Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia informática

Unidade Curricular de Laboratórios de Informática IV

Ano Letivo de 2021/2022

Trabalho Prático de LI4

Grupo 17

Cristiano Pereira, 93726

João Martins, 91669

Jorge Lima, 93183

Rúben Santos, 93257

Data de Receção	
Responsável	
Avaliação	
Observações	

Trabalho Prático de LI4

Grupo 17

Cristiano Pereira, 93726

João Martins, 91669

Jorge Lima, 93183

Rúben Santos, 93257

Resumo

No âmbito da Unidade Curricular (UC) de Laboratórios de Informática IV (LI4), foi proposto o desenvolvimento de um guia turístico-cultural dos museus da cidade de Braga, apelidado por *DiscoveryDoor*.

Foi desenvolvida uma aplicação mobile, que partiu de uma especificação realizada pelo grupo dezoito (18). Inicialmente foi realizada uma análise meticulosa do documento, que resultou na produção de um conjunto de notas, entregues ao docente responsável pela UC. Este processo de análise continuou após a entrega, e ocupou uma parte substancial do nosso tempo, sendo que serviu de fundação para grande parte do conteúdo apresentado no Capítulo 2 e, subsequentemente, para o desenvolvimento da aplicação.

Em relação à nossa análise da especificação que nos foi entregue, constatámos algumas inconsistências. Neste documento iremos descrever sucintamente os problemas que encontramos e como os solucionamos.

Em relação à aplicação em si, esta cumpre o objetivo proposto de guiar os seus utilizadores pela cidade de Braga, apresentando os mais variados pontos de interesse cultural, nomeadamente museus. Os utilizadores podem procurar por museus do seu interesse ou, se preferirem, podem filtrar os pontos de interesse por categoria, distância ou preço do bilhete de entrada. Assim que selecionarem um destino, é lhes apresentado um conjunto de informações do museu em questão, sendo que podem obter direções a partir da sua localização. Os utilizadores podem também deixar um comentário e uma avaliação do museu visitado, sendo que o registo da sua visita fica guardado num histórico de visitas, que pode ser consultado posteriormente.

Área de Aplicação: Engenharia de Software: Desenvolvimento de Sistemas de Software, Gestão de Projetos de Software.

Palavras-Chave: Desenvolvimento de Aplicações, Java; Arquitetura de Aplicações; Bases de Dados, MySQL; Interfaces Gráficas Flutter, Desenvolvimento Web, REST API; Maps Platform; Plataformas Cloud, Azure.

Índice

Índice de Figuras

1. Introdução

Neste capítulo apresentamos uma descrição geral do trabalho realizado, começando com um pequeno guia de como está organizado este documento. Partimos depois para a caracterização do trabalho desenvolvido nesta segunda fase do trabalho, os recursos utilizados e, finalmente, o plano de desenvolvimento seguido.

1.1. Enquadramento geral

O Capítulo 1 apresenta o trabalho realizado nesta segunda fase, descreve também a estrutura e metodologia do desenvolvimento, assim como os recursos utilizados – software de desenvolvimento e de suporte.
O Capítulo 2 apresenta a estratégia e metodologia adotada para desenvolver o sistema proposto. É aqui onde justificamos as decisões que tomamos no processo de desenvolvimento, nomeadamente aquilo que decidimos implementar de forma diferente ou que decidimos adicionar e que não é proposto na especificação. É neste capítulo que apresentamos e justificamos a arquitetura da nossa aplicação e de todos os seus componentes.
O Capítulo 3 (último) expõe as nossas dificuldades com a implementação da aplicação e destaca aquilo que achamos interessante desenvolver no futuro, de modo a tornar a solução mais correta.

1.2. Caracterização do Trabalho a Desenvolver

Neste sub-capítulo procuramos caracterizar o trabalho que temos de fazer, de acordo com a especificação fornecida. Foram realizadas posteriores alterações à proposta inicial, sendo que estas se encontram descritas e justificadas no Capítulo 2.

1.2.1 Implementação da Interface Gráfica

No que diz respeito à parte visual da aplicação, foi sugerida a criação dos seguintes menus, apresentados na secção de Interfaces do Sistema.

- Menu de *Login* – relativo ao *Use Case*: Autenticação.
- Menu de Registo – relativo ao *Use Case*: Criar Conta.

- Menu Principal – apresentado após menu de *Login*, constituído pelo mapa da área próxima da localização do utilizador e uma barra de pesquisa.
- Menu “Lista Filtrada de Museus” – filtrar os museus por raio, classificação e tema.
- Menu “Museu Selecionado” – apresentado após selecionar museu da lista filtrada de museus ou pesquisar um nome de museu na barra de pesquisa do menu principal.
- Pop-up “Museu Não Encontrado” – fluxo de Exceção do Use Case: Pesquisar um museu dado um nome.
- Menu “Informação do Museu” – apresentado após interagir com o menu do museu selecionado. Apresenta informações do museu, algumas fotos e a possibilidade de pedir direções.
- Menu “Rota para Museu” – apresentado quando o utilizador desejar obter direções para um determinado museu. O caminho aparece destacado no mapa.
- Menu “Review de Museu” – apresentado se o utilizador desejar fazer um comentário/avaliar o museu selecionado.
- Menu “Histórico de Visitas” – apresenta os museus já visitados, assim como algumas informações da visita, em formato de lista.
- Menu de “Logout” – apresentado quando o utilizador sai da aplicação, pede para que seja introduzida uma nota de 1-5 para avaliar o sistema (mostrado sempre que o utilizador faz L*ogout*).

Foram também propostos alguns menus para a aplicação do administrador, nomeadamente:

- Menu de *Login* – relativo ao *Use Case*: Autenticação.
- Menu de “Lista de Museus do Administrador” – apresentado após a autenticação de um administrador, exibe todos os museus do administrador e suas informações. Oferece a possibilidade para eliminar museus.
- Menu “Editar ou Criar Meseu” – relativo ao *Use Case*: Adicionar Museu. Não existe *Use Case*: Editar Museu.

Em anexo, no documento que serve de guião para este projeto, podem ser consultadas as interfaces sugeridas. As questões acerca da correção das interfaces são expostas no Capítulo 2.

1.2.2 Lógica da Aplicação

No que diz respeito às funcionalidades da aplicação, extraímos a seguinte lista a partir dos *Use Case* propostos na especificação.

- Autenticação – Autenticar um utilizador na aplicação, através do seu nome de utilizador e palavra-passe. Este processo requer acesso à localização do utilizador que pode ou não ser fornecida.

- Criar conta – Criar e armazenar um par (nome de utilizador, palavra-passe) que possibilite a autenticação de um Utilizador.
- Pesquisar um museu dado um nome – Consiste em procurar pelo museu desejado através do seu nome. A procura resulta no museu desejado, caso este exista.
- Pesquisar vários museus dadas especificações – Consiste em fornecer uma lista de museus baseada em filtros escolhidos pelo utilizador (distância, preço, tema, classificação).
- Obter instruções sobre a rota para um museu – Consiste em obter uma lista de instruções que guiem o utilizador desde a sua localização atual até ao museu destino.
- Aceder a registos de museus visitados – Os museus visitados ficam registados num histórico, que o utilizador pode consultar posteriormente à visita.
- Fazer *review* do museu – O cliente deixa um comentário e/ou avaliação do museu que visitou. Não é verificado se o utilizador visitou realmente este museu.
- Avaliar Sistema – Após o *Logout*, o utilizador avalia o desempenho da aplicação.
- Adicionar Museu – Esta é uma funcionalidade do administrador, sendo que consiste em adicionar um novo museu à base de dados, com toda a informação relevante.
- Remover Museu – Esta é uma funcionalidade do administrador, sendo que consiste em remover um museu da base da dados.

De modo geral, isto é o que nos pedem para desenvolver, de acordo com a especificação fornecida. As questões acerca da correção dos *Use Case* são expostas no Capítulo 2.

1.2.3 Base de Dados

A base da dados proposta traduz a seguinte lista de objetivos:

- Utilizador – Esta tabela tem como função armazenar a informação do utilizador, nomeadamente: nome de utilizador, palavra-passe, avaliação do sistema, registo (histórico visitas), comentários, avaliação dos museus.
- Museu – Esta tabela tem como função armazenar a informação dos museus, nomeadamente: nome, categoria, preço de entrada, website, contacto de telefone, localização, número de estrelas, fotografias.
- Registo – Esta tabela tem como função armazenar o histórico de visitas do utilizador, sendo constituída por: museu, data e hora, id.
- Administrador – Esta tabela tem como função armazenar a informação do administrador, nomeadamente: identificador e palavra-passe.

É proposto o desenvolvimento de mais duas tabelas: avaliação e comentário. Porém, o esquema conceptual do sistema de dados, apresentado na especificação recebida, não traduz a mesma necessidade, dado que estes últimos representam informação relativa ao utilizador e portanto são armazenados na tabela de utilizador.

No que diz respeito à base de dados apresentada, não se encontra explícito como devemos organizar as tabelas, quais são as chaves principais, quais são tipos dos elementos que vamos guardar. É pedido para que o registo tenha o percurso realizado pelo utilizador, mas não se encontra explicado como é que devemos fazer ou sequer armazenar esta informação.
As restantes questões acerca da correção da base de dados são expostas no Capítulo 2.

1.3. Recursos Utilizados

Os recursos sugeridos no enunciado recebido não foram aceites na totalidade, de acordo com as justificações apresentadas abaixo.

- **Visual Paradigm**
  - Utilizado para criar todos os diagramas apresentados neste documento e outros rascunhos, que serviram de suporte ao desenvolvimento.
  - Esta escolha de software foi feita, em parte, pois todos os elementos do grupo tem acesso e já estão familiarizados com a ferramenta.
  - Foi usada também devido às suas funcionalidades de gerar código a partir de modelos e vice-versa.
  - É uma ferramenta que possibilita o desenvolvimento de praticamente todos os diagramas e modelos que existem.
  - É amplamente utilizada em ambientes académicos.
- **GanttProject**
  - Uma ferramenta utilizada unicamente para fazer diagramas de Gantt.
  - A utilização deste software deve-se ao facto de que o Visual Paradigm não foi capaz de responder à nossa necessidade de criar um diagrama de Gantt.
  - Todos os elementos do grupo reportaram dificuldades ao tentar fazer um diagrama de Gantt no Visual Paradigm (o projeto não conseguia ser fechado e retomado mais tarde).
  - É uma aplicação fácil de usar, com a possibilidade de alocar recursos às tarefas.
  - É uma aplicação gratuita e matura, sendo que se encontra no mercado desde 2003 e é utilizada por milhares de utilizadores.
  - Oferece a possibilidade de exportar o diagrama como imagem ou pdf.
- **Java**
  - Esta foi a linguagem de programação escolhida para desenvolver o *backend* da aplicação.
  - É uma linguagem *open-source,* orientada a objetos, fácil de usar, fácil para fazer *debug* e independente da plataforma em que é desenvolvida.
  - Todos os elementos do grupo são proficientes no que diz respeito à manipulação dos mais diversos mecanismos da linguagem – desde encapsulamento, herança e polimorfismo.

- É uma linguagem que faz uso do mecanismo de *garbage collection*, sendo que *não* temos de ter preocupações com gestão de memória.
- É uma linguagem utilizada a nível global e detêm mais de um terço do mercado de desenvolvimento.
- Dada a sua dimensão, não é difícil encontrar soluções para os mais variados problemas.
- Tem uma documentação bastante compreensiva.

- **Flutter**
  - Este foi o *kit* de desenvolvimento utilizado para implementar a interface gráfica da aplicação mobile do utilizador, assim como a aplicação web do administrador.
  - É *open-source* e está implementado na linguagem *Dart* – linguagem de *script* orientada à web, multi-paradigma e desenvolvida pela *Google*.
  - O *Flutter* é desenvolvido pela *Google* e é multi-plataforma, ou seja, permite desenvolver aplicações *Android*, *iOS*, *Linux*, *Windows*, *macOS* e *Web*, entre outras.
  - É um *kit* de desenvolvimento altamente produtivo, devido ao facto de ser multi-plataforma.
  - Permite que o desenvolvimento seja rápido mantendo a sua simplicidade.
  - É fácil de utilizar e parece uma escolha óbvia para quem nunca fez uma interface gráfica, pois tem incluídas as mais variadas bibliotecas com muitos elementos gráficos, p. ex botões, menus, abas laterais, entre outros.
  - Fácil incorporação com a API do *Google Maps* (*Maps Platform*), devido ao facto da linguagem e do kit serem desenvolvidos e suportados pela *Google*.
  - Documentação e muitos tutoriais de desenvolvimento disponíveis online.
- **Maps Platform**
  - Esta plataforma é utilizada para obter direções entre locais, assim como a representação gráfica do mapa e da localização do utilizador.
  - É utilizada por mais de 80% das aplicações mobile que implementam mapas, direções e informação sobre pontos de interesse.
  - A sua API é fácil de utilizar e tem integração com o kit de desenvolvimento escolhido para o projeto (*Flutter*).
  - É desenvolvido pela Google, e possui um alcance geográfico muito bom, incluindo, como é óbvio, a cidade de Braga, que é o alvo da aplicação.
- **MySQL**
  - Este foi o sistema de gestão de base de dados utilizado no nosso projeto.
  - É um dos sistemas mais utilizados e detêm quase 50% do mercado, sendo que compete com outras dezenas de gestores de base de dados.
  - É suportado pela plataforma de *clouding* utilizada neste projeto.
  - É recomendada e utilizada pelos líderes da indústria de tecnologia, desde empresas como o *Facebook*, *Bank of America, Tesla, Uber, Booking.com, Spotify, YouTube, Netflix*, entre muitos outros.

- **Azure**
  - Esta foi a plataforma de *cloud* que utilizamos neste projeto, onde fica alojada a nossa base de dados e o *backend* da aplicação.
  - É um serviço fácil e intuitivo de utilizar, sendo que nos adaptamos com facilidade e rapidamente às suas funcionalidades.
  - Esta plataforma é a que tem mais *data centers* das demais plataformas de cloud e, no nosso caso, é a mais indicada devido à rapidez com que os pedidos são respondidos.
  - É desenvolvida pela *Microsoft*, empresa que detêm mais de 75% do mercado de sistemas operativos para *desktops*.
- **REST API**
  - São simples e fáceis de utilizar, sendo que são um standard na indústria do desenvolvimento de aplicações web.
  - É escalável e *stateless*, ou seja, não é guardado um estado do servidor e cada pedido é independente dos demais.
  - Faz a gestão da comunicação entre o cliente e o servidor e garante que os pedidos do cliente são processados concorrentemente.

Em suma, estes foram os recursos utilizados para desenvolver o projeto. Ao contrário do que foi sugerido no enunciado recebido, não utilizamos a linguagem de programação *C#* nem a plataforma *Unity*. Como explicamos acima, consideramos a linguagem *Java* e o *Kit* de desenvolvimento *Flutter* mais apropriado para este problema.

1.4. Plano de Desenvolvimento

O diagrama de *Gantt* apresentado abaixo, foi desenvolvido no dia 24 de dezembro de 2021 e foi cumprido integralmente por todos os elementos do grupo. As tarefas que ocorrem em paralelo são feitas por diferentes elementos do grupo.

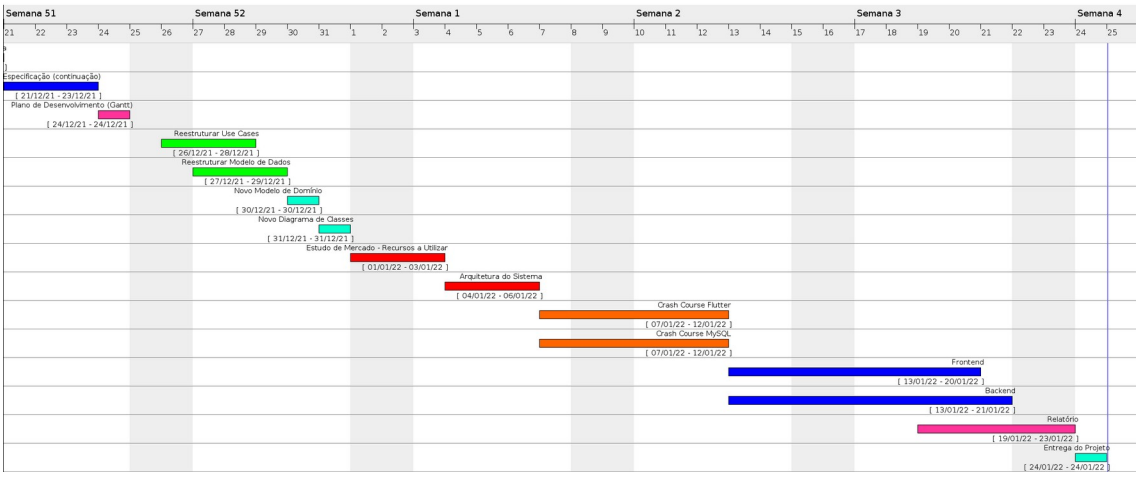

Figure 1: Plano de Desenvolvimento (Diagrama de Gantt).

O diagrama é constituído pelas seguintes tarefas:

1. **Avaliar Especificação** (21 a 23 de dezembro): Após a entrega da apreciação do enunciado recebido, continuamos o processo de análise do documento, de forma a compreender completamente a modelação que nos foi entregue. Três dias são suficientes para terminar a análise.
2. **Reestruturar Use Cases** (26 a 28 de dezembro): Após a análise da especificação, foi necessária a reestruturação dos use cases, por motivos explicados no Sub-Capítulo 2.2. Três dias são suficientes para a reestruturação, sendo que tem de ser, em parte, intercalada com a reestruturação do modelo de dados.
3. **Reestruturar Modelo de Dados** (27 a 29 de dezembro): Após a análise da especificação, foi necessária a reestruturação do modelo de dados, por motivos explicados no Sub-Capítulo 2.4.3. Três dias são suficientes para este processo, sendo que o mesmo tem de ser, em parte, intercalado com a reestruturação dos use cases.
4. **Novo Modelo de Domínio** (30 de dezembro): Foi necessário voltar a modelar o problema, por motivos explicados no Sub-Capítulo 2.2 Sendo que toda a gente participa neste processo, uma tarde é suficiente para o fazer.
5. **Novo Diagrama de Classes** (31 de dezembro): Foi necessário adaptar o diagrama de classes, sendo que o que foi proposto não se enquadra com os Use Case nem Arquitetura de uma Aplicação mobile ou web. Explicamos mais acerca desta reestruturação no Sub-Capítulo 2.2. Sendo que toda a gente participa neste processo, uma tarde é suficiente para o fazer.
6. **Estudo de Mercado – Recursos a Utilizar** (1 a 3 de janeiro): É necessário definir com que tecnologias vamos trabalhar - que linguagens de programação vamos utilizar, como vamos desenvolver a parte gráfica do projeto, como vamos desenvolver a base de dados, entre outros. Para tal, precisamos de fazer um estudo de mercado. Três dias

são necessários para tomar estas decisões, sendo que terão um impacto profundo na arquitetura da mesma.

