura de resposta deve ser consistente conforme o tipo de pedido feito;
- Confiabilidade – Nada a assinalar;
- Performance – O tempo de resposta a um pedido não deve exceder os 20% de aumento em relação ao mesmo pedido feito à Slingshot;
- Suportabilidade – O sistema deve permitir distribuir os pedidos pelos servidores existentes; O sistema deve armazenar os pedidos feitos, evitando que estes sejam perdidos em caso de indisponibilização do sistema.

5.2 Desenho da Solução

Nesta secção será apresentada a estrutura da solução a ser desenvolvida, a partir da análise feita nas secções anteriores.

5.2.1 Modelação de Domínio

Começou-se por estabelecer uma divisão primária daquilo que são as entidades dentro da Slingshot. Neste momento pode dividir-se a plataforma em 5 componentes: uma para cada

canal existente (Email, Sms, Push), uma para os casos de uso que fazem uso dos canais (Multi-Channel, Alerts, Verify), e uma para as restantes funcionalidades.

Para facilitar a implementação da solução, foi desenhado e definido o contrato em formato YAML em especificação OpenAPI. Esta abordagem de *contract first* tem várias vantagens para o desenvolvimento de projetos como este. Primeiro, diminui as decisões a serem tomadas na fase de desenvolvimento, uma vez que estas foram tomadas na definição do contrato. Depois, qualquer alteração ao contrato pode ser rapidamente feita e visualizada. Finalmente, o desenho em OpenAPI permite integrar com várias ferramentas que permitem não só criar uma maquetização do resultado final como desenvolver projetos base para servidores e clientes, através de ferramentas como Swagger Editor.

5.2.1.1 Email

Para realizar o envio de um email é necessário conhecer os parâmetros necessários para tal. Isto inclui dados como o remetente, o destinatário, corpo da mensagem e outras propriedades. Algumas destas propriedades serão mais relevantes para alguns utilizadores do que para outros. Por este motivo é importante definir as propriedades essenciais ao envio de qualquer mensagem de email, e identificar entidades que necessitarão de maior configuração.

Na Tabela 18 estão descritos os componentes para o envio de um email através da Slingshot. Muitos destes parâmetros não serão necessários para o envio de uma mensagem de email do utilizador comum. No entanto, a Slingshot permite tanto customização da mensagem que o cliente irá receber, como a forma como a mensagem é processada dentro da plataforma.

Tabela 18 - Componentes de uma mensagem Email via Slingshot

Entidade	**Descrição**
Email	O endereço de email do destinatário.
Subject	O assunto do email.
Message	O corpo da mensagem. Pode ser formatado em HTML ou Texto.
Sender	A identificação do perfil do remetente.
Domain	O domínio do email do remetente.
Custom Data	Informação a anexar a mensagem. É invisível ao destinatário, servindo principalmente para o remetente inserir contexto em futura analise.
Merge Tags	Pares de Chave-Valor, que substituem a ocorrência de chaves nas mensagens por valores personalizados para cada destinatário.
Reply To	A identificação do perfil do remetente para o qual o destinatário poderá responder.
Tracking Enabled	Opções de rastreamento da mensagem. Pode rastrear aberturas e cliques do email.
Registered	Opção de processar uma mensagem de forma segura, podendo ser consultada no futuro para contextos formais e/ou legais.
Priority	Prioridade de envio do email.
Attachments	Ficheiros para anexar ao email.
Headers	Cabeçalhos do email. Tem a opção de exibir a opção de deixar de subscrever, ou apresentar um endereço de subscrição.
TemplateId	Identificação de uma mensagem pré-definida no sistema. A sua utilização substitui o assunto, corpo da mensagem e rastreamento (Tracking).
Group	Nome utilizado para agregar mensagens para analise futura.
Events	Eventos que são captados após o email ser enviado.
Status	Estado da mensagem.

Alguns dos componentes apresentados na Tabela 18 são expectáveis, tal como o email do destinatário, assunto e corpo de mensagem. O remetente, assim como o domínio, possuem outro tipo de processamento e validação, pelo que devem ser previamente configurados. Por este motivo, esta configuração não fará sentido ser feita no momento de envio.

Na Figura 34 é apresentada a estrutura de dados necessária para o envio de um email.

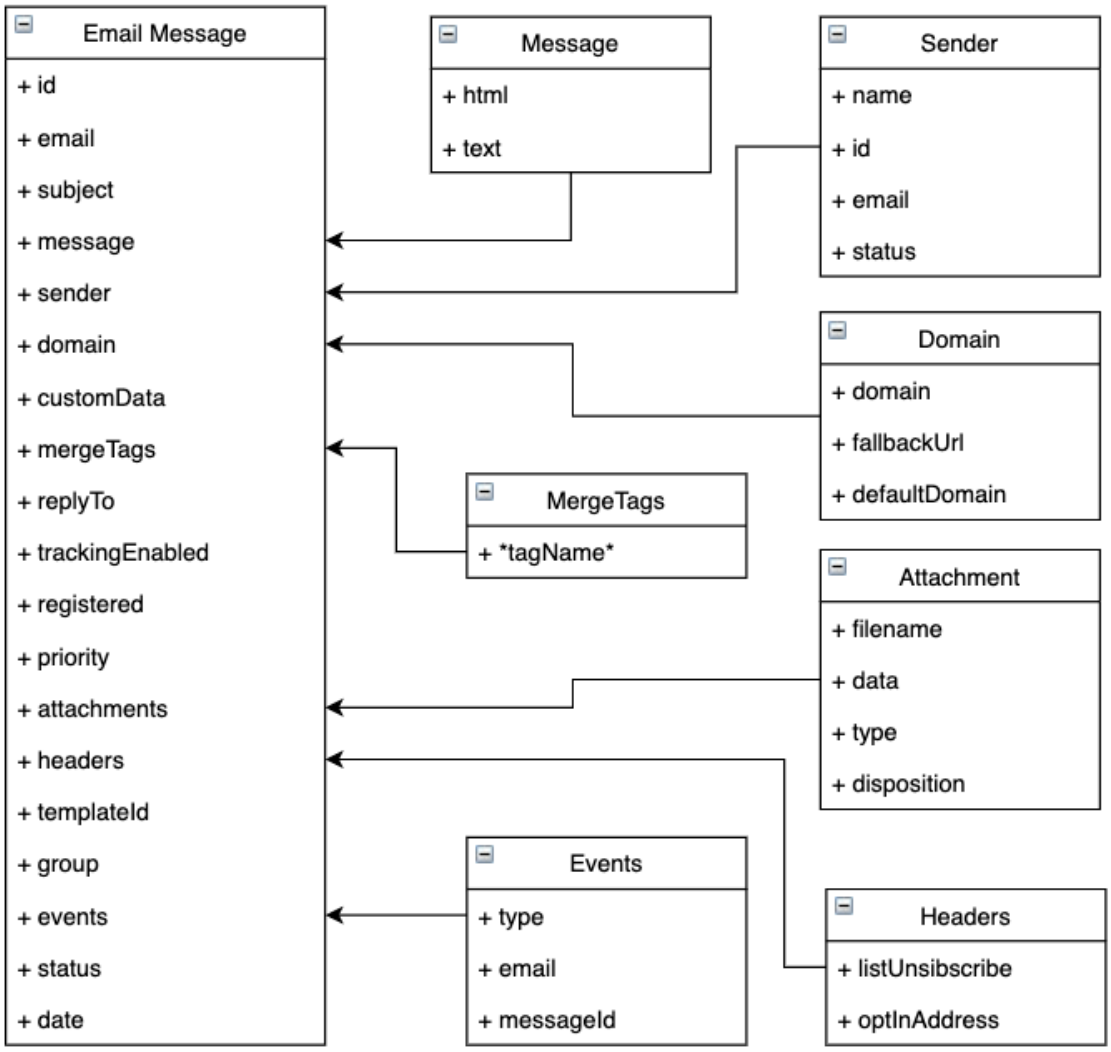

Figura 34 - Estrutura de dados relacionados com o envio de uma mensagem Email

Na Figura 34 verifica-se que existem diferenças entre o tipo de dados e estrutura dos mesmos. Alguns parâmetros são simples, como a opção de fazer o acompanhamento do email. Outros parâmetros, como cabeçalhos (*Headers*) exigem uma estrutura de dados mais complexa. Note-se que o parâmetro de 'templateId' não possui estrutura no envio, sendo que este é configurado antes do envio, e referenciado no pedido.

A divisão que foi feita foi a seguinte:

- Send Sms – Serviços de envio de mensagens Email
- Domains – Serviços de gestão de domínios
- Templates – Serviços de gestão de Templates de Email
- Senders – Serviços de gestão de Remetentes Email
- Messages – Serviços de consulta de mensagens
- Reports – Serviços de consulta de relatórios

A categoria Send Email deverá ser a mais utilizada, uma vez que é a operação a partir do qual as outras estão dependentes. Por outro lado, o facto de ser uma categoria a parte permite complementar a documentação do canal, enriquecendo-a com descrições mais pormenorizadas das capacidades, opções e limitações do canal.

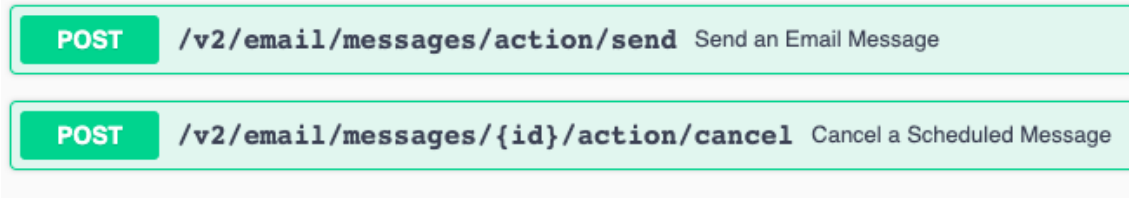

Figura 35 - Serviços relacionados com Send Email

Na Figura 35 estão representados os serviços relacionados com ações sobre o envio de emails. Existem duas ações possíveis que podem ser feitas em relação ao envio de uma mensagem de email: o envio em si, e o cancelamento. O envio, sendo uma ação que gera uma nova mensagem no sistema, faz sentido que seja acompanhada de um método Post. O cancelamento, no entanto, causa uma mudança no fluxo do envio de uma mensagem email, que sugere a utilização de um método *Put*. No entanto, o cancelamento é visto como uma ação sobre o fluxo, não a mensagem em si, pelo que, neste caso, é acompanhada por um método *Post*.

Depois do envio de mensagens pode configurar-se quais os componentes que podem ser definidos antes do envio de uma mensagem email. Começa-se pela análise aos serviços de gestão de domínio, apresentados na Figura 36.

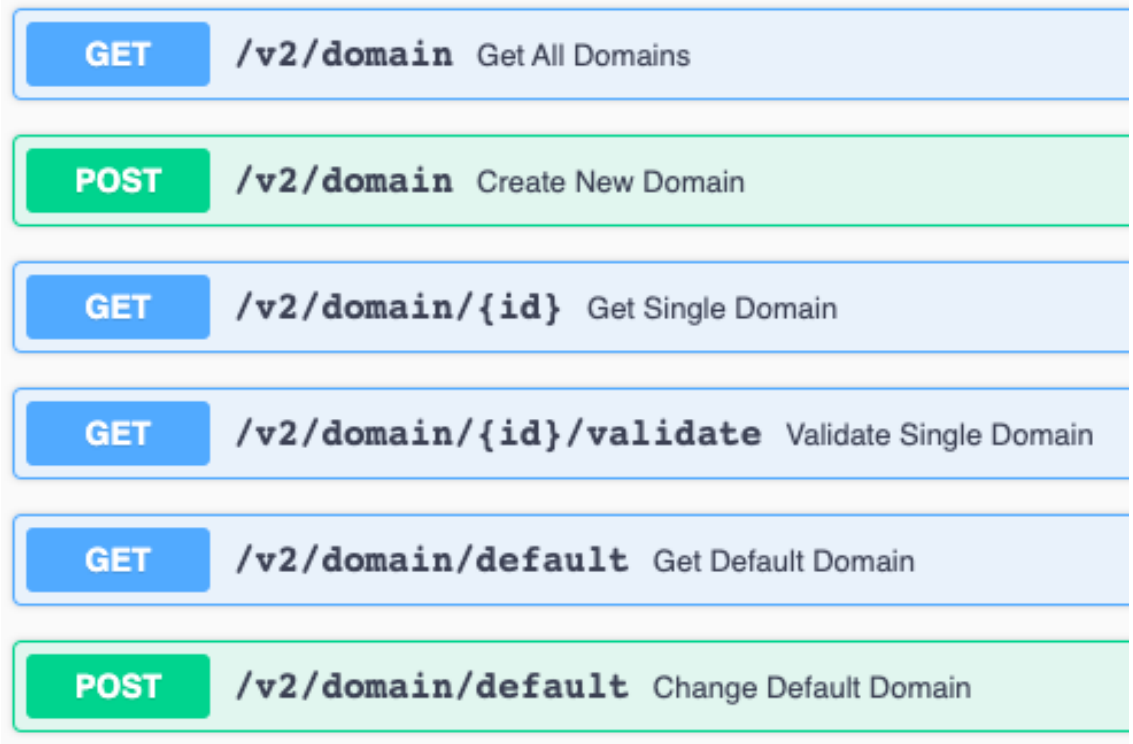

Figura 36 - Serviços relacionados com Domains de Email

O utilizador pode configurar vários domínios, consultá-los e fazer alterações neles, pelo que esta é uma oportunidade de aplicar conceitos RESTful. No entanto, as funcionalidades

disponíveis tanto pela Slingshot como E-goi não permitem a sua implementação, uma vez que, como os domínios têm de ser validados, não fará sentido alterá-los após a validação.

Outro componente que deve ser configurada antes do envio de emails são os Templates. As funcionalidades relacionadas com eles são apresentadas na Figura 37.

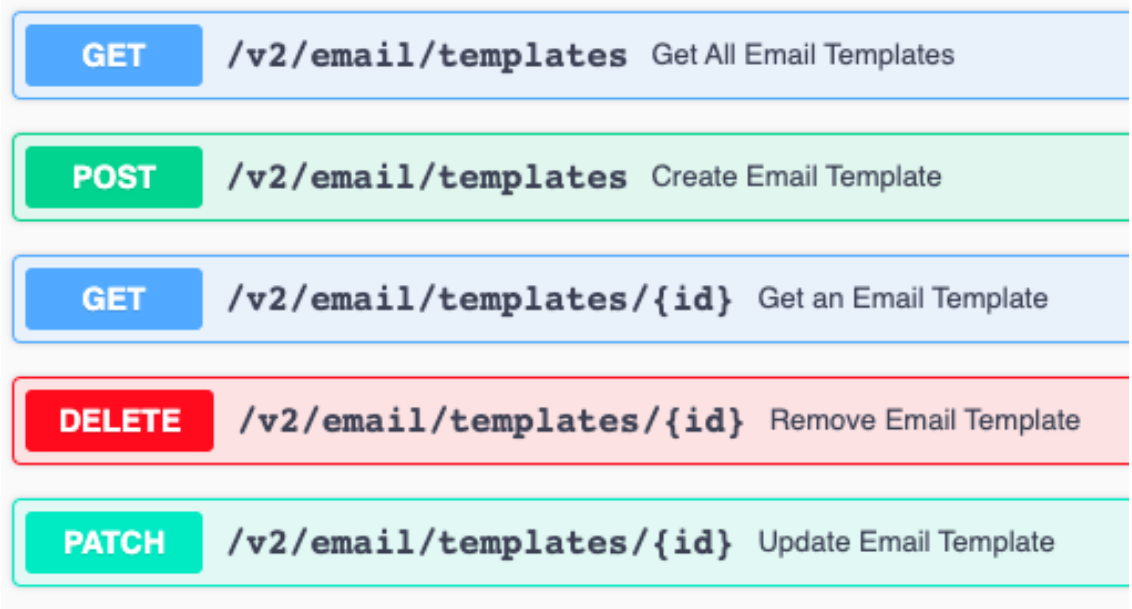

Figura 37 - Serviços relacionados com Templates de Email

Uma vez que um Template necessita de ser definido antes do envio de uma mensagem Email, torna-se necessário fazer a sua gestão dentro da Slingshot. A categoria dos Templates permite estabelecer uma entidade definida dentro do contexto do canal. Uma vez que um template pode ser criado, alterado e removido, pode aplicar-se o conceito de CRUD. Aqui se apresenta uma oportunidade de aplicar um endereço RESTful. Note-se que o método HTTP selecionado para edição de um Template é um *Patch*. Este método descreve melhor o processamento por parte do serviço, uma vez que as alterações são consideradas parciais, e não integrais.

Passando para a componente de Senders, têm-se a configuração dos remetentes, apresentada na Figura 38.

Figura 38 - Serviços relacionados com Senders de Email

A categoria de Senders, tal como nos Templates, permite fazer a gestão de remetentes. Estes terão de ser definidos antes do envio de uma mensagem Email, de forma a proceder a validação do mesmo. Neste momento, a Slingshot apenas faz a listagem dos remetentes disponíveis dentro da aplicação, sendo que a criação está a cargo da plataforma E-goi. No entanto, existe a

opção de gestão de remetentes na sua API interna, que permite acrescentar a funcionalidade de criação de remetente.

Finalmente, têm-se as componentes de produção de resultados. Aqui são apresentados serviços de consulta, quer das mensagens de forma individual, quer das estatísticas relacionadas com os envios. Na Figura 39 é apresentado o serviço único de Report.

Figura 39 - Serviços relacionados com Reports de Email

Este serviço retorna as estatísticas dos envios de mensagens, que permite ao utilizador verificar o sucesso dos seus envios. No entanto, o utilizador poderá querer listar as mensagens que enviou. Para tal é disponibilizada a componente de Messages, apresentada na Figura 40.

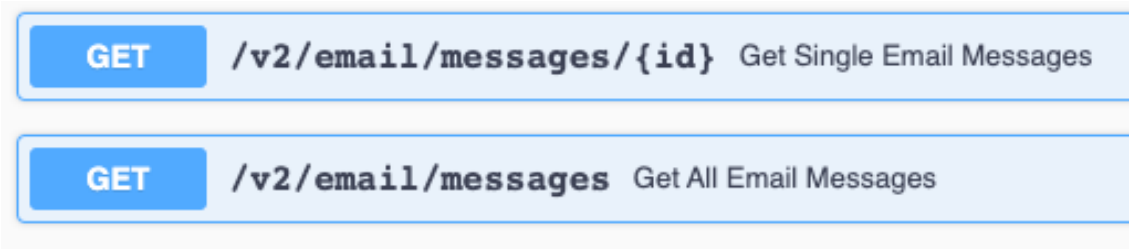

Figura 40 - Serviços relacionados com Messages de Email

Dado o número de mensagens enviadas pela Slingshot, será irrealista que a listagem de todas as mensagens seja eficiente. Assim, divide-se a carga entre os dois serviços, reduzindo os pormenores na listagem global, e aumentando-os na listagem individual de cada mensagem.

5.2.1.2 Sms

Tal como no email, é necessário estudar quais os parâmetros necessários para o envio de uma Sms. Uma vez que os canais são diferentes, as mensagens também o serão. No entanto, existem alguns componentes que serão semelhantes do ponto de vista do comportamento, pelo que alguns deles terão estrutura de serviços semelhantes.

De forma a melhor analisar os componentes relacionados com uma mensagem Sms estes são apresentados na Tabela 19.

Tabela 19 - Componentes de uma mensagem Sms via Slingshot

Entidade	**Descrição**
Mobile	O número de telemóvel do destinatário.
Message	O corpo da mensagem.
Sender	A identificação do perfil do remetente.
Options	Opções de personalização do envio da mensagem. Inclui codificação da mensagem e número máximo de mensagens a enviar (quando o número de carateres excede o número permitido por mensagem)
Custom Data	Informação a anexar a mensagem. É invisível ao destinatário, servindo principalmente para o remetente inserir contexto em futura analise.
Merge Tags	Pares de Chave-Valor, que substituem a ocorrência de chaves nas mensagens por valores personalizados para cada destinatário.
Registered	Opção de processar uma mensagem de forma segura, podendo ser consultada no futuro para contextos formais e/ou legais.
TemplateId	Identificação de uma mensagem pré-definida no sistema. A sua utilização substitui o corpo da mensagem.
Group	Nome utilizado para agregar mensagens para analise futura.
Events	Eventos que são captados após o email ser enviado.
Status	Estado da mensagem.
Date	Data de criação da mensagem.

Comparando a Tabela 18 com a Tabela 19, verifica-se que existem muitos componentes em comum. Conceitos como Registered, Grupos ou Estado são comuns aos dois canais. Estes componentes permitem homogeneizar o processamento dos pedidos e tratamento da informação. Conseguir replicar comportamentos entre canais diminui o grau de complexidade de uso para o utilizador.

Nos pontos em que as tabelas divergem, os motivos para tal resumem-se a forma de processamento dos canais e a limitações dos mesmos. Em primeiro lugar, o corpo de uma mensagem Sms é mais limitado, uma vez que não suporta formatação de texto. Não fará sentido reutilizar uma estrutura de mensagem como o email. Por outro lado, apesar do conceito de Options de Sms ser bastante semelhante ao conceito de Header de Email, as opções dos dois são especificas ao canal em que se inserem.

Na Figura 41 é apresentada a estrutura de dados necessária para o envio de uma Sms.

