INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Sistema de Gestão de Formulários para a Plataforma de Marketing Digital E-goi

TIAGO ALEXANDRE MOTA DOS SANTOS
Julho de 2018

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Sistema de Gestão de Formulários para a Plataforma de *Marketing* Digital E-goi

Tiago Alexandre Mota dos Santos

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas Gráficos e Multimédia

Orientadora: Isabel Azevedo

Supervisor: Ivo Pereira

Júri:

Presidente:

Vogais:

Porto, julho 2018

Dedicatória

A todos aqueles que me acompanharam durante o percurso académico,

em particular à minha família e amigos.

Resumo

A E-goi é uma plataforma de *marketing* digital multicanal que permite aos seus utilizadores crescerem os seus negócios através da gestão da sua base de dados de contactos, envio de campanhas, gestão de formulários, e outras funcionalidades necessárias aos profissionais de *marketing*. O sistema de gestão de formulários atual tem uma série de problemas como: elevado custo de manutenção, falta de funcionalidades e poucas métricas de análise de resultados, levando a que poucos utilizadores do E-goi utilizem esta componente do sistema.

O trabalho de mestrado descrito neste documento teve como objetivo o desenvolvimento de parte do novo sistema de gestão de formulários da plataforma E-goi, em particular os serviços de gestão da sua estrutura de dados. Foi proposta uma solução para os problemas apresentados, e para isso foi necessário perceber o contexto do sistema e as necessidades dos clientes, assim como analisar conceitos importantes na área de *marketing* digital, como por exemplo *web analytics*. Também foi clarificado o conceito de formulário e analisadas e comparadas soluções existentes de gestão de formulários. São apresentados os benefícios para o cliente, a proposta de valor e a oportunidade proporcionada por esta solução.

Esta dissertação documenta também o desenvolvimento técnico da solução de *software* através dos seus requisitos, modelo e artefactos de desenho, terminando com as experiências de avaliação realizadas para validar que a solução resolve o problema.

Palavras-chave: Formulário, *Web Analytics*, *Marketing* Digital.

Abstract

This thesis documents the development of a new form management system for the E-goi platform, specifically the services to manage the data structure. E-goi is a multichannel marketing platform that allows its users to grow their businesses by managing their contact database, sending campaigns, managing forms and other functionalities required by digital marketing professionals. The current form management system has a series of problems: not enough functionalities, high maintenance costs and not enough metrics to analyze form results, which leads to a weak usage of this component of the platform.

The present document proposes a solution for the problems above through the development of a new system. To understand the context of the system and the client needs some important concepts are analyzed, like digital marketing and web analytics. The meaning of form is also clarified, for that, existing solutions are analyzed and compared. The benefits for the client, as well as the value proposition for this solution, are defined through a value analysis.

This dissertation also documents the technical software development of the solution, through requirements and design artifacts, ending with the tests planned to validate that the solution developed really solves the problem.

Keywords: Form, Web Analytics, Digital Marketing.

Agradecimentos

Agradeço a todos aqueles que se cruzaram comigo durante este percurso académico e que, de alguma forma, contribuíram para o resultado deste trabalho.

Agradeço ao Instituto Superior de Engenharia do Porto pela qualidade da formação prestada, em especial a todos os seus docentes que através do seu conhecimento contribuíram para chegar a esta etapa.

Agradeço à professora Isabel Azevedo, pela sua disponibilidade, por ter ajudado a conduzir este projeto, e pelas críticas e sugestões que ajudaram a melhorar esta dissertação.

Agradeço à organização E-goi e todos os seus colaboradores por esta oportunidade e pelo incrível ambiente criado que permitiu guiar este projeto no melhor caminho possível. Em particular agradeço ao Ivo Pereira, supervisor deste projeto, que me acompanhou de forma incansável durante todo este percurso.

A toda a minha família e amigos que me ajudaram a chegar a finalista do Mestrado em Engenharia Informática.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Extratos de Código

Lista de Acrónimos e Siglas

AHP	*Analytic Hierarchy Process*
API	*Application Programming Interface*
BLOB	*Binary Large Object*
DDD	*Domain Driven Design*
FFE	*Fuzzy Front End*
HTML	*Hyper Text Markup Language*
IAM	Inconsistência Aleatória Média
IC	Índice de Consistência
IP	*Internet Protocol*
MVC	*Model View Controller*
NCD	*New Concept Development*
NPD	*New Product Development*
ORM	*Object Relationship Mapping*
PG	Prioridade Global
PML	Prioridade Média Local
RC	Relação de Consistência
REST	*Representational State Transfer*
SEM	*Search Engine Marketing*
SEO	*Search Engine Optimization*
SMS	*Short Message Service*
SWOT	*Strengths Weaknesses Opportunities Threats*
URL	*Uniform Resource Locator*
US	*User Story*

1 Introdução

Neste capítulo inicial é apresentado o problema que se pretende resolver com o trabalho descrito neste documento, assim como o seu contexto e os objetivos a atingir, os principais contributos da solução apresentada e ainda a abordagem seguida para resolver o problema identificado.

1.1 Contexto

Os formulários são usados pelo ser humano no seu dia a dia para aceder, por exemplo, ao *e-mail* ou para utilizar o cartão de crédito, e podem estar associados a tarefas com diferentes graus de importância, desde um simples questionário de avaliação, à votação para eleger o próximo governador. Um formulário pode ser definido como uma ferramenta utilizada para requisitar informação. Este é composto por um conjunto de campos a serem preenchidos por pessoas interessadas e, o seu objetivo é fornecer um modelo capaz de ser usado repetidamente para adquirir informação [1]. Os formulários podem ter diversos tipos de utilização, mas as equipas de *marketing* das organizações servem-se destes para angariar novos clientes, ou para obter a opinião dos seus clientes acerca do seu produto ou serviço. Os profissionais de *marketing* utilizam plataformas digitais de automação de *marketing* que, de entre muitas funcionalidades, permitem a criação e gestão de formulários.

O E-goi [2] é uma plataforma digital de automação de *marketing* multicanal através de canais como *e-mail*, SMS, *web*, voz e redes sociais. Esta plataforma auxilia o profissional de *marketing* no seu trabalho e o processo de tomada de decisão, através da automação de campanhas, criação de formulários, seguimento de comércio eletrónico, entre outras funcionalidades que permitem gerir o mercado da organização e também despertar interesse de clientes na sua oferta e potenciar a fidelização.

A gestão de formulários inclui a sua criação, através dos campos que o constituem e como este é apresentado, a sua publicação ou integração na página da organização criadora, a submissão do formulário por parte dos seus visitantes, e finalmente a análise dos resultados dessas submissões e do comportamento dos visitantes perante o formulário. As decisões tomadas na criação de um formulário podem ter bastante influência nas respostas dos seus visitantes, a simples definição de uma resposta por defeito pode induzir totalmente os resultados obtidos

[3], assim como a escolha do tipo de campo utilizado para requerer um pedaço da informação pode ditar por completo o sucesso ou insucesso de um formulário [4].

1.2 Problema

Com o crescimento da plataforma E-goi o sistema de gestão de formulários tornou-se desatualizado no que se refere à sua arquitetura, tecnologias e funcionalidades. A complexidade dos mecanismos de interação do sistema dificulta a sua utilização, enquanto que a complexidade da arquitetura e estrutura da aplicação dificultam a sua expansão e manutenção por parte dos desenvolvedores. Grande parte dos utilizadores do produto E-goi (cerca de 90% segundo estimativa fornecida pela empresa) não utilizam esta componente de gestão de formulários, e optam por concorrentes. Por outro lado, dos que utilizam, são poucos os que tem um número de visitas significativas nos seus formulários e um sucesso com os resultados obtidos.

Ao longo dos anos de atuação da plataforma E-goi a equipa percebeu, através de dúvidas dos clientes manifestadas à equipa de suporte, alguma insatisfação face ao sistema de gestão de formulários, revelando dificuldades na gestão dos formulários e principalmente na análise dos seus resultados. As dificuldades devem-se a ser um sistema bastante aberto o que o torna difícil aprender, dificultando também aos seus utilizadores a criação dos formulários que idealizaram de forma imparcial. Outra dificuldade é a falta de algumas funcionalidades relevantes a vários clientes da plataforma. Nota-se uma falta de clareza em transmitir corretamente os conceitos de formulário e sistema de gestão de formulários, que se reflete na falta de funcionalidades e em funcionalidades inadequadas.

As funcionalidades do sistema atual são suficientes para alguns clientes do E-goi, principalmente aqueles que procuram uma solução rápida e que têm um baixo volume de visitas nos seus formulários. Contudo, clientes que utilizam formulários regularmente e que têm um volume de visitantes maior necessitam de certas funções que o sistema não oferece, o que leva a que procurem essas soluções noutros sistemas. Funcionalidades como a definição de condições entre campos, que permite a criação de formulários dinâmicos adaptando-se às respostas dos utilizadores, análise dos resultados das submissões, que inclui a análise das repostas das submissões efetuadas e a análise do comportamento dos visitantes, são funcionalidades diferenciadoras em alguns concorrentes e em falta no sistema de formulários

do E-goi (cf. 3). Existem outras funcionalidades em falta (cf. 3.1) e é importante que um novo sistema esteja atualizado em comparação com outras soluções existentes, de forma a aumentar a sua utilização e minimizar a perda de clientes.

Os problemas arquiteturais do sistema atual (cf. 5.3) levam a que seja bastante difícil acrescentar essas funcionalidades, o que junto ao seu custo de manutenção justifica a criação de um novo sistema.

O sistema de gestão de formulários do E-goi oferece muitas mais funcionalidades do que as relacionadas com formulários, como por exemplo a criação de uma página completa, contudo essas funções do sistema encontram-se misturadas e não concretamente clarificadas criando dependências entre as funcionalidades oferecidas, e não evidenciando de forma compreensível as possíveis finalidades do sistema.

Identificam-se assim os seguintes problemas:

- Falta de clareza nos conceitos de formulário e sistema de gestão de formulários;
- Elevado custo de manutenção do sistema e dificuldade de expansão;
- Dificuldade na gestão do formulário;
- Falta de funcionalidades;
- Falta de métricas de análise de resultados.

A fraca utilização do sistema de gestão de formulários potencia a necessidade de desenvolvimento de um novo sistema que deve ser simples de manter, expandir e que responda às necessidades dos clientes (cf. 4).

1.3 Objetivos

O objetivo desta dissertação é a resolução dos problemas apresentados, desenvolvendo parte de um sistema de gestão de formulários em ambiente *web* para a plataforma E-goi, em particular os serviços de gestão da estrutura de dados. O sistema deve ser adaptado ao público a que se destina, neste caso, profissionais de *marketing* ou qualquer pessoa que pretenda crescer os seus projetos através de campanhas digitais.

Dado o elevado custo de manutenção do sistema atual é necessário desenvolver um novo sistema utilizando boas práticas de engenharia e perceber que funcionalidades não existem, mas que são necessárias e importantes para o mercado onde o sistema se insere. Para tal é necessário fazer uma análise de mercado que inclua uma análise das funcionalidades dos concorrentes. É apropriado estudar as reações e comportamentos de utentes de formulários e que linhas guia são utilizadas no seu desenho, para que o sistema desenvolvido permita criar formulários que sigam essas linhas (cf. 2.2.2).

De um ponto de vista tecnológico pretende-se um sistema usável em ambiente *web* para ser integrado no E-goi.

É importante que o novo sistema forneça as funcionalidades ao seu público alvo. Outro aspeto que o novo sistema deve ter em conta é a análise de resultados dos formulários, o qual deve apresentar métricas relevantes (cf. 2.3) para as organizações poderem tirar conclusões que permitam melhorar o seu negócio.

Os objetivos do trabalho desenvolvido foram:

- Investigar o estado de arte de soluções existentes e clarificar o conceito e abordagens de gestão de formulários;
- Desenhar e implementar a estrutura de dados necessária para suportar o sistema de gestão de formulários;
- Desenvolver os serviços de gestão da estrutura de dados dos formulários e os serviços de análise de resultados dos formulários;
- Implementar a camada pública de acesso aos formulários;
- Integrar o novo sistema de gestão de formulários no produto E-goi.

1.4 Contributos

O valor de um produto é criado quando na perspetiva de um cliente se este sente que obtém benefícios perante os sacrifícios necessários para adquirir e utilizar o produto ou serviço da organização. Os sacrifícios necessários não incluem apenas o custo monetário do produto, mas todo o investimento de tempo e esforço que é necessário para a sua utilização. A perceção de benefícios pode ser adquirida diretamente através da aquisição do produto, ou de forma

indireta, como por exemplo, através da rápida aprendizagem de utilização [5]. O processo de inovação e criação continua de valor é um aspeto onde existe um enorme investimento por parte das organizações na tentativa de superar as expetativas dos seus clientes, porém é um processo difícil de controlar e só pode ser executado quando, no ponto de vista do mercado, os benefícios oferecidos pela organização são superiores aos sacrifícios necessários [6].

A solução apresentada nesta dissertação apresenta valor para os profissionais de *marketing* e para todos aqueles que pretendem aumentar a utilização dos seus produtos ou serviços, mas, em particular, é criado valor para os utilizadores do produto E-goi. Note-se que, apesar do presente projeto se basear em conceitos de um sistema já existente, existem várias melhorias.

Este projeto pretende inovar no ambiente de *marketing* digital através da integração no E-goi de um sistema de gestão de formulários capaz de efetuar uma análise estatística detalhada das submissões dos formulários, processando as respostas submetidas e dados dos visitantes. O processo de inovação também passa por adicionar melhorias em relação ao sistema existente que incluem a possibilidade de criar formulários dinâmicos, tendo em conta as respostas dos visitantes nos campos do formulário.

1.5 Abordagem Preconizada

A resolução do problema proposto foi efetuada seguindo uma abordagem que permitiu aumentar a granularidade do problema e, em simultâneo, expandir a sua compreensão progredindo em direção à solução.

Inicialmente foi elaborado um estudo para perceber e definir concretamente os conceitos do contexto em que o problema está inserido. No caso desta dissertação *marketing* digital é o contexto, e é necessário clarificar e definir o conceito de formulário e conteúdos relacionados, tais como as funcionalidades presentes em sistemas de gestão de formulários que podem ser identificadas através da análise de soluções existentes.

Seguido à clarificação dos conceitos e análise de soluções de gestão de formulários existentes decide-se qual a abordagem a utilizar para resolver o problema, estudando o valor que esta traz para o negócio e que oportunidade pretende preencher.

Após identificada a abordagem de gestão de formulários que permita resolver os problemas identificados, a solução é desenhada, desenvolvida e testada para que possa ser validado que a solução resolve de facto o problema.

Finalmente são enumerados os objetivos atingidos, dificuldades encontradas e trabalho a ser realizado futuramente para dar continuidade ao trabalho desenvolvido.

A informação necessária para a resolução do problema, assim como a informação sobre o próprio problema, foi obtida a partir de algumas das partes interessadas no sistema, nomeadamente os colaboradores da E-goi, através de reuniões constantes. Para perceber as componentes técnicas do sistema foi feito um trabalho direto com os desenvolvedores da plataforma.

1.6 Estrutura do Documento

O presente documento está estruturado em diversos capítulos:

- **Introdução:** este capítulo apresenta o problema abordado por esta dissertação, a solução proposta, abordagem de resolução e estrutura deste documento;
- **Contexto:** neste capítulo são descritos conceitos relevantes para o problema;
- **Estado da arte:** este capítulo descreve o estado da arte de soluções e abordagens existentes;
- **Análise de Valor:** neste capítulo é descrito o valor criado pela solução proposta e as necessidades que esta vem preencher;
- **Desenvolvimento da Solução:** este capítulo apresenta o desenvolvimento técnico da solução através dos seus requisitos, desenho e detalhes de implementação;
- **Experiências e Validação da Solução:** neste capítulo são descritas as grandezas e metodologias utilizadas para avaliar a solução, assim como são descritas as experiências realizadas e os resultados obtidos;
- **Conclusões**: neste capítulo são apresentados os objetivos alcançados, como o problema foi abordado e trabalho futuro.

2 Contexto

Neste capítulo é apresentado de uma forma detalhada o contexto envolvente da solução proposta nesta dissertação, assim como os intervenientes, processos e conceitos de negócio. São definidos os conceitos de *marketing*, canais, *marketing* digital, formulários e *Web Analytics*, que facilitam e compreensão do sistema pretendido e o meio envolvente.

2.1 Marketing

Marketing é qualquer relacionamento interpessoal ou interorganizacional, que tem na sua essência uma transação que pretende satisfazer as necessidades de ambas as partes da mesma. Neste processo existem três elementos principais: o profissional de *marketing* (conhecido como *marketeer*), o produto ou serviço transacionado, e o mercado alvo [7].

Marketing também pode ser definido como o processo social pelo qual os indivíduos e grupos obtêm o que necessitam ou querem, criando e trocando produtos e valor com os outros [8]. Desta definição retiram-se conceitos importantes como: necessidades, valor, transação e os indivíduos que fazem parte deste processo.

Uma necessidade pode ser entendida como um sentimento de falta de algo, já o querer é uma vontade específica de realizar ou adquirir algo que pode ou não preencher uma necessidade. Muitas vezes aquilo que um indivíduo quer não é o que ela precisa na realidade. Valor é a capacidade que um produto ou serviço tem de satisfazer as necessidades de um cliente. A transação é o processo de obter um produto ou serviço oferecendo algo em retorno, para ser um processo viável é necessário que ambos os extremos da transação consigam oferecer valor ao outro. Tendo em conta os conceitos anteriores pode-se definir um mercado como um conjunto de indivíduos com a mesma necessidade, dispostos a transacionar algo em troca de um produto ou serviço que satisfaça essa necessidade [9].

Assim, o *marketing* procura diminuir a barreira entre os clientes e o produto/serviço identificando as suas necessidades e entregando valor.

Para definir uma estratégia de *marketing* é necessário compreender o ciclo de vida de um mercado:

- **Lançamento**: o mercado sofre uma inovação seja tecnológica ou em habilidades;

- **Crescimento**: ocorre uma grande expansão do volume de negócio, aumentando clientes e vendas;
- **Maturidade**: já não ocorre crescimento, dá-se uma estabilidade do volume de negócio, das tecnologias, e é baixa a probabilidade de entrada de novos concorrentes;
- **Declínio**: o volume de negócio decresce.

Após identificar a fase em que o mercado se encontra para delinear uma estratégia de marketing é necessário definir um conjunto de variáveis, Jerome McCarthy [10] identifica-as como quatro:

- **Produto**: bem ou serviço que a organização fornece ao seu mercado;
- **Preço**: montante que é necessário pagar para adquirir o produto/serviço;
- **Distribuição**: processo de disponibilizar o produto ao consumidor final;
- **Comunicação**: processo de dar a conhecer o produto/serviço e os seus benefícios aos possíveis clientes através de múltiplos canais.

2.1.1 Canais de *Marketing*

Um canal de *marketing* é o conjunto de todas as pessoas, organizações, meios e atividades envolvidas em levar o produto/serviço a possíveis clientes [11]. A escolha dos canais a utilizar e a forma de chegar ao cliente são aspetos bastante importantes na definição de uma estratégia de negócio, pois os mesmos canais têm retornos diferentes consoante o negócio em que são utilizados. Consequentemente é necessário analisar qual o canal mais apropriado. Exemplos de canais tradicionais são publicações em jornais ou revistas, a rádio, a televisão, a comunicação presencial e o telefone. Canais mais recentes tipicamente usam a Internet como veículo de comunicação, como as redes sociais, e-mail, motores de busca e a própria página *online* da organização. Verifica-se que a evolução destes canais é constante assim como o aparecimento ou criação de novos canais.

O *marketing* pode ser classificado de diversas formas [12]. Destacam-se as mais relevantes para o trabalho a desenvolver:

- **Direto**: são enviadas mensagens personalizadas diretamente para um consumidor específico, por exemplo, através de um telefonema;

- **Indireto**: a ação de divulgação é realizada através de intermediários apanhando o consumidor de surpresa, por exemplo, através de anúncios durante um filme ou programa televisivo;
- ***Marketing* social** [13]: ação que muda os comportamentos de um público alvo para atingir objetivos do negócio, sejam eles económicos, políticos ou sociais. Um exemplo de *marketing* social é a divulgação de eventos, contribuindo indiretamente para a divulgação da organização que organiza o evento perante os indivíduos interessados nos objetivos do evento;
- ***Marketing* digital** [14]: *marketing* realizado com suporte a meios digitais. A importância da utilização dos meios digitais para promoção de uma organização varia consoante o produto/serviço disponibilizado por esta. Esta forma de *marketing* é essencial para aproximar a organização dos seus clientes, principalmente em negócios onde existe uma grande dispersão geográfica.

2.1.2 Marketing Digital

O *marketing* digital é a comunicação com o mercado através da Internet ou outros meios digitais, com objetivo de comercializar ou fidelizar novos clientes [15]. Esta forma de *marketing* vem potenciar os resultados do *marketing* direto tradicional, mas através de meios digitais, podendo conseguir obter nestes novos canais uma maior eficiência e eficácia. A importância da utilização dos meios digitais para promoção de uma organização varia consoante o produto/serviço disponibilizado por esta. Para muitas organizações que não estão limitados por condições físicas do produto ou serviço, este acaba por ser um meio de grande aquisição de clientes [14]. Porém a utilização deste meio não é exclusiva, a organização pode recorrer a outros canais para implementar uma estratégia multicanal. Mesmo dentro do *marketing* digital podem ser utilizadas várias estratégias [15]:

- ***Search Engine Optimization* (SEO)**: otimização de uma página para obter melhor posicionamento nos motores de busca;
- ***Search Engine Marketing* (SEM)**: pagar para uma página ganhar mais visibilidade nas pesquisas de um determinado motor de busca;
- ***Points of Sale E-Management***: venda inteligente de artigos, personalizada em função do histórico de compras do cliente;
- **Cupões *online***: cupões de desconto distribuídos através da Internet;

- ***E-commerce***: comercio eletrónico através de uma loja *online* que disponibiliza a compra dos produtos da organização;
- ***E-mail marketing***: marketing realizado através do canal de *e-mail*, que tem a vantagem de os comportamentos de interação com a mensagem poderem ser facilmente analisados, porém é um canal muito usado e é muito fácil um *e-mail* ficar esquecido ou ser bloqueado por filtros de *spam*;
- ***Marketing* de conteúdos**: permite criar e distribuir valor através de conteúdos que não sejam necessariamente os produtos ou serviços da organização, mas que levem os clientes a suscitar interesse por eles;
- ***Marketing* por jogos**: jogos especificamente desenvolvidos para promover marcas, produtos ou serviços;
- **SMS *marketing***: conhecimento ou subscrição aos produtos/serviços da organização através de SMS. Textos curtos permitem a simples perceção dos serviços da empresa, comandos automatizados permitem a fácil subscrição.

Existem muitas outras formas de *marketing* digital, tais como as notificações *push*, recebidas em telemóveis e, mais recentemente, nos *browsers,* usadas para publicitar marcas ou relembrar clientes de campanhas. Destaca-se também o *marketing* por influenciadores, que visa tirar partido de personalidades com um vasto público sobre a sua influência para publicitar um produto ou serviço, entre outras.

Novos canais e formas de comunicar com o mercado surgem a todo o instante, é importante que os profissionais as usem de forma eficiente e eficaz. O *marketing* digital traz diversas vantagens, como facilitar a comunicação das marcas com os clientes, simplificar o suporte, melhorar a segmentação do mercado, tornar a comunicação e distribuição multicanal e facilitar a medição e análise de resultados. Em geral o meio digital torna o marketing mais eficaz [16] pois os esforços são colocados em resolver os problemas dos clientes e não no canal usado para chegar a ele. Contudo também existem algumas desvantagens, tais como problemas em localizações onde há fraco acesso à Internet e a elevada concorrência utilizando os mesmos canais.

O *marketing* digital utiliza estratégias de atração e conversão de clientes em que são os próprios clientes que procuram a organização e esta gradualmente leva-os a fidelizarem-se. A *Figura 1* apresenta um diagrama em funil com o conjunto de fases de aquisição e atração de clientes no

marketing digital, onde em cada fase há uma diminuição do número de clientes que chegam até ela, mas um aumento da relação de confiança da organização com os clientes.

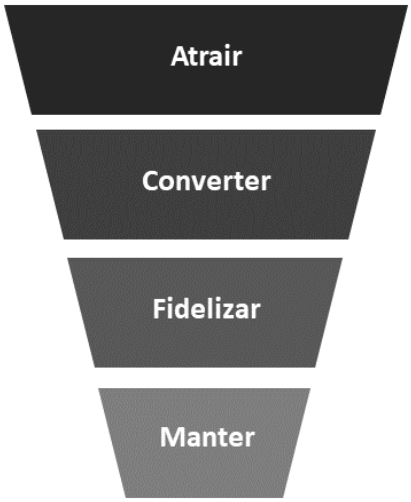

Figura 1 Fases de aquisição de clientes no marketing digital [17]

A aquisição de clientes começa com a atração de visitas para conhecer o negócio da organização, para tal são utilizadas técnicas como SEO, anúncios e *marketing* por conteúdos. Para avançar no aumento de confiança e relacionamento com a organização é necessário converter algumas das visitas atraídas em possíveis clientes, para tal são utilizadas técnicas como formulários e páginas com ofertas e informação sobre a organização. Os formulários têm bastante importância nesta fase pois são a principal fonte de entrada de contactos para a lista de possíveis clientes da organização. Dos possíveis clientes angariados é desejável que estes tomem a opção de se fidelizar à organização passando a usufruir dos seus produtos ou serviços, para otimizar a procura do momento de fidelização ideal recorre-se a áreas de *marketing* digital como *e-mail marketing* e automação de *marketing*. Após a fidelização de um novo cliente é essencial analisar detalhadamente os comportamentos e perfil do cliente para o manter fiel ao produto ou serviço da organização o máximo tempo possível.

2.2 Formulários

Sendo o E-goi um sistema de *marketing* digital multicanal, permite diversas funcionalidades necessárias na área, tais como:

- Criação e gestão das listas de contactos de uma organização, adicionando novos contactos automaticamente quando ocorre a subscrição de um cliente;
- Criação e personalização de campanhas que podem ser enviadas por diversos canais e posteriormente analisados os resultados das mesmas;
- Criação de formulários que podem ser integrados nas páginas das organizações, formulários esses que podem ter diversos objetivos, como adquirir novos subscritores para as listas de *marketing* da organização, angariar novas fidelizações, recomendar a organização, preencher questionários de satisfação, entre outros;
- Entre outras funcionalidades que permitem os profissionais de *marketing* gerir o processo de aquisição de clientes de uma organização.

Os formulários existem desde que existe a necessidade de requisitar informação, seja através de um impresso para realizar uma inscrição, aos tempos mais modernos em que a informação é requisita e registada através de um dispositivo eletrónico. Um formulário pode ser definido como uma ferramenta utilizada para requisitar informação, e é composto por um conjunto campos a serem preenchidos juntamente com conteúdo fixo [1]. Num ambiente *web* um formulário permite ao utilizador inserir dados para estes serem enviados e processados por um servidor. Contém uma lista de campos que podem ser de diversos tipos como caixas de texto, escolha múltipla, seleção, entre outros. Um sistema de formulários digital permite a criação de formulários através de um editor gráfico e a sua gestão, desde a personalização do formulário à análise dos dados submetidos.

Em seguida são apresentados tipos de formulários e exemplos da sua utilização, boas práticas de desenho de formulários e um fluxo típico de gestão destes.

2.2.1 Tipos

Os formulários podem ter diversas utilizações. Esta subsecção apresenta os tipos de formulários principais, identificados no domínio E-goi, e a sua utilização do âmbito de *marketing* digital: inscrição, recomendação, atualização e questionário.

2.2.1.1 Inscrição

A *Figura 2* apresenta um exemplo de um formulário de inscrição onde são solicitados os dados pessoais do utilizador. No contexto de *marketing* digital, os formulários de inscrição (também conhecidos por formulários de subscrição), são usados quando a organização pretende obter novos subscritores para as suas listas de contactos ou obter contactos para receber as suas notícias. Este tipo de formulário é o mais utilizado pelas equipas de *marketing* pois é através dele que são conquistados potenciais clientes (também conhecidos como *lead*). Muitas empresas utilizam recompensas em troca dos dados dos visitantes como forma de incentivo ao preenchimento do formulário.

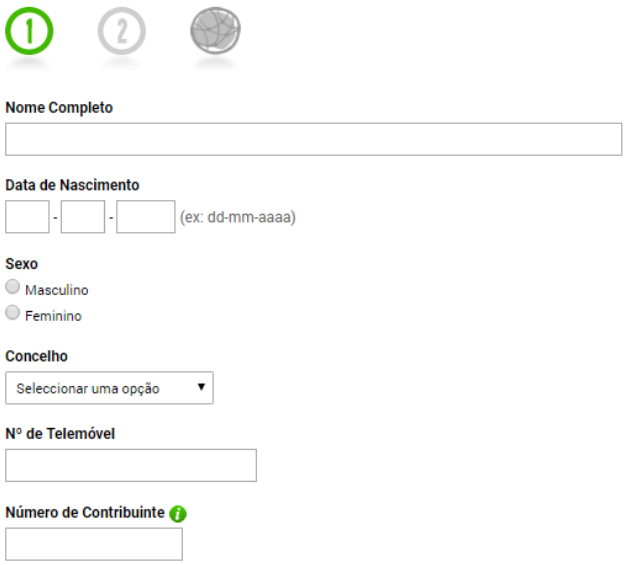

Figura 2 Formulário de inscrição [18]

2.2.1.2 Recomendação

A *Figura 3* mostra um formulário de recomendação, onde o utilizador do formulário pode sugerir uma lista de amigos através dos seus contactos *e-mail* para usufruir do serviço da organização. Esta é também uma forma de angariação de possíveis clientes e uma técnica utilizada em conjunto com este tipo de formulário é a oferta de recompensas relacionadas com o serviço da organização por cada amigo convidado, ou por cada que venha a utilizar o serviço.

Figura 3 Formulário de recomendação [19]

2.2.1.3 Atualização

A *Figura 4* representa um formulário de atualização, onde é sugerida ao utilizador informação em falta não obrigatória aquando da criação do seu perfil. Este tipo de formulário é bastante útil quando é necessário adicionar nova informação para requisitá-la a utilizadores já inscritos. Os formulários de atualização diferem dos formulários de inscrição pois os primeiros não adicionam um novo subscritor, apenas atualizam a sua informação.

Figura 4 Formulário de atualização [20]

2.2.1.4 Questionário

A *Figura 5* apresenta um exemplo de um questionário, onde é efetuada uma questão ao utilizador e é apresentado um conjunto de respostas das quais este pode escolher. Um questionário é um tipo de formulário que pode ser bastante extenso, portanto é aconselhada a sua divisão num conjunto de passos tal como indicado no exemplo através da barra de progresso. Os questionários estão tipicamente associados à recolha de informação ou avaliação direta de um serviço ou problema, portanto no fim do preenchimento é atribuída uma classificação ao utilizador com base nas suas repostas. No contexto de *marketing* digital os questionários são utilizados para as organizações obterem as opiniões dos seus clientes sobre o seu trabalho e eventualmente efetuarem alterações caso necessário.

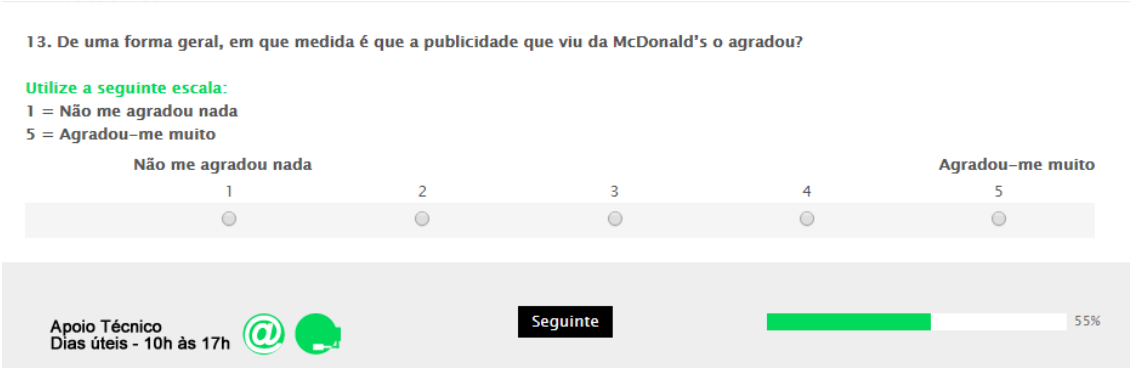

Figura 5 Questionário [18]

Existem mais tipos de formulários para além dos definidos, como formulários de pagamento, doação, ou formulários de concursos.

2.2.2 Desenho

O desenho de um formulário, nomeadamente o seu aspeto e conteúdo, tem grande impacto nas suas respostas, podendo potenciar uma experiência satisfatória com influência no conteúdo das respostas [21].

Um formulário não é bom só pelo seu aspeto gráfico. A informação que é requerida ao utilizador e a forma como é pedida são os fatores mais importantes. Para desenhar um formulário

adequado é necessário conhecer o público alvo e como tipicamente reage a formulários. Algumas pessoas podem ter tendência a ler toda a informação, sendo importante reduzir ao máximo informação desnecessária no formulário, enquanto outras podem apenas preencher os campos importantes com o objetivo de finalizar o preenchimento rapidamente. Conseguir classificar o público alvo de um formulário pode ajudar a tomar decisões no seu desenho. Reduzir a informação pessoal necessária a introduzir pelo utilizador também é uma boa prática, assim como ordenar as perguntas de forma a ganhar confiança do utilizador de pergunta a pergunta, por exemplo, requisitar em primeiro lugar o seu nome e só depois a sua morada [1].

Oferecer recompensas ao utilizador é também um aspeto motivador para ele continuar a preencher o formulário [4]. O tamanho adequado para o mesmo depende da relação que a organização tem com o visitante do formulário. Pode ser necessário uma pessoa que seja cliente há muito tempo preencher bastante informação e ela não se sentir desmotivada para tal. Já um novo cliente se quiser subscrever às notícias da organização ou aos seus serviços vai querer preencher a menor quantidade de informação possível. O tamanho do formulário deve estar diretamente relacionado com a sua recompensa de valor assim como o esforço necessário para o preencher. Definir concretamente o objetivo do formulário e evitar que o utilizador tenha de repetir ações ou responder à mesma pergunta mais que uma vez são também boas práticas a ter em consideração.

Outros guias que podem ser seguidos na criação de um formulário, incluem [4]:

- Reduzir o esforço mental requerido ao utilizador, fazendo apenas perguntas simples e evitando colocar o utilizador num papel que tenha de tomar decisões;
- Agrupar questões relacionadas;
- Evitar perguntas abertas, requerem mais esforço para analisar e podem levar a falsas conclusões pois as mesmas pessoas podem dar respostas diferentes em situações diferentes;
- Escolher controlos adequados para as perguntas, por exemplo, selecionar em vez de escrever caso as respostas sejam conhecidas à partida, mas permitir escrever caso exista uma longa lista de escolhas;
- Não escolher valores por defeito pelo utilizador pois pode influenciar a sua resposta [3];
- Identificar claramente se pode ser escolhida mais que uma opção ou não,

- Fazer perguntas de forma suficientemente explícita para o utilizador perceber a pergunta sem a leitura das escolhas possíveis.

Um aspeto bastante importante a ter em consideração é quando o formulário for muito extenso e não exista a possibilidade de diminuir o número de questões. Nestes casos é considerada boa prática dividi-lo em secções ou páginas, mas mantendo a coerência de tema em cada secção e fornecendo ao utilizador uma visualização da progressão indicando quantas secções são necessárias preencher. O número de campos ao qual o formulário pode ser considerado extenso varia de acordo com o contexto de aplicação do formulário e com a relação da organização com o cliente.

2.2.3 Fluxo de Gestão

A gestão de formulários segue um fluxo de ações ou fases tal como indicado na *Figura 6*. Este fluxo é identificado e proposto no âmbito desta dissertação tendo em conta as soluções analisadas (cf. 3.1).

A primeira fase é a criação ou construção do formulário através dos seus campos configurando as propriedades de cada campo. A segunda fase é a edição ou configuração dos aspetos gráficos do formulário e como este é apresentado ao utilizador. Após criação e edição do formulário é feita a sua publicação, que pode ser realizada de diversas formas, como integração na página da organização, envio por *e-mail* ou integração em sistemas externos à organização como as redes sociais.

Após o formulário ser criado e disponibilizado ao seu utilizador é essencial avaliar os comportamentos e reações destes, recolhendo e analisando os resultados obtidos para com base nesses poderem ser aplicadas melhorias. Os resultados podem incluir as respostas diretas das submissões do formulário com significado para organização, ou apenas informação estatísticas como as localizações de acesso e dispositivos. Uma vez analisados os resultados, é possível tirar conclusões, é importante que as conclusões sejam fiáveis e o formulário seja imparcial às mesmas. A análise dos resultados pode levar a perceber problemas no formulário, como por exemplo, dificuldades a preencher um determinado campo, o que implica voltar a uma das fases anteriores para reestruturação do formulário.

A qualquer fase do fluxo é possível voltar a fases anteriores para alteração do formulário, por exemplo, após a análise dos primeiros resultados caso se perceba que falta um campo no formulário é possível voltar à fase de criação para adicionar o campo. Também se pode perceber que o formulário não tem resultados significativos numa determinada plataforma e ser necessário voltar à fase de publicação para eliminar a publicação correspondente a essa plataforma. A possibilidade de voltar atrás em qualquer fase torna este fluxo iterativo, que permite uma constante melhoria. Antes da criação do formulário inicial também é necessária uma fase de estudo do público alvo e de que informação será requerida e como, contudo, essa fase é deixada fora deste fluxo pois é responsabilidade dos criadores do formulário e não do sistema de gestão.

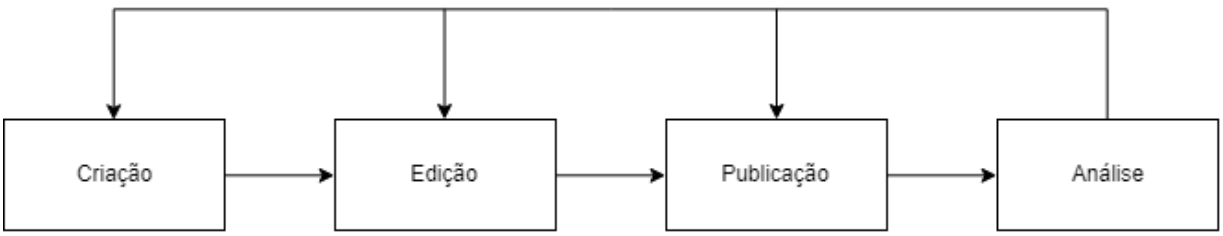

Figura 6 Fluxo de gestão de formulários

Existem diversos aspetos a ter em consideração na criação e gestão de um formulário, desde os elementos a utilizar ao conteúdo que será requerido. Um sistema de gestão de formulários deve ser adaptado ao público alvo e ao negócio onde está inserido, deve ser considerado qual o grau de liberdade dos formulários que serão criados em termos de aspeto e conteúdo, que elementos e campos vão ser disponibilizados para inserir no formulário, que opções são possíveis de configurar, e ainda qual a granularidade de resultados que é necessário obter.

2.3 Web Analytics

A análise dos resultados dos formulários inclui dois tipos de dados: a informação diretamente relacionada com os campos e respostas dos visitantes (e.g. *e-mails* submetidos num formulário de inscrição), e a informação estatística relacionada com as visitas ao formulário e comportamento dos visitantes (e.g. localização de acesso, número de visitas, taxa de *clicks*). O processo de aquisição da informação das visitas, no contexto *web*, é conhecido por *web analytics*. A Web Analytics Association define *web analytics* como "a coleção, medida, análise e

a elaboração de relatórios de dados adquiridos na Internet com o objetivo de perceber e otimizar a utilização da *web*" [22].

Esta associação divide dados possíveis de analisar em dois tipos:

- **Métricas**: que são valores com significado bastante específico;
- **Dimensões**: que são categorias compostas por métricas.

2.3.1 Métricas

Métricas são dados com um significado específico, são quantificáveis ou limitadas a um conjunto de valores, estas permitem medir e caraterizar uma determinada ação ou evento.

As métricas principais [23] incluem:

- **Número de visitas**: total de visitas que acederam à página ou conteúdo;
- **Duração da visita**: tempo total de acesso o conteúdo durante uma visita;
- **Número de visitantes habituais**: número de visitantes que visitaram mais que uma vez a página ou conteúdo;
- **Número de visualizações**: número de vezes que um determinado conteúdo ou página foi visualizado;
- **Fontes**: origem do tráfego do visitante que o lhe vou a chegar a determinado conteúdo;
- **Dispositivos**: especificações técnicas dos dispositivos e plataformas utilizados durante a visita;
- **Taxa de rejeição**: rácio entre o número de visitas total e as visitas que saíram da página executando apenas uma ação e sem visitar outro conteúdo;
- **Número de conversões**: total de visitas que atingiram o objetivo que a página oferece, por exemplo a subscrição ao conteúdo da organização;
- **Taxa de *clicks*:** porção de visitantes que carregaram num determinado conteúdo.

2.3.2 Dimensões

Dimensão é um conjunto de métricas com significado comum ou que pretendem avaliar o mesmo conceito de formas diferentes.

As dimensões podem ser divididas em quatro categorias de dados principais:

- **Visitante**: dimensão que representa o utilizador que esta a aceder à página. Existem diferentes metodologias para seguir um utilizador da *web*, tais como, autenticação, *cookies* e IP. Autenticação é o método mais exato para identificar um visitante, mas como muitos *sites* não necessitam de autenticação são usadas cookies ou outros métodos para o identificar, porém existem problemas nesta metodologia quando são usados dispositivos diferentes pela mesma pessoa ou quando os *cookies* são removidos;
- **Visita**: dimensão que representa uma visita de um utilizador a uma página *web*. Associadas à visita estão as ações realizadas durante a mesma (e.g. *click* numa hiperligação, carregamento de um ficheiro). A ação de atualização de uma página não conta como nova visita, mas é registado como ação integrante da visita, uma visita é dada por terminada tipicamente após 30 minutos sem ocorrer ações. Algumas métricas associadas à visita são a data de início e fim, o *browser* usado, o dispositivo e a localização de acesso;
- **Página**: dimensão que representa uma página *web* e as métricas associadas, como por exemplo, o número de visualizações da página.
- **Evento**: dimensão que representa uma ação na página, por exemplo, inserir texto num campo, carregar num botão ou premir uma tecla.

Outra dimensão de dados que podem ser obtidos na *web* são os referentes, um conceito que representa a fonte que originou a chegada do visitante à página que se encontra atualmente, os referentes podem ser de diversos tipos:

- **Internos**, por exemplo, um URL dentro do *site* que a página se insere;
- **Externos**, como um URL fora do *site* que a página se insere;
- **Referentes de pesquisa**, como os motores de busca;
- **Navegação direta**, ou seja, sem referente no caso do visitante introduzir diretamente o URL.

Outras métricas permitem caraterizar o comportamento de um utilizador durante uma visita, como o rácio de saída de uma página, o número de visitas que se resumem a visitar uma única página sem executar qualquer outra ação na mesma (conhecido como taxa de rejeição) e o número de páginas por visita. A conversão de uma visita também pode ser caraterizada como uma dimensão, que mede o número de visitas que representaram sucesso para o negócio (e.g.

compra numa loja online), métricas possíveis de medir nesta dimensão são o número de conversões e o rácio de conversão tendo em conta o número total de visitas.

O sistema de gestão de formulários deve ser capaz de recolher e medir toda esta informação acerca dos visitantes dos formulários, desde os resultados das submissões efetuados às interações e comportamentos com os campos que o constituem.

3 Estado da arte

Neste capítulo é feita uma análise de soluções existentes de gestão de formulários com o objetivo de identificar abordagens utilizadas nesta área. Também são analisadas soluções tecnológicas de *web analytics* para perceber se alguma destas pode ser utilizada no projeto.

3.1 Soluções de Gestão de Formulários Existentes

Nesta secção são analisadas e comparadas soluções similares à proposta existentes à data desta dissertação. Esta análise pretende perceber o estado de arte de sistemas concorrentes com o objetivo de identificar abordagens de gestão de formulários e definir qual a mais apropriada à resolução do problema. Os sistemas analisados foram escolhidos tendo em conta a sua popularidade, funcionalidades, contexto de aplicação e custo, procurando uma representação ampla e realista dos sistemas existentes. No final são comparados segundo as suas funcionalidades para perceber quais as adequadas no âmbito de *marketing* digital e em falta no sistema de formulários do E-goi.

3.1.1 JotForm

Trata-se de um sistema de gestão de formulários lançado pela empresa JotForm [24], em 2006, que permite a criação de formulários através de um editor de arrastar e largar campos. Os campos podem ter significado contextual como *e-mail*, telefone, data, ou podem ser básicos como entrada de texto e escolha múltipla. Os elementos são inseridos e apresentados de forma sequencial e as propriedades específicas de cada tipo de elemento podem ser editadas, para além dessas propriedades todos os elementos permitem editar o seu aspeto configurando parâmetros como o alinhamento, tipo de letra, cor, entre outros. Este sistema permite configurar o estado do formulário, por exemplo, desativando-o numa determinada data ou quando atingir um limite de submissões, ou impedir que um determinado utilizador submeta uma resposta ao formulário mais que uma vez. Pode ser definida a língua em que este é apresentado e as traduções específicas para cada campo.

Uma configuração oferecida pelo JotForm é a encriptação dos dados submetidos nos campos, tornando a submissão mais segura e impedindo danos à privacidade do utilizador. Outras funcionalidades deste sistema permitem definir uma lista de contactos a serem notificados

quando ocorre uma submissão de um formulário, a criação de formulários dinâmicos através da definição de condições entre campos, a possibilidade de integrar recursos de outras aplicações no formulário, a definição de como cada campo é validado e quais as mensagens de erro apresentadas.

Com o formulário completamente definido e configurado este pode ser publicado e partilhado, o qual pode ser feito através de uma hiperligação, incorporado diretamente noutra página ou partilhado através de funcionalidades de integração com outros sistemas como as redes sociais. Finalmente o JotForm permite analisar os resultados das submissões, desde a visualização de cada submissão, a estatísticas das mesmas como o rácio de conversão, tempo médio de preenchimento do formulário, dispositivos e plataformas utilizadas.

Os preços deste sistema variam tendo em conta o número de submissões, espaço utilizado e número de visualizações.

3.1.2 Google Forms

É um sistema de gestão de formulários desenvolvido pela Google [25] e lançado em 2012, é o sistema mais popular na área por consequência da visibilidade da organização desenvolvedora e da sua simplicidade de utilização. Este sistema permite definir campo a campo o formulário, identificando o tipo de campo e a informação presente neste.

O Google Forms permite limitar o número de respostas de um formulário, mostrar uma barra de progresso, apresentar os campos segundo uma ordem aleatória e definir a mensagem de confirmação. Em termos de apresentação apenas pode ser configurada a cor do formulário.

Dada a simplicidade dos formulários criados, estes podem ser transformados em questionários aos quais são adicionadas funcionalidades de classificação das respostas. A publicação do formulário pode ser feita através de uma hiperligação, incorporação noutros sistemas ou diretamente através de um *e-mail.*

3.1.3 Wufoo

O sistema de gestão de formulários desenvolvido pela empresa Wufoo [26] em 2006 (atualmente SurveyMonkey), permite a construção do formulário campo a campo através de uma ferramenta de arrastar e largar campos. Este sistema também permite a criação e personalização de temas que representam não só a estrutura do formulário, mas também o seu aspeto, possibilitando ainda a definição de lógica condicional entre campos.

As configurações do formulário incluem configurações básicas como nome, descrição e língua, e configurações avançadas como a mensagem de confirmação e a limitação do número de submissões. Os formulários criados com o Wufoo são responsivos funcionando tanto em ambiente *web* como ambiente móvel. Os resultados dos formulários podem ser analisados em tempo real ou através de relatórios, onde constam dados de cada visita e os resultados obtidos com a mesma.

Os preços deste sistema variam tendo em conta o número de formulários pretendidos na conta, o número de campos do formulário e o número de submissões.

3.1.4 Form.io

Form.io [27] é um gestor de formulários de código aberto que permite a sua criação através de um editor de arrastar e largar campos, criando automaticamente os recursos necessários para gestão de cada formulário, o que torna este sistema apropriado para aplicações sem servidor. Associado a esta funcionalidade de criação dos recursos é possível criar papéis de utilizadores juntamente com funcionalidades de autenticação para limitar que tipos de utilizadores podem aceder a um formulário. Outras funcionalidades do Form.io incluem lógica condicional, versionamento de formulários, definição de um fluxo entre formulários e encriptação de campos.

Este sistema permite criar formulários multilingue e dinâmicos com fluxo entre vários formulários possibilitando uma enorme diversidade de utilização, contudo é bastante limitado na edição do aspeto dos formulários criados.

O preço deste sistema é definido mediante as funcionalidades que são disponibilizadas.

3.1.5 E-goi

É uma plataforma de automação de *marketing* digital que de entre funcionalidades de automação de campanhas para múltiplos canais e gestão de base de dados de contactos, também permite a criação de formulários. O sistema de gestão de formulários do produto E-goi permite a sua criação através de um editor de arrastar e largar elementos que podem corresponder a campos, podendo ser mapeados aos campos das listas de contactos para adicionar clientes a uma lista automaticamente. Os elementos também podem ser componentes encontrados em *interfaces web* como imagens, inserção de texto, botões, entre outros. Os elementos podem ser organizados segundo uma grelha, e não só as suas propriedades específicas podem ser ajustadas como também propriedades relacionadas com o aspeto do elemento e como este se apresenta na página.

No que concerne às configurações do formulário é possível definir a língua, limitar o número de submissões por quantidade ou por utilizador, definir o resultado da submissão, ativar o *double opt-in* para um utilizador confirmar a sua subscrição, entre outras possibilidades de configuração.

A publicação do formulário pode ser realizada diretamente através de uma hiperligação, *pop-up*, ou integrações com plataformas e tecnologias externas. Os relatórios de análise de resultados fornecem o número de visitas e a taxa de rejeição por publicação do formulário, para além de ser possível analisar cada resposta submetida individualmente.

Os preços do sistema são calculados tendo em conta o número de contactos do utilizador ou o número de campanhas que pretende enviar, não havendo assim custo associado à utilização do gestor de formulários.

3.1.6 MailChimp

Trata-se de uma plataforma de automação de *marketing* desenvolvida pela Rocket Science Group [28] em 2001, permite gerir o público de um negócio num ambiente digital, enviar campanhas *e-mail*, anúncios e também permite a criação de formulários. Os formulários são criados através de um editor de arrastar e largar campos, que são apresentados de forma sequencial, podem ser atribuídas variáveis aos campos para utilização dos dados desses campos na mensagem ou *e-mail* de confirmação de forma automática.

As configurações do formulário permitem editar o seu aspeto gráfico, ativar *double opt-in*, utilização de *captcha* para evitar abusos por computadores, definir e configurar as notificações pós submissões do formulário e limitar a quantidade de submissões. A publicação do formulário pode ser feita diretamente para uma hiperligação, *pop-up*, ou integração em páginas ou sistemas externos. As estatísticas dos resultados das submissões fornecem dados como o número de visitas e o número de subscrições, mas também é dada a possibilidade de integrar com plataformas externas de *web analytics.*

Os preços do sistema MailChimp são apresentados tendo em conta o número de subscritores pretendidos e o número de *e-mails* a enviar, sendo o sistema de gestão de formulários de utilização gratuita.

3.1.7 Mautic

Mautic [29] é um sistema de *marketing* digital de código aberto e alojamento por parte do utilizador, possibilita a gestão de possíveis clientes, análise do histórico de cada um, envio de campanhas *e-mail*, criação de páginas, gestão de formulários e análise de resultados. O sistema de gestão de formulários permite definir os campos de forma sequencial editando as propriedades específicas de cada tipo de campo e sem configurações de apresentação. Permite também definir uma lista de ações a executar na submissão do formulário (e.g. reencaminhamento para uma página, envio de um *e-mail* de notificação, descarregamento de um ficheiro).

O Mautic também permite a criação de uma página e integração do formulário nessa página, o editor de página funciona através de um sistema de arrastar e largar elementos permitindo editar detalhadamente o aspeto de cada elemento e a sua localização na página. Esta funcionalidade separa a criação da página e seu aspeto da criação do formulário, sendo o mesmo integrado na página posteriormente.

O Mautic é um sistema gratuito, contudo é necessário ser alocado pela organização que o utiliza. Este sistema é extensível o que possibilita o desenvolvimento de funcionalidades específicas mediante a o contexto da sua utilização.

3.1.8 GrapesJS

GrapesJS [30] é um sistema gratuito de código aberto para criação de páginas *web*. O editor gráfico de arrastar e largar elementos permite dividir a página em secções com várias colunas e introduzir elementos em cada coluna. Os elementos podem ser elementos básicos encontrados em *interfaces web* como texto, imagens, vídeos, listas, ou elementos complexos como barras de navegação e formulários.

O GrapesJS permite editar as propriedades de cada elemento com bastante detalhe, desde dimensão e tipografia a decorações e animações. Este sistema apenas permite criar a página e exportá-la não tendo funcionalidades específicas de formulários nem de análise de resultados. Contudo esta ferramenta é bastante poderosa pois permite programaticamente a total customização do editor, criando um editor adaptado às necessidades do utilizador tanto em aspeto como nas funcionalidades e elementos que permite inserir.

3.1.9 Comparação

Os sistemas de gestão de formulários analisados permitem obter uma representação ampla do mercado destas soluções, nesta lista encontram-se soluções com diversos graus de maturidade, funcionalidades idênticas e funcionalidades específicas, de utilização gratuita ou paga, diferentes níveis de detalhe na criação do formulário, e aplicação em diferentes contextos. Alguns dos sistemas são aplicados especificamente no contexto de *marketing,* outros na criação de páginas ou *sites* completos, outros na criação de questionários ou apenas na gestão de formulários.

Muitas das funcionalidades de gestão dos formulários são comuns entre os sistemas, outras são específicas ao contexto de aplicação, a *Tabela 1* compara as soluções descritas segundo as suas funcionalidades.

Tabela 1 Comparação de funcionalidades de sistemas de gestão de formulários

	Jot Form	Google Forms	Wufoo	Form. io	E-goi (antigo)	Mail Chimp	Mautic	Grapes JS
Inserir/editar elemento					✓		✓	✓
Inserir/editar campo	✓	✓	✓	✓	✓	✓	✓	✓
Definir estilo global	✓	✓	✓		✓	✓	✓	✓
Definir ação de submissão	✓	✓	✓	✓	✓	✓	✓	✓
Configurar mensagem de confirmação	✓	✓	✓	✓	✓	✓	✓	
Alterar estado do formulário	✓		✓				✓	
Definir limite de submissões	✓	✓	✓		✓	✓	✓	
Ativar encriptação de campos	✓			✓				
Definir validações	✓			✓				
Configurar notificações	✓	✓	✓	✓	✓			
Definir lógica condicional	✓		✓					

	Jot Form	Googl Forms	Wufoo	Form. io	E-goi	Mail Chimp	Mautic	Grapes JS
Ativar *double opt-in*	✓		✓		✓	✓	✓	
Associar lista de contactos					✓	✓	✓	
Mapear campo da lista a campo do formulário					✓	✓	✓	
Definir traduções	✓					✓		
Inserir código próprio	✓	✓	✓		✓	✓	✓	✓
Publicar	✓		✓	✓	✓	✓	✓	✓
Criar de *template*	✓		✓	✓	✓	✓	✓	
Analisar resultados (visitas)						✓	✓	
Analisar resultados (submissões)	✓	✓	✓	✓	✓	✓	✓	
Integrar com sistema externo	✓		✓	✓	✓	✓	✓	

Todos os sistemas analisados permitem inserir os campos no formulário, tipicamente através de controlos de arrastar e soltar, onde é possível arrastar de uma lista de campos e soltá-lo num painel principal onde acontece a construção do mesmo. Alguns sistemas permitem a inserção de outros elementos gráficos para além dos campos, elementos como texto, botões, imagens, vídeos, ou elementos mais específicos e não disponíveis em todos os sistemas, como, elemento de *captcha*, grelhas de opções, carregamento de ficheiros, entre outros. Esses elementos permitem construir uma página *web* completa onde pode ser ou não integrado um formulário.

Os campos do formulário têm propriedades específicas e diferentes entre si, por exemplo, num texto simples pode ser editado o estilo de letra, cor do texto, e o texto que é efetivamente apresentado, enquanto num campo de escolha múltipla podem ser definidas as opções de escolha. Existem também propriedades que podem ser definidas de um ponto de vista global e podem ser partilhadas por elementos, como o estilo de letra, tamanho, cores do texto e cor de fundo. Todos os sistemas de formulários têm a opção de publicar o formulário, que corresponde ao seu alojamento e geração de um URL que permita o mesmo ser partilhado e acedido por utilizadores. Contudo, muitos sistemas permitem partilhar o formulário de diferentes formas, diretamente para uma rede social, através de uma janela *pop-up,* ou até exportar o código HTML criado para ser adicionado diretamente noutra página. Outra funcionalidade importante é a visualização de resultados e estatísticas acerca das submissões, em geral todos os sistemas permitem ver o número de visitas e a taxa de rejeição, porém poucos sistemas permitem ver informação detalhada sobre os visitantes.

Certas propriedades dos formulários podem ser configuradas, como o número limite de submissões, a ação de submissão, que ação chamada após o visitante submeter o formulário, a possibilidade de ativar o *double opt-in* no caso de formulários de inscrição, que faz com que o visitante receba um *e-mail* de confirmação antes de passar a subscritor. Também é possível alterar o estado do formulário com base numa certa condição, por exemplo, desativar o formulário numa determinada data ou consoante o IP do visitante.

Integrações com sistemas externos como sistemas de pagamentos podem ser adicionadas como se fossem elementos. A encriptação de campos é uma opção que quando ativada encripta o campo quando o formulário é submetido para melhorar a segurança da informação e privacidade dos utilizadores. A funcionalidade de lógica condicional permite definir condições entre os campos, o que leva a que um campo só seja apresentado caso o visitante introduza um certo valor noutro campo, criando assim formulários dinâmicos adaptáveis à informação inserida pelos seus visitantes.

Existem ainda outras funcionalidades nestes sistemas, como atribuição de variáveis aos campos do formulário que permitem a utilização dos dados inseridos nas mensagens de submissão, customização da página de submissão, envio de notificações após a submissão do formulário, entre outras. Muitos sistemas também disponibilizam as suas funcionalidades de gestão do formulário através de uma API pública.

No caso dos sistemas de criação de formulários integrados em aplicações de *marketing* digital existem funcionalidades de mapeamento dos campos do formulário com atributos das listas de contactos. Nestes sistemas existem também funcionalidades de associação de um visitante a uma lista de contactos de forma automática, e a capacidade de usar dados do subscritor nas mensagens do formulário. Os sistemas de gestão de formulários aplicados em *marketing* digital são tipicamente mais fechados e limitados em termos de personalização, pois oferecem funcionalidades de integração com as listas de contactos o que leva à limitação dos campos possíveis de incluir no formulário para diminuir as configurações necessárias. Por exemplo, num formulário de inscrição pode ser adicionado um campo do tipo *e-mail*. Associado a este campo estão validações relacionadas com o *e-mail* e a lógica de inserção de um novo *e-mail* na lista de contactos. Caso o utilizador tivesse a possibilidade de inserir um campo de entrada de texto estão funcionalidades não iriam estar disponíveis ou teriam de ser configuradas.

3.2 Abordagens de Gestão de Formulários

Após a comparação e análise de diversas soluções existentes identificam-se três abordagens principais seguidas por sistemas de gestão de formulários que condicionam diretamente as funcionalidades que estes permitem:

- Criação de formulários apenas;
- Criação de formulários e página em simultâneo;
- Criação de formulários e página separadamente com funcionalidades de integração.

Existem sistemas que apenas permitem a criação de formulários como o JotForm, Google Forms e Wufoo; sistemas que permitem criar uma página e em simultâneo inserir um formulário nessa página, como o E-goi e o GrapesJS; e sistemas que permitem criar o formulário e a página em separado e integrá-los posteriormente como o Mautic e o MailChimp. A *Tabela 2* compara as vantagens e desvantagens de cada abordagem.

Tabela 2 Vantagens e desvantagens das abordagens de criação de formulários

	Vantagens	**Desvantagens**
Criação de formulários apenas	Simplicidade de utilização	Poucas opções de configuração Impossibilidade de criar página em que o formulário é incorporado
Criação de formulários e página em simultâneo	Fácil integração do formulário na página	Complexidade Impossibilidade de criar o formulário independente da página
Criação de formulários e página separadamente	Criação de formulário independente Simplicidade na criação do formulário	Necessidade de funcionalidades de integração

Um sistema que permita a criação de formulários apenas incluindo os seus campos, mas não permitindo inserção de outros elementos tem a vantagem de ser de simples utilização, contudo não permite criar e configurar a página em que este está inserido, tornando-o um sistema muito limitado e necessário de recorrer a outros sistemas. Um sistema que permita criar os formulários e a página que este está inserido em simultâneo, apresenta muitas funcionalidades possíveis, mas é complexo de utilizar pois mistura a criação da página com a criação e gestão dos formulários, e impossibilita a criação de um formulário de forma independente. Um sistema que ofereça a possibilidade de criar formulários e a página onde este é inserido de forma independente apresenta as vantagens das abordagens anteriores tendo apenas como desvantagem a necessidade de funcionalidades extra de integração do formulário na página. A última abordagem é a utilizada no projeto desta dissertação, tendo em conta as vantagens apresentadas e visto ser a utilizada por os outros sistemas no contexto de *marketing* digital.

Um dos problemas identificados (cf. 1.2) é a falta de clareza no conceito de formulário, dado o sistema de formulários antigo do E-goi permitir em simultâneo a criação de páginas *web* e de formulários, acoplando assim as páginas com os formulários. A decisão tomada da abordagem

a utilizar vem auxiliar a resolução deste problema, pois uma nova componente da plataforma E-goi será a criação de páginas independentes dos formulários, e outra componente do sistema será a criação dos formulários através dos seus campos e opções. A componente de formulários permitirá a sua criação, através dos campos que o constituem, sem muita customização gráfica, assim como a sua publicação e análise de resultados. A componente de criação de páginas permitirá integrar um formulário criado e editar as propriedades gráficas dos seus elementos mais detalhadamente. A decisão da divisão dos componentes visa resolver o problema identificado e tem em conta a área de atuação do E-goi, *marketing* digital. Os formulários nesta área têm de ser integrados com as listas de contactos e campanhas dos utilizadores, o que os torna mais fechados no que toca à sua customização. Esta dissertação apresenta a solução da componente de formulários.

3.3 Soluções de *Web Analytics* Existentes

Nesta secção são descritas e comparadas plataformas de *web analytics* (cf. 2.3), com o objetivo de perceber se é possível utilizar uma destas plataformas para o projeto proposto e qual a mais adequada. A necessidade de uma plataforma de *web analytics* surge do detalhe necessário na análise dos resultados das submissões dos formulários. A escolha das soluções para análise tem em conta caraterísticas como as funcionalidades oferecidas, capacidade de hospedagem pelos utilizadores da plataforma e licença de utilização, obtendo uma representação diversificada destas soluções.

3.3.1 Piwik

O Piwik [31] é uma plataforma código-aberto de *web analytics*, oferece funcionalidades que permitem seguir visitas a páginas *web* e obter informações do visitante, tais como a sua localização, dispositivos de acesso, resolução de ecrã utilizada e páginas mais visitadas. Esta plataforma permite visualizar os visitantes em tempo real, métricas específicas para *e-commerce*, acompanhamento de eventos, análise de referentes, entre outras funcionalidades. Sendo o Piwik uma plataforma completamente extensível qualquer um pode implementar as suas próprias funcionalidades. Algumas das funcionalidades estendidas pelos próprios criadores ou por terceiros incluem métricas específicas para análise de multimédia, análise de formulários, relatórios customizados, análise da distribuição de *clicks* de um utilizador pela

página, entre outras. Tanto o rastreamento de visitantes como a geração de relatórios das suas visitas podem ser efetuados através dos clientes disponibilizados pelo Piwik, ou diretamente através de uma API REST.

O Piwik pode ser hospedado na *cloud*, ou hospedado num servidor próprio, tendo algumas dependências tecnológicas base necessárias estarem instaladas no servidor, como PHP e uma base de dados MySQL. Aquando da instalação são definidos os dados de administrador de acesso ao sistema, os dados de acesso à base de dados, e informação da primeira página a rastrear, que inclui o seu nome e URL. Para começar a rastrear a página é necessário inserir código Javascript nas páginas a analisar. Todos os dados e estatísticas estão disponíveis através de uma *interface* gráfica *web* ou móvel disponibilizada pelo Piwik.

3.3.2 Open Web Analytics

O Open Web Analytics [32] é um *software* de código aberto utilizado para seguir e analisar visitantes de *sites* e aplicações. Esta plataforma foi criada para permitir facilmente adicionar funcionalidades de *web analytics* a qualquer aplicação. Essas funcionalidades permitem seguir visitas e visualizações de páginas ao longo do tempo, identificar visitantes únicos e as suas respetivas visitas, analisar a distribuição de *clicks* do visitante na página, obter dados dos visitantes como a localização e idade, e ainda analisar referentes e dispositivos de visita.

Esta plataforma esta preparada para uma completa extensão e implementação de novas funcionalidades, as tecnologias de desenvolvimento incluem PHP, base de dados MySQL, utiliza Javascript nas páginas a analisar como método de seguimento de visitas, e disponibiliza uma API REST para acesso aos relatórios.

3.3.3 Mint

O Mint [33] é um sistema extensível de análise de *websites* com alojamento por parte do utilizador. Entre as funcionalidades principais permite analisar visitas, referentes, pesquisas que levaram a chegar à página, categorizar páginas por visualizações, analisar dados das subscrições, analisar dispositivos e plataformas de acesso. O Mint é um sistema proprietário que utiliza como tecnologias PHP e MySQL, e como método de seguimento de visitas são utilizadas *cookies* através de código Javascript na página a analisar.

3.3.4 Google Analytics

Serviço de *web analytics* alojado e disponibilizado pela Google [34], atualmente é o serviço mais popular de análise de estatísticas de *websites.* As funcionalidades do Google Analytics incluem rastreamento da qualidade de páginas através da análise de conversões e objetivos. Um objetivo pode ser, por exemplo, o *click* num botão ou o descarregamento de um ficheiro. A visualização do desempenho das páginas pode ser efetuada através de gráficos em funil, onde é apresentada a fonte de chegada dos visitantes através dos referentes, e termina com os seus dados de visita como duração e localização. Esta plataforma de análise oferece funcionalidades em tempo real, como saber que visitantes estão na página, a sua localização e qual a sua fonte de entrada na página. O método de rastreamento de visitantes utilizado pelo sistema faz uso da tecnologia Javascript através de código inserido em cada página a ser analisada.

3.3.5 Comparação

As plataformas de *web analytics* descritas foram escolhidas de entre várias existentes no mercado tendo em conta caraterísticas como o tipo de hospedagem, o tipo de licença e as suas funcionalidades, procurando uma boa representação das diferentes plataformas existentes. A *Tabela 3* compara as plataformas descritas segundo essas caraterísticas.

Tabela 3 Comparação de plataformas de web analytics

	Piwik	Open Web Analytics	Mint	Google Analytics
Hospedagem pelo utilizador	✓	✓	✓	
Licença aberta	✓	✓	✓	
Domínios ilimitados	✓			✓
Geolocalização	✓	✓	✓	✓
Visualizações de páginas	✓	✓	✓	✓
Visitantes	✓	✓	✓	✓
Visitantes únicos	✓	✓	✓	✓
Duração da visita	✓	✓		✓
Referentes	✓	✓	✓	✓
Conteúdo	✓	✓	✓	✓
Eventos	✓	✓	✓	✓
Tempo real	✓		✓	✓
Dispositivo	✓	✓	✓	✓
Extensível	✓	✓	✓	

As plataformas Piwik, Open Web Analytics e Mint são tanto de hospedagem pelo utilizador como de licença aberta e permitem a sua fácil extensão, portanto são ideais para casos onde é necessário implementar novas funcionalidades nas mesmas, facilitando também o seu suporte e escalabilidade, dado serem hospedadas por quem as utiliza podem ser também escaladas e alteradas como e quando necessário. Já o serviço Google Analytics é hospedado e fornecido pela organização que o desenvolve, o que simplifica a sua utilização e manutenção, e permite reduzir custos de utilização, contudo não pode ser escalado e caso seja alterado por parte dos desenvolvedores pode também levar à alteração do sistema que o utiliza.

A plataforma Piwik permite gratuitamente rastrear domínios ilimitados, enquanto a plataforma Google Analytics também permite, mas a um custo monetário a partir de um certo número de domínios.

No que toca as funcionalidades de *web analytics* todas as plataformas permitem analisar muita informação como, número de visitantes (únicos ou repetidos), a localização dos mesmos, quanto tempo ficaram na página, qual a fonte que os levou à página, dados do dispositivo de acesso e análise de eventos específicos. Salienta-se a falta de análise em tempo real no sistema Open Web Analytics, a falta da duração da visita e poucos dados acerca da geolocalização do visitante na plataforma Mint.

Dos sistemas de código aberto e hospedagem própria o Piwik é aquele oferece um mais vasto conjunto de funcionalidades, já o sistema Google Analytics também oferece as mesmas funcionalidades, mas disponibiliza-as como um serviço o que tem as suas vantagens e desvantagens conforme referido. Após a comparação destas plataformas, através de caraterísticas como as suas funcionalidades, é necessário tomar a decisão de qual utilizar no projeto proposto, essa tomada de decisão é descrita na secção 4.4 utilizando um método de auxílio ao processo de tomada de decisão.

4 Análise de Valor

Neste capítulo é apresentada uma análise de valor da solução proposta, que inclui o processo de negócio e inovação, a proposta de valor do projeto, o modelo de negócio envolvente à solução, e é ilustrada a tomada de uma decisão no âmbito do projeto utilizando um método analítico de apoio à tomada de decisão.

4.1 Processo de Negócio e Inovação

A rápida mudança do mundo que nos rodeia obriga as organizações a acompanhar essa evolução para que consigam sobreviver ao mercado, sendo necessário a criação de novas soluções para preencher as novas necessidades dos clientes. O propósito da inovação é criar valor de negócio que pode ser apresentado de diferentes formas, como novas funcionalidades num produto existente, melhorias, novos modelos de negócio, ou produtos inteiramente novos. O processo de inovação permite descobrir, criar e desenvolver novas ideias, e é um processo que tipicamente começa com muitas ideias e termina apenas com algumas a tornarem-se num produto final.

O processo de inovação pode ser dividido em três fases [35]:

- ***Fuzzy Front End* (linha da frente de inovação)**: fase inicial do processo de inovação onde são identificadas as ideias e oportunidades. É um processo tipicamente experimental e incerto;
- ***New Product Development***: transformação da oportunidade identificada num produto. É um processo orientado a objetivos, orçamentado, e com um grau de certeza crescente;
- **Comercialização**: introdução e distribuição do novo produto no mercado.

A fase da linha da frente de inovação é executada através de um processo pouco estruturado, sem terminologia definida, o que dificulta a comparação dos processos de inovação entre organizações. Tal levou Peter Koen a desenvolver o modelo *New Concept Development* (NCD) com o objetivo de definir uma linguagem e atividades concretas para a linha da frente de inovação [35].

O modelo NCD, como representado na *Figura 7*, é composto por três partes:

- O motor: elementos controlados pela organização como a sua cultura e estratégia de negócio. É importante sincronizar o processo de inovação com a estratégia de negócio e cultura da organização garantindo a criação de novos produtos com valor não só para o cliente, mas também para a organização;
- Cinco elementos da atividade de inovação (identificação da oportunidade, análise da oportunidade, geração e enriquecimento de ideias, seleção da ideia e definição do conceito);
- Fatores externos de influência da organização (canais de distribuição, lei, concorrência, clientes e ambiente político-económico).

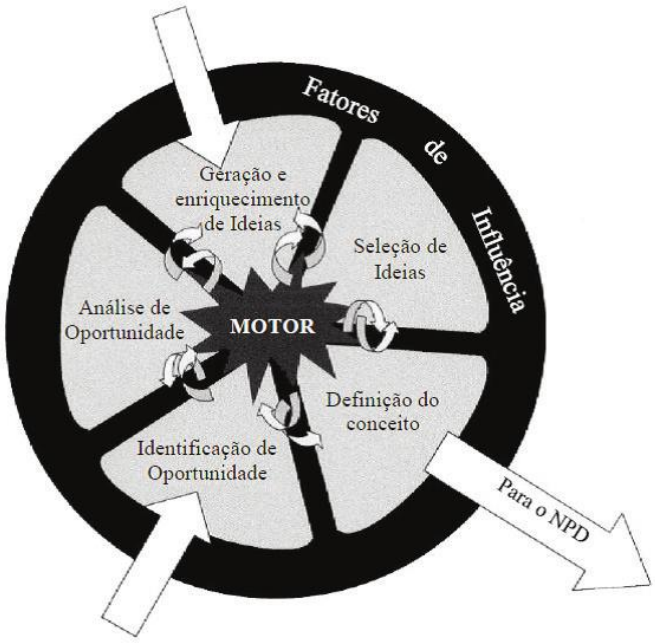

Figura 7 Modelo NCD [35]

4.1.1 Identificação da Oportunidade

Diversos fatores levam à identificação da oportunidade de um novo sistema de gestão de formulários para o produto E-goi. A análise SWOT [36] apresentada na *Figura 8* auxilia a identificação de forças, fraquezas, oportunidades e ameaças relativamente ao produto E-goi na sua totalidade, mas com destaque para pontos que levam à identificação da necessidade de criação de um novo sistema de formulários.

Analisando o ambiente contextual da organização identificam-se como forças a boa reputação no mercado de língua portuguesa, um preço bastante competitivo, uma procura constante de inovação em canais de *marketing*, entre outras. Das principais fraquezas destacam-se as tecnologias desatualizadas devido à evolução tecnológica que não foi acompanhada pelo desenvolvimento do produto, os elevados custos de manutenção devido à ausência de práticas de engenharia no desenvolvimento, a falta de métricas de análise de resultados e funcionalidades no sistema de gestão de formulários, entre outras. De uma perspetiva externa identificam-se novas oportunidades como a existência de novas tecnologias e métodos de engenharia que podem reduzir os custos de manutenção do produto, o constante aparecimento de novos canais de *marketing*, como as redes sociais, um canal ainda pouco explorado pelo produto E-goi, e ainda se identificam como oportunidades a possibilidade de reestruturar o sistema de gestão de formulários. Contudo, de um ponto de vista externo à organização, também se identificam ameaças, como alterações da regulamentação de proteção de dados, que pode levar a necessidades legais de reestruturação da estrutura de dados, baixa cota de mercado internacional onde a concorrência é dominante, aparecimento de novos concorrentes no mercado nacional, e ainda o facto dos formulários criados com o sistema obterem poucas visitas e uma forte rejeição.

Figura 8 Análise SWOT

Após a análise do ambiente contextual do negócio identifica-se a necessidade por parte das organizações do meio de captar clientes e fideliza-los ao seu produto ou serviço, responsabilidade essa que é normalmente deixada à equipa de *marketing.* Uma das formas de conectar com o mercado é através de formulários, os quais podem ter diversos propósitos, como recolher informação do visitante desse formulário adicionando-o à lista de contactos da organização e tornando-o num possível cliente, recomendar a organização e obter a opinião dos clientes da organização.

A criação e gestão do formulário está diretamente ligada à taxa de conversão deste, a criação de formulários apelativos, simples de preencher, podem levar ao seu sucesso junto dos clientes. Para que tal seja possível é necessário fornecer aos utilizadores um sistema de criação que permita criar formulários com as caraterísticas indicadas, e que seja apropriado ao público a que se destina. Para além da criação do formulário é necessário saber os resultados que este

obtém perante os visitantes para que o formulário possa ser melhorado. Com as tecnologias atuais é possível analisar dados dos visitantes para saber como estes se comportam perante a informação requisitada, é possível perceber quais os que completam o formulário e quais os que desistem. Essa informação permite perceber o sucesso ou insucesso de um formulário, contudo nada diz acerca das razões que levaram a chegar a tal conclusão. Para colmatar essa falha surge a oportunidade de apresentar métricas ao nível dos campos do formulário, identificando, por exemplo, quais os campos com mais retenção por parte dos utilizadores e qual o campo onde mais visitantes desistem. Com informação com tal granularidade é possível perceber as razões que levaram à desistência, corrigir os erros detetados, e como resultado, aumentar as conversões dos formulários.

As funcionalidades oferecidas pelo editor de formulários devem ser adequadas ao público alvo, neste caso, profissionais de *marketing.* Caso as funcionalidades estejam em demasia e necessitem de muitas competências técnicas podem levar o profissional a desistir da utilização do sistema ou caso faltem funcionalidades que este necessita o mesmo pode acontecer. No caso do sistema de gestão de formulários do produto E-goi identificou-se a falta de uma funcionalidade que permita criar condições entre campos criando assim formulários dinâmicos, dependentes do conteúdo inserido por um dado visitante.

Apesar de não diretamente relacionado com o cliente outra fraqueza do sistema de gestão de formulários atual do produto E-goi é o elevado custo de manutenção, que tem como causa a falta de práticas de engenharia usadas no seu desenvolvimento. Sendo uma parte essencial do produto é necessário um constante aperfeiçoamento, identifica-se assim a necessidade de criação de um sistema novo utilizando boas práticas de engenharia. O novo sistema de permitir uma mais rápida e eficaz manutenção do produto correspondendo de forma mais eficaz às necessidades dos clientes.

Dos fatores que levaram à identificação desta oportunidade salienta-se a falta de métricas que permitam analisar resultados dos formulários, o elevado custo de manutenção, e ainda o facto de existirem concorrentes que oferecem sistemas de gestão de formulários com funcionalidades diferenciadoras que levam os clientes a procurarem os seus sistemas.

4.1.2 Análise da Oportunidade

As equipas de *marketing* ocupam grande parte das organizações em todo o mundo, sendo um setor em crescimento onde cada vez são feitos mais investimentos. Como tal é necessário analisar o mercado subjacente a esta oportunidade para perceber se existe ou não necessidade de mercado que suporte este projeto.

Um estudo estatístico feito pelo Econsultancy [37], em 2016, revela que apenas 22% dos negócios estão satisfeitos com a sua taxa de conversão, pois o investimento feito para obter conversões tem um retorno de menos de 8%. Para além disso 53% das equipas de *marketing* gasta mais de metade do seu orçamento no aumento de conversões [38]. Sendo um mercado com grande investimento e com baixo retorno continua a ser uma grande fonte de novos clientes e as organizações continuam o seu investimento neste setor. A *Figura 9* apresenta as principais dificuldades de *marketing* das organizações, o principal desafio reside em gerar possíveis clientes e aumentar as conversões destes em clientes efetivos.

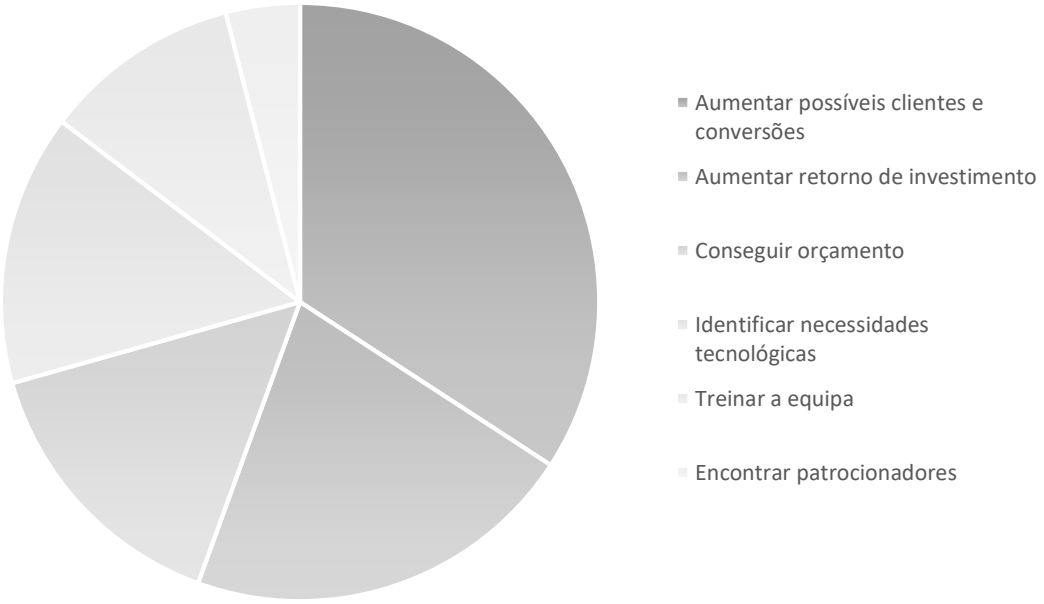

Figura 9 Distribuição das principais dificuldades das equipas de marketing [39]

As taxas de conversão variam mediante o tipo de formulário e a indústria a que se aplicam, formulários de concursos têm cerca de 28% de taxa de conversão, enquanto formulários de

inscrição têm apenas 3%. A indústria da religião é a que tem uma taxa de conversão maior, cerca de 20%, enquanto na área da saúde apenas 9% dos formulários são completados [40].

A *Figura 10* apresenta as taxas de conversão por tipo de formulário.

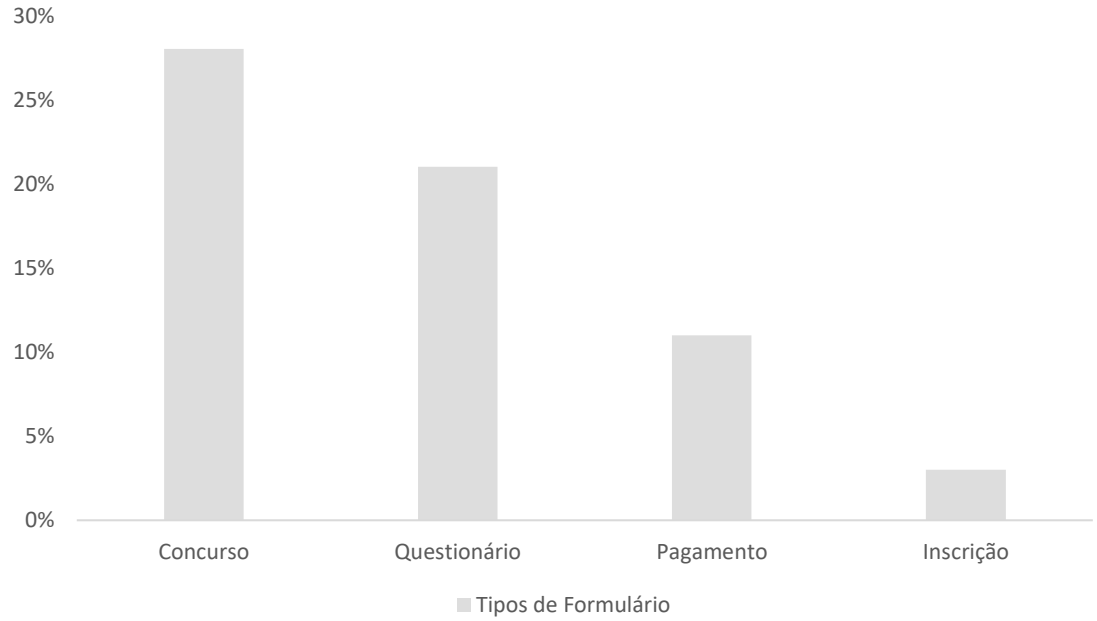

Figura 10 Percentagem de conversão por tipo de formulário [40]

Especificamente no produto E-goi sabe-se que a muitos dos seus clientes não utilizam o sistema de criação de formulários, e daqueles que usam são poucos os que têm números significativos de visitas diárias. Grandes clientes, que usam formulários e o produto E-goi no seu dia-a-dia, optam por outros sistemas para criar os seus formulários.

Conclui-se que existe uma procura significativa de conversões de clientes e os formulários são uma ferramenta essencial para atingir esse objetivo. Os dados apresentados fundamentam a necessidade de criar um novo sistema de gestão de formulários que consiga não só aumentar o seu número de utilizadores, mas também aumentar as conversões dos formulários criados, permitindo criar diversos tipos de formulários com diversas finalidades.

4.1.3 Fatores de Influência Externa

Os fatores de influência externa não são controlados pela organização, mas têm grande impacto na definição de soluções inovadoras. Fatores sociais, tecnológicos, económicos ou políticos, podem criar novas ideias ou excluir ideias de serem desenvolvidas, em seguida são sintetizados os fatores externos que levaram ao aparecimento da solução aqui proposta.

O aparecimento de novas tecnologias influencia a necessidade de criação de um novo sistema de formulários, permitindo reduzir os custos de manutenção e responder mais rapidamente às necessidades dos clientes.

Outro fator externo que influencia esta solução são os baixos resultados dos formulários criados. Tal deve-se à falta de funcionalidades que permitam analisar os resultados obtidos, como a exploração de métricas específicas dos campos do formulário, o que leva os clientes do produto E-goi a optarem por produtos concorrentes apenas para a criação de certos tipos de formulários. A entrada de novos concorrentes que tentem captar o segmento de mercado do E-goi é uma constante ameaça o que torna necessário a constante procura e desenvolvimento de novas soluções.

4.2 Proposta de Valor

Valor para o cliente é a perceção de quanto vale um produto ou serviço na sua perspetiva, e se este sente que obtém benefícios perante o que pagou [41]. Valor é criado focando no produto, nos seus benefícios, mas também em todo o ambiente cultural da organização que o desenvolve. Ao criar valor para o cliente (mais benefícios em relação a sacrifícios) a sua satisfação com a utilização do produto/serviço vai aumentar. Os benefícios incluem todas as vantagens de um produto/serviço que o cliente espera, mas também todas aquelas que o surpreendam. Fatores como a durabilidade, qualidade, funcionalidade, aspeto, são fatores que podem ser considerados benefícios. O valor percecionado é a diferença entre os benefícios obtidos do ponto de vista do cliente e o esforço que este aplicou. O esforço que um cliente aplica não inclui apenas o custo monetário, mas outros sacrifícios como consumo de tempo, energia, e esforço físico aplicado [5].

A solução apresentada nesta dissertação apresenta valor para os profissionais de *marketing*, principalmente aqueles que já utilizam o produto E-goi. A solução proposta oferece a este segmento a capacidade de criar formulários de forma simples, mas analisando os formulários ao detalhe dos campos que o constituem dando a capacidade ao profissional de corrigir problemas no seu formulário e assim aumentar as suas conversões. Este sistema é integrado diretamente com a plataforma E-goi permitindo também a utilização das listas de contactos das organizações nos formulários, não necessitando de integrações com outros sistemas. Para além disso são ainda adicionadas novas funcionalidades que trazem valor para o cliente, como a possibilidade de criação de formulários dinâmicos através de condições entre campos.

A *Tabela 4* apresenta a relação entre os benefícios e sacrifícios para o cliente ao utilizar este produto.

Tabela 4 Benefícios e sacrifícios da solução proposta

Domínio / Âmbito	**Produto / Serviço**	**Relacionamento**
Benefício	Métricas específicas para análise de formulários Formulários dinâmicos	Eficiência Adaptação
Sacrifício	Custo do sistema Esforço de criação dos formulários Esforço de integração com a sua página	Tempo investido

Numa perspetiva longitudinal de valor o cliente sabe à partida as vantagens de utilização deste sistema e o valor que irá trazer ao seu negócio. O momento de criação dos formulários é quando o cliente usufrui da maioria das funcionalidades, desde o mapeamento de campos com os dados das suas listas de contactos à geração de formulários dinâmicos. Após a publicação do

formulário o cliente pode verificar como este está a suceder perante os respetivos visitantes. Podem ser feitas alterações mediante os dados recolhidos que ajudam o profissional a tomar decisões sobre como mudar o seu formulário para obter mais conversões. Por fim espera-se que os clientes se apercebam dos ganhos adquiridos com o novo sistema e continuem a utilizá-lo para os mais diversos propósitos.

4.3 Modelo de Negócio Canvas

Após uma análise à proposta de valor apresentada por este negócio é necessário sintetizar determinada informação, como os segmentos de mercado abrangidos por esta solução, qual o valor que lhes será entregue, como chegar até eles e quais as competências necessárias. O modelo Canvas [42] é uma ferramenta que permite definir modelos de negócio para qualquer projeto, sintetizando essa informação. Este modelo está dividido em parceiros-chave, atividades-chave, recursos-chave, proposta de valor, relação com os clientes, segmentos de marcado, custos e fontes de retorno.

A *Figura 11* ilustra o modelo de negócio Canvas elaborado para o negócio envolvente a esta dissertação.

Identifica-se como atividades principais o desenvolvimento e manutenção do sistema, que inclui os testes e correções necessárias. Os recursos necessários são máquinas de desenvolvimento, as aplicações necessárias e os desenvolvedores. Como proposta de valor identifica-se a possibilidade de criar e gerir formulários, a melhoria na análise de resultados e introdução de novas funcionalidades como criação de formulários dinâmicos. A relação com o cliente pretende-se que seja individualizada através do suporte e mantida durante um longo prazo para que este uso o produto durante o maior tempo possível. Os canais utilizados para distribuir o produto serão a plataforma E-goi, anúncios e *webinars* ou outras formas de divulgação realizadas pela organização. Os segmentos de mercado são equipas de *marketing* no mercado nacional ou internacional ou pessoas com interesse em crescer os clientes dos seus negócios e produtos, principalmente os que já utilizam o produto E-goi. Os custos do sistema envolvem custos de recursos humanos, máquinas de desenvolvimento, *software, marketing* e distribuição. Finalmente o retorno será obtido através do custo de utilização da plataforma e manutenção para certos utilizadores.

<table>
<tr>
<td rowspan="2">Parceiros-chave</td>
<td>Atividades-chave

Desenvolvimento e manutenção do sistema.</td>
<td rowspan="2">Proposta de valor

Criação de formulários;

Melhoria na análise de resultados;

Introdução de novas funcionalidades.</td>
<td>Relação com os clientes

Individual;

Longo prazo.</td>
<td rowspan="2">Segmentos de mercado

Equipas de marketing mercado nacional;

Equipas de marketing mercado internacional.</td>
</tr>
<tr>
<td>Recursos-chave

Recursos humanos;

Software;

Máquinas de desenvolvimento.</td>
<td>Canais

Plataforma online (E-goi);

Anúncios;

Webinar.</td>
</tr>
<tr>
<td colspan="3">Custos

Recursos humanos;

Máquinas;

Marketing;

Software.</td>
<td colspan="2">Fontes de retorno

Custo do sistema;

Manutenção.</td>
</tr>
</table>

Figura 11 Modelo de negócio Canvas

4.4 Analytic Hierarchy Process (AHP)

O ser humano toma decisões no seu dia-a-dia, muitas das quais são tomadas de forma instintiva sem existir uma avaliação cuidada. No entanto certas decisões podem levar a grandes implicações, o que torna necessário um processo de tomada de decisão que procure a melhor solução à luz de um conjunto de critérios.

O *Analytic Hierarchy Process* (AHP) é um método de apoio à tomada de decisão, de forma racional e matemática, criado por Thomas L. Saaty na década de 70 e é desde então é um dos métodos de apoio à tomada de decisão mais usados [43]. Este método é composto por três etapas:

- **Construção da hierarquia**: constrói-se uma estrutura em árvore com três níveis, na raiz é colocado o objetivo da decisão, no segundo nível da hierarquia são colocados os critérios, e no último nível são colocadas todas as alternativas possíveis. Esta estrutura permite representar de uma forma visual o problema facilitando a sua compreensão;
- **Definição de prioridades**: definir as prioridades entre cada par de critérios, e entre critérios e alternativas;
- **Consistência**: dado as prioridades serem definidas de forma subjetiva e baseada na opinião é necessário calcular o índice de consistência e verificar se existe consistência suficiente para a decisão ser validada.

Um aspeto importante da solução proposta é a capacidade de análise da interação dos visitantes com o formulário. O processo de medição, recolha e análise dessa informação em ambiente *web* é conhecido como *web analytics*. Existem diversas plataformas que permitem recolher essa informação, no âmbito desta dissertação o método AHP é utilizado de forma ilustrativa para a tomada de decisão sobre qual plataforma de *web analytics* utilizar. De entre as plataformas analisadas na secção 3.3 escolheu-se como alternativas de comparação a plataforma Piwik [44] e o Google Anaylitcs [34], contudo existe também a opção de desenvolvimento de uma plataforma personalizada para o projeto proposto, portanto essa alternativa também é considerada nesta decisão. Os critérios importantes na escolha de qual plataforma utilizar são o tempo de desenvolvimento, as funcionalidades e a suportabilidade, sendo esses os critérios utilizados na comparação. A figura *Figura 12* ilustra o modelo hierárquico que representa a estrutura de decisão criada.

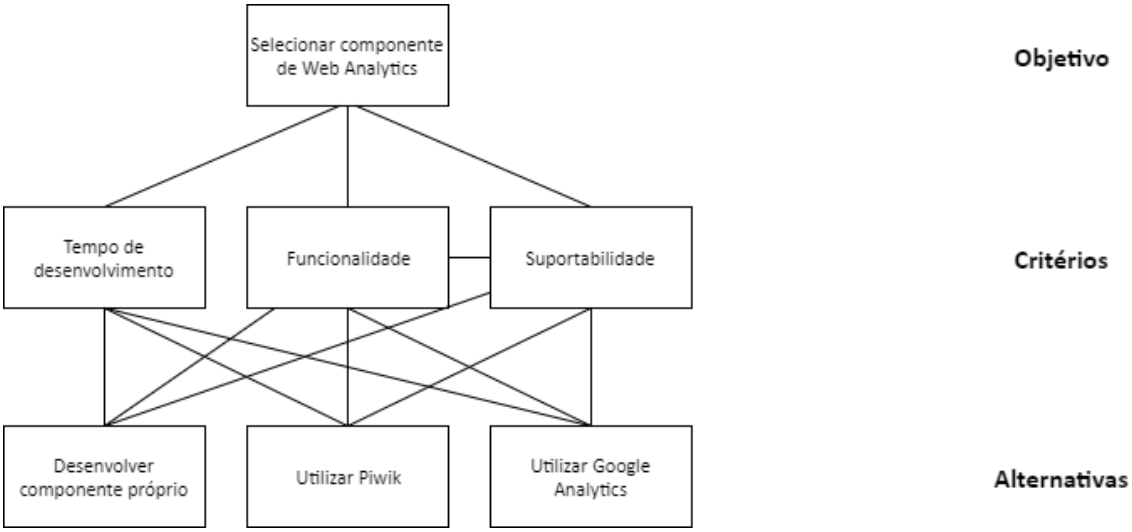

Figura 12 Estrutura hierárquica de decisão

Após a definição da hierarquia de decisão que permite modelar o problema através dos critérios e alternativas, é necessário estabelecer a importância relativa entre cada critério de decisão, e para tal utilizou-se a escala de Saaty [45]. Para atribuição de um peso a cada critério calculou-se a prioridade média local (PML), fazendo uma média das linhas da tabela de comparação normalizada. A *Tabela 5* apresenta a comparação de prioridades entre cada critério.

Tabela 5 Tabela de comparação de prioridades entre cada critério

Prioridade	**Tempo de desenvolvimento**	**Funcionalidade**	**Suportabilidade**	**PML**
Tempo de desenvolvimento	1	0.25	0.5	0.14
Funcionalidade	4	1	3	0.62
Suportabilidade	2	0.333	1	0.24

Na etapa seguinte é necessário estabelecer as prioridades entre cada alternativa à luz de cada critério. Para tal é utilizada novamente a escala de Saaty e calculada a PML para cada alternativa, os resultados são apresentados na *Tabela 6*, *Tabela 7* e *Tabela 8*.

Tabela 6 Tabela de julgamento das prioridades à luz do critério Tempo

Tempo de desenvolvimento	Componente próprio	Piwik	Google Analytics	PML
Componente próprio	1	0.125	0.125	0.06
Piwik	8	1	1	0.47
Google Analytics	9	1	1	0.47

Tabela 7 Tabela de julgamento das prioridades à luz do critério Funcionalidade

Funcionalidade	Componente próprio	Piwik	Google Analytics	PML
Componente próprio	1	0.143	0.143	0.07
Piwik	7	1	2	0.57
Google Analytics	7	0.5	1	0.36

Tabela 8 Tabela de julgamento das prioridades à luz do critério Suportabilidade

Suportabilidade	Componente próprio	Piwik	Google Analytics	PML
Componente próprio	1	2	7	0.62
Piwik	0.5	1	3	0.29
Google Analytics	0.143	0.333	1	0.09

Quando opiniões se contradizem na tabela de prioridades pode levar a inconsistência nos resultados, por esse motivo é necessário verificar a consistência das opiniões calculando o Índice de Consistência (IC) e a Relação de Consistência (RC). O IC pode ser calculado segundo a seguinte fórmula, onde $\boldsymbol{n}$ representa o número de alternativas:

$$\boldsymbol{IC} = \frac{(\boldsymbol{\lambda_{max}} - \boldsymbol{n})}{(\boldsymbol{n} - \mathbf{1})})$$

Para calcular o RC basta dividir o IC pela Inconsistência Aleatória Média (IAM), um valor aleatório, definido pelo método, apresentado na *Tabela 9* segundo o número de alternativas.

Tabela 9 Inconsistência Aleatória Média

Número de alternativas	**1**	**2**	**3**	**4**	**5**
IAM	0.00	0.00	0.58	0.90	1.12

O método AHP exige que o RC seja menor ou igual a 0.1 para que exista consistência entre as opiniões. O IC e o RC estão apresentados na tabela seguinte para cada um dos critérios. Como é possível verificar na *Tabela 10* todos são menores que 0.1 o que permite tirar conclusões com as prioridades definidas.

Tabela 10 Índice de consistência e relação de consistência para cada critério

Critérios	**IC**	**RC**
Tempo de desenvolvimento	0.007	0.01
Funcionalidade	0.03	0.06
Suportabilidade	0.01	0.02

Por fim calcula-se o vetor de prioridade global combinando a tabela de comparação das alternativas com a tabela de comparação dos critérios como apresentado:

$$PG_{Próprio} = 0{,}14 \times 0.06 + 0.62 \times 0.07 + 0.24 \times 0.62 = 0.20$$

$$PG_{Piwik} = 0{,}14 \times 0.47 + 0.62 \times 0.57 + 0.24 \times 0.29 = 0.49$$

$$PG_{GoogleAnalytics} = 0{,}14 \times 0.47 + 0.62 \times 0.36 + 0.24 \times 0.09 = 0.31$$

De acordo com os resultados obtidos a melhor alternativa é a utilização da plataforma Piwik, pois é a que tem maior prioridade global. Para além disso também foram obtidos bons resultados para esta alternativa nos critérios de tempo de desenvolvimento e funcionalidade, apenas obtendo um resultado fraco no critério de suportabilidade, onde a alternativa de desenvolvimento de um componente próprio se destacou.

Conclui-se, através utilização do método AHP, que a utilização do Piwik é a melhor opção de plataforma de *web analytics* tendo em conta os critérios e as restantes alternativas definidas.

5 Desenvolvimento da Solução

Este capítulo aborda o desenvolvimento técnico da solução de *software* proposta. São apresentados os requisitos do sistema, a modelação do negócio, o desenho arquitetural, o desenho detalhado, alguns detalhes de implementação e os testes de *software* realizados.

5.1 Análise de Requisitos

A engenharia de requisitos é uma disciplina da engenharia de *software* que estuda a criação, a análise, o desenvolvimento e a manutenção das capacidades e condições que o sistema deve satisfazer para resolver o problema [46]. Esta fase do desenvolvimento de *software* envolve estudar as necessidades das partes interessadas no sistema com o objetivo de compreender e definir corretamente os requisitos garantindo a qualidade do produto final.

Os requisitos identificados podem ser funcionais, que identificam funcionalidades da aplicação, ou não funcionais, que identificam atributos de qualidade que devem ser seguidos, a sua identificação é essencial para o desenvolvimento da solução, pois todas as fases posteriores dependem da mesma.

5.1.1 Partes Interessadas (*stakeholders*)

As partes interessadas são todas as entidades que são influenciadas, direta ou indiretamente, pelo desenvolvimento do sistema. A identificação dos *stakeholders* é importante pois são diferentes pontos de vista que devem ser considerados aquando da especificação e desenvolvimento do sistema.

Em seguida são apresentadas as partes interessadas do sistema proposto:

- **Organização:** organização proprietária do produto, pretende o correto funcionamento do sistema de forma a promover aos seus utilizadores a melhor experiência possível;
- **Profissionais de *marketing*:** pretendem criar os seus formulários de forma simples e rápida, mas com toda a informação necessária. Os utilizadores do E-goi são tipicamente profissionais de *marketing* das organizações que pretendem expandir as suas listas de contactos e clientes. Um utilizador também pode ser qualquer indivíduo que tenha interesse em expandir o seu negócio, através das funcionalidades da plataforma para

comunicação. Neste contexto todos os utilizadores são referidos como "profissionais de *marketing*" dado o interesse comum entre todos;

- **Visitantes:** pretendem submeter rapidamente os formulários e que estes sejam apelativos, sigam boas práticas de desenho, sejam seguros e fiáveis.

5.1.2 Atores

Os atores são os utilizadores efetivos do sistema, que executam ações sobre este, e podem desempenhar um papel na sua utilização, sendo eles também uma das partes interessadas. Neste sistema existem dois tipos de atores:

- **Profissionais de *marketing*,** que criam e publicam os seus formulários. Os principais utilizadores do E-goi são indivíduos que pretendam crescer as suas listas de contactos e o número de clientes dos seus negócios, principalmente equipas e profissionais de *marketing* das organizações;
- **Visitantes**, que preenchem e submetem os formulários. Podem ser contactos dos profissionais, ou ter acesso ao formulário a partir de alguma publicação.

5.1.3 Requisitos Funcionais

Os requisitos funcionais são ações que são executadas pelos atores do sistema. O projeto seguiu uma metodologia de desenvolvimento ágil [47], e como forma de especificar os requisitos foram utilizadas *user stories* (US) e critérios de aceitação.

A *user story* descreve uma ação de um ator do sistema em conjunto com o seu objetivo e valor produzido. Os critérios de aceitação contemplam um conjunto de requisitos funcionais, que captam tarefas e processos de negócios, e não funcionais, que captam condições como usabilidade, desempenho, suportabilidade, entre outras. Aplicando o acrónimo SMART [48] na engenharia de requisitos de *software*, as *user stories* devem ser específicas, mensuráveis, atribuídas a uma pessoa, realistas e limitadas no tempo. Uma *user story* que não cumpra estas caraterísticas tem fortes possibilidades de não ser concluída.

Os requisitos funcionais são apresentados nas próximas subsecções.

5.1.3.1 US01 - Criar formulário

Como profissional de *marketing* quero criar um formulário para recolher informação para as minhas listas de contactos.

Critérios de aceitação:

- É necessário inserir o nome do formulário e uma lista de contactos associada ao mesmo;
- Permite criar formulários de diversos tipos (inscrição, recomendação e questionário), onde cada um permite inserir os campos adequados.
    - Os formulários de inscrição registam os novos subscritores/contactos (podem ser inseridos campos como *e-mail*, nome, morada, telefone, etc);
    - Os formulários de recomendação enviam pedidos de inscrição para os contactos recomendados (podem ser inseridos campos como *e-mail*, nome, morada, telefone, etc);
    - Os questionários efetuam questões e podem ser classificados quanto aos seus resultados (podem ser inseridos campos como texto simples, escolha múltipla, grelha de opções, etc).

5.1.3.2 US02 - Inserir campo

Como profissional de *marketing* quero inserir um campo no formulário para definir a informação a ser preenchida pelos visitantes.

Critérios de aceitação:

- Permite inserir campos de diversos tipos (e.g. entrada de texto, escolha múltipla, *e-mail*, telefone, etc);
- Permite definir propriedades genéricas dos campos (obrigatoriedade, texto, valor por defeito e mensagens de validação e ajuda);
- Permite definir propriedades específicas dependendo do seu tipo (e.g. num campo do tipo texto é possível configurar o próprio texto apresentado, num campo de escolha múltipla é possível definir as opções, etc);

- É possível editar propriedades relacionadas com o estilo do campo (tipo de letra, tamanho, bordas, margens, cores e alinhamento).

5.1.3.3 US03 - Definir estilo

Como profissional de *marketing* quero definir o estilo do formulário através das suas propriedades (tipo de letra, cores, tamanho, bordas, margens e alinhamento), para personalizar a aparência do formulário.

Critérios de aceitação:

- O estilo definido é aplicado ao formulário;
- O estilo definido é utilizado como base para os campos do formulário.

5.1.3.4 US04 - Publicar formulário

Como profissional de *marketing* quero publicar o formulário, para torna-lo acessível aos visitantes.

Critérios de aceitação:

- O formulário pode ser publicado como hiperligação, *pop-up*, exportado diretamente para HTML ou integrado com sistemas externos (redes sociais);
- O formulário é disponibilizado publicamente através de um URL (tendo em conta as limitações definidas).

5.1.3.5 US05 - Definir limitações do formulário

Como profissional de *marketing* quero definir limitações do formulário, para que o estado do mesmo (ativo/inativo) seja alterado conforme os critérios definidos.

Critérios de aceitação:

- É possível definir um limite máximo de visitas, quando o limite for atingido o formulário é desativado;

- É possível definir um limite máximo de submissões, quando o limite for atingido o formulário é desativado;
- É possível limitar o formulário a um intervalo de datas (e.g. data em que o formulário passa a ativo e data em que passa a inativo);
- É possível limitar o formulário a uma submissão por IP;
- É possível limitar o acesso ao formulário apenas a contactos da lista associada ao formulário;
- Valida os limites introduzidos.

5.1.3.6 US06 - Definir ação de submissão

Como profissional de *marketing* quero definir uma ou várias ações, que ocorrem quando um formulário é submetido para os dados submetidos serem tratados.

Critérios de aceitação:

- As ações são executadas quando um visitante submete um formulário (é efetuado o pedido a um URL passando os dados submetidos);
- Permite inserir URL da ação;
- Valida URL da ação.

5.1.3.7 US07 - Configurar mensagem de submissão

Como profissional de *marketing* quero configurar a mensagem de submissão que é apresentada para o visitante visualizar a mensagem após a submissão do formulário.

Critérios de aceitação:

- Quando um visitante submete um formulário é apresentada a mensagem definida.

5.1.3.8 US08 - Definir lógica condicional entre campos

Como profissional de *marketing* quero definir lógica condicional entre campos para os formulários variarem conforme as respostas dos visitantes, criando assim formulários dinâmicos.

Critérios de aceitação:

- É possível definir um campo, um valor a comparar com o valor do campo, uma operação de comparação, e um outro campo que é ativado ou desativado consoante o resultado da comparação.

5.1.3.9 US09 - Mapear campo da lista de contactos com campo do formulário

Como profissional de *marketing* quero mapear um campo do formulário com um campo da lista de contactos E-goi associado ao formulário para automaticamente atualizar os dados da lista através do formulário.

Critérios de aceitação:

- Quando é submetido um formulário de inscrição o novo contacto é automaticamente adicionado à lista conforme o mapeamento de campos definido.

5.1.3.10 US10 - Submeter formulário

Como visitante quero submeter o formulário, após o seu preenchimento, para as minhas respostas ficarem registadas e as ações associadas serem executadas.

Critérios de aceitação:

- O formulário é disponibilizado aos visitantes através de um URL público (qualquer utilizador na Internet pode aceder a não ser que tenham sido definidas limitações);
- Valida os dados introduzidos;
- Completa automaticamente os campos quando o formulário puder ser submetido mais de uma vez;
- Após a submissão do formulário o visitante é reencaminhado para uma página de sucesso;
- Após a submissão do formulário são executadas as ações necessárias:
    - Num formulário de inscrição os dados submetidos são inseridos como contacto na lista associada;

- Num formulário de recomendação são enviados formulários de inscrição para os possíveis contactos recomendados;
- Outras ações definidas pelo profissional.

5.1.3.11 US11 - Ver estatísticas de visitas ao formulário

Como profissional de *marketing* quero ver estatísticas do formulário para perceber as tendências das visitas.

Critérios de aceitação:

- São apresentadas estatísticas relacionadas com as visitas do formulário, como localização de acesso, dispositivos associados, data e duração.

5.1.3.12 US12 - Analisar resultados dos formulários

Como profissional de *marketing* quero analisar os resultados dos formulários para perceber as respostas e submissões dos visitantes.

Critérios de aceitação:

- É possível visualizar as respostas dos visitantes campo a campo;
- No caso de campos com opções limitadas é possível visualizar as percentagens de resposta para cada opção;
- É possível visualizar estatísticas específicas dos campos (e.g. campos menos preenchidos, campos com mais retenção, …).

5.1.3.13 US13 - Inserir código próprio

Como profissional de *marketing* quero inserir o meu próprio código no código existente do formulário, para ser possível efetuar uma personalização detalhada.

Critérios de aceitação:

- O código inserido é apresentado/executado no formulário.

5.1.3.14 US14 - Ativar/desativar notificações

Como profissional de *marketing* quero ativar/desativar notificações de submissão de formulários para notificar outros profissionais quando um formulário for submetido, e configurar os utilizadores que recebem notificações e a mensagem de notificação.

Critérios de aceitação:

- É possível configurar, através do *e-mail,* que profissionais recebem as notificações;
- Quando as notificações estão ativadas e é submetido um formulário os profissionais definidos recebem um *e-mail*.

5.1.4 Outros Requisitos

Alguns requisitos não funcionais não se relacionam diretamente com as *user stories*, pois são atributos de qualidade utilizados para desrever comportamentos que este deve seguir. Estes requisitos para este projeto são definidos com base no modelo FURPS+ [49] e são apresentados na lista seguinte:

- **Funcionalidade**:
    - Nenhuma ação deve ser destrutiva;
    - Funcionalidade do sistema de gestão de formulários devem ser oferecidas através de uma API;
    - A API deve devolver o código de erro adequado ao tipo de erro.
- **Usabilidade**:
    - A interação entre os utilizadores e o sistema deve ser simples e adaptada à situação em causa.
- **Confiabilidade**:
    - O sistema deve prevenir perda de dados fazendo gravações automáticas;
    - Os resultados devem estar sempre disponíveis para os utilizadores;
    - Os dados dos campos dos formulários devem ser encriptados.
- **Desempenho**:

- A utilização do sistema por parte do cliente não deve ter impacto noutros clientes;
- O acesso a resultados deve ser feito em tempo útil para o utilizador.

- **Suportabilidade**:
  - O sistema executa em ambiente *web*;
  - A estrutura de classes deve ser desenvolvida de modo a permitir facilmente adicionar novas funcionalidades (cf. 5.4.3 e 5.5.2).

- **Outros**:
  - **Limitações de desenho**:
    - O desenho do sistema deve seguir padrões de desenho de *software*;
    - A estrutura de dados deve ser desnormalizada para otimizar processamento (cf. 5.4.3 e 5.5.2);
    - O sistema deve ser integrado na plataforma E-goi.
  - **Limitações de implementação**:
    - Utilização das tecnologias Zend Framework 2, PHP 5.6, MySQL/MariaDB e ActiveMQ;
    - Deve ser usada a tecnologia Doctrine ORM para mapeamento objeto-relacional através dos serviços já existentes;
    - O sistema deve maximizar a reutilização de componentes já existentes.

5.2 Modelação do Domínio

O modelo de domínio é um artefacto que permite fornecer uma representação visual do negócio através da linguagem UML, nele são representados os conceitos de negócio e as relações entre eles [50]. A *Figura 13* apresenta o diagrama de modelo de domínio da área de negócio do sistema proposto.

O sistema proposto está incluído na plataforma E-goi tendo algumas dependências contextuais desta. A plataforma E-goi permite aos seus utilizadores gerir a sua lista de contactos, através dos contactos presentes na lista e os dados/campos de cada contacto. As listas são utilizadas para

enviar campanhas e formulários para os contactos através de múltiplos canais. O componente de gestão de formulários permite criar e publicar formulários configurando os campos e dados do formulário, assim como o tipo de publicação e as suas limitações. O formulário pertence a uma conta, a qual pode ser partilhada por diversos utilizadores, uma lista de contactos dessa conta deve estar associada ao formulário para adicionar conteúdo à lista mediante as respostas do formulário. Cada visitante do formulário pode efetuar a sua submissão após preencher os seus campos obrigatórios. Depois de publicar o formulário o profissional pode visualizar estatísticas das visitas e submissões dos formulários.

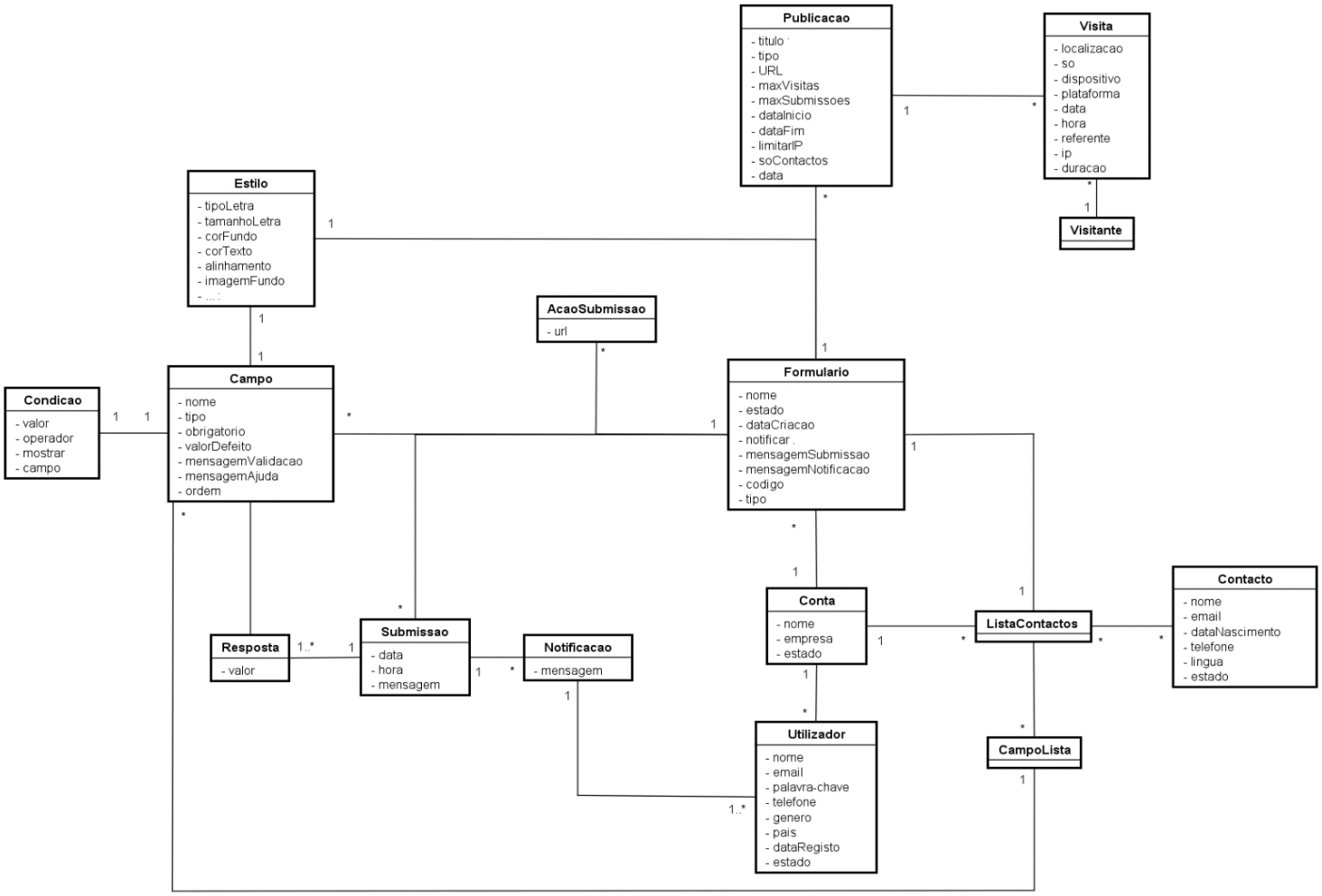

Figura 13 Diagrama de modelo de domínio

O glossário do domínio descreve cada entidade e conceito do negócio, com o objetivo de eliminar ambiguidades e definir claramente o significado de cada conceito no contexto inserido. Na *Tabela 11* são descritas as entidades de negócio de forma detalhada.

Tabela 11 Glossário

Entidade	Descrição
Campo	Espaço do formulário onde podem ser introduzidos dados (e.g. entrada de texto, escolha múltipla, seleção, *e-mail*, telefone …)
Cliente	O mesmo que utilizador, ou seja, aquele que faz uso do sistema (este termo é mais utilizado no contexto de acesso aos dados de um utilizador)
Conta	Espaço único de um ou vários utilizadores do produto E-goi onde são associados os seus movimentos, dados e ações dentro do sistema
Contacto	Integrante da rede de possíveis clientes de uma conta
Conversão	Ação que ocorre quando um possível cliente passa a cliente
Entrada	Valor de um campo de uma submissão do formulário (o mesmo que Resposta)
Estado	Circunstância do formulário, pode estar ativo, inativo ou eliminado
Estilo	Conjunto de propriedades relacionadas com a aparência de objetos (e.g. cores, tipos de letra, bordas, etc)
Evento	Ação possível de executar dentro de uma página (e.g. *click* numa hiperligação, descarregamento de um ficheiro, etc)
Formulário	Ferramenta para requisitar informação através de um conjunto de campos onde podem ser preenchidos dados. Num ambiente web um formulário permite ao utilizador inserir dados para estes serem enviados e processados por um servidor
Inscrição	Inscrição ou subscrição é um tipo de formulário com campos e comportamentos específicos para um utilizador subscrever os serviços de uma organização
Notificação	Mensagem de aviso quando ocorre uma submissão do formulário, configurada através da mensagem de notificação e dos contactos para o qual vai ser enviada

Propriedade	Configuração específica de um campo (e.g. num campo do tipo texto pode ser definido o tipo e tamanho da letra)
Publicação	Forma de distribuição do formulário, pode ser partilhado de diversas formas: hiperligação, *pop-up*, integrações com sistemas externos, entre outras
Questionário	Tipo de formulário composto por uma série de questões onde tipicamente é atribuída uma classificação após o seu preenchimento
Recomendação	Tipo de formulário que tem como objetivo obter recomendações de possíveis clientes para subscrever ao serviço da organização
Resposta	Entrada única de um formulário (o mesmo que Entrada)
Submissão	Ação que ocorre quando um formulário é enviado, esta é composta pelo conjunto de respostas inseridas naquele formulário
Subscritor	Possível cliente de uma conta que segue as suas notícias (o mesmo que Contacto)
Tema	Conjunto de estilos de formulário pré-definidos e reutilizáveis
Tipo de formulário	Formulário específico para uma certa atividade com campos apropriados para essa atividade
Utilizador	Pessoa que faz uso do sistema, no contexto desta dissertação pode ser a pessoa que usa o sistema de gestão de formulários (geralmente um profissional de *marketing*) ou a pessoa que interage diretamente com o formulário (visitante)
Visita	Ação que ocorre quando um utilizador entra num formulário
Visitante	Pessoa que visita um formulário

5.3 Design Arquitetural

Na presente secção é descrita a arquitetura da solução desenvolvida, considerados os requisitos descritos na secção 5.1. Como ponto de partida para a arquitetura da solução é explicada a arquitetura da plataforma E-goi.

5.3.1 Arquitetura da Plataforma E-goi

Sendo um dos requisitos principais a integração com a plataforma E-goi, é importante perceber e analisar a arquitetura desta plataforma para escolher o caminho arquitetural a seguir, as suas limitações, e a mais valia da solução proposta.

O E-goi é um sistema com mais de uma década de existência, ao longo da qual ocorreu inovação tecnológica e principalmente de metodologias de desenvolvimento. A equipa responsável pela plataforma tentou acompanhar essa inovação e adaptar as partes possíveis do sistema. Contudo, grande parte do núcleo da plataforma ainda se encontra desenvolvido segundo práticas que tornam difícil a manutenção do sistema por parte dos desenvolvedores.

O diagrama de componentes da *Figura 14* representa parte da vista lógica da plataforma E-goi. Este diagrama ilustra a vista do sistema, o qual já sofreu diversas alterações ao longo dos anos da sua manutenção. Cada componente do diagrama representa uma parte do sistema, a arquitetura ou estrutura desse componente é independente dos outros e pode utilizar variadas tecnologias. De seguida são descritas as responsabilidades e finalidades de cada componente:

- ***bo, bo3:*** estes componentes foram os primeiros a existir e são a base de todo o E-goi. Durante bastante tempo foram os únicos e continham toda a lógica da aplicação. Seguem maioritariamente um paradigma procedimental e não seguem padrões de desenho ou boas práticas. Estão divididos numa lista de ficheiros ou *scripts*, com pouca lógica na sua nomenclatura e uma reduzida separação em pastas. Cada ficheiro contém responsabilidades de visualização, lógica de negócio, serviços e persistência de dados, misturando a utilização de tecnologias como PHP, HTML, CSS, Javascript, SQL, entre outras. Estes componentes tratam aspetos de negócio como: contas, utilizadores, campanhas, canais das campanhas, envio das campanhas através dos canais (*e-mail, sms, …*), formulários, relatórios, automatismos, pagamentos, e todas as restantes funcionalidades do E-goi. A diferença entre os componentes *bo* e *bo3* é que o primeiro executa apenas numa máquina com lógica comum a todos os clientes e trata da gestão de pedidos para outras máquinas onde executa o *bo3.* O sistema de formulários encontra-se desenvolvido em alguns ficheiros destes componentes, o modo como está desenvolvido torna bastante difícil a sua manutenção, o que foi um dos problemas que levou à existência desta dissertação. Futuramente os desenvolvedores pretendem que

estes componentes deixem de existir e sejam substituídos por outros que sigam boas práticas arquiteturais e de desenho;

- ***www, www3:*** estes componentes são similares aos componentes *bo*, contudo diferem em que a maior parte da lógica existente neles trata de ações públicas, ou seja, ações que não necessitam de autenticação. Por exemplo, quando um formulário é publicado qualquer utilizador da Internet pode aceder ao mesmo através do seu URL, não necessitando de autenticação. É no *www3* que existe também uma API pública utilizada por outros desenvolvedores;
- ***services*****:** o *services* é um componente recente que tenta migrar a lógica de serviços e gestão da estrutura de dados (*backend*) existente no *bo,* seguindo boas práticas de desenvolvimento. Atualmente parte da lógica de criação de conta, gestão de utilizadores, criação e envio de campanhas, geração de relatórios, entre outras, já se encontra neste componente. Os módulos do *services* encontram-se desenvolvidos segundo uma variação do padrão arquitetural MVC através da tecnologia Zend Framework. Este utiliza a tecnologia ActiveMQ para gestão de filas e a tecnologia MySQL como sistema de gestão de base de dados;
- ***bo-ui:*** este componente pretende executar apenas lógica de visualização (*frontend*). Em conjunto com o *services* pretende desacoplar a lógica de visualização da lógica de serviço e gestão da estrutura de dados existem no *bo.* O *bo-ui* encontra-se desenvolvido em AngularJS e atualmente contém a *interface* de criação e envio de campanhas, assim como o ecrã principal;
- ***egoi-apps:*** este componente também é recente e é responsável por conter as aplicações gráficas de maior complexidade e que possam ser, de certa forma, separadas das funcionalidades principais do e-goi, como é o caso da criação de automatismos (*autobot-builder*). Atualmente as aplicações existentes neste componente encontram-se desenvolvidas com a tecnologia Angular e segundo as implicações arquiteturais induzidas por a mesma;
- ***egoi-public:*** este componente pretende substituir as funcionalidades de camada pública existente no *www* e *www3.* Encontra-se desenvolvido em Zend Framework seguindo uma arquitetura semelhante à do *services*, contudo ainda existe pouca lógica migrada para este componente, pois apenas a lógica de camada pública do canal *Web Push* se encontra aqui desenvolvida.

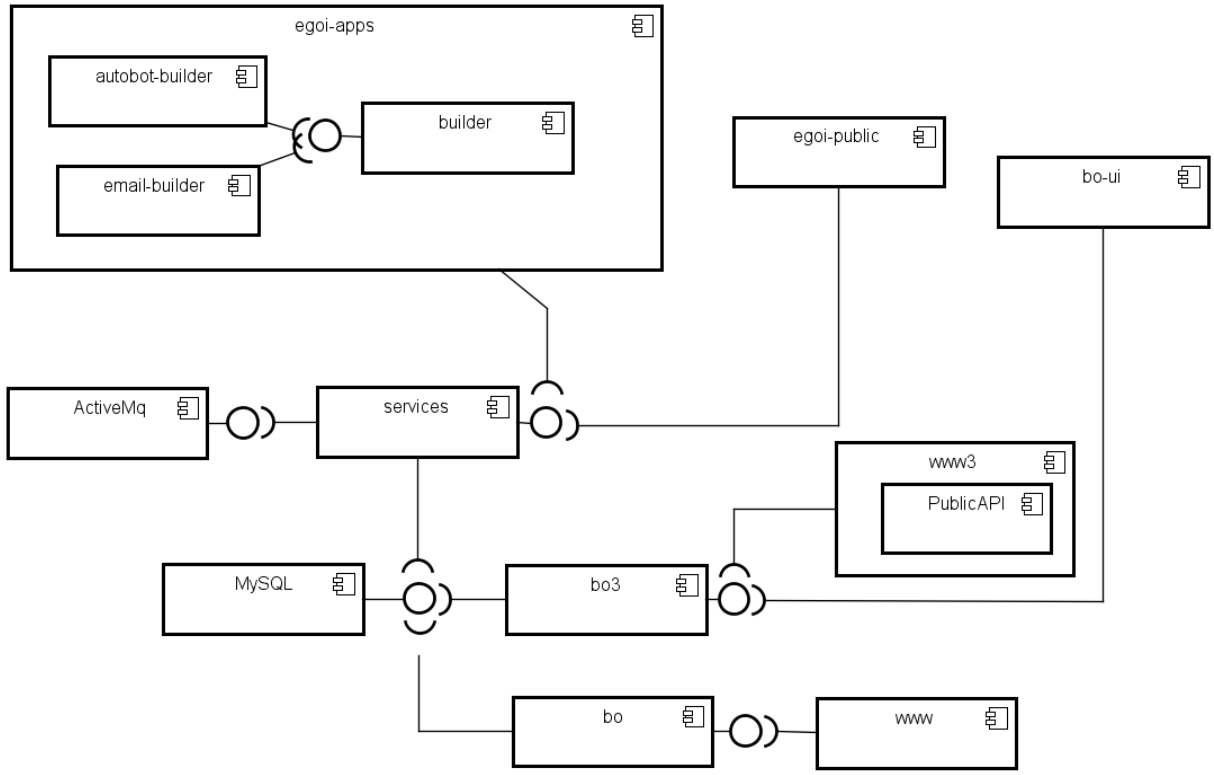

Figura 14 Vista lógica da plataforma E-goi

Após entender os principais componentes da plataforma e como estes interagem entre si, é necessário perceber como a sua execução se distribui fisicamente. A *Figura 15* ilustra a vista de implantação da plataforma. Tanto os componentes de *interface* (*bo-ui* e *egoi-apps*), como o *services* e o *egoi-public*, executam cada um na sua máquina ou *clusters* de máquinas. Os componentes *bo3* e *www3* executam num conjunto de máquinas onde também estão alocadas as bases de dados. Cada uma dessas máquinas guarda um conjunto de utilizadores diferentes. Por exemplo, na máquina *bo11* podem estar os dados de alguns utilizadores, enquanto na máquina *bo12* estão alocados dados de outros utilizadores diferentes. O componente *bo*, que executa na máquina *bo19*, tem consigo na base de dados a informação sobre em qual servidor se encontra cada utilizador. A principal responsabilidade do componente *bo* é a distribuição dos utilizadores pelos diferentes servidores. Um aspeto particular da plataforma é que cada utilizador tem a sua própria base de dados, o nome da mesma é construído com base no identificador do utilizador. Quando é necessário executar uma operação em alguma tabela, é concatenado ao nome da tabela o nome da base de dados do utilizador. Os administradores do

sistema utilizam esta distribuição de modo a que exista o menor impacto possível entre utilizadores, e para facilitar a migração de utilizadores e gestão de recursos (este assunto é abordado com mais detalhe na secção 5.4.3).

O componente *services* liga-se diretamente às bases de dados do *bo*, fazendo uso da informação registada na máquina *bo19* para saber a que servidor se ligar consoante o pedido do utilizador que estiver a tratar. Quando é recebido algum evento (e.g. envio de *email*, abertura de *email*, …) este é colocado numa fila, para ser processado posteriormente, através da tecnologia ActiveMQ que executa noutra máquina. As aplicações gráficas existentes no componente *egoi-apps* fazem uso do *services* para tratar da gestão de dados e não utilizam os componentes *bo*. Apenas o componente de *interface* gráfica *bo-ui* continua a utiliza diretamente o *bo.*

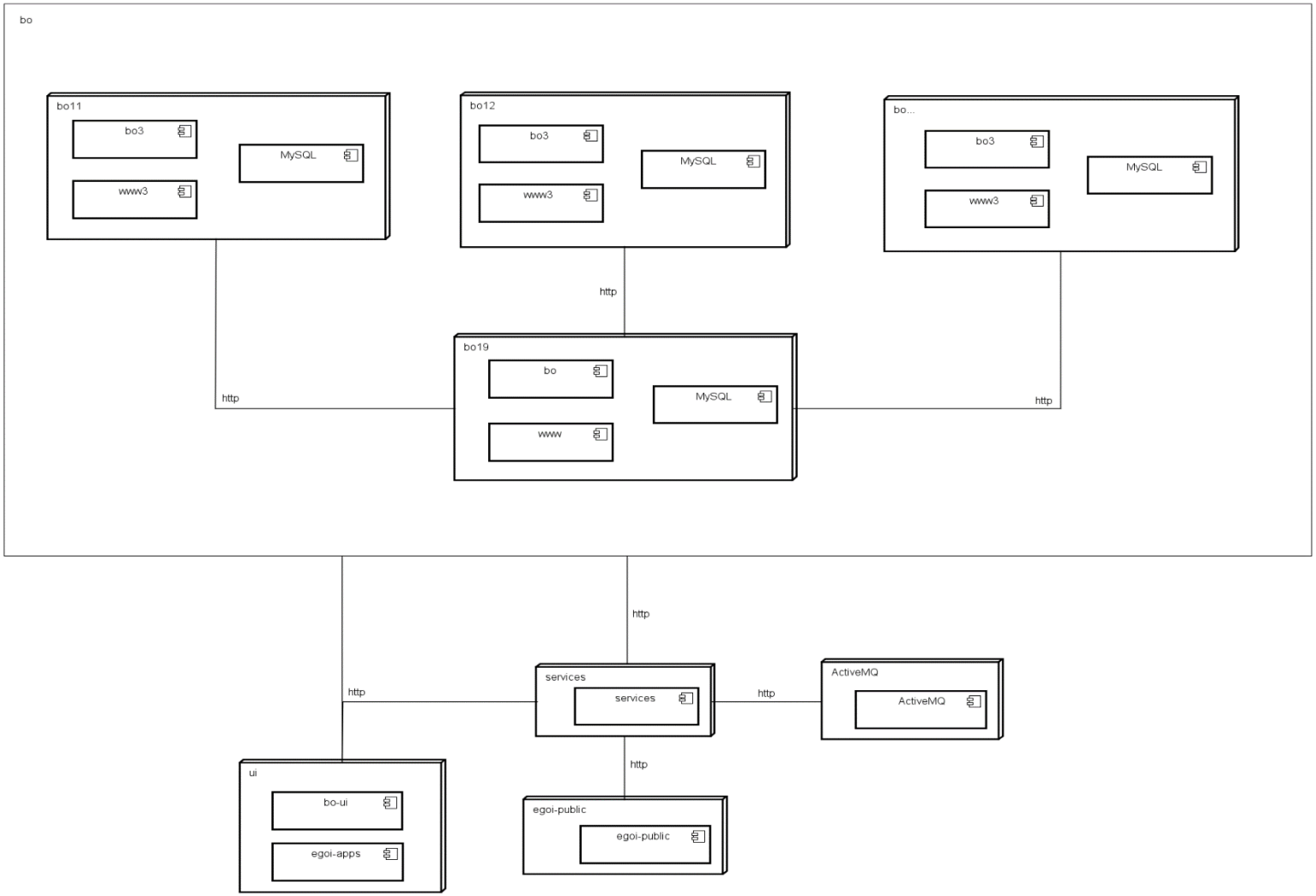

Figura 15 Vista de implantação da plataforma E-goi

O *services* é onde se encontra parte da lógica de gestão da estrutura de dados, e utiliza o estilo arquitetural REST para outros componentes acederem aos seus recursos. A *Figura 16* representa um pedaço da sua vista lógica. Parte das funcionalidades de gestão de utilizadores já se encontram implementadas neste componente, assim como a gestão das listas e contactos. Este componente já trata também de parte da criação de campanhas para múltiplos canais, como é o caso dos componentes *Email, SMS*, *Push* e *Web Push*. A lógica de gestão de eventos (e.g. abertura de um *e-mail*) encontra-se implementa no *services,* fazendo a distribuição desses eventos para a máquina onde executa o ActiveMQ. Existe bastante mais lógica de negócio já implementada no *services*, contudo grande parte da lógica ainda se encontra no *bo*, como é o caso do envio de campanhas, parte da gestão de contas, criação de formulários, entre outra. Futuramente os desenvolvedores pretendem que toda a lógica de negócio esteja migrada para este componente.

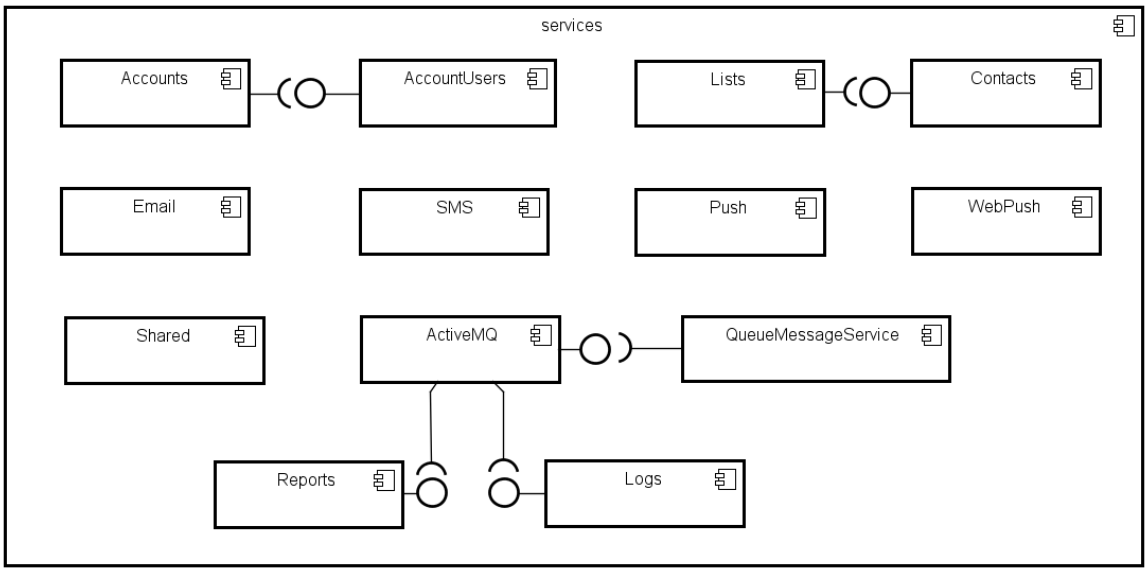

Figura 16 Vista lógica do services

Cada módulo existente no *services* segue uma variação do padrão arquitetural MVC, mas com algumas dependências impostas pela tecnologia Zend Framework. A *Figura 17* representa a arquitetura seguida pelos módulos do *services*. A responsabilidade de cada componente é explicada de seguida:

- ***Resource:*** componente que recebe o pedido e é responsável por controlar o seu fluxo do mesmo.

- ***Service:*** componente responsável por serviços externos, que tenham de ser efetuados de forma assíncrona dado o elevado volume de processamento. Este componente também é responsável por disponibilizar serviços a outros módulos do *services* que queiram aceder aos dados do módulo onde se encontra. Oferece funcionalidades de persistência e consulta de dados que através do *Repository*;
- ***Factory:*** componente responsável por criar instâncias de outros componentes;
- ***Repository:*** componente responsável por persistir os dados na base de dados. Utiliza as entidades do componente *Entity* para persistir os dados;
- ***Entity:*** representa as entidades do negócio através dos seus dados. Este componente mapeia diretamente os atributos de cada classe entidade com os campos na base de dados;
- ***Model*****:** representa entidades do negócio através dos seus comportamentos. Difere do componente *Entity* pois apenas contém comportamentos, os dados necessários são injetados;
- ***Transformer:*** responsável por transformar dados recebidos pelo recurso em instâncias de *Entity* ou o inverso;
- ***Validator:*** componente responsável por efetuar as validações de negócio. É utilizado pelo *Resource* para validar os dados no início do fluxo do pedido.

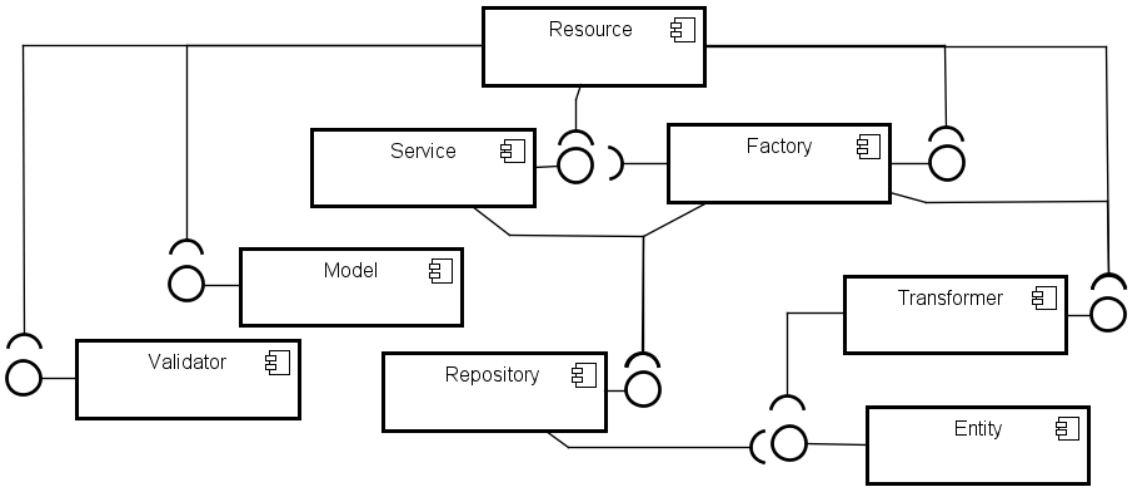

Figura 17 Vista lógica dos módulos do services

5.3.2 Arquitetura da Solução

Tendo em mente a arquitetura da plataforma E-goi, é possível construir uma solução arquitetural para o novo sistema de formulários a ser integrado nesta plataforma. A fase de arquitetura permite estruturar o sistema em grandes componentes de *software*, estabelecendo as relações e mecanismos de comunicação entre eles de acordo com os requisitos. A arquitetura deve ser maioritariamente estabelecida no início do projeto devido aos custos elevados da sua alteração. Esta define as restrições de implementação, permite estimar custos e é maioritariamente orientada a requisitos de qualidade (cf. 5.1.4).

Dos princípios arquiteturais [51] mais importantes que foram utilizados destacam-se:

- Separação de responsabilidades;
- Princípio da responsabilidade única;
- Princípio do menor conhecimento;
- Não duplicação de responsabilidades.

Atendendo aos requisitos do projeto, restrições arquiteturais identificadas e princípios mencionados, propõe-se uma solução que adota um estilo arquitetural baseado em componentes e camadas, e que segue a estrutura dos componentes mais recentes da plataforma E-goi. A *Figura 18* representa a vista lógica da solução proposta através de um diagrama de componentes com base nas restrições e princípios arquiteturais mencionados.

Para a implementação dos serviços e gestão da estrutura de dados é utilizado o componente do E-goi que trata essas responsabilidades, o *services*. Neste componente é criado o módulo *Forms*, para manter toda a lógica de domínio relacionada com formulários. Este módulo segue a arquitetura dos restantes módulos existentes no *services* (cf. 5.3.1) e permite reutilizar lógica do domínio da E-goi já existente. O módulo *Lists* é utilizado para mapeamento dos campos do formulário com os campos das listas de contactos. O módulo *Log* é responsável por registar informação de eventos que ocorram nos formulários (e.g. visitas). O módulo *Report* é responsável por gerar relatórios. Existem ainda outros módulos que podem ser reutilizados para atingir os requisitos, com o objetivo de reutilizar lógica existente e não duplicar responsabilidades, como é o caso do módulo que representa os utilizadores e contas.

Para conter as responsabilidades de visualização é criado um componente no *egoi-apps*, o *form-builder*[1]. Este componente reutiliza o *builder*[2] que já implementa funcionalidades gráficas necessárias do editor e que são reutilizadas por outro editor da plataforma. Para realizar uma análise mais detalhada dos comportamentos dos visitantes o *form-builder* utiliza as funcionalidades do *Piwik* (cf. 3.3.1).

De forma a tratar as responsabilidades públicas associadas à submissão dos formulários é proposto o módulo *Forms* no componente *egoi-public* que trata responsabilidades de camada pública. Tanto o *egoi-public* como o *form-builder* utilizam os serviços REST do *services* para aceder à lógica de negócio e aos dados persistidos.

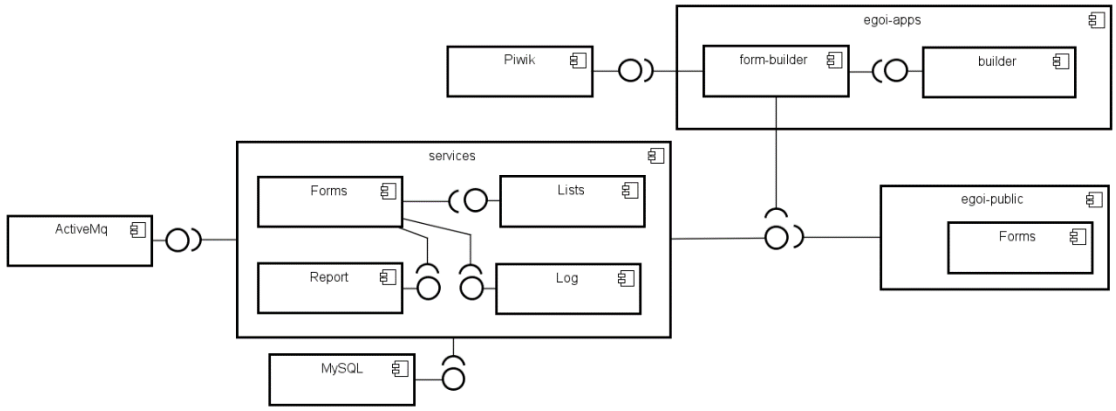

Figura 18 Vista lógica do sistema

[1] O desenvolvimento componente *form-builder* não é objetivo do trabalho efetuado nesta dissertação.
[2] O componente *builder* ainda se encontra em fase *beta* à data desta dissertação.

O diagrama de implantação apresentado na *Figura 19* representa as máquinas físicas envolvidas no sistema. Este é semelhante ao diagrama de implantação da plataforma E-goi apresentado, no entanto são acrescentados os novos componentes propostos. Na máquina do cliente executa o seu *browser* e neste a plataforma E-goi, a qual comunica com o componente com o componente de formulários a executar noutra máquina (ou conjunto de máquinas) que devolve os elementos de visualização necessários para a máquina cliente. O *form-builder* comunica através do protocolo HTTP com o componente *Piwik* durante a construção dos formulários. Este componente introduz nos formulários nestes a informação dos visitantes a ser analisada, a qual é comunicada pela máquina cliente novamente para o Piwik quando ocorre uma visita ao formulário. O *form-builder* e o *egoi-public* comunicam com os grupos de máquinas onde executa o *services*, através de HTTP, para este processar lógica de negócio e persistir os dados na base de dados através do componente MySQL.

A separação em máquinas dos componentes é utilizada para permitir escalar o sistema caso o número de utilizadores aumente e os recursos existentes não sejam suficientes, por exemplo, caso os recursos disponibilizados para o sistema de análise não sejam suficientes é possível aumentar o número de máquinas ou a capacidade de processamento alocado para este.

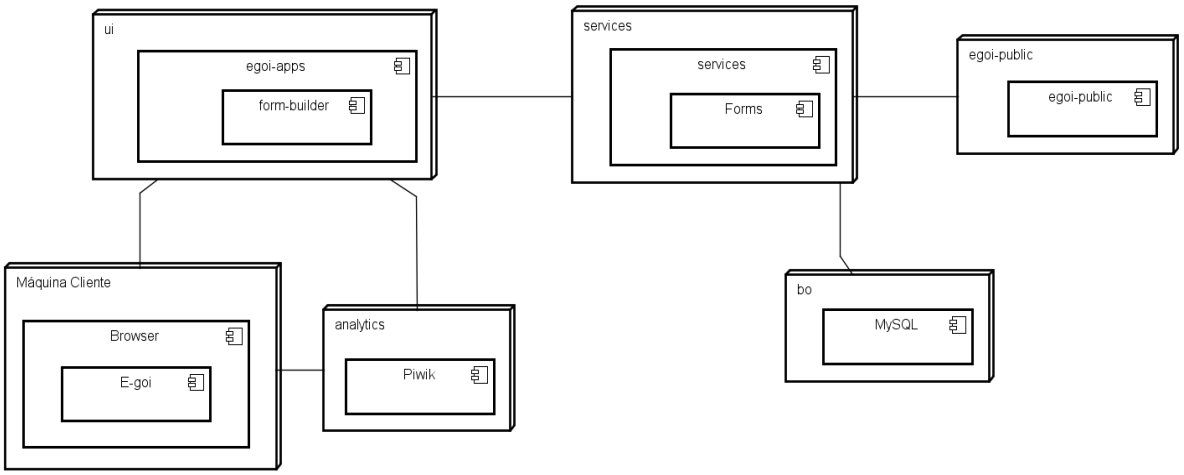

Figura 19 Vista de implantação

5.3.2.1 Alternativas Arquiteturais

O processo de desenvolvimento da arquitetura de uma aplicação inicia-se identificando os objetivos do sistema, passando à identificação de cenários chave que envolvem decisões que têm implicações na arquitetura, definindo os tipos e estilos arquiteturais a utilizar e terminando

definindo um conjunto de soluções candidatas que cumpram os requisitos. A *Figura 20* representa a vista lógica de um conjunto de alternativas que foram ponderadas como arquitetura do sistema proposto.

Uma alternativa é a construção de todo o sistema (incluindo serviços e responsabilidades de gestão de estrutura de dados) inteiramente no componente *e-goi-apps*, representado pelo componente *Forms*. A vantagem desta alternativa é reduzir as dependências, contudo seria necessário outro componente de base de dados a executar no mesmo grupo de máquinas que os componentes dentro de *e-goi-apps* o que dificultaria a manutenção do sistema. Esta solução dificultaria ainda a integração com o E-goi, pois teriam de ser criados serviços ainda não existentes no componente *services* (por exemplo para gestão dos campos das listas de contactos). Outra desvantagem desta solução é a duplicação de lógica e responsabilidades de acesso a dados já existente no *services*, como representado pelo componente *Repository,* o que sobrecarregaria este componente de responsabilidades.

Outra alternativa, abordada nas secções 3.3.5 e 4.4, é o desenvolvimento de um componente de *web analytics* específico para o sistema de formulários, o qual é representado pelo componente *FormWebAnalytics*. Esta alternativa não é implementada dado o elevado custo de desenvolvimento, e a existência de ferramentas de código aberto para este fim, permitindo a sua completa personalização e alojamento. Assim sendo opta-se por reutilizar tecnologias já existentes.

A última alternativa apresentada pelo diagrama é a utilização do padrão arquitetural MVC puro na aplicação cliente, como representado no componente *Forms*, esta alternativa está dependente da tecnologia utilizada para desenvolver o componente de visualização, excluído do âmbito desta dissertação.

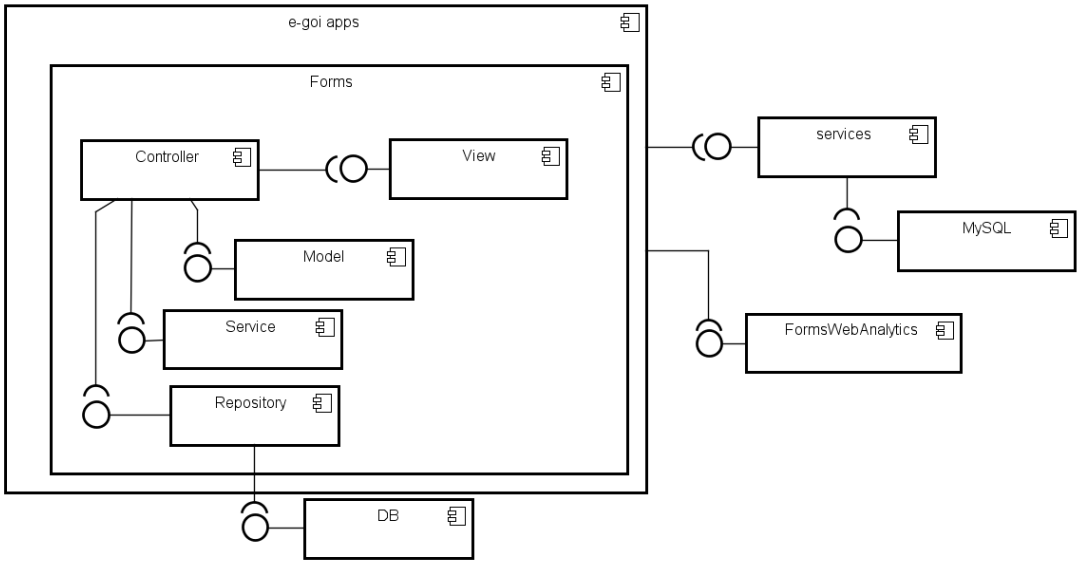

Figura 20 Alternativas vista lógica

5.3.2.2 Padrões Arquiteturais

Um estilo ou padrão arquitetural é um conjunto de princípios ou soluções para problemas recorrentes que podem ser reutilizadas aumentando a qualidade do sistema e facilitando a sua manutenção [51]. Um padrão descreve um problema recorrente e uma solução para o mesmo, a sua aplicação permite fornecer uma linguagem comum entre arquitetos facilitando a comunicação e desenvolvimento.

A arquitetura proposta utiliza os seguintes estilos arquiteturais:

- **Cliente/Servidor**: descreve um sistema distribuído onde se separa o cliente do servidor. A lógica de negócio principal encontra-se no servidor tanto como o acesso à base de dados. A utilização deste padrão facilita a manutenção do sistema pois centraliza o acesso aos dados;
- **Componentes**: uma arquitetura estruturada em componentes ou camadas permite decompor o sistema segundo as suas partes lógicas, tornando-o facilmente extensível, reutilizável, testável, coeso e com baixo acoplamento;
- ***Domain Driven Design* (DDD)** [52]**:** o desenho da arquitetura é baseado nos conceitos de negócio, assim como os comportamentos e relações. Este estilo arquitetural permite que qualquer elemento integrante do negócio perceba facilmente o desenho;

- ***Model View Controller* (MVC):** padrão arquitetural que divide o sistema em três componentes principais, *Model*, responsável por representar os conceitos e comportamentos de negócio, *View*, responsável por funcionalidades de apresentação, e *Controller*, responsável por controlar o fluxo dos casos de uso do sistema.

5.4 Design Detalhado

Esta secção documenta a solução concetual para o problema proposto, tendo como ponto de partida os artefactos criados pelas disciplinas de requisitos, análise e arquitetura, produzindo artefactos a utilizar na disciplina de implementação.

A secção de *design* tem como objetivo detalhar as especificações da solução concetual orientada a objetos, seguindo uma abordagem de desenho conduzido pelo domínio e atribuição de responsabilidades. O *design* detalhado apresenta o detalhe da solução para o problema proposto seguindo a arquitetura elaborada. Esta secção encontra-se organizada segundo atividades de grande implicação no desenho da solução, sejam *user stories* completas ou apenas uma camada da arquitetura. O desenho da solução é orientado aos conceitos de domínio, classes candidatas e às responsabilidades necessárias de acordo com os requisitos, seguindo sempre boas práticas e padrões de desenho de software, tais como GRASP [46], SOLID [53], DDD [52] e GoF [54], os quais são importantes para a compreensão desta secção.

5.4.1 Módulo *services*

A arquitetura do projeto define a necessidade de criar um módulo no componente *services* para tratar dos serviços de gestão da estrutura de dados. Para isso é necessário perceber as classes e comportamentos que já se encontram definidos neste componente e que podem ser estendidos. O *services* é essencialmente uma API REST privada utilizada internamente pelos desenvolvedores e segue práticas comuns deste tipo de APIs [55]. O objetivo desta secção não é documentar os serviços da API mas explicar que opções de desenho foram tomadas para adicionar a lógica de negócio relacionada com formulários.

A *Figura 21* representa as classes utilizadas na inicialização de um módulo e gestão do fluxo de um pedido. O *services* utiliza a *framework* Zend, algumas classes são inerentes a esta tecnologia, é o caso da classe *AbstractRestfulController* que representa as funcionalidades principais de um controlador de pedidos REST. Essa classe é estendida pela *EgoiAbstractRestfulController* que

adiciona funcionalidades específicas do E-goi, todos os controladores ou recursos do sistema devem estender esta classe. Esta define métodos para criar, atualizar, eliminar ou consultar um recurso, assim como métodos para tratamentos dos erros, os quais ficam registados no sistema. Quando ocorre a inicialização de um módulo é chamada a classe *Module* (deve ser definida em cada módulo) através do método *onBootstrap* que executa tarefas necessárias à inicialização do módulo. No caso dos módulos do *services* este método define o método *checkForPermissions* da classe *Permissions* como método a ser chamado sempre que ocorre um pedido para verificar as permissões de quem fez o pedido (o qual é feito através de outro módulo específico para autenticação). Quando é recebido um pedido, e após serem validadas as permissões, este é passado ao controlador para ser tratado.

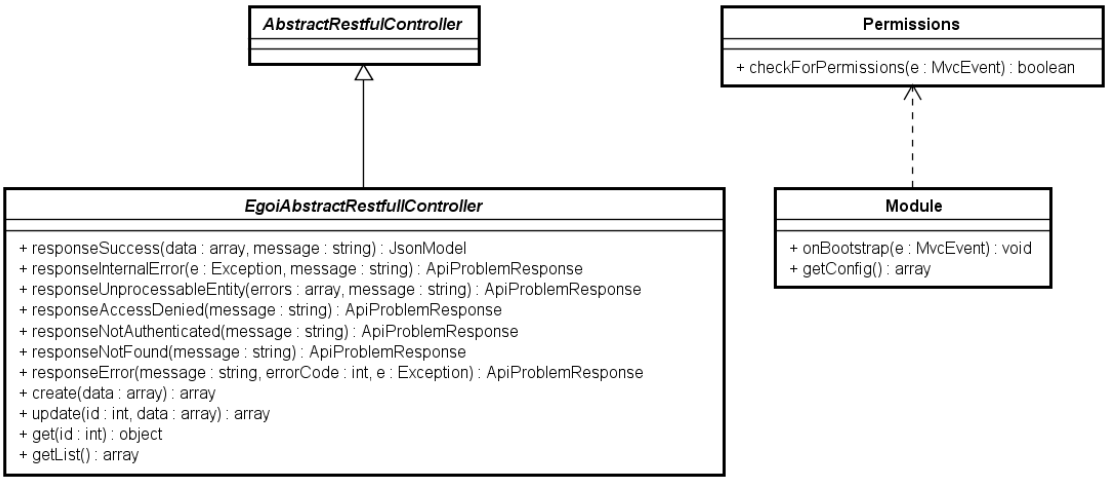

Figura 21 Classes utilizadas na inicialização de um módulo

Os módulos do *services* seguem os conceitos de DDD, pois para cada entidade do domínio desse módulo (ou conceitos relacionados com a entidade) é criado um grupo de classes para tratar dessa entidade. A *Figura 22* representa um conjunto de classes abstratas que devem ser estendidas para tratar o fluxo de dados e lógica associada a uma entidade. A classe *AbstractEntity* representa a própria entidade, esta contém métodos de acesso às suas propriedades. A classe *AbstractRepository* trata de representar os comportamentos de persistência da entidade, assim como comportamentos específicos do acesso às bases de dados do E-goi, como criação de uma conexão com um servidor onde se encontra um cliente. A classe *AbstractService* define comportamentos de um serviço, tipicamente os serviços funcionam como fachada de acesso aos repositórios ou definem serviços de domínio. As instâncias de serviços são criadas através de uma *Factory,* a qual é definida nas configurações do módulo.

Para qualquer classe aceder a um serviço deve fazê-lo através dos comportamentos definidos na *interface ServiceLocatorInterface*. Quando um controlador recebe um pedido utiliza as funcionalidades desta *interface* para aceder a um serviço, passando os dados recebidos ao serviço, este por sua vez utiliza as funcionalidades de um repositório para persistir os dados, por exemplo. No caso de se tratar de um pedido de consulta, antes de a resposta ser devolvida pelo controlador esta passa por um dos métodos *transform* da classe *AbstractTransformer* que é responsável por transformar a entidade nos dados de negócio a serem devolvidos pela resposta.

Após a compreensão das funcionalidades já oferecidas pelo componente *services* e como estas podem ser estendidas para implementar novas, é necessário desenhar o sistema para ter em consideração os requisitos, tal é explicado nas secções seguintes para cada um dos requisitos identificados.

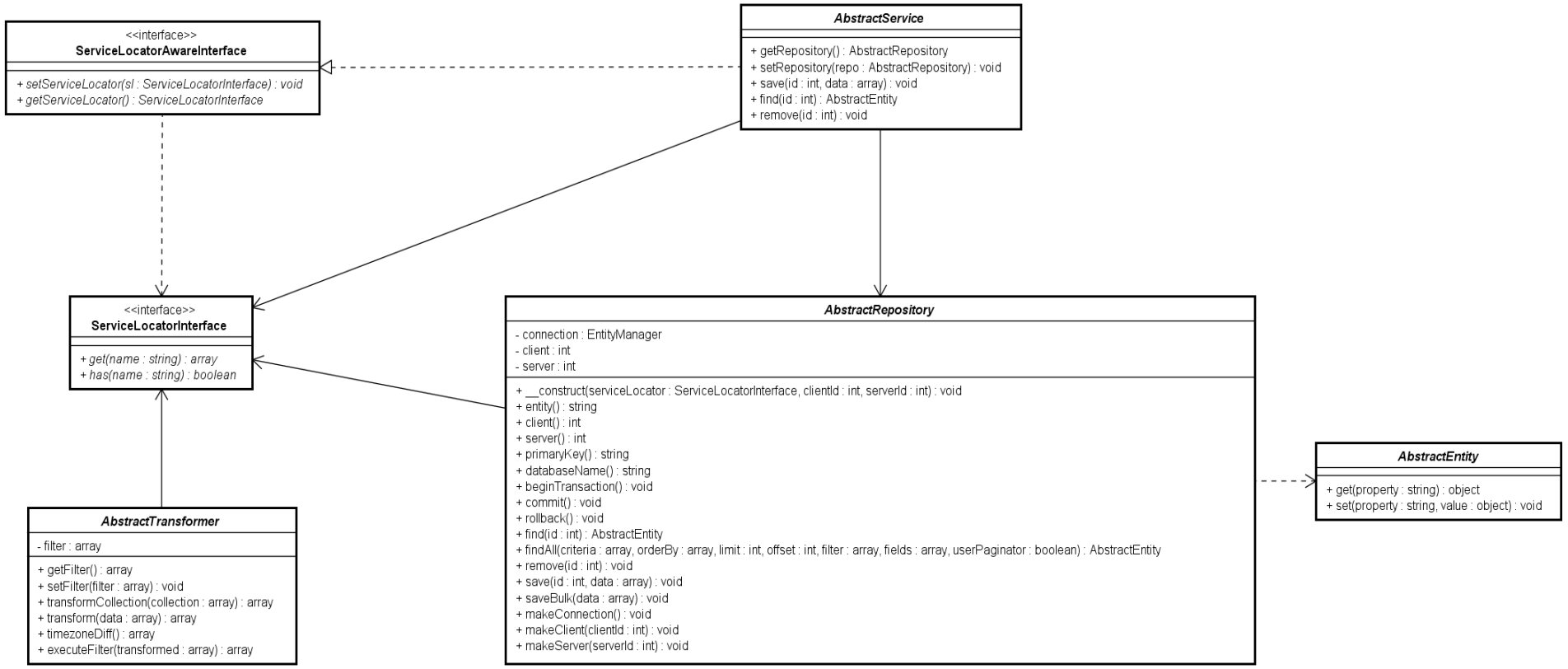

Figura 22 Classes abstratas que representam os comportamentos das diferentes camadas do service

5.4.2 User Stories

Esta secção descreve o *design* detalhado de algumas das *user stories* enumeradas (cf. 5.1.3). Outras *user stories* que não apresentem grandes variações de fluxo e não precisem de uma descrição tão detalhada são apresentadas em anexo (cf. Anexo A).

5.4.2.1 US01 – Criar Formulário

Através da análise dos requisitos identifica-se a necessidade da criação de um recurso para gerir a entidade formulário, sendo esta a entidade mais importante do negócio. A *Figura 23* representa um diagrama de sequência com o fluxo da criação de um formulário. O utilizador escolhe qual o tipo de formulário que pretende criar, o seu nome, e a lista de contactos associada, passando estes dados ao componente de visualização, *form-builder*, o qual não é objeto de desenvolvimento deste projeto. Este componente tem a necessidade de persistir os dados recebidos, para isso passa-os para o *services* através de um pedido HTTP POST, neste caso para o URL *services.e-goi.com/api/forms* passando os dados no corpo da mensagem e a chave do utilizador no cabeçalho. A nomenclatura do URL e formato dos pedidos segue a arquitetura REST como já mencionado. Esse pedido chamará o método *create* do controlador/recurso que trata os formulários. A responsabilidade deste recurso é controlar os fluxos de gestão da entidade formulário, neste caso a criação. Quando o *FormResource* recebe os dados deve validá-los inicialmente para em caso de falha, notificar o cliente, ou prosseguir com a criação em caso de sucesso. Segundo o princípio GRASP *Information Expert* a validação dos dados devia ser da responsabilidade de quem contém os dados, ou seja, a entidade formulário. Contudo, e como abordado anteriormente na secção arquitetural, o *services* define uma camada específica para validação de dados pois a entidade ainda não se encontra criada no início do fluxo do pedido.

Após a validação dos dados, e caso o formato destes seja válido, é utilizado o método *get* da *ServiceLocatorInterface* para aceder a um serviço, utilizando o princípio de inversão de dependências não existe dependência de um serviço concreto, mas de uma abstração. Neste caso é criado o serviço *FormService* para realizar a lógica de serviço relacionada com formulários. O recurso chama o método *save* do serviço que por sua vez chamada o método *save* do *FormRepository* o qual é injetado como dependência do serviço. O repositório recebe os dados, os quais são passados no formato JSON, e é responsável por criar uma instância de *Form*, entidade formulário, pois é este que contém e utiliza a entidade para a persistir na base de dados. Finalmente quando a instância é retornada o *FormResource* verifica se a instância foi

criada com sucesso, ou se ocorreu algum erro, devolvendo uma resposta adequada. As respostas são devolvidas através dos métodos da classe pai do recurso, o *EgoiAbstractRestfulController*. Todas as classes utilizadas no fluxo do serviço devem estender as classes abstratas apropriadas representadas anteriormente.

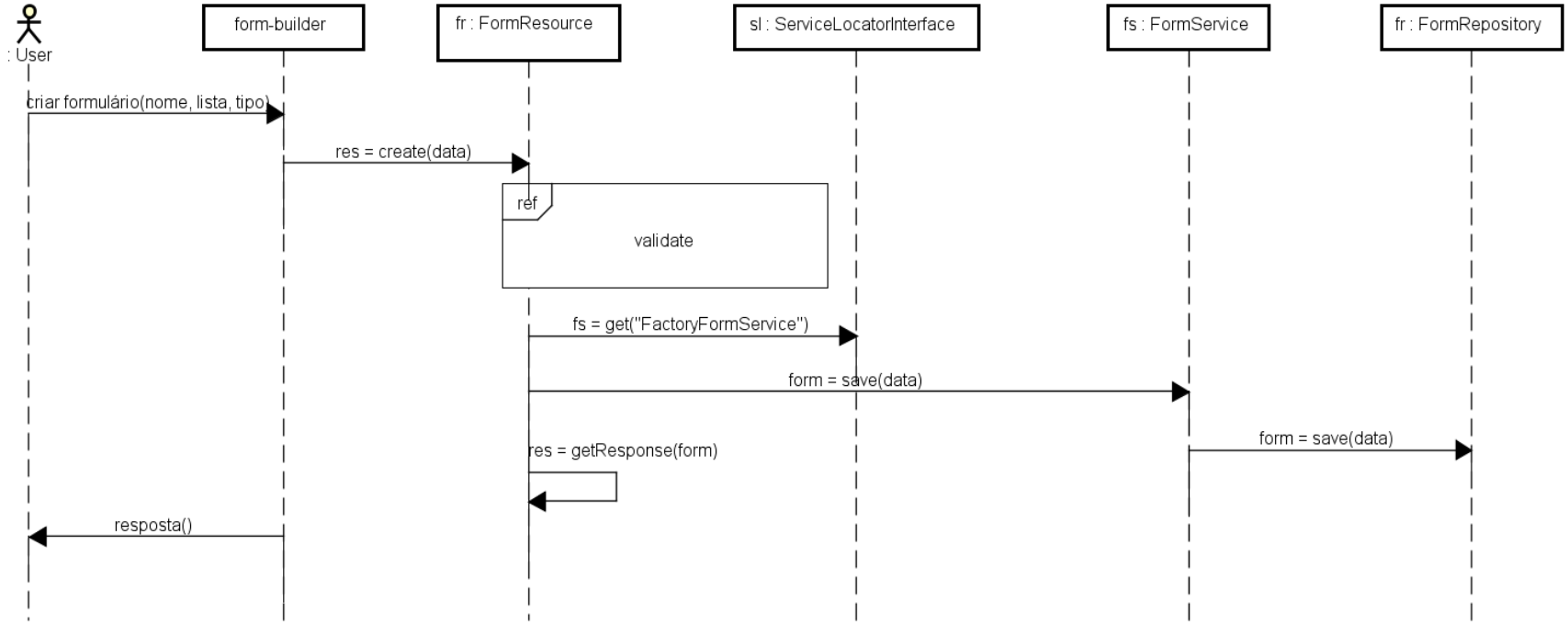

Figura 23 Diagrama de sequência: Criação de formulário

A *Figura 24* mostra o fluxo de criação de um serviço e injeção do repositório neste. É criada a classe *FactoryFormService* que contém o método *createService*, este é responsável por criar o repositório, criar o serviço passando o repositório criado, e devolvê-lo. As fábricas são definidas nas configurações, são criadas na inicialização e podem ser acedidas através da *ServiceLocatorInterface* (este é apenas um pormenor de implementação, mas é importante para o desenho dos serviços).

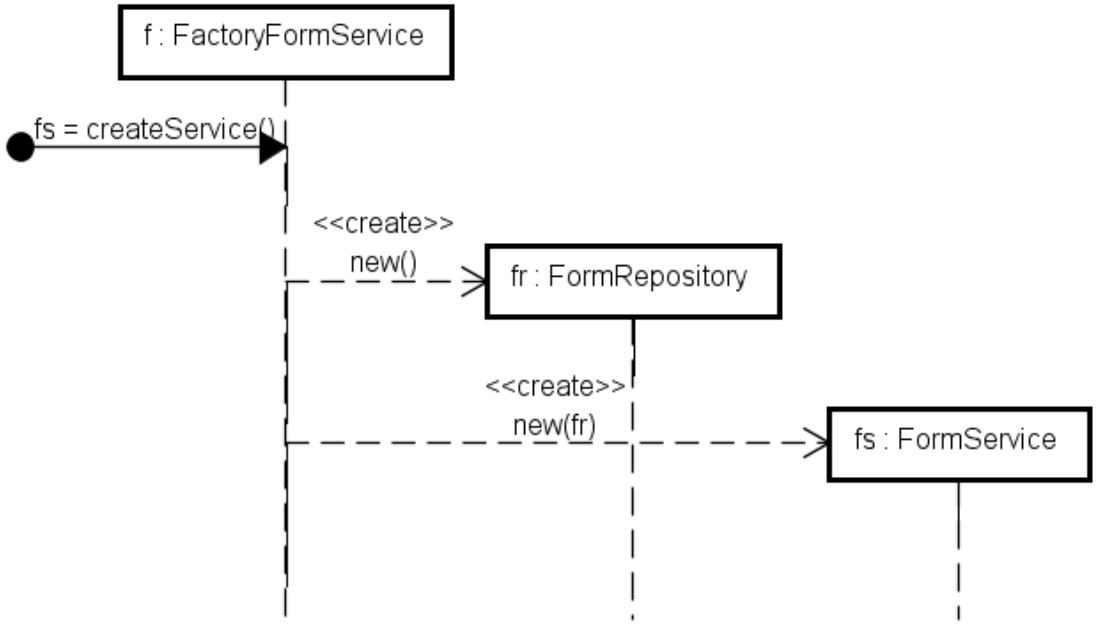

Figura 24 Diagrama de sequência: Criação de um serviço

Para validar os dados recebidos pelo recurso são utilizadas as classes *InputFilter* e *InputFactory* como representado na *Figura 25*. A fábrica recebe os filtros que serão utilizados na validação para passá-los ao *InputFilter* na sua criação. Os filtros devem validar cada campo dos dados e podem ser classes definidas no componente *Validator* para validações específicas do negócio. Após verificar se os dados são válidos, através do método *isValid*, é devolvida uma resposta adequada.

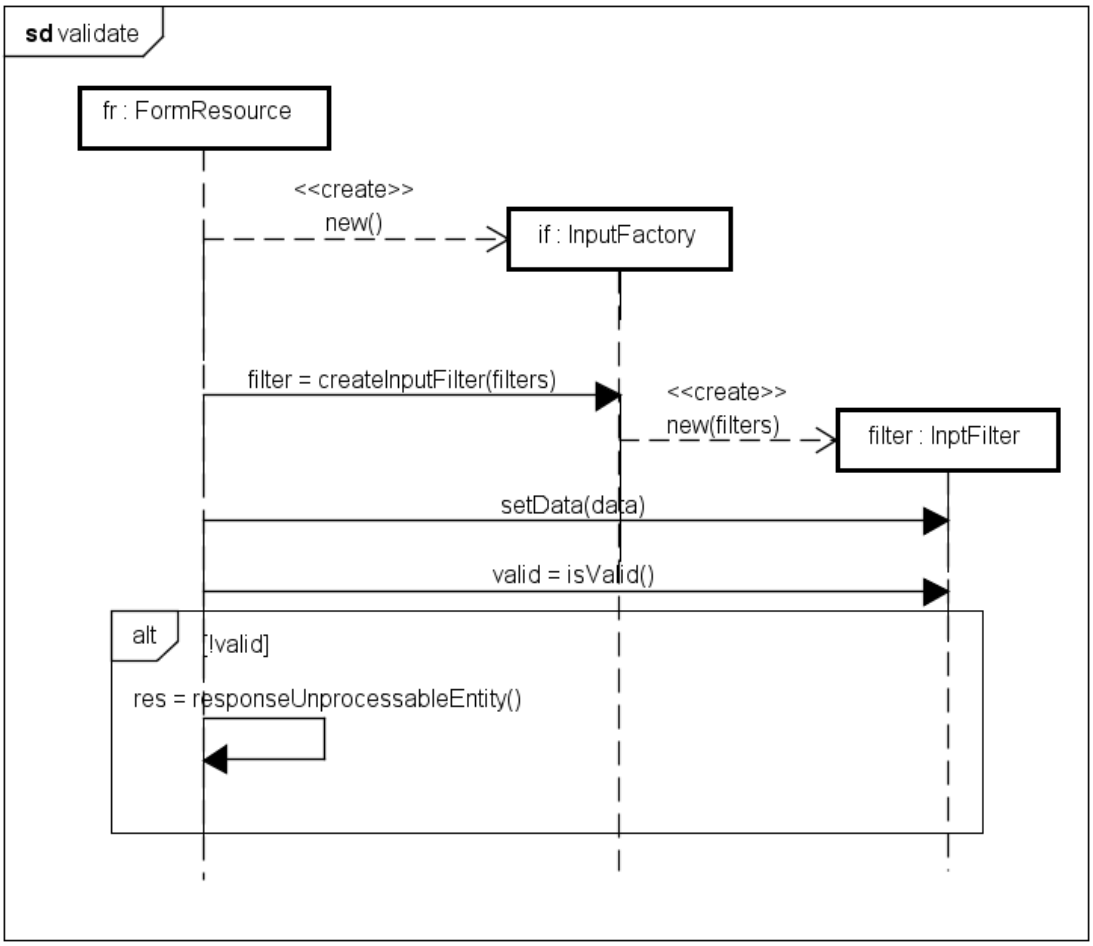

Figura 25 Diagrama de sequência: Validação

Para além da criação do formulário o *FormResource* também trata de outros comportamentos. O diagrama da *Figura 26* mostra a consulta de uma lista de formulários. As classes utilizadas são as mesmas que na criação, contudo o pedido efetuado dá origem à chamada do método *getList*. Podem ser passados filtros no pedido, os quais são obtidos através do método *getDataFromQuery*, esses filtros são passados para o método *findAll* do *Service* que chama o método *findAll* do repositório, este sim é responsável por conectar à base de dados para obter a lista de todos os formulários daquele utilizador. Quando os formulários são devolvidos pelo repositório o recurso necessita de os transformar em algo que possa ser devolvido como resposta do serviço. Essa responsabilidade é atribuída à classe *FormTransformer*, as instâncias dessa classe também podem ser obtidas a partir de uma fábrica, não existindo assim responsabilidades de criação no recurso. O *FormTransformer* deve devolver os formulários num formato adequado através do método *transform*. Por fim o *FormResource* devolve os formulários na resposta ao pedido.

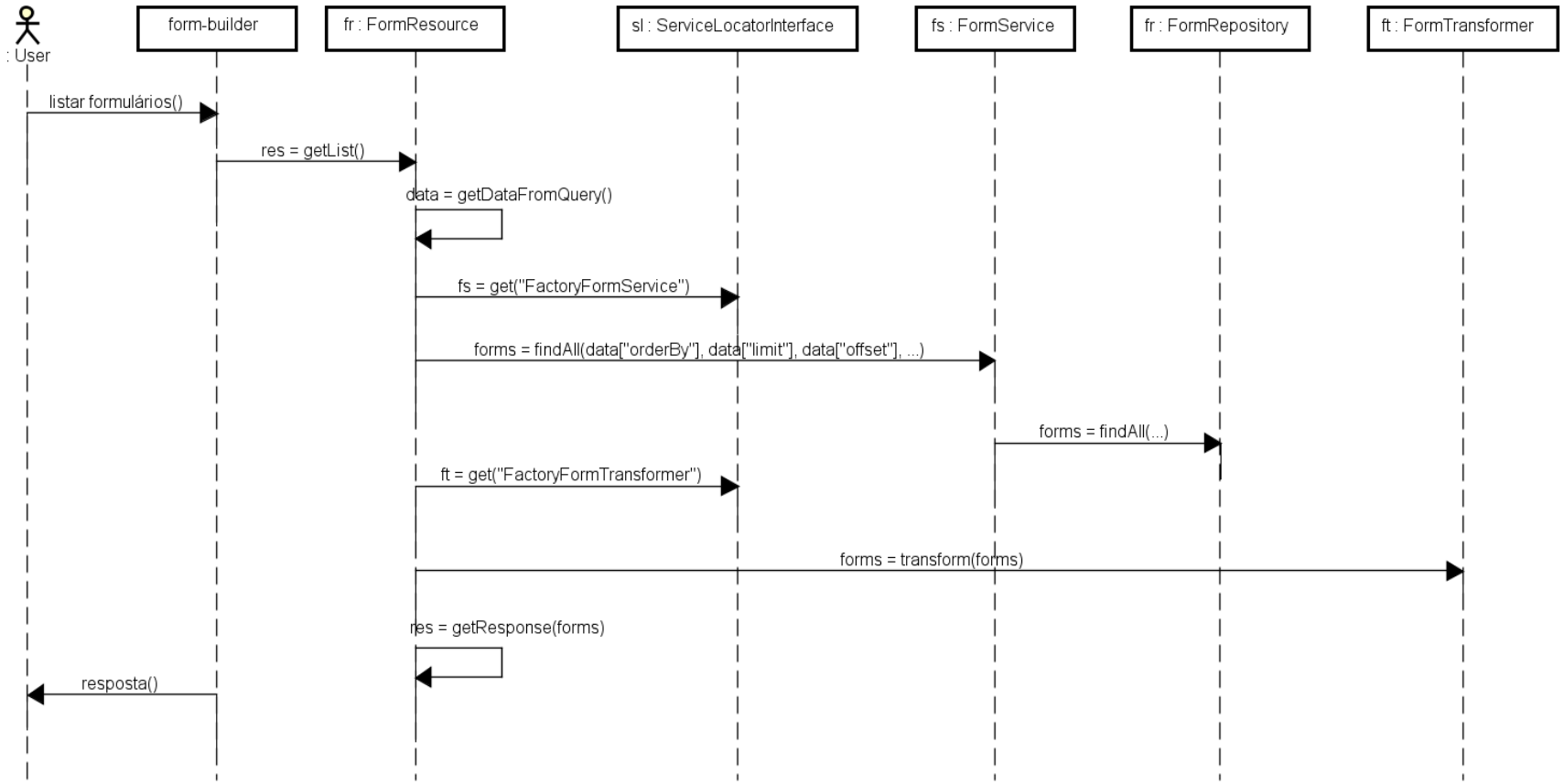

Figura 26 Diagrama de sequência: Consulta de formulário

Como ponto de vista mais completo do serviço de criação do formulário é apresentado um diagrama de classes na *Figura 27*. Para além das classes já mencionadas encontra-se também a classe *FormValidator* para executar validações específicas do formulário. A entidade *Form* também é representada através de alguns dos seus atributos como o nome, tipo, estado, lista associada, utilizador que criou o formulário e data de criação. As entidades contêm apenas comportamentos de acesso e alteração dos dados da mesma. O diagrama ilustra também os comportamentos de outras classes e encontram-se representados todos os pedidos possíveis para o *FormResource*.

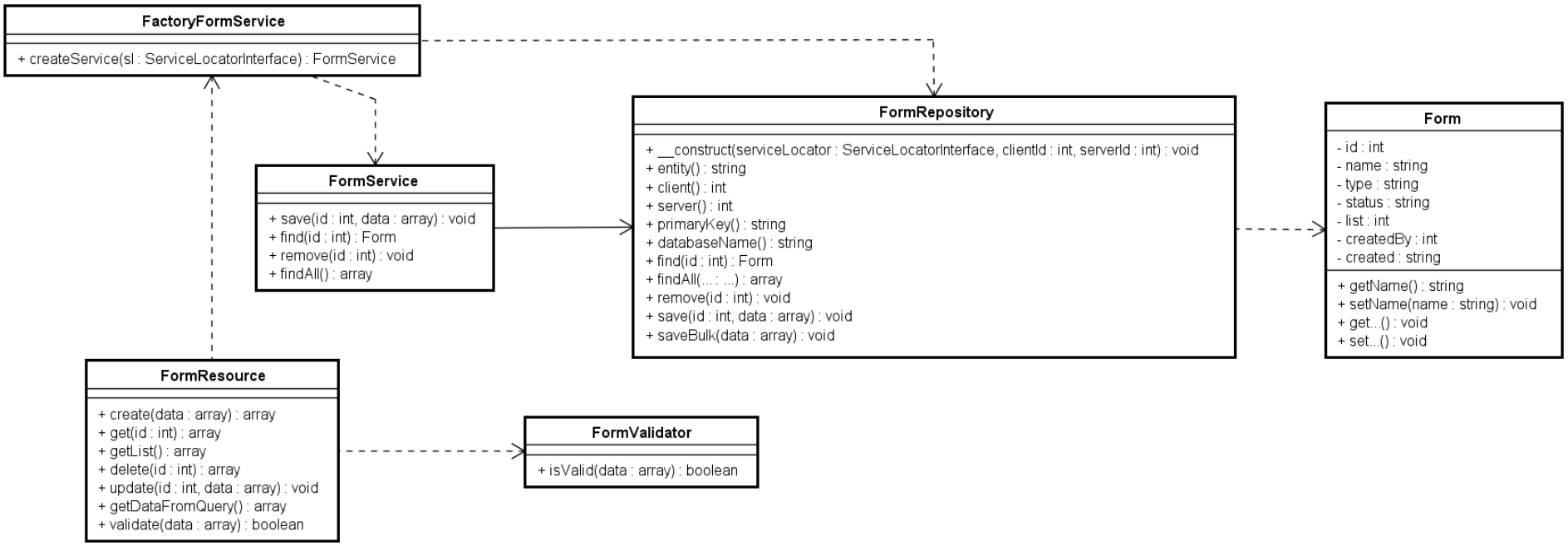

Figura 27 Diagrama de classes: Classes utilizadas na criação do formulário

5.4.2.2 US04 – Publicar Formulário

A publicação do formulário corresponde à sua disponibilização pública para os visitantes e ao armazenamento das configurações das publicações. Os serviços de publicação são similares aos restantes no que diz respeito ao armazenamento dos dados, contudo com lógica específica da publicação. Podem existir diversos tipos de publicação, por exemplo, integrar o formulário numa rede social, o que pode diferenciar as publicações nos seus comportamentos. Identifica-se assim uma possível variação, atendendo ao princípio *Protected Variations* é necessário proteger o sistema contra essa variação. Para tal é utilizado o padrão *Strategy* implementando uma *interface* que define o comportamento da publicação. A *Figura 28* ilustra o fluxo da publicação do ponto de vista do serviço, este contém a *interface PublishForm* que define o comportamento da publicação.

Tal como nos outros serviços são criadas as classes para gerir a entidade de publicação. Quando o formulário é publicado o recurso deve guardar as configurações da publicação. Porém, é necessário efetivamente publicar o formulário (que corresponde ao seu alojamento). Tal é feito recorrendo à *interface* de publicação criada, a qual, no tipo básico de publicação, irá chamar o serviço de publicação da camada pública (cf. 5.3.1) pois é este que é responsável pelo alojamento. O método de publicação, *publish*, é chamado passando os dados da publicação, o identificador da publicação, o identificador do cliente e o *PublishFormService,* que efetuará o pedido à camada pública. Os dados da publicação contêm o seu tipo, as limitações, o URL (ainda não gerado), e o conteúdo HTML desta, o qual vem diretamente da camada de visualização pois é esta que tem essa responsabilidade. O identificador do cliente é obtido através do *PublishFormRepository* pois este sabe qual é o cliente devido às funcionalidades de autenticação. Após a publicação é devolvido o URL gerado o qual é adicionado ao objeto *PublishFormEntity* e por fim este é atualizado.

A *Figura 29* mostra as classes envolvidas na publicação do formulário, mais uma vez a classe candidata de publicação passou a classe de *software, PublishFormEntity*. Porém esta apenas contém os dados da publicação, pois os comportamentos são definidos na *interface PublishForm*.

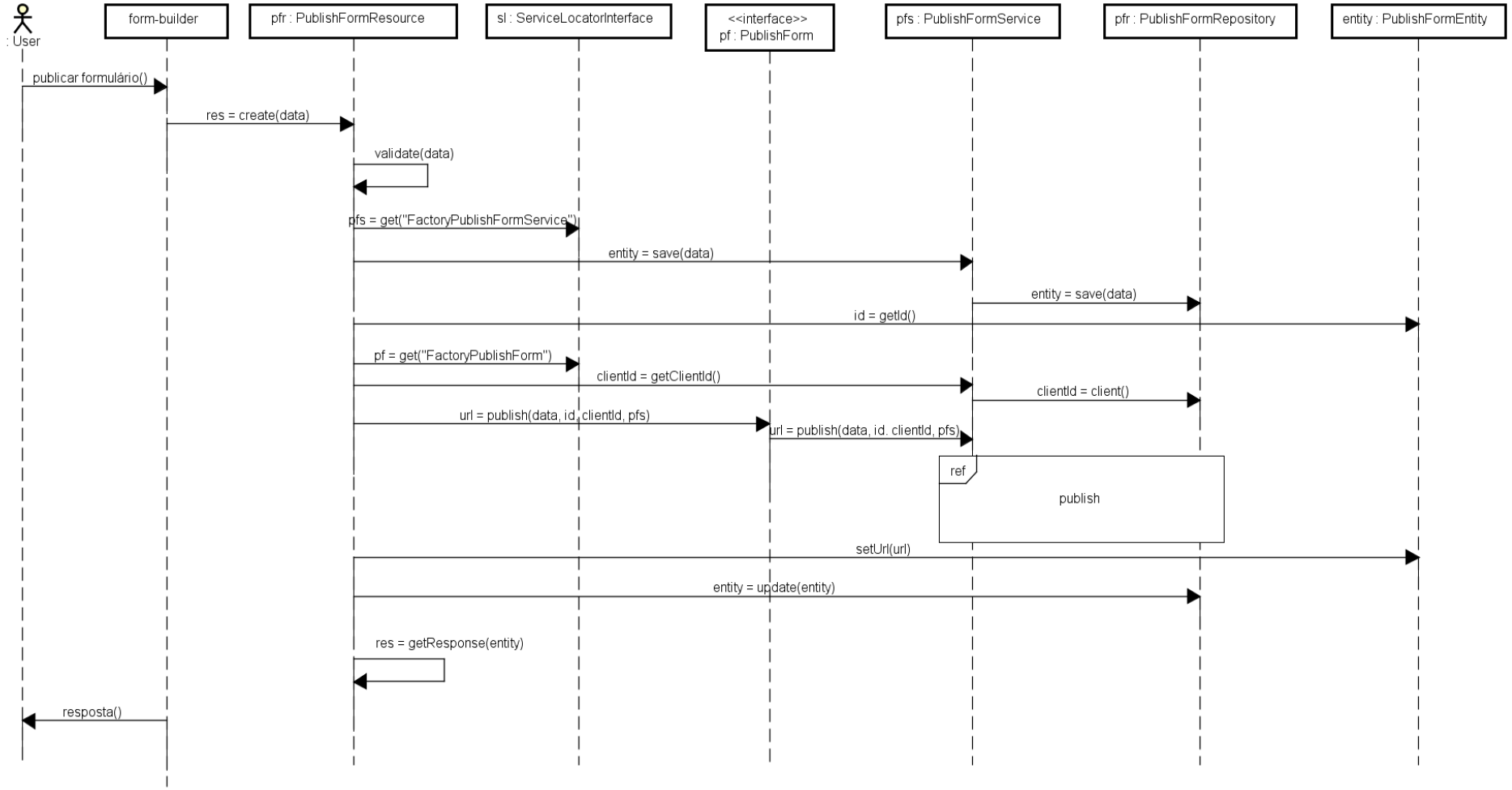

Figura 28 Diagrama de sequência: Publicação de formulário

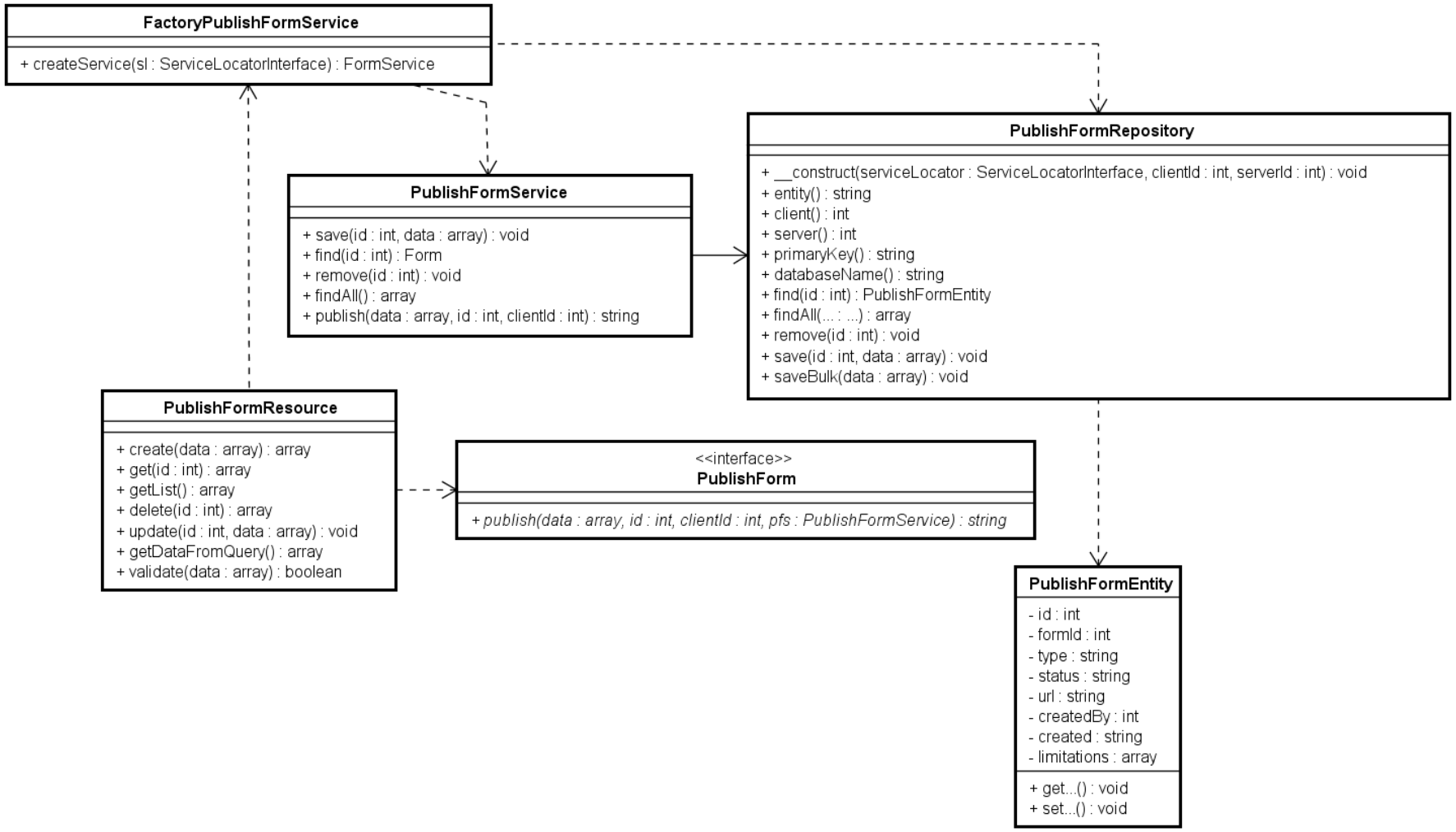

Figura 29 Diagrama de classes: Publicação de formulário

A publicação do formulário deve ficar acessível a qualquer visitante através de um URL, tal é feito através da camada pública, como indicado na *Figura 30*. O componente *egoi-public* segue uma estrutura similar ao componente *services*. É necessário um controlador para controlar o fluxo do pedido de publicação, *FormController*, após receber o pedido este verifica se a chave introduzida é válida, pois a publicação só pode ser feita a partir de um serviço com permissões para tal (esse serviço deve passar uma chave específica à camada pública). Após a validação da chave é necessário validar o conteúdo para verificar que veio num formato correto. A responsabilidade de executar a lógica de negócio é atribuída à classe *PublishForm* esta surge diretamente do domínio. A criação de instâncias desta classe é feita através de uma fábrica. A classe *PublishForm* deve ser capaz de gerar um código único da publicação, tal deve ser feito recorrendo ao identificador do cliente, do formulário e da publicação. O código gerado é utilizado para gerar o URL da publicação do formulário. A própria publicação do formulário consiste em criar um ficheiro com os seus conteúdos, ou seja, com o HTML do formulário, ficheiro esse que deve ser acedido através do URL gerado. Por requisito devem ser guardadas junto da publicação as suas limitações (e.g. número máximo de visitas) e a contagem atual de limitações, desta forma o desenho do sistema permitirá o acesso em grande escala aos formulários pois apenas é necessário consultar simples ficheiros com os dados para executar as suas limitações. O fim do fluxo de publicação na camada pública consiste em devolver o URL como resposta.

A *Figura 31* representa as classes mais importantes deste passo da publicação, o *FormController* define a ação de publicação, e a classe *PublishForm* que contém a lógica de publicação e alojamento do formulário. A classe *FormController* estende o *EgoiPublicAbstractController*, este último é proposto nesta dissertação de forma a definir as funcionalidades de um controlador da camada pública, entre elas a validação da chave do pedido, autenticação de um utilizador e definição de ações comuns. Desta forma aumenta-se a reutilização de lógica e diminui-se a lógica duplicada.

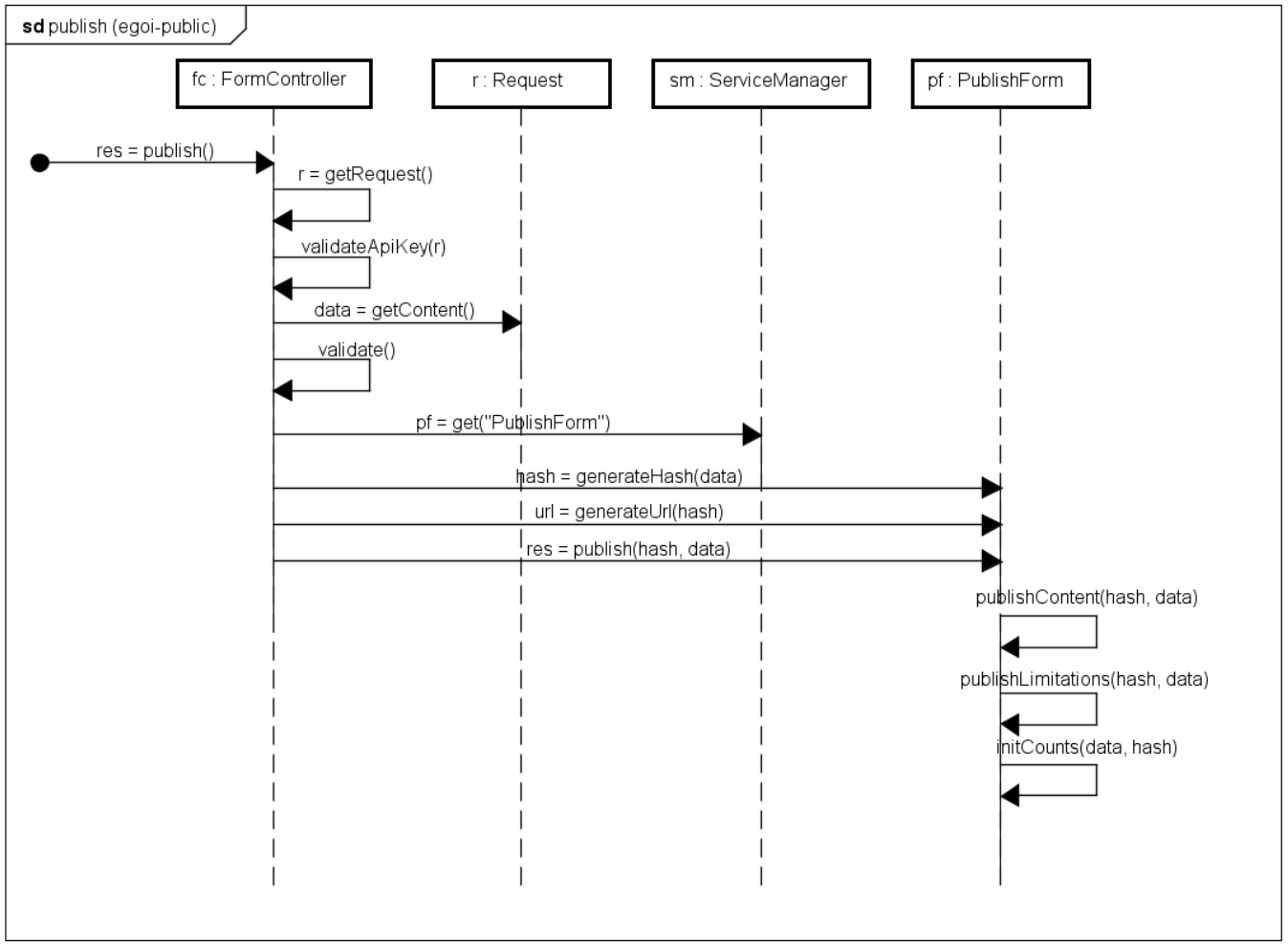

Figura 30 Diagrama de sequência: Publicação (egoi-public)

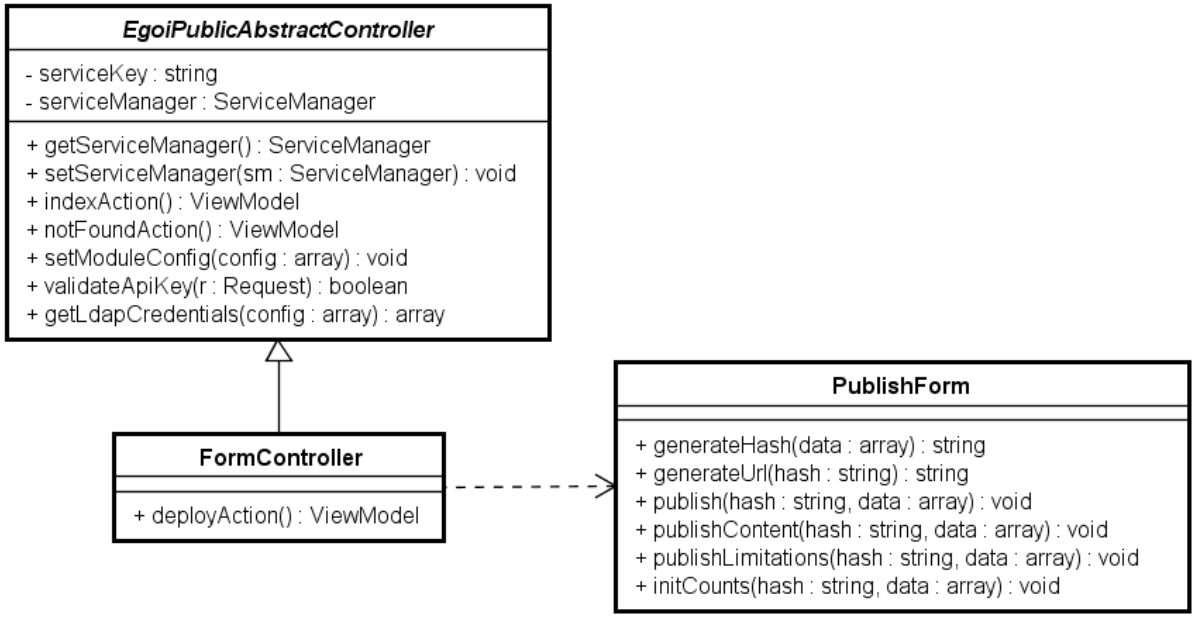

Figura 31 Diagrama de classes: Publicação (egoi-public)

5.4.2.3 US10 – Submeter Formulário

A submissão do formulário é um dos requisitos mais importantes e complexos do sistema. O visitante deve conseguir aceder ao formulário, preencher os seus campos e registar a sua submissão. Qualquer pessoa que tenha acesso ao formulário, mediante as suas limitações, pode efetuar uma submissão, portanto é importante que o sistema esteja preparado para tal. Dado o fluxo de visitantes a um formulário poder ser elevado e o formulário ter de ser disponibilizado publicamente, este é alojado num conjunto de máquinas separado dos restantes serviços, com um componente para fazer a sua gestão, o *egoi-public*.

Quando um visitante acede a um formulário para realizar uma submissão é necessário validar se formulário pode ser disponibilizado mediante as suas limitações. A *Figura 32* mostra a lógica executada quando um visitante acede a um formulário. Para controlar o fluxo da submissão é criado o *SubmissionController*, este é chamado pela máquina que executa o *egoi-public* quando um visitante aceder ao URL de um formulário. O controlador deve chamar o método *validateLimitations* da classe *PublishForm* que contém a lógica de validação das limitações. Para obter uma instância da classe *PublishForm* é utilizado o *ServiceManager* que, através do método *get*, chama a fábrica passada, a qual devolve uma instância para a classe. A responsabilidade de validação é atribuída à classe *PublishForm* tendo em conta que esta contém a lógica de publicação que alocou os dados necessários para executar a validação. O método *validateLimitations* deve aceder ao ficheiro estático que contém as limitações e ao ficheiro que contém as contagens, verificando se as contagens não ultrapassam os limites definidos para poder devolver o formulário ao visitante. O método *validateLimitations* necessita de receber o URL para, através dele, saber qual é o formulário a tratar, e o IP da máquina que efetuou o pedido para validar as limitações relacionadas com o mesmo. Dado o nome do ficheiro das limitações ser construído com base no código que é gerado para construir o URL do formulário, é possível usar o URL para saber qual o formulário a tratar. A lista de IP's que já acederam ao formulário deve também estar registada no ficheiro das contagens.

Após validadas limitações como o número de visitas, o número de submissões efetuadas, e as restantes mencionadas, e caso não tenha sido atingido nenhum limite, o formulário pode então ser devolvido ao visitante.

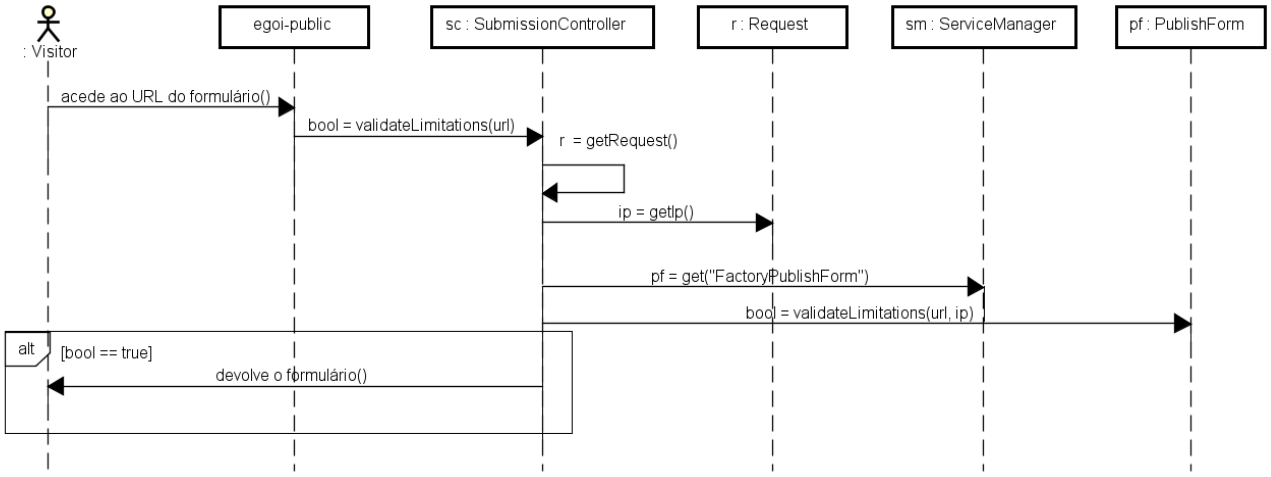

Figura 32 Diagrama de sequência: Acesso ao formulário

Quando um visitante acede a um formulário e este lhe é devolvido, é necessário registar uma nova visita antes de o visitante preencher os campos ou submeter. As visitas podem ser registadas de duas formas: através de um sistema de *web analytics*; ou através de funcionalidades dos serviços do E-goi. Escolhe-se utilizar as duas formas para registar as visitas. A razão de utilizar duas formas diferentes advém de ser necessário manter as vistas no *egoi-public* para tratar das limitações, contudo os algoritmos do sistema de *web analytics* permitem obter bastante mais informação. Para além disso é possível garantir fiabilidade para os utilizadores, pois caso não seja possível manter um sistema externo ou este se revele ineficiente é sempre possível fornecer dados básicos das visitas. A *Figura 33* apresenta a utilização dos serviços do E-goi para registar a visita, assim como o fluxo de submissão (a utilização do componente de *web analytics* é na secção 5.4.4).

Cada vez que ocorre uma nova visita o próprio componente de visualização do formulário passa ao *SubmissionController* do *egoi-public* os dados do visitante (e.g. código do formulário, *browser* em utilização, especificações do dispositivo, …) através da ação *visit*. Ao receber esta ação o controlador deve atualizar os dados das contagens presentes na camada pública, para isso é criado o método *newVisit* da classe *PublishForm* que deve adicionar a nova visita ao ficheiro de contagens. Para além disso é necessário registar os dados da nova visita na base de dados através do componente *services*. Este componente contém módulos para receber eventos como uma visita que podem ser ocorrer com elevada frequência. O *services* oferece o módulo *QueueMessageService* que é responsável por tratar eventos/mensagens de forma

assíncrona no caso de mensagens que possam ser enviadas com uma frequência elevada como acontece com as visitas. Portanto é criado um novo serviço, o *SubmissionFormService*, que através do método *newVisit* fica responsável por enviar o novo evento para o *services*. Tendo em conta que o *services* é uma API privada com autenticação é necessário passar uma chave de um utilizador no cabeçalho do pedido ou então passar as suas credenciais como indicado. A reutilização do módulo *QueueMessageService* revela-se importante pois este a partir dos eventos tratados por este também é possível facilmente registá-los na base de dados ou gerar relatórios.

Caso o visitante submeta o formulário o componente de visualização deve executar as validações necessárias (e.g. campos obrigatórios, …), para, em caso de sucesso, chamar a ação de submissão do *SubmissionController*. Quando ocorre uma submissão é necessário atualizar os ficheiros de contagem no *egoi-public* adicionando uma nova submissão ao formulário a tratar, novamente é utilizada a classe *PublishForm* através do método *submit*. Esta responsabilidade é atribuída a esta classe pois é ela que contém a lógica e nomenclatura de criação dos ficheiros de limitações e contagens. Falta apenas guardar os dados que foram efetivamente submetidos, para isso é criado um recurso no *services,* o *SubmissionResource,* que deve tratar lógica de gestão de dados das submissões. Mais uma vez é utilizado o *SubmissionFormService* para efetuar o pedido aos *services* com os campos que foram preenchidos. No final da submissão é apresentada a página de sucesso ao visitante.

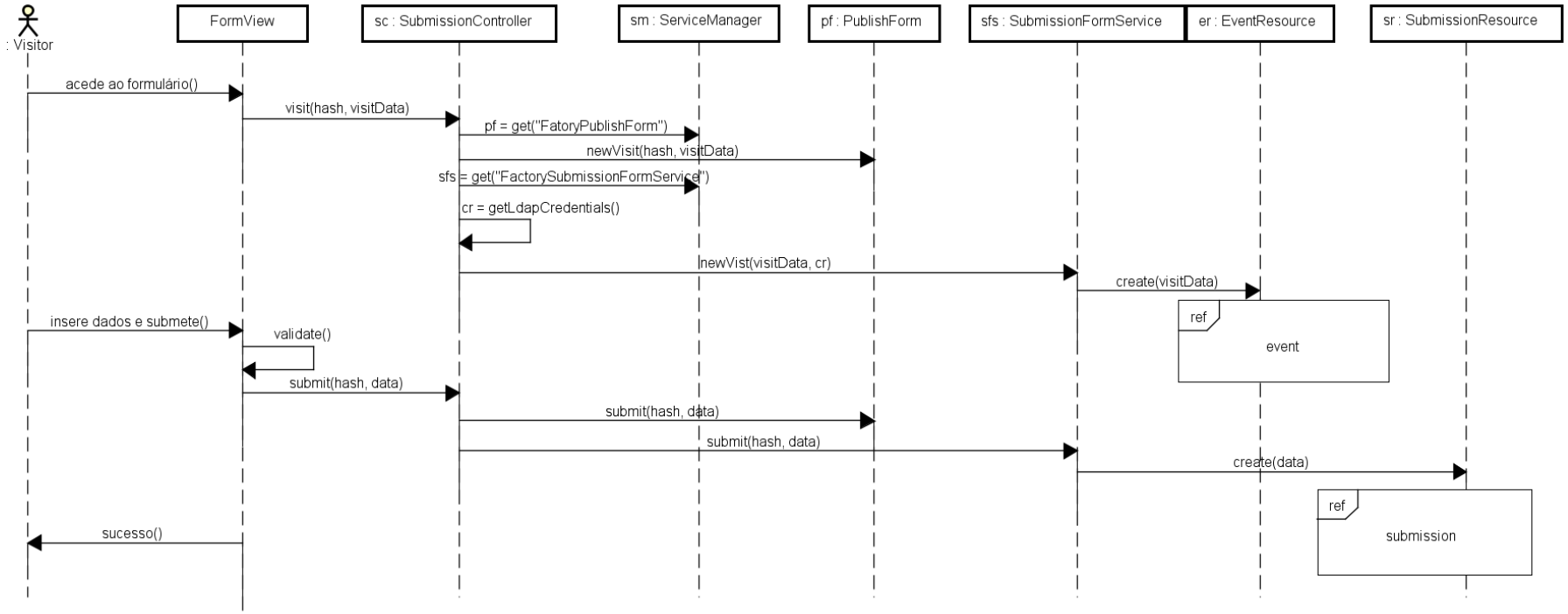

Figura 33 Diagrama de sequência: Visita e submissão

O *services* inclui o módulo *QueueMessageService* que recebe eventos ou mensagens (e.g. visitas, *clicks*, aberturas, …) através do controlador *EventResource*. Este módulo utiliza a tecnologia ActiveMQ abstraindo a sua utilização para colocar eventos numa fila e processá-los assincronamente quando possível. Existem dois conceitos fundamentais num sistema de gestão de filas, o produtor, que coloca a mensagem na fila, e o consumidor, que se encontra continuamente à espera de mensagens para processar. No caso das visitas a um formulário faz sentido processá-las através deste módulo pois trata-se de um evento de domínio que não precisa de ser processado de forma síncrona e assim o sistema encontra-se preparado para um fluxo elevado de eventos deste tipo visto que é necessário aceder à base de dados para os registar.

A *Figura 34* representa o comportamento do produtor da mensagem para a fila quando é recebido o evento. O recurso, baseado nos dados do evento passado (neste caso dados de uma visita a um formulário), determina qual método do serviço a chamar, e chama esse método dinamicamente. A concretização do método deve tratar das visitas do formulário, neste caso criando uma instância de *MessageFormVisit,* classe que representa a visita ao formulário e que advém diretamente do modelo de domínio (o prefixo *Message* é apenas adicionado para manter coerência com as restantes mensagens do módulo). Quando a mensagem é criada com os dados da visita é chamado o método *publish* do serviço que ira criar o produtor, passando a mensagem criada e qual o produtor a criar. O *ProducerService* cria então o *ProducerForm*, classe que é criada para produzir os eventos relacionados com formulário, chamando o método *publish* que tem o comportamento de adicionar a nova mensagem à fila.

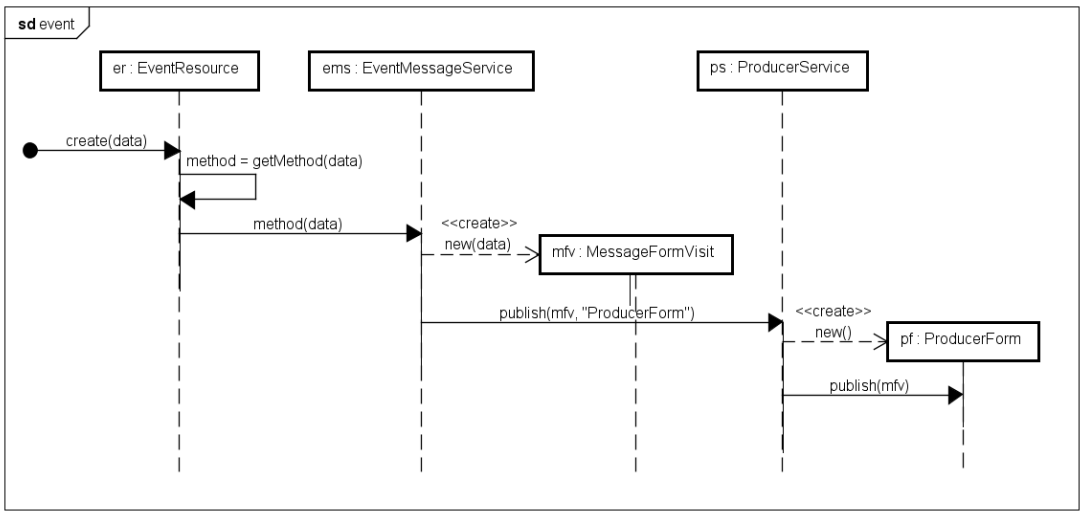

Figura 34 Diagrama de sequência: Produção do evento de vista

A mensagem que é adicionada à fila necessita de ser consumida, para isso é criada a classe *ConsumerForm* que recebe os eventos relacionados com formulários como indicado pela *Figura 35*. O consumidor deve registar a mensagem na base de dados, contudo não é responsabilidade do consumidor conectar à base de dados. No *services* os eventos são adicionados à base de dados através do módulo *Log,* basta para isso criar um serviço e repositório específico nesse módulo. O consumidor obtém os dados da visita através do *getBody* da mensagem, prepara o formato dos dados e passa-os para o *EgoiFormLogService* através do método *save*. Outra alternativa a usar o módulo *Log* seria passar ao módulo de formulários criado e criar um serviço para guardar as visitas, contudo esta decisão foi tomada tendo em vista manter a coerência com o registo dos restantes eventos. Um os requisitos do sistema é a geração de relatórios que devem conter informação das visitas. Se o relatório for gerado quando é requisitado pode tornar a sua consulta lenta. Portanto uma decisão tomada é a geração do relatório no momento da inserção, visto que esta é feita de forma assíncrona, não prejudica a experiência do utilizador final. O componente *services* contém um módulo para a geração de relatórios, o módulo *Report*, basta estender um serviço específico deste módulo e implementar um novo repositório. O módulo *Report* tem apenas a responsabilidade de receber os dados de um evento e guarda-lo na base de dados, contudo esse evento deve ir formatado conforme será guardado. O consumidor chamada então o método *save* do *EgoiFormReportService* passando os dados formatados. Quando este método é chamado o repositório criado no módulo de relatórios deve adicionar a nova visita ao relatório já existente. Quando for efetuada a consulta do relatório este apenas precisa de ser devolvido sem ser necessário outro tipo de operações.

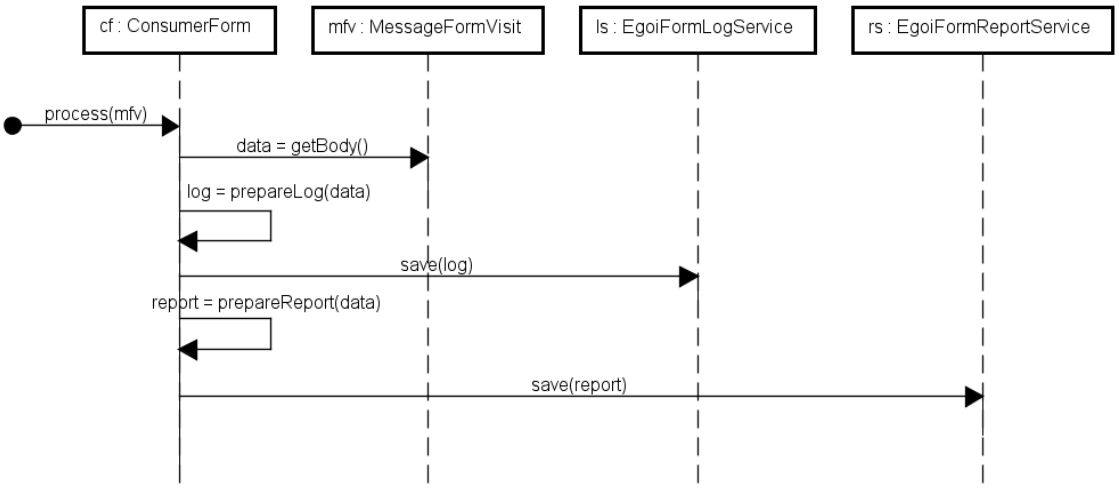

Figura 35 Diagrama de sequência: Consumo do evento de vista

Para os dados da submissão serem registados na base de dados é necessário criar um novo recurso para tratar as submissões no módulo de formulários do componente *services*, surge então o *SubmissionResource* para gerir o fluxo da submissão como representado na *Figura 36*. É requisito saber quando foi efetuada a submissão e quem a efetuou, no caso de se tratar de um contacto a submeter, para isso é criada a entidade *Submission* que representa a submissão, e são criadas também as classes responsáveis pelo seu tratamento. O *SubmissionResource* deve então chamar o método *save* do *SubmissionService* que chamará o mesmo método do repositório. Após criar a nova submissão é necessário guardar as respostas, a entidade de domínio que representa a resposta passa então a classe de *software* dando origem à classe *Answer*, esta tem a responsabilidade de guardar o valor da resposta, o campo e formulário a que pertence. É então chamado o método *save* do *AnswerService* passando os dados das repostas que agora incluem também o identificador da submissão. Este método chamará o respetivo método do repositório que deve criar as instâncias de *Answer* passando o valor da resposta, o campo e formulário a que pertence, toda esta informação presente na variável *data*.

Uma responsabilidade bastante importante da submissão é executar as ações de submissão associadas ao formulário. As ações podem ser definidas pelo utilizador ou ser dependentes do tipo de formulário. Atribui-se a responsabilidade de mandar executar as ações de submissão definidas pelo utilizador ao *SubmissionService*. Este serviço deve através do repositório recolher a lista de ações e executar os pedidos para cada ação. Na prática a ação corresponde a fazer um pedido a um URL definido pelo utilizador passando os novos dados da submissão. A responsabilidade de executar a ação dependente do tipo de formulário é atribuída ao próprio formulário. É criada uma *interface* de modelo, *FormModel*, que representa o comportamento variável dos formulários através do método *executeSubmissionActions*. Esta *interface* permite prevenir o sistema contra variações nos tipos de formulários dando uso ao padrão *Strategy*. A classe *FormService* é responsável por criar a classe de modelo concreta do formulário baseado no seu tipo. O método que executa a ação de submissão recebe a nova submissão com as suas respostas. No caso de um formulário de inscrição, por exemplo, este método deve adicionar como contacto da lista associada ao formulário os novos dados inseridos no formulário, utilizando para isso o módulo *Contacts* já existente nos *services*.

Finalizando a execução de todas as ações o *SubmissionResource* determina o sucesso ou insucesso da operação e devolve a resposta adequada.

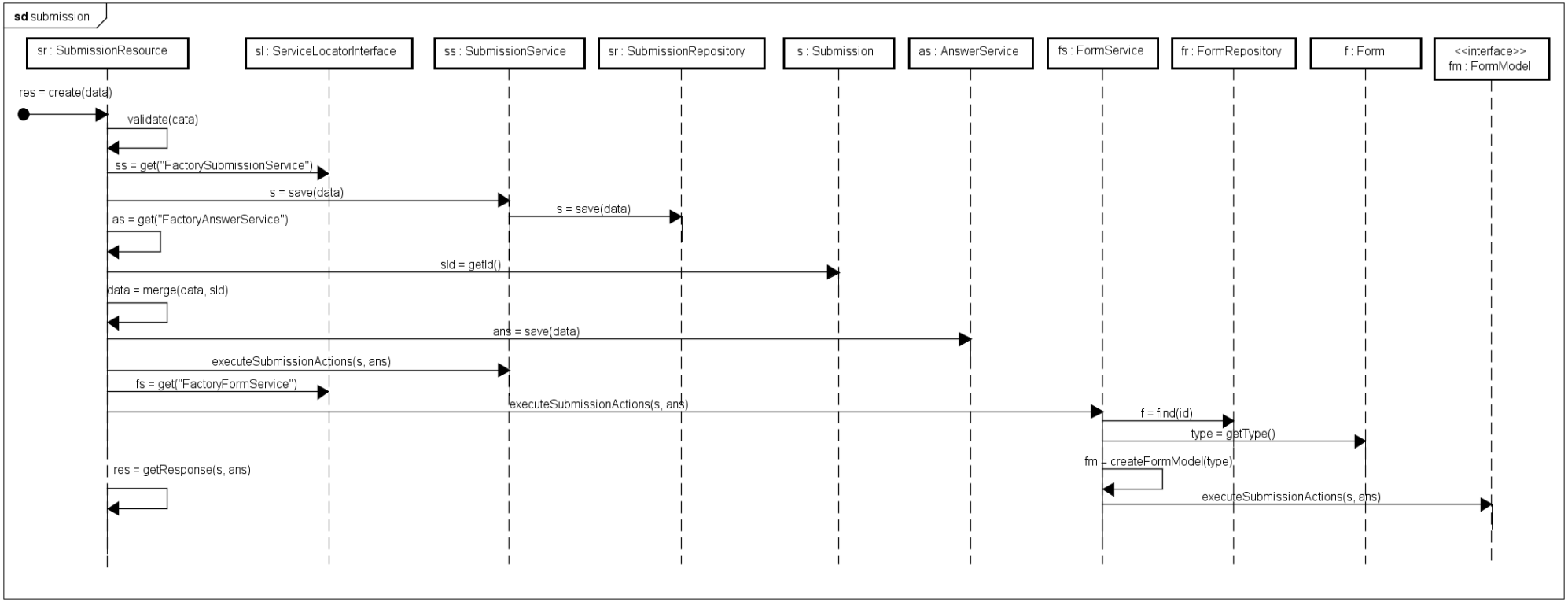

Figura 36 Diagrama de sequência: Recurso de submissão

5.4.3 Estrutura de Dados

A estrutura de dados identifica concretamente quais são os dados que se pretendem guardar e como estão divididos em entidades e são relacionados entre si [56]. A plataforma E-goi utiliza um modelo de dados relacional em que os dados estão estruturados em tabelas e relações, utilizando MySQL como sistema de gestão de base de dados. É um requisito não funcional do sistema que a estrutura de dados seja o mais desnormalizada possível para reduzir relações entre tabelas. A normalização de dados é um conjunto de regras que permite diminuir a redundância e duplicação de dados, para isso aumentando o número de tabelas e relações entre elas. O aumento de relações leva à utilização da operação de junção no momento da consulta, o que diminui a velocidade de processamento do sistema [57]. Já no momento de escrita a normalização faz com que esta seja mais eficiente visto que diminui a quantidade de dados. O contrário acontece quando uma estrutura de dados é desnormalizada, a consulta torna-se mais eficiente, mas a velocidade de escrita diminui devido à duplicação de dados. É necessário decidir a solução apropriada para o problema a resolver.

Esta secção documenta a solução de estrutura de dados desenvolvida assim como as decisões efetuadas no processo.

5.4.3.1 Solução

A *Figura 37* representa a solução final da estrutura de dados desenvolvida para o sistema proposto nesta dissertação. As tabelas criadas derivam diretamente das entidades de negócio e contêm os atributos necessários persistir. Não estão incluídas tabelas que não tenham sido desenhadas no âmbito deste projeto, ou seja, tabelas já existentes no E-goi como as contas, utilizadores, listas e contactos, não são apresentadas, mas são identificadas através de chaves estrangeiras.

Para perceber completamente a solução proposta é necessário conhecer um aspeto particular da tecnologia utilizada. É utilizada a tecnologia MariaDB, uma variação do MySQL. Esta tecnologia introduz o conceito de colunas dinâmicas [58] e métodos para a sua gestão. As colunas dinâmicas permitem guardar diferentes colunas em apenas um campo de uma linha através da utilização do tipo de dados BLOB. A funcionalidade obtida já era possível previamente à existência desta tecnologia, mas esta oferece funções para tratar automaticamente da gestão das colunas. As colunas dinâmicas são bastante úteis para guardar dados que possam ter diferentes atributos ou que possam variar ao longo do tempo, como

acontece com os tipos de formulários (cada tipo pode ter dados diferentes) ou com os diferentes tipos de campos que podem aumentar a qualquer altura. É importante que a estrutura de dados do E-goi seja alterada o menos possível, pois devido à forma como as máquinas que contêm a base de dados se encontram distribuídas (cf. 5.3.1) os custos de alteração da base de dados são elevados. Os custos de alteração envolvem a criação de *scripts* para efetuar a alteração na base de dados de cada cliente, caso exista um erro no *script* pode levar a consequências graves como a perda de dados do cliente. Conclui-se que a estrutura de dados deve ser alterada o menos possível e ser desenhada de uma forma que permita ser facilmente expandida e escalada. Esta dissertação propõe a utilização de colunas dinâmicas para manutenção da estrutura de dados protegendo esta de possíveis pontos de variação. Na prática sempre que existe um atributo de uma entidade em que os seus dados possam ser variáveis ou possam alterar ao longo do tempo, esse atributo pode ser guardado através de uma coluna dinâmica. Contudo, esta regra criada apenas funciona para atributos que não sejam pesquisáveis, visto que a consulta condicionada a colunas dinâmicas é de difícil execução. A implicação desta proposta é que seja deixado a cargo da camada de repositório, ou até ao componente que utiliza os serviços, a atribuição de semântica aos atributos consoante o tipo de dados a tratar. A vantagem desta solução é facilitar a expansão e manutenção da estrutura de dados protegendo-a contra possíveis variações sem necessitar de alteração. A desvantagem é que necessita de lógica extra para a utilização dos dados. Tendo em mente a utilização deste princípio criado é possível perceber a solução desenvolvida.

A maior parte das tabelas da solução desenvolvida contém os atributos *created*, *created_by, updated* e *updated_by*, que guardam respetivamente a data de criação da entidade, o utilizador que a criou, a data de atualização e o último utilizador que a atualizou. A tabela *form* guarda dados do formulário como o identificador da lista de contactos associada, o nome do formulário, tipo, estado, mensagem de submissão, código próprio do utilizador, ações de submissão, e opções variáveis consoante o tipo de formulário. As opções que variam consoante os tipos de formulário são guardadas através de colunas dinâmicas (o limite dos dados representado no diagrama não corresponde exatamente ao que está implementado para campos de maior dimensão). A tabela *field* representa a entidade campo e guarda atributos como o tipo do campo, propriedades genéricas de todos os campos, propriedades específicas de cada tipo de campo, através do BLOB *options* como coluna dinâmica, atributos relacionados com a condição entre campos, o campo da lista de contactos a que é mapeado, atributos relacionados com a ordem do campo no formulário, atributos que guardam lógica de validação e ainda o estilo

gráfico do campo. O estilo é guardado como coluna dinâmica pelo BLOB *styles* pois pode ser diferente para cada tipo de campo, essa diferenciação é feita pelo componente de visualização pois é esse componente que trata responsabilidades gráficas. A tabela *publish* guarda as publicações do formulário através do tipo de publicação, URL da publicação, estado, limitações e formulário a que pertence. A tabela *submission* representa as submissões do formulário principalmente através da data de submissão, a qual contém um conjunto de respostas que são registadas na tabela *answer* através dos dados da resposta, o campo que foi respondido e o formulário e submissão a que pertence. A tabela *notification* guarda o *e-mail,* data, formulário a que pertence e a mensagem a enviar. A tabela *visit_log* regista os dados das visitas como o identificador do contacto que fez a visita (encontra-se a *null* caso o visitante não tenha sido um contacto), lista e formulário a que pertence, dados da localização, sistema operativo, dispositivo, data e endereço IP. Cada tabela tem um repositório específico que efetua a sua gestão, no caso da tabela de *visit_log* esse repositório encontra-se no módulo *Log* (cf. 5.4.2.3). As tabelas *visit_count* e *answer_count* são utilizadas pelo módulo *Report* e servem para guardar os relatórios. À parte das chaves estrangeiras, das datas e do número de visitas total, todos os restantes atributos são colunas dinâmicas. Por exemplo, na tabela *visit_count* no atributo *by_browser* é guardado o número de visitas por cada um dos *browsers* identificados nas visitas. Esta tabela é alterada sempre que é feita a inserção de uma nova visita, mas de forma assíncrona, ou seja, o utilizador não é prejudicado no ponto de vista de desempenho. A existência destas tabelas em conjunto com a lógica de atualização das mesmas no momento da inserção de visitas ou respostas, torna a consulta de dados instantânea. Quando é preciso devolver as estatísticas de visitas de um formulário basta apenas aceder à tabela *visit_count* e devolver a linha correspondente ao *form_id*. Mesmo com estas otimizações de desenho, pode não ser possível devolver a totalidade dos dados derivado ao seu elevado volume. Para resolver esse problema é adicionado o atributo *partion_created* que, em conjunto com funcionalidades da tecnologia, permite particionar uma tabela segundo uma data, levando a que rapidamente possam ser devolvidas as visitas, por exemplo, num determinado ano, mês, semana ou dia.

A solução de estrutura de dados alcançada identifica possíveis pontos de variação dos dados e é construída com objetivo de ser facilmente expandida, otimizando a consulta onde existe um grande volume de dados. Contudo existe sempre a necessidade de alterações caso sejam adicionadas novas funcionalidades, assim como alternativas a esta solução.

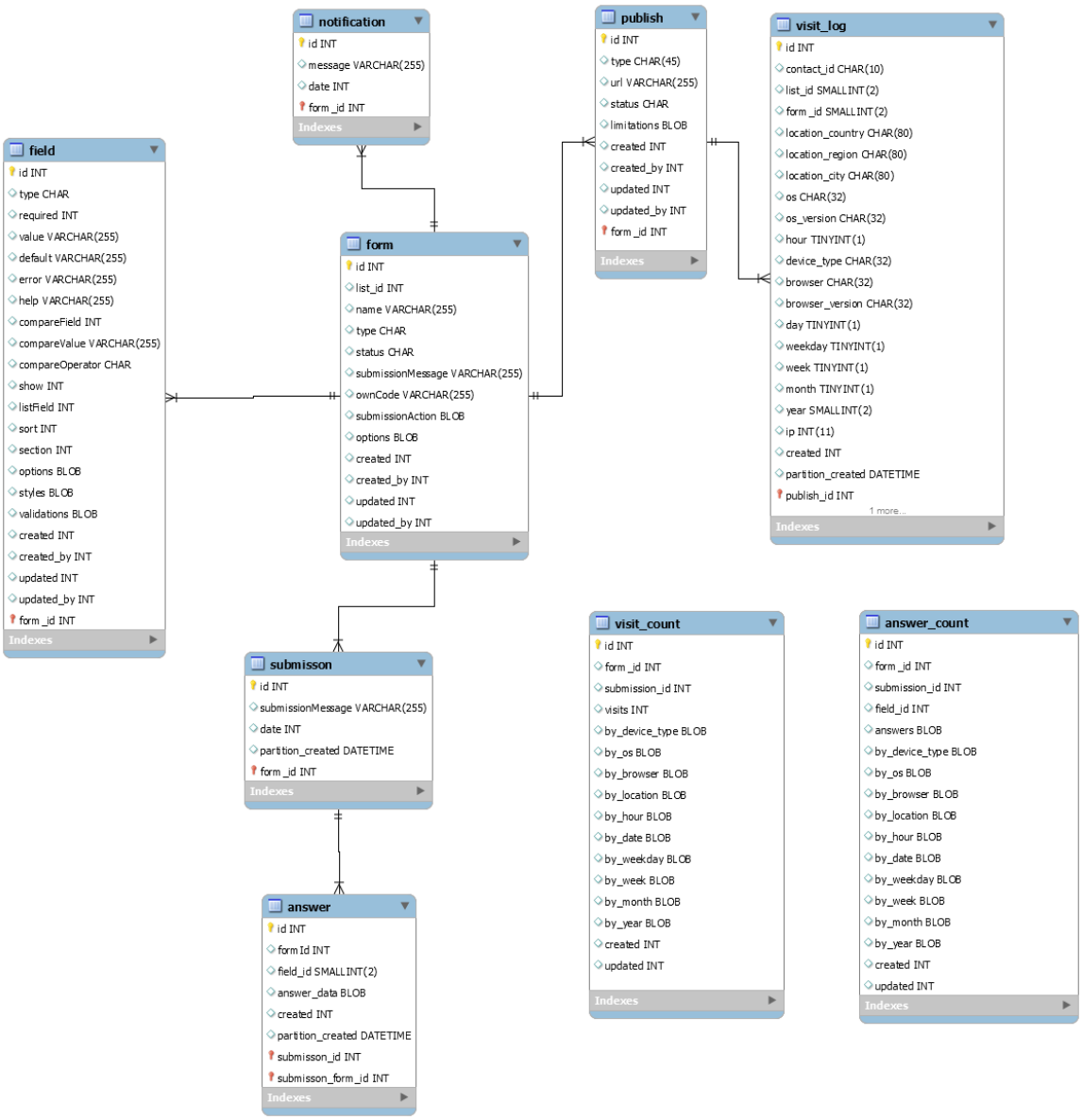

Figura 37 Estrutura de dados

5.4.3.2 Alternativas

A *Figura 38* representa uma estrutura de dados alternativa à solução proposta. Esta alternativa foi ponderada antes da própria solução, pois segue um modelo mais orientado ao domínio e entidades de negócio. É também uma estrutura bastante mais normalizada, em comparação com a outra solução, o que indica que não há tanta duplicação de dados, contudo possui um pior desempenho quanto à consulta de dados. Esta alternativa permite guardar exatamente o mesmo conteúdo, apenas são dividas algumas tabelas em várias e criadas outras. É criada uma tabela por cada tipo de formulário e são adicionados atributos específicos do tipo em vez de recorrer à utilização de colunas dinâmicas. É também criada uma tabela para as ações de

submissão, *SubmissionAction*, não precisando de replicar ações, mas apenas associa-las. É também partido o atributo que guarda os estilos dos campos numa tabela, porém mantendo a utilização de colunas dinâmicas visto ser um campo bastante variável, mesmo assim para campos com estilos iguais é possível reutilizar a entrada da tabela. É adicionada uma tabela para manter a ordem dos campos no formulário e separados os dados de condicionamento lógico entre campos na tabela *ConditionalLogic*. Não são representadas apenas as tabelas de relatórios. Para além disso a estrutura podia ser ainda mais normalizada, por exemplo separando o atributo limitações da publicação para uma tabela com os campos de limitação específicos.

Esta solução não é adotada devido a não ser otimizada para desempenho e não seguir os requisitos e necessidades da plataforma E-goi. É também uma solução mais difícil de expandir sempre que se pretende adicionar novos tipos de dados, por exemplo se for adicionado um novo tipo de formulário é necessário criar uma nova tabela, o que na arquitetura E-goi obrigaria a executar um *script* para cada cliente.

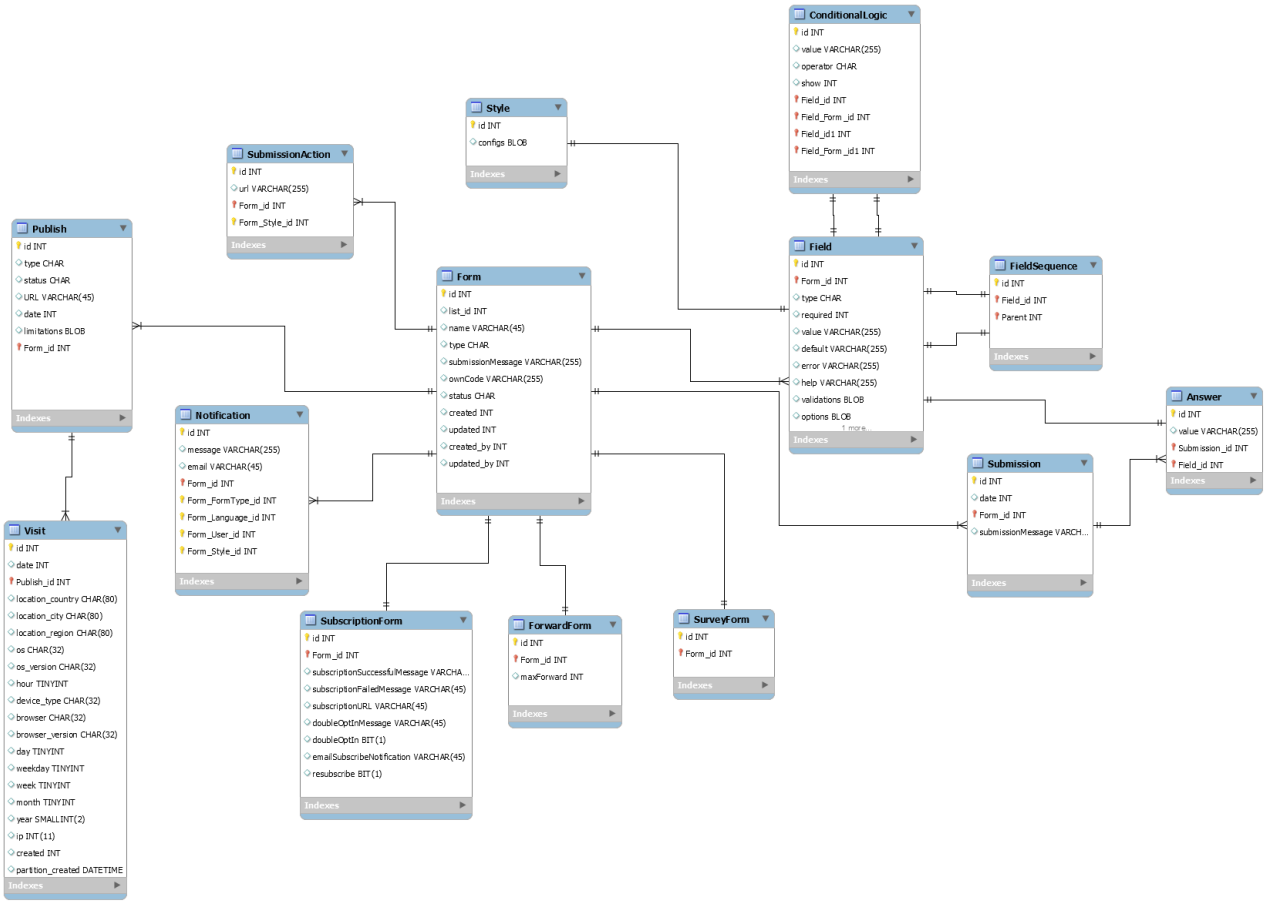

Figura 38 Estrutura de dados alternativa

Considerando o reduzido grau de normalização da solução desenhada, outra possível alternativa seria a utilização de uma base de dados não relacional [59]. As bases de dados não relacionais não modelam os dados com tabelas mas com objetos, por exemplo. Este tipo de dados permite escalar facilmente o número de máquinas quando necessário aumentar o armazenamento, que é algo que a estrutura atual do E-goi esta a ir ao encontro. As bases de dados não relacionais são ideais onde não se conhece concretamente todos os atributos possíveis ou onde estes podem variar frequentemente. Contudo estas soluções também podem comprometer a consistência dos dados pois existe bastante duplicação destes. Outras qualidades como desempenho e disponibilidade teriam de ser avaliadas e comparadas para este caso específico. Porém esta possível alternativa não foi concretamente avaliada pois não segue os requisitos tecnológicos impostos pela plataforma E-goi, já que esta utiliza uma base de dados relacional e existem dependências de dados no sistema de formulários. Para este tipo de estrutura de dados ser utilizada no sistema seria necessário redesenhar e alterar toda a plataforma E-goi o que não é objeto de estudo desta dissertação.

5.4.4 Integração com o Piwik

O Piwik é uma plataforma código-aberto de *web analytics* que permite seguir visitas a páginas *web* e obter informação acerca dos visitantes. A plataforma pode ser hospedada na *cloud*, ou num servidor próprio dependendo apenas das tecnologias PHP e MySQL. O Piwik disponibiliza uma API REST para aceder a relatórios de visitas ou um conjunto de clientes em diversas linguagens de programação. Nesta secção é apresentada a arquitetura da plataforma, os dados principais que disponibiliza, e como é utilizada no sistema de formulários.

5.4.4.1 Arquitetura

O Piwik está estruturado em dois componentes principais, o componente *Core* e o componente *Plugins*, como indicado na *Figura 39*. O componente *Core* representa a base da plataforma, assim como pontos de extensão que são utilizados pelo componente *Plugins* para adicionar funcionalidades à aplicação. Muitas das funcionalidades principais da plataforma são implementadas através do componente *Plugins*.

Entre os diversos componentes integrantes do *Core* destacam-se os seguintes componentes:

- ***HTTP Reporting API:*** fornece serviços para aceder aos relatórios;

- ***HTTP Tracking API:*** permite inserir dados sobre os visitantes, visitas e as ações efetuadas;
- ***Archive:*** permite gerar arquivos usando todos os dados inseridos, o que é necessário para ser possível aceder a informação em tempo real devido à quantidade de dados que podem ser requisitados por um ou vários utilizadores em simultâneo;
- ***DataAccess:*** fornece uma forma de agregar e representar os dados completos para auxiliar a geração de arquivos;
- ***Plugin***: é o responsável por fornecer os pontos de extensão da plataforma.

Os pontos de extensão são implementados no componente *Plugins*, é neste componente que grande parte das funcionalidades estão definidas. Estendendo a classe *API,* existente neste componente, o *plugin* pode gerir e aceder aos seus próprios relatórios. Estendendo a classe *RequestProcessor* é possível alterar a forma como as visitas são rastreadas. Para definir como os dados são arquivados é necessário estender a classe *Archiver*. A classe *Controller* permite disponibilizar novos métodos através da API. Existem muitas outras formas de expandir as funcionalidades desta plataforma sem ser necessário alterar o código existente.

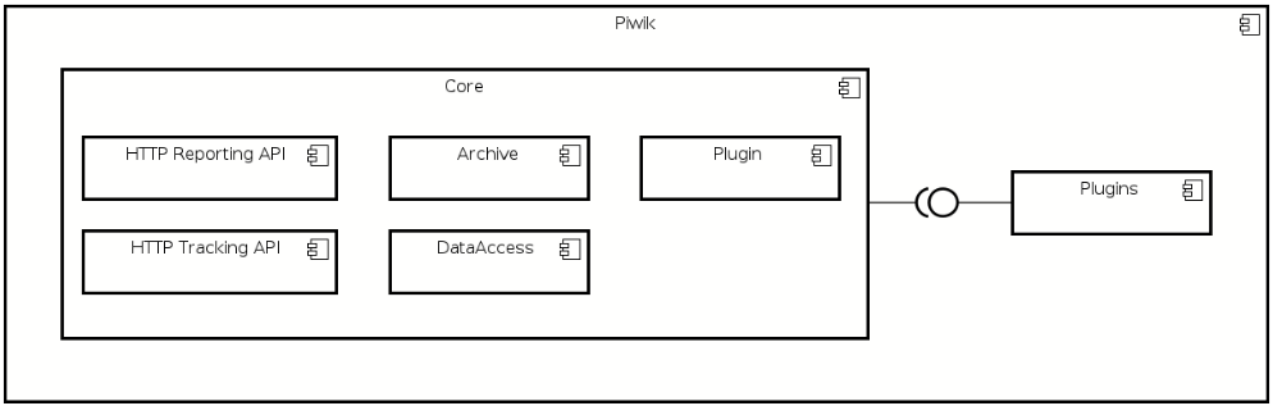

Figura 39 Arquitetura do Piwik

5.4.4.2 Estrutura de Dados

O Piwik permite guardar e manipular diversa informação, a *Figura 40* representa parte da estrutura de dados através de algumas das tabelas que são utilizadas para registar os dados. As tabelas mais importantes incluem:

- ***piwik_log_visit***: contém uma entrada por visita e guarda informação da mesma (e.g. identificador do visitante, localização, data da visita, ações efetuadas durante a visita, dados sobre as tecnologias utilizadas, fontes de tráfego, …);
- ***piwik_log_action***: contém as ações possíveis de executar na página. Uma ação é um evento que leva à mudança de um estado e pode ser identificada através do seu URL destino (e.g. *click* num botão);
- ***piwik_log_link_visit_action***: regista as ações que ocorreram durante uma visita específica;
- ***piwik_goal***: regista os objetivos de negócio definidos (e.g. 1000 visitas atingidas);
- ***piwik_log_conversion:*** regista as ações que atingiram objetivos;
- ***piwik_site:*** armazena as páginas que estão a ser rastreadas pela plataforma.

Existem ainda outras tabelas que não são representadas aqui, qualquer *plugin* pode adicionar tabelas à estrutura de dados. O *plugin CustomDimensions* adiciona uma tabela onde são armazenadas dimensões personalizadas através da sua identificação, nome, tipo e estado. Uma dimensão é essencialmente um novo atributo a ser registado. Cada vez que ocorre uma visita é adicionada numa nova coluna da tabela *log_visit* os dados da dimensão criada utilizados naquela visita.

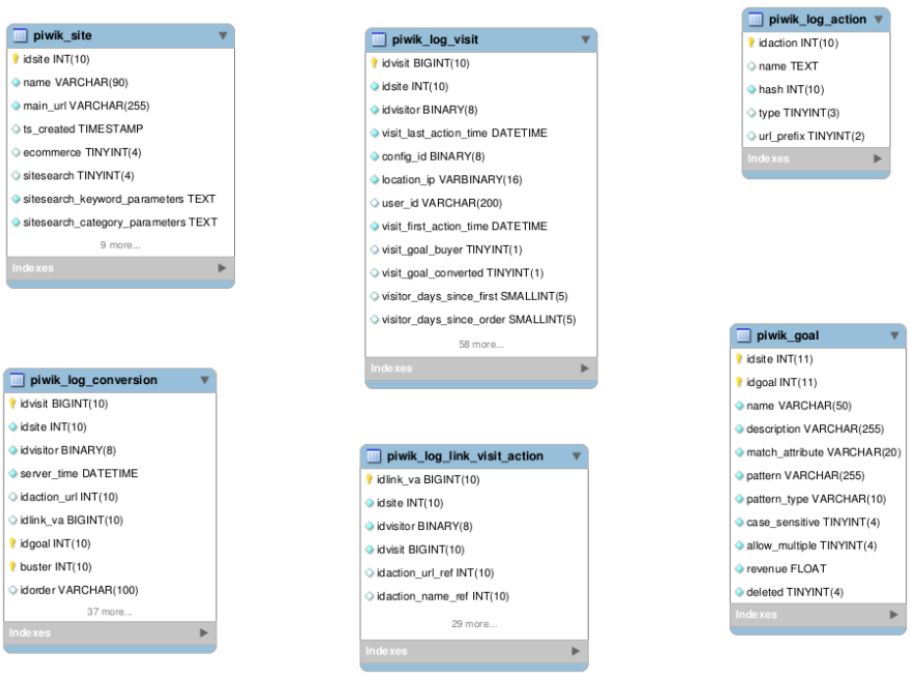

Figura 40 Estrutura de dados do Piwik

A estrutura de dados do apresentada não esta otimizada para a extração de dados quando o tráfego é elevado, o que leva a problemas de desempenho. Para resolver esse problema os desenvolvedores do Piwik utilizam o conceito de arquivos. Os arquivos são tabelas geradas segundo um intervalo de tempo. Os dados inseridos nessas tabelas são organizados por data (dia, semana, mês, ano ou um período personalizado), mas incluem dados diários. Quando é necessário selecionar dados de um mês ou de um ano estes são agrupados através dos arquivos diários. Os dados arquivados podem ser métricas numéricas ou dados bidimensionais. Por exemplo, pode ser guardado num arquivo o número de visitas para uma determinada semana, assim como o número de visitas por dia dessa semana, por dispositivo utilizado ou por qualquer outra dimensão. Com a utilização dos arquivos é possível consultar dados em tempo real. A utilização desta abordagem só é problemática quando os dados arquivar também são em elevado volume, para tal é necessário gerar o arquivo com a frequência necessário pelo problema específico a tratar.

A utilização do Piwik permite armazenar mais dados do que o sistema simples de gestão de visitas utilizado nos serviços do E-goi. Esta plataforma permite identificar concretamente o visitante, enumerar cada visita assim como as ações efetuadas nesta e os objetivos de negócio atingidos. Por esse motivo é benéfica a utilização desta plataforma no sistema de formulários.

5.4.4.3 Piwik no Sistema de Formulários

Quando o Piwik é instalado na máquina onde vai executar são definidos os dados de administrador de acesso ao sistema, os dados de aceso à base de dados, e informação do primeiro *site* a rastear que inclui o seu nome, URL e fuso horário. Para começar a rastear o *site* é necessário inserir código Javascript nas páginas a analisar. Todos os dados e estatísticas estão disponíveis através de uma *interface web* ou móvel disponibilizada pelo Piwik, assim como através da API. Para utilizar o Piwik no sistema de formulários é necessário criar apenas um *site* para representar o sistema e adicionar novas dimensões, para guardar os dados específicos dos formulários. Os dados específicos do sistema de formulários a guardar por visita são o identificador do formulário, o identificador da publicação, o identificador do utilizador e o identificador do contacto (no caso do visitante ser um contacto). Portanto para cada um desses dados é necessário criar uma dimensão, assim quando for inserida uma nova visita é possível identificar a que formulário e publicação ela pertence. As novas dimensões podem ser

adicionadas através de um novo *plugin*, caso seja necessário alterar comportamentos de arquivo de dados, rastreamento ou relatórios. As dimensões também podem ser adicionadas através do *plugin CustomDimensions* para criar dimensões personalizadas, caso não sejam necessários alterar comportamentos no fluxo da aplicação. Como não existe a necessidade de alterar comportamentos basta apenas utilizar este *plugin* para inserir as novas dimensões. As dimensões podem ser inseridas através da *interface web* disponibilizada pelo Piwik ou através da API, as quais podem ser ativadas ou desativadas a qualquer altura.

No sistema de formulários o Piwik é utilizado pelo componente *form-builder* (cf. 5.3.2). Este componente deve inserir código Javascript disponibilizado pelo Piwik quando construir o HTML do formulário antes de o enviar para publicação. Este código Javascript deve incluir o identificador do *site* criado, o identificador do formulário e publicação, e chamar o pedido de inserção de uma nova visita da API do Piwik, que é depois chamado quando um formulário é carregado. Para mostrar os dados armazenados basta chamar o método responsável por tal da API do Piwik.

Para além dos dados apresentados esta plataforma inclui ainda um módulo específico para formulários. Este módulo guarda dados como campos menos preenchidos do formulário, tempo de preenchimento de cada campo, taxa de retenção por campo, quantidade de texto por campo, entre outros. Com a simples aquisição de um módulo e configuração de uma plataforma são adicionados ao sistema imensos dados úteis que podem ser disponibilizados ao utilizador.

5.5 Implementação

A implementação utiliza os artefactos produzidos pela disciplina de *design* para produzir o código necessário do sistema. Nesta secção são apresentados apenas alguns detalhes do código implementado, dando a conhecer a solução desenvolvida nos diversos componentes do sistema. Alguns dos métodos apresentados encontram-se abreviados para apresentar apenas a lógica principal, escondendo certos pormenores ou secções repetitivas. Esta secção está dividida segundo alguns dos requisitos e camadas onde o código apresentado é utilizado.

5.5.1 User Stories

Esta secção apresenta alguns detalhes de implementação para as *user stories* definidas (cf. 5.1.3).

5.5.1.1 US01 – Criar Formulário

Os serviços desenvolvidos no componente *services* começam o seu fluxo na classe *Resource* (controlador) do serviço específico que é chamado. Por exemplo, quando é efetuado um pedido para o URL *services.e-goi.com/api/forms* através do método HTTP POST é chamada a função *create* da classe *FormResource* passando como parâmetro os dados presentes no corpo do pedido (este comportamento é definido pela tecnologia Zend [60]). O *Extrato de código 1* apresenta o método *create* da classe *FormResource*. Este método começa por validar o formato dos dados que foram recebidos, criado o filtro de validação através do método *validateInputPost* e verificado se este é válido através do método *isValid*. Caso os dados sejam inválidos é devolvido o erro *UnprocessableEntity* como resposta utilizando o método *response UnprocessableEntity* definido na classe pai. Caso os dados sejam válidos é chamado o método *save* do *FormService* passando os dados. Este serviço é obtido pelo método *get* de uma implementação de *ServiceLocatorInterface* que é injetada no controlador no momento da sua criação. Por fim o controlador verifica o que foi devolvido pelo método *save*, caso tenha sido devolvida uma exceção é retornada uma resposta de erro, caso contrário é retornada uma resposta de sucesso. Todos os métodos que sejam estendidos do *EgoiAbstractRestfulController* funcionam de forma similar, utilizando as funcionalidades da classe pai para gestão enviar a resposta, utilizando o *InputFilter* para validação dos dados e as funcionalidades da camada de serviço para executar as operações de negócio.

```php
public function create($data)
{
    $inputFilter = $this->validateInputPost();
    $inputFilter->setData($data);

    if (!$inputFilter->isValid()) {
        $messages = ['validation_messages'
         => $inputFilter->getMessages()];
        return $this->responseUnprocessableEntity($messages);
    }

    $entity = $this->getServiceLocator()->
         get('FactoryFormService')->save($data);

    return $entity instanceof Exception ?
        $this->responseInternalError($entity) :
        $this->responseSuccess([self::RESPONSE_SUCCESS_MESSAGE]);
}
```

Extrato de código 1 Método create da classe FormResource

O *Extrato de código 2* apresenta parte da lógica de validação contida no método *validateInputPost*. Neste método é criado um filtro, instância de *InputFilter*, através da classe *InputFactory*, esta recebe os filtros a serem passados à instância a criar. Os filtros são definidos num *array* para cada parâmetro que é necessário validar. No exemplo apresentado está definido um filtro para o campo *name* do formulário, onde são passados filtros já existentes na chave *filters,* e novos filtros definidos através da chave *validators*. Neste caso é criada a classe *FormValidator* no componente *Validator* para fazer as validações específicas do formulário. Esta classe implementa o método *isValid* que recebe os dados e faz a validações específicas de negócio. Para definir mais campos a validar é só adicionar ao *array* conforme o campo apresentado.

```
protected function validateInputPost(array $options = [])
{
    $factory = new InputFactory();
    $filters = [
        [
            'name'        => 'name',
            'required'    => true,
            'filters'     => [['name' => 'StripTags'], ['name'
                             => 'StringTrim']],
            'validators' => [
                [
                    'name'    => 'Form\Validator\FormValidator',
                    'options' => ['serviceLocator'
                          => $this->getServiceLocator()]
                ],
            ]
        ]
    ];

    return $factory->createInputFilter($filters);
}
```

Extrato de código 2 Exemplo de lógica de validação

O método *save* da classe *FormService* chama o método *save* do *FormRepository* representado no *Extrato de código 3*. Considerando que a grande responsabilidade do componente *services* é funcionar como intermediário entre a base de dados e agregador da lógica de negócio, grandes das responsabilidades deste componente estão na camada *Repository* e por isso é onde ocorrem também as principais variações de lógica do sistema. Contudo os métodos do repositório também se encontram estruturados de forma semelhante. O método *save* começa por criar uma instância da entidade a persistir através do seu nome, como é criado um repositório por entidade, este sabe qual é a entidade de que trata. No caso apresentado é criada uma instância de *Form*, classe que representa os dados do formulário. Em seguida são adicionados dados à instância que não vêm nos dados recebidos, ou que devem ser inicializados no momento da criação. Por exemplo, é definido o estado do formulário como *inactive* pois este acabou de ser criado, no repositório de publicação este eventualmente é definido como *active*. São inicializados também outros dados como a data de criação da entidade e o utilizador que a criou, para isso é utilizado o método *getReference* da classe que representa a conexão à base de dados para obter a referência para o utilizador que está conectado, ou seja, o utilizador que criou a entidade. O *services* utiliza a tecnologia Doctrine ORM para fazer o mapeamento objeto-relacional, para isso é definido nas entidades a correspondência entre os seus atributos e os campos da tabela em que é guardada. Também são utilizadas as funcionalidades desta tecnologia para aceder à base de dados. A etapa final do repositório é persistir a entidade que

foi criada ou manipulada, para isso cria o objeto *DoctrineObject* que através do método *hydrate* preenche os atributos da entidade com os dados recebidos e que foram validados previamente. Utilizando as funcionalidades da classe *EntityManager* presente na variável *connection* a entidade é finalmente persistida na base de dados.

```
public function save(array $data, $id = null)
{
    $entityName = $this->entity();
    $entity = new $entityName();

    $entity->setStatus(Form::INACTIVE_STATUS);
    $entity->setCreated(time());
    $entity->setCreatedBy($this->connection
        ->getReference(self::USER_ENTITY_NAME, $this->user()));
    $entity->setUpdatedBy($this->connection
        ->getReference(self::USER_ENTITY_NAME, 0));

    $hydrator = new DoctrineObject($this->connection);
    $entity = $hydrator->hydrate($data, $entity);
    $this->connection->persist($entity);
    $this->connection->flush();

    return $entity;
}
```

Extrato de código 3 Método save do FormRepository

5.5.1.2 US04 – Publicar Formulário

A lógica de publicação do formulário ocorre principalmente no componente *egoi-public*. Apenas a persistência das entidades de publicação acontece no *services*. Apesar de ser um componente diferente a tecnologia utilizada é a mesma e por isso o fluxo também é semelhante. Quando o pedido de publicação é recebido no *egoi-public* é necessário criar na máquina os ficheiros associados ao formulário, que incluem um ficheiro com as limitações daquela publicação (e.g. máximo de visitas), um ficheiro com as contagens (e.g. visitas atingidas) e o ficheiro com o conteúdo HTML do formulário. A criação dos ficheiros com as contagens e limitações é feita para não ser necessário aceder à base de dados quando ocorre uma nova visita no formulário, podendo assim ser feito imediatamente na máquina onde o formulário está publicado. O *Extrato de código 4* mostra a criação do ficheiro com o conteúdo HTML do formulário. Este ficheiro é criado numa pasta pública, definida pelo componente em que qualquer um pode aceder, dentro dessa pasta existe outra pasta específica para o conteúdo do formulário. O nome do ficheiro do formulário é uma *hash* gerada que também é utilizada no URL do formulário. A lógica de geração da *hash* não pode ser demonstrada nesta dissertação por pedido de confidencialidade do E-goi, mas envolve a utilização dos identificadores do utilizador, do formulário e da publicação, garantindo a sua unicidade.

```
private function publishContent($moduleConfig, $form) {
    file_put_contents(PUBLIC_PATH . $moduleConfig['forms'] .
        $moduleConfig['content'] . $form['hash']
        .'.html', $form['html']);
}
```

Extrato de código 4 Publicação do conteúdo do formulário no egoi-public

5.5.1.3 US10 – Submeter Formulário

O fluxo de submissão do de um formulário começa com a adição de uma nova visita após este ser disponibilizado ao utilizador. Para o formulário ser disponibilizado é verificado se nenhuma limitação foi atingida utilizando os ficheiros criados previamente da camada pública. Quando é inserida uma nova visita esta necessita de ser adicionada a esses ficheiros. Um pedido de nova visita chega à camada pública e é tratado pelo método *newVisit* da classe *PublishForm* como representado pelo *Extrato de código 5*. O método começa por ler o ficheiro de contagens correspondente ao formulário, que se encontra na pasta definida nas configurações, e que tem no nome a *hash* gerada. A *hash* está presente no URL que é acedido o que permite distinguir qual o formulário a tratar. O ficheiro JSON de contagens é então lido para um *array,* após isso é adicionada a nova visita. Por fim é guardado o ficheiro novamente, o que corresponde à sua substituição pelos dados atualizados.

```
public function newVisit($moduleConfig, $hash, $visit) {
    $filename = PUBLIC_PATH . $moduleConfig['forms'] . $moduleConfig['counts']
        . $hash .'.json';
    $counts = $this->jsonFileToArray($filename);

    $counts['visits'] = $counts['visits'] + $visit['visits'];
    $this->updateCounts($moduleConfig, $hash, $counts);
}
```

