

UNIVERSIDADE DO MINHO

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA

ENGENHARIA DE APLICAÇÕES

(2º SEMESTRE / 4º ANO)

Free Tour

Grupo 7:

Diogo Braga (a82547)
Francisco Reinolds (a82982)
João Silva (a82005)
Nelson Sousa (a82053)

Conteúdo

Lista de Figuras	iii
Lista de Tabelas	v
1 Introdução	1
1.1 Contextualização	1
1.2 Motivação e Objetivos	1
1.3 Definição da Estrutura do Relatório	2
2 Análise de Requisitos	3
2.1 Stakeholders	3
2.1.1 O Cliente	3
2.1.2 O <i>Customer</i>	3
2.1.3 Outros <i>Stakeholders</i>	4
2.1.4 Os Utilizadores	6
2.1.5 Personas	8
2.1.6 Impacto na conceção	10
2.2 Modelo de Domínio	12
2.3 Identificação de Requisitos	14
2.3.1 Requisitos Funcionais	14
2.3.2 Requisitos Não-funcionais	17
2.4 Diagrama de Use Cases	19
2.5 Modelos de Tarefas	20
2.5.1 Publicitação duma Tour	20
2.5.2 Inscrição numa Tour	22
3 Conceção	24
3.1 Arquitetura do Sistema	24
3.2 PIM	25
3.3 PSM	27
3.4 Base de Dados	29

CONTEÚDO

3.5	Prototipagem da Interface	31
3.5.1	Páginas mais Relevantes	32
3.5.2	Técnicas de Avaliação	35
4	Desenvolvimento	39
4.1	Tecnologias a Utilizar	39
4.1.1	Spring	39
4.1.2	Hibernate	39
4.1.3	Vue	39
4.1.4	Leaflet	39
4.1.5	JWT	40
4.1.6	SendGrid	40
4.1.7	QR code API	40
4.1.8	DataSet de Cidades	40
4.2	Decisões Arquiteturais	41
4.2.1	Imagens	41
4.2.2	Cidades	41
4.3	Endpoints	42
5	Deployment e Testes	47
5.1	Deployment	47
5.1.1	Local	48
5.1.2	Máquinas separadas - Cloud	50
5.1.3	Scripts de Criação	53
5.2	Testes	55
5.2.1	Home	55
5.2.2	Login	56
5.2.3	Search	57
6	Conclusão	60
6.1	Dificuldades	60
6.2	Trabalho Futuro	61

Lista de Figuras

2.1	Mapa de <i>Stakeholders</i>	5
2.2	Persona Mila Henry	8
2.3	Persona José Santos	9
2.4	Modelo de Domínio - Parte 1	12
2.5	Modelo de Domínio - Parte 2	13
2.6	Diagrama de Use Cases	19
2.7	Tarefa Publicar uma Visita (antes da Plataforma)	20
2.8	Tarefa Publicar uma Visita (com a Plataforma)	21
2.9	Tarefa Inscrever em Visita (antes da Plataforma)	22
2.10	Tarefa Inscrever em Visita (com a Plataforma)	23
3.1	PIM parte 1	25
3.2	PIM parte 2	25
3.3	PIM parte 3	26
3.4	PSM - Entidades	27
3.5	PSM - Controladores e Serviços	28
3.6	Modelo lógico da Base de Dados - parte 1	29
3.7	Modelo lógico da Base de Dados - parte 2	30
3.8	Página de Criação de uma Tour	32
3.9	Página de uma Tour	34
3.10	Modal de Inscrição de uma Tour	35
3.11	Cognitive Walkthrough - Inscrever em Tour	36
3.12	Homepage	37
3.13	Página de Pesquisa de Tours	38
5.1	Imagens Docker dos componentes	48
5.2	Infraestrutura Local usando Docker-Compose	48
5.3	Infraestrutura na Cloud	50
5.4	Script de Criação	54
5.5	Ambientes de testes	54
5.6	Testes no endpoint home	55

LISTA DE FIGURAS

5.7	Testes no endpoint home usando um utilizador autenticado	56
5.8	Testes no endpoint login	57
5.9	Testes no endpoint search	58
5.10	Testes no endpoint search utilizando filtros	59

Lista de Tabelas

2.1	Perfil dum Turista	6
2.2	Perfil dum Guia	6
2.3	Perfil dum Administrador	7

Capítulo 1

Introdução

O presente relatório detalha um sistema criado para monitorizar o agendamento e realização de *tours*. Este foi proposto no âmbito das Unidades Curriculares de Arquiteturas Aplicacionais e Sistemas Interativos, inseridas no perfil de especialização em Engenharia de Aplicações.

1.1 Contextualização

Quem viaja nos dias de hoje procura mais do que ver os locais por onde passa. Um factor importante é entender o que se está a ver, a sua história e as razões por detrás de cada monumento erguido. Para isso existem as visitas guiadas, onde um guia cobra um determinado montante para levar um grupo a visitar determinado local, explicando a história e curiosidades do mesmo. O problema destas visitas é possuir, por norma, um valor elevado. Devido a tal, nos tempos mais recentes, começou a tornar-se popular um conceito chamado **free tour**.

Uma *free tour* é um plano com acesso guiado e gratuito a variadas atrações turísticas, anunciada por qualquer pessoa que tenha intenções de fazer de guia. As pessoas inscrevem-se na *tour*, e obtêm um guia sem pagar preços exorbitantes. No fim da *tour* o guia tem a liberdade de recolher uma gorjeta pelo serviço prestado, não estando as pessoas obrigadas a fazê-lo. Perante esta situação, surgiu a ideia de informatizar todo este processo.

1.2 Motivação e Objetivos

O objetivo deste trabalho é criar uma plataforma web para alojar este serviço, isto é, um sistema para monitorizar o agendamento e realização de *free tours*. Nesta plataforma teríamos dois tipos de entidades, os **guias** e os **turistas**.

1.3. DEFINIÇÃO DA ESTRUTURA DO RELATÓRIO

Os guias irão ter a possibilidade de anunciar na plataforma as suas *free tours*. Estes terão o papel de especificar o local, a data, o percurso, a duração e quaisquer outras informações que considerem relevantes. Os turistas poderão visualizar as *free tours*, filtrando-as por local, duração, ou pela classificação do guia. Para além disso será possível aos turistas inscreverem-se na visita. No final, estes devem avaliar o guia, e caso queiram, atribuir-lhe duma gorjeta, sendo esta oferta é realizada pessoalmente no final da *tour*. Ambas as entidades são registadas na plataforma, sem qualquer restrição, de forma a que qualquer pessoa possa ser guia ou turista.

A criação deste software vem no sentido de facilitar as viagens de qualquer pessoa, assim como torna-las ainda mais interessantes. Tal é fundamentado na realidade de que existem imensas pessoas que se vêem impossibilitadas de obter mais conhecimento dumha determinada cidade por causa dos custos que estão envolvidos. A informatização deste processo, em particular o contacto entre guias e turistas de forma a agendar um plano de viagem, é algo que facilita em larga margem a que estas *tours* de facto aconteçam. Sem uma plataforma concebida para auxiliar este agendamento, as *tours* aconteciam em menor número. É também neste sentido que surge este projeto, com intenções globais de facilitar a difusão de cultura entre pessoas.

1.3 Definição da Estrutura do Relatório

Este relatório está dividido em quatro principais capítulos, divididos com base numa abordagem centrada no utilizador. Primeiro, a análise de requisitos, seguindo-se a conceção e o desenvolvimento e, por final, o *deployment* e testes. Existe, depois, um capítulo para escrita da conclusão.

Na análise de requisitos, são apresentados os *stakeholders* e o trabalho de pesquisa realizado em volta do utilizador, seguindo-se a apresentação dos modelos arquiteturais e comportamentais, assim como a identificação dos requisitos realizada. Na conceção são abordados os diagramas de classe, a base de dados e a prototipagem da interface. No desenvolvimento são referenciadas as tecnologias utilizadas e os *endpoints* em causa na aplicação. De seguida, são apresentadas as infraestruturas utilizadas e discutidos os testes realizados ao sistema.

Capítulo 2

Análise de Requisitos

2.1 Stakeholders

Nesta secção vamos descrever as pessoas que têm interesse no produto, ou seja, os **Stakeholders**. Os *Stakeholders* não são apenas os indivíduos ou instituições que pretendem investir capital no produto.

Todos aqueles que pretendam utilizar ou tirar qualquer outro partido do produto, podem ser considerados *Stakeholders*, por isso neste conjunto inserem-se os clientes, os *customers* e ainda os utilizadores.

2.1.1 O Cliente

Sendo que o cliente é definido por alguém que faz o investimento no produto, não há ninguém que respeite esta definição tão bem como o nosso próprio grupo de trabalho.

Após bem definir as nossas motivações, já descritas no capítulo 1, decidimos investir no projeto e criar a plataforma desejada.

2.1.2 O *Customer*

O *customer* é definido como sendo alguém que paga para adquirir/utilizar o sistema, quando este está pronto para ser utilizado. Dada esta noção torna-se claro quem são os *customers* deste projeto. Este sistema é projetado para as pessoas que pretendem ser guias e desejam publicitar as suas *tours* no sistema.

No fundo o *customer* é também um consumidor final e utilizador, que paga o serviço de publicitação prestado pelo software produzido.

2.1. STAKEHOLDERS

2.1.3 Outros *Stakeholders*

Tendo em conta a dimensão do projeto, vão ser algumas as partes interessadas. Assim, foi criada uma lista dos principais *Stakeholders*.

- **Turistas** - interessados devido ao facto de puderem participar numa visita guiada sem que haja um compromisso monetário.
- **Guias** - teriam interesse devido ao facto de poderem publicitar-se assim como as suas visitas guiadas numa plataforma dedicada a este propósito.
- **Administradores** - teriam interesse devido ao facto de poderem interagir com o sistema para resolverem problemas em produção.
- **Direções gerais de património cultural** - tem interesse devido ao facto do património cultural que detém ser visitado durante as visitas que irão decorrer.
- **Equipa de Desenvolvimento** - interessados em desenvolver o produto e em fazer a manutenção deste. Pode ser dividida por **Programadores** e **Mantenção**.
- **Equipa de Marketing** - interessados em publicitar o produto.
- **Equipa de Suporte** - interessados em realizar tarefas de suporte aos utilizadores do produto.
- **Equipa de Auditoria** - uma vez que existem pagamentos associados, existe interesse em gerir e examinar as tarefas desenvolvidas.

Para uma análise mais aprofundada podemos criar um *Stakeholder Map* que permite observar o papel de cada *Stakeholder* no projeto.

2.1. STAKEHOLDERS



Figura 2.1: Mapa de *Stakeholders*

2.1. STAKEHOLDERS

2.1.4 Os Utilizadores

Os utilizadores representam grupos de pessoas que irão usufruir do produto. É por isso, importante documentá-los detalhadamente para mais tarde podermos retirar requisitos que sejam centrados nos utilizadores e ter um produto que esteja conectado com as necessidades do público-alvo.

De seguida apresentam-se alguns perfis de utilizador.

Turistas:

Informação sobre o Utilizador	Grupo Etário: 18+ Formação Académica: Qualquer Competências: Generalizadas Tipo de utilizador/experiência: Qualquer
Utilização do Sistema	Opcional ou obrigatória: Opcional
Informação sobre Trabalho	Classe de Utilizador: Direto Descrição do Trabalho: Navegar no site Tarefas Principais: Procurar/Inscrever-se em visitas Responsabilidades: Sem Responsabilidade

Tabela 2.1: Perfil dum Turista

Guias:

Informação sobre o Utilizador	Grupo Etário: 18+ Formação Académica: Qualquer Competências: Generalizadas Tipo de utilizador/experiência: Qualquer
Utilização do Sistema	Opcional ou obrigatória: Opcional
Informação sobre Trabalho	Classe de Utilizador: Direto Descrição do Trabalho: Navegar no site Tarefas Principais: Publicar Tours Responsabilidades: Comparecer à Tour

Tabela 2.2: Perfil dum Guia

2.1. STAKEHOLDERS

Administradores:

Informação sobre o Utilizador	Grupo Etário: 20+ Formação Académica: Área da informática Competências: Conhecer o modo de funcionamento da plataforma Tipo de utilizador/experiência: utilizador experiente
Utilização do Sistema	Opcional ou obrigatória: obrigatória
Informação sobre Trabalho	Classe de Utilizador: Direto Descrição do Trabalho: Analisar plataforma Tarefas Principais: Eliminar possíveis erros Responsabilidades: não ser um BOFH

Tabela 2.3: Perfil dum Administrador

2.1. STAKEHOLDERS

2.1.5 Personas

Nesta secção apresentamos duas *personas* que retratam o conjunto de utilizadores do sistema, que tanto podem ser guias como turistas.

Turista

Figura 2.2: Persona Mila Henry



¹

Esta é a **Mila Henry**. A Mila nasceu em Londres e tem 36 anos. Estudou Biologia e Geologia na universidade de Cambridge e no momento é professora do Ensino Secundário numa escola em Londres.

Sendo que nos tempos livres gosta de viajar, conhecer novos lugares bem como a história que os rodeia, e como o gosta de fazer a par da sua família, necessita duma plataforma que lhe ofereça um leque de opções de visitas guiadas para o seu destino. Como os seus filhos são crianças percebem apenas a sua língua nativa pelo que a **Mila**

tende a procurar apenas visitas que sejam orientadas em inglês. Como as suas viagens são na sua maioria com a sua família, a **Mila** gostaria de poder inscrever-se numa visita incluindo os seus familiares.

Para além disto ela gosta de garantir que nenhuma das suas escolhas é um tiro no escuro e prefere saber tudo o que a visita inclui e por onde passará assim como o guia que a encaminhará durante a visita. Se algo correr mal e a família não possa realizar a visita, a **Mila** gostaria de poder cancelar a sua inscrição de modo a não ocupar lugares da visita que poderão ser ocupados por pessoas que de facto possam ir.

A **Mila** gosta de utilizar a *internet* e de fazer buscas personalizadas dados alguns critérios. Para além disso a **Mila** gosta de utilizar o seu Mail para conferir toda a sua vida e perceber que eventos se irão realizar num futuro próximo, funcionando o Mail como um *reminder* e agenda.

Quando está com as suas amigas gosta de partilhar as viagens que tem feito e mostra-lhes os sítios por onde passou, assim como as experiências que teve.

¹Imagem gerada em <https://www.thispersondoesnotexist.com/> devido a direitos de imagem.

2.1. STAKEHOLDERS

Guia e Turista

Figura 2.3: Persona José Santos



²

Este é o **José Santos**. O José nasceu em Guimarães e tem 21 anos. Atualmente estuda Estatística Aplicada na Universidade do Minho e no seu tempo livre gosta de ganhar algum dinheiro a fazer visitas guiadas por Guimarães uma vez que esta é a sua cidade natal e conhece tudo sobre ela.

Basicamente o **José** amealha algum dinheiro para poder realizar as suas viagens e conhecer outros sítios, participando também em muitas visitas guiadas pois é dessa forma que aumenta a sua experiência e retira várias lições.

O **José** nasceu numa geração al-

tamente informada sobre os perigos ambientais das ações de cada um, e por isso, ao invés de imprimir papéis de propaganda para colar em cafés, vitrinas, etc, preferia ter uma plataforma na qual pagasse a publicitação do seu trabalho até porque vivemos na era da tecnologia e este é um meio de disseminação de informação bem mais abrangente do que qualquer papel colado num café possa ser.