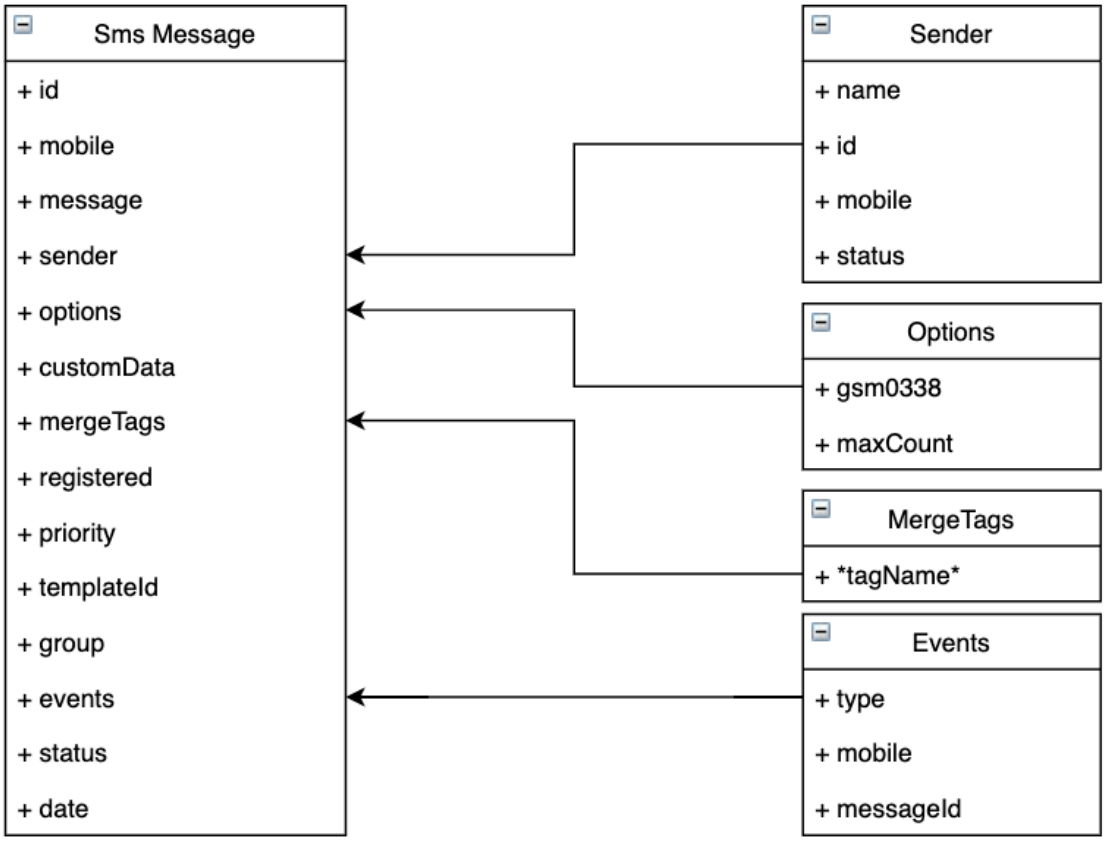

Figura 41 - Estrutura de dados relacionados com o envio de uma mensagem Sms

Tal como na Figura 34, existem parâmetros mais simples, e outros mais complexos. As opções mais simples são independentes para cada mensagem enviada, e apenas alteram o processamento da mensagem, pelo que não garante uma oportunidade de conceito de configuração. No entanto, componentes como Sender poderão ser configuradas antes, assim como o 'templateId', que apenas necessita de ser referenciado no pedido.

A divisão que foi feita foi a seguinte:

- Send Sms – Serviços de envio de mensagens Sms
- Senders – Serviços de gestão de Remetentes
- Templates – Serviços de gestão de Templates de Sms
- Messages – Serviços de consulta de mensagens
- Reports – Serviços de consulta de relatórios

A categoria Send Sms deverá ser a mais utilizada, uma vez que é a operação a partir do qual as outras estão dependentes. Os serviços deste conjunto são apresentados na Figura 42.

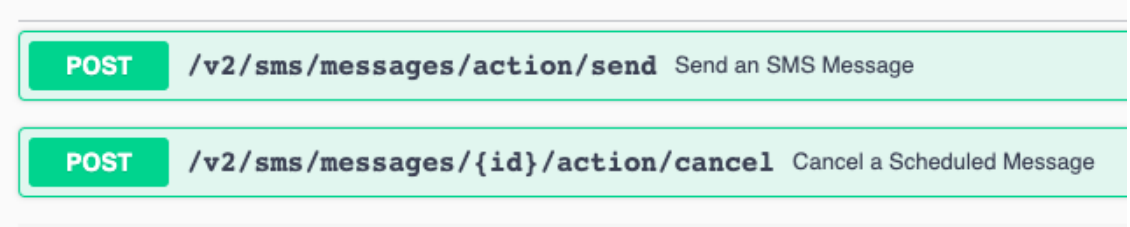

Figura 42 - Serviços relacionados com Send Sms

Tal como no caso do Send Email, o utilizador pode enviar uma ou várias mensagens através do respetivo serviço, e uma mensagem que esteja planeada pode ser cancelada.

Depois de configurado o envio, tem-se a configuração dos Senders, cujos serviços são apresentados na Figura 43.

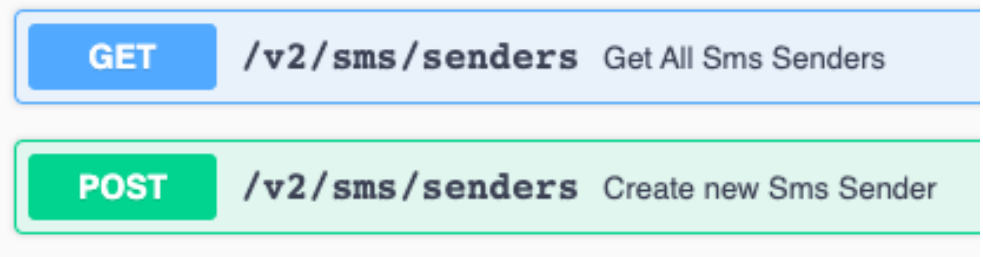

Figura 43 - Serviços relacionados com Senders de Sms

A categoria de Senders permite fazer a gestão de remetentes. Estes terão de ser definidos antes do envio de uma mensagem Sms, de forma a proceder a validação do mesmo. Neste momento, a Slingshot apenas faz a listagem dos remetentes disponíveis dentro da aplicação, sendo que a criação está a cargo da plataforma E-goi. No entanto, existe a opção de gestão de remetentes na sua API interna, pelo que se disponibilizou essa opção na nova versão da Slingshot.

Passando para os Templates, são apresentados os serviços relacionados com estes na Figura 44

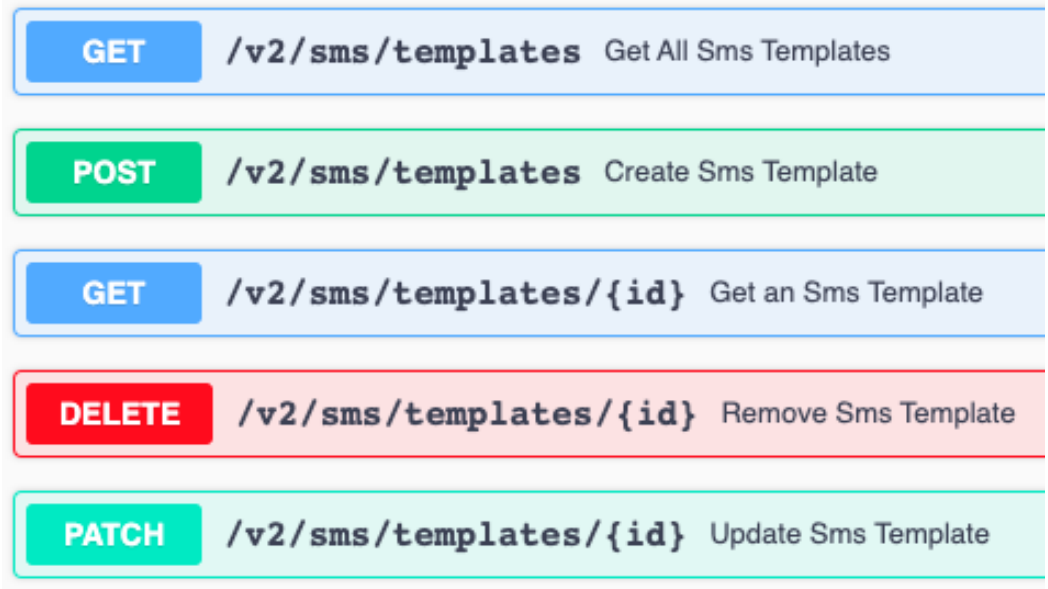

Figura 44 - Serviços relacionados com Templates de Sms

Uma vez que um Template necessita de ser definido antes do envio de uma mensagem Sms, torna-se necessário fazer a sua gestão dentro da Slingshot. A categoria dos Templates permite estabelecer uma entidade definida dentro do contexto do canal. Uma vez que um template pode ser criado, alterado e removido, pode aplicar-se o conceito de CRUD. Aqui se apresenta uma oportunidade de aplicar o nível 2 da aplicação de RESTful.

Finalmente têm-se os conjuntos de funcionalidades relacionada com a consulta de dados após o envio. Tal como no Email, poderão ser listados os dados relacionados com as mensagens que foram enviadas até ao momento do envio. Na Figura 45 é apresentado o serviço que retorna os dados relacionados com as estatísticas dos envios.

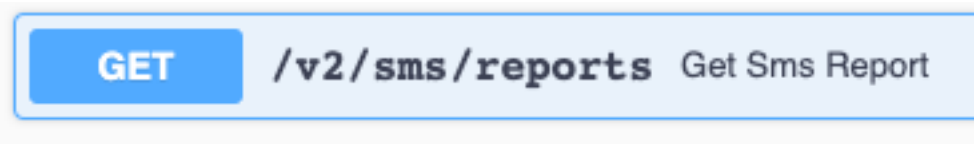

Figura 45 - Serviços relacionados com Report de Sms

Para complementar estes dados, tal como no caso dos Emails, também podem ser listadas as mensagens que forma enviadas. Os serviços relacionados com esta funcionalidade são apresentados na Figura 46.

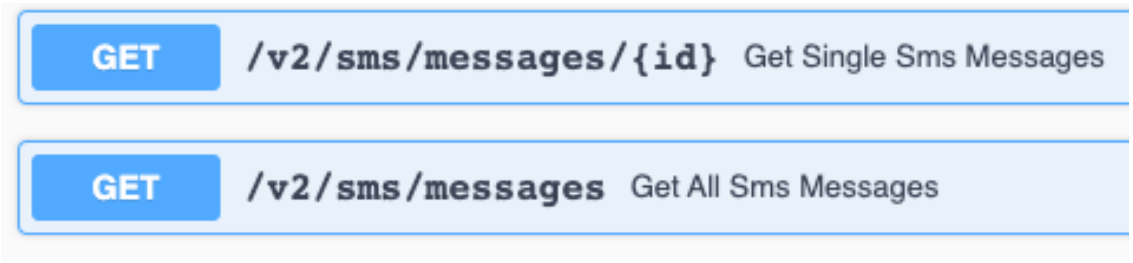

Figura 46 - Serviços relacionados com Messages de Sms

Tal como no Email, as mensagens a serem retornadas podem ser menos ou mais detalhadas, dependendo do serviço que é chamado. No caso de serem todas, a informação é condensada, enquanto que na listagem individual é apresentada a mensagem com maior detalhe.

5.2.1.3 Push

Para completar a atual oferta de canais por parte da Slingshot, será feita uma análise as mensagens via Push. Este canal é atualmente o mais recente, pelo que será o que oferece as configurações mais limitadas. No entanto, apresenta conceitos novos na API, como aplicações para dispositivos móveis. Para simplificar a listagem das componentes estas estão listadas na Tabela 20.

Tabela 20 - Componentes de uma mensagem Push via Slingshot

Entidade	**Descrição**
Device	O Id do telemovel. É utilizado o *Mobile Identification Number* (MIN) para identificar o dispositivo que irá receber a mensagem. Atua como identificador do destinatário.
App	A aplicação que envia a mensagem. Atua como identificador do remetente.
Title	O título da mensagem. Possui maior destaque.
Message	O corpo da mensagem.
TemplateId	Identificação de uma mensagem pré-definida no sistema. A sua utilização substitui o corpo da mensagem.
Events	Eventos que são captados após o email ser enviado.
Status	Estado da mensagem.
Date	Data de criação do email.

Ao contrário dos canais acima mencionados, uma mensagem Push não é tão rica em opções de customização no envio de mensagens. Atualmente não é possível agendar mensagens, associar a um grupo, ou utilizar o Registered, entre outras funcionalidades.

Na Figura 47 é visível a estrutura de dados necessária para o envio de uma mensagem Push

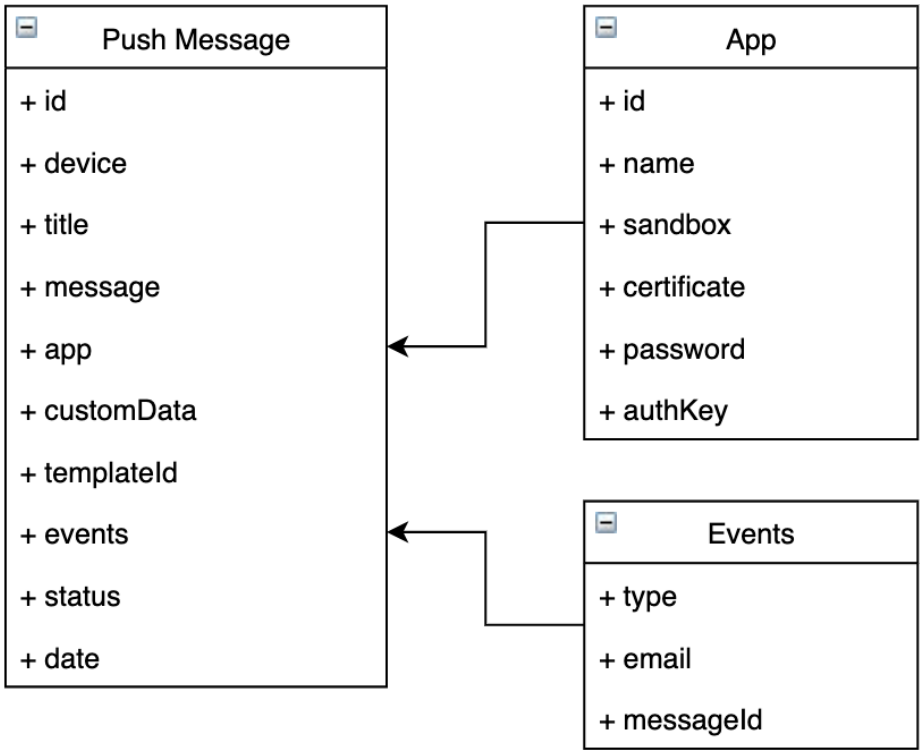

Figura 47 - Estrutura de dados relacionados com o envio de uma mensagem Push

Analisando a Figura 47 verifica-se que a complexidade de configuração deste canal provém da configuração da aplicação. Como existem menos funcionalidades associadas ao canal, é natural que exista menor número de componentes, quer a ser configurada antes do envio, quer no envio em si.

A divisão de funcionalidades que foi feita foi a seguinte:

- Send Push – Métodos de envio de mensagens Sms
- Templates – Métodos de gestão de Templates de Sms
- Mobile Apps – Métodos de gestão de Remetentes
- Reports – Métodos de consulta de relatórios

Tal como nos outros canais, o envio de mensagens Push deve ser a funcionalidade mais importante no canal. As funcionalidades associadas são apresentadas na Figura 48.

Figura 48 - Serviços relacionados com Send Push

Esta divisão, semelhante aos outros canais, cria a oportunidade de enriquecer os serviços com documentação relevante ao envio. Note-se que, tal como nos outros canais, as funcionalidades que o utilizador pode executar são o envio e cancelamento de uma mensagem agendada.

Uma mensagem Push não possui um conceito de remetente semelhante aos canais Email e Sms. Possui, no entanto, o conceito de App, que é associada à mensagem a ser enviada. As funcionalidades relacionadas com a mesma são apresentadas na Figura 49.

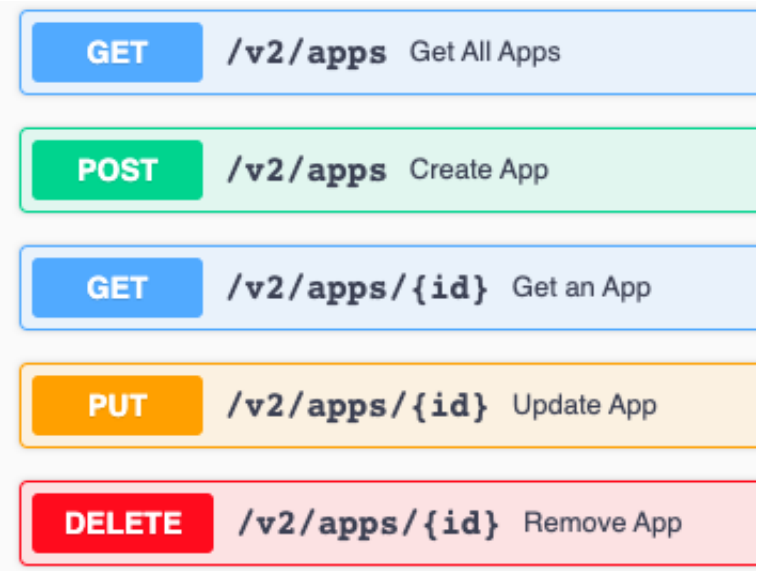

Figura 49 - Serviços relacionados com as Apps

A categoria de Mobile Apps, no contexto de mensagens Push, substitui o conceito de Sender. Neste caso, o envio da mensagem fica associado a aplicação do cliente, uma vez que servirá como intermediário entre a aplicação do cliente e o destinatário. Estes terão de ser definidos antes do envio de uma mensagem Sms, de forma a proceder a validação do mesmo. Ao contrário dos restantes canais, a Slingshot já possui os serviços necessários à gestão de Apps, pelo que se poderá integrar essa funcionalidade.

À semelhança dos outros canais, também podem ser configurados Templates Push, que poderão ser utilizados no momento de envio. Os serviços relacionados com esta funcionalidade são apresentados na Figura 50.

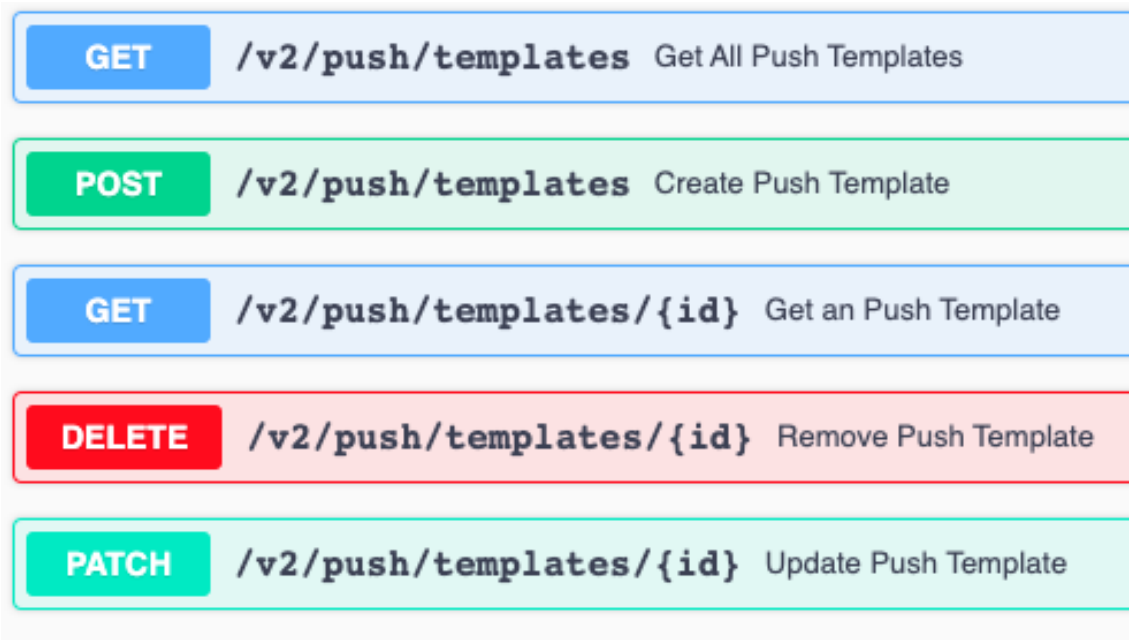

Figura 50 - Serviços relacionados com Templates de Push

Uma vez que um template necessita de ser definido antes do envio de uma mensagem Sms, torna-se necessário fazer a sua gestão dentro da Slingshot. A categoria dos Templates permite estabelecer uma entidade definida dentro do contexto do canal. Uma vez que um template pode ser criado, alterado e removido, pode aplicar-se o conceito de CRUD. Aqui se apresenta uma oportunidade de aplicar um endereço RESTful, e subir ao nível dois de aplicação REST.

Finalmente, surge as funcionalidades relacionadas com o processamento de dados após o envio de mensagens Push. O serviço único associado a esta funcionalidade é apresentado na Figura 51.

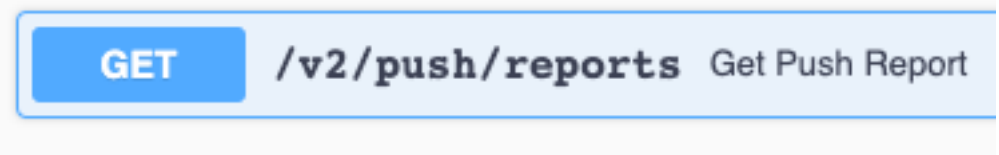

Figura 51 - Serviços relacionados com Report de Push

Neste momento, não existe forma de listar a mensagem Push que tenha sido enviada. Esta é uma funcionalidade que nem a Slingshot nem a E-goi possuem, pelo não poderá ser disponibilizada.

5.2.1.4 Casos de Uso

Nesta categoria estarão os casos de uso que envolvem os canais acima mencionados. Neste momento, a Slingshot possui três casos de uso: Multi-Channel, Verify e Alerts.

Os Alerts são mensagens que são enviadas de forma regular, até que esta seja reconhecida por um destinatário (carregando no link que vai com a mensagem), ou até que seja atingido o limite de ciclos definido. Os componentes de um Alert estão apresentados na Tabela 21.

Tabela 21 - Componentes de um Alert

Entidade	**Descrição**
Message	A mensagem a ser enviada. Deve ser adaptada a cada canal. Tanto se poderá utilizar a definição da mensagem como um templateId.
Interval	O intervalo de espera entre o envio de cada mensagem
MaxAttempts	O número máximo de iterações que o alerta faz. Pode encarar-se como o número máximo de mensagens a ser enviadas.

De acordo com a Tabela 21, nenhuma das entidades de um Alert estão dependentes do momento do envio. Portanto, um Alert pode e deve ser configurado antes de lançar um alerta. Este conceito de Alert pré-configurado tem a denominação de Flow. Como tal, deve existir um conjunto de funcionalidades exclusivo a gestão dos Flows.

Estando definido o Flow, deve analisar-se a execução do mesmo. Do ponto de vista de transmissão de informação, resta apenas definir um remetente e um ou vários destinatários. No caso de enviar para vários destinatários, o Alert é interrompido assim que um dos destinatários acuse a receção da mensagem.

Na Figura 52 é apresentada a estrutura de dados para o envio de um Alert.

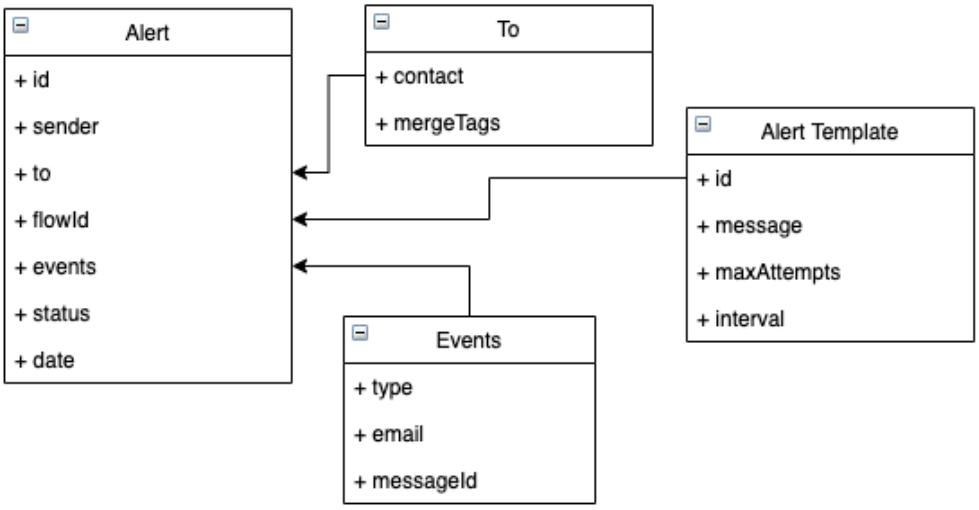

Figura 52 - Estrutura de dados relacionados com o envio de um Alert

Uma vez que a mesma mensagem é enviada pelo mesmo canal, o remetente será o mesmo para cada iteração do Alert. No entanto, como vários destinatários poderão receber o Alert, surge necessidade de adaptar o corpo da mensagem. Agrupando cada destinatário aos mergeTags, completa-se o objeto do "To". Uma vez que não é prático adaptar a mensagem a cada destinatário antes do envio desta, não fará sentido ter um caminho para essa entidade.

Em relação aos serviços relacionados com o Alert, este terá três estados: iniciado, interrompido e terminado. Neste caso, assume-se que o Alert é interrompido através de ação humana (através do reconhecimento da mensagem, ou até do cancelamento por parte do utilizador), e terminado ao atingir o limite de iterações. Os serviços relacionados com a execução de um Alert são apresentados na Figura 53.

Figura 53 - Serviços relacionados com Execução de um Alert

Como mencionado acima, antes de enviar um Alert é necessário configurá-lo. Esta configuração poderá ser feita através das chamadas aos serviços apresentados na Figura 54.

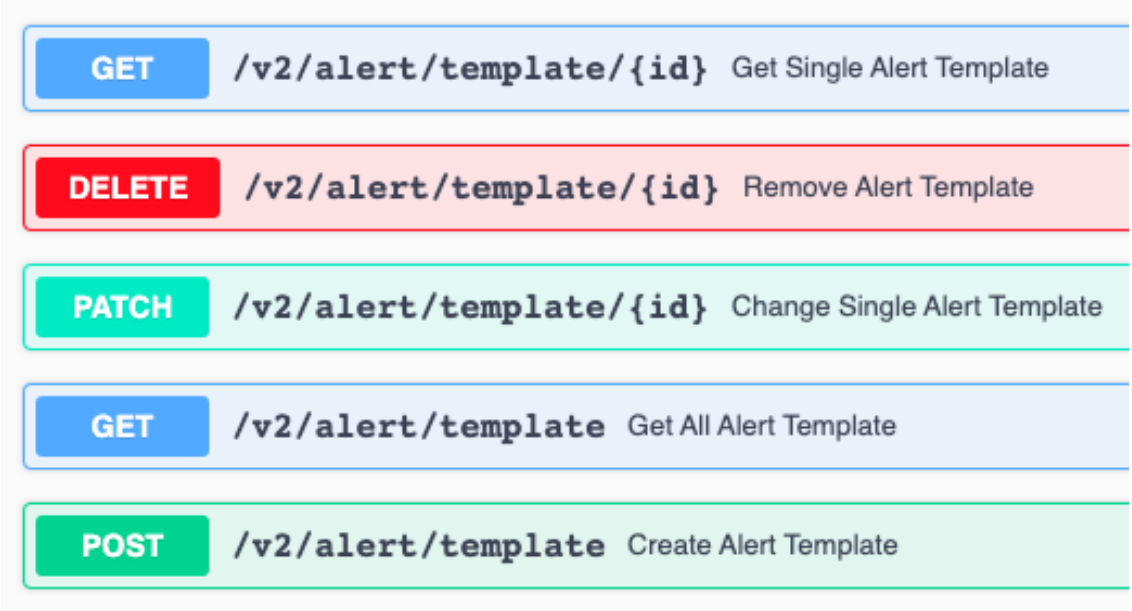

Figura 54 - Serviços relacionados com Templates de Alerts

Estes serviços serão configurados utilizando o padrão CRUD, e tendo por base o nível 2 de implementação REST. De notar que, para a edição do Template, foi utilizado o método HTTP *Patch*. Este método foi selecionado para melhor representar a funcionalidade por parte da Slingshot. De facto, a edição do mesmo é feita de forma seletiva, pelo que este método se aplica melhor ao contexto do que o método *Put*.

Estando definidas as necessidades para o Alert, será feita uma análise a componente de Multi-channel. Um multi-channel, sendo uma sequência de mensagens pré-definidas, engloba várias e diferentes partes. Por exemplo, o corpo de uma mensagem de email é diferente do corpo de mensagem Push.

As componentes de um Flow de MultiChannel estão apresentados Tabela 22. À semelhança dos Alerts, um Flow representa a sequência de mensagens a ser enviadas, juntamente com algum contexto para o utilizador. Desta forma, o utilizador poderá preparar vários Flows, iniciando-os conforme a necessidade.

Tabela 22 - Componentes de um Flow de um Multi-Channel

Entidade	**Descrição**
Name	O nome do Flow
Group	O grupo para associar as mensagens (se possivel).
Messages	A sequência de mensagens a ser enviadas, juntamente com a sua condição de envio.

Apesar de não mostrar na Tabela 22, as mensagens são definidas pelo canal, corpo da mensagem e as condições para prosseguir com o Flow. Esta relação é melhor entendida através da Figura 55.

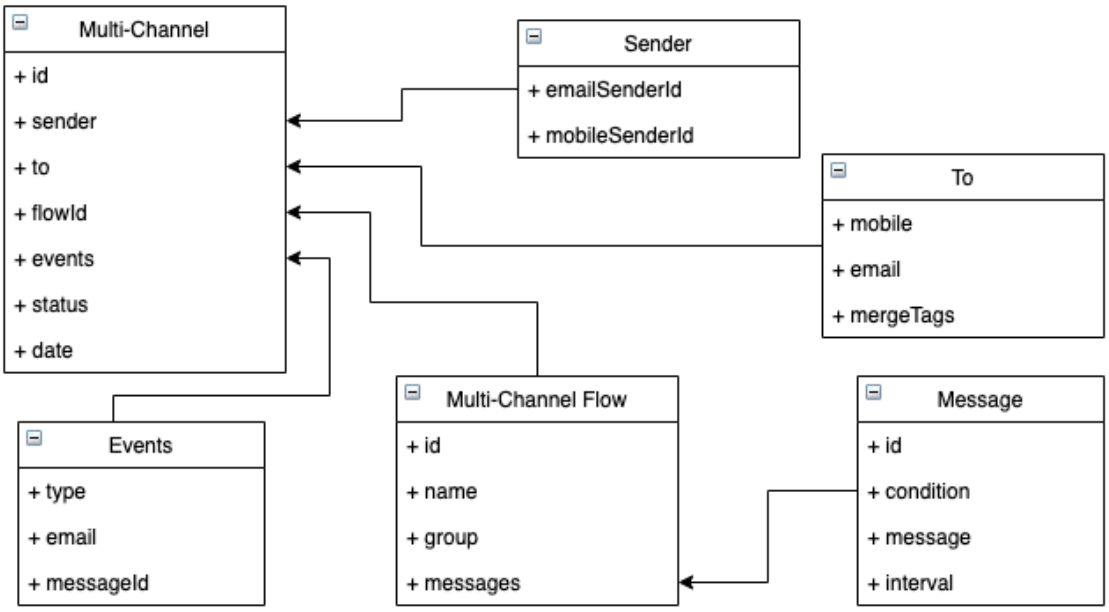

Figura 55 - Estrutura de dados relacionados com o envio de um Multi-channel

Tal como é visível na Figura 55, uma mensagem em si possui não só o corpo da mensagem como qual a condição a ser cumprida para a despoletar, e quanto tempo aguardar para verificar a condição da mensagem seguinte.

No caso do Multi-Channel, a opção de acompanhamento do estado da mensagem é automaticamente ativa. Caso contrário, seria impossível perceber qual o estado da mensagem, e colocaria a execução do Flow em risco.

Uma vez que o Flow pode envolver vários canais, é necessário definir os remetentes e destinatários para cada um. Como tal, tal como é visível na Figura 55, a entidade do Sender e To possuem campos para o envio de Email e Sms. Nota para o facto do Sender ser definido através de um id, uma vez que o remetente deve ser previamente configurado e validado na aplicação E-goi.

Para dar início ao Multi-channel, definiu-se o serviço apresentado na Figura 56. Este serviço é o único que influencia o processamento do Multi-channel. Tal como o canal Push, esta funcionalidade é uma das mais recentes, pelo que a sua introdução no mercado é mais vantajosa para a E-goi acostumar os seus clientes, desenvolvendo funcionalidades como cancelamento e outras se se verificar interesse nelas.

Figura 56 - Serviços relacionados com Execução de Multi-channel

Tal como com os Templates de Alerts, o Multi-channel necessita de ser configurado antes de ser executado. A gestão destas entidades é feita através dos caminhos apresentados na Figura 57.

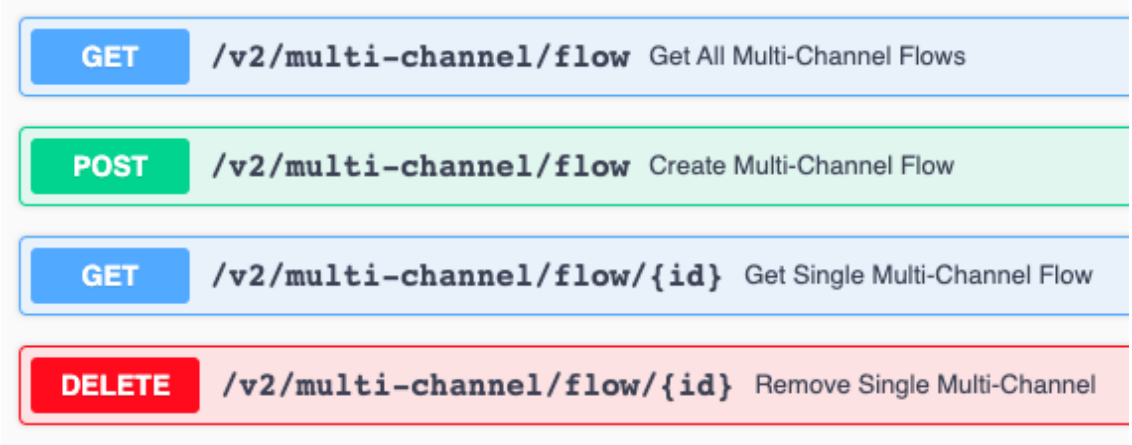

Figura 57 - Serviços relacionados com Flows de Multi-channel

As funcionalidades apresentadas na Figura 57 não apresenta todas as caraterísticas do padrão CRUD. O serviço de edição não é disponível ao utilizador, uma vez que, durante o desenvolvimento da funcionalidade, não se havia chegado a consenso sobre o comportamento da edição. Uma vez que os Flows podem ser configurados para terem uma duração de vários dias, a edição do mesmo colocaria em causa o comportamento dos fluxos em execução. No entanto, o utilizador tem as opções de criar e remover os Flows, que lhe permite fazer a gestão dos mesmos, embora não ao nível 2 de REST.

Para terminar a funcionalidade do Multi-channel existe o serviço que retorna os dados dos fluxos que foram executados, tal como mostra a Figura 58.

Figura 58 - Serviços relacionados com Reports de Multi-channel

Este serviço retorna os dados sobre os Flows que foram executados e os dados globais de cada etapa, pelo que o utilizador poderá consultar qual o comportamento dos destinatários das mensagens.

Como terceiro e último caso de uso a analisar existe o Verify. Este caso de uso, tal como os mencionados acima, disponibiliza um caso de uso específico, envolvendo uma série de mensagens.

Na Tabela 23 é apresentado as componentes de um Verify.

Tabela 23 - Componentes para o envio de um Verify

Entidade	**Descrição**
To	O destinatário.
Sender	A identificação do remetente a ser utilizado.
Code	As caraterísticas do código a serem enviadas
Message	A mensagem que acompanha e contextualiza o código.

O Verify, tal como apresentado na Tabela 23, é composto por todas as partes necessárias ao envio de uma mensagem Sms. A diferença para tal é o código. A mensagem é acompanhada por um código, gerado pela Slingshot, que servirá como autenticação do destinatário.

Na Figura 59 é apresentada a estrutura de dados relacionada com um Verify.

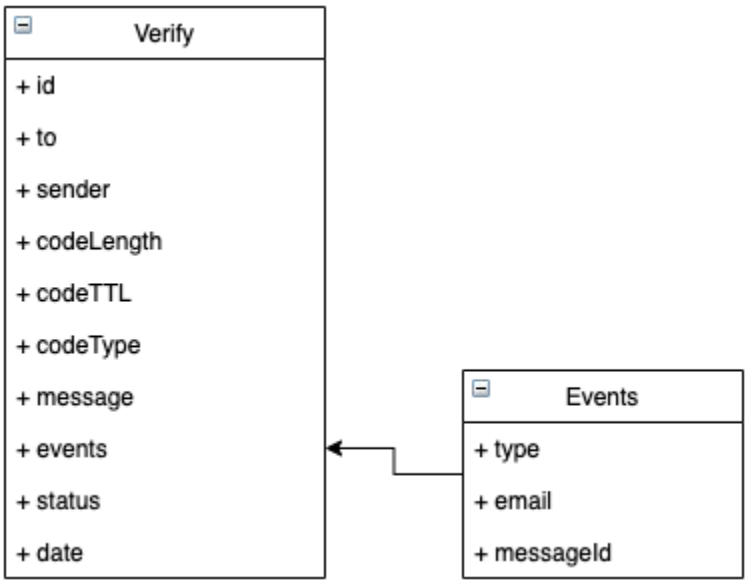

Figura 59 - Estrutura de dados relacionados com o envio de um Verify

Tal como apresentado na Figura 59, a estrutura de dados é bastante simples. A parte diferenciadora para o envio de uma mensagem Sms normal são as carateristicas do código. O cliente pode escolher qual o tamanho, tipo de código (alfabético, numérico ou ambos) e tempo de validade do mesmo.

Fazendo uma análise ao diagrama de sequência na Figura 15, verifica-se que poderão ocorrer alguns casos alternativos. Por exemplo, o utilizador pode requisitar que o código seja novamente enviado, ou a aplicação verifica que não é mais necessário aquele fluxo. Para complementar as ações possíveis num Verify, e de forma a abranger estes casos alternativos, deve ainda existir o caso de uso de cancelamento do Verify, e reenvio do código.

Na Figura 60 são apresentados os serviços disponibilizados em relação ao Verify. O caso do Verify é peculiar em comparação com os casos de uso acima mencionados. De facto, o Verify não é tratado em lado algum como uma entidade de gestão de dados, sendo que a sua natureza é praticamente funcional. A justificação para este comportamento prende-se com a natureza sigilosa do Verify.

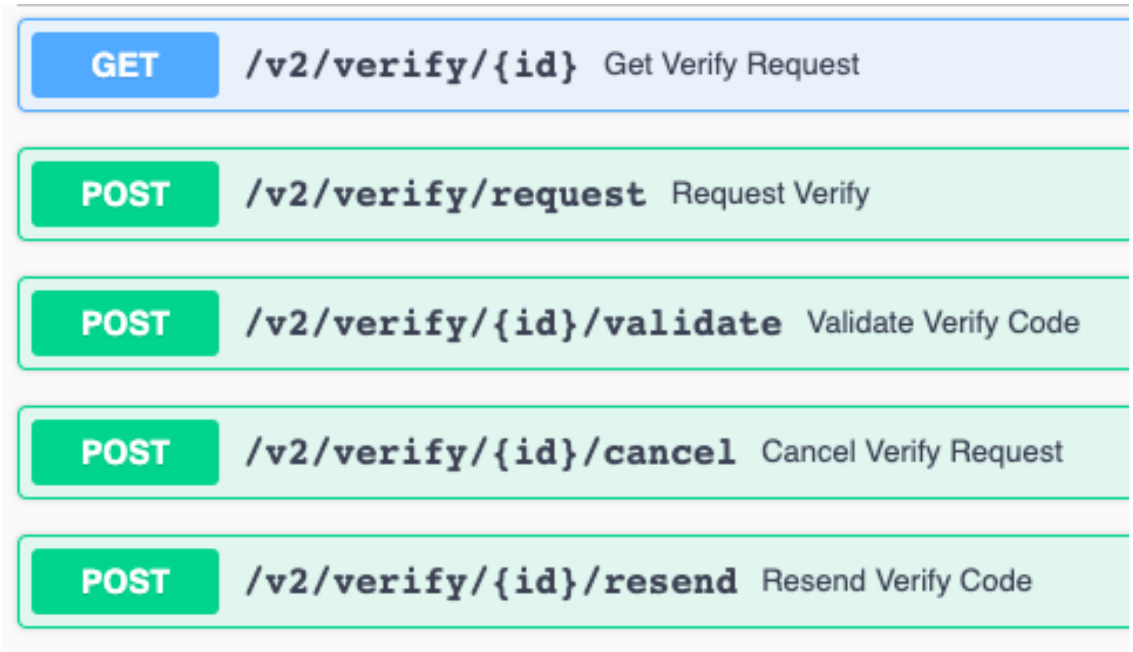

Figura 60 - Serviços relacionados com Verify

O desenho dos caminhos apresentados na Figura 60 permite criar uma perceção de que cada execução do Verify é descartável. Por um lado, os pedidos do método HTTP Post executam alterações sobre um Verify único, promovendo o controlo total por parte do utilizador. Por outro lado, a informação que é originada, como o código e os parâmetros que o geraram são apenas relevantes na janela em que o Verify está ativo e/ou é validado. Por este motivo conclui-se que não existe necessidade de gestão de dados após o envio de cada Verify.

5.2.1.5 Diversos

Nesta categoria serão inseridas as funcionalidades que tanto não encaixam em apenas uma das categorias acima mencionadas, ou que não encaixam em nenhuma. Apesar disso, estas são funcionalidades úteis para a utilização da API.

Nesta categoria existem dois conceitos que surgem em comum a mais que dois canais: os Webhooks e os Groups. Começando pela primeira, estes representam uma escuta de eventos de mensagens. Como podem ser configurados antes do envio de mensagens, deve-se analisar a sua estrutura apresentada na Tabela 24.

Tabela 24 - Componentes de um Webhook

Entidade	**Descrição**
Channel	O canal para captar as ações do webhook.
Actions	As ações a serem captadas.
Url	O url do servidor para o qual é enviada a notificação.

Os Webhooks podem ser configurados para os três canais, sendo que a configuração se aplica apenas para o canal configurado. Uma vez que pode ser encarada como uma entidade CRUD, fará sentido que seja uma própria seção. Os serviços relacionados com essa gestão estão apresentados na Figura 61.

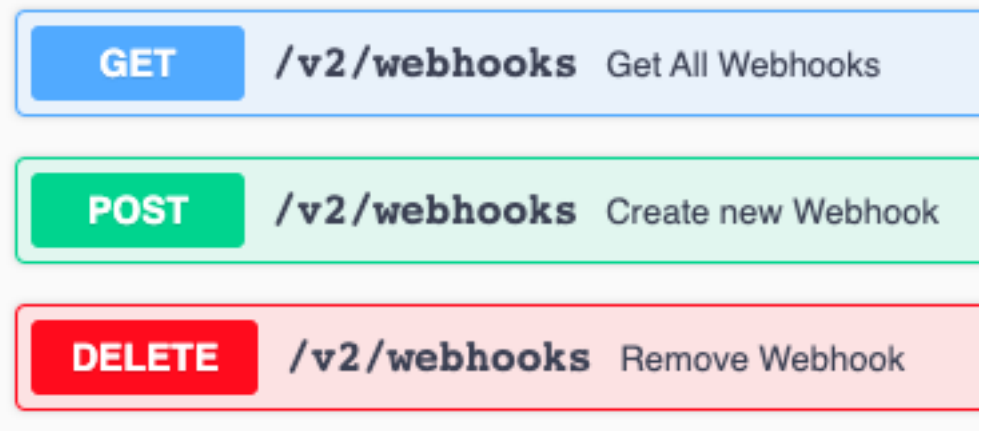

Figura 61 - Serviços relacionados com Webhooks

No entanto, tal como mostra a Tabela 24, a informação a ser gerida pode ser considerada insuficiente para justificar a existência de um método de edição.

O último conceito que justifica colocar-se numa própria categoria é o da consulta de grupos de mensagens, como apresenta a Figura 62. Os grupos são associados no envio das mensagens, e não permitem a sua edição após envio, pelo que as funcionalidades se resumem a listagem dos grupos existentes.

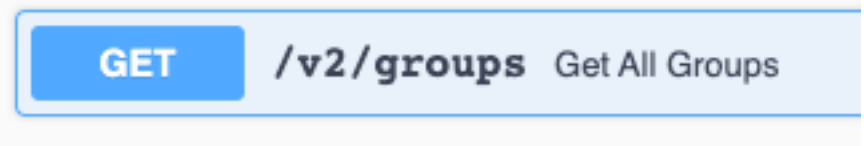

Figura 62 - Serviços relacionados com Groups

Neste momento, a Slingshot apenas permite consultar os grupos que são criados. Isto porque os grupos podem ser criados no momento de envio de uma campanha. No entanto, à semelhança dos remetentes nos canais Email e Sms, existe a possibilidade de gestão de grupos dentro da plataforma.

Finalmente, agrupam-se os restantes serviços nos caminhos apresentados na Figura 63.

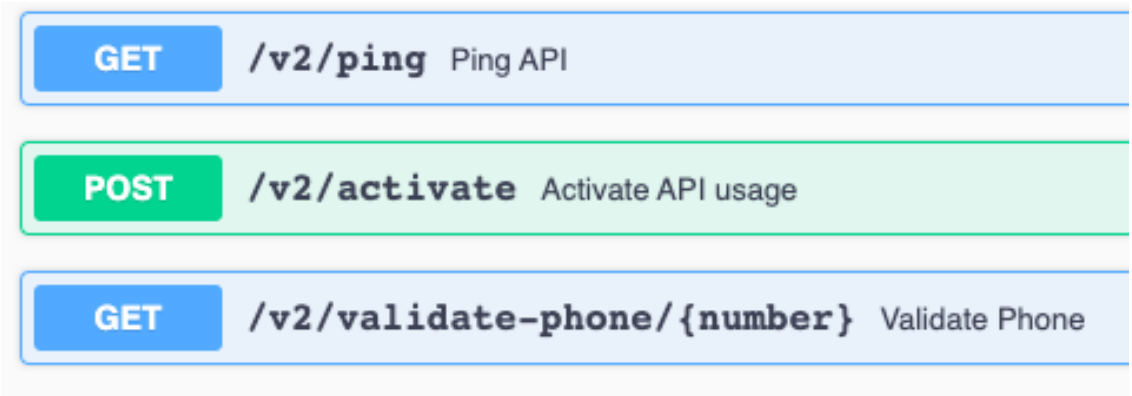

Figura 63 - Serviços relacionados com Utilities

Estas funcionalidades deverão ser das menos utilizadas na API, uma vez que são consideradas um ponto de entrada e aprendizagem na utilização da aplicação. Apesar do seu papel inicial ser fundamental para iniciar o uso da API, os serviços não deverão ter grande procura. Por outro lado, a E-goi disponibiliza métodos alternativos de ativar a API para o utilizador. Por estes motivos, estes métodos são agrupados no fim da API.

5.2.2 Orientações para documentação

Para uma correta documentação da API, deve ter-se em consideração a sua estrutura e componentes. A atual versão da Slingshot está dividida em duas partes: documentação e serviços. Na documentação, a API descreve as funcionalidades que possui, enquanto que nos serviços estão definidas as funcionalidades da API. Esta divisão causa uma desconexão entre descrição e funcionalidade, uma vez que a informação se encontra em dois lugares diferentes.

Para melhor agregar a informação, deve-se juntar a informação relevante aos serviços apropriados, de forma a diminuir o esforço do utilizador no estudo da API. Os cinco grupos definidos acima devem ter uma introdução que contextualiza as funcionalidades que serão apresentadas. Depois, cada conjunto de serviços dentro do grupo deve também fazer uma introdução as entidades que esse conjunto gere. Finalmente, cada serviço deve ter apenas o mínimo de introdução relevante, uma vez que se espera que o utilizador compreenda a funcionalidade do serviço com base na informação apresentada. A Figura 64 apresenta a estrutura da divisão da documentação, tendo em conta as componentes descritas na seção 5.2.1.

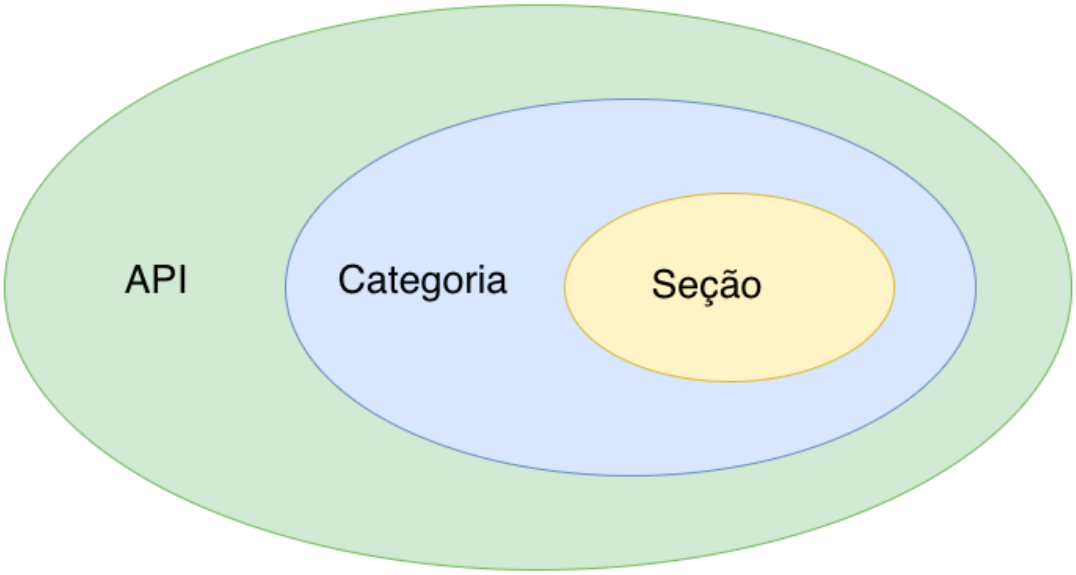

Figura 64 - Conceito para estrutura da documentação

Tal como apresentado na Figura 64, cada componente da documentação deverá estar relacionado com a respetiva componente da API. A API deverá ter uma introdução a mesma, com uma explicação breve das funcionalidades da mesma, contexto de uso e normas gerais de utilização. Um nível abaixo, cada categoria deve introduzir o conceito da mesma, com uma explicação mais detalhada das componentes gerais da categoria. No nível mais baixo da seção, a documentação deve ser extremamente descritiva em relação as funcionalidades contidas

dentro da seção. Desta forma cria-se uma espécie de granularidade, onde a aproximação aos serviços aumenta a especificação da informação relacionada com esse mesmo serviço.

Estabelecidas a estrutura da documentação, é pertinente estruturar a informação dentro de cada método. É necessário discernir qual a informação necessária, qual a informação interessante, e qual a informação acessória. Neste caso, a informação necessária será a mínima necessária á utilização daquele método em específico. No caso de uma API toma a forma do método HTTP, endereço url, parâmetros de entrada e, quando necessário, corpo do pedido. A partir desta informação, um utilizador poderá executar pedidos a API com sucesso. Considerando o caso em que a documentação não esta presente, como acontece atualmente na Slingshot, a falta de maior contexto poderá ser pouco apelativa para o utilizador, que, na maior parte dos casos, sentirá necessidade de executar alguns testes, de forma a melhor compreender o comportamento do método e da API.

É neste ponto que entra a informação interessante ao serviço. Este tipo de informação não é fundamental para utilizar a API, mas ajuda a orientar o utilizador a ligar-se à sua própria aplicação. Casos como descrição dos parâmetros/corpo de mensagem, corpo da resposta ajudam o utilizador não só a compreender a funcionalidade do método como também a preparar as suas aplicações para lidar com pedidos e resposta da API. Desta forma diminui-se a probabilidade de erro dentro da aplicação do utilizador, e aumenta a confiança do mesmo.

Finalmente, existe a informação acessória ao método. Enquanto que a informação interessante contextualiza o método, a informação acessória apresenta alternativas do fluxo. São exemplos desta utilização coisas como exemplos, excertos de código e corpo de mensagem de erro. Esta informação, do ponto de vista da interação com a API, não é de forma alguma necessária para a utilização da API, ou mesmo para contextualizar a mesma. No entanto, ajudam o utilizador a diminui a curva de aprendizagem da utilização da API. Excertos de código oferecem um termo de comparação de utilização da API, que o utilizador apenas necessita de inserir diretamente na aplicação. As mensagens de erro, por outro lado, permitem reduzir o escopo de cenários alternativos. Controlo e diminuição da entropia de resultados de um pedido permite melhor gestão de erros do lado da aplicação do utilizador.

Na Figura 65 é apresentado uma sugestão de estrutura da documentação de um serviço. Esta documentação junta os tipos de informação e componentes apresentados em cima. À primeira vista, é visível não só o método HTTP e o endereço, como qual o corpo do pedido (uma vez que se trata de um método que o pede). Em destaque também é visível uma breve explicação de cada parâmetro do corpo da mensagem. Finalmente, do lado direito existem abas a esconder a informação acessória, como as mensagens de erro e exemplos/excertos de código.

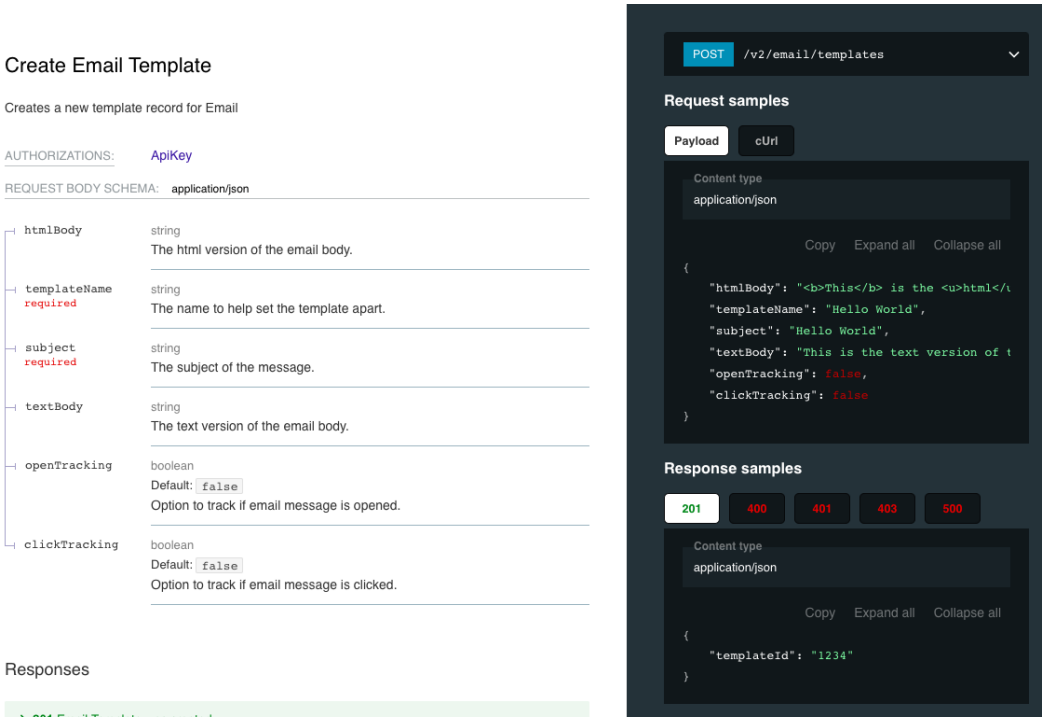

Figura 65 - Sugestão de estrutura de informação de um método

A estrutura da Figura 65 será o padrão que será utilizado para a documentação de cada serviço.

5.3 Arquitetura do Sistema

Antes de iniciar a implementação da API deve considerar-se como esta se integrará com a estrutura pré-existente da E-goi. A Slingshot encontra-se localizada num único servidor, na arquitetura demonstrada na Figura 66. Note-se que a ligação entre a aplicação do cliente e o servidor de envio de email traduz-se na ligação SMTP ao servidor de envio.

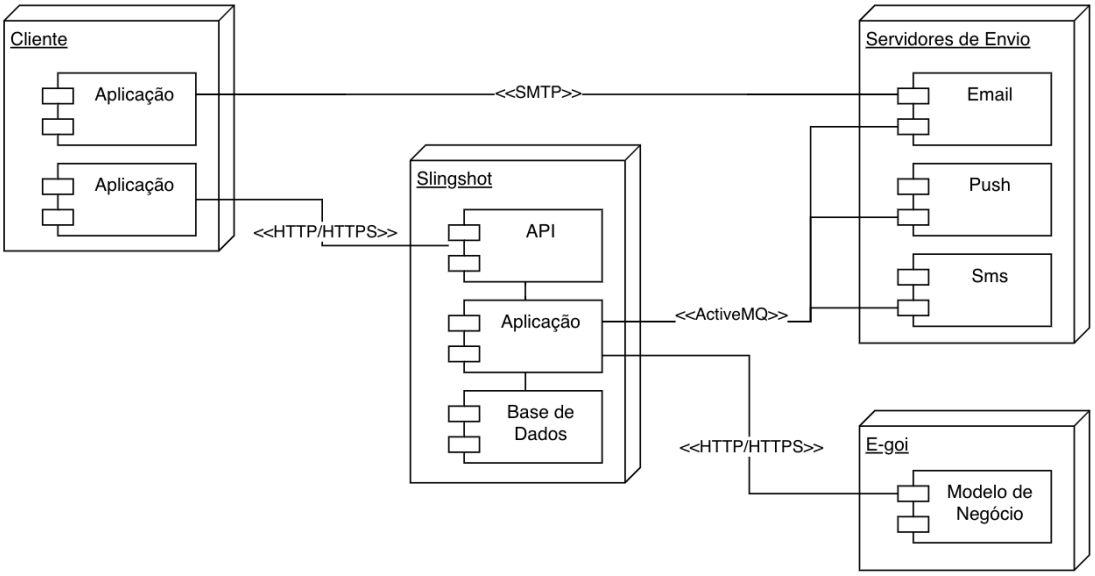

Figura 66 – Diagrama de implantação da arquitetura atual da Slingshot

Uma das soluções consideradas foi a de refazer a aplicação da Slingshot, tendo em conta padrões de desenvolvimento. A arquitetura da mesma está apresentada na Figura 67.

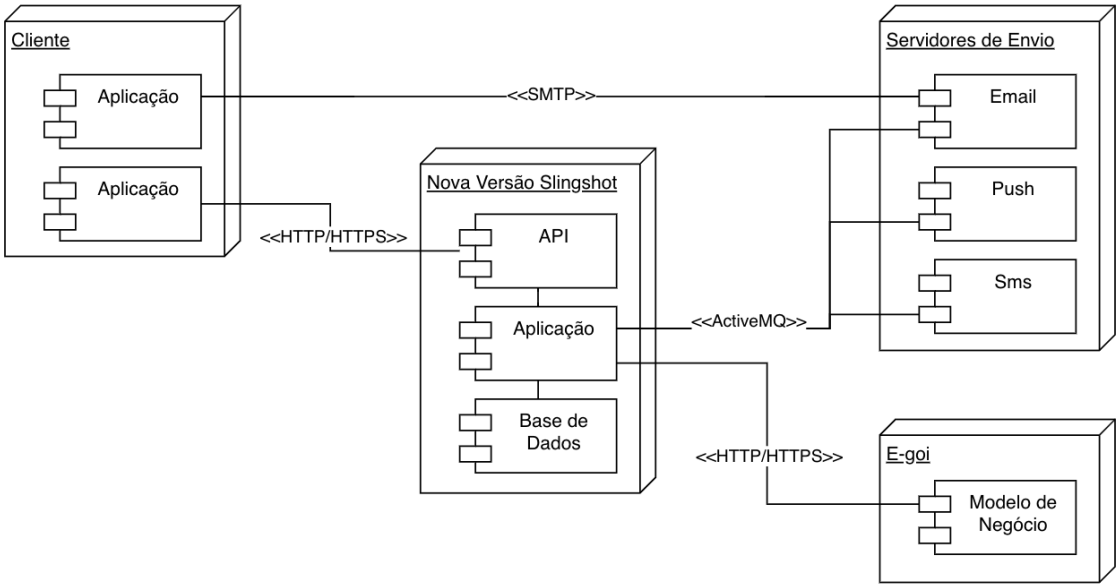

Figura 67 – Diagrama de implantação da proposta da Nova Versão (Refazer API)

As vantagens desta abordagem prendem-se com uma correta implementação do produto, que poderá implicar melhoria de performance e diminuição dos recursos necessários para correção e implementação.

No entanto, esta abordagem envolve um grande esforço de desenvolvimento, entre o estudo da plataforma, desenho, implementação e testes. Por outro lado, o investimento na plataforma atual corre o risco de estagnar e adiar correções e melhorias necessárias. Tendo em conta o ambiente de Continuous Integration/Continuous Delivery praticado pela E-goi, esta opção é fortemente desencorajada.

Outra abordagem envolve o desenvolvimento de uma nova versão da Slingshot, desenvolvida de forma paralela, para potencial substituição da primeira versão da mesma. Esta implantação é apresentada na Figura 68. Esta abordagem evita que a versão anterior seja abandonada durante o período de desenvolvimento da nova versão.

As desvantagens desta abordagem envolvem a manutenção de duas aplicações, potencialmente em ambientes diferentes. Desta forma é aumentado o custo de manutenção da Slingshot. Por outro lado, de forma a privilegiar a nova versão é necessário que a versão anterior seja colocada em detrimento e eventual abandono. Garantir que a transição dos clientes da antiga para a nova seja tranquila envolve um esforço muito grande, podendo demorar um longo período de tempo, e mesmo assim não resolver todas as dependências de clientes. Finalmente, a nova versão poderá estar sujeita a problemas que tenham sido previamente resolvidas na versão anterior.

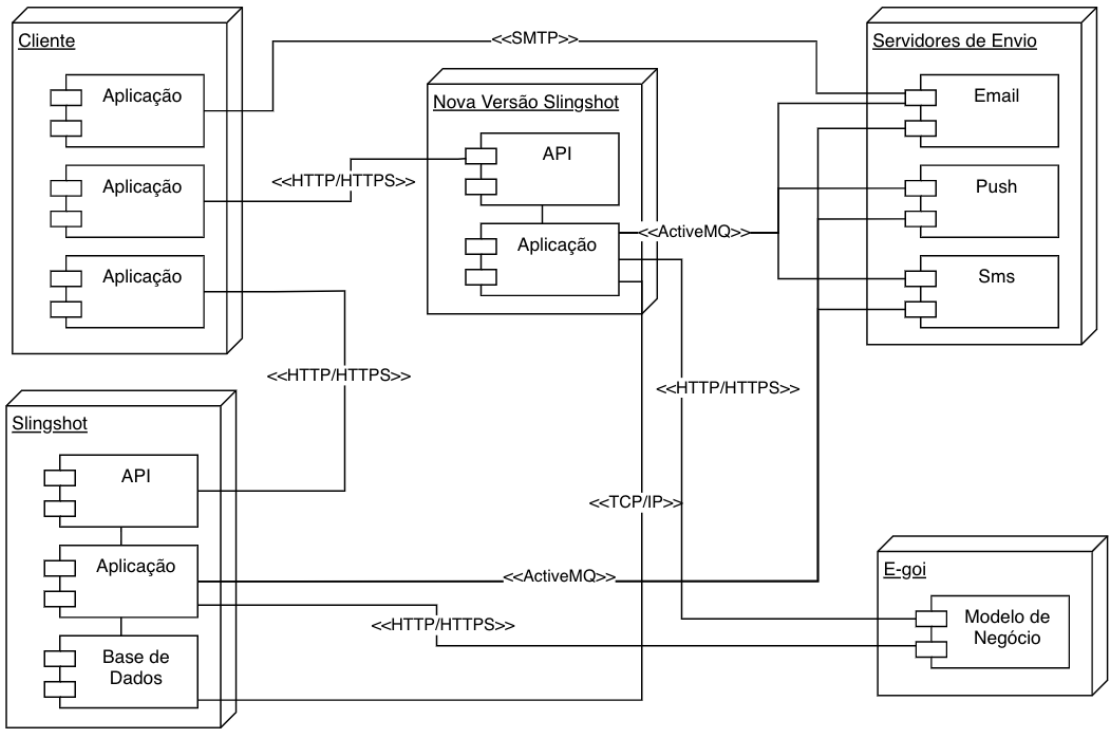

Figura 68 - Diagrama de implantação da proposta da Nova Versão (Nova aplicação)

A última abordagem a analisar será o de desenvolver uma nova camada, que fará a comunicação entre o cliente e a versão anterior, apresentada na Figura 69. Esta abordagem permite o desenvolvimento de melhorias e novas funcionalidades da versão anterior. Também permite manter um *failsafe* na versão anterior, no caso de indisponibilidade. Finalmente, a colocação deste intermediário permite, no futuro, adaptá-lo para um *load balancer/single entrypoint*, ou até substituir a própria Slingshot.

As vantagens desta abordagem são semelhantes aquelas encontradas na abordagem anterior. A versão atual da Slingshot é preservada e pode continuar a ser desenvolvida. A principal diferença prende-se com o investimento necessário para desenvolver a nova versão. Esta será uma espécie de interprete entre a Slingshot e aplicações clientes, mascarando a complexidade de algumas ações. Outra vantagem prende-se com a gestão de novos servidores. Neste momento a Slingshot encontra-se implantada num único servidor. De forma a escalar a capacidade da aplicação pode tornar-se necessário implantar a Slingshot noutros servidores. De forma a manter um único ponto de entrada para todos os clientes, utilizar a nova versão como *gateway* de todos os servidores abstrai o cliente da gestão da Slingshot.

No entanto, torna-se necessário manter especial atenção a alterações de contrato da Slingshot. A nova versão estará diretamente dependente da mesma, pelo que qualquer alteração na Slingshot deve ser refletida na funcionalidade homóloga aquela que foi a funcionalidade alterada. Isto implica algum esforço de manutenção, e aumenta a probabilidade de instabilidades.

Outra desvantagem prende-se com o inconveniente da criação de mais uma camada de processamento do pedido. Efetivamente, um pedido do utilizador é representado por dois: o primeiro entre o utilizador e a nova versão, e o segundo da nova versão para a Slingshot.

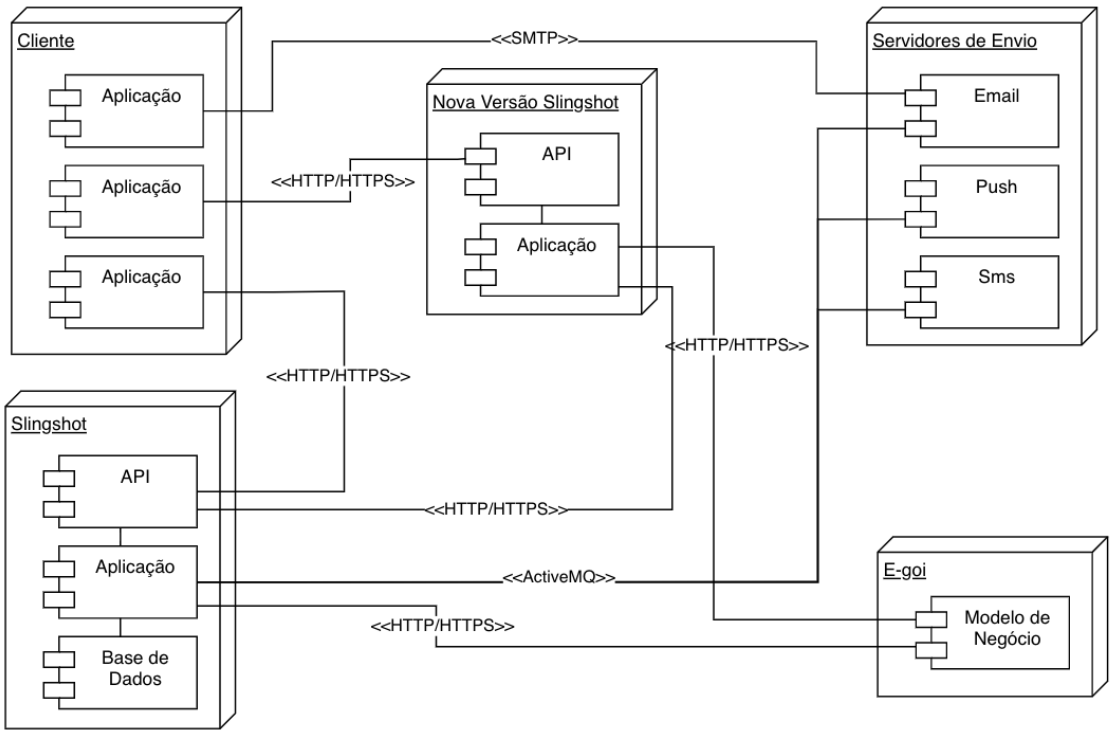

Figura 69 - Implantação da nova versão da Slingshot

A solução que foi selecionada foi a solução apresentada na Figura 69. Esta opção foi tomada tendo por base a análise feita acima entre as propostas, o levantamento de requisitos e meios disponibilizados pela E-goi, e a analise de valor apresentada na seção 3.2.

Inicialmente a estrutura da aplicação a ser adotada seria uma de microsserviços. Esta abordagem teria como vantagem dividir a carga de pedidos feitos à API, e encapsular as funcionalidades de cada um dos cinco conjuntos de funcionalidades (Email, Sms, Push, Casos de uso e Outros). No entanto, a carga de manutenção de cada uma das aplicações, aliada ao facto de a nova versão fazer os pedidos à Slingshot, não justificou esta implementação.

6 Implementação da Solução

Neste capítulo será apresentado o processo de implementação da solução. Será feita uma análise de como os componentes da solução foram desenvolvidos, quer a aplicação em si, quer a documentação que acompanha o contrato para os utilizadores. Todas as decisões tomadas são apresentadas e justificadas, de forma a contextualizar a solução final.

6.1 Estrutura da Aplicação

Tendo definido e desenhado o contrato, passou-se a etapa de implementação do contrato. Para tal desenvolveu-se uma aplicação que atua como API.

Para auxiliar a implementação foi adotada a Framework Spring. Esta Framework foi desenvolvida para Java, que facilita a gestão de dependências, gestão de eventos e *data binding*.

Alternativas a Spring incluem Node/Express.js, ASP.NET, entre outras. No entanto, Spring foi selecionado por várias razões. Em primeiro lugar, é uma das frameworks privilegiadas na E-goi. A Slingshot foi desenvolvida através da mesma Framework, e a utilização de ferramentas semelhantes diminui o esforço de aprendizagem. Não havendo garantias da continuidade do desenvolvedor na equipa, garantir que quem fica compreende como desenvolver e manter a aplicação facilita o seu trabalho. Outra razão prende-se com a linguagem utilizada. Java facilita o processamento de dados estruturados, diminuindo a probabilidade de resultados inesperados. Finalmente, Spring oferece várias garantias de estabilidade da aplicação. A gestão de dependências é bastante simples utilizando Gradle ou Maven. Spring também possui várias integrações e ferramentas, entre as quais OpenAPI, que facilita não só o desenvolvimento e manutenção da aplicação, como atualização das mesmas.

Após a escolha das tecnologias, desenhou-se o processamento de pedidos feitos a nova versão. Este desenho permite definir quais as componentes que necessitam de ser desenvolvidas, de forma a reduzir e modularizar o processo de desenvolvimento da solução. O resultado deste desenho é apresentado na Figura 70.

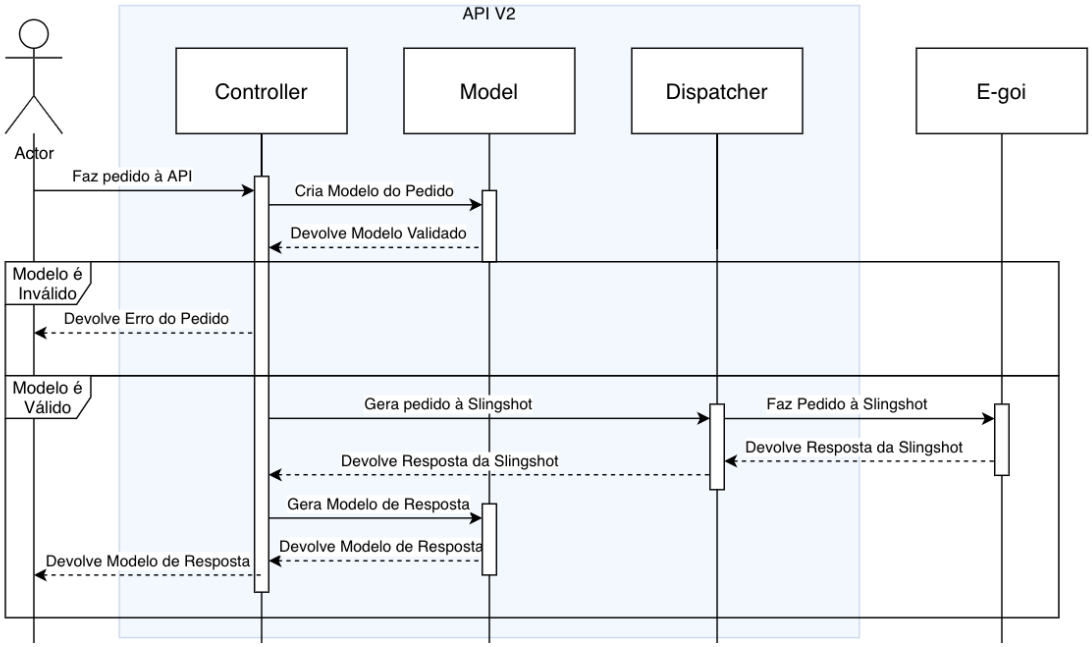

Figura 70 - Diagrama de Sequência de um Pedido a API V2

Tal como apresentado na Figura 70, a API será composta por três grandes componentes. O primeiro componente é o controlador, que fará o processamento do pedido, desde que é recebido até que envia uma resposta. O segundo componente são os modelos, que estruturam os dados com as quais a aplicação lida e os validam. O terceiro componente é o Dispatcher, que é o responsável por fazer os pedidos a E-goi. Estes pedidos podem ser tanto para a Slingshot (onde a base das funcionalidades existe) como à API interna da E-goi, que é utilizada para a plataforma da E-goi.

Tendo definido as componentes apresentadas na Figura 70, desenvolveram-se e especificaram-se as suas componentes internas. Estes componentes que fazem parte da aplicação estão apresentadas na Figura 71. A aplicação é divida em duas grandes partes. A primeira é a gestão de pedidos feita pelos Modules. O pedido é recebido e contruído numa entidade que a aplicação pode processar. Depois, o pedido passa para a segunda parte do Dispatcher, que gera o pedido a ser feito a Slingshot e o executa. Ao receber o resultado este é convertido num objeto que o Modules pode processar. Finalmente, Modules constrói o corpo de resposta e envia para o cliente que fez o pedido. Se em qualquer ponto falhar alguma validação, ou ocorrer alguma falhar de processamento, é lançada uma exceção a partir da componente Exceptions, que retorna ao cliente a mensagem de erro própria.

A componente se Security é a primeira componente a processar o pedido. A função dela é de verificar a existência da apikey no pedido feito pelo cliente. Por este motivo utiliza a componente das Exceptions, lançando o erro apropriado. Existiria a hipótese de, neste passo do processo, validar a apikey, fazendo um pedido à Slingshot a confirmar a sua validade. Este passo não foi implementado, uma vez que poderia comprometer a velocidade de resposta da aplicação.

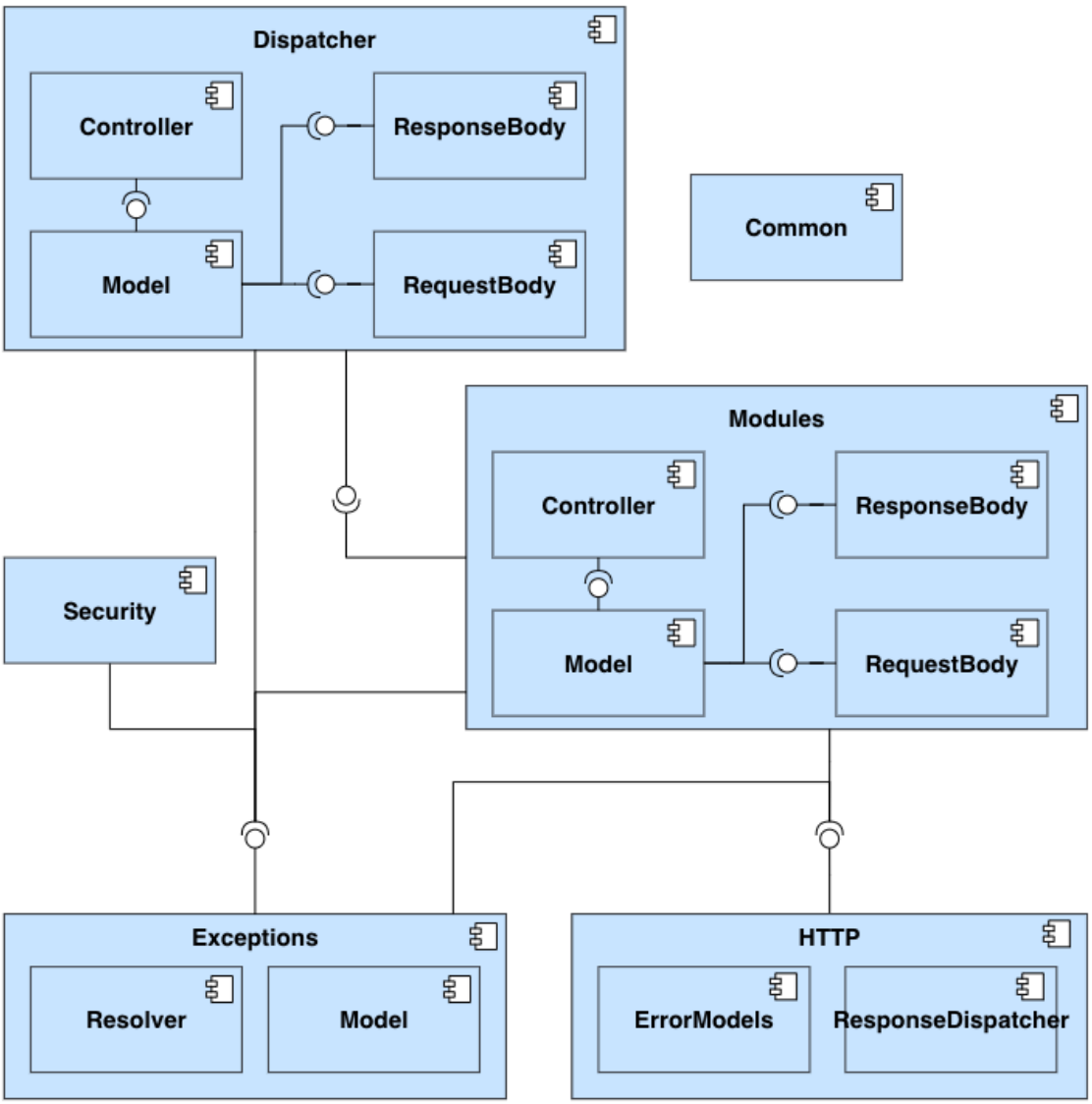

Figura 71 - Diagrama de componentes da aplicação

Finalmente, todas as componentes estão de alguma forma ligadas à componente de HTTP, que é responsável por gerar uma resposta ao cliente. A implementação deste componente afunila o processamento de pedidos, garantindo que qualquer pedido possui uma mesma base de processamento, e promovendo a reutilização de código.

6.2 Gestão de erros

No processamento de pedidos, desde receção do mesmo até a devolução de uma resposta, existem vários cenários que poderão não produzir o efeito esperado. Idealmente, pedidos semelhantes produzem respostas semelhantes. Desta forma diminui-se a variância de respostas para o cliente. Para tal torna-se necessário simplificar o fluxo de processamento de dados, e identificar potenciais pontos de quebra.

De forma a cobrir o maior número de casos, foram desenhadas as seguintes exceções:

- ApiKeyNotPresentException – para casos em que a apikey não foi encontrada no pedido;
- BadRequestException – para casos em que o pedido foi mal construído;
- InternalServerErrorException – para casos em que a aplicação não consegue resolver o pedido;
- InvalidApiKeyException – para casos em que a apikey não é válida;
- MissingRequestBodyException – para casos em que o pedido não tem um corpo de mensagem quando deveria ter;
- NotFoundException – para casos em que os recursos necessários não existem;

Para fazer o lançamento de erros dentro da aplicação foi utilizada a anotação @ControllerAdvice. Esta anotação permite estabelecer uma classe que terá definições como outras classes lidam com exceções.

A resolução das exceções são será o retorno de uma mensagem de erro para o cliente com a mensagem apropriada. Esta será composta pelas componentes apresentadas na Tabela 25. As Esta estrutura foi escolhida por ser a estrutura utilizada pela E-goi em outras API's. Apesar disso, considerou-se que a estrutura é adequada, uma vez que permite devolver a informação relevante, como o código de erro, o tipo de erro no contexto da aplicação, e uma mensagem a descrever o problema.

Tabela 25 - Estrutura de uma mensagem de erro

Parâmetro	**Descrição**	**Exemplo**
type	O tipo de erro no contexto da aplicação. É uma espécie de chave para o cliente capturar a resposta e processá-la da forma que melhor entender.	missing_parameters
title	O título do erro no contexto HTTP	BAD REQUEST
status	O código de erro.	400
detail	Uma descrição sucinta do problema, acompanhada de uma potencial solução. Permite ao *developer* compreender o que deve fazer.	Missing the following parameters: htmlBody, senderId

6.3 Autenticação

De forma a autenticar o utilizador é necessário que este apresente alguma informação que claramente o identifique para com a API. Tal como mencionado na seção 2.1.1.1, existem três formas de autenticar o utilizador para a API: utilizando uma apikey, OAuth ou login.

Neste momento a Slingshot utiliza uma apikey, única para cada cliente, para atingir este efeito. Esta forma de autenticação foi aproveitada para a nova versão. Esta decisão foi tomada uma vez que a estrutura para a validar do lado da Slingshot já existe. Fazer o reaproveitamento desse componente permite reduzir o tempo de resposta a um pedido e evitar que os clientes utilizem dois métodos de autenticação nas suas aplicações. Esta apikey também é utilizada em outras API's da E-goi, pelo que é reforçado a sua reutilização. Comparando as restantes formas de autenticação, também permite uma menor carga de processamento. Uma vez que a Slingshot é uma aplicação autossuficiente e sem dependências de terceiros, não fará sentido utilizar OAuth como método de autenticação. Finalmente, a utilização de login exige que seja utilizada um conjunto de informação privada e potencialmente pessoal do cliente. Combinações com base num identificador e código secreto representam um potencial perigo de invasão de privacidade. Por outro lado, representa um passo extra para a aplicação, uma vez que exige o maior processamento de dados.

Para além da forma de identificação também se torna necessário definir como se lidará com a mesma. Após a identificação do cliente e validação dele, os seus pedidos subsequentes deverão ser identificados. Existem várias hipóteses, entre as quais a geração de um token por parte do servidor, manutenção da sessão dos clientes por parte do servidor, ou envio de uma chave. Num contexto de API Economy, onde otimização de processos é fundamental, a solução com menor processamento deverá ser a preferida. Por outro lado, a sessão vai conta os princípios REST que se pretendem implementar. Deste modo, a utilização de uma apikey estática poderá dar as maiores garantias de velocidade, aliada a uma camada de segurança razoável.

Em relação ao envio da apikey, foi considerado o envio do mesmo através dos seguintes parâmetros: Cookie e Cabeçalho. Na Figura 72 é apresentada as diferenças entre o envio de ambas.

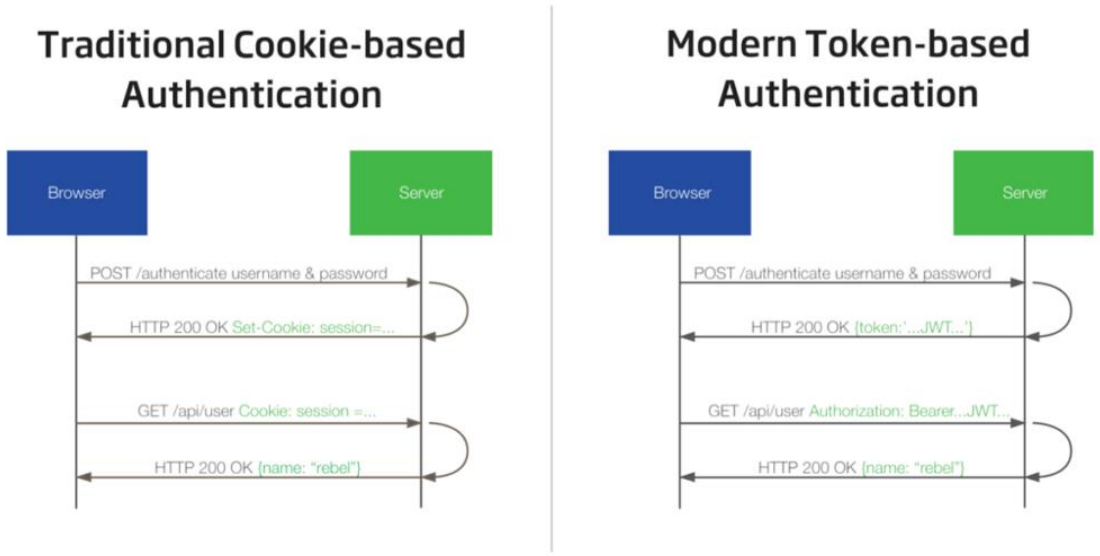

Figura 72 - Diferença entre autenticação através de Header e Cookie (Chang, 2018)

No cado da Figura 72, o Browser faz o login antes de receber o token gerado. Este passo não acontece na API, exceto no serviço de ativação da API. A principal diferença entre os dois está no local onde a apikey é armazenada. No caso do cookie, esta é armazenada num ficheiro dentro da máquina do cliente, gerado automaticamente pela aplicação. No caso do cabeçalho, o utilizador consegue ter mais controlo, podendo armazenar a apikey em locais como ficheiro, base de dados, ou injetando diretamente no código. Pelo maior grau de flexibilidade que oferece, optou-se por enviar a apikey através de cabeçalho.

Dentro do código, foi desenvolvido um método que valida se o pedido feito contém a apikey. Este método foi depois associado a um intercetor, que capta o pedido assim que este é recebido, e corre o método. Este método é apresentado na Figura 73.

```
@Component
public class ApiKeyHeaderHandlerInterceptor extends HandlerInterceptorAdapter {
    private static final Logger LOGGER = Logger.getLogger(ApiKeyHeaderHandlerInterceptor.class.getName());

    @Override
    public boolean preHandle(HttpServletRequest request,
                             HttpServletResponse response, Object handler) throws ApiKeyNotPresentException {
        String apiKey = request.getHeader( name: "ApiKey") == null ? "" : request.getHeader( name: "ApiKey");
        LOGGER.info(() -> "ApiKey is: " + apiKey);
        if (apiKey.equalsIgnoreCase( anotherString: "")) {
            throw new ApiKeyNotPresentException(request);
        }
        return true;
    }

}
```

Figura 73 - Interceptor de ApiKey

Utilizando o Interceptor apresentado na Figura 73 pode ser feita uma análise inicial, verificando se o pedido possui a identificação do utilizador. No caso da API não encontrar este pedido, esta retorna automaticamente uma resposta de erro do tipo 401 – Não Autorizado.

Uma vez que nem todos os serviços da API necessitam da ApiKey, foram acrescentadas as exceções configuradas na Figura 74.

```
@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(this.apiRequest)
            .addPathPatterns("/v2/**");
    registry.addInterceptor(this.apikeyHeader)
            .addPathPatterns("/v2/**")
            .excludePathPatterns(
                    "/v2/activate",
                    "/v2/ping",
                    "/v2/redoc",
                    "/v2/openapi",
                    "/v2/supportedCountries");
}
```

Figura 74 - Configuração dos Interceptors

Na Figura 74 é apresentada a configuração de dois Interceptors. O primeiro funciona apenas como um registo de todos os pedidos feitos a API. O segundo é onde é validada a ApiKey. Foram expluídos os métodos de ativação, ping da API e consulta dos países suportados. A ativação da API não fará sentido necessitar autenticação, uma vez que representa a porta de entrada para o uso da API. Os serviços de ping e consulta dos países suportados são pedidos do tipo *Get*, que não necessitam de informação do utilizador e possuem mínimo de processamento dentro da aplicação. Os restantes três endereços ('/redoc' e '/openapi') funcionam como a documentação da API. O endereço '/openapi' gera o contrato da API na especificação OpenAPI, que deverá ser disponibilizada sem necessidade de autenticação. O endereço '/redoc' retorna a documentação da API no formato Redoc, que também deverá ser universalmente disponibilizada.