7. **Arquitetura do Sistema** (4 a 6 de janeiro): Como vai ser a arquitetura do sistema? Não deve ser difícil desenhar um modelo do sistema depois de escolhermos que tecnologias vamos utilizar. Dois dias são suficientes para desenhar a arquitetura do sistema.
8. ***Crash Course – Flutter*** (7 a 12 de janeiro): Metade dos elementos do grupo desenvolve, ao longo de seis dias, as suas *hard skills* de programação utilizando o *kit* de desenvolvimento *Flutter*. Dado que nenhum elemento está suficientemente familiarizado com o desenvolvimento de interfaces gráficas, são necessários seis dias para esta tarefa.
9. **Crash Course – *MySQL*** (7 a 12 de janeiro): Metade dos elementos do grupo desenvolve, ao longo de seis dias, algumas noções de como utilizar a plataforma de gestão de bases de dados *MySQL*. Dado que nenhum elemento está suficientemente familiarizado com o desenvolvimento de bases de dados, são necessários pelo menos quatro, mas idealmente seis dias para esta tarefa.
10. **Frontend** (13 a 20 de janeiro): O desenvolvimento do *frontend* consiste em seguir as interfaces já propostas (com as devidas alterações aos *Use Case* e Bases de Dados discutidas posteriormente). Prevemos que oito dias são necessários e, provavelmente, suficientes para implementar e testar o *frontend*.
11. **Backend** (13 a 21 de janeiro): O desenvolvimento do *backend* consiste em seguir os *Use Case* já alterados. Damos mais um dia do que para o desenvolvimento do *frontend*, para que possamos popular a base de dados.
12. **Relatório** (19 a 23 de janeiro): O relatório pode começar a ser feito por um elemento do grupo, enquanto que os restantes lidam com eventuais problemas de implementação. Seis dias para escrever um relatório, dado que já temos imensos apontamentos, diagramas e atas de reuniões posteriores, será suficiente.
13. **Entrega do Projeto** (24 de janeiro): Neste último dia, iremos limar as últimas arestas, gerar o ficheiro para entregar o projeto e enviar os documentos para a equipa docente (*.apk* e *.pdf*).

2. Desenvolvimento da Aplicação

Neste capítulo é discutida a metodologia de desenvolvimento utilizada, assim como todos os detalhes que dizem respeito à implementação da solução. São apresentadas as alterações que foram feitas à especificação recebida assim como a arquitetura escolhida para a aplicação, acompanhada com uma explicação dos seus componentes.

2.1. Estratégia e Metodologia de Desenvolvimento

Tal como referido no Capítulo 1 (Plano de desenvolvimento), para começar este projeto tivemos que analisar e avaliar a especificação que nos foi entregue. Os problemas encontrados foram resolvidos em reuniões marcadas para o efeito. Estas reuniões encontram-se descritas no plano de desenvolvimento.

A partir do momento em que a especificação recebida foi alterada ao ponto de não existirem mais ambiguidades, adotamos um desenvolvimento iterativo e incremental do sistema. Esta metodologia consiste em: fazer uma análise do estado atual do sistema, alterar o design se encontrarmos novos problemas com a implementação e implementar a solução. Após a primeira iteração de desenvolvimento obtemos um protótipo do sistema e voltamos ao início, ou seja, voltamos a fazer uma análise do estado atual do protótipo desenvolvido.

Uma das vantagens desta metodologia é que esta permite a exploração do sistema desde cedo, incluindo a identificação de novos requisitos e de novos problemas. Temos assim uma maior capacidade de lidar com incerteza e maior controlo de risco. Isto, contudo, implica que decisões tomadas inicialmente podem revelar-se inadequadas mais tarde

2.2. Caracterização da Aplicação e seus Serviços

O objetivo principal da aplicação é a criação de um guia turístico-cultural de museus da cidade de Braga. A aplicação permite localizar e obter direções para os museus. Regista também o *feedback* dos utilizadores em relação às suas visitas.

Para implementar aquilo que é pedido na especificação recebida, decidimos dividir o projeto em dois componentes:

1. A aplicação mobile (dirigida aos utilizadores).
2. A aplicação web (dirigida aos administrador).

O desenvolvimento destas duas aplicações não é explicitamente pedido na especificação do projeto, mas é subentendida esta divisão devido à forma como as funcionalidades estão expostas e, principalmente, nos esboços das interfaces que nos foram apresentadas.

2.2.1 Aplicação mobile

No Sub-Capítulo 1.2 apresentamos sucintamente as funcionalidades propostas na especificação que recebemos. Abaixo, procuramos descrever as funcionalidades que foram desenvolvidas, realçando tudo aquilo que decidimos alterar, eliminar e adicionar.

- **Autenticação**
  O nosso sistema permite a autenticação do utilizador. Caso o nome ou a palavra-passe esteja errada, o sistema informa o erro na autenticação e volta a pedir as informações. Se a autenticação foi bem-sucedida o sistema obtém a localização do utilizador, caso esta não esteja disponível o sistema pede permissão. Após a autenticação, é apresentado o menu principal com o mapa da em volta da localização do utilizador ou, caso o sistema não tenha acesso à localização, é exibido o mapa da cidade de Braga.
- **Criar conta**
  Os utilizadores podem criar uma nova conta. É pedido que seja fornecido um nome de utilizador e uma palavra-passe. Caso o nome de utilizador esteja repetido, o nosso sistema volta a pedir as informações. Apesar de não estar especificado, para garantir a funcionalidade da aplicação o sistema dirige o utilizador para o menu principal caso a criação de conta seja bem-sucedida. Permitimos também que este cancele o registo da conta a qualquer momento e volte para o estado anterior.
- **Pesquisa de um museu por nome**
  A pesquisa do museu por nome é uma de duas maneiras que possibilitam encontrar os museus disponíveis. O utilizador insere o nome do museu e o sistema procura, caso o museu exista, vai para a página desse museu, caso contrário o sistema indica, através de uma mensagem de pop-up, que não foi encontrado.
- **Pesquisar vários museus dadas especificações**
  Neste segundo método de pesquisa, o utilizador pode escolher alguns filtros para os museus que pretende encontrar. Em qualquer altura da pesquisa o utilizador pode ajustar os parâmetros a seu gosto. Apesar de não estar explícito na especificação, subentendemos alguns deste filtros a partir do esboço da interface correspondente. A nossa implementação disponibiliza uma procura por raio a partir da localização do utilizador em que o museu se encontra, pelo seu preço, a sua classificação e a sua categoria. No caso do raio e do preço foi especificado que estes só podem estar dentro de certos intervalos de valores, mas não foram especificados estes valores. Depois de alguma consideração escolhemos os intervalos de 0 a 20 km e 0 a 100 € como valores

razoáveis. No caso de nenhum museu estar disponível de acordo com os filtros utilizados, é apresentada uma lista vazia.

- **Obter instruções sobre a rota para o museu**

  É gerada uma rota para o museu indicado, sendo esta exibida ao utilizador a partir do mapa. O progresso do utilizador é também exibido, à semelhança de uma aplicação do tipo Google Maps. Aqui o utilizador também pode visualizar o tempo que demora a chegar ao dado museu usando esta rota e um meio de transporte indicado. Se o meio de transporte não for indicado assumimos que este viaje a pé. Quando chegar ao destino o utilizador indica ao sistema que o fez e o sistema gera um registo da visita do museu. Não foi especificado em que consistia este registo.

- **Aceder a registos de museus visitados**

  O utilizador pode ver o seu histórico de visitas. Mais uma vez, assumimos que a informação apresentada no esboço das interfaces desta funcionalidade, é aquela que temos de guardar, isto porque não é especificado qual a informação que deve ser guardada neste registo. Além das informações do museu, o sistema guarda a hora e data da visita, assim como o tempo que o utilizador demorou a chegar ao museu.

- **Avaliar sistema**

  Quando o cliente deseja sair da aplicação, o sistema pede ao utilizador para avaliar a sua experiência com a aplicação numa escala de 0 a 5.

- **Fazer review do museu**

  O sistema disponibiliza a possibilidade de avaliar um museu. É pedido ao utilizador para inserir uma classificação e um comentário, se assim o desejar. Esta operação pode ser cancelada voltando para o estado anterior. Caso a *review* seja registada com sucesso, é guardada e é atualizado a classificação média do museu.

- **Terminar Sessão**

  Esta funcionalidade permite ao utilizador sair da aplicação e terminar a sua sessão. Este foi um dos *Use Case* criado para permitir que o utilizador possa avaliar o sistema. Terminar a sessão inclui avaliar o sistema.

2.2.2 Aplicação web

Na aplicação web, o administrador, assim que autenticado, pode inserir, editar e remover museus existentes.

Na especificação, a autenticação do utilizador e do administrador é considerada uma só, mas, nesta nova redefinição do sistema, isto não acontece e a autenticação do administrador é uma funcionalidade à parte. Tal como na aplicação mobilem, vimos a necessidade de acrescentar a funcionalidade de terminar sessão.

Desta forma, apresentamos as seguintes funcionalidades implementadas na aplicação web desenvolvida:

- **Autenticação**

  O sistema permite a autenticação do administrador, caso o nome ou a palavra-passe esteja errada, o sistema informa o erro na autenticação e volta a pedir as informações. Se a autenticação foi bem-sucedida o sistema apresenta o menu principal da aplicação web, onde se encontram todos os museus da aplicação em forma de uma lista.

- **Adicionar Museu**

  Mais uma vez, a especificação é um pouco vaga no que diz respeito a esta funcionalidade, e tivemos de retiramos as informações a partir dos esboços das interfaces. Pedimos ao administrador que nos forneça o nome, preço, localização, website, contactos, categoria e imagens do museu. Caso o nome do museu se repita no nosso sistema ou não ser preenchido algum campo, o sistema volta a pedir as informações.

- **Remover Museu**

  O administrador pode optar por remover um dos museus. Quando faz este pedido é imediatamente removido do sistema este museu. A especificação faz referência ao caso do museu já não existir no sistema, mas isto nunca vai acontecer pois a remoção apenas pode acontecer selecionando um dos museus já dentro do sistema.

- **Editar Museu**

  Esta é uma funcionalidade que não está especificada mas que é subentendida no esboço das interfaces. Visto que assumimos muitas coisas com base nestes esboços, decidimos assumir que será da vontade do grupo que realizou a especificação o desenvolvimento de uma funcionalidade de edição de um museu existente. Nesta edição, podemos alterar as informações atuais de um museu.

- **Terminar Sessão**

  Esta funcionalidade permite ao administrador sair da aplicação e terminar a sua sessão.

2.3. Arquitetura da Aplicação

A arquitetura geral da aplicação desenvolvida pode ser vista a partir da seguinte maqueta. Esta é uma versão simplificada do sistema, posteriormente serão fornecidos mais detalhes.

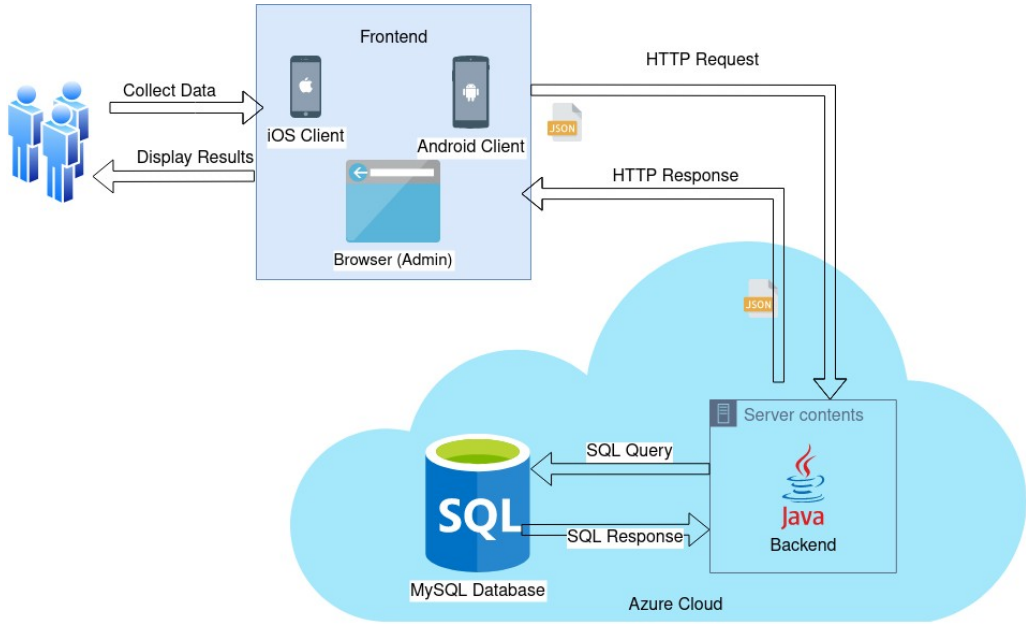

Figure 2: Maqueta do Sistema

A aplicação foi implementada seguindo a arquitetura REST. Os pedidos dos clientes são realizados por HTTP, sendo que a comunicação cliente-servidor é *stateless*, ou seja, não é guardado um estado do servidor e cada pedido é independente dos demais.

Esta arquitetura é fácil de aprender e utilizar e é direcionada para um modelo cliente-servidor, tal como o nosso problema exige. O REST é uma API escalável, sendo que usa JSON para trocas de dados. O JSON é mais compacto e o seu *parsing* é mais rápido do que XML. Finalmente, esta arquitetura oferece muita flexibilidade, isto porque os dados não estão associados exclusivamente com os recursos ou métodos, o que implica que o REST consiga gerir múltiplos tipos de pedidos e retornar diferentes formatos de dados.

Optámos também por utilizar uma plataforma de cloud para facilitar a forma como iremos disponibilizar e testar a aplicação ao longo do desenvolvimento. Desta forma a aplicação está em execução 24h, e podemos todos trabalhar no desenvolvimento da mesma. Após o lançamento da aplicação, uma solução cloud é a maneira mais fácil de manter o serviço operacional.

2.4. Componentes da Aplicação

A aplicação, como podemos ver na figura 2, é constituída pelos seguintes componentes: *frontend*, *backend* e base de dados.

Neste sub-capítulo temos como objetivo definir cada componente e descrever o seu papel na solução desenvolvida.

2.4.1 Frontend

O *frontend* é o componente que interage com os utilizadores, sendo que é responsável por ler inputs e apresentar conteúdo, recorrendo a chamadas ao *backend* ou a rotinas locais. Em suma, este componente implementa a interface gráfica e interage com a lógica da aplicação, via a REST API.

O *frontend* foi implementado com o kit de desenvolvimento Flutter, o que faz com que a aplicação seja compatível com dispositivos Android e iOS. Este componente interage com o backend via pedidos HTTP, sendo que recebe as informações em formato JSON.

Abaixo podemos ver as interfaces desenvolvidas, acompanhadas de breves descrições.

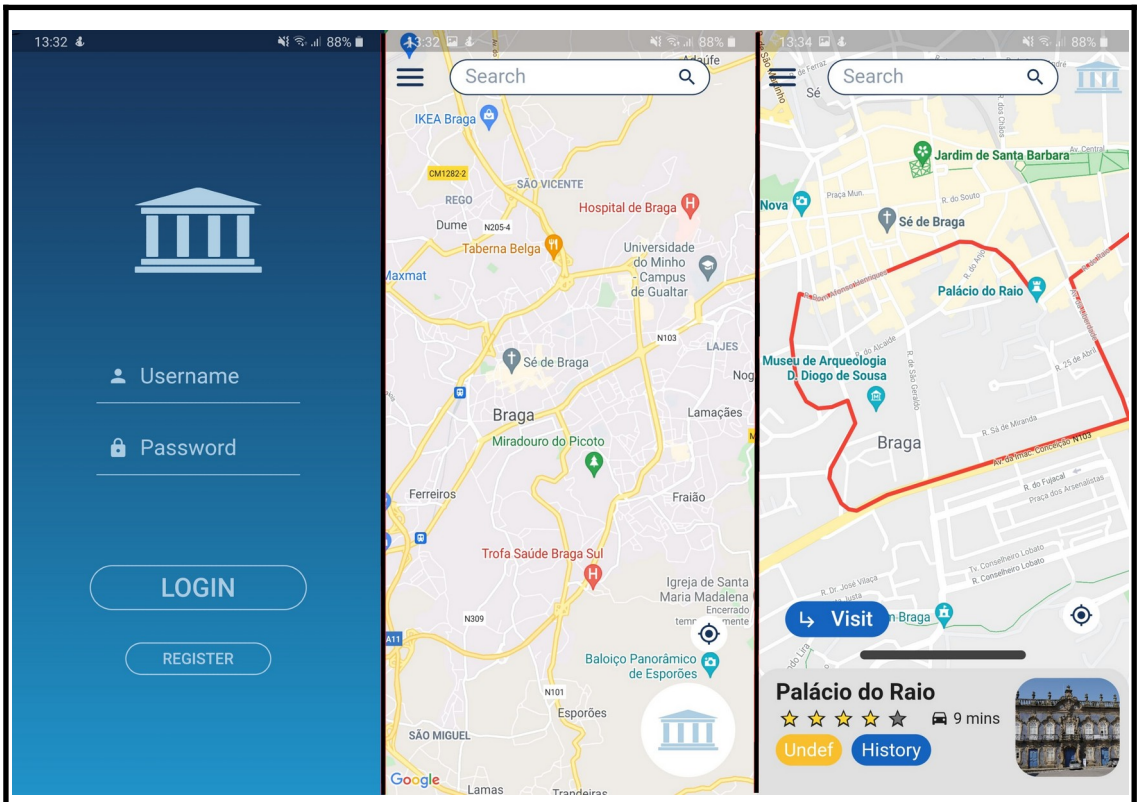

Figure 3: Login, Menu Principal e Menu de pré-visualização de museu.

A imagem da esquerda representa o menu de login, onde os utilizadores se autenticam no sistema ou onde podem decidir fazer um registo, caso não tenham conta criada.

A imagem central é o menu principal, que surge após o login. Neste menu podemos procurar por museus a partir do seu nome (barra de pesquisa acima), podemos utilizar o botão no canto inferior direito para pesquisar museus a partir de filtros e podemos utilizar a barra lateral (botão canto superior esquerdo) para aceder ao histórico ou terminar a sessão.

A imagem da direita surge após uma pesquisa bem sucedida de um museu a partir do seu nome. Podemos arrastar este menu para cima para acedermos ao menu do museu ou podemos ativar o botão *visit* para obter direções para o museu a partir da nossa localização.

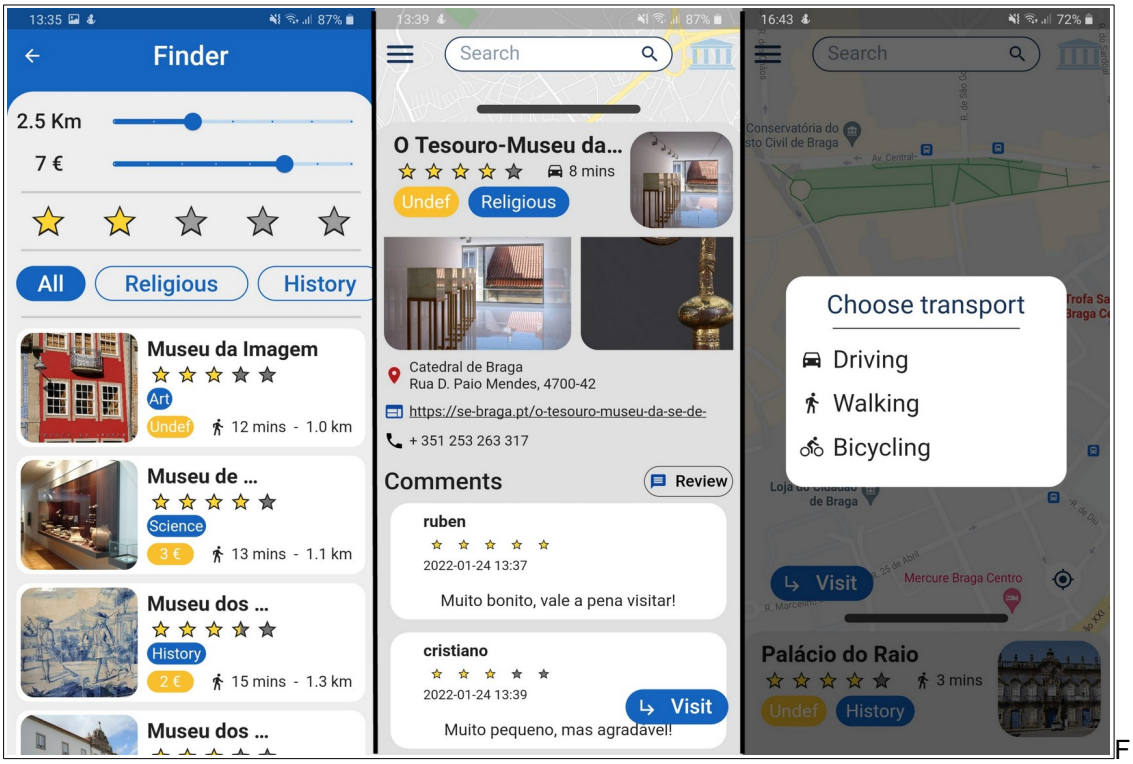

Figure 4: Menu de procura por filtros, Menu de museu, Popup para escolher transporte.

A imagem da esquerda representa a procura de museus utilizando filtros, neste caso os filtros aplicados são: museus num raio de 2,5 Km, cujo bilhete custa menos de 7€ e que tem uma pontuação de pelo menos 2 estrelas.

A imagem central corresponde ao menu apresentado após selecionar um dos museus da lista da imagem da esquerda. Aqui podemos fazer comentários, avaliar o museu, consultar algumas informações e imagens e obter direções para o museu respetivo.

Finalmente, a imagem da direita corresponde ao *popup* apresentado pelo sistema quando ativamos o botão com o símbolo do carro/bicicleta/pessoa. Caso o utilizador selecione um meio de transporte diferente daquele já selecionado, o caminho apresentado no ecrã é alterado.

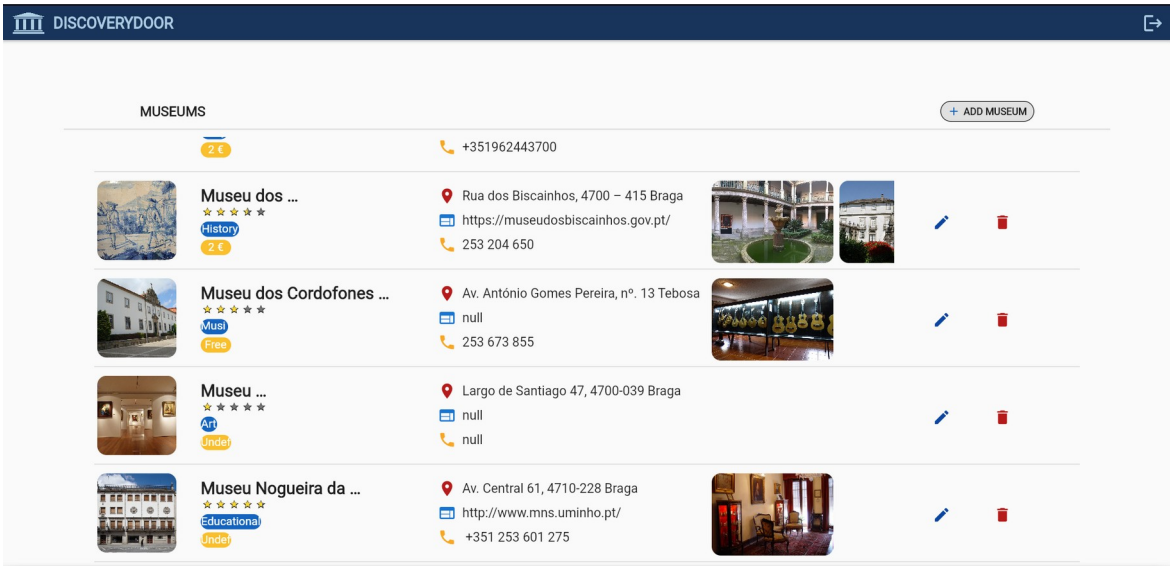

Figure 5: Menu principal da aplicação web (para administradores).

A figura acima diz respeito ao menu principal da aplicação desenvolvida para os administradores do sistema. Esta aplicação oferece a possibilidade para os administradores adicionarem, removerem e editarem os museus existentes.

2.4.2 Backend

O *backend* é o componente que responde a pedidos do *frontend*, recorrendo a *queries* à base de dados ou a rotinas locais.

Os pedidos são recebidos com a ajuda da REST API, sendo que o protocolo utilizado para enviar e receber dados do *frontend* é o HTTP. A REST API garante que o *backend* recebe e processa os pedidos dos clientes de forma concorrente, sem que nós, programadores, nos tenhamos de preocupar com estes detalhes.

É aqui onde são implementadas todas as funcionalidades da aplicação descritas anteriormente no Capítulo 2. O backend é implementado em Java, pelos motivos apresentados no Capítulo 1.

2.4.3 Base de Dados

A base de dados é o componente onde são armazenados todos os dados da aplicação. Abaixo podemos ver a sua estrutura assim como uma breve descrição de cada entidade.

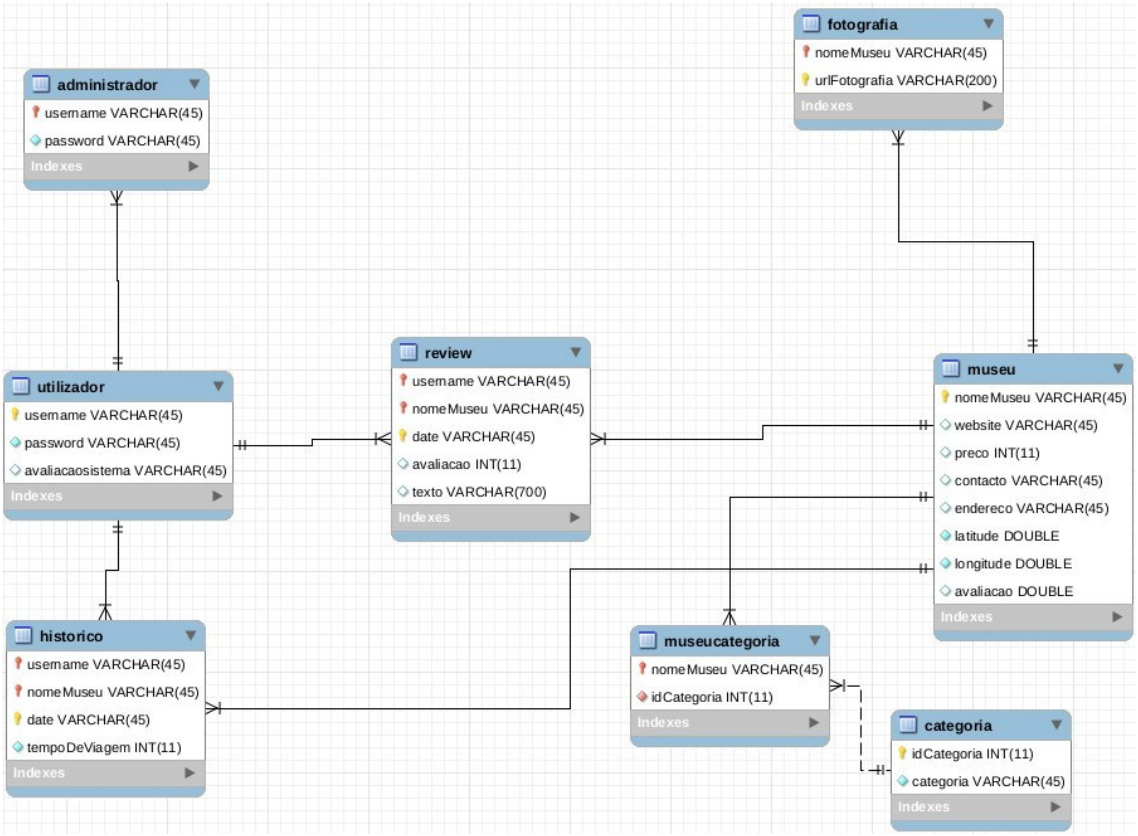

Figure 6: Estrutura da Base de Dados

- **Administrador**

  Entidade relativa aos administradores, que adiciona/remove museus e gere a aplicação em geral. Contém um *username* e uma *password* e, segundo a especificação, o administrador é também um utilizador, logo é utilizado o *username* como chave estrangeira.

- **Utilizador**

  Entidade que representa os utilizadores da aplicação. Contém *username*, que é a chave primária para identificar nas restantes tabelas, a *password* respetiva e avaliação do sistema. Outros elementos como histórico e comentários/avaliação do museu especificados nesta parte do relatório contêm a sua própria tabela, não estando contidos nesta.

- **Review**

  Esta tabela, na especificação recebida, está definida textualmente e no esquema da base de dados. As definições são contraditórias, portanto optamos por seguir o diagrama e juntar a avaliação com o comentário. Removemos, no entanto, o *idComentario* pois não achamos que este tenha qualquer utilidade. A tabela contém o *username* do utilizador que fez o comentário/avaliação, o *nomeMuseu* do Museu a que o utilizador se refere, a data em qual foi feita, a avaliação de 0 a 5 e o texto que contém o comentário.

- **Histórico**

  Entidade que contém a informação relativa aos museus que os utilizadores visitaram. De forma semelhante à tabela *Review* removemos o *idRegisto* e adicionamos o *username* do utilizador. Contém ainda a data, a duração da viagem e o *nomeMuseu* como definido na descrição textual da especificação recebida.

- **Museu**

  Entidade responsável por guardar a informação do museu, tal como descrito na especificação recebida. Contém o nome do museu, preço de entrada, contacto, localização, website, localização e avaliação, no entanto, não contém categoria e fotografias pois criamos duas outras tabelas para estes dois elementos.

- **Fotografia**

  Entidade que guarda as fotografias dos museus. Utilizamos o nome do museu para identificar a que museu a fotografia pertence e um URL da imagem para utilizarmos, similar à tabela *Review* optamos por remover o elemento *idFotografia*.

- **museuCategoria**

  Responsável por ligar as categorias aos museus. Contém o nome do museu e o id da categoria.

- **Categoria**

  Tabela onde são armazenadas todas as categorias suportadas pela aplicação. Contem o id da categoria e o nome da mesma.

3. Conclusões e Trabalho Futuro

Como grupo, concluímos que ficamos satisfeitos com o trabalho realizado, pois conseguimos implementar tudo aquilo que é pedido na especificação recebida. Um dos aspetos que podemos melhorar num futuro próximo é a abordagem que tomamos na implementação deste projeto. Uma abordagem iterativa e incremental de um sistema não pode ser aplicada em projetos de maiores dimensões.

Considerando agora a aplicação em si, existem alguns problemas, nomeadamente:

1. É uma aplicação facilmente substituída por outra (p.ex *Google Maps*) pois não apresenta nenhum fator de diferenciação suficientemente relevante.
2. Tem como foco apenas a cidade de Braga.

Uma solução futura pode partir da expansão do sistema para áreas maiores, mas nesse caso voltamos a encontrar outro problema, o facto de ser necessário a manutenção por parte de um grupo de administradores. Com isto queremos dizer que o sistema, assim como está definido, apresenta um problema de escalabilidade.

Para resolver o problema anterior é necessário construir um sistema dotado de uma manutenção dinâmica e descentralizada, podendo, por exemplo, utilizar *feedback* dos utilizadores para retirar trabalho aos administradores.

Apesar destes *problemas*, somos da opinião que o projeto foi de indispensável importância para a nossa formação académica, pois foi através deste que fomos introduzidos a novos conceitos, nomeadamente: estruturação de aplicações Web, criamos a nossa primeira base de dados, utilizamos uma API (REST API) e aprendemos a utilizar novas ferramentas como mySQL e Flutter. Partimos assim para projetos futuros com todas estas ideias em mente, fortalecendo a nossa experiência de alunos de Engenharia de Software.

Referências

MySQL, *MySQL Costumers*. Disponível em https://www.mysql.com/customers/.

IMB Technology, *What is a REST API?*. Disponível em https://youtu.be/lsMQRaeKNDk.

RedHat, What is a REST API? Disponível em https://www.redhat.com/en/topics/api/what-is-a-rest-api.

The Net Ninja, *Flutter Tutorial for Beginners*. Disponível em YouTube-Playlist.

Academind, *Flutter Crash Course*. Disponível em https://youtu.be/x0uinJvhNxI.

Flutter, *Flutter Documentation*. Disponível em https://docs.flutter.dev/.

Raja Yogan, Flutter Maps. Disponível em YouTube-Playlist.

Google, *Google Maps Platform*. Disponível em https://developers.google.com/maps.

Programming with Mosh, *MySQL Tutorial for Beginners [Full Course]*. Disponível em https://youtu.be/7S_tz1z_5bA.

FreeCodeCamp.org, *SQL Tutorial – Full Database Course for Beginners*. Disponível em https://youtu.be/HXV3zeQKqGY.

Anexos

Folha de cálculo com a apreciação da especifcação recebida.

Base de Dados.