O **José** sabe falar português, pois é a sua língua natal, mas também é fluente em espanhol e inglês pelo que consegue realizar visitas guiadas nestas línguas. Sempre que realiza uma visita, o **José** gosta de manter uma linha de pensamento e de relato de acontecimentos pelo que tende a realizar sempre o mesmo percurso nas suas visitas.

Como o **José** é um jovem relativamente ocupado, visto ainda ser universitário, tende a esquecer-se de muitas tarefas que tem pendentes e por isso, gosta de utilizar plataformas que lhe enviem mails a lembrar-lhe de eventos e que lhe mostrem os eventos que já passaram, assim como aqueles que ainda estão por vir.

²Imagem gerada em <https://www.thispersondoesnotexist.com/> devido a direitos de imagem.

2.1. STAKEHOLDERS

2.1.6 Impacto na conceção

Após analisar as *Personas* existentes facilmente se identificaram alguns aspectos que deveriam constar na plataforma a desenvolver.

- Apresentar opções de visitas guiadas por destino, e por uma série de critérios.
 - A Mila gostava que lhe fosse oferecido um leque de opções de visitas guiadas para o seu destino.
 - A Mila gosta de fazer buscas personalizadas dados alguns critérios.
- As visitas devem ter a informação acerca dos idiomas em que será feita.
 - A Mila tem filhos pequenos que apenas entendem a sua língua nativa.
 - O José é fluente em 3 línguas, podendo realizar a visita em qualquer uma destas 3 línguas.
- A inscrição numa visita deve permitir a inscrição em grupo.
 - A Mila faz viagens em família logo tem que inscrever todos os membros nas visitas.
- A plataforma deve mostrar toda a informação relativa a uma visita.
 - A Mila é controladora e gosta de saber tudo acerca daquilo em que participa.
- A inscrição numa visita deve poder ser cancelada.
 - A Mila gostaria de poder cancelar a sua inscrição de modo a não ocupar lugares da visita.
- Desenvolvimento duma aplicação Web.
 - A Mila gosta de utilizar a internet.
- A plataforma deve enviar notificações de eventos futuros para o Mail.
 - A Mila gosta de utilizar o Mail como *reminder* e agenda
 - O José é ocupado e precisa de algo que o relembrre dos eventos em que irá participar.
- A plataforma deve guardar histórico de visitas realizadas.
 - A Mila gosta de mostrar às suas amigas os sítios que visitou.

2.1. STAKEHOLDERS

- A plataforma deve permitir a um utilizador poder ser guia e turista ao mesmo tempo.
 - O José ganha dinheiro a realizar visitas guiadas para que possa fazer as suas viagens e participar noutras visitas.
- A plataforma deve permitir guardar visitas já realizadas para poderem ser reutilizadas no futuro.
 - O José gosta de manter uma linha de pensamento e de relato de acontecimentos pelo que tende a realizar sempre o mesmo percurso nas suas visitas.

2.2 MODELO DE DOMÍNIO

2.2 Modelo de Domínio

Neste modelo de domínio mostramos os diferentes conceitos e interações presentes no nosso problema. Como podemos observar possuímos 2 tipos de utilizadores o guia e o turista. Um com o objetivo de organizar tours e outro com o objetivo de participar nelas. O conceito central do problema é então a tour que possui diversas propriedades como o roteiro, a data, a categoria, as avaliações e outras especificadas no presente diagrama.

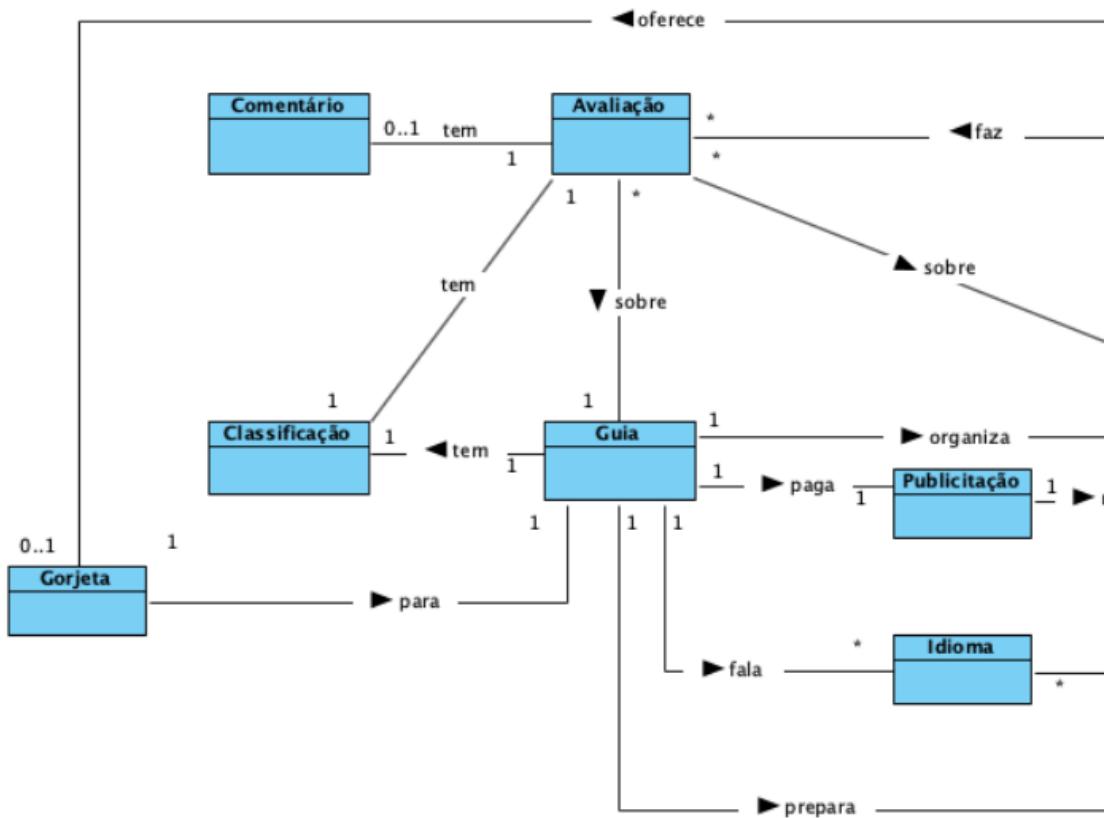


Figura 2.4: Modelo de Domínio - Parte 1

2.2. MODELO DE DOMÍNIO

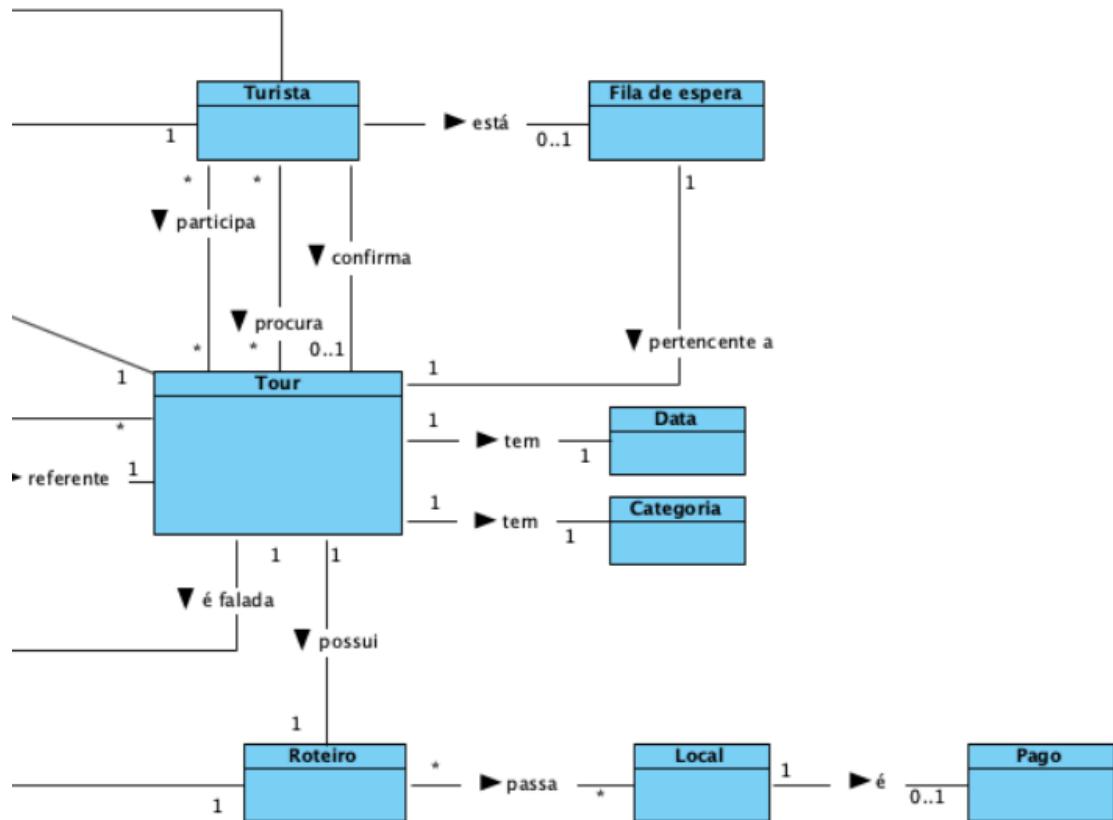


Figura 2.5: Modelo de Domínio - Parte 2

2.3 Identificação de Requisitos

Para realizar a conceção de um sistema completo e com um bom funcionamento é fulcral, antes da implementação, construir um plano de requisitos referentes à aplicação e às funcionalidades que esta necessita. Esta secção descreve, portanto, os requisitos do sistema.

2.3.1 Requisitos Funcionais

Inicialmente, a descrição passa pelos requisitos funcionais, que detalham as características que a aplicação deve possuir para cumprir com o seu propósito e, ainda, um conjunto de requisitos que se focam na manutenção da aplicação. Como nesta aplicação existem vários tipos de utilizadores, os requisitos associados a cada um vão ser enunciados separadamente.

Nesta primeira listagem, são enunciados os requisitos relativos ao guia:

- **RF1** - O utilizador deve ser capaz de registar-se no sistema, indicando:
 - email (obrigatório);
 - password (obrigatório);
 - nome (obrigatório);
 - data de nascimento;
 - fotografia de perfil recente;
 - número telefónico;
 - competências linguísticas;
 - informações adicionais.
- **RF2** - O utilizador deve ser capaz de autenticar-se no sistema, indicando:
 - email;
 - password.
- **RF3** - O utilizador deve poder recuperar os seus dados de acesso ao sistema;
- **RF4** - O utilizador deve conseguir visualizar o seu perfil, incluindo:
 - email;
 - password;
 - nome;

2.3. IDENTIFICAÇÃO DE REQUISITOS

- data de nascimento;
 - fotografia de perfil recente;
 - número telefónico;
 - competências linguísticas;
 - classificação;
 - informações adicionais;
 - *tours* em que vai participar;
 - *tours* em que participou;
 - *tours* em que é responsável.
- **RF5** - O utilizador deve ser capaz de editar o seu perfil;
 - **RF6** - O utilizador, caso possua todos os campos do perfil preenchidos, deve conseguir publicar uma *tour*, indicando:
 - nome;
 - localização;
 - data e hora de início;
 - duração;
 - idioma(s) falado(s);
 - descrição detalhada;
 - número máximo de turistas;
 - roteiro com os pontos turísticos a visitar;
 - imagens relativas ao roteiro.
 - **RF7** - O utilizador deve ser capaz de cancelar a tour até 24 horas antes desta se realizar, caso seja o organizador;
 - **RF8** - O utilizador deve conseguir visualizar a lista de *tours* referentes a uma localização;
 - **RF9** - O utilizador deve poder filtrar as *tours* que existem numa localização, por:
 - data;
 - classificação;
 - categoria;

2.3. IDENTIFICAÇÃO DE REQUISITOS

– linguagem.

- **RF10** - O utilizador deve conseguir visualizar as informações relativas a uma *tour* em específico;
- **RF11** - O utilizador deve ser capaz de inscrever-se para participar numa *tour*, desde que não seja o organizador e que a *tour* não se encontre lotada;
- **RF12** - O utilizador deve conseguir colocar-se em fila de espera, caso a *tour* se encontre lotada;
- **RF13** - O utilizador, caso se encontre em fila de espera e surja uma vaga, deve ficar associado à lista de inscritos da respetiva *tour*;
- **RF14** - O utilizador deve conseguir consultar o perfil do guia responsável pela *tour*;
- **RF15** - O utilizador deve ser capaz de cancelar a sua inscrição na tour até 24 horas antes desta se realizar;
- **RF16** - O utilizador deve poder avaliar a *tour* após esta ter sido concluída, de forma quantitativa e/ou com um comentário;
- **RF17** - O utilizador deve ser capaz de remover a sua conta do sistema.

A um nível de administração, o sistema apresenta os seguintes requisitos:

- **RF18** - O administrador deve ser capaz de autenticar-se no sistema;
- **RF19** - O administrador deve ser capaz de adicionar e remover utilizadores;
- **RF20** - O administrador deve ser capaz de remover *tours*;
- **RF21** - O administrador deve poder eliminar comentários realizados.

2.3. IDENTIFICAÇÃO DE REQUISITOS

2.3.2 Requisitos Não-funcionais

Os requisitos não funcionais são requisitos relacionados com a utilização da aplicação em termos de desempenho, confiabilidade, disponibilidade, manutenção e tecnologias envolvidas. Estes requisitos dizem respeito a como as funcionalidades são entregues ao utilizador do software. Nesta secção dividimos os requisitos em três conjuntos: de produto, organizacionais e externos.

Requisitos de Aparência

- O sistema terá um visual simples e elegante (validado pelos utilizadores) de modo a simplificar a experiência dos utilizadores;
- O sistema terá predominantemente uma cor branca.

Requisitos de Usabilidade e Humanidade

- Qualquer pessoa que possua o 4º ano de escolaridade deverá, após uma curta sessão de treino, conseguir utilizar todas as funcionalidades do sistema;
- Toda a informação será clara para consulta, inclusive para pessoas daltónicas;

Requisitos de Performance

- O sistema terá de ser preciso quanto aos locais que nele constem, visto serem pontos de encontro entre os utilizadores (erro de precisão máximo de 5 metros);
- O sistema terá alta taxa de disponibilidade, com valores superiores a 95%;
- O sistema deverá continuar a funcionar caso um dos servidores falhe e exista pelo menos um funcional, mantendo assim redundância;
- O sistema será capaz de aguentar grandes cargas de clientes, criando assim maior fiabilidade (suporte de inscrições na ordem das centenas por minuto e de publicações na ordem das dezenas por minuto);
- O sistema será capaz de suportar um aumento no número de clientes, prevendo assim tendências de crescimento que podem acontecer (suporte de aumentos na ordem dos 5% de clientes mensalmente).

2.3. IDENTIFICAÇÃO DE REQUISITOS

Requisitos Operacionais e de Ambiente

- O sistema funcionará com pouco acesso à Internet, necessitando apenas de uma conexão estável com pelo menos 1Mbps tanto de upload, como de download;
- O sistema deverá funcionar corretamente em todos os browsers, nas suas mais recentes versões, que suportem todas as tecnologias utilizadas na plataforma.

Requisitos de Manutenção e Suporte

- O sistema será modular de maneira a garantir uma manutenção mais acessível, encontrando-se separada em 3 camadas (apresentação, negócio e dados);
- A framework da camada de apresentação utilizada será o `Vue.js`;
- A framework da camada de negócio a ser utilizada é o `Spring`;
- As frameworks da camada de dados a serem utilizadas serão o `Hibernate` e o `PostgreSQL`;
- O código fonte do sistema deverá estar bem documentado, apresentando no mínimo 1 comentário por cada função/método desenvolvidos que explique as variáveis utilizadas;
- O utilizador saberá utilizar o sistema após uma curta sessão instrucional.

Requisitos de Segurança

- O sistema terá de ter acesso a informações sensíveis do utilizador, como detalhes da sua conta;
- Não será divulgadas as informações de nenhum utilizador, de modo a garantir a privacidade dos dados;

2.4. DIAGRAMA DE USE CASES

2.4 Diagrama de Use Cases

Um use case pode ser definido como um cenário que descreve o que o utilizador espera do sistema. Desta forma, cada use case representa uma tarefa que envolve interação com o sistema. O diagrama de Use Cases fornece, portanto, uma visão geral de toda a interação possível entre o utilizador e o sistema.

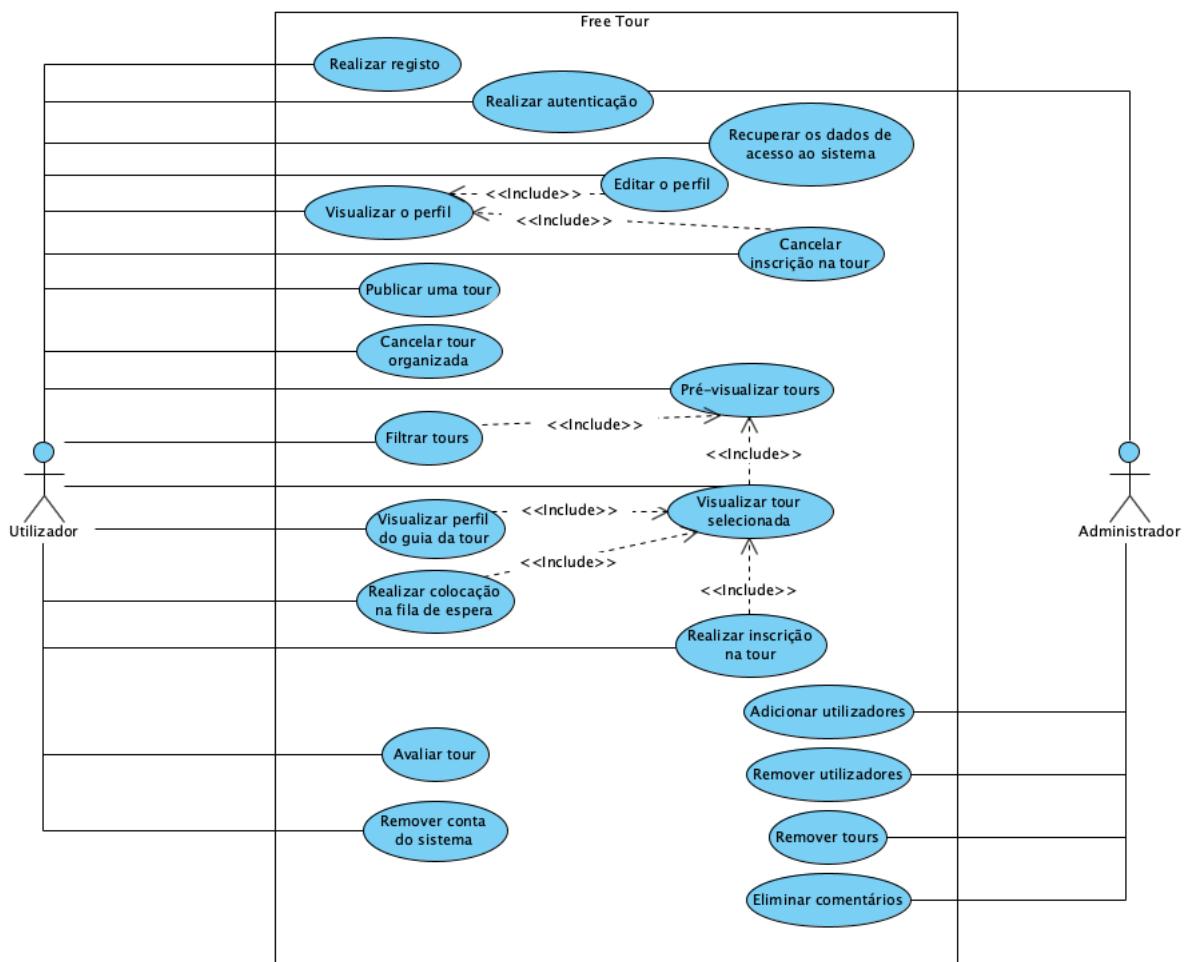


Figura 2.6: Diagrama de Use Cases

2.5 Modelos de Tarefas

Após analisar os utilizadores do sistema, verificamos também a forma como realizavam as principais tarefas que se tornarão funcionalidades da plataforma, ou seja, a publicitação duma visita e a inscrição numa visita.

2.5.1 Publicita o duma Tour

Antes da existência da plataforma, um guia fazia o seguinte.

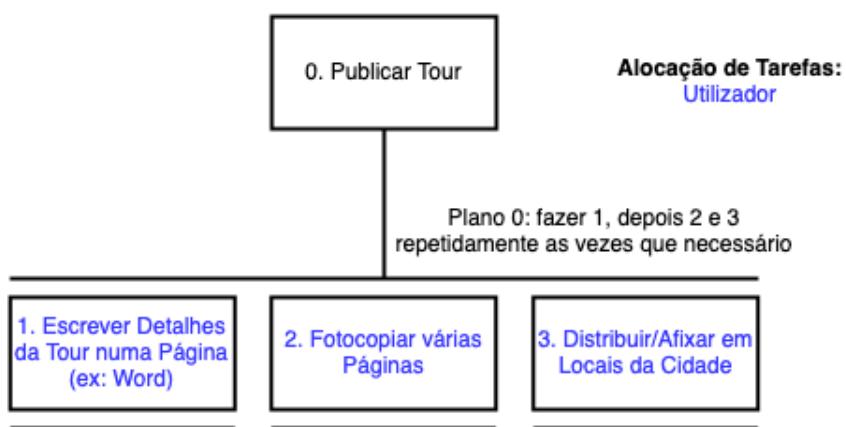


Figura 2.7: Tarefa Publicar uma Visita (antes da Plataforma)

Após a existência da plataforma, um guia faz o seguinte.

2.5. MODELOS DE TAREFAS

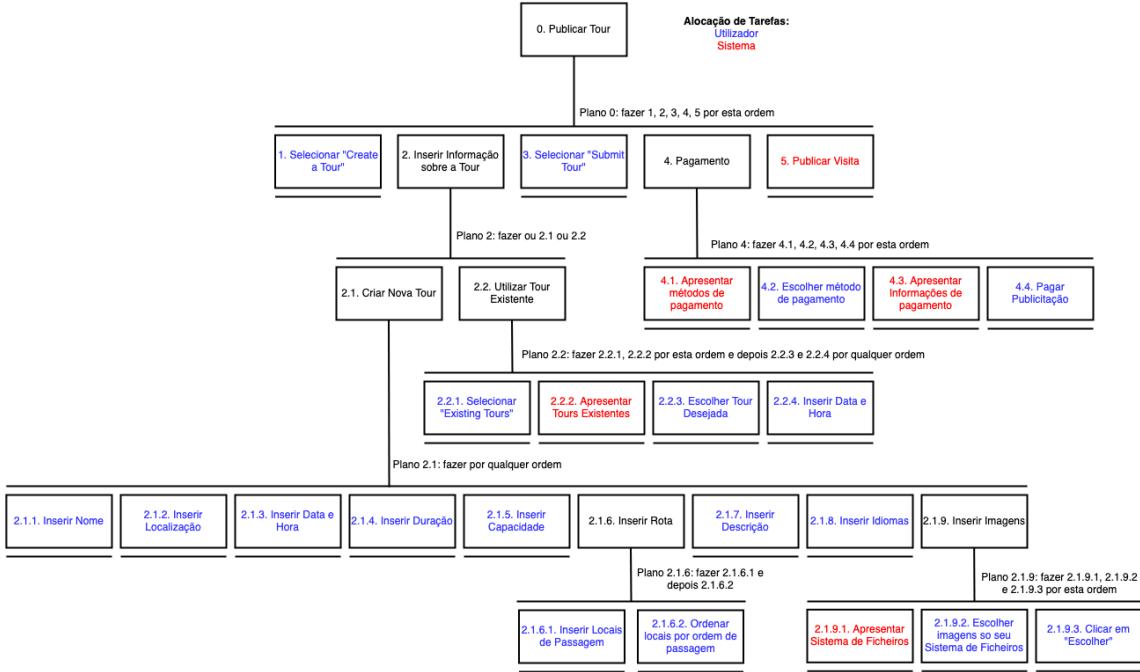


Figura 2.8: Tarefa Publicar uma Visita (com a Plataforma)

Na verdade, parece que a quantidade de funções que o guia tem que realizar antes da existência da plataforma são muito menos do que com a existência da plataforma. Se assim fosse, a utilização da plataforma não traria nenhum benefício aquando da publicitação duma *tour*. Neste caso, as funções associadas a esta tarefa antes da existência duma plataforma, são funções que requerem muito tempo para serem realizadas (ex: distribuir/afixar publicidade por vários locais; imprimir fotocópias) e um gasto monetário indefinido (ex: imprimir um número variado de fotocópias).

Com a plataforma, apesar de termos um número de funções maiores, todas elas são bem mais curtas e muito menos dispendiosas a nível temporário e também monetário, ficando uma publicitação duma *tour* à distância de poucos cliques.

2.5. MODELOS DE TAREFAS

2.5.2 Inscrição numa Tour

Antes da existência da plataforma, um turista para se inscrever numa *tour* teria de fazer o seguinte.

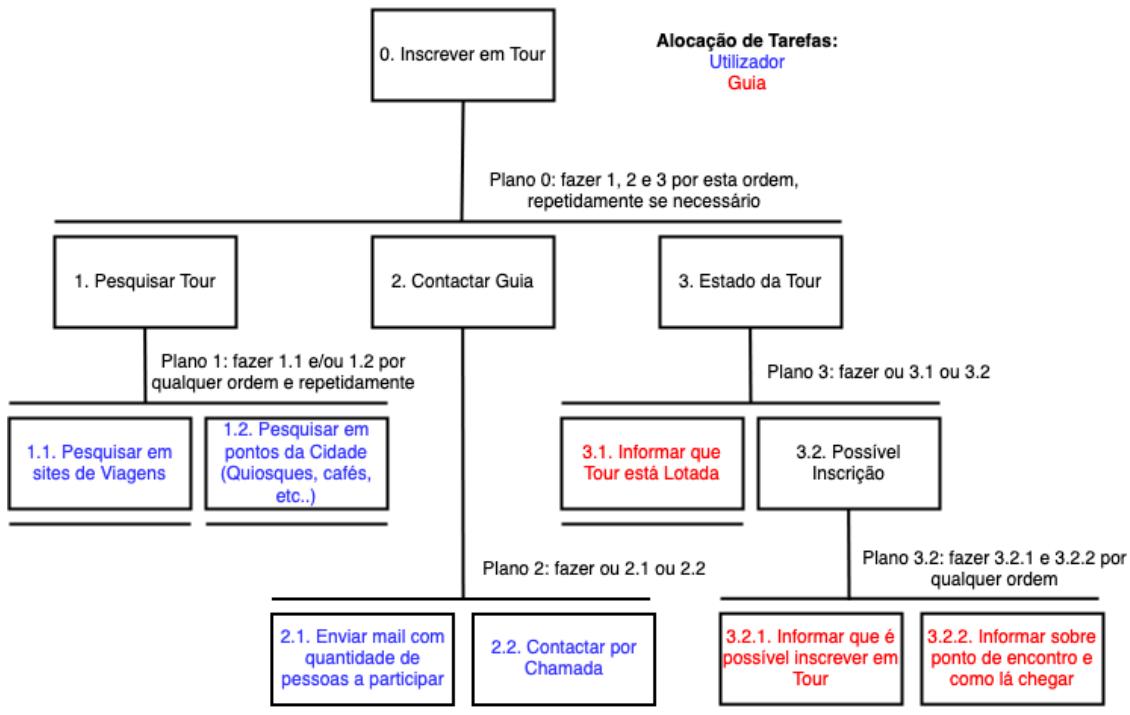


Figura 2.9: Tarefa Inscrever em Visita (antes da Plataforma)

Após a existência da plataforma, um turista para se inscrever numa *tour* tem de fazer o seguinte.

2.5. MODELOS DE TAREFAS

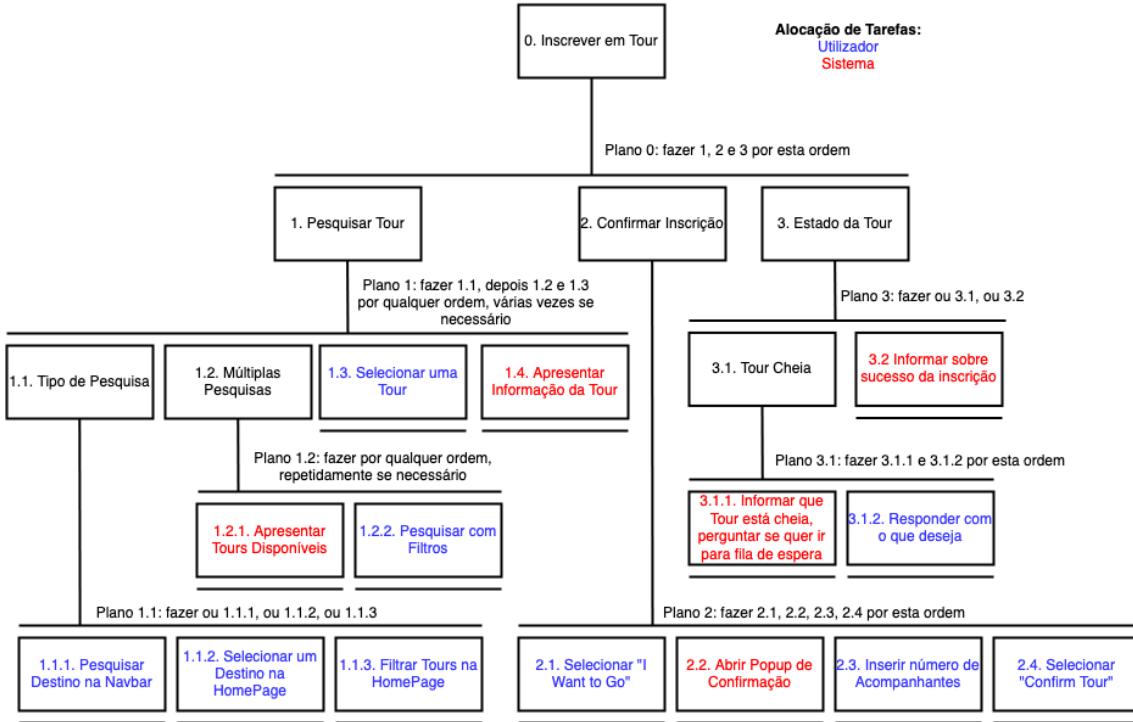


Figura 2.10: Tarefa Inscrever em Visita (com a Plataforma)

Da mesma forma que na publicitação duma *tour*, as funções associadas a esta tarefa antes da existência da plataforma exigem mais tempo e transtornos para serem finalizadas com sucesso. Apesar da tarefa com a plataforma parecer mais extensa, tal não se verifica pois as ações são mais simples e curtas, sendo por isso muito menos dispendiosas em tempo.