De notar que o caminho de validação de números de telemóvel necessita de autenticação. Esta validação exige processamento por parte da Slingshot, pelo que permitir o seu acesso a qualquer utilizador pode inferir alguns riscos. A decisão de necessitar autenticação foi tomada para limitar não só a utilização da API por utilizadores que não sejam clientes E-goi como também evitar ataques do tipo *code injection*.

6.4 Versionamento

De forma a diferenciar as versões da API, é necessário definir como a evolução da API seria representada. É necessário definir não só como a nova versão é diferenciada da Slingshot, e como a evolução das duas afetará o versionamento.

Uma vez que a Slingshot e a nova versão ficarão implantadas em máquinas diferentes, pode fazer sentido que possua um endereço diferente. Neste momento, a Slingshot encontra-se hospedada no servidor 'www51.e-goi.com/'. Este endereço deixa transparecer duas

caraterísticas da Slingshot. A primeira é que não existe um endereço específico para a Slingshot, sendo que o utilizador não tem uma referência de que é a aplicação que pretende. A segunda é que a Slingshot se encontra hospedada no servidor 51. Este tipo de informação, para além de não ser necessária para o utilizador, não esconde a existência de outras máquinas. Esta é uma falha de segurança, sendo que um utilizador mais malicioso poderá experimentar aceder a outras máquinas com o intuito de explorar a arquitetura do sistema ou até desativar o servidor.

Tendo em conta a informação descrita, estabeleceu-se que a nova versão terá o próprio endereço, que abstrai a complexidade. A implantação desta nova API será a base a partir da qual qualquer nova versão será referenciada. Possuir um endereço único para aceder a aplicação simplifica o acesso a API, uma vez que o utilizador possui apenas um endereço único.

Mesmo com a nova versão da API num endereço diferente, é necessário definir qual a versão a ser chamada. As opções disponíveis serão através do cabeçalho, query, corpo da mensagem ou no endereço do pedido. Uma vez que a nova versão suporta serviços ao nível 2 da aplicação de Rest, existirão pedidos que não suportarão corpo de mensagem. Outra opção que pode ser removida é na query, visto que, em alguns casos, o endereço poderá correr o risco de incorrer num erro HTTP do tipo 414 – URI Demasiado Comprido. Apesar deste erro não ser lançado neste momento pela Slingshot ou pela nova versão, é considerado boa prática evitar que ocorra. A opção escolhida foi através do endereço, sendo a opção com a versão mais visível para o utilizador.

Como mencionado acima, a Slingshot não possui sistema de versionamento, sendo que os clientes utilizam automaticamente a versão mais atualizada da API. Esta abordagem não é necessariamente um problema, uma vez que as alterações feitas não afetam negativamente os serviços previamente disponibilizados. Por outro lado, não fará sentido disponibilizar versões anteriores após acrescentar novas funcionalidades, com o risco do utilizador chamar um serviço numa versão da API que não o disponibiliza. Finalmente, a disponibilização de várias versões da API implica custos acrescidos de hospedagem da aplicação.

Pelos motivos mencionados no parágrafo anterior, pode considerar-se disponibilizar alguma forma de versionamento da API na nova versão. No entanto, esta opção foi descartada por enquanto, uma vez que as dependências com outras API's influenciam as funcionalidade disponibilizadas. Apesar da arquitetura atual da nova versão, não será esperado que uma versão esteja dependente de outra por duração indeterminada.

Optou-se por apresentar apenas o valor da versão do tipo *major*. Desta forma diferencia-se claramente as diferenças entre as versões disponibilizadas. O endereço para a nova versão será algo como 'slingshot.egoiapp.com/v2/'.

6.5 Geração de SDK

Para facilitar a integração da nova versão da API, poderá ser vantajoso explorar formas alternativas de disponibilização da API. Uma dessas formas é a geração de SDK's. A principal ferramenta utilizada para gerar os SDK foi a ferramenta OpenAPI Generator (Cheng, Schubert, & Pietrzyk, 2020). Esta é uma das ferramentas desenvolvida com base na especificação OpenAPI. Como a nova versão da API adota a especificação OpenAPI, é natural que se utilize esta ferramenta.

O processo de geração de SDK's no fluxo normal de implantação da API foi através do ficheiro Jar da ferramenta. Uma vez que a aplicação corre em ambiente Java, não existem dependências da aplicação ou máquina de implantação, e é possível gerar automaticamente SDK's para as linguagens selecionadas.

No entanto, esta componente não foi desenvolvida com sucesso. O principal motivo para o insucesso deve-se à incapacidade atual da OpenAPI Generator em gerar com sucesso corpos de mensagens do tipo "oneOf". Esta notação indica que existem dois tipos de corpos de mensagem, e é utilizada, por exemplo, nos Alerts, onde cada canal possui o próprio corpo de mensagem.

Uma proposta de resolução deste problema seria simplificar a especificação, de forma a remover estas dificuldades na geração do SDK. No entanto, esta solução foi rejeitada, uma vez que se considera a apresentação das alternativas do corpo das mensagens mais vantajosa para o utilizador do que a utilização de SDK's.

Outra proposta de resolução seria gerar os SDK's e corrigir individualmente os problemas que ocorressem ao processar o SDK. Inicialmente tentou-se seguir por esta alternativa, corrigindo os erros que fossem aparecendo na geração do SDK. Esta proposta acabou por ser eventualmente abandonada. Os motivos para tal foram o investimento necessário para corrigir os SDK's para cada linguagem não justificaria os benefícios para os utilizadores. Por outro lado, a documentação atual da API pode ser considerada clara o suficiente, sendo que o utilizador terá maior flexibilidade para integrar a API como desejar. Finalmente, a correção manual de erros nos SDK's não permite a automação do processo de implantação, obrigando a um investimento de cada vez que é feita uma alteração à API.

6.6 Documentação da API

Estando definido o contrato (ao longo da seção 5.2.1), deve procurar escrever o contrato numa linguagem padrão. Para a definição do contrato foi utilizada a especificação OpenAPI. A especificação surgiu de uma evolução da especificação Swagger em 2016, e atualmente é cuidada pela OpenAPI Iniciative, sobre a alçada da Linux Foundation. É principalmente usada para descrever API's REST, integrando informação como endereços, parâmetros e autenticação (SmartBear Software, s.d.).

A especificação OpenAPI pode ser descrita em JSON ou em YAML. Estas são linguagens que podem ser facilmente interpretadas por pessoas e máquinas, que facilita tanto quem a define ou estuda como ferramentas e aplicações. Esta versatilidade resulta na existência de várias ferramentas de edição da API.

Para facilitar a visualização da documentação foi utilizada a framework Redoc. Esta framework permite apresentar documentação de APIs de forma elegante e interativa. O suporte da especificação OpenAPI é a principal vantagem desta framework, juntamente com outras como especificações adicionais e pouca pégada de espaço e processamento.

6.6.1 Introdução

De forma a contextualizar um novo utilizador, fez-se uma pequena introdução é à API. Nesta introdução é dada uma breve explicação do que são mensagens transacionais, em que contextos são usadas, e quais as diferenças para mensagens de marketing.

Também se achou pertinente fazer um *disclaimer* para o uso da API. Na documentação da Slingshot não existe nenhum aviso ou precaução para o uso da API. Apesar disso, a E-goi reserva-se o direito de limitar e até bloquear os envios de um utilizador que coloque em causa a integridade da Slingshot. Nesta caso, uma mensagem a explicar as consequencias do mau uso da API permite notificar os utilizadores para as consequencias do mau uso da API e desencorajar potenciais transgressores.

Finalmente, após a introdução da API apresenta-se como o utilizador pode autenticar-se perante a API. A este tópico é dedicado uma seção, contendo informação como o utilizador pode começar a utilizar a API, e como doravante deverá integrar o meio de autenticação nos pedidos que fará a ela. São apresentadas as duas formas de geração da Apikey: através da plataforma E-goi, e através dos serviços reservados para o efeito.

6.6.2 Email

Apesar de uma mensagem de email ser simples na sua essencia, a Slingshot disponibiliza várias funcionalidades e formas alternativas de como enviar uma mensagem de email. Espera-se que um utilizador se torne experiente suficiente para customizar os seus envios, e garantir maior controlo sobre o resultado esperado. Como tal, para cada serviço definido, será feita uma pequena introdução ao contexto da entidade em questão.

Começando pelo envio de emails, é feita uma pequena introdução à estrutura de um email. É esperado que um utilizador da Slingshot tenha o conhecimento básico sobre o envio de emails, pelo, neste caso, é mais vantajoso limitar a informação ao mínimo. Após o envio de uma mensagem email, este passa por vários estados. De forma a contextualizar o utilizador nos passos necessários ao envio de uma mensagem email, e orienta-lo para definição de webhooks ou semelhantes, é apresentado um esquema de estados, juntamente com uma breve explicação de cada um.

Na secção seguinte é apresentada a forma de enviar anexos nas mensagens de email. Um anexo possui várias opções de configuração, como a localização deste no email ou tipo de dados. Assim, é explicada como o utilizador pode inserir os anexos do email. Tamém existe uma listagem dos tipos de ficheiros suportados pela Slingshot e limite de tamanho.

Os emails da Slingshot têm a opção de serem registadas com condições especiais, que podem ser utilizadas em contexto júridico. Esta funcionalidade tem o nome de Registered, e, sendo uma funcionalidade especifica da Slingshot, fará sentido que esteja inserida nesta secção.

Finalmente, uma mensagem de email pode ser escrita utilizando códigos de personalização. Estes códigos serão substituídos pela Slingshot no momento de envio, pelo que esta informação também é relevante de ser apresentada nesta secção.

Depois do envio passa-se para os métodos de configuração prévia para o envio de uma mensagem de email. Aqui insere-se os domínios, templates e remetentes. Cada um destes componentes terá a sua própria documentação, que inclui o que compõe a entidade de cada uma, como configurar e quais as vantagens da configuração de cada um.

Para fechar a documentação do canal têm-se os métodos de produção de resultados, como é o caso das mensagens e relatórios. Estes métodos são essencialmente de consulta, pelo que o resultado de cada método se encontra no método em si. Por este motivo, não existe grande necessidade de grandes explicações a esta funcionalidade.

6.6.3 Sms

Semelhante ao canal de email, uma mensagem de Sms é simples de utilizar, mas requer algum domínio para produzir os resultados esperados. Para tal, a documentação das propriedades deste canal é importante para que o utilizador compreenda quais as capacidades e limitações da Slingshot em relação a este canal.

Os métodos mais utilizados desta componente serão o relacionados com o envio de mensagens Sms, pelo que deverá ser a primeira secção. À semelhança do email, é dada uma pequena apresentação das componentes de uma mensagem Sms, juntamente com os estados e respetiva explicação que pode passar.

Ao contrário do canal de email, mensagens Sms possuem mais restrições ao seu envio. Existem várias razões por trás delas, entre as quais limitações do meio (limite de carateres) a controlo de conteúdo da parte das operadoras, e até bloqueio de mensagens por motivos legais. Como tal, é importante deixar claro quais serão as limitações existentes, tanto para prevenir o utilizador como proteger a E-goi de clientes insatisfeitos.

Para terminar a secção de envio de mensagens Sms, tal como no email, é apresentado o conceito de mensagens Registered e códigos de customização. Apesar de ambas terem o mesmo funcionamento, a razão de se repetir para cada canal é de não existem garantias de que um cliente irá analisar a documentação completa e fazer uso de todos os canais e funcionalidades. Deste modo, mantem-se a divisão por canal e encapsula-se a informação próxima do método de envio do canal respetivo.

Na parte da configuração de mensagens Sms pode definir-se opções como remetentes e Templates. Estas componentes terão uma documentação bastante semelhante as do canal Email, ajustando apenas as especificações do canal. Desta forma mantem-se a coesão na documentação entre os canais.

6.6.4 Push

O canal Push, à semelhança dos restantes canais, também encapsula em si uma breve explicação das caraterísticas de uma mensagem push.

Tal como nos canais email e sms, é necessário definir o remetente da mensagem. Neste caso uma mensagem push é associada e enviada em nome de uma aplicação, que deve ser definida e aprovada antes do envio. Uma vez que este passo pode ser algo mais complexo, a secção para as Apps é acompanhada por documentação sobre como definir a aplicação.

Finalmente, existem as secções de Templates e Reports. Estas secções serão semelhantes as secções dos outros canais de envios.

6.6.5 Casos de uso

Esta secção da documentação será dividida para cada um dos três casos de uso. Uma vez que a caraterística de cada canal é apresentada nos respetivos canais, não fará sentido incluir aqui informação já apresentada. Deste modo, apenas se fará uma explicação mais detalhada sobre como cada caso de uso funciona, quais os potenciais casos de uso, e em certos casos, como configurar.

Para cada caso de uso é explicado qual o seu principal objetivo seguido de uma breve explicação de como o configurar e executar. Cada caso de uso possui o seu próprio comportamento, e faz-se recurso a diagramas para mostrar ao utilizador como os pedidos são geridos, e como este pode acompanhar o estado do caso de uso.

O Verify, como caso de uso, é um pouco diferente dos outros, na medida em que não necessita de configuração prévia para utilizar (exceto aquela necessária para enviar Sms).

6.6.6 Utilities

Nesta secção encontram-se os métodos que serão menos utilizados. Como tal, não se espera que o utilizador necessite de maior contexto para as funcionalidades. Por exemplo, o conceito do grupo já foi mencionado no envio de cada canal, pelo que a funcionalidade de ir buscar todos os grupos deve ser autoexplicativa.

A exceção a este caso é a documentação dos Webhooks. É explicado ao utilizador o que é um Webhook, e como este funciona com um exemplo. Depois, é explicado, por canal, quais os estados da mensagem a serem capturadas, qual a ordem que os eventos são esperados que aconteçam, e uma pequena descrição do que o estado representa.

6.6.7 Geração de excertos de código

De forma a gerar excertos de código, optou-se inicialmente por utilizar as propriedades das notações do Spring para produzir os excertos de código. Desta forma os exemplos serão injetados automaticamente quando o contrato é gerado. No entanto, as notações exigem que os dados das notações sejam estáticos. Isto levanta problemas na automatização da geração de código. O esforço de gerar os excertos para cada linguagem e cada método é notório. Por exemplo, se houver 30 serviços, e 5 linguagens a gerar para cada serviço, tem-se um total de 150 excertos de código a serem escritos. Por outro lado, qualquer alteração no contrato exige o ajuste dos respetivos exemplos de código. Por estes motivos, abandonou-se esta opção.

Como opção seguinte considerou-se utilizar o gerador de código automático do Postman. As vantagens da sua utilização é a integração da mesma como um passo no processo de implantação da aplicação, e a variedade de linguagens que disponibiliza.

Apesar disso, a sua integração originou alguns problemas na injeção dos exemplos no contrato da API. Isto porque a API em Spring gera o contrato quando requisitado, em vez de ter um ficheiro com o mesmo já definido.

Por este motivo, a solução implementada foi a de implementação de um novo componente, que ficaria responsável por gerar os excertos de código. Para cada método HTTP foi

desenvolvido um modelo de pedido, replicando este processo para cada linguagem a ser utilizada.

```
public static final String get(String endpoint) {
    return "var request = require('request');\n" +
            "var options = {\n" +
            "  'method': 'GET',\n" +
            "  'url': '" + Constants.BASE_URL + endpoint + "',\n" +
            "  'headers': {\n" +
            "    'ApiKey': 'YOUR-APIKEY-HERE'\n" +
            "  }\n" +
            "};\n" +
            "request(options, function (error, response) {\n" +
            "  if (error) throw new Error(error);\n" +
            "  console.log(response.body);\n" +
            "});";
}
```

Figura 75 - Modelo para geração de um pedido *Get* utilizando NodeJS (Request)

Assim, é possível reaproveitar o método da Figura 75 para representar todos os métodos Get nessa linguagem. As vantagens desta abordagem são controlo como o código é gerado e integrado na documentação. No entanto, esta abordagem requer algum estudo nas linguagens escolhidas, de forma a mantê-las atualizadas com o tempo.

Em relação as linguagens, é necessário definir quais as que serão suportadas. Deve fazer-se uma análise a quais devem ser adicionadas à documentação. Por um lado, incluir todas as linguagens iria sobrecarregar não só a página de documentação como o utilizador. Por outro lado, corre-se o risco de incluir linguagens que serão pouco utilizadas, contribuindo para a sobrecarga de informação.

Para determinar qual a linguagem que se deve utilizar foi estudado o índice de TIOBE. Este índice mede a popularidade de acordo com o número de pesquisas nos vários motores de pesquisa, como Google e Bing. Pode representar, atualmente, a popularidade de linguagens em uso. Assim sendo, as linguagens mais utilizadas de acordo com este índice são as que estão apresentadas na Tabela 26.

Tabela 26 - Popularidade de linguagens de programação (TIOBE - The Software Quality Company, 2020)

Sep 2020	Programming Language	Ratings	Change
1	C	15.95%	+0.74%
2	Java	13.48%	-3.18%
3	Python	10.47%	+0.59%
4	C++	7.11%	+1.48%
5	C#	4.58%	+1.18%
6	Visual Basic	4.12%	+0.83%
7	JavaScript	2.54%	+0.41%
8	PHP	2.49%	+0.62%
9	R	2.37%	+1.33%
10	SQL	1.76%	-0.19%

Para um termo de comparação do índice com outro com critérios diferentes, utilizou-se a aplicação online PYPL – Popularity of Programing Languages. Este índice apresenta as linguagens com tutoriais mais procuradas no motor de pesquisa Google. Este índice representa, de certa forma, a popularidade das linguagens por novos programadores, e serve como um espelho para tendências futuras. O índice é apresentado na Tabela 27.

Tabela 27 - Popularidade de linguagens de programação (Carbonnelle, 2020)

Rank	Language	Share	Trend
1	Python	31.56 %	+2.9 %
2	Java	16.4 %	-3.1 %
3	Javascript	8.38 %	+0.3 %
4	C#	6.5 %	-0.8 %
5	PHP	5.85 %	-0.5 %
6	C/C++	5.8 %	+0.0 %
7	R	4.08 %	+0.3 %
8	Objective-C	2.79 %	+0.2 %
9	Swift	2.35 %	-0.1 %
10	TypeScript	1.92 %	+0.1 %

Tanto na Tabela 26 como na Tabela 27 estão listadas algumas linguagens que não serão uteis para o contexto de uma API. Linguagens como R, SQL não são utilizadas como linguagens para o desenvolvimento de aplicações.

Tendo uma ideia da popularidade de cada linguagem, as selecionadas para gerar exemplos de código são as seguintes:

- Python
- Java
- PHP
- C
- C#
- Javascript

Tendo definidas as linguagens a serem adicionadas, procedeu-se à implementação da geração de excertos de código para cada linguagem. Depois, foi adicionada à aplicação a inicialização e configuração dos mesmos.

A injeção dos exemplos de código na definição do contrato é feita chamando o método que devolve o contrato. Com o corpo de resposta injeta-se as linguagens que devem gerar os excertos de código no serviço correto, acrescentando ainda um exemplo do corpo da mensagem. O resultado é apresentado na Figura 76.

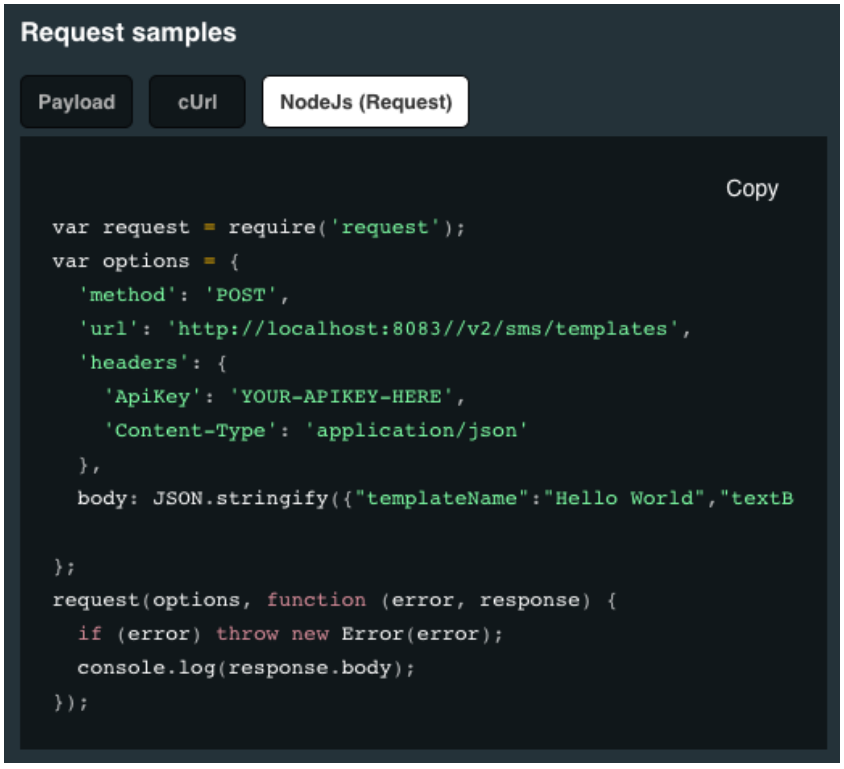

Figura 76 - Exemplo de excerto de código

O exemplo da Figura 76 mostra a estrutura de código de todos os excertos. Começa com a importação das bibliotecas necessárias, de forma a poder executar pedidos HTTP. Depois é construído o pedido e executado. Finalmente, o resultado é processado. Para facilitar a

compreensão do código, os passos apresentados acima serão o mais simples possível, apenas validando o resultado do pedido e imprimindo-o.

7 Avaliação da Solução

Neste capítulo serão apresentadas as formas como a aplicação será testada. Serão descritos os critérios mais relevantes para testar a aplicação, ao que se segue a apresentação e discussão dos resultados.

7.1 Plano de Testes

Antes de se desenharem os testes é necessário dividir as vertentes em que a solução deva ser testada. De forma a simplificar o planeamento dos testes a serem efetuados serão definidas duas vertentes de testes.

A primeira vertente de testes prende-se com o aspeto técnico da solução, e se os serviços que a compõem funcionam como esperado. Será testada a qualidade de implementação através de testes ao código e estrutura informática. Desta forma espera-se mostrar que a aplicação é funcional, cumpre com o contrato desenhado, e apresenta resultados de processamento que rivalizam com a versão anterior da API.

A segunda vertente de testes prende-se com a usabilidade, e o impacto na satisfação de quem a utiliza. Será avaliada a experiência do utilizador, da perspetiva de quem utiliza pela primeira vez e da de quem tem experiência. Serão comparados os resultados entre a API atual e a nova versão, de forma a avaliar se a solução desenvolvida possui maior grau de aceitação.

7.1.1 Hipóteses a avaliar

De forma a estabelecer objetivos de teste, é necessário estabelecer objetivos que se esperam mostrar, a fim de garantir a validação rigorosa da solução.

As hipóteses a testar serão as seguintes:

1. Todos os serviços presentes na solução são funcionais, quer do ponto de vista de utilização, quer do ponto de vista de gestão de pedidos;
2. A diferença de desempenho entre a solução e a aplicação atual (tempo de resposta, lançamento de exceções) não é maior que 20%;
3. A solução apresenta 20% maior grau de satisfação geral por parte dos seus utilizadores em relação a versão atual.

As hipóteses 1 e 2 estão associadas à primeira vertente técnica. Com estas hipóteses valida-se que a API corresponde ao exigido do ponto de vista dos requisitos funcionais e não-funcionais de uso da mesma. A hipótese 3 esta associada a segunda vertente de utilização, validando o aumento da satisfação da utilização face a versão atual da Slingshot.

7.1.2 Metodologia da avaliação

Estabelecidas as hipóteses a serem testadas, definem-se os testes que testem as hipóteses, de forma a averiguar se são validas.

Os testes a primeira hipótese serão feitos através da análise do comportamento da API em relação a vários tipos de pedidos. Serão definidos quais os comportamentos alternativos, e verificar que a API responde de acordo com o que será esperado.

Os testes à segunda hipótese serão efetuados através de chamadas a API. Serão feitos pedidos a vários serviços em várias iterações, medindo os tempos de resposta de cada pedido. Com os tempos de resposta a cada pedido será feita uma análise comparativa dos termos, de forma a medir a diferença no tempo de resposta entre versões.

Os testes a terceira hipótese serão efetuados através de inquéritos. A solução será apresentada aos colaboradores E-goi, que preencherão um inquérito comparando vários aspetos da utilização entre as versões da API. As respostas serão analisadas, comparando a satisfação que cada uma das versões provoca naquele que será o grupo que utilizará a API.

7.2 Resultados dos testes

Uma das formas de validar não só o contrato como o cumprimento do mesmo passa por testar a aplicação para os mais variados tipos de pedidos que poderá estar sujeito.

7.2.1 Testes de Aceitação

Uma API está sujeita a vários tipos de pedidos, que devolvem resultados diferentes conforme os parâmetros enviados. Por exemplo, o processamento e resultado de um pedido não autenticado é diferente de um que falta um parâmetro essencial. Esta discrepância entre os casos de uso alternativos obriga a definir uma base de testes para verificar o comportamento da API face diferentes comportamentos. Desta forma, foram definidas as seguintes questões como base de teste (De, 2017):

1. Qual é o comportamento por omissão da API quando nenhum argumento de *query* é passado?

2. Qual é o comportamento da API quando os parâmetros de *query* são passados com os valores corretos?
3. Qual é comportamento da API quando um argumento é passado sem nenhum valor?
4. Qual é comportamento da API quando um argumento é passado com um valor incorreto?
5. Qual é o formato de dados por omissão quando nenhum formato é especificado no pedido?
6. Quais são os códigos HTTP para diferentes casos de sucesso ou insucesso?
7. Qual o comportamento quando um utilizador sem permissões tenta efetuar o pedido?

De forma a testar dos seguintes, foi desenhado um script para fazer pedidos a todos os serviços a API. Este script foi desenvolvido de forma modular, de forma a testar os vários comportamentos, como a ausência de Apikey ou sem nenhum. Os resultados produzidos foram aqueles apresentados na Tabela 28.

Tabela 28 - Resultado dos testes de aceitação

Questão	Resultado	Exemplo
1	A nova versão preenche sempre que possível o parâmetro com o valor mais provável e menos intrusivo no resultado para o cliente. Caso seja necessário e não tenha valor previsto, retorna uma mensagem de erro.	No caso de o utilizador não especificar a prioridade num pedido de envio de email, este parâmetro é assumido como não-prioritário
2	A nova versão valida os dados inseridos, altera-os para um formato que a Slingshot reconhece (se for o caso) e constrói um pedido válido.	No envio de uma mensagem de Sms, a nova versão altera o parâmetro "phone" para o parâmetro "mobile", que é aceite pela Slingshot.
3	A nova versão assume o valor como vazio, e no caso de ser um parâmetro necessário ao pedido, devolve uma mensagem de erro.	Na criação de uma App, o nome da app em branco é capturado, e retorna uma mensagem de erro a informar que esse parâmetro (e outros na mesma situação) encontra-se vazio.
4	No caso de o valor ser inteligível pela aplicação, o valor é processado. Caso contrário, retorna uma mensagem de erro.	Se o número de tentativas máximo de um Alert for menor que 0, a nova versão retorna um erro.
5	Quando nenhum formato de dados é especificado, a API notifica o utilizador.	Ao fazer um pedido sem o cabeçalho *Content-Type*, a API devolve um erro do tipo 415 – Formato não suportado.
6	Os códigos HTTP vão de encontro ao que é definido no protocolo HTTP.	Ao ser enviado um pedido sem apikey, é retornado um erro HTTP do tipo 401-Não Autorizado.
7	A API faz o pedido à Slingshot, capturando o erro do tipo não autorizado.	Ao apagar uma mensagem planeada, a Slingshot levanta um erro do tipo 403 - Proibido, que é processada pela nova versão.

Dado o comportamento da nova versão face as questões levantadas, pode considerar-se que foi validada a hipótese 2.

7.2.2 Testes de Resposta

De forma a validar a hipótese 2 da secção 7.1.1, é necessário validar, para todos os serviços disponibilizados os tipos de respostas. Para correr este plano de testes foi desenvolvido um pequeno *script* que faz pedidos para cada um dos serviços, marcando a hora de início e fim do pedido, e calculando a diferença.

Foi desenvolvido um script para cada a aplicação, e outro para a Slingshot. Cada script executa o mesmo pedido com um intervalo de 5 segundos entre cada pedido, num total de 30 pedidos

por cada serviço. Desta forma simula-se um número modesto de pedidos, gerando para cada pedido um conjunto de resultados que representará o tempo médio de resposta de cada API.

Para cada API foi desenvolvido um plano de testes semelhantes entre eles, de forma a produzirem resultados semelhantes que possam ser comparáveis. Os resultados da API, neste contexto, não foram considerados, exceto quando o resultado de uma API não correspondesse ao resultado do mesmo tipo de pedido à outra API.

Os resultados de cada script foram armazenados em ficheiros CSV separados, de forma a facilitar a computação do mesmo. Cada linha representa um pedido e o respetivo resultado, incluindo o tempo de resposta.

A partir dos dados em cada ficheiro CSV mencionado no parágrafo anterior extraiu-se os tempos de resposta para cada API. Para comparar os tempos de respostas é feito recurso ao teste T, que compara as distribuições médias dos resultados. No entanto, é necessário determinar se as amostras possuem variâncias semelhantes. Para tal, é feito recurso ao teste F, que analisa as variações dos resultados. Para ambos os testes foi considerado um nível de significância de 0.05. O processamento dos dados até ao final desta secção feito com recurso à linguagem R, que permite fazer processamento e análises de dados.

Para validar que as amostras são homogéneas, corre-se o teste de F. Para melhor analisar o resultado do teste determinam-se as hipóteses possíveis. Estas são as seguintes:

```
h0: as variâncias são homogéneas
h1: as variâncias não são homogéneas
```

Tendo as hipóteses definidas, executou-se o teste de F, cujos resultados foram os seguintes:

```
F test to compare two variances

data:  v1template and v2template
F = 0.15407, num df = 659, denom df = 685, p-value < 2.2e-16
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.1324386 0.1792644
sample estimates:
ratio of variances
         0.1540656
```

Uma vez que o valor de p é menor que 0.05, rejeita-se a hipótese nula. Desta forma, mostra-se que as amostras não são independentes. Assim pode seguir-se para o teste T. Uma vez que as amostras são independentes, é feito o teste T de Welch, uma alternativa ao teste T para amostras independentes.

Tal como acima, determina-se as hipóteses que serão para ser analisadas.

```
h0: a média das amostras são semelhantes
h1: a média das amostras não são semelhantes
```

Novamente, tendo as hipóteses definidas, executou-se o teste T de Welch, cujos resultados foram os seguintes:

```
        Welch Two Sample t-test

data:  v1template and v2template
t = -4.1039, df = 898.01, p-value = 4.433e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -188.45720  -66.51902
sample estimates:
mean of x mean of y
 143.3136  270.8017
```

Uma vez que o valor de p é novamente menor que 0.05, rejeita-se a hipótese nula. Conclui-se assim que existe uma diferença entre as médias dos tempos de resposta entre as API's.

Estabelecida a diferença entre médias e a margem de confiança de variação, pode calcular-se a percentagem de variação entre as médias. Uma vez que ambos os valores de variação são negativos, conclui-se que a média dos tempos de resposta é maior na nova versão quando comparada a Slingshot. Fazendo uma regra de 3 simples, calcula-se as médias de diferenças entre a média dos resultados para o tempo de resposta da Slingshot e a cada variância dos tempos de resposta da nova versão. O resultado é um acréscimo de tempo de resposta entre 35,27% e 131,5%.

Dados os resultados deste teste, considera-se que a nova versão não passou na hipótese 1 do plano de testes. Esta conclusão surge da necessidade da nova versão de processar e construir os pedidos da API. Por outro lado, todos os pedidos feitos à nova versão serão processados e remetidos para a Slingshot, independentemente do historial de pedidos. Neste caso, poderá considerar-se estabelecer um mecanismo de cache, que armazena a resposta da Slingshot por um período limitado de tempo, respondendo ao pedido

No entanto, note-se que a média no tempo de resposta da Slingshot é cerca de 143ms e da nova versão 271ms. Estes valores foram medidos em milésimas de segundo, pelo que a diferença entre tempos de resposta para um utilizador poderá ser insignificante.

7.2.3 Testes de Funcionalidade

Para testar a estrutura do contrato, juntamente com a documentação da API, foi desenhado um inquérito, que será respondido por desenvolvedores dentro da E-goi. A lógica de definir este grupo prende-se com a orientação mais técnica da API. Apesar da mesma poder ser utilizada por qualquer tipo de utilizador, a sua integração eficaz é facilitada por alguém com conhecimento. Desta forma, une-se a nova versão ao conceito de B2D, que será o público alvo da Slingshot.

As questões a serem colocadas, juntamente com o aspeto que pretende avaliar são as seguintes:

1. Qual o grau de satisfação com a velocidade de resposta da API? (Componente técnica)

2. Quão fácil é utilizar a API? (Facilidade de aprendizagem)
3. Quão fácil foi a aprendizagem inicial de como utilizar a API? (Facilidade de aprendizagem)
4. Qual a satisfação com a documentação da API? (Qualidade da documentação)
5. Qual a satisfação com a estrutura da documentação da API? (Qualidade da documentação)
6. Quão úteis são as mensagens de erro? (Qualidade da documentação)
7. Quão úteis são as os exemplos de parâmetros? (Qualidade da documentação)
8. Qual a satisfação com os excertos de código? (Qualidade da documentação)
9. Qual a satisfação com a quantidade de funcionalidades disponibilizadas? (Disponibilização de funcionalidades)
10. Qual a opinião sobre a utilidade das funcionalidades disponibilizadas? (Disponibilização de funcionalidades)

Tendo definido as áreas a avaliar, procedeu-se ao preenchimento do mesmo pelo público alvo definido acima. É apresentado no Anexo A o inquérito colocado. Para cada pergunta é respondida duas vezes, uma para cada versão a ser comparada. As respostas são valores de 1 a 5, onde 1 representa o menos grau de satisfação e 5 representa o maior grau de satisfação. Deste processo resultaram 20 respostas, pelo que se pode assumir que existe uma boa base para analisar o desempenho da nova versão neste objetivo.

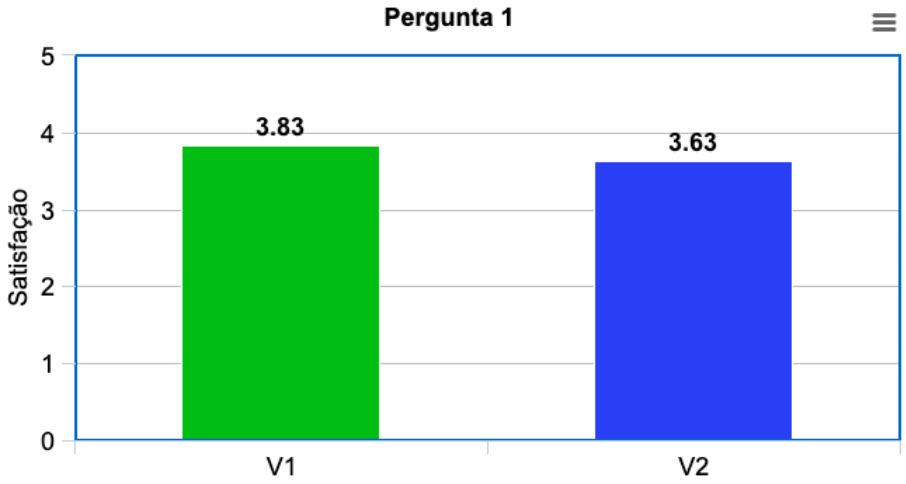

Figura 77 - Média de respostas à Componente Técnica da API

Tal como apresentado na Figura 77, verifica-se que as diferenças entre tempos de respostas das APIs se revelam praticamente irrelevantes para os inquiridos. Apesar da nova versão ser um pouco mais lenta, uma vez que faz uso à Slingshot, verifica-se que a diferença para o utilizador é quase impercetível. De facto, como será visível mais em baixo, o tempo de resposta dos pedidos são medidos em milésimas de segundo, que será muito difícil para um ser humano perceber.

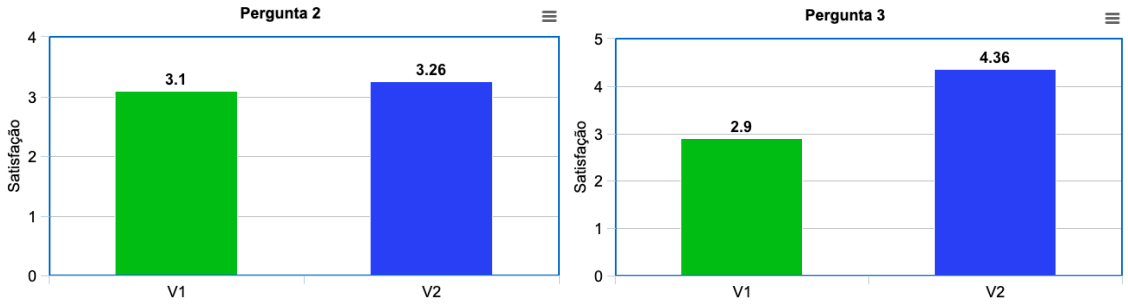

Figura 78 - Média de respostas à componente de Facilidade de Aprendizagem da API

No ponto de vista de um utilizador novo da API, a nova versão revela ser a que maior satisfação produz nos utilizadores, como demonstra a Figura 78. No entanto, a necessidade de configuração da API, em conjunto com o esforço de integração para o utilizador continuam a ser um aspeto a melhorar na nova versão. Pode especular-se que o desenvolvimento de SDK's poderia produzir uma maior diferença no grau de satisfação do utilizador.

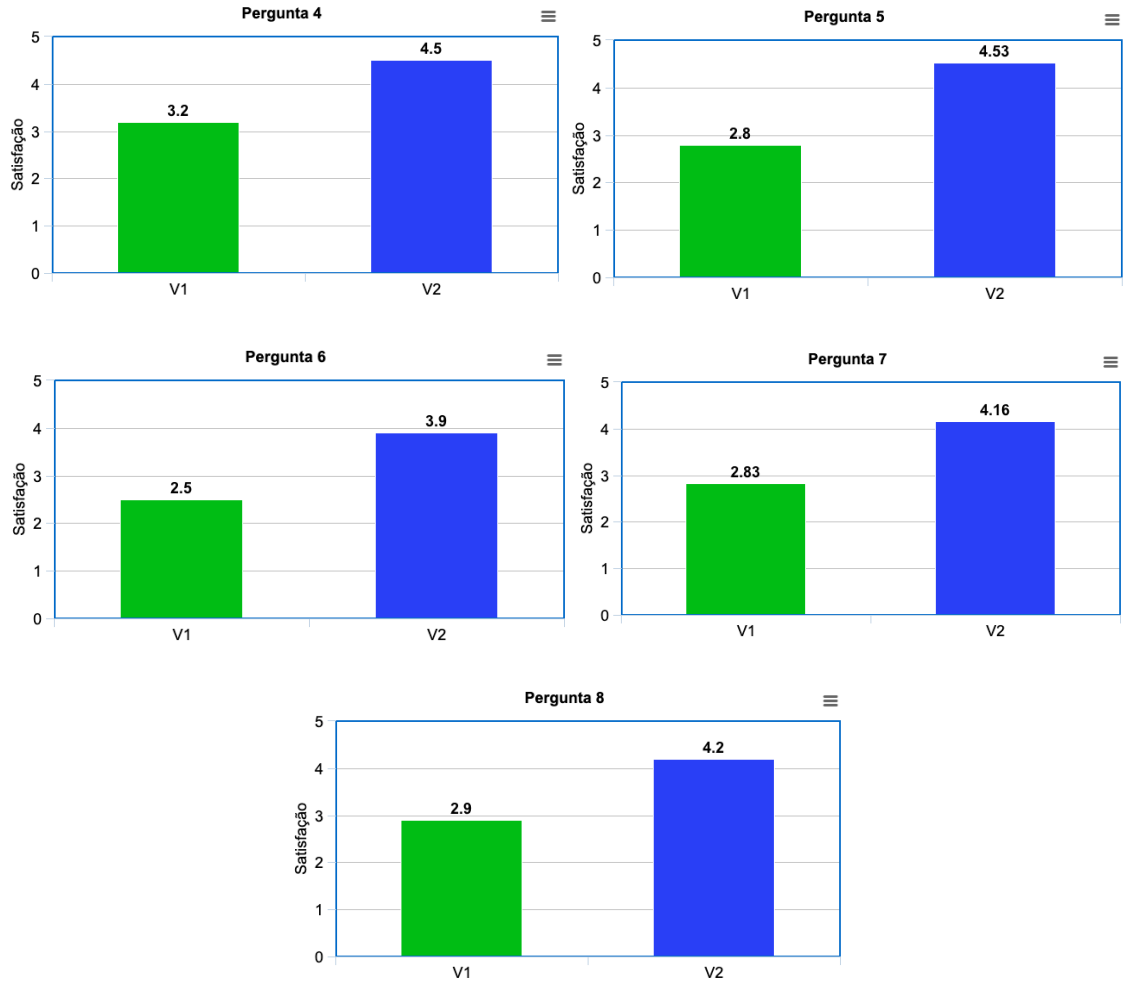

Figura 79 - Média de respostas à Qualidade da documentação da API

Em relação à documentação da API e respetivos métodos, considera-se que existe um grau de satisfação para o utilizador maior para a nova versão do que para a Slingshot. De facto, este é dos pontos que apesar de existir na Slingshot, não apresenta consistência entre serviços, ou maior especificação para além do essencial. Ao resolver estes problemas, a documentação fica mais rica, que se traduz nos resultados da Figura 79.

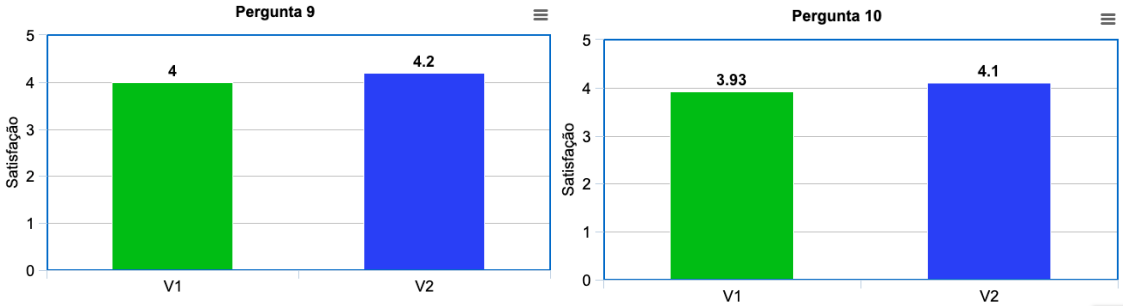

Figura 80 - Média de respostas aos Disponibilização de funcionalidades

Em relação as questões acerca da funcionalidade, tal como apresentado na Figura 80, pode concluir-se que não existe diferença significativa entre as funcionalidades das duas API's. As funcionalidades principais foram mantidas, acrescido ao facto da nova versão consumir a Slingshot.

Finalmente, avaliam-se os resultados obtidos, de forma a validar a hipótese 3. Visto que os inquéritos abrangem a avaliação de vários critérios, não se perspetiva que haja relação entre a resposta de cada inquérito. Desta forma, fez-se uma média de todas as respostas de cada API, que produziu os resultados apresentados na Tabela 29.

Tabela 29 - Resultados dos inquéritos

	V1	**V2**
Média	3,2	4,213
% (relativa a V1)	100	131,656

Os resultados apresentados na Tabela 29 sugerem que a nova versão da API gera maior satisfação. De facto, a média geral sobe um valor inteiro quando comparada com a versão anterior. Através de uma regra de três simples, conclui-se que a nova versão gera um grau de satisfação 30% superior à anterior. Por este motivo, valida-se a hipótese 3.

8 Conclusões

Neste capítulo serão apresentadas as conclusões acerca do projeto. Será tido em conta o atingimento dos objetivos, o estado do projeto, limitações e futuro do projeto.

8.1 Objetivos atingidos

A ideia inicial deste projeto partiu inicialmente eventualidade de se restruturar o contrato da Slingshot. dados os problemas identificados na seção 4.1.2. Dada a história de desenvolvimento da Slingshot, as limitações serão expectáveis. No entanto, a sua correção após adoção por parte dos clientes pode levar a quebrar as compatibilidades existentes. Por outro lado, as necessidades do negócio levaram ao desenvolvimento de novas funcionalidades. Por estes motivos, a E-goi não sentia necessidade imediata, pelo que abria o caminho para o desenvolvimento do projeto descrito neste documento.

Começou-se por se fazer um estudo da Slingshot no seu estado atual. Depois, fez-se um levantamento do mercado e análise aos concorrentes da Slingshot. Foram analisadas a documentação das API's, comparando com o que a Slingshot já possui. Destas análises foi feito o levantamento das limitações da mesma, quer do contrato, quer da documentação, que compuseram os requisitos do projeto.

Tendo definido quais os requisitos, definiu-se como a nova versão se iria encaixar na estrutura atual da Slingshot na E-goi. Foi feito um levantamento junto dos responsáveis, e debatidas várias soluções. Para auxiliar a melhor alternativa, foi feita uma análise de valor. Foi feita uma breve análise de negócio, de forma a contextualizar as alternativas apresentadas na análise. Estas alternativas passaram por um processo de decisão através do modelo AHP.

Concluiu-se que, dadas as necessidades do projeto de estruturar o contrato e documentação, uma nova versão da Slingshot seria desenvolvida, consumindo a API atual. Esta versão iria funcionar como prova de conceito, podendo ser ajustada e implantada caso o produto final o justificasse.

Tendo uma base de contexto do projeto partiu-se para o desenho do contrato da nova versão. Foram analisadas as melhores práticas na área, através do estudo de documentação, artigos e livros, de forma a alinhar o contrato com as normas definidas e seguidas pela maioria dos programadores. Desta análise produziu-se o contrato da nova versão da API. Também se fez um estudo dos objetos e entidades da API, de forma a produzir documentação relevante.

Seguiu-se a etapa de implementação, onde o contrato foi desenvolvido. O recurso à Framework Spring permitiu o desenvolvimento fluido dos serviços a serem implementados. Por outro lado, a modularização das componentes da aplicação permitiu a criação de módulos, que aplica o padrão de alta coesão e baixo acoplamento. Desta forma reduz-se o número de fluxos alternativos, promove-se a reutilização de código e diminui a carga de processamento.

Após a implementação da API esta foi testada, de forma a validar os objetivos a que se propõe. Para tal foi desenvolvido um script, que faria pedidos a vários serviços, num total de 30 ciclos. Este script gerou dados que puderam avaliar não só a qualidade dos serviços implementados, como o tempo de resposta. Apesar do tempo de resposta não se encaixar nos objetivos inicialmente definidos, os serviços produziram resultados que iam de acordo com o desenho do contrato. Também foram feitos inquéritos a outros programadores, que compararam ambas as versões da Slingshot. Os resultados permitiram validar que a nova versão produz maior grau de satisfação de uso quando comparado com a anterior. Neste sentido, conclui-se que o projeto atingiu o objetivo de aceitação por parte dos utilizadores da API.

Neste momento este projeto considera-se terminado com sucesso. A aplicação desenvolvida satisfez a maior parte dos objetivos, apenas falhando no tempo de resposta. No entanto, o projeto levantou o interesse da E-goi, que o acolheu para desenvolvimento futuro. Por este motivo considera-se que o projeto teve sucesso para todas as partes envolvidas.

8.2 Limitações

Apesar de se poder considerar que o projeto teve sucesso, algumas lacunas ficaram por preencher. Estas lacunas provieram tanto de incompatibilidades técnicas. Estas falhas poderão comprometer a implementação direta da API, pelo que a sua análise deve ser feita com atenção.

O desenvolvimento de SDK's não foi possível, uma vez que não se justificou durante o desenvolvimento. No entanto, a E-goi proporciona integrações através de SDK's para outros produtos, pelo que a ausência desta integração na nova versão da API quebra a homogeneidade de oferta da API.

Outro fator de desconforto foi a manutenção da Slingshot. Esta recebe melhoramentos constante, que por vezes quebrava o funcionamento normal da nova versão. Na etapa de desenvolvimento foi adicionado o canal Push, pelo que o projeto foi adaptado de forma a contemplar este novo canal. A sua incorporação, apesar de simples, demorou algum tempo a ser integrada, que limitou o desenvolvimento de outras funcionalidades.

Também o tempo de resposta deixou a desejar quando testado. A nova versão possui a sua própria validação de dados, que reduz o número de pedidos a Slingshot, e reduz o tempo de resposta. No entanto, a nova versão acrescenta uma camada de complexidade entre o utilizador e a Slingshot, que pode comprometer o tempo de resposta.

Finalmente, o contexto sociólogas em que o projeto foi desenvolvido, fruto da disseminação e consequente tentativas de contenção do vírus COVID-19, condicionou o desenvolvimento do projeto. O projeto, apesar de ser principalmente desenvolvido individualmente e com ferramentas que permitem o desenvolvimento remoto, sofreu uma quebra temporária no seu desenvolvimento. Apesar disso, esta dificuldade foi ultrapassada com o tempo e impacto tecnológico mínimo.

8.3 Continuação do projeto

A nova versão da Slingshot foi considerada um projeto interessante, e aceite pela E-goi para disponibilizar ao público.

Este projeto ficou a cargo da equipa da Slingshot, que adaptou a nova versão para o público. Alguns ajustes foram feitos, de forma a integrar a mesma no fluxo de desenvolvimento da Slingshot. O número de serviços disponibilizados foi reduzido, limitando-se apenas aqueles que seriam essenciais ou produzissem valor. Os que foram disponibilizados foram revistos e ajustados, de forma a garantir que não existe quebras de comunicação entre a nova versão e a Slingshot. Também o agrupamento de conceitos e serviços foi alterado.

Neste momento, a nova versão encontra-se disponibilizada para o público e em uso, quer pelos clientes, quer pela própria E-goi, que desenvolve novas aplicações com base na nova versão. Neste momento, após cerca de um mês de disponibilização, a nova versão recebeu cerca de 17000 pedidos com tendência para escalar, dados os fatores mencionados. Também foi desenvolvida uma alternativa de geração de excertos de código, de forma a homogeneizar o processo de implantação com as praticadas pela E-goi. Por fim, o desenvolvimento de SDK's também será um ponto de investimento, completando assim o desenvolvimento do desenho inicial do projeto.

Referências

American Marketing Association. (2017). *What is Marketing? — The Definition of Marketing*. Retrieved from AMA: https://www.ama.org/the-definition-of-marketing-what-is-marketing/

Atlassian. (2019, Abril 23). *Webhooks*. Retrieved from Jira Server Developer: https://developer.atlassian.com/server/jira/platform/webhooks/

Aué, J., Aniche, M., Lobbezoo, M., & van Deursen, A. (2018). An Exploratory Study on Faults in Web API Integration in a Large-Scale Payment Company. *ICSE-SEIP '18: 40th International Conference on Software Engineering: Software Engineering in Practice Track* (pp. 13-22). Gothenburg, Suécia: Institute of Electrical and Electronics Engineers.

Bonardi, M., Brioschi, M., Fuggetta, A., Verga, E. S., & Zuccalà, M. (2016). Fostering Collaboration Through API Economy: The E015 Digital Ecosystem. *3rd International Workshop on Software Engineering Research and Industrial Practice* (pp. 32-38). Nova Iorque, Estados Unidos da América: Association for Computing Machinery.

Brito, A., Hora, A., & Valente, M. T. (2019, Agosto). You Broke My Code: Understanding the Motivations for Breaking Changes in APIs. *Empirical Software Engeneering*.

Brito, G., Hora, A., Valente, M. T., & Robbes, R. (2016). Do Developers Deprecate APIs with ReplacementMessages? A Large-Scale Analysis on Java Systems. *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* (pp. 360-369). Osaka, Japão: Curran Associates, Inc.

Buse, R. P., & Weimer, W. (2014). Synthesizing API Usage Examples. *Proceedings - International Conference on Software Engineering* (pp. 782–792). IEEE Press.

Bush, T. (2019, Maio 16). *5 Examples of Excellent API Documentation (and Why We Think So)*. Retrieved from Nordic APIs: https://nordicapis.com/5-examples-of-excellent-api-documentation/

Carbonnelle, P. (2020). *PYPL PopularitY of Programming Language index*. (Github) Retrieved Julho 20, 2020, from http://pypl.github.io/PYPL.html

Celar, S., Mudnic, E., & Seremet, Z. (2016). State-Of-The-Art of Messaging for Distributed Computing Systems. *Proceedings of the 27th International DAAAM Symposium 2016* (pp. 298-307). Vienna, Austria: DAAAM International.

Chang, P. (2018, Novembro 28). *Session VS Cookie VS Token. Difference between Cookie and Session | by Peter Chang | Medium*. (Medium) Retrieved Agosto 29, 2020, from https://medium.com/@peterchang_82818/difference-session-cookie-token-vs-token-authentication-based-traditional-store-a177e8474ee3

Cheng, W., Schubert, J., & Pietrzyk, A. (2020, Setembro 25). *GitHub - OpenAPITools/openapi-generator: OpenAPI Generator allows generation of API client libraries (SDK generation), server stubs, documentation and configuration automatically given an*

OpenAPI Spec (v2, v3). (Github) Retrieved Setembro 28, 2020, from https://github.com/OpenAPITools/openapi-generator

Claessens, M. (2018, Abril 15). *Relationship Marketing Strategies – Understanding Relationship Marketing*. Retrieved from Marketing-Insider - Principles of Marketing explained: https://marketing-insider.eu/relationship-marketing-strategies/

De, B. (2017). *API Management.* Apress.

Dodson, B., Sengupta, D., Boneh, D., & Lam, M. S. (2010). Secure, Consumer-Friendly Web Authentication and Payments with a Phone. *International Conference on Mobile Computing, Applications, and Services* (pp. 17-38). California, Estados Unidos da América: Springer.

Doerrfeld, B., Wood, C., Anthony, A., Sandoval, K., & Lauret, A. (2016). *The API Economy - Disruption and the Business of API.* Vargarda, Suécia: Nordic APIs.

E-goi. (2016). *Caso de Sucesso E-goi - Rio 2016*. Retrieved from www.e-goi.com/wp-content/uploads/Case_Study_Rio2016_PT.pdf

E-goi. (2020). *E-goi - TRANSACTIONAL*. Retrieved from E-goi - API: developers.e-goi.com/transactional

Facebook. (2018, Junho). *GraphQL*. Retrieved from http://spec.graphql.org/June2018

Fagerholm, F., & Münch, J. (2012). Developer experience: Concept and definition. *Proceedings of the International Conference on Software and System Process* (pp. 73-77). Zurique, Suiça: IEEE.

Foote, B., & Yoder, J. (1997, Setembro 10). *The Selfish Class*. Retrieved from http://www.laputan.org/selfish/selfish.html

Fowler, M. (2010, March 18). *Martin Fowler*. Retrieved from Richardson Maturity Model: https://martinfowler.com/articles/richardsonMaturityModel.html

Glickenhouse, A. (2017, Outubro 20). *API Economy - 4 Business Drivers and 7 Use Case Categories - Series Overview*. Retrieved from IBM Developer: https://developer.ibm.com/apiconnect/2017/07/20/api-economy-4-business-drivers-7-use-case-categories-series-overview/

Glickenhouse, A. (2017, Agosto 10). *API Economy Business Drivers: #2 - Reach*. Retrieved from API Connect: https://developer.ibm.com/apiconnect/2017/08/03/api-economy-business-drivers-2-reach/

Glickenhouse, A. (2018, Janeiro 4). *What is an API? and What is the API Economy?* Retrieved from API Connect: https://developer.ibm.com/apiconnect/2018/01/04/api-api-economy/

Hernández, A. L. (2018). *The Future of E-commerce: The Road to 2026.* OVUM.

Infobip. (2020). *Getting Started: HTTP API Introduction*. Retrieved from Infobip: https://dev.infobip.com/getting-started

ITU. (2019). *Measuring digital development: Facts and figures.* Genebra, Suiça: ITU Publications.

Le, A. N.-H. (2014). Corporate rebranding and brand preference: brand name attitude and product expertise as moderators. *Asia Pacific Journal of Marketing and Logistics*, 26.

Lee, B. D. (2018, Dezembro). Ten Simple Rules For Documenting Scientific Software. *PLoS Computational Biology*, pp. 1-6.

Mahato, D., Dudhal, D., Revagade, D., & Bhargava, Y. (2019). A Method to Detect Inconsistent Annotations in a Medical Document using UMLS. *FIRE 2019 Forum for Information Retrieval Evaluation* (pp. 47-51). Kolkata, India: Association for Computing Machinery.

Mailjet. (2018). *Mailjet for Developers*. Retrieved from Mailjet: https://dev.mailjet.com/email/guides/getting-started/

Massé, M. (2012). *REST API: Design Rulebook.* O' Reilly.

McDonnel, T., Ray, B., & Kim, M. (2013). An Empirical Study of API Stability and Adoption in the Android Ecosystem. *Proceedings of the 2013 IEEE International Conference on Software Maintenance* (pp. 70-79). Washington, DC, Estados Unidos da América: IEEE Computer Society.

Meng, M., Steinhardt, S., & Schubert, A. (2018). Application Programming Interface Documentation: What Do Software Developers Want? *Journal of Technical Writing and Communication*, 295-330.

Meng, M., Steinhardt, S., & Schubert, A. (2019, Agosto). How Developers Use API Documentation: An Observation Study. *Communication Design Quarterly*, pp. 40-49.

Microsoft. (2019, Janeiro 30). *Arquiteturas comuns de aplicativo Web*. Retrieved from Microsoft Docs: https://docs.microsoft.com/pt-br/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures

Minds, J. (2017, Dezembro 15). *DEVELOPERS WON'T FIX YOUR PROBLEM — AN ATTEMPT TO MAKE SENSE OF DIGITAL INNOVATION*. Retrieved from Medium: https://medium.com/@JungleMinds/developers-wont-fix-your-problem-an-attempt-to-make-sense-of-digital-innovation-8cd27cbf4900

Moura, G. A. (2018, Outubro 2). *Value Proposition Canvas: Conheça o Canvas de Proposta de Valor*. Retrieved from UCJ - UFMG Consultoria Jr: https://ucj.com.br/value-proposition-canvas-proposta-de-valor/

Neumann, A., Laranjeiro, N., & Bernardino, J. (2018, Junho). An Analysis of Public REST Web Service APIs. *IEEE Transactions on Services Computing*, p. 99.

Nexmo. (2020). *Documentation*. Retrieved from Nexmo Developer: https://developer.nexmo.com/documentation

Oluwatosin, H. S. (2014, Fevereiro). Client-Server Model. *IOSR Journal of Computer Engineering*, pp. 67-71.

Osterwalder, A., Pigneur, Y., Bernarda, G., & Smith, A. (2014). *Value Proposition Design.* Wiley.

P., R. (2018, Fevereiro 23). *Lifecycle of a Request-Response Process for a Spring REST API*. Retrieved from DZone Integration: https://dzone.com/articles/lifecycle-of-a-request-response-process-for-a-spri

Posa, R. (2015, Maio 27). *Node JS Architecture - Single Threaded Event Loop*. Retrieved from JournalDev: https://www.journaldev.com/7462/node-js-architecture-single-threaded-event-loop

Quartz+Co. (2007). *Business Development Management: Best Practices in Managing and Executing Business Development.* Quartz+Co.

Raemaekers, S., van Deursen, A., & Visser, J. (2014). Semantic Versioning versus Breaking Changes: A Study of the Maven Repository. *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*, (pp. 215-224). Victoria, BC.

Rao, J. M., & Reiley, D. H. (2012). The Economics of Spam. *Journal of Economic Perspective, 26*, 87-110.

Rao, P. M., & Klein, J. A. (1994, February). Growing importance of marketing strategies for the software industry. *Industrial Marketing Management*, pp. 29-37 .

REST API Versioning Guide. (2018). Retrieved from REST API Tutorial: https://restfulapi.net/versioning/

Richardson, L., & Amundsen, M. (2013). *RESTful Web APIs.* Sebastopol, Califórnia: O'Reilly.

Robillard, M. P. (2009). What Makes APIs Hard to Learn? Answers from Developers. *IEEE Software*, 27-34.

Robillard, M. P., & DeLine, R. (2011). A field study of API learning obstacles. *Empirical Software Engineering, 16*, 703-732.

Saaty, T. L. (2008). Decision making with the analytic hierarchy process. *International Journal of Services Sciences, 1*, 83-98.

Sandoval, K. (2018, Fevereiro 6). *3 Common Methods of API Authentication Explained*. Retrieved from Nordic APIs: https://nordicapis.com/3-common-methods-api-authentication-explained/

Sawant, A. A., Robbes, R., & Bacchelli, A. (2019, Dezembro). To react, or not to react: Patterns of reaction to API deprecation. *Empirical Software Engineering*, pp. 3824-33870.

SendGrid. (2020). *Getting Started with the SendGrid API*. Retrieved from SendGrid Documentation: https://sendgrid.com/docs/for-developers/sending-email/api-getting-started/

Sign-Up.to. (2015). *Email Marketing Benchmark Report.* Woking, UK: Sign-Up.to.

SmartBear Software. (n.d.). *About Swagger Specification | Documentation | Swagger*. (SmartBear Software) Retrieved Setembro 15, 2020, from https://swagger.io/docs/specification/about/

SparkPost. (2018). *Transacional Email Benchmark Report.* San Francisco, California.

Tandon, A., Tripathi, V., & Gupta, A. (2014). The transformation of value and evolution of customer experience: an exploration of the typologies, facets and significance. *International Journal of Indian Culture and Business Management*, 425-441.

Tauberer, J. (2014, Fevereiro 10). *What makes a good API?* Retrieved from Joshua Tauberer's Archived Blog: https://joshdata.wordpress.com/2014/02/10/what-makes-a-good-api/

TECNICON. (2019, Outubro 18). *Como Utilizar a Cadeia de Valor na Gestão de Processos*. Retrieved from TECNICON: https://www.tecnicon.com.br/blog/432-Como_Utilizar_a_Cadeia_de_Valor_na_Gestao_de_Processos_

The Rocket Science Group. (2014). *API*. Retrieved from Mandrill: https://mandrillapp.com/api/docs/

The Rocket Science Group. (2016). *Documentation*. Retrieved from Mandrill: https://mandrillapp.com/docs/

Thijssen, J. (2016, Janeiro 16). *What are idempotent and/or safe methods?* Retrieved from The RESTful cookbook: http://restcookbook.com/HTTP%20Methods/idempotency/

TIOBE - The Software Quality Company. (2020, Julho). *index | TIOBE - The Software Quality Company*. (TIOBE - The Software Quality Company) Retrieved Julho 20, 2020, from https://www.tiobe.com/tiobe-index/

Todor, R. D. (2016). Blending traditional and digital marketing. *Bulletin of the Transilvania University of Braşov*, pp. 51-57.

Tricentis. (2017). *Software Fail Watch 5th Edition.*

Turkalj, D., Biloš, A., & Kelić, I. (2016). OPEN-RATE CONTROLLED EXPERIMENT IN E-MAIL MARKETING CAMPAIGNS. *Market-Tržište, 28*, 93-109.

Van Alstyne, M. W., Parker, G. G., & Choudary, S. P. (2016). Pipelines, Platforms, and the New Rules of Strategy. *Harvard Business Review*.

Webber, J., Parastatidis, S., & RObinson, I. (2010). *REST in Practice - Hipermedia and Systems Architecture.* Estados Unidos da América: O'Reilly Media, Inc.

Wodehouse, C. (2016). *The API Economy: How to Plug In and Build Your Business.* Santa Clara, California: Upwork, Inc.

Zachariadis, M., & Ozcan, P. (2017, Junho 15). The API Economy and Digital Transformation in Financial Services: The Case of Open Banking. *SWIFT Institute Working Paper*.

Zhong, H., & Su, Z. (2013). Detecting API documentation errors. *OOPSLA '13: Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications* (pp. 803-816). Nova Yorque, Estados Unidos da América: Association for Computing Machinery.

Anexo A – Inquérito de Satisfação

Este inquérito destina-se avaliar a satisfação de utilizar a nova versão da API da Slingshot quando comparada com a versão atual. Para cada uma das questões deve selecionar, para cada questão, qual o número que melhor descreve a satisfação do utilizador. A escala é a seguinte:

1 - Muito insatisfeito	2 - Insatisfeito	3 - Neutro	4 - Satisfeito	5 - Muito satisfeito

1 - Qual o grau de satisfação com a **velocidade de resposta** da API?

V1	1	2	3	4	5
V2	1	2	3	4	5

2 - Quão fácil é **utilizar** a API?

V1	1	2	3	4	5
V2	1	2	3	4	5

3 - Quão fácil foi a **aprendizagem inicial** de como utilizar a API?

V1	1	2	3	4	5
V2	1	2	3	4	5

4 - Qual a satisfação com a **documentação** da API?

V1	1	2	3	4	5
V2	1	2	3	4	5

5 - Qual a satisfação com a **estrutura da documentação** da API?

V1	1	2	3	4	5
V2	1	2	3	4	5

6 - Quão úteis são as **mensagens de erro**?

V1	1	2	3	4	5
V2	1	2	3	4	5

7 - Quão úteis são as os **exemplos de parâmetros**?

V1	1	2	3	4	5
V2	1	2	3	4	5

8 - Qual a satisfação com os **excertos de código**?

V1	1	2	3	4	5
V2	1	2	3	4	5

9 - Qual a satisfação com a **quantidade de funcionalidades** disponibilizadas?

1	1	2	3	4	5
V2	1	2	3	4	5

10 - Qual a satisfação com a **utilidade das funcionalidades** disponibilizadas?

V1	1	2	3	4	5
V2	1	2	3	4	5