Extrato de código 5 Atualização das visitas no egoi-public

Para além da visita ser atualizada na camada pública esta necessita de ser inserida na base de dados. Existe a necessidade então de fazer um pedido ao componente de serviços para inserir a visita. Como o fluxo de visitas pode ser elevado é necessário utilizar um sistema de gestão de filas para garantir que nenhuma vista é perdida ou descartada enquanto estiver a ser inserida outra visita na base de dados. O *services* inclui um módulo para tratar deste tipo de eventos, o módulo *QueueMessageService.* Este módulo recebe o evento e adiciona-o a uma fila através da tecnologia ActiveMQ [61]. É criada uma classe neste módulo que é responsável por produzir, para a fila, eventos específicos de formulários estendendo as classes já existentes para tal. O verdadeiro processamento do evento acontece quando este chega à classe que consome os

eventos do formulário. O *Extrato de código 6* apresenta o método *process* que é chamado pela tecnologia quando ocorre ao evento associado à classe. O *process* valida a estrutura da mensagem e chama dinamicamente um método para tratar do evento específico que ocorreu. O nome do método é construído a partir do nome do evento que é recebido na mensagem. Desta forma quando é preciso adicionar um novo evento basta adicionar um método para o tratar.

```
public function process(Frame $message)
{
    $data = Json::decode($message->getBody(), Json::TYPE_ARRAY);

    if ($this->isValidStructureMessage($data)) {
        $method = $this->makeMethodCall($data['event']);
        $this->$method($data);
    }
}
```

Extrato de código 6 Método process do consumidor de eventos

No caso do evento de visita a um formulário é chamado o método *makeFormVisit* representado no *Extrato de código 7*. Este método é responsável por formatar os dados recebidos e chamar as ações necessárias executar na visita. É chamado o método *processFormReport* para processar o relatório das visitas e o método *processFormEventLog* para chamar o serviço do módulo *Log* que ira persistir a visita na base de dados. Este módulo guarda as entidades relacionadas com eventos como é o caso das visitas.

```
public function makeFormVisit($data)
{
    $data['eventData']['date'] = $this->makeDate($data['eventData']['date']);

    $reportEntity = $this->processFormReport(
        array_merge(
            [
                'form_id' => $data['eventData']['formId'],
                'visits' => $data['eventData']['stats']['visits']
            ],
            $this->makeFormDynamicColumnValue($data)
        ),
        $data['eventData']['formId'],
        $data['clientId'],
        $data['serverId']
    );

    $this->processFormEventLog($data);
}
```

Extrato de código 7 Processamento do evento de visita

O *processFormReport*, *Extrato de código 8*, processa o relatório do formulário. Na prática basta apenas persistir o relatório através do módulo *Reports* pois este já se encontra formatado em colunas dinâmicas. É utilizado a *ServiceLocatorInterface* para obter uma instância do serviço de

relatórios. O relatório é guardado através do método *save* do *FormCountService* que por sua ve chama o método correspondente do *FormCountRepository*. O repositório antes guardar o novo relatório de visitas faz a soma com o relatório já existente de visitas para aquele formulário como será indicado na secção 5.5.2.

```
protected function processFormReport(array $data, $formId, $clientId,
$serverId)
{
    $service = $this->getServiceLocator()->get('FactoryService')
         ->makeService(
        '\Reports\Repository\FormCountRepository',
        '\Reports\Service\FormCountService',
        false,
        $clientId,
        $serverId
    );

    $reportData = $service->findReport($formId);
    return $service->save($data,
         !empty($reportData) ? $formId : null);
}
```

Extrato de código 8 Processamento do relatório do formulário

O método *makeFormDynamicColumnValue, Extrato de código 9*, apresenta parte da formatação do relatório no formato necessário às colunas dinâmicas. É apresentado apenas para os dados relacionados com a localização, os restantes atributos seguem um formato semelhante. É guardada uma estrutura em formato de árvore contendo uma lista de países, uma lista de cidades para cada país e uma lista de regiões por cada cidade. O número de visitas é apresentada para cada uma das regiões como mostra o exemplo (e.g. *{'Portugal' : {'Porto' : {'Matosinhos' : 10, 'Campanhã' : 2}}}*). Para obter as visitas de uma cidade soma-se as visitas das regiões dessa cidade. Para obter as visitas de um país soma-se as visitas das cidades desse país. Os dados dos restantes campos dinâmicos são apresentados no relatório de forma semelhante. Para além deste tipo de dados também existem dados simplesmente numéricos que são guardados apenas através do seu valor.

```php
protected function makeFormDynamicColumnValue(array $data)
{
    $stats = $this
          ->makeStatsDynamicColumn($data['eventData']['stats']);

    if (! empty($data['eventData']['location']['country'])
        && ! empty($data['eventData']['location']['city'])
        && ! empty($data['eventData']['location']['region'])) {
        $newData['by_location'] = [
            $data['eventData']['location']['country'] => [
                $data['eventData']['location']['city'] => [
                    $data['eventData']['location']['region'] => $stats
                ]
            ]
        ];
    }
    return $newData;
}
```

Extrato de código 9 Criação das colunas dinâmicas do relatório de visitas

Depois de processada a visita, o utilizador preenche os campos do formulário e submete-o. A gestão das submissões e respostas das submissões é tratada por um recurso, e entidades associadas, como nos restantes serviços do sistema. Uma das etapas do fluxo de submissão é a execução das ações de submissão do formulário. É criada a *interface FormModel*, para representar este comportamento. Esta *interface* é implementada pelas classes que representam os tipos específicos de formulário. Existem tipos de formulário que podem ter ações comuns, por exemplo a execução dos automatismos associados a uma conta E-goi tem de ser despoletada quando ocorre uma submissão para qualquer formulário. Para evitar duplicação de ações e facilitar a adição de ações de submissão é criada a classe *SubmissionAction* que define o método *execute*. Cada ação de submissão deve estender esta classe. A associação entre o tipo de formulário e a lista de ações de submissão a executar para esse tipo de formulário, é feita através das configurações do módulo. O *Extrato de código 10* apresenta o método *executeSubmissionActions* definido na *interface FormModel* e implementado por um dos tipos de formulário. Este método começa por formatar os dados conforme necessário para aquele formulário. Depois são obtidas todas as ações associadas ao formulário através das configurações, utilizando o método *getSubmissionActions* que recebe o nome da classe. Finalmente são executadas todas as ações de submissão obtidas passando os dados formatados.

```
protected function executeSubmissionActions($submission, $answers) {
    $data = $this->buildData($submission, $answers);
    $actions = $this->getSubmissionActions(get_class($this));

    foreach ($actions as $submissionAction)
        $submissionAction->execute($data);
}
```

Extrato de código 10 Processamento das ações de submissão

O *Extrato de código 11* mostra a execução de uma ação de submissão, neste caso a ação de adição de novos contactos que é necessária nos formulários de inscrição. Este método simplesmente trata de formatar os dados dos contactos, filtrando dados desnecessários das respostas e submissões, passando esses dados para serem guardar pelo módulo de contactos, chamando o serviço desse *ContactService* desse módulo. Mais uma vez é efetuada a comunicação entre módulos do *services* através da camada de serviços.

```
protected function execute($data) {
    $data = $this->buildContactData($data);
    $service = $this->getServiceLocator()->get('FactoryService')
         ->makeService(
        '\Contacts\Repository\ContactRepository',
        '\Contacts\Service\ContactService',
        false,
        $data['clientId'],
        $data['serverId']
    );

    return $service->save($data);
}
```

Extrato de código 11 Método execute da ação de submissão que insere os novos contactos

5.5.2 Estrutura de Dados

O código que trata de aceder à base de dados e manipular dados está todo contido na camada *Repository*. A maior parte desse código é similar ao apresentado em 5.5.1, pois simplesmente trata de persistir ou manipular entidades. Nesta secção são apresentados alguns pormenores da manipulação de colunas dinâmicas visto que a tecnologia Doctrine não oferece funcionalidades para tal o que leva à necessidade de o fazer utilizando SQL.

O *Extrato de código 12* apresenta o método *create,* do repositório de relatórios do formulário, que é chamado pelo método *save* quando ocorre a inserção de um novo relatório. Este método tem os comportamentos de construir e executar o SQL de inserção do relatório. Primeiro são construídas as colunas e os valores dos campos numéricos, que estão presentes no *array mappingStatsColumns*. Na coluna de visitas esta tem o nome *visits* e o valor é o número de visitas, ou 0 caso não existam visitas adicionadas naquele relatório. Em segundo lugar são

construídas as colunas e valores dos campos dinâmicos. No caso do campo de localização a coluna tem o nome de *by_location* e o valor tem o formato devolvido pelo método *buildDynamicColumnInsert*.

```
public function create(array $data) {
    $sql = 'INSERT INTO egoi_form_count';
    $columns = '(form_id, created';
    $values = ' VALUES(' . $data['form_id'] . ', ' . time();

    foreach ($this->mappingStatsColumns as $key => $value) {
       $columns .= ', ' . $key;
       $values .= ', ' . (!isset($data[$key]) ? 0 : $data[$key]);
    }

    foreach ($data as $key => $value) {
        $columns .= ', ' . $key;
        $values .= ', ' . $this->buildDynamicColumnInsert($value);
    }
    $sql .= '))';

    // execute sql

    return $lastInsertId;
}
```

Extrato de código 12 Inserção de relatório com colunas dinâmicas

O método *buildDynamicColumnInsert*, *Extrato de código 13*, devolve o SQL para o valor da *query* de inserção de uma coluna dinâmica. A tecnologia MariaDB oferece a função *COLUMN_CREATE* para criar colunas dinâmicas, na função é passado o nome da coluna e o valor da mesma. Para criar colunas dinâmicas em que o valor de uma coluna é outra coluna dinâmica esta função deve ser chamada outra vez. O *buildDynamicColumnInsert* é um método recursivo pois quando o valor a inserir é uma coluna dinâmica este chama-se a si próprio. A condição de paragem é executada quando o valor a adicionar à *query* é um valor numérico e não uma coluna dinâmica. Considerando a seguinte coluna dinâmica que representa uma localização:

{'Portugal' : {'Porto' : {'Matosinhos' : 10, 'Campanhã' : 2}}}

O resultado devolvido seria:

COLUMN_CREATE('Portugal', COLUMN_CREATE('Porto', COLUMN_CREATE('Matosinhos', '10', 'Campanhã', '2')))

```
private function buildDynamicColumnInsert($dynamicColumn) {
    $sql = 'COLUMN_CREATE('; $i = 0;
    foreach ($dynamicColumn as $key => $value) {
        if ($i++ != 0) {
            $sql .= ', ';
        }
        $sql .= '\'' . $key . '\'';
        if (is_array($value)) {
            $sql .= ', ' . $this->buildDynamicColumnInsert($value);
        } else {
            $sql .= ', ' . $value;
        }
    }
    $sql .= ')';
    return $sql;
}
```

Extrato de código 13 Construção do SQL de inserção de uma coluna dinâmica

O *Extrato de código 14* mostra o método *sumDynamicColumns* que tem como objetivo executar a soma de duas colunas dinâmicas devolvendo o resultando. Tal como o anterior é um método recursivo em que a condição de paragem acontece quando o valor a analisar não for uma coluna dinâmica, mas sim um valor numérico. Este método percorre a segunda coluna adicionado à primeira os valores à chave correspondente, ou inserindo a chave e o valor caso essa chave ainda não exista na primeira coluna.

```
private function sumDynamicColumns($col1, $col2)
{
    foreach ($col2 as $key => $value) {
        if (array_key_exists($key, $col1)) {
            if (is_array($value)) {
                $col1[$key] = $this->sumDynamicColumns($col1[$key], $value);
            } else {
                $col1[$key] += (int)$value;
            }
        } else {
            $col1[$key] = $value;
        }
    }

    return $col1;
}
```

Extrato de código 14 Soma de colunas dinâmicas

5.5.3 Integração com o Piwik

No ponto de vista de implementação para integrar o sistema com o Piwik basta utilizar as funcionalidades fornecidas pela API deste. Após a sua instalação basta criar o *site* que

representará a plataforma E-goi. O *site* pode ser criado manualmente ou pode ser chamado um método da API através do seguinte URL:

http://piwik.e-goi.com/piwik?module=API&method=SitesManager.addSite&siteName=e-goi&urls=www.e-goi.com&token_auth={token}

O URL é chamado para a máquina onde executa o Piwik, passando como parâmetro o nome do método a chamar, o nome do *site* a criar e o seu URL. Também é necessário passar um *token* de autenticação que pode ser obtido através da conta de administrador da plataforma, ou qualquer outra conta que tenha sido criada.

Após a criação do *site* inserem-se as dimensões, que podem ser inseridas manualmente para o *site* criado, ou chamando:

http://piwik.e-goi.com/piwik?module=API&method=CustomDimensions.configureNewCustomDimension&idSite=1&name=form&scope=visit&active=1&token_auth={token}

Para criar as dimensões é passado o identificador do *site*, como só existirá um *site* o identificador será sempre 1. Passa-se também o nome da dimensão, *form*, o âmbito da dimensão, *visit*, para a dimensão ser avaliada uma vez por visita, e identifica-se ainda se a dimensão está ativa ou não. O mesmo procedimento deve ser executado para as restantes dimensões.

Para inserir uma visita num formulário pode ser usado o cliente Javascript ou fazer uma chamada à *HTTP Tracking API* do Piwik. Na inserção da visita são passados o URL do formulário, o identificador do *site* (no caso será sempre o mesmo), um número aleatório para impedir que o pedido seja apanhado por uma *proxy*, e os identificadores do utilizador, formulário e publicação (também pode ser passado o identificador do contacto caso a visita tenha sido executada por um). Como o identificador do formulário e da publicação visita são passados como parâmetro é possível fazer a associação entre o formulário e a visita. O URL mostra o pedido a efetuar:

http://piwik.egoi.com/piwik.php?&action_name=Visit form&url={form_url}&idsite=1&rand=351459&user=33&form=10&publish=1238

Podem ser passados bastantes mais parâmetros que se queiram registar ou que alterem o comportamento do pedido, mas desta forma o Piwik já regista automaticamente bastante informação sobre a visita. Para visualizar a informação registada pode ser utilizada a *interface*

web disponibilizada. Contudo para mostrar essa informação aos utilizadores é necessário fazer uma chamada à *HTTP Reporting API* através do URL:

http://piwik.egoi.com/?token_auth={token}&module=API&method=VisitsSummary.get&idSite=1&period=year&date=2018-1-1&segment=user==33,form==10,publish==1238

Esta API é a mesma que é utilizada para inserir o *site* e as dimensões, mas neste caso é chamado o método *VisitsSummary.get* que devolve a informação das visitas. Como parâmetro é passado o identificador do *site*, o período de visitas a devolver, a data que será afetada pelo período e segmentações adicionais. O parâmetro *segment* permite adicionar filtragem aos dados devolvidos, no exemplo apresentado são devolvidas as visitas efetuadas no ano de 2018 para a publicação 1238 do formulário 10 do utilizador 33.

Para o Piwik conseguir recolher informação do visitante é necessário inserir o código Javascript representado no *Extrato de código 15*. Este código deve ser inserido no ficheiro HTML de cada formulário onde é passado o URL da máquina que executa o Piwik e identificador do *site* (os quais são valores constantes). Para inserir a visita quando o código é carregado pode ser utilizado o cliente Javascript do Piwik ou feito um pedido para o URL definido acima. De qualquer das formas é sempre necessário passar o identificador do utilizador, do formulário e da publicação, portanto o código deve ser construído para cada formulário no momento da sua publicação.

```
<script type="text/javascript">
          var _paq = _paq || [];
          _paq.push(['trackPageView']);
          _paq.push(['enableLinkTracking']);
          (function() {
                    var u="//piwik.e-goi.com/";
                    _paq.push(['setTrackerUrl', u+'piwik.php']);
                    _paq.push(['setSiteId', 1]);
                    var d=document, g=d.createElement('script'),
          s=d.getElementsByTagName('script')[0];
                    g.type='text/javascript'; g.async=true; g.defer=true;
          g.src=u+'piwik.js';
          s.parentNode.insertBefore(g,s);
          })();
</script>
```

Extrato de código 15 Código de tracking do Piwik

5.6 Testes de *Software*

Os testes de *software* pretendem garantir a qualidade do código desenvolvido e diminuir os custos de manutenção. A utilização de padrões e princípios de desenho (cf. 5.4) também contribuem nesse sentido, melhorando os custos de manutenção relativamente ao sistema antigo dado que este não seguia boas práticas. Dos testes realizados 100% são bem-sucedidos com uma cobertura de 95%.

Para a realização dos testes foi utilizada uma abordagem baseada no modelo em V [62], que para cada fase do desenvolvimento de software atribui um tipo de testes.

- **Testes unitários**: testam unidades isoladas de código, tipicamente fornecendo um valor de entrada e comparando o resultado obtido com o resultado esperado. A realização deste tipo de testes permite não só detetar falhas no código e melhorar a qualidade do produto, como também garantir que alterações efetuadas não entram em conflito com os testes previamente desenvolvidos;
- **Testes de integração**: testam a integração de diversos componentes, quer internos à aplicação ou externos. São considerados testes de integração os testes à camada aplicacional, pois utilizam diversos componentes, os testes à camada de dados que acedem à base de dados, pois utilizam um componente externo, os testes à camada de persistência por integrarem diversos componentes da aplicação, e os testes que garantem comunicação de serviços;
- **Testes de aceitação**: permitem validar se a aplicação cumpre os requisitos. Estes testes testam o sistema como um todo e o seu resultado deve ser uma resposta na forma de "aceite" ou "rejeitado".

Os testes unitários são utilizados para garantir a qualidade da disciplina de implementação, para a disciplina de *design* são realizados testes de integração e para a disciplina de requisitos são realizados testes de aceitação. Para realização dos testes unitários e de integração foi utilizada a *framework* PHPUnit [63]. As secções seguintes apresentam exemplos de cada um dos tipos de testes utilizados.

5.6.1 Testes Unitários

Os testes unitários testam unidades de código isoladas comparando o resultado obtido com um resultado esperado. O *Extrato de código 16* contempla um exemplo de um teste unitário. Neste exemplo o alvo do teste é o método *publishContent* da classe *PublishForm.* Inicialmente são criados os dados necessários para a execução do método, que incluem conteúdo e código do formulário, assim como algumas configurações do módulo. É criada uma instância da classe sobre teste e executado o respetivo método passando os dados criado. A fase final do teste utiliza os métodos *assertTrue* e *assertEquals* da *framework* de testes para comparar o resultado obtido com o resultado esperado. Neste caso o resultado esperado é a criação de um ficheiro cujo nome é o código do formulário e o seu conteúdo é o HTML do formulário.

```
public function testPublishContent()
{
    define('PUBLIC_PATH', __DIR__);
    $moduleConfig['forms'] = 'forms/';
    $moduleConfig['content'] = 'content/';
    $html = '<!DOCTYPE html><html><body><h1>Test</h1></body></html>';
    $form['hash'] = 'eAgh6534dH';
    $form['html'] = $html;

    $instance = new PublishForm();
    $instance->publishContent($moduleConfig, $form);

    $filename = PUBLIC_PATH . $moduleConfig['forms'].
        $moduleConfig['content'] . $form['hash'] . 'html';
    $this->assertTrue(file_exists($filename));
    $this->assertEquals($form['html'], file_get_contents($filename));
}
```

Extrato de código 16 Exemplo de teste unitário

5.6.2 Testes de Integração

Os testes de integração testam a integração de diversos componentes e camadas do sistema, por exemplo a integração entre a camada de serviço e a camada de repositório, ou a integração entre os repositórios e a base de dados. O *Extrato de código 17* apresenta um exemplo de teste integração. No exemplo apresentado é testada a integração do serviço *PublishFormService,* do componente *services,* com as funcionalidades do componente *egoi-public* que este chama. É criada uma instância do serviço passando os valores do repositório e do *ServiceLocatorInterface* como *null* pois não são utilizados no método. É chamado o método *publish* passando os dados necessários, que incluem as limitações e dados do formulário e os identificadores da publicação

e do utilizador. O método *publish* irá chamar as funcionalidades de publicação do componente *egoi-public*, para validar o sucesso é verificado se foi devolvido algum URL da publicação do formulário. Não é possível comparar o URL com um texto específico pois o algoritmo de geração deste encontra-se na camada pública não sendo possível replicar a lógica de geração no componente de testes.

```
public function testPublish()
{
    $limitations['maxVisits'] = 1000;
    $limitations['maxSubmissions'] = 0;
    $limitations['onePerContact'] = false;
    $limitations['onePerIp'] = false;
    $html = '<!DOCTYPE html><html><body><h1>Test</h1></body></html>';
    $data['formId'] = 1;
    $data['html'] = $html;
    $data['limitations'] = $limitations;

    $instance = new PublishFormService(null, null);
    $url = $instance->publish($data, 1, 50);

    $this->assertTrue($url != null);
}
```

Extrato de código 17 Exemplo de teste de integração

5.6.3 Testes de Aceitação

Os testes de aceitação permitem validar se o sistema cumpre os requisitos definidos nos critérios de aceitação. Estes testes testam os componentes criados como um todo. O resultado do teste deve ser uma resposta no formato de “aceite” ou “rejeitado”, sendo necessário obter uma resposta afirmativa em relação aos testes definidos para os critérios de aceitação para o produto poder ser entregue. Podem ser realizados um ou mais testes de aceitação para cada item dos requisitos de forma a validar diferentes cenários. O *Extrato de código 18* apresenta um exemplo do teste de aceitação do primeiro requisito de criação de formulários. Este requisito deve permitir criar a entidade formulário através do seu nome, lista de contactos associada e tipo. O procedimento necessário para realizar o teste é fazer um pedido ao serviço desenvolvido passando os dados do formulário. Os critérios de validação incluem os critérios de aceitação mais algumas mensagens de erro específicas da validação que devem ser consideradas na validação do serviço. Como todos os critérios são aceites o teste pode ser considerado como bem-sucedido.

ID	TA_1a
Cenário	Criar formulário
Procedimento	• Autenticar no componente *services;* • Enviar pedido HTTP POST para *services.e-goi.com/api/forms* passando a *apikey* obtida na autenticação e os dados do formulário.
Critérios	1. Devolve mensagem de erro (*UnprocessableEntity*) caso falte algum dado obrigatório do formulário (nome, tipo e lista de contactos); 2. Devolve mensagem de erro (*UnprocessableEntity*) caso os dados do formulário não sejam válidos; 3. Devolve mensagem de erro (*AccessDenied*) caso o utilizador não tenha permissões para criar formulários; 4. Devolve mensagem de erro (*InternalServerError*) em caso de erro do servidor; 5. Devolve mensagem de sucesso (*OK*) caso o formulário tenha sido criado com sucesso.
Resultado dos critérios	1. Sim; 2. Sim; 3. Sim; 4. Sim; 5. Sim.
Resultado	Aceite

Extrato de código 18 Exemplo de teste de aceitação

6 Experiências e Validação da Solução

Neste capítulo são apresentadas as experiências planeadas para validar que a solução desenvolvida resolve o problema proposto, assim como os resultados obtidos por estas.

6.1 Experiências

As experiências permitem validar que o problema definido (cf. 1.2) foi realmente resolvido e avaliar a qualidade da solução produzida. As experiências são realizadas através de testes de *software* e testes de satisfação. Certos testes são executados apenas no final do desenvolvimento da solução, procurando falhas na solução produzida, enquanto outros testes são executados durante o desenvolvimento, e com frequência desenhados numa fase bastante inicial procurando prevenir falhas e aumentar a qualidade do sistema. Previamente à execução dos testes é necessário perceber que grandezas se vão avaliar que permitam validar que o problema foi resolvido, quais as hipóteses de avaliação a definir, e que metodologias de teste serão utilizadas para avaliar essas hipóteses. Após definidas as grandezas, as hipóteses, as metodologias de teste, estes podem ser executados.

6.1.1 Grandezas a Avaliar

As grandezas a avaliar permitem perceber que aspetos são necessários ter em consideração para avaliar a solução desenvolvida. As experiências elaboradas no âmbito desta dissertação permitem avaliar duas grandezas principais:

- **Qualidade técnica do sistema**: avalia a qualidade do código desenvolvido e a manutenibilidade da solução;
- **Satisfação das partes interessadas**: avalia a satisfação das partes interessadas no sistema em relação à abordagem proposta para o novo sistema de formulários, assim como as suas funcionalidades.

As grandezas definidas têm como objetivo avaliar partes do problema distintas e validar se este ficou realmente resolvido. A grandeza de qualidade técnica permite avaliar o problema relacionado com o custo de manutenção do sistema, pois para produzir um sistema com o menor custo de manutenção possível é necessário desenvolvê-lo utilizando boas práticas de

engenharia que permitam aumentar a qualidade do mesmo. A manutenibilidade de um sistema não é uma caraterística que possa ser facilmente avaliada pois não é quantificável. Literatura neste tópico identifica caraterísticas como a testabilidade, facilidade de compreensão, facilidade de alteração e mapeamento entre os requisitos e a implementação [64], para tentar medir a facilidade com que um sistema pode ser mantido. Essas caraterísticas tratam aspetos como a quantidade de testes unitários e de integração de um sistema, quantidade de comentários em relação com o número de linhas de código, padrões e princípios utilizados e ainda se existe uma ligação direta entre os conceitos de negócio e o código desenvolvido. Essas caraterísticas e métricas para a avaliação da manutenibilidade de um sistema são bastante subjetivas e imprecisas, não existindo ainda nenhum modelo exato para atingir esse objetivo. Nesta dissertação é avaliada a testabilidade do sistema através dos testes de *software* realizados, enquanto que outras caraterísticas são comparadas de forma subjetiva com o sistema antigo.

A grandeza de satisfação das partes interessadas permite validar se os restantes problemas definidos ficaram resolvidos. Problemas como a falta de clareza no conceito de formulário, a falta de funcionalidades e métricas de análise de resultados, são avaliados através das opiniões dos utilizadores, visto serem problemas dificilmente quantificáveis. Nesta dissertação a satisfação avalia como as partes interessadas se sentem em relação às funcionalidades, métricas e abordagens propostas com o objetivo de validar a solução desenvolvida. Como o objetivo do projeto não incluía o desenvolvimento de um editor gráfico, mas sim dos serviços de gestão, a satisfação não pode ser avaliada do ponto de vista de usabilidade. Pelo mesmo motivo o problema de dificuldade de gestão do formulário é o único a não ser considerado por as grandezas enumeradas.

6.1.2 Hipóteses

Uma hipótese estatística é uma suposição de algo que é provável, para comprovar a veracidade da hipótese são utilizados testes de hipóteses [65]. Os testes de hipóteses têm como objetivo decidir sobre duas ou mais hipóteses, aceitar a hipótese nula ou a hipótese alternativa.

Tendo em conta as grandezas definidas são definidas as seguintes hipóteses para avaliar a solução:

- **Hipótese 1:** 100% dos testes de *software* têm sucesso;

- **Hipótese 2:** Pelo menos 90% do código é coberto pelos testes de *software:*
- **Hipótese 3**: A satisfação com abordagem proposta para o novo sistema está acima da média;
- **Hipótese 4**: A satisfação média em relação às funcionalidades do novo sistema é superior à do antigo;
- **Hipótese 5**: A satisfação média em relação às métricas de análise de resultados do novo sistema é superior à do antigo.

As hipóteses apresentadas permitem validar que a solução desenvolvida resolve realmente o problema. Tanto a hipótese 1 como a hipótese 2 permitem verificar que parte do problema do custo de manutenção do sistema ficou resolvido, aumentando a sua qualidade técnica. A hipótese 3 permite verificar que a abordagem proposta para o novo sistema é adequada, validando o conceito de formulário definido. As hipóteses 4 e 5 permitem validar que as funcionalidades implementadas e propostas são adequadas para o novo sistema, assim como as métricas de análise de resultados.

6.1.3 Metodologias de Avaliação

Nesta secção é apresenta a metodologia utilizada para obter os dados necessários para avaliar cada uma das hipóteses definidas.

Os dados necessários às hipóteses 1 e 2, assim como a avaliação da qualidade técnica do sistema em geral, são obtidos através dos testes de *software* executados durante o desenvolvimento (cf. 5.6).

As hipóteses 3, 4 e 5 avaliam a satisfação das partes interessadas com a abordagem proposta para o sistema. Para avaliar a satisfação foi efetuada uma sessão de avaliação que seguiu a estrutura apresentada no Anexo B, assim como o questionário exibido as participantes. Uma sessão de avaliação com apenas 5 participantes permite identificar aproximadamente 80% dos problemas, com pelo menos 10 participantes permite identificar 90% dos problemas [66]. Para a sessão realizada foram selecionados cerca de 30 participantes com uma faixa etária que variou entre os 21 e 55 anos. A população selecionada faz parte do contexto E-goi, sendo os elementos de diferentes equipas e departamentos da organização. Os participantes escolhidos possuem caraterísticas diversificadas do ponto de vista técnico e de utilização do sistema de formulários,

alguns utilizam o sistema diariamente para suportar o *marketing* da própria organização enquanto outros trabalharam no seu desenvolvimento. A sessão de avaliação contou com uma explicação inicial sobre a abordagem proposta seguida do preenchimento dos questionários.

6.1.4 Testes Estatísticos

Os testes estatísticos permitem rejeitar ou aceitar uma hipótese a um certo grau de confiança. Para comparar as hipóteses definidas são utilizados os seguintes testes estatísticos:

- **Hipótese 1**: Não necessita de teste estatístico, apenas a avaliação direta dos resultados dos testes de *software*;
- **Hipótese 2**: Não necessita de teste estatístico, apenas a avaliação direta dos resultados dos testes de *software*;
- **Hipótese 3**: Teste paramétrico (teste t) para o valor médio de uma população;
- **Hipótese 4**: Teste paramétrico (teste t) para a diferença de valores médios entre duas populações emparelhadas com variâncias desconhecidas;
- **Hipótese 5**: Teste paramétrico (teste t) para a diferença de valores médios entre duas populações emparelhadas com variâncias desconhecidas.

Os testes estatísticos realizados são paramétricos pois segundo o teorema do limite central, é possível assumir a distribuição normal das médias de uma amostra igual ou superior a trinta elementos [67]. As amostras são emparelhadas pois são executados para o mesmo grupo de sujeitos tendo em conta dois sistemas diferentes para resolver o mesmo problema.

6.2 Resultados

Nesta secção são apresentados os resultados dos testes de satisfação efetuados.

Os testes de satisfação permitem avaliar o agrado das partes interessadas com a abordagem proposta para o novo sistema de gestão de formulários. Estes testes verificam que uma certa parte do problema definido no início da dissertação ficou resolvido. Para a realização dos testes foi executada uma sessão de avaliação, previamente ao início do desenvolvimento, que contou com 30 participantes que preencheram um questionário no total de 12 perguntas (cf. Anexo B). A sessão de avaliação começa com uma breve explicação dos objetivos a avaliar e abordagem

proposta. Passando depois ao questionário que contém perguntas que permite validar com as partes interessadas que o conceito de formulário ficou clarificado, assim como se as funcionalidades e métricas propostas são indicadas, comparando com o antigo sistema de formulários. Grande parte das perguntas do questionário utilizam a escala de Likert [68] para os participantes mostrarem o seu grau de concordância com as afirmações. A escala utilizada serviu-se de 5 valores qualitativos (e.g. discordo totalmente, discordo parcialmente, indiferente, concordo parcialmente e discordo totalmente). Devido à ordem dos valores pode-se classificar as vaiáveis presentes no questionário como qualitativas ordinais. Para representação visual dos dados foram utilizados gráficos de barras e tabelas que ajudam a comparar as diferentes respostas obtidas para cada pergunta.

A *Figura 41* apresenta os resultados das 5 perguntas iniciais do questionário que abordam o conceito de formulário. Pela pergunta 1 percebe-se bastante concordância na principal definição de formulário, onde este é definido como uma ferramenta para requisitar informação. Já pelos resultados da pergunta 2 percebe-se que mais de metade das pessoas acha que um formulário tem obrigatoriamente de requisitar informação (o que está correto de acordo com a definição proposta), mesmo assim bastantes pessoas responderam negativamente a essa obrigatoriedade. Os resultados da pergunta 3 mostram bastante concordância com o fluxo de um sistema de gestão de formulários também identificado por esta dissertação. A pergunta 4 foi a que obteve piores resultados do ponto de vista da abordagem proposta (cf. 3.2) pois bastante participantes acham que um sistema de gestão de formulários deve permitir a criação da página em que o formulário está integrado no momento da criação do formulário (abordagem utilizada pelo sistema antigo). Contudo a maior parte dos participantes concorda com a abordagem proposta de criar o formulário e a página em separado e permitir integração. Os resultados da pergunta 5 mostram uma concordância quase total em relação ás diferenças de um sistema de gestão de formulários aplicado no contexto de *marketing* digital de um sistema sem contexto específico.

Em geral os resultados obtidos mostram bastante concordância com o conceito de formulário proposto assim como o conceito de sistema de gestão de formulários.

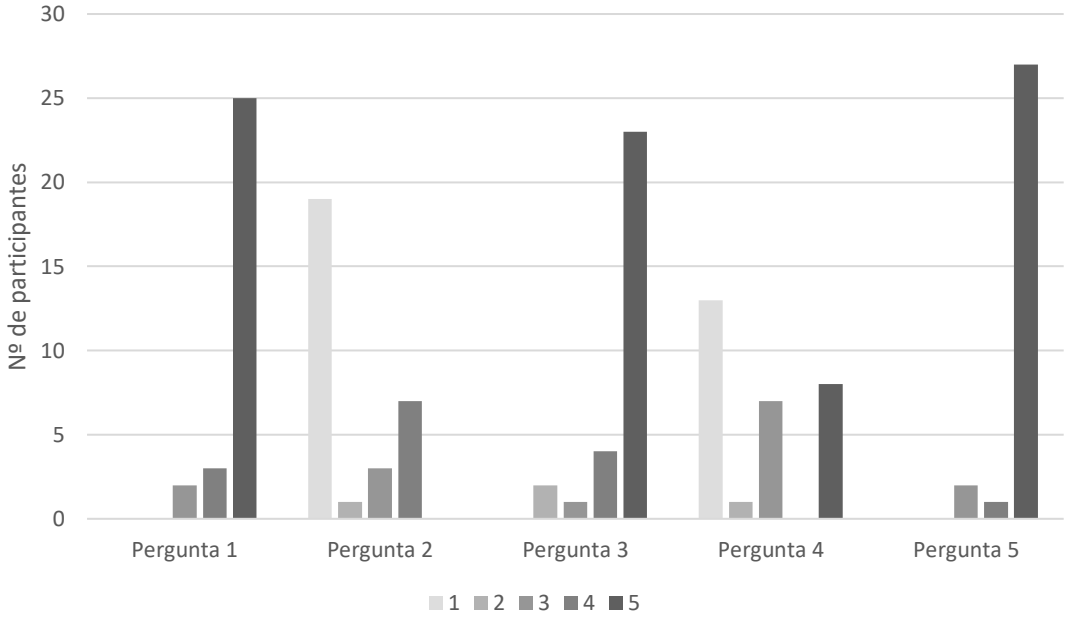

Figura 41 Resultados da perguntas sobre o conceito de formulário

A *Figura 42* apresenta os resultados relativamente à pergunta 12 sobre a abordagem proposta. Todos os participantes concordam com a abordagem proposta, 17 dos quais acham uma boa abordagem e os restantes acham uma muito boa abordagem.

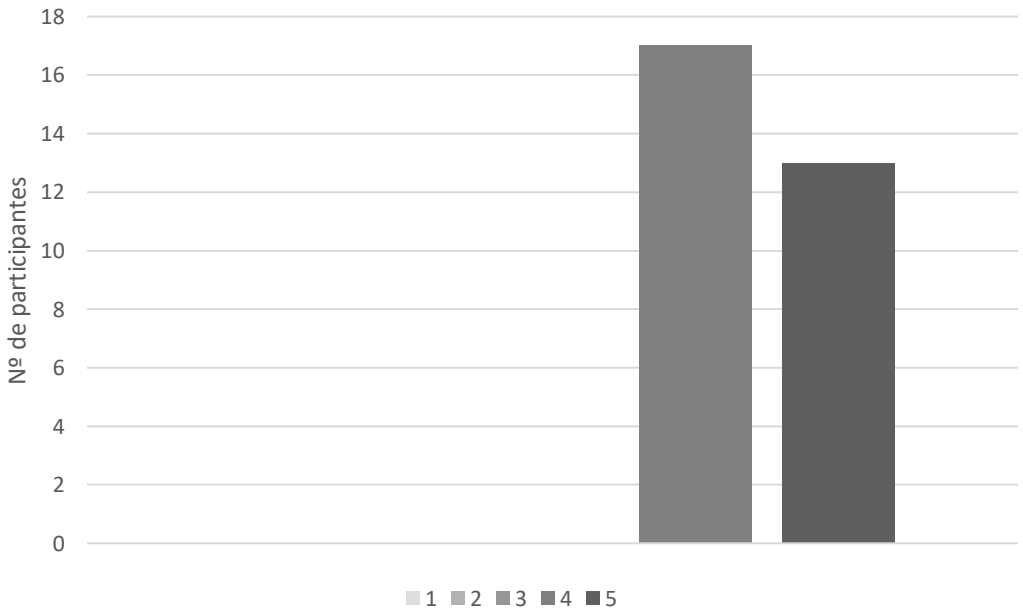

Figura 42 Resultado da pergunta sobre a abordagem proposta

Para validar estatisticamente que a abordagem proposta é adequada é executado o seguinte teste estatístico definido para a hipótese 3:

$$H_0 : \mu = \mu_0 \quad vs \quad H_1 : \mu > \mu_0$$

Este teste testa se a média de uma população esta acima da média da escala. Como neste caso são utilizados cinco valores, μ_0 representa 2.5, enquanto que μ representa a média das respostas da pergunta 12. Realizando o teste paramétrico t teste com um grau de confiança de 95% obtém-se um *p-value* de 2.2×10^{-16}. Como o *p-value* é menor que 0.05 rejeita-se H_0 a um nível de significância de 5%. Pode-se concluir que a média as repostas relativamente à abordagem proposta esta significativamente acima da média da escala.

A *Figura 43* mostra alguns dos resultados obtidos na pergunta sobre as funcionalidades mais importantes. As funcionalidades que não aparecem no gráfico obtiveram pontuação máxima. Observa-se uma atribuição de média importância à funcionalidade de inserção de elemento gráfico, que vai de acordo com os resultados das questões iniciais. A funcionalidade de definição de lógica específica para validação dos campos apresenta uma fraca importância. A funcionalidade considerada menos importante pelos participantes é a definição de traduções. Esta funcionalidade também não é oferecida pelo novo sistema, opta-se pela abordagem de criação de um novo formulário (através de outro já criado) numa língua diferente, o que vai de acordo às necessidade e contexto da E-goi.

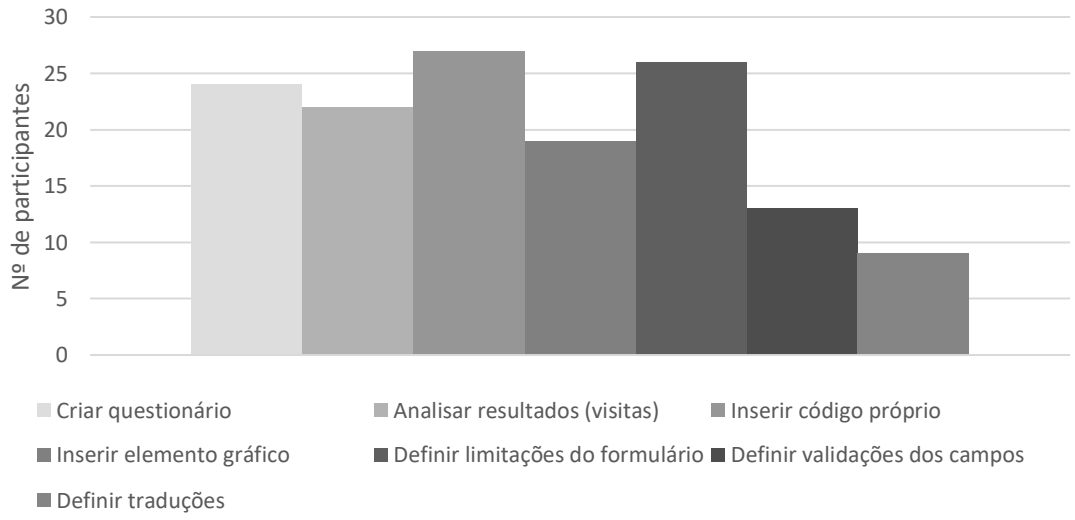

Figura 43 Resultado da pergunta sobre as funcionalidades importantes

A *Figura 44* mostra os resultados de importância das métricas de análise de resultados dos formulários. A métrica de objetivos é considerada a menos importante, seguida da métrica de referentes. As métricas que não se encontram no gráfico (e.g. submissões) tiveram pontuação máxima de importância. O novo sistema permite todo o tipo de métricas definidas distintamente se destacando neste âmbito comparativamente ao antigo sistema que só permitia a contagem das visitas e as repostas submetidas.

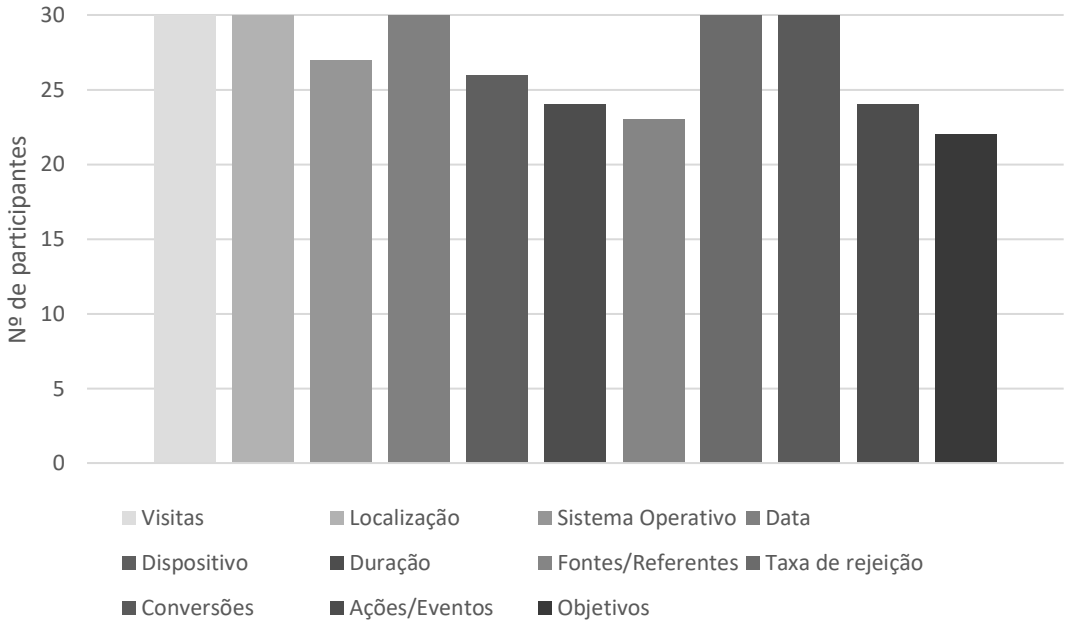

Figura 44 Resultado da pergunta sobre as métricas importantes

A *Tabela 12* apresenta a média, moda e mediana das perguntas que comparam as funcionalidades e métricas dos dois sistemas. A média da satisfação em relação às funcionalidades é mais alta no novo sistema por 7 décimas, o que revela que é um indicativo que o novo sistema é possivelmente melhor, do ponto de vista de funcionalidades, mas o sistema antigo também era razoável visto ter uma média positiva. Já do ponto de vista da satisfação com as métricas de análise de resultados existe uma discrepância enorme onde as novas métricas apresentam uma média de 4.9 e as métricas do sistema antigo apenas 1.3. Esta insatisfação confirma um dos problemas no âmbito do nascimento desta dissertação, a falta de métricas de análise de resultados no sistema de gestão de formulário do E-goi. O novo sistema vem resolver esse problema obtendo uma satisfação quase excelente pelas partes interessadas.

Tabela 12 Média, moda e mediana da comparação de funcionalidades e métricas do antigo e do novo sistema

Perguntas	Média	Moda	Mediana
7. Como classifica as funcionalidades propostas para o novo sistema?	4.1	4	4
8. Como classifica as funcionalidades do sistema antigo?	3.4	3	3
10. Como classifica as métricas de análise de resultados do antigo sistema de formulários?	1.3	1	1
11. Como classifica as métricas de análise de resultados do novo sistema de formulários?	4.9	5	5

Para validar que existe diferença significativa entre a satisfação média das funcionalidades do novo sistema em relação ao antigo é realizado um teste estatístico para os valores médios entre duas populações emparelhadas (como definido pela hipótese 4). O mesmo é feito para a comparação da satisfação em relação às métricas (como definido pela hipótese 5). Antes de aplicar o teste t às médias de duas amostras é necessário verificar quanto à igualdade das suas variâncias:

$$H_0 : \ {\sigma^2}_A = {\sigma^2}_B \quad vs \quad H_1 : \ {\sigma^2}_A \neq {\sigma^2}_B$$

Para verificar as variâncias é realizado um teste de hipóteses como indicado, em que A corresponde ao novo sistema e B ao antigo. O resultado dos testes, realizados com um grau de confiança de 95%, obtém um *p-value* 0.04972, para a comparação da variância das funcionalidades, e um *p-value* de 0.00022 na comparação das variâncias das métricas. Em ambos os casos como o *p-value* é menor que 0.05 rejeita-se hipótese nula e aceita-se a alternativa. Conclui-se então que as variâncias das amostras são diferentes em ambos os casos, tal deve ser levado em consideração na execução do teste.

Finalmente para validar se existe diferença significativa entre as médias das amostras executa-se o seguinte teste:

$$H_0 : \ \mu_A = \mu_B \quad vs \quad H_1 : \ \mu_A > \mu_B$$

Neste teste μ representa a média da amostra dos resultados das perguntas. O alvo do teste é verificar se a média dos resultados das perguntas (satisfação) do novo sistema de formulários é maior que o antigo. O teste é executado para a comparação de funcionalidades e repetido para a comparação de métricas. Executando os testes, com um grau de confiança de 95%, obtém-se um *p-value* de 4.36×10^{-8} no caso das funcionalidades e um *p-value* de 2.20×10^{-16} no caso das métricas. Novamente como o *p-value* é menor que 0.05 rejeita-se a hipótese nula a um nível de significância de 5%. Conclui-se que existem diferenças significativas nas médias das amostras referentes aos resultados de satisfação, sendo a média do novo sistema de formulários superior, o que revela que é o sistema mais satisfatório.

Os resultados dos dados obtidos e dos testes estatísticos efetuados permitem validar as hipóteses definidas. É verificado que existe uma satisfação positiva com abordagem proposta para o novo sistema e que a satisfação média em relação às funcionalidades e métricas de análise de resultados do novo sistema é superior à do antigo.

7 Conclusões

Este capítulo apresenta uma síntese dos objetivos alcançados, problemas resolvidos, as principais limitações da solução desenvolvida e as necessidades de trabalho futuro para dar continuidade à solução.

Foi desenvolvida parte do novo sistema de gestão de formulários da plataforma E-goi. O novo sistema contém funcionalidades adequadas às necessidades dos clientes e resolve um conjunto de problemas existentes no sistema antigo, nomeadamente:

- A falta de clareza nos conceitos de formulário e sistema de gestão de formulários;
- O elevado custo de manutenção do sistema e dificuldade de expansão;
- A falta de funcionalidades;
- A falta de métricas de análise de resultados.

Para atingir os objetivos propostos e o resolver os problemas definidos foi inicialmente feita uma análise do estado de arte de sistemas de gestão de formulários, para perceber as abordagens utilizadas por outras soluções existentes e selecionar a adequada ao problema. Na sequência desta análise foi definido conceito de formulário e outros conceitos relevantes no contexto do sistema, como *marketing* digital e *web analytics*. Foi também elaborada uma análise de valor onde foi identificada e apresentada a oportunidade que este projeto vem preencher, juntamente com a proposta de valor que traz aos clientes. Após a análise de valor foi documentado o desenvolvimento da solução, onde são apresentados os requisitos funcionais e não funcionais do projeto e elaborada a modelação do domínio explicando em detalhe os conceitos do negócio. Do desenvolvimento da solução consta também o desenho arquitetural, o desenho detalhado e alguns detalhes da implementação. Finalmente foram definidos os testes que permitem validar que o problema foi realmente resolvido e apresentados os seus resultados.

7.1 Objetivos Alcançados

Os objetivos definidos no início da dissertação (cf. 1.3) foram alcançados na sua totalidade pelo trabalho efetuado. A conclusão dos objetivos contribuiu para a resolução de alguns dos problemas definidos (cf. 1.2).

Foi investigado o estado de arte de soluções existentes (cf. 3.1), que envolveu a comparação das funcionalidades e caraterísticas de diversos sistemas existentes no mercado, desde sistemas de gestão de formulários aplicados no contexto de *marketing* a sistemas de gestão de formulários específicos à criação de formulários. A comparação dos sistemas permitiu identificar diferentes abordagens utilizadas e selecionar uma para o sistema desenvolvido. A conclusão deste objetivo permitiu resolver o problema de falta funcionalidades identificando as funcionalidades necessárias pelas partes interessadas.

Outro objetivo atingido foi a clarificação do conceito de formulário (cf. 2.2), propondo um formulário como uma ferramenta de requisição de informação. Também foi clarificado o conceito de sistema de gestão de formulários na área de *marketing* digital, e identificado um fluxo de gestão utilizado pelos profissionais de *marketing*. Foi identificada uma abordagem a seguir pelo sistema desenvolvido, que consiste na criação do formulário e página onde este está integrado de forma separada com funcionalidades de integração. A conclusão deste objetivo permitiu resolver o problema relacionado com a clarificação do conceito de formulário.

O desenho e implementação da estrutura de dados também foi um objetivo concluído (cf. 5.4.3). Foi necessário perceber a arquitetura, tecnologias e princípios utilizados pela estrutura do E-goi e seguir esses princípios na solução desenvolvida. No âmbito deste objetivo foi proposto um método para melhorar a manutenção e expansão da estrutura de dados através da utilização de colunas dinâmicas. O método proposto foi criado especificamente para a plataforma E-goi, mas também pode ser aplicado noutros sistema que utilizem bases de dados relacionais, e que tenham necessidade de expandir frequentemente a base de dados com o mínimo de alterações possível.

Um dos objetivos principais alcançados no desenvolvimento da solução (cf. 5) foi a implementação dos serviços de gestão da estrutura de dados. Este objetivo resultou na expansão da API interna utilizada pelos desenvolvedores do E-goi para implementação da lógica de domínio e gestão dos dados. Os serviços implementados permitem gerir as entidades de negócio relacionadas com formulários, aplicando a lógica necessária neste contexto.

A análise de resultados das submissões foi um objetivo bastante abordado no decorrer da dissertação. Este objetivo foi realizado através de funcionalidades para visualizar as respostas das submissões assim como os dados das visitas associadas a um formulário. As visitas são analisadas com informação básica através de implementação de funcionalidades nos serviços,

e analisadas com informação bastante completa através da utilização de um sistema de *web analytics*, o Piwik. A conclusão deste objetivo permitiu resolver o problema da falta de métricas de análise de resultados.

A camada pública de acesso aos formulários também foi implementada, tendo especial atenção e cuidado ao elevado número de acessos que podem ocorrer nos formulários alojados. Este objetivo foi implementando focando a solução desenvolvida no desempenho e segurança. No âmbito da camada pública foram aplicados padrões e princípios de desenho de *software* que permitiu não só aumentar a qualidade do código desenvolvido, mas também do código existente.

A integração do sistema de gestão de formulários na plataforma E-goi é conseguida implementado diretamente as novas funcionalidades nos componentes existentes desta plataforma.

Os objetivos atingidos permitiram resolver a maior parte dos problemas que proporcionaram esta dissertação, contudo alguns não foram resolvidos ou foram resolvidos apenas de forma parcial. O elevado custo de manutenção foi parcialmente resolvido, visto terem sido usados princípios e boas práticas de engenharia no desenvolvimento do sistema, incluindo a utilização de padrões e testes de *software*. Contudo a resolução deste problema não pode ser validada matematicamente, porque é algo bastante subjetivo e onde deve haver um trabalho constante de melhoria. A resolução deste problema também foi dificultada pelos condicionamentos arquiteturais impostos pela plataforma E-goi, o qual o novo sistema de gestão de formulários teve de ficar acoplado. Mesmo assim a manutenção e expansão do novo sistema de gestão de formulários já não é um problema tão grande como no antigo. O problema da dificuldade identificada na gestão do formulário não pode ser totalmente resolvido, pois não houve integração com um componente de visualização para validar a resolução deste problema. Porém o trabalho efetuado no âmbito da identificação das funcionalidades necessárias vem ajudar a resolver parte deste problema.

Em geral a solução desenvolvida no âmbito desta dissertação revela-se bem-sucedida, tanto pelos objetivos, como pelos problemas resolvidos que ficaram validados pelos resultados dos testes efetuados.

7.2 Trabalho Futuro

Apesar do sucesso geral com o projeto efetuado alguns aspetos necessitam de desenvolvimento futuro de forma a concluir e melhorar o sistema de gestão de formulários.

Para concluir o sistema de gestão de formulários é necessária a sua integração com um componente de visualização ou editor gráfico para este ser utilizado pelos utilizadores finais. A implementação do componente de visualização não era objetivo desta dissertação. Este componente esta em desenvolvimento por outros colaboradores do E-goi e não foi terminado em tempo útil para integração com os serviços criados. Também será necessário integrar o componente de visualização com o componente de *web analytics* configurado para fornecer os resultados aos utilizadores conforme a solução criada.

Após a finalização do sistema de gestão de formulários como um todo é fundamental testar com utilizadores finais para perceber os problemas identificados por estes profissionais e implementar as melhorias necessárias.

Será necessária uma constante análise de mercado e dos resultados obtidos através da utilização do sistema. Essa análise permitirá perceber o ambiente contextual do sistema e detetar possíveis alterações de valor nas funcionalidades oferecidas, ou detetar a existência de novas funcionalidades utilizadas por concorrentes ou novas necessidades. Conforme as necessidades identificadas pode ser necessário implementar novas funcionalidades no sistema.

Como todos os projetos de *software* um aspeto importante é a sua manutenção. Para dar continuidade ao trabalho efetuado, para além de adicionar novas funcionalidades, devem ser corrigidos os erros detetados.

Outra possibilidade de trabalho futuro, no âmbito desta dissertação, é a publicação de artigos científicos que permitam transmitir o conhecimento adquirido sobre sistemas de gestão de formulários aplicados em *marketing* digital. Para além disso a proposta de utilização de colunas dinâmicas para manutenção da estrutura de dados é um aspeto que pode ajudar a resolver outros problemas similares.

Referências

[1] C. Jarret and G. Gaffney, Forms that Work, Morgan Kaufmann, 2009.

[2] E-goi, "E-goi," E-goi, 2003. [Online]. Available: https://www.e-goi.pt/. [Accessed Outubro 2017].

[3] E. J. Johnson and D. Goldstein, "Do Defaults Save Lives?," vol. 302, pp. 1338-1339, 21 Novembro 2003.

[4] L. Wroblewski, Web Form Design, Rosenfeld Media, 2008.

[5] Z. Yan and R. Peterson, "Customer Perceived Value, Satisfaction and Loyalty: The Role of Switching Costs," vol. 21, no. 10, pp. 799-822, 24 Agosto 2004.

[6] L. Morris, Innovation Master Plan Framework, InnovationLabs LLC, 2010.

[7] W. Stanton, Fundamentals of Marketing, Grolier, Incorporated, 1987.

[8] P. Kotler, Marketing Para O Século XXI, Futura, 2000.

[9] P. Kotler and G. Armstrong, Principles of Marketing, Pearson education, 2010.

[10] J. McCarthy, Basic marketing: A managerial approach, R. D. Irwin, 1968.

[11] A. T. Coughlan, E. Anderson, L. W. Stern and A. El-Ansary, Marketing Channels, Prentice Hall, 2006.

[12] B. Rosenbloom, Marketing Channels, Cengage Learning, 2012.

[13] M. Lokhande and M.A.Lokhande, " Social Marketing," *Indian Journal of Marketing,* vol. XXXIII, pp. 16-19, 2003.

[14] D. Chaffey, F. Elis-Chadwick, K. Johnston and R. Mayer, Internet Marketing Strategy, Implementation and Practice, Pearson Education, 2000.

[15] D. Ryan, Understanding Digital Marketing, Kogan Page, 2017.

[16] M. Trusov, R. E. Bucklin and K. Pauwels, "Effects of Word-of-Mouth Versus Traditional Marketing: Findings from an Internet Social Networking Site," *Journal of Marketing,* vol. 73, no. 5, pp. 90-102, 2009.

[17] B. Halligan and D. Shah, Inbound Marketing, Revised and Updated: Attract, Engage, and Delight Customers Online, John Wiley & Sons, 2014.

[18] Netsonda, “Netsonda,” 2000. [Online]. Available: http://www.netsonda.pt/. [Accessed Fevereiro 2018].

[19] M. Olson and D. Green, “Minimundos,” 2008. [Online]. Available: http://www.minimundos.com.br. [Accessed Fevereiro 2018].

[20] M. Zuckerberg, “Facebook,” Facebook, 4 Fevereiro 2004. [Online]. Available: https://www.facebook.com/. [Accessed Fevereiro 2018].

[21] C.-B. J., S.-P. J.C. and V.-I. A., “How Different Versions of Layout and Complexity of Web Forms Affect Users After They Start It? A Pilot Experience,” 2018.

[22] Web Analytics Association, “Web Analytics Definitions,” pp. 5-8, Setembro 2008.

[23] Web Analytics Association, “Web Analytics Definitions,” pp. 6-34, Agosto 16 2007.

[24] JotForm, “JotForm,” JotForm, 2006. [Online]. Available: https://www.jotform.com/. [Accessed Janeiro 2018].

[25] Google, “Google Forms,” Google, 31 Outubro 2012. [Online]. Available: https://www.google.com/forms/about/. [Accessed Janeiro 2018].

[26] Survey Monkey, “Wufoo,” Survey Monkey, 2010. [Online]. Available: https://www.wufoo.com/. [Accessed Janeiro 2018].

[27] Form.io, “Form.io,” Form.io, 2015. [Online]. Available: https://form.io/. [Accessed Janeiro 2018].

[28] MailChimp, MailChimp, 2001. [Online]. Available: https://mailchimp.com/. [Accessed Janeiro 2018].

[29] Mautic, “Mautic,” 2014. [Online]. Available: https://www.mautic.org/. [Accessed Janeiro 2018].

[30] A. Arseniev, “GrapesJS,” 2016. [Online]. Available: http://grapesjs.com/. [Accessed Dezembro 2017].

[31] M. Aubry, “Piwik,” 2007. [Online]. Available: www.piwik.org. [Accessed Novembro 2017].

[32] P. Adams, “Open Web Analytics,” 2006. [Online]. Available: http://www.openwebanalytics.com/. [Accessed Janeiro 2018].

[33] A. Patzer, “Mint,” 27 Agosto 2006. [Online]. Available: https://www.mint.com/. [Accessed Janeiro 2018].

[34] Google, “Google Analytics,” 14 Novembro 2005. [Online]. Available: https://www.google.com/analytics/. [Accessed Novembro 2017].

[35] P. Koen, G. Ajamian, S. Boyce, A. Clamen and E. Fisher, Fuzzy Front End: Effective Methods, Tools, and Techniques, Wiley, 2002.

[36] I. Osita, I. Onyebuchi and N. Justina, “Organization’s stability and productivity: the role of SWOT analysis an acronym for strength, weakness, opportunities and threat,” *International Journal of Innovative and Applied Research,* vol. 2, no. 9, pp. 23-32, 2014.

[37] Econsultancy, “Econsultancy,” Econsultancy, 2016. [Online]. Available: https://www.referralsaasquatch.com/29-digital-marketing-statistics-2017/. [Accessed Janeiro 2018].

[38] BrightTALK, “BrightTALK,” BrightTALK, 2015. [Online]. Available: https://www.hubspot.com/marketing-statistics. [Accessed Janeiro 2018].

[39] State Of Inbound, “State Of Inbound,” 2017. [Online]. Available: http://www.stateofinbound.com/. [Accessed Janeiro 2018].

[40] Marketing Charts, “How Online Form Conversion Rates Differ by Form Type,” 2014. [Online]. Available: https://www.marketingcharts.com/industries/non-profit-43183. [Accessed Dezembro 2017].

[41] G. Mahajan, “Customer Think,” Janeiro 2016. [Online]. Available: http://customerthink.com/what-is-customer-value-and-how-can-you-create-it/. [Accessed Novmbro 2017].

[42] A. Osterwalder and Y. Pigneur, Business model generation: a handbook for visionaries, game changers, and challengers, John Wiley and Sons, 2010.

[43] T. Saaty, Decision Making for Leaders: The Analytic Hierarchy Process for Decisions in a Complex World, RWS Publications, 1990.

[44] InnoCraft, “Piwik,” Innocraft, 2010. [Online]. Available: https://matomo.org/. [Accessed Novembro 2017].

[45] T. Saaty, “Decision making with the analytic hierarchy process,” vol. 1, no. 1, pp. 83-98, 2008.

[46] C. Larman, Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Interative Development, Pearson Education India, 2012.

[47] M. O. Ahmad, J. Markkula and M. Oivo, “Kanban in software development: A systematic literature review,” in *39th Euromicro Conference on Software Engineering and Advanced Applications*, Espanha, 2013.

[48] T. H. Poister, Measuring Performance in Public and Nonprofit Organizations, John Wiley & Sons, 2008.

[49] P. Eeles, Capturing architectural requirements, IBM Rational developer works, 2005.

[50] M. Fowler, Patterns of Enterprise Application Architecture, Addison Wesley, 2003.

[51] Microsoft Patterns & Practices Team, Microsoft® Application Architecture Guide, Microsoft Press, 2009.

[52] E. Evans, Domain-Drive Design, Addison-Wesley Professional, 2003.

[53] S. Metz, “SOLID Design,” 30 Maio 2009. [Online]. Available: http://skmetz.home.mindspring.com/img27.html. [Accessed 3 Março 2018].

[54] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1994.

[55] B. De, API Management An Architect’s Guide to Developing and Managing APIs for Your Organization, Apress, 2017.

[56] D. Maier, The theory of relational databases, Computer science press, 1983.

[57] R. Elmasri and N. Shamkant, Fundamentals of database systems, Pearson Education India, 2008.

[58] MariaDB Corporation, “MariaDB Dynamic Columns,” 22 Janeiro 2009. [Online]. Available: https://mariadb.com/kb/en/library/dynamic-columns/. [Accessed 18 Fevereiro 2018].

[59] N. Leavitt, “Will NoSQL databases live up to their promise?,” vol. 43, no. 2, pp. 12-14, 2010.

[60] A. Padilla, Beginning Zend Framework, apress, 2009.

[61] B. Snyder, D. Bosnanac and R. Davies, ActiveMQ in action, Greenwich Conn.: Manning, 2011.

[62] B. Marick, *New Models for Test Development,* Testing Foundations, 1999.

[63] S. Bergmann, “PHPUnit,” 15 Março 2004. [Online]. Available: https://phpunit.de/. [Accessed 22 Fevereiro 2018].

[64] N. KUKREJA, “Measuring Software Maintainability,” 3 Fevereiro 2015. [Online]. Available: https://quandarypeak.com/2015/02/measuring-software-maintainability/. [Accessed 2018 Junho 1].

[65] J. Neyman and E. Pearson, “On the Problem of the Most Efficient Tests of Statistical Hypotheses,” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences,* vol. 231, pp. 289-337, 1933.

[66] R. Virzi, “Refining the test phase of usability evaluation: How many subjects is enough?,” *Human Factors,* vol. 34, no. 4, pp. 457-468, 1992.

[67] R. Hogg, E. Tanis and D. Zimmerman, Probability and Statistical Inference, Prentice Hall, 2010.

[68] M. S. Matell and J. Jacoby, “Is there an optimal number of alternatives for Likert scale items? Study I: Reliability and validity,” *Educational and psychological measurement,* vol. 31, pp. 657-674, 1971.

[69] J. J. Garret, The elements of user experience: user-centered design for the web and beyond, Pearson Education, 2010.

[70] D. Hix and R. Hartson, Developing user interfaces: ensuring usability through product & process, John Wiley & Sons, 1993.

[71] J. Nielsen, Usability engineering, Elsevier, 1994.

[72] N. Borden, “The Concept of the Marketing Mix,” vol. 4, no. 2, pp. 2-7, 1964.

[73] D. Ryan and C. Jones, Understanding Digital Marketing: Marketing Strategies for Engaging the Digital Generation, Kogan Page, 2016.

Anexo A Design Detalhado (User Stories)

US02 – Inserir Campo

Pela descrição do requisito de inserção de campo percebe-se que o formulário deve ter uma lista de campos associada. Os campos têm propriedades genéricas, comuns a todos os campos, e propriedades específicas de certos tipos de campo. Como existem diversos tipos de campos e estes podem expandir com uma frequência elevada, pretende-se que a estrutura de dados seja o mais independente possível dos tipos de campo. Identifica-se então a necessidade da criação da entidade campo, *Field*, e das classes necessárias à sua gestão, dentro do módulo de formulários.

Devido à forma como os módulos do *services* estão desenhados a adição de uma nova entidade resume-se a repetir o processo já efetuado anteriormente, adicionado lógica específica daquela entidade. A *Figura 45* representa o fluxo de inserção de um campo no formulário. Por requisito os campos só são guardados quando o formulário é guardado, para otimizar processamento e não ter de guardar o campo sempre que ocorre uma alteração. Contudo para prever possível perda de dados o formulário deve ser guardado automaticamente segundo um intervalo de tempo para assegurar que o utilizador não perde os seus dados em caso de falha.

Quando um formulário é guardado há a necessidade de guardar os seus campos, para isso é utilizado o método de atualização do recurso de campos do formulário. Em vez de ser atualizado cada campo é mais rápido “apagar” os campos existentes e adicionar os novos. Os campos não devem ser mesmo apagados, mas apenas atualizado um estado interno que os marca como tal para não ocorrer perda de relações. Após os campos serem apagados através do método *remove* é necessário guardar os novos, o que é feito através do *saveBulk* do serviço e do repositório pois pode ser guardada uma lista de campos e não apenas um. O serviço que trata os campos recebe o identificador do formulário para tratar apenas dos campos pertencentes aquele formulário.

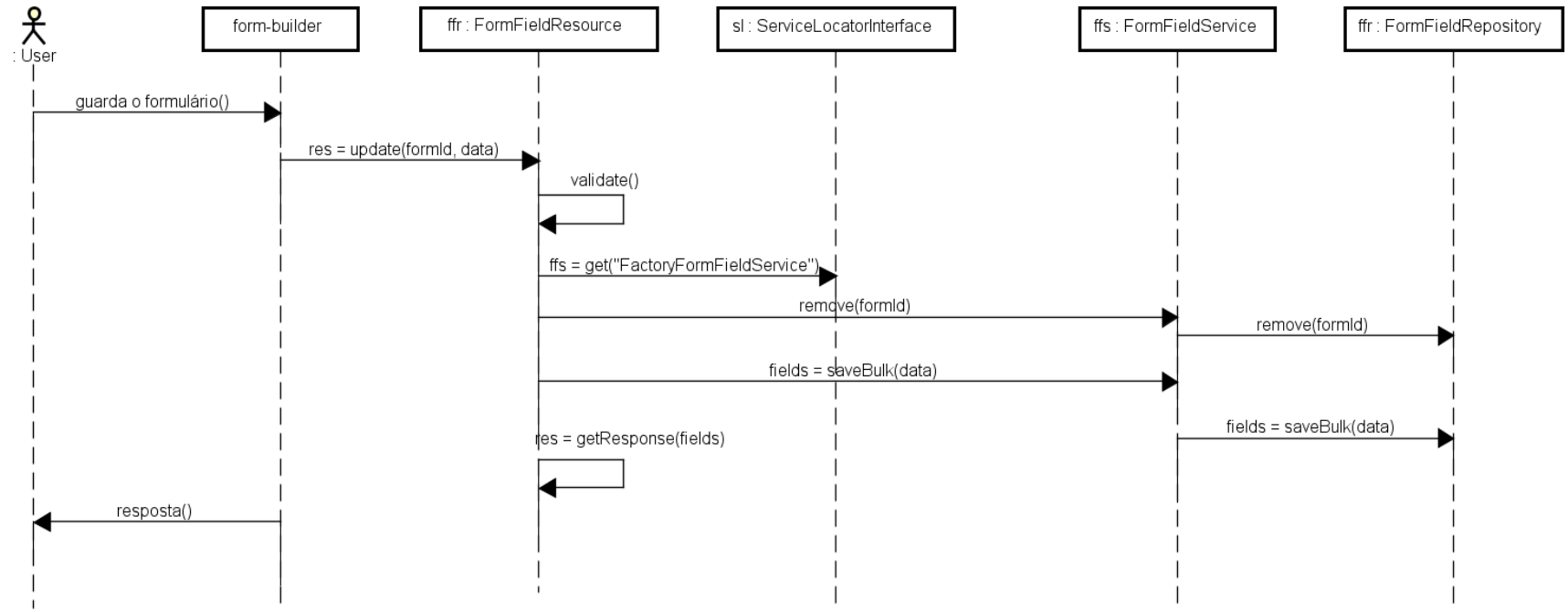

Figura 45 Diagrama de sequência: Inserção de campo

US05 – Definir Limitações

O requisito de limitações indica que uma publicação do formulário pode ser desativada quando atingir um certo limite ou condição. O limite pode ser um número de visitas, um número de submissões, ou até limitar uma submissão por IP ou por contacto. Como as limitações pertencem a uma publicação é da responsabilidade desta entidade manter estes dados. Considerando que as classes que tratam da entidade publicação já estão previstas, basta apenas criar um método de atualização como indicado *Figura 46*. É a classe *PublishFormRepository* que trata de atualizar os dados da publicação adicionando os novos campos das limitações. Para além de atualizar a entidade é necessário atualizar os ficheiros da camada pública, para isso é chamado o método *publish* novamente que voltará a gerar os ficheiros com as limitações no componente *egoi-public*.

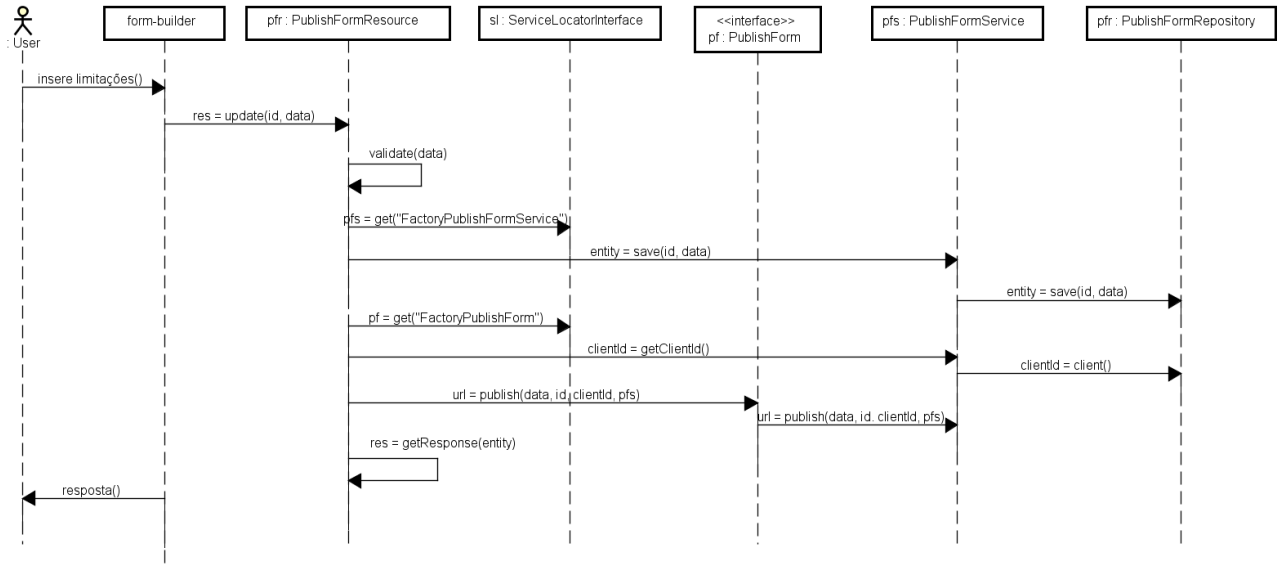

Figura 46 Diagrama de sequência: Definir limitações

US08 e US09

O requisito de lógica condicional entre campos (US08) pretende que um campo seja apresentado ou ocultado conforme o valor de outro campo. Considerando que um campo apenas é afetado por outro e não por um conjunto, não há necessidade de passar a classe candidata que representa a condição a classe de *software*. Para a realização do condicionamento basta adicionar os atributos necessários à entidade *FormField*, que contemplam o campo com que deve ser feita a comparação, o valor que esse campo deve conter, a operação de comparação e ainda se o campo atual deve ser apresentado ou ocultado.

Para cumprir o requisito de mapear um campo da lista de contactos com um campo do formulário (US09) é necessário que a entidade *FormField* armazene qual o campo da lista a qual está mapeada, se algum. Também é necessário que o componente *form-builder* utilize os módulos já existentes do *services* para mostrar ao utilizador todos os campos possíveis da lista de contactos associada ao formulário.

De uma perspetiva de fluxo do serviço basta atualizar os campos do formulário quando este é guardado, tal como é feito na inserção (cf. 0) mas agora contendo os novos dados. A *Figura 47* mostra as classes envolvidas no serviço que trata dos campos, onde está incluída a entidade que representa o campo através dos seus dados.

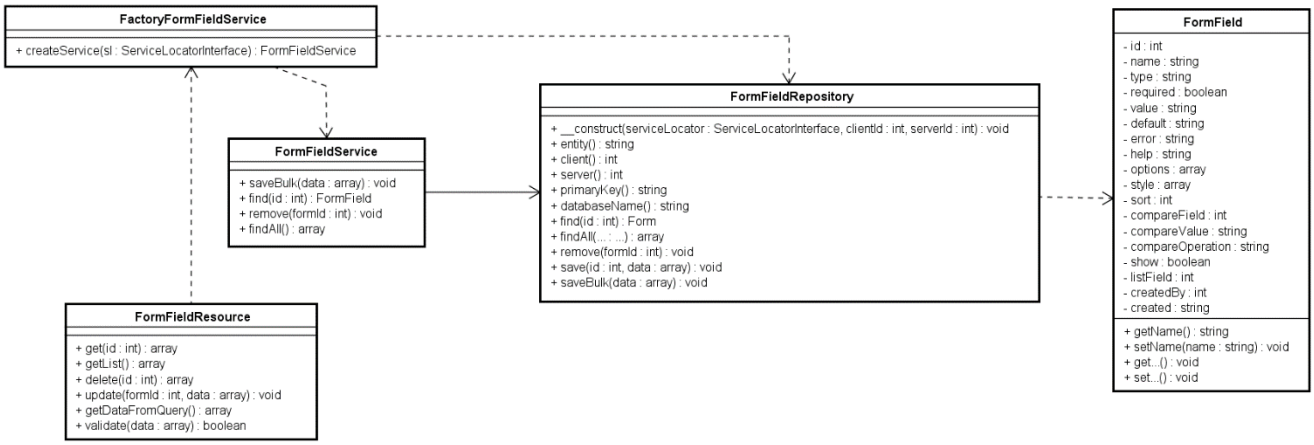

Figura 47 Diagrama de classes: Campo

US11 – Ver Estatísticas de Visitas ao Formulário

O utilizador deve conseguir visualizar um relatório com os dados das visitas de um formulário específico. Os dados contêm informação acerca do número de visitas, *browsers* utilizados, dispositivos utilizados, localizações de acesso, entre outros. A gestão das visitas pode ser realizada de duas formas, através de um sistema externo de *web analytics* ou através de alguns módulos do *services*. São utilizadas as duas formas dado que o sistema de *web analytics* apresenta informação mais completa, como dados de cada visita e visitante um por um, e contém algoritmos mais fiáveis para seguir as visitas. Contudo também é necessário registar as visitas através dos próprios serviços do E-goi para manter registos básicos em caso de falha do sistema externo e porque certos dados das visitas são necessários às funcionalidades dos serviços. Nesta secção é apresentado o desenho da geração de relatórios/estatísticas de visitas utilizando os serviços criados, a utilização do sistema externo é apresentada na secção 5.5.3.

A *Figura 48* mostra a utilização módulo *Report* dos *services* para a geração de relatórios. O E-goi já tem funcionalidades que permitem ver relatórios de campanhas que contém dados como as suas aberturas e taxa de rejeição, os relatórios dos formulários devem funcionar de forma semelhante. Como tal são implementadas classes específicas para tratar destes relatórios no módulo *Report*. A classe *FormReportResource* representa o controlador e recebe o pedido para consular o relatório de um formulário específico. Esta deve validar o identificador do formulário e os campos que pretendem ser vistos no relatório. Após a validação é utilizado o *FormCountService,* que por sua vez faz um pedido ao repositório para aceder ao relatório. O *FormCountRepository* necessita de fazer a consulta à base de dados onde estão guardados os relatórios de cada publicação e fazer a sua soma para obter o relatório do formulário. O *FormCountTransformer* é responsável por transformar o relatório no formato que é guardado (entidade *FormCount*) para o formato que será devolvido. O nome *Count* é adicionado às entidades por convenção do módulo, para distinguir das entidades dos módulos originais diferenciando-as do módulo de relatórios. O relatório é gerado quando ocorre a inserção de novos dados.

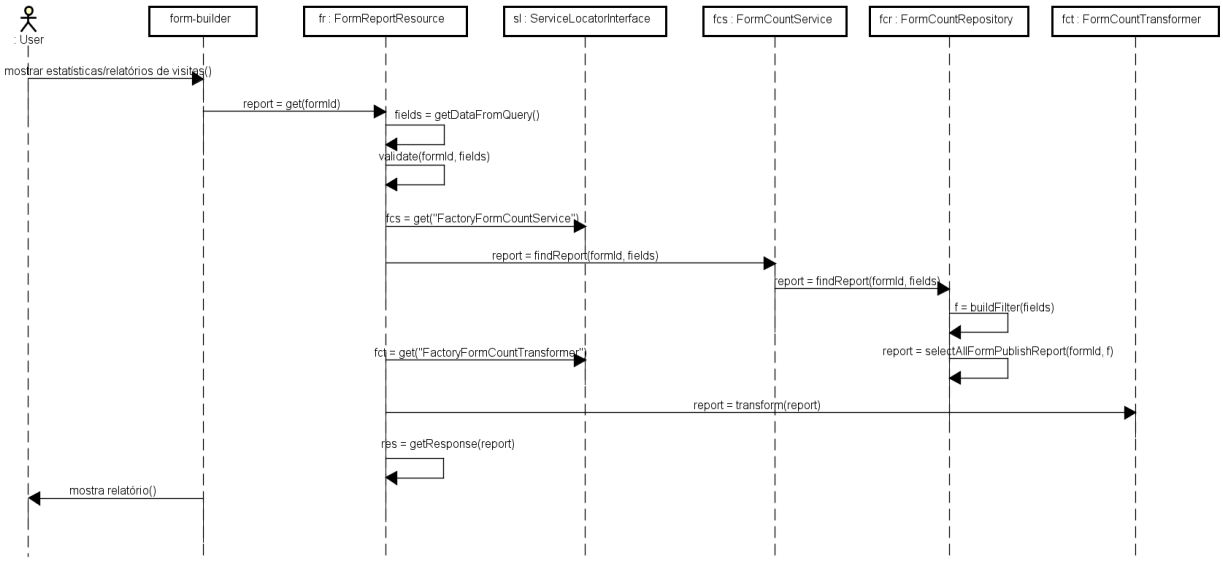

Figura 48 Diagrama de sequência: Ver estatísticas/relatório de visitas

US12 – Analisar Resultados do Formulário

A análise de resultados do formulário deve permitir ao utilizador visualizar as respostas que foram dadas no formulário. Basta para isso utilizar o recurso *AnswerResource*, as entidades e classes associadas foram criadas no desenho do fluxo de submissão. Este requisito também diz que deve ser visualizada informação estatística sobre cada campo, tal é conseguido recorrendo ao sistema externo de *web analytics*. A *Figura 49* representa a utilização do serviço de respostas para obter as respostas obtidas num determinado formulário.

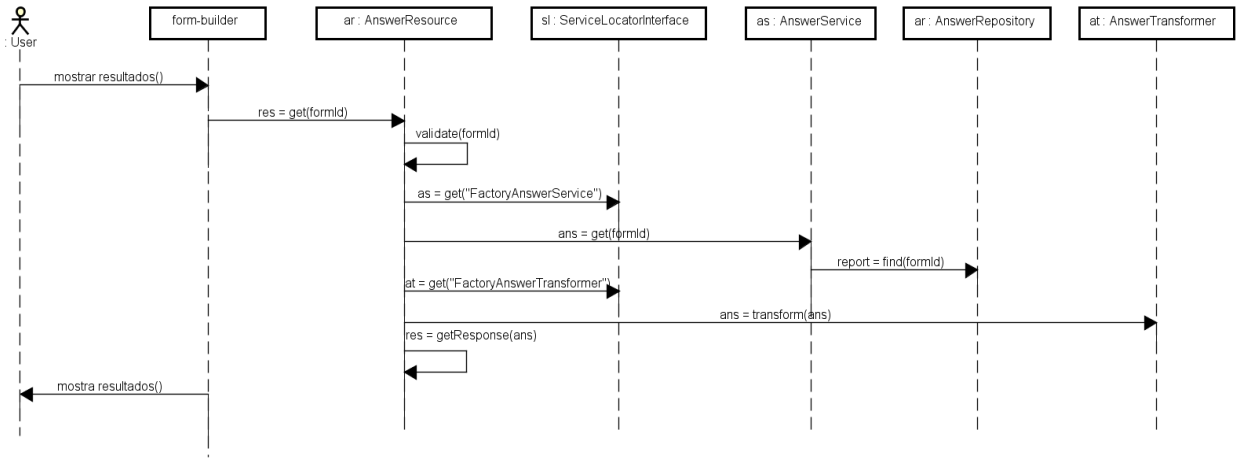

Figura 49 Diagrama de sequência: Análise de resultados do formulário

Anexo B Questionário

Sistema de Gestão de Formulários

Questionário

Esta sessão tem como objetivo avaliar a abordagem proposta para o novo sistema de gestão de formulários da plataforma E-goi em comparação com o antigo, assim como as funcionalidades propostas e métricas de análise de resultados. É expectável que no fim desta sessão esteja claro o conceito de sistema de gestão de formulários e abordagem a seguir.

Na tabela seguinte está descrita a estruturação desta sessão, com os tempos estimados para cada tarefa, num total previsto de 45 minutos.

1	Descrição da sessão	5 min
2	Descrição da abordagem proposta	10 min
3	Questionário	25 min
4	Conclusão da sessão	5 min

Os questionários contém maioritariamente afirmações no qual as deve avaliar numa escala de 1 a 5 (discordo totalmente, discordo parcialmente, indiferente, concordo parcialmente, concordo totalmente) fazendo um círculo em volta da opção pretendida. Os questionários contêm também perguntas de escolha múltipla, no qual deve assinalar as opções que achar indicadas, e de resposta aberta.

Sistema de Gestão de Formulários

Questionário

	Discordo totalmente	Discordo parcialmente	Indiferente	Concordo parcialmente	Concordo totalmente
1. Um formulário é uma ferramenta utilizada para requisitar informação através de um conjunto campos a serem preenchidos.	1	2	3	4	5
2. Um formulário é uma página *Web* que pode ou não requisitar informação.	1	2	3	4	5
3. Um sistema de gestão de formulários permite criar o formulário através dos seus campos, personaliza-lo, publicar o formulário e analisar os resultados obtidos.	1	2	3	4	5
4. Um sistema de gestão de formulários deve permitir a criação do formulário assim como a página em que o mesmo está integrado.	1	2	3	4	5
5. Existe diferença entre um sistema de gestão de formulários aplicado no contexto de *marketing* digital e um sistema de gestão de formulários sem contexto concreto.	1	2	3	4	5

6. Que funcionalidades considera importantes?

Criar formulário	☐
Criar formulário inscrição	☐
Criar formulário recomendação	☐
Criar questionário	☐
Inserir Campo	☐
Definir estilo	☐
Publicar	☐
Definir ações de submissão	☐
Configurar mensagem de submissão	☐
Definir lógica condicional entre campos	☐
Mapear lista de contactos com formulários	☐
Associar campo do formulário a campo da lista	☐
Analisar resultados (visitas)	☐
Analisar resultados (submissões)	☐
Inserir código próprio	☐
Enviar notificações de submissão	☐
Submeter	☐
Inserir elemento gráfico	☐
Definir limitações do formulário	☐
Definir validações dos campos	☐
Definir traduções	☐
Outras Quais?	

_____

	Muito Mau	Mau	Suficiente	Bom	Muito Bom
7. Como classifica as funcionalidades propostas para o novo sistema?	1	2	3	4	5
8. Como classifica as funcionalidades do sistema antigo?	1	2	3	4	5

9. Que métricas de análise de resultados considera importantes?

Visitas	☐
Localização	☐
Sistema Operativo	☐
Data	☐
Dispositivo	☐
Duração	☐
Fontes/Referentes	☐
Taxa de rejeição	☐
Conversões	☐
Ações/Eventos	☐
Objetivos	☐
Submissões	☐
Outra Qual?	

_____

	Muito Mau	Mau	Suficiente	Bom	Muito Bom
10. Como classifica as métricas de análise de resultados do antigo sistema de formulários?	1	2	3	4	5
11. Como classifica as métricas de análise de resultados do novo sistema de formulários?	1	2	3	4	5
12. No geral como classifica a abordagem proposta para o novo sistema em comparação com o antigo?	1	2	3	4	5

Observações:

_____
_____
_____
_____

Data __/__/__ Participante nº __