Capítulo 3

Conceção

3.1 Arquitetura do Sistema

Neste secção iremos mostrar e explicar a arquitetura da nossa aplicação, bem como fazer referência a algumas decisões mais importantes. Para isso iremos recorrer a modelos como o *Platform Independent Model (PIM)*, o *Platform Specific Model (PSM)*, e ainda o modelo conceptual da base de dados.

3.2 PIM

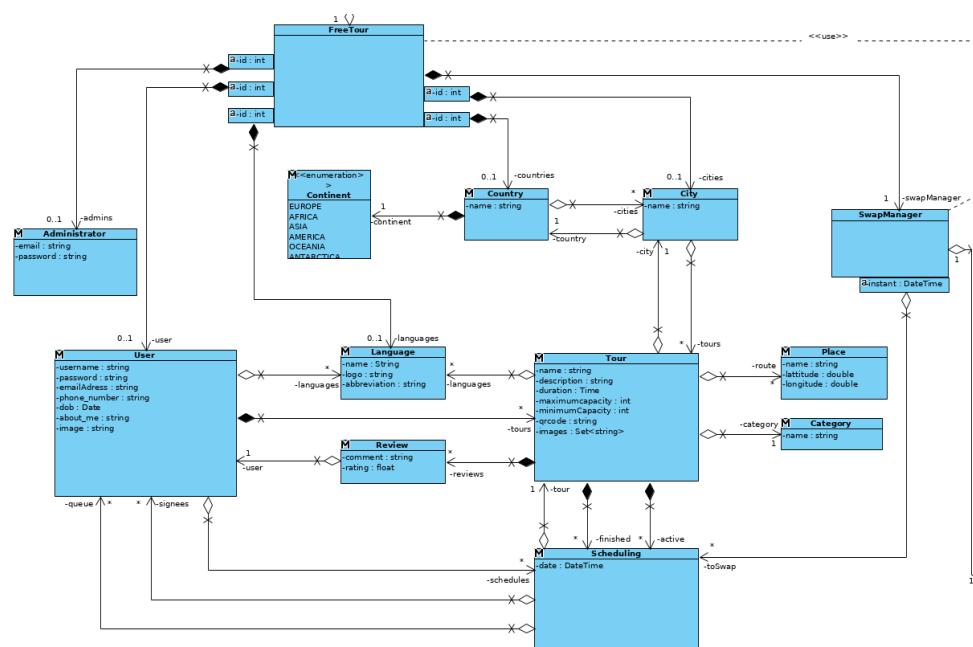


Figura 3.1: PIM parte 1

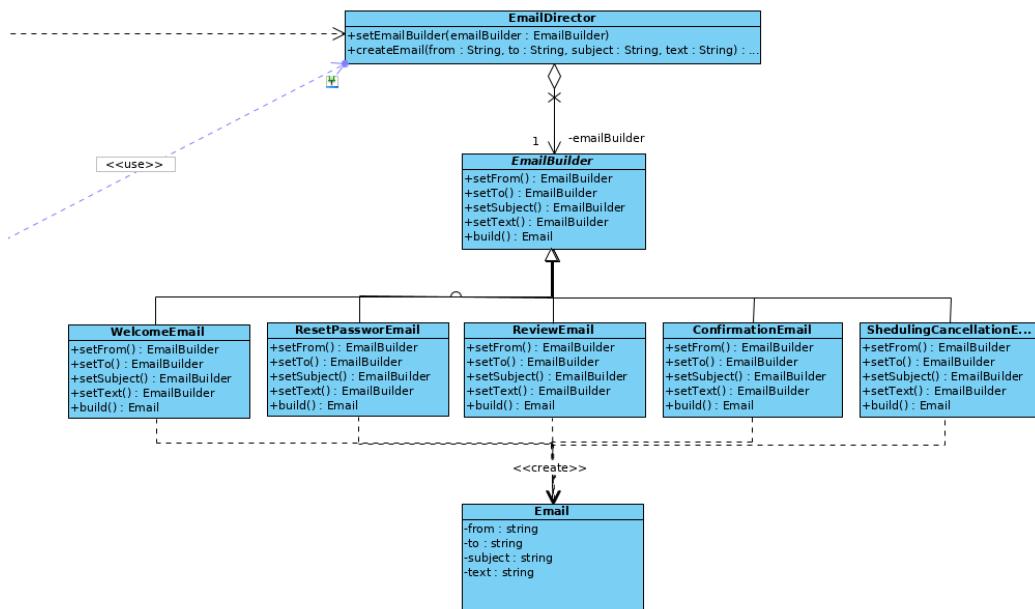


Figura 3.2: PIM parte 2

3.2. PIM

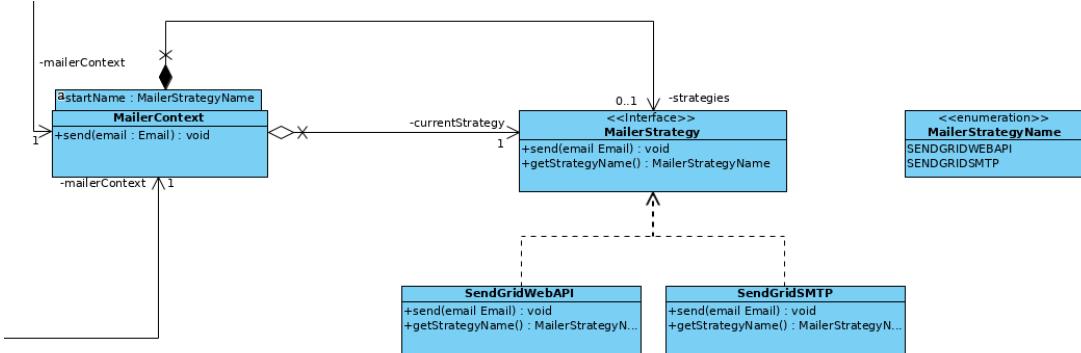


Figura 3.3: PIM parte 3

Este modelo representa então a arquitetura genérica do nosso sistema. Como primeira parte temos a base, constituída pelas diferentes entidades do sistema. Para além disso possuímos o facade FreeTour que irá possuir todos os métodos necessários para exportar as funcionalidades da aplicação para o exterior.

Existe ainda uma classe SwapManager diferente de todas as outras cuja função é criar threads de maneira a gerir a lógica de controlo de um schedule. Isto é, quando um schedule é criado é criada também uma thread pelo SwapManager que adormece e volta a acordar quando o schedule acaba, encerrando o mesmo e enviando mails aos participantes para fazerem reviews.

Neste modelo usamos duas *Patterns* abordadas nas aulas de Arquiteturas aplicacionais, mais concretamente o **Builder** e o **Strategy**. O **Builder** foi utilizado para a criação de emails como mostra a figura 3.2.

Justifica-se o uso desta *pattern* para separar a construção dum objeto complexo (Email), da sua representação para que o mesmo processo de construção crie diferentes representações do objeto.

O uso da *pattern Strategy*, representada na figura 3.3, justifica-se pois apenas necessitamos de realizar uma função que é o envio de um email, mas como recorremos a APIs externas, ficamos dependentes do funcionamento destas. Caso uma falhe, podemos integrar outra facilmente, desde que esta tenha a funcionalidade de enviar emails. Desta forma, abstrai-se o algoritmo de envio.

Este modelo é uma abstração da realidade, e estará sujeito a algumas alterações consoante a tecnologias a utilizar.

3.3. *PSM*

3.3 PSM

Como a plataforma onde vai ser desenvolvido o backend e a maioria da aplicação, é o *Spring*, decidimos especificar o nosso *PIM* para um *PSM* tendo em conta as regras e anotações dessa framework.

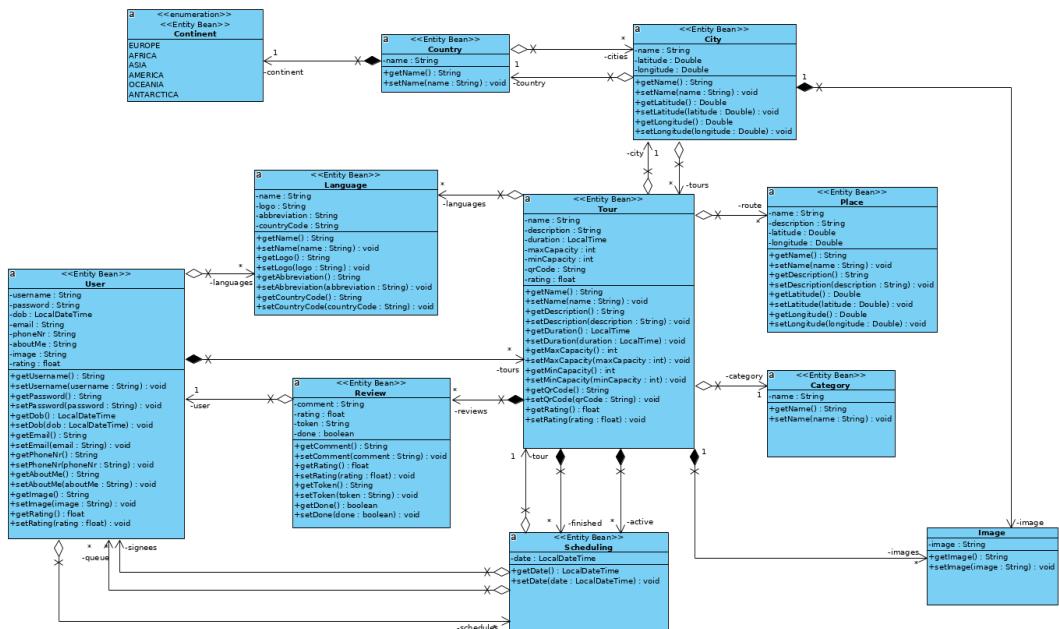


Figura 3.4: PSM - Entidades

Todas as entidades são então anotadas com a tag *Entity* isto faz com que o spring persista estas classes na base de dados, usando o hibernate.

De seguida são criados os controllers, com a anotação *RestController*, que faz com que o spring abra o controlador para o exterior, preparando-o para receber e responder a pedidos *REST*. É nestes controllers que estão definidos os endpoints da aplicacão e a sua lógica de negócio.

Por último possuímos os serviços, que são injetados nos controladores. Estes são responsáveis por gerir as entidades do sistema, bem como executar outras operações de lógica de negócios, podendo ser comparados aos *Session Beans* do Java EE.

3.3. PSM

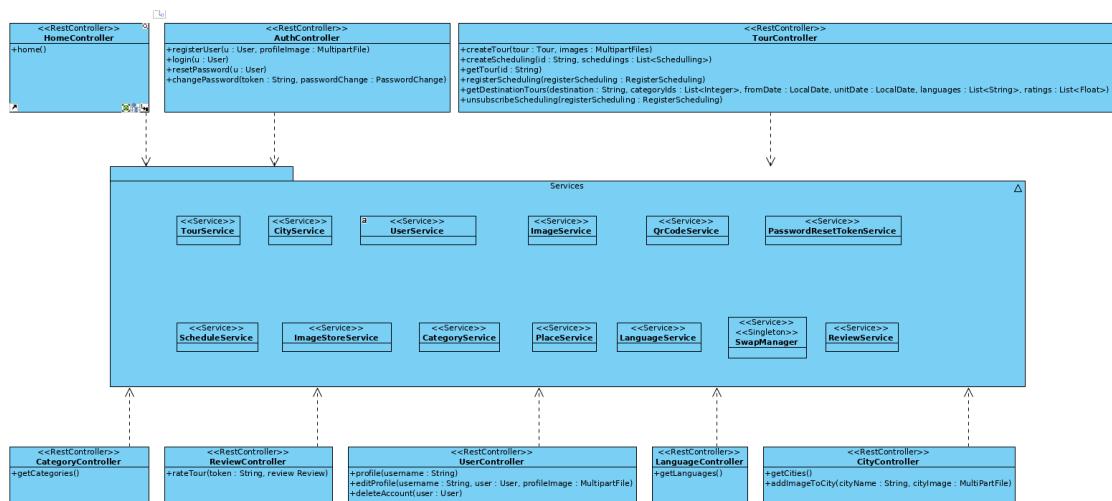


Figura 3.5: PSM - Controladores e Serviços

3.4. BASE DE DADOS

3.4 Base de Dados

Dado os modelos acima propostos, um possível modelo para a Base de Dados que albergará o sistema é o que se encontra na seguinte figura.

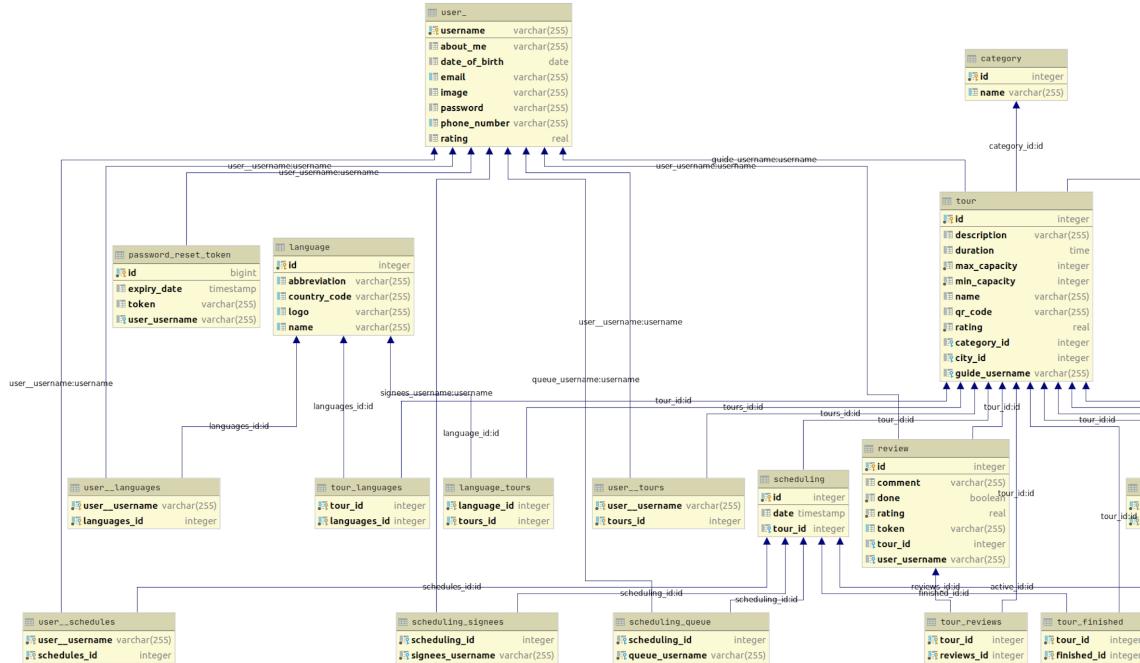


Figura 3.6: Modelo lógico da Base de Dados - parte 1

3.4. BASE DE DADOS

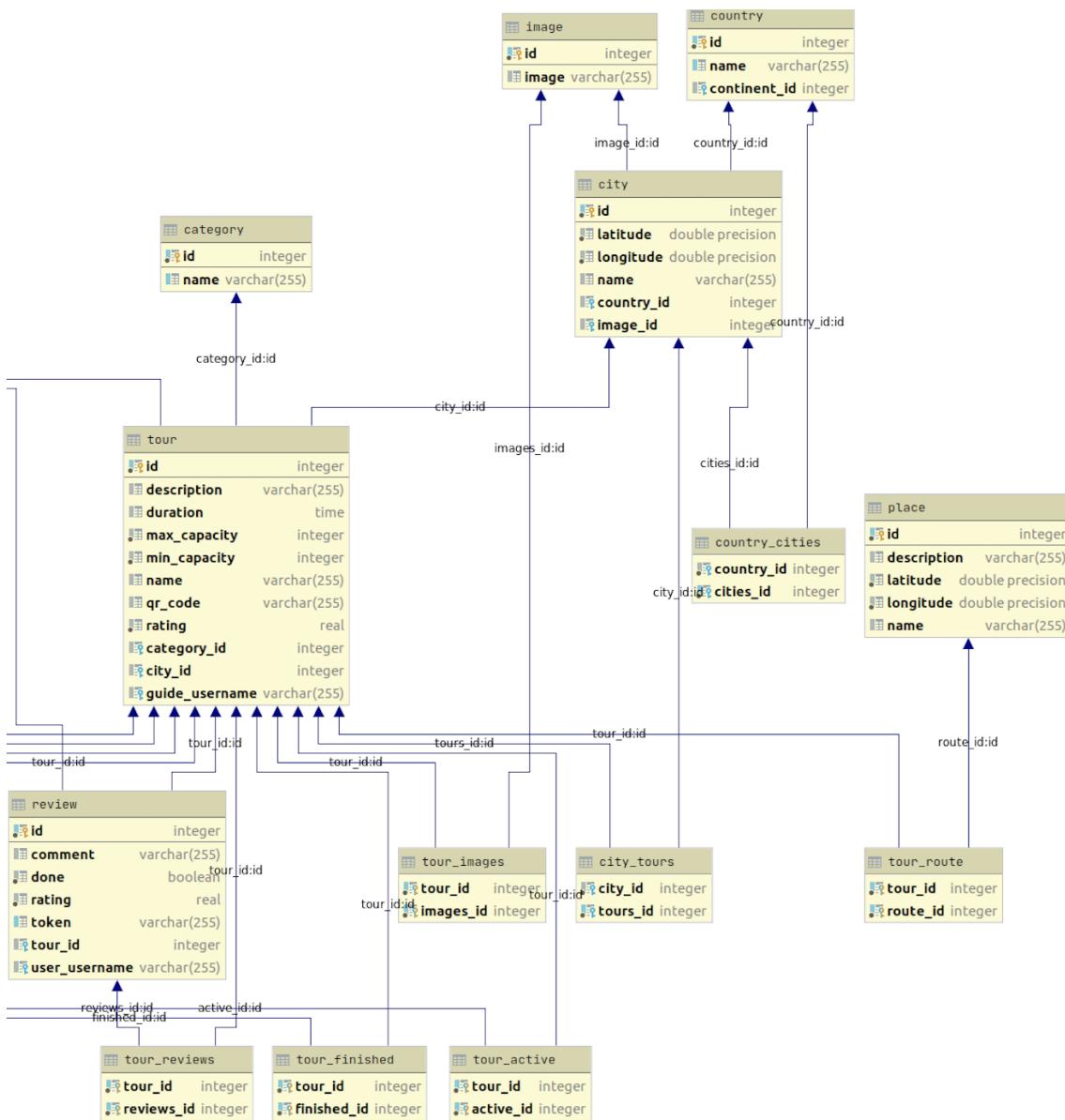


Figura 3.7: Modelo lógico da Base de Dados - parte 2

A figura 3.6 e a figura 3.7 fazem parte do mesmo modelo apenas estão separadas para ficarem minimamente percepíveis.

Devido às múltiplas relações entre classes e objetos, e ao método de criação de tabelas Tabela-Classe, caímos num modelo bastante extenso que suporta todas as funcionalidades que são requeridas ao nosso sistema.

Mais tarde a Base de Dados será gerida e mantida por um mecanismo *ORM* (Hibernate) pelo que a gestão e manutenção da Base de Dados em si sairá do controlo

dos programadores.

3.5 Prototipagem da Interface

Para a prototipagem da interface, foi utilizada a plataforma Draw.io, um site que permite a conceção de esquemas, interfaces e mais. Dado o intuito da nossa plataforma, e os requisitos já mencionados, é crucial que a interface seja fácil de utilizar e intuitiva, visto que sendo pública, poderá ser utilizada por pessoas que não possuam grandes capacidades técnicas. A título de exemplo, iremos mostrar os protótipos referentes às tarefas mais relevantes, como por exemplo criação de uma Tour e inscrição numa Tour.

3.5. PROTOTIPAGEM DA INTERFACE

3.5.1 Páginas mais Relevantes

Página de Criação de uma Tour

The wireframe shows the 'Create a Tour' page. At the top, there's a navigation bar with 'FreeTours', a search bar ('Destination...'), and links for 'Create a Tour', 'Username', and 'Logout'. The main content area has a title 'Create a Tour'. It includes fields for 'Tour Name' (containing 'Amazing Berlin City Tour'), 'Location' (containing 'Berlin, Germany'), 'Duration' (containing '03 hours 30 minutes'), 'Capacity' (containing '0 people'), 'Language' (containing 'English, French.'), and a 'Description' box with placeholder text about Mexico. Below that is a 'Route' section with 'Map' and 'Markers List' boxes. Under 'Images', there's a paperclip icon and a box for 'Annexed Images'. A 'Submit tour' button is at the bottom.

Figura 3.8: Página de Criação de uma Tour.

No que toca à página de criação de uma Tour, esta é bastante simples sendo que a maior parte dos campos são caixas de texto e menus de dropdown. A parte mais complexa desta página, encontra-se na secção do mapa, onde são inseridos os pontos que serão visitados. Para inserir um ponto no mapa, o utilizador efetua um

3.5. PROTOTIPAGEM DA INTERFACE

duplo clique no local que pretende adicionar. Ao fazê-lo, é aberto um componente lateral, no qual existem caixas de texto, para inserir o nome do local e uma descrição do mesmo, juntamente com um botão de modo a confirmar a adição do ponto ao trajeto. Depois de inserido, o utilizador pode voltar a adicionar mais pontos ao mapa, através deste processo e a cada adição, a lista de pontos ao lado, é atualizada para refletir o trajeto planeado.

Cremos que esta abordagem é adequada, visto que de maneira geral, os utilizadores estão habituados ao conceito do duplo clique para selecionar algo, daí ter sido implementada. Como tal, achamos que as ações a tomar nesta página são claras, mas avaliarmo-las-emos mais à frente.

3.5. PROTOTIPOGEM DA INTERFACE

Página de uma Tour

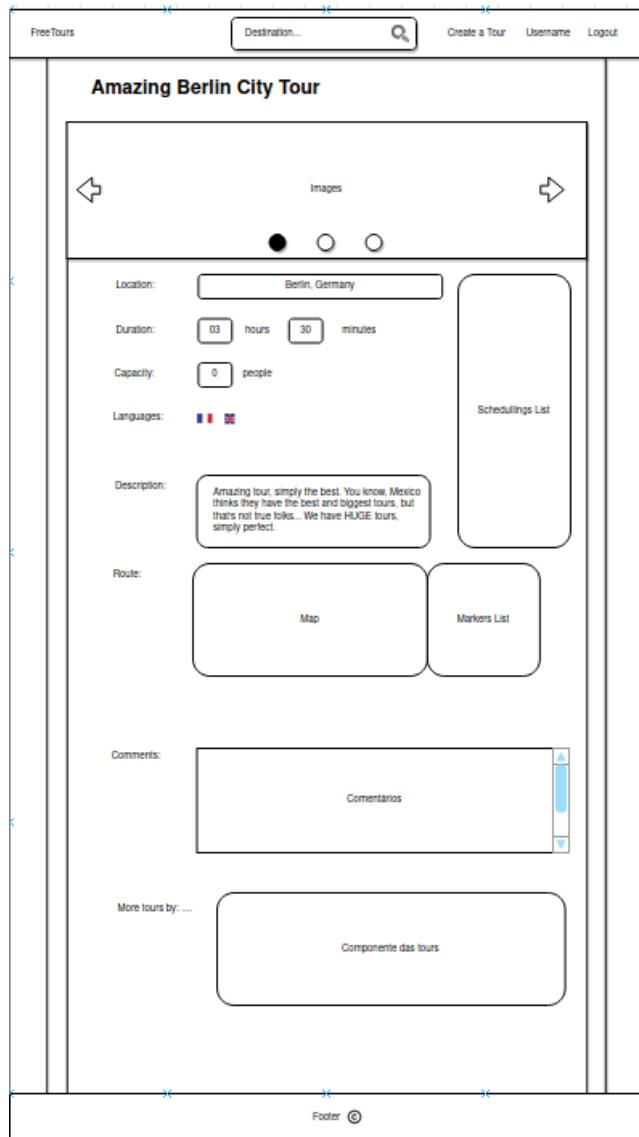


Figura 3.9: Página de uma Tour.

Na página da Tour, encontramos todas as informações referentes à mesma, desde fotos, a cidade onde é feita, as datas e horas, entre outras.

Para o utilizador se inscrever na Tour, este deve de selecionar o horário que pretende na lista de Schedullings na parte direita da página, que abrirá o modal seguinte, que permite a sua inscrição.

3.5. PROTOTIPAGEM DA INTERFACE

Modal de Inscrição de uma Tour

Quanto ao modal da inscrição numa Tour, este é também simples, visto que tem apenas uma caixa de texto que permite ao utilizador adicionar pessoas que irão à visita consigo, e um botão para confirmar a inserção.

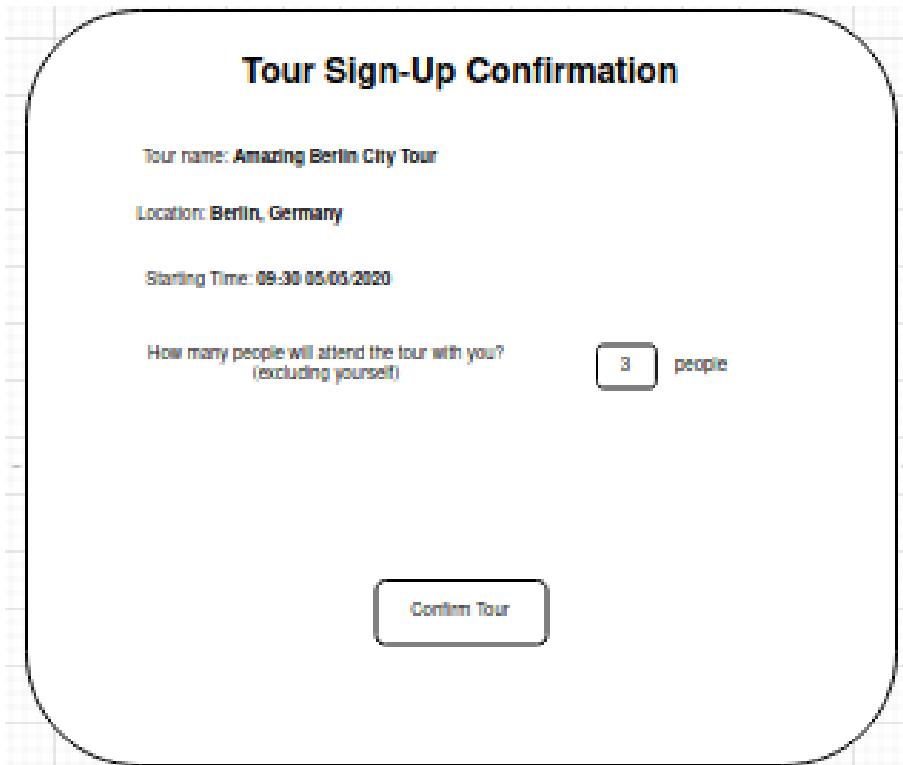


Figura 3.10: Modal de Inscrição de uma Tour.

3.5.2 Técnicas de Avaliação

Como técnica de avaliação utilizada para testar a nossa interface, decidimos efetuar um Cognitive Walkthrough de modo a quantificar a eficiência da interface a permitir a inscrição de um utilizador numa Tour, tendo o seguinte resultado.

3.5. PROTOTIPAGEM DA INTERFACE

System:	Fácil de Usar	Inscrever em Tour	
Identifico:	OK (S/N)	OK (S/N)	
Preciso:	Outras	Outras	
Procurar Cláusula Detalhe	1	1	Risco (1-3)
Escolher Tour	2	2	Problema Sugerido
Selecionar um dos Schedulings	3	1	
Indicar quantas pessoas irão à Tour	4	1	No momento não é possível no sistema o acompanhamento da tour, no momento tem informações sobre a tour e um botão que leva a pagina da tour na qual é possível efetuar a inscrição.
Inscrever na Tour	5	1	O nome só é inserido quando o usuário clica no botão Inscrever na Tour.

Figura 3.11: Cognitive Walkthrough - Inscrever em Tour.

Em geral, a nossa interface saiu-se bem, sendo que poderiam ter sido adicionados mais detalhes, a nível da prototipagem, de modo a dar uma experiência

3.5. PROTOTIPAGEM DA INTERFACE

mais detalhada ao utilizador. Desta maneira, teria sido ganho feedback adicional, relativamente à qualidade da interface.

O primeiro passo ocorre na homepage 3.12, sendo que existe duas maneiras através da qual o Utilizador consegue chegar à página que contém Tours de uma cidade. Primeiro, através da barra de pesquisa no cabeçalho da página e também através da secção abaixo da "Highlighted Tours" através da qual é possível filtrar as Tours pela cidade em que ocorrem, a data em que ocorre e a categoria da Tour a realizar. Após a seleção da cidade a procurar, é apresentada a página de Pesquisa de Tours 3.13 na qual são listadas as Tours que ocorrem nessa cidade, num certo período temporal. Nessa página, o utilizador poderá selecionar a que mais lhe interessar, avançando para a página da Tour, na qual poderá efetuar a sua inscrição.

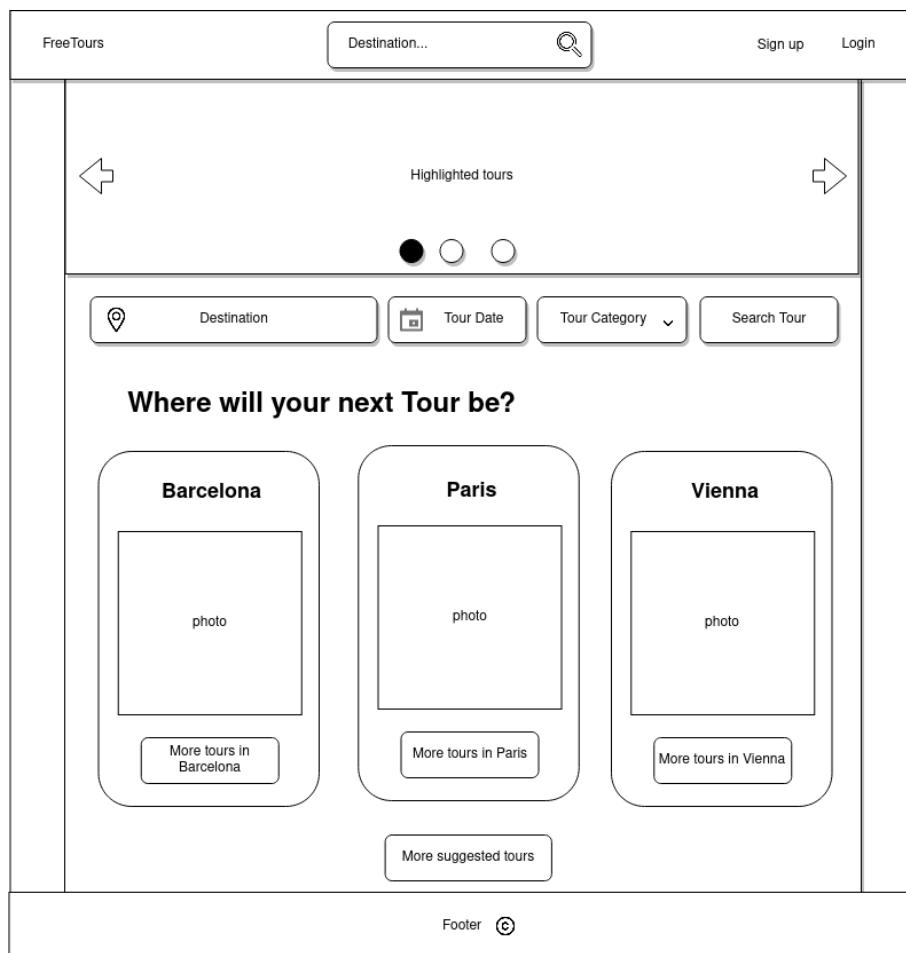


Figura 3.12: Homepage.

3.5. PROTOTIPOGEM DA INTERFACE

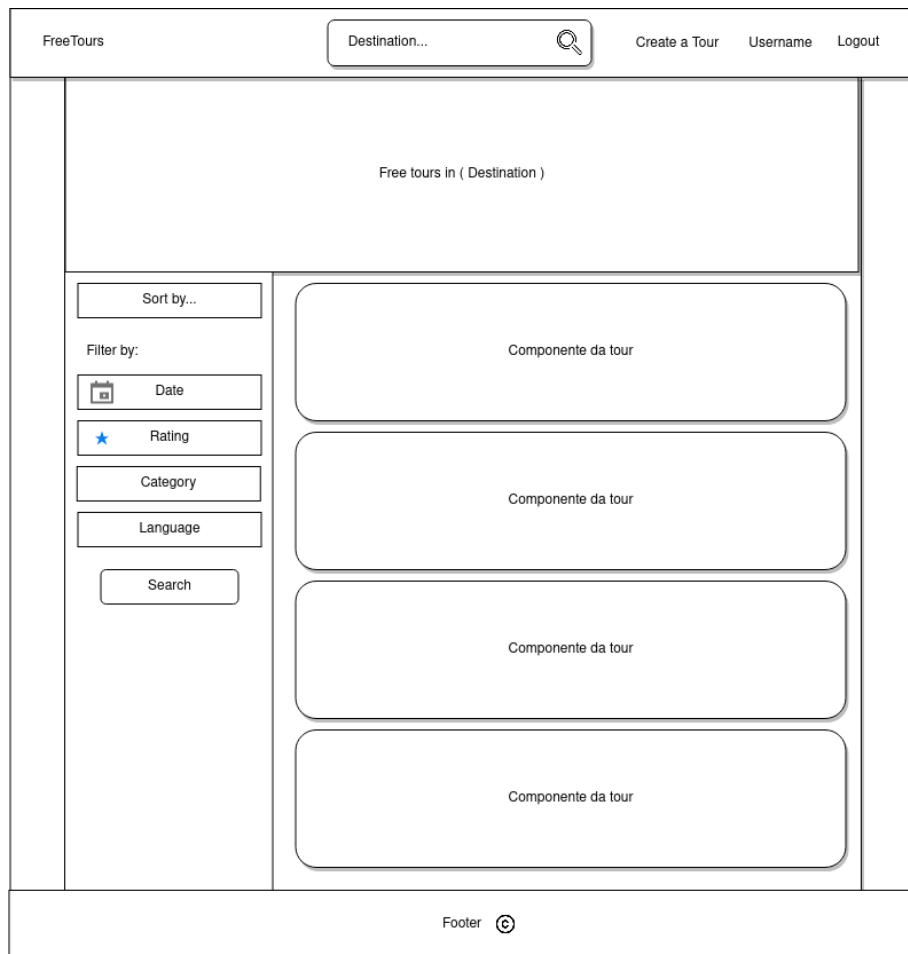


Figura 3.13: Página de Pesquisa de Tours.

Capítulo 4

Desenvolvimento

4.1 Tecnologias a Utilizar

4.1.1 Spring

Escolhemos como framework para suportar o backend o *Spring*, esta tecnologia está bem situada no mercado e tem tudo o que é preciso para construir uma *Rest Api* em termos de funcionalidades e segurança.

4.1.2 Hibernate

Com a funcionalidade de persistir as nossas entidades, escolhemos a framework hibernate. Esta facilita o acesso aos dados guardados, utilizando tecnicas como o *Hibernate Query Language (HQL)*.

4.1.3 Vue

Para tecnologia de frontend optamos pelo *Vue*, uma vez que foi lecionada numa das cadeiras. E para alem disto apresenta vários pontos fortes, como a construção orientada ao componente, e o facto de ser *single page oriented* o que permite criar com facilidade paginas dinâmicas e responsivas.

4.1.4 Leaflet

O *Leaflet* é uma biblioteca para *Vue* capaz de gerar um mapa, onde é possível especificar um conjunto de pontos formando uma rota. Para além disso possui outras funcionalidades não utilizadas neste projeto.

4.1.5 JWT

Para autenticação e sessão, decidimos optar pela tecnologia *JSON Web Token (JWT)*, para isso é gerado um token jwt criptográficamente seguro a quando o login de qualquer utilizador, esse token é armazenado pelo frontend e é depois passado no campo *Authorization* do header de cada pedido. O backend ao receber este token consegue validar a sessão e saber quem é o utilizador que está a efetuar o pedido.

4.1.6 SendGrid

Usamos a API externa do *SendGrid* para o envio de emails, evitando que o nosso servidor aplicacional ficasse carregado com mais uma tarefa. Esta API oferece duas formas de enviar emails que são ambas utilizadas pelo nosso *software*.

4.1.7 QR code API

Usamos esta API para codificar o *link* duma Visita num *QR Code*. Desta forma, acedemos diretamente do *Frontend* à API recebendo a imagem do *QR Code* da visita em questão. Os *QRs* servirão para publicitar a visita que está a ser acedida pelo *frontend*.

4.1.8 DataSet de Cidades

Para abastecer a nossa aplicação com cidades, países e localizações, recorremos a um *Dataset* que contém todas as cidades do mundo que tenham uma população superior a 5000. Como existiam milhares de registos tratamos apenas de inserir as cidades que tivessem mais de 40000 habitantes apenas, inserindo apenas 10000 cidades na Base de Dados.

4.2 Decisões Arquiteturais

Existiram algumas decisões que tivemos de tomar por razões de desempenho e integridade da aplicação.

4.2.1 Imagens

A existência de imagens constitui um problema, pois geralmente as imagens são ficheiros algo pesados e consequentemente o seu *download* ou *upload* consomem alguma largura de banda. Ora uma decisão importante tomada foi não armazenar as imagens no *Backend* pois caso contrário este ficaria sobrecarregado a servir imagens, e o espaço da Base de Dados seria rapidamente consumido.

Uma solução para este problema poderia passar por ser o armazenamento das imagens no servidor de *frontend* pelo que estes são otimizados para servir recursos estáticos. Mais uma vez esta solução não se mostrou adequada pois no caso de existir mais do que um *Web Server* em funcionamento teria que haver algum mecanismo de replicação de ficheiros, e teríamos que abrir a possibilidade dos *Web Servers* poderem receber pedidos.

A solução que encontrámos por fim foi algo que híbrida, pois as imagens passam uma única vez no *Backend* e são partilhadas por um sistema de ficheiros distribuído (NFS ou Volumes Docker) para os servidores de *Frontend*. Desta forma, quem partilha imagens é o *Backend* e quem lhes acede é o *Frontend*.

4.2.2 Cidades

Quanto às cidades, como temos um grande número delas (10000), e como queremos uma experiência agradável do lado do utilizador, quando este tentar procurar por visitas em cidades, estas têm que ser dispostas para ele poder escolher. O problema surge no facto de existirem vários sítios onde se podem inserir cidades e por isso, teríamos que pedir todas essas vezes ao *Backend* pelas cidades o que pode ser algo pesado.

De forma a mitigar este problema, quando abrimos uma página de *Frontend*, as cidades são todas descarregadas duma vez ficando persistidas e prontas para uso, nunca sendo preciso pedir novamente as cidades ao *Backend* durante o período de utilização da aplicação.

4.3. ENDPOINTS

4.3 Endpoints

De forma a estabelecer conexão entre o *Backend* e o *Frontend*, foram criados endpoints para transferência de informação útil entre ambos os componentes.

AuthController

Neste *controller* estão presentes os *endpoints* relacionados com a autenticação dos utilizadores.

/sign_up

POST Multipart:

```
user {username*; password*; email*; phoneNumber; dateOfBirth;  
      aboutMe; languages;}  
profileImage {image...}
```

/sign_in

```
POST {email; password}  
<- JWT
```

/reset_password

```
POST {email}
```

/change_password

```
?token=:token  
POST {password, password_confirmation}
```

4.3. ENDPOINTS

CategoryController

Este *controller* possibilita o acesso às categorias existentes no sistema.

```
/categories
```

```
GET <- {categories...}
```

CityController

Este *controller* possibilita o acesso às cidades existentes no sistema, assim como a inserção de imagens relativas a cidades.

```
/cities
```

```
GET <- {cities...}
```

```
/city
```

```
/:cityName
```

```
POST {cityImages...}
```

HomeController

Este *controller* possui um *endpoint* `home` que retorna os dados necessários para a página inicial do *website*.

```
/home
```

```
GET <- {mostPopularCities; nextTours; suggestedTours;}
```

4.3. ENDPOINTS

LanguageController

Este *controller* possibilita o acesso às linguagens existentes no sistema.

/languages

GET <- {languages...}

ReviewController

Neste *controller* estão presentes os *endpoints* relacionados com as *reviews* dos utilizadores.

/review

?token=:token

POST {review;}

4.3. ENDPOINTS

TourController

Este *controller* possui os *endpoints* relacionados com as *tours* e os agendamentos, como a procura, inscrições e cancelamentos.

/createTour

POST Multipart:

```
tour {name; description; duration; maxCapacity;  
      minCapacity; location; city; route; category;  
      languages;}  
images {image...}  
<- tourId
```

/createScheduling

/:idTour

POST {dates...}

/tour

/:idTour

GET <- {tour; moreToursBy;}

/schedule_signin

POST {schedulingId; nrPeople;}

/schedule

/unsubscribe

POST {schedulingId;}

/search

/:destination

```
GET ?rating= &beginDate= &endDate= &category= &languages=  
<- {tours...}
```

4.3. ENDPOINTS

UserController

Este *controller* possui os *endpoints* para acesso aos dados relacionados com o utilizador, mais propriamente o seu perfil e a remoção da conta.

```
/profile
  /:username
    GET <- {profileInfo;}
    POST Multipart:
      user {username*; password*; email*; phoneNumber; dateOfBirth;
            aboutMe; languages;}
      profileImage {image;}

/profile
  /delete_account
    POST {username*; password*; email*; phoneNumber;
          dateOfBirth; aboutMe; languages;}
```

Capítulo 5

Deployment e Testes

5.1 Deployment

Para colocar o software construído disponível para utilização foi necessário que existisse o *deployment* dos componentes que constituem o mesmo. Para este efeito necessitamos dum **Web Server** para servir o *frontend*, um **Servidor Aplicacional** para servir o *Backend* e ainda duma **Base de Dados**.

Construímos dois métodos para *Deployment*, pois necessitamos de testar a arquitetura de diferentes formas. Desta forma, temos um *Deployment* local, onde todas as componentes correm localmente numa máquina, e temos ainda um *Deployment* na *Cloud* onde recorremos ao uso de diferentes máquinas virtuais (distintas) para as componentes correrem.

Em ambos os casos recorremos ao uso do **Docker**, pois as componentes que constituem o nosso sistema são devidamente containerizados para terem um ambiente de execução isolado e quase totalmente independente do *hardware* em que opera, poupando alguns problemas em configurações.

Como mostra a figura 5.1, construímos 3 *imagens* para os componentes do sistema que mais tarde serão usados para criar os *containers* necessários. Para a **Base de Dados** usamos uma imagem do *PostgreSQL*. Para o **Backend** construímos uma imagem que contém a aplicação correspondente ao *Backend* que é composto por *Spring* e *Hibernate*, e que terão um ambiente de execução oferecido por um servidor *Tomcat*. Para o **Frontend** construímos uma imagem que contém a aplicação correspondente ao *Frontend* que é composto por *Vue*, e terá um ambiente de execução oferecido por um servidor *NGINX*.

5.1. DEPLOYMENT

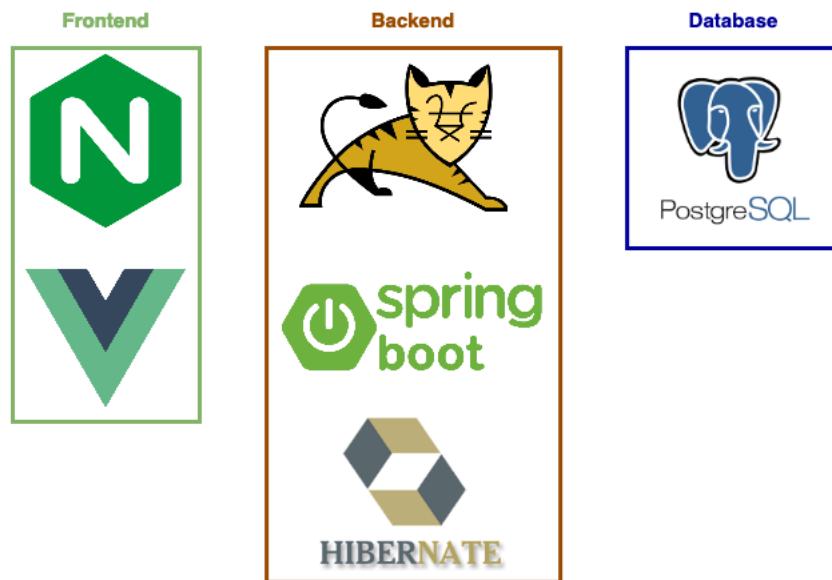


Figura 5.1: Imagens Docker dos componentes

5.1.1 Local

A primeira solução para *Deployment* que construímos faz uso do **Docker-Compose** para construir as imagens necessárias e colocá-las em execução em *containers* tal como mostra a figura 5.2.

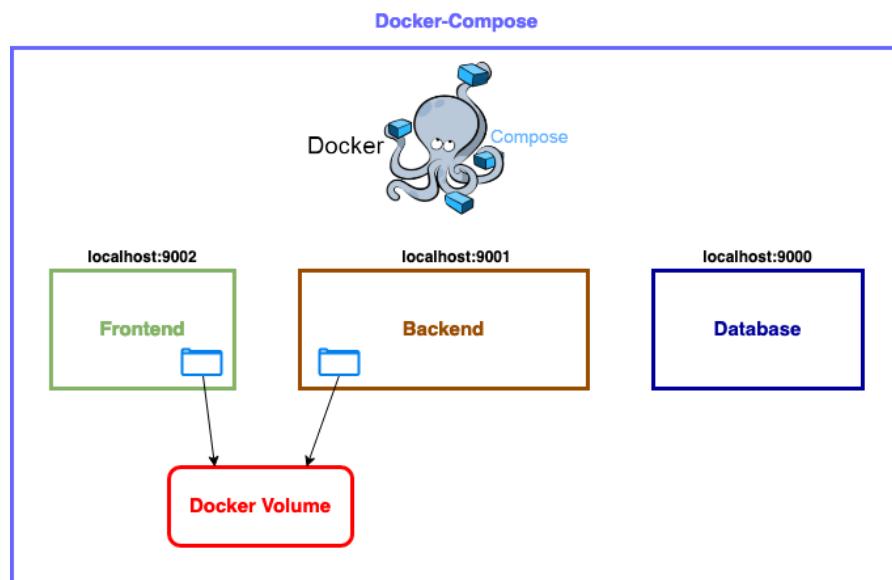


Figura 5.2: Infraestrutura Local usando Docker-Compose

5.1. DEPLOYMENT

Esta solução leva-nos à inserção doutro componente no sistema que é um **Volume do Docker** que é necessário para assegurar a troca de imagens entre *Frontend* e *Backend*. Esta componente funciona como uma pasta partilhada entre os dois *containers*, pois o **Backend** recebe as imagens provenientes de pedidos, guarda-las neste volume, e mais tarde o *Frontend* quando necessitar de aceder a estas imagens, simplesmente as lê.

Comandos

A partir da *root* da pasta do projeto, os comandos a executar para dar *deploy* desta solução são os seguintes:

1. Construir imagens e colocá-las em execução em *containers*:

```
docker-compose up --build
```

A partir deste ponto os componentes do sistema ficam disponíveis localmente nas portas que a figura 5.2 refere (9000, 9001, 9002). Fica a faltar executar os *scripts* de criação abordados numa secção posterior.

5.1. DEPLOYMENT

5.1.2 Máquinas separadas - Cloud

A segunda solução para *Deployment* que construímos faz uso do **Google Cloud Platform** que facilita a criação e gestão de máquinas virtuais, podendo assim ser possível criar máquinas e gerir as regras de rede e *firewalls* mais facilmente (comparado ao deployment nas máquinas de cada membro do grupo). Com esta solução não descartamos a utilização de **Docker** para construir as imagens necessárias e colocá-las em execução em *containers*, só que desta vez, apenas corre um container em cada máquina, como mostra a figura 5.3.

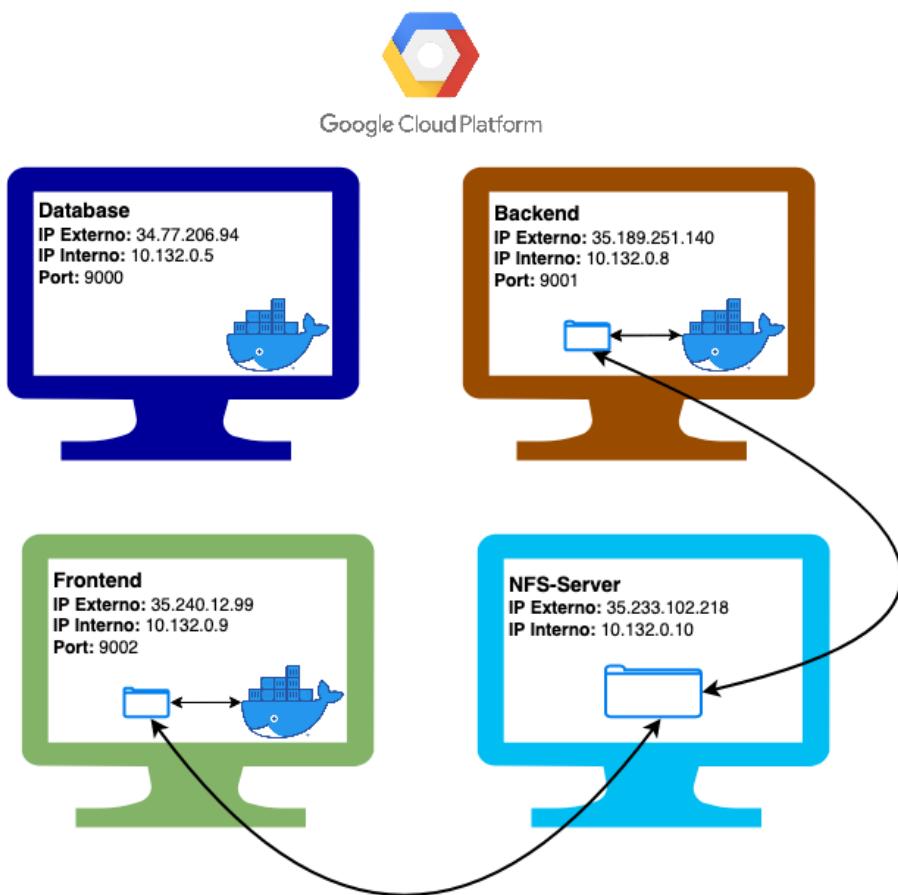


Figura 5.3: Infraestrutura na Cloud

Um dos principais problemas com esta solução foi a partilha de imagens que não podia ser gerida de forma convencional pelo *Docker* como na solução anterior (Volumes). Isto deve-se ao facto dos *containers* não estarem em execução na mesma máquina.

Para contornar este problema, adicionamos um componente à infraestrutura,

5.1. DEPLOYMENT

que é um **NFS Server**, que oferece um sistema de ficheiros distribuído que possibilita a partilha de ficheiros e pastas, o que por si só nos é extremamente conveniente para a partilha de imagens.

Com esta solução continuamos a usar Volumes de *Docker* mas não com a mesma utilidade da solução anterior. Desta vez, os volumes usados são construídos a partir da pasta partilhada pelo *NFS*. Assim, o *container* escreve(lê) para(do) seu volume, que por si só já é a pasta distribuída pelo *NFS*.

Comandos

1. Configuração do *NFS*:

A configuração deste componente é algo extensa e foi feita seguindo exatamente o tutorial do seguinte link:

<https://www.digitalocean.com/community/tutorials/how-to-set-up-a-n-nfs-mount-on-ubuntu-20-04-pt>

Seguindo o tutorial, a máquina referida como **host**, é a nossa máquina *NFS-Server*, enquanto que os **client** são as nossas máquinas do *Frontend* e do *Backend*.

O caminho para a pasta partilhada nas máquinas *client* é `/home/joao/shared_images`, daí algumas referências nos seguintes comandos a esta pasta.

2. Deploy da Base de Dados:

Na máquina da Base de Dados basta correr o comando:

```
docker run -d --restart always --name postgres
-p 9000:5432 -e POSTGRES_USER=postgres -e
POSTGRES_PASSWORD=postgres -e
POSTGRES_DB=freetour -e
PGDATA=/var/lib/postgresql/data/pgdata
postgres:11.5
```

Desta forma, a Base de Dados fica a executar num *container* que é acessível na porta **9000**, e um novo esquema de Base de Dados é criado (*freetour*).

3. Deploy do Backend:

Na máquina do *Backend*, é preciso criar o *WAR File* antes de criar uma imagem para o *container*, mas como o nosso projeto contém alguns testes precisamos dum a base de dados local só para que estes testes executem. Criar a Base de Dados local pode ser conseguido através de:

5.1. DEPLOYMENT

```
docker run -d --name postgres -p 5432:5432 -e  
POSTGRES_USER=postgres -e  
POSTGRES_PASSWORD=postgres -e  
POSTGRES_DB=freetour -e  
PGDATA=/var/lib/postgresql/data/pgdata  
postgres:11.5
```

Após este *container* estar operacional é necessário criar o ficheiro *WAR*. Então, indo para a *root* da pasta do trabalho:

```
cd backend/  
  
.mvnw install  
  
docker stop postgres  
  
docker rm postgres  
  
docker build -t backend .  
  
docker run -d --restart always --name backend -p  
9001:8080 -e  
SPRING_DATASOURCE_URL=jdbc:postgresql:  
//10.132.0.5:9000/freetour -e  
SPRING_DATASOURCE_USERNAME=postgres -e  
SPRING_DATASOURCE_PASSWORD=postgres -e  
SERVER_PORT=8080 -e  
FRONTEND_URL=http://35.240.12.99:9002 -e  
APP_SHARED_IMAGES=/images/ -v  
/home/joao/shared_images:/images backend
```

Desta forma, o *Backend* fica a executar num *container* que é acessível na porta **9001**, conecta-se à Base de Dados previamente construída noutra máquina, define o *URL* do *frontend*, e a localização da pasta distribuída, associando-lhe também um volume.

4. Deploy do Frontend:

Na máquina do *Frontend* é preciso criar a imagem para o *container* executar da seguinte forma, partindo da *root* da pasta do projeto:

```
cd frontend/  
  
docker build --build-arg  
  VUE_APP_API_URL=http://35.189.251.140:9001  
  --build-arg  
  VUE_APP_FRONTEND_URL=http://35.240.12.99:9002  
  -t frontend .  
  
docker run -d --restart always --name frontend  
  -p 9002:80 -v /home/joao/shared_images:/images  
  frontend
```

Desta forma, o *Frontend* tem definido o *URL* do *Backend*, assim como o seu (para a procura de imagens), cria o volume associado à pasta distribuída e fica disponível na porta **9002**.

A partir deste ponto todas as máquinas necessárias estão disponíveis, ficando a faltar apenas executar os *scripts* de criação abordados na secção posterior.

5.1.3 Scripts de Criação

Após realizar algum tipo de *Deployment* definido acima, faltam criar e inserir alguns dados na infraestrutura.

No nosso caso, existem dois momentos onde a inserção de dados é feita. A primeira é feita aquando da inicialização do *Backend*, e é feita automaticamente pelo *Spring*. Neste momento, inserem-se dados relativos a cidades, países e categorias de *Tours*. Estes dados residem no ficheiro *backend/src/main/resources/data.sql*, carregado na inicialização da aplicação. Esta inserção é algo demorada ($\approx 40/50$ segundos) pelo que deve ser esperado algum tempo (≈ 1 minuto) até carregar os próximos dados.

O segundo momento de inserção de dados ocorre após o *Backend* estar operacional e visa inserir utilizadores, visitas e agendamentos de visitas. Esta inserção é feita recorrendo ao ficheiro *backend/src/requests/CreationRequests.http*. Não existe forma de executar estes pedidos por linha de comandos pois este tipo de ficheiros são uma ferramenta inserida pelo **IntelliJ** que contêm a sua própria notação.

Usando então o *IntelliJ* deve-se abrir o projeto relativo ao *backend*, e de seguida abrir este ficheiro, que é interpretado pelo editor de texto de forma diferente e que permite executá-lo como indica a seguinte imagem.

5.1. DEPLOYMENT

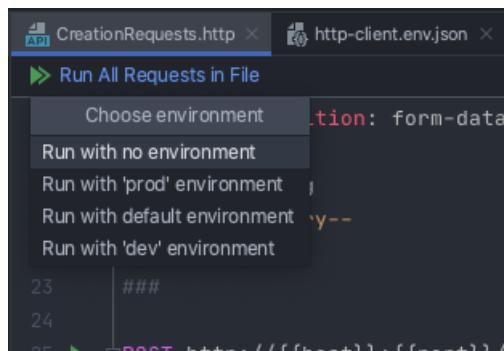


Figura 5.4: Script de Criação

Existe um ficheiro na mesma pasta que define o endereço ao qual se irá fazer pedidos, é o ficheiro *http-client.env.json*. Dependendo da máquina que queremos testar podemos escolher ou o ambiente **dev** ou **prod**, como mostra a figura 5.4.

O ficheiro de ambiente é definido da seguinte forma:

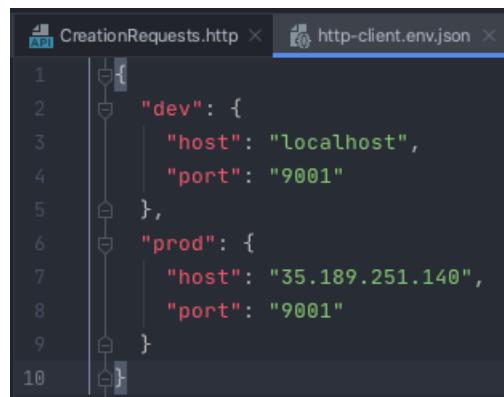


Figura 5.5: Ambientes de testes

Após a execução deste ficheiro o sistema fica aprovado com alguns dados.

5.2 Testes

Para testar a nossa aplicação recorremos à ferramenta *Jmeter*, que permite executar vários tipos de testes a aplicações web possuindo varias ferramentas de analise.

Os testes foram feitos recorrendo à aplicação *deployed* na *cloud*, onde frontend, backend, e base de dados estão divididos por 3 máquinas cada uma com 2 cpus, 7.5Gb de memoria, 10GB ssd, e como sistema operativo o ubuntu 20.04 LTS.

Optamos então por testar aqueles que na nossa opinião poderão ser os endpoints mais acedidos, o endpoint da home, do search, e do login.

Para cada um destes endpoints era elaborado um pedido a ser executado por determinado número de utilizadores ao mesmo tempo, por cada teste este processo era repetido 5 vezes para dar uma maior carga ao servidor. Assim quando o número de utilizadores é 20 isto quer dizer que vão ser mandados 20 pedidos ao mesmo tempo num determinado endpoint um total de 5 vezes. Somado dá 100 pedidos nesse teste.

5.2.1 Home

No endpoint home, foi necessário fazer duas requests diferentes para os testes. Uma vez que o comportamento do endpoint varia caso consoante a presença ou não do token de autenticação do utilizador no pedido.

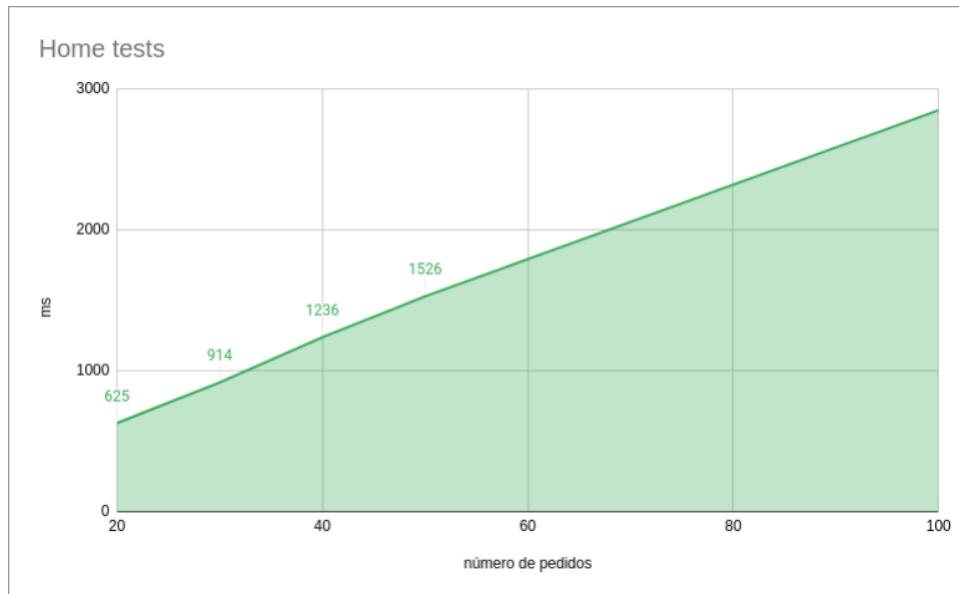


Figura 5.6: Testes no endpoint home

5.2. TESTES

Este primeiro gráfico diz respeito ao teste no endpoint home sem usar o token de autenticação.

Como podemos ver o desempenho é razoável até por volta dos 60 pedidos em simultâneo, onde vemos que o tempo de resposta médio anda abaixo dos 2000 milissegundos. Relembrando que os 60 pedidos são enviados 5 vezes o que perfaz um total de 300 pedidos neste caso.

No caso dos testes usando o token de autenticação, como podemos observar no seguinte gráfico, os resultado não são muito dispares dos apresentados anteriormente. Havendo apenas uma ligeira melhoria.

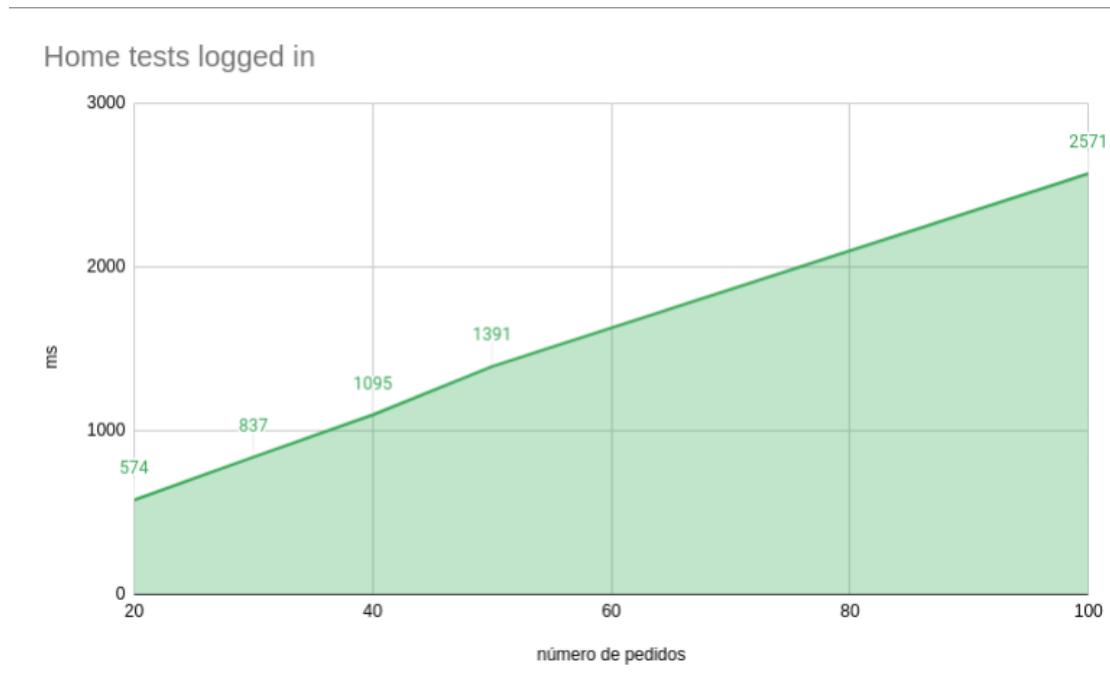


Figura 5.7: Testes no endpoint home usando um utilizador autenticado

5.2.2 Login

Para o endpoint de login foi elaborado um simples pedido que tentasse autenticar um utilizador. Os resultados do teste foram os seguintes:

5.2. TESTES

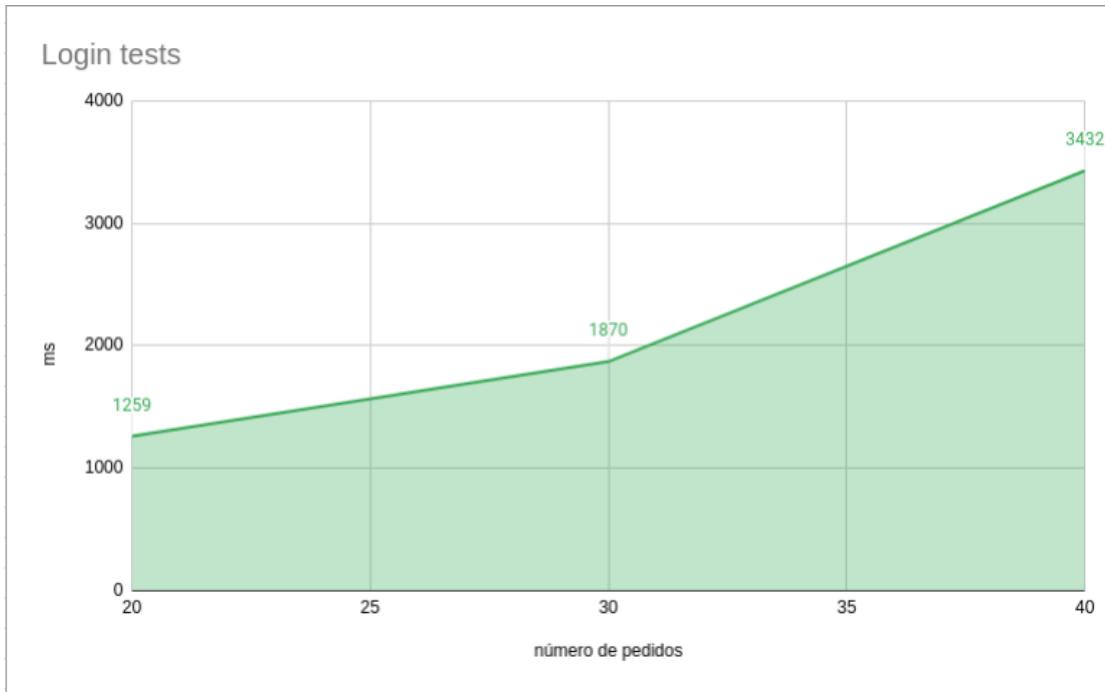


Figura 5.8: Testes no endpoint login

Como podemos ver pelo gráfico os resultados não são muito positivos, bastando 35 pedidos ao mesmo tempo, fazendo um total de 175 pedidos, para que o tempo médio de resposta supere os 2000 milissegundos.

Este mau desempenho pode dever-se ao facto de o servidor estar a gerar um token jwt de todas as vezes que alguém chama o endpoint login.

5.2.3 Search

Este endpoint tem como objetivo retornar as tours ativas de uma dada cidade. Sendo para além disto possível passar outros filtros adicionais como o rating, a categoria as linguagens e a data da mesma.

Para testar foram então criados 2 tipos de pedidos, um que apenas procurava as tour ativas numa determinada cidade e outro que utilizava filtros adicionais como o intervalo de datas, as linguagens em que a tour era falada e o rating.

Os resultados do teste utilizando o primeiro pedido foram os seguintes:

5.2. TESTES

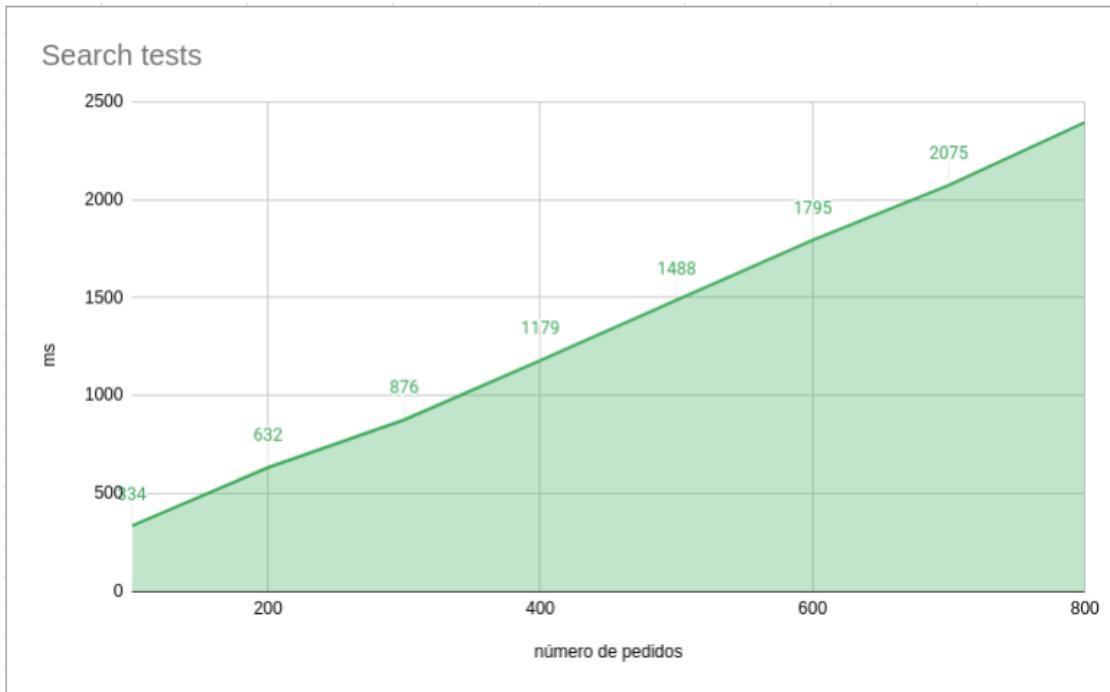


Figura 5.9: Testes no endpoint search

Como podemos ver pelo gráfico, este parece ser um endpoint bastante sólido capaz de aguentar com 600 pedidos em simultâneo, 3000 no total, e ainda assim manter o tempo de resposta médio abaixo dos 2000 milissegundos.

No caso do segundo tipo de pedido, utilizando os filtros, os resultados foram muito parecidos, sendo até ligeiramente melhores. Como podemos observar no seguinte gráfico:

5.2. TESTES

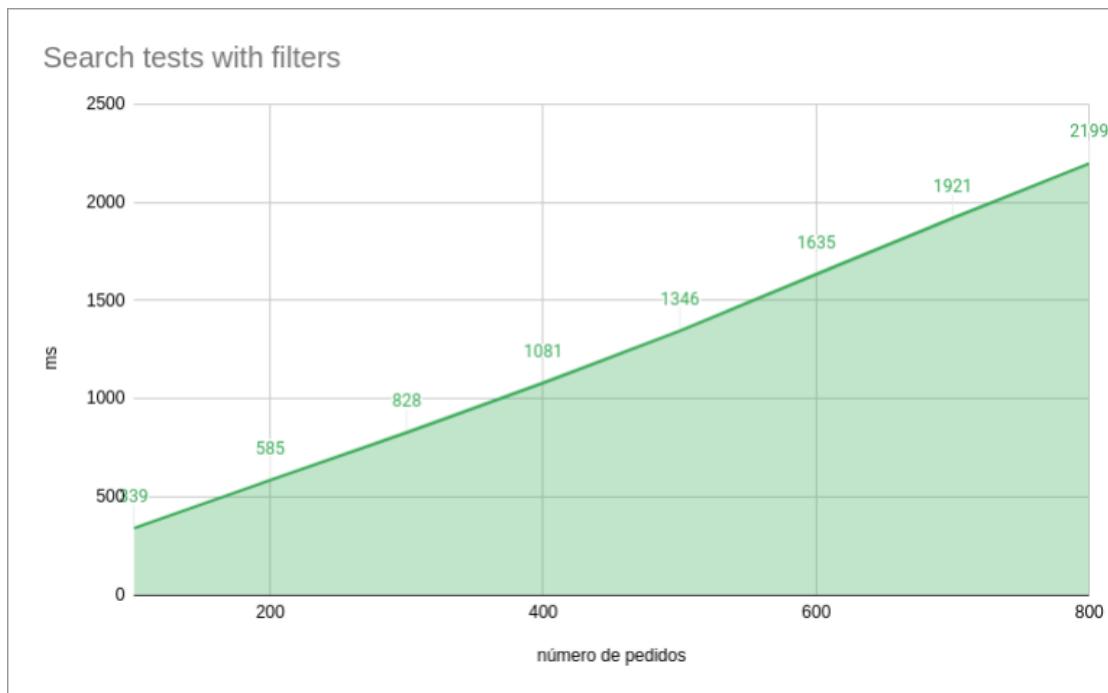


Figura 5.10: Testes no endpoint search utilizando filtros

É de ter em conta que a quantidade de tours presentes na base de dados na altura dos testes era pequena, e que é de esperar que estes resultados piorem com o aumento de tours na base de dados.

Capítulo 6

Conclusão

Dado por concluído o trabalho, surge a necessidade de avaliar a solução final tendo em conta o problema apresentado na introdução. O *FreeTour* atingiu o objetivo inicial de criar uma plataforma web para monitorizar o agendamento e realização de *free tours*. Além dum maior conhecimento de desenvolvimento aplicacional que este projeto proporcionou a nível teórico, foi também uma fonte de pesquisa e aperfeiçoamento de várias técnicas práticas, tanto na lógica do desenvolvimento arquitetural de aplicações, como na conceção de sistemas interativos centrados no utilizador.

6.1 Dificuldades

Com bases bem cimentadas nos assuntos em causa, devido aos conteúdos programáticos lecionados no perfil, o grupo não sentiu muitas dificuldades no desenvolvimento do projeto. As dificuldades que surgiram estavam principalmente relacionadas com detalhes das tecnologias utilizadas, sendo que estas foram esbatidas procurando sempre soluções, tanto novas como já existentes. Um dos problemas a referir, em específico no Spring, era a recursão que existia nos pedidos *get* ao *backend*, em caso de entidades nas quais coincidissem atributos. Por exemplo, no caso das tours e das cidades, uma tour possuía uma cidade e, essa cidade, possuindo várias tours, pesquisava também nessas tours pelas cidades, e assim sucessivamente, criando uma recursão infinita que se tornava um problema no pedido. Esse problema foi resolvido limpando a informação a partir do momento que era repetida mas, apesar de solucionado, foi um problema regularmente presente. Outro problema interessante, mas que se mostrou desafiante, foi a necessidade de utilização de threads assíncronas devido ao SwapManager. Este assunto foi uma novidade e, devido a tal, foi necessário procurar nova informação e realizar novas soluções para este novo problema.

6.2 Trabalho Futuro

Como trabalho futuro seria interessante a introdução de um perfil de administrador capaz de gerir a plataforma a nível de frontend e backend, criando para isso vistas e páginas personalizadas para este e dando a hipótese de fazer operações de gestão da plataforma. Outro ponto seria a introdução de um método de pagamento por parte de quem quer publicitar a tour como forma de rentabilizar o site.

Seria de grande interesse também analisar a base de dados de forma a optimizar o desempenho da mesma, gerindo índices e ou vistas personalizadas, bem como as propriedades do motor da própria.

Em termos de tolerância a falhas seria interessante persistir ou replicar de alguma forma as threads do SwapManager para que este não perca o seu estado se o servidor de backend falhar. Ainda neste tema poderíamos replicar todas as partes da nossa arquitetura com o objetivo de garantir escalabilidade e resiliência.

Um deployment automatizado é nos dias que correm uma peça de grande importância em qualquer software, para que seja possível seguir uma lógica de *continuos delivery/continuos integration*, para que a nossa aplicação não fique para trás seria interessante implementar um deployment automático usando ferramentas como *Ansible*.

Por ultimo, é de grande importância olhar para todos os endpoints com o objetivo de procurar otimizações de desempenho que permitam reduzir o tempo de resposta dos mesmos. Podendo até recorrer se a métodos assíncronos para garantir melhores resultados.