UNIVERSIDADE DO MINHO
Mestrado Engenharia Informática

BusPrime

Arquiteturas Aplicacionais

Sistemas Interativos Confiáveis

Autores
Bruno Filipe de Sousa Dias PG47068
João Carlos Peixoto Freitas PG47850
Luis Filipe Cruz Sobral A89474
Rui Alexandre da Costa Gonçalves PG47630
Tiago Araújo Alves PG47686

17 de junho de 2022

Conteúdo

1 Introdução **1**
1.1 Motivação . . . 1
1.2 Contextualização . . . 1
1.3 Caraterização do utilizador . . . 1

2 Modelação **2**
2.1 Requisitos . . . 2
2.1.1 Funcionais . . . 2
2.1.2 Não Funcionais . . . 3
2.2 Diagrama de Use Cases . . . 4
2.3 Especificação de Use Cases . . . 6
2.3.1 Use Case: Registar . . . 6
2.3.2 Use Case: Iniciar sessão . . . 6
2.3.3 Use Case: Terminar sessão . . . 6
2.3.4 Use Case: Procurar percurso . . . 7
2.3.5 Use Case: Filtrar percurso . . . 7
2.3.6 Use Case: Comprar Viagem . . . 7
2.3.7 Use Case: Adicionar percurso favorito . . . 8
2.3.8 Use Case: Remover percurso favorito . . . 8
2.3.9 Use Case: Adicionar paragens favoritas . . . 8
2.3.10 Use Case: Consultar perguntas frequentes . . . 8
2.4 Diagrama de classes . . . 9
2.5 Protótipo da interface . . . 9
2.5.1 Protótipo - Página inicial . . . 10
2.5.2 Protótipo - Registar . . . 10
2.5.3 Protótipo - Login . . . 11
2.5.4 Protótipo - Página inicial logado . . . 11
2.5.5 Protótipo - Perfil utilizador . . . 12
2.5.6 Protótipo - Lista de Viagens . . . 12
2.5.7 Protótipo - Adicionar Percurso aos Favoritos . . . 13
2.5.8 Protótipo - Lista de Percursos Favoritos . . . 13
2.5.9 Protótipo - Lista de Paragens favoritas . . . 14
2.5.10 Protótipo - Adicionar Paragem aos Favoritos . . . 14

3 Implementação **15**
3.1 Povoamento . . . 15
3.1.1 Percursos intra-cidades . . . 15
3.1.2 Percursos inter-cidades . . . 15
3.2 Frontend . . . 15
3.2.1 Views . . . 15
3.2.2 Pedidos à API . . . 15
3.3 Backend . . . 15
3.3.1 Controllers (Servlets) . . . 16
3.3.2 Enterprise Java Beans . . . 16
3.3.3 Persistência . . . 16

4 Deployment **17**
4.1 Arquitetura Pretendida 17
4.1.1 Front-end 18
4.1.2 Back-end 18
4.1.3 Camada de Integração 18
4.2 Resultados 18

5 Análise de Carga **19**

6 Interface **20**
6.0.1 Interface - Home Page Visitante 20
6.0.2 Interface - Home Page 20
6.0.3 Interface - Home Page Continuação 21
6.0.4 Interface - Login 21
6.0.5 Interface - Login Fail 22
6.0.6 Interface - Login Erro 22
6.0.7 Interface - Registar 23
6.0.8 Interface - Registar Fail 23
6.0.9 Interface - Registar Erro 24
6.0.10 Interface - Perfil 24
6.0.11 Interface - Terminar Sessão 25
6.0.12 Interface - Carteira 25
6.0.13 Interface - Acrescentar Dinheiro 26
6.0.14 Interface - Acrescentar Dinheiro Fail 26
6.0.15 Interface - Bilhetes 27
6.0.16 Interface - Bilhetes Informação 27
6.0.17 Interface - Paragens Favoritas 28
6.0.18 Interface - Acrescentar Paragens Favoritas 28
6.0.19 Interface - Acrescentar Paragens Favoritas Fail 29
6.0.20 Interface - Eliminar Paragens Favoritas 29
6.0.21 Interface - Percursos Favoritos 30
6.0.22 Interface - Eliminar Percursos Favoritos 30
6.0.23 Interface - Go To Percurso Favorito 31
6.0.24 Interface - Consulta de Viagens 31
6.0.25 Interface - Consulta de Viagens Sem Percurso Favorito . 32
6.0.26 Interface - Adicionar Percursos Favoritos 32
6.0.27 Interface - Consulta de Viagens Visitante 33
6.0.28 Interface - Consulta de Viagens Filtro 33
6.0.29 Interface - Comprar Viagem 34
6.0.30 Interface - About Us 34
6.0.31 Interface - About Us Hover 35
6.0.32 Interface - Perguntas Frequentes 35

7 Análise de usabilidade **36**
7.1 Princípios de usabilidade 36
7.1.1 Learnability 36
7.1.2 Flexibility 36
7.1.3 Robustness 36

7.2 Heurísticas de Nielsen 36
7.2.1 Visibility of system status 37
7.2.2 Match between system and the real world 37
7.2.3 User control and freedom 37
7.2.4 Consistency and standards 37
7.2.5 Error prevention 37
7.2.6 Recognition rather than recall 37
7.2.7 Flexibility and efficiency of use 38
7.2.8 Aesthetic and minimalist design 38
7.2.9 Help users recognize and recover from errors 38
7.2.10 Help and documentation 38

8 Conclusão **39**

Lista de Figuras

1 Diagrama de use cases 5
2 Diagrama de Classes 9
3 Protótipo - Página inicial 10
4 Protótipo - Registar 10
5 Protótipo - Login 11
6 Protótipo - Página inicial logado 11
7 Protótipo - Perfil utilizador 12
8 Protótipo - Lista de Viagens 12
9 Protótipo - Adicionar Percurso aos Favoritos 13
10 13
11 Protótipo - Lista de Paragens favoritas 14
12 Protótipo - Adicionar Paragem aos Favoritos 14
13 Arquitetura proposta 17
14 Interface - Home Page Visitante 20
15 Interface - Home Page 20
16 Interface - Home Page Continuação 21
17 Interface - Login 21
18 Interface - Login Fail 22
19 Interface - Login Erro 22
20 Interface - Registar 23
21 Interface - Registar Fail 23
22 Interface - Registar Erro 24
23 Interface - Perfil 24
24 Interface - Terminar Sessão 25
25 Interface - Carteira 25
26 Interface - Acrescentar Dinheiro 26
27 Interface - Acrescentar Dinheiro Fail 26
28 Interface - Bilhetes 27
29 Interface - Bilhetes Informação 27
30 Interface - Paragens Favoritas 28
31 Interface - Acrescentar Paragens Favoritas 28
32 Interface - Acrescentar Paragens Favoritas Fail 29
33 Interface - Eliminar Paragens Favoritas 29
34 Interface - Percursos Favoritos 30
35 Interface - Eliminar Percursos Favoritos 30
36 Interface - Interface - Go To Percurso Favorito 31
37 Interface - Consulta de Viagens 31
38 Interface - Consulta de Viagens Sem Percurso Favorito 32
39 Interface - Adicionar Percursos Favoritos 32
40 Interface - Consulta de Viagens Visitante 33
41 Interface - Consulta de Viagens Visitante 33
42 Interface - Comprar Viagem 34
43 Interface - About Us 34
44 Interface - About Us Hover 35
45 Interface - Perguntas Frequentes 35

1 Introdução

1.1 Motivação

As redes de transportes são um elemento essencial em qualquer sistema económico pelo que é de extrema relevância que esta área acompanhe o desenvolvimento tecnológico.

Surgiu assim a ideia de criar uma aplicação web e mobile que permitisse visualizar e comprar bilhetes para qualquer viagem de autocarro disponibilizada por todas as empresas que disponibilizam este tipo de serviços. A única plataforma que disponibiliza algo deste género é o Google Maps, porém este serviço tem algumas limitações tais como não ser uma aplicação focada no transporte de autocarros, não disponibilizar preços e disponibilizar viagens de poucas empresas.

1.2 Contextualização

A PrimeBus é uma plataforma que permite comprar bilhetes para viagens de autocarro inter-cidades e intra-cidades, sendo que os utilizadores podem também adicionar percursos e paragens favoritas.

A aplicação tem as seguintes características principais:

- Os utilizadores podem registar-se, aceder ao seu perfil, editar perfil e depositar dinheiro na sua carteira da aplicação.
- Cada utilizador pode adicionar/remover percursos ou paragens favoritos e aceder à lista de bilhetes comprados.
- É possível visualizar e filtrar percursos e ordená-los por preços ou duração.
- O utilizador pode escolher uma viagem e comprar bilhetes.

1.3 Caraterização do utilizador

O publico alvo da aplicação são os utilizadores das redes de transportes públicos, estes têm a seguintes características:

1. Experiência em andar em transportes públicos
2. Mais de 16 anos

2 Modelação

A modelação tem um papel fundamental para o desenvolvimento de uma aplicação com todos os requisitos pretendidos. Esta permite visualizar o sistema a ser implementado e especificar a sua estrutura e comportamento. Nesta secção está presente a definição dos requisitos funcionais e não funcionais da aplicação, bem como uma demonstração do diagrama de use cases, o diagrama de classes desenvolvido, modelo de tarefas e um primeiro protótipo da interface da aplicação **Bus Prime**.

2.1 Requisitos

O levantamento de requisitos foi sobretudo obtido através de técnicas de *brainstorming*, que nos permitiu obter as principais funcionalidades da aplicação bem como algumas características não funcionais da aplicação.

2.1.1 Funcionais

Os requisitos funcionais dizem respeito a todas as funcionalidades ou pressupostos de como o sistema deve reagir a entradas específicas e como se deverá comportar. Estes tipos de requisitos podem-se entender como funcionalidades que o utilizador pode interagir de forma direta e mais visível.

De seguida é apresentada a lista dos requisitos funcionais levantados pelo grupo.

- Registar
- Iniciar sessão
- Terminar sessão
- Editar perfil
- Procurar percurso
- Filtrar percurso
- Adicionar percurso favorito
- Remover percurso favorito
- Listar percursos favoritos
- Adicionar paragens favoritas
- Remover paragens favoritas
- Listar paragens favoritas
- Depositar dinheiro
- Listar paragens existentes
- Comprar bilhetes

- Observar histórico de bilhetes comprados
- Efetuar pergunta
- Responder pergunta

2.1.2 Não Funcionais

Requisitos não funcionais são as propriedades comportamentais relacionadas ao uso da aplicação em termos de desempenho, usabilidade, segurança, disponibilidade, manutenção e tecnologias envolvidas. Estes requisitos dizem respeito a como as funcionalidades serão entregues ao utilizador do software.

De seguida são apresentados os requisitos não funcionais levantados pela equipa.

Aparência

Os requisitos de aparência referem-se ao aspeto visual e à estética do sistema, ou seja, ao interface gráfico.

- Logótipo apelativo
- Predominância de 2 cores
- O produto deve ser esclarecedor e atrativo

Usabilidade

Os requisitos de usabilidade, descrevem a facilidade de utilização do sistema, ou seja, a experiência do utilizador.

- A aplicação deverá ser fácil de usar por todos os utilizadores do sistema
- A aplicação deve ser de fácil uso para novos utilizadores
- A aplicação deverá dar um feedback claro sobre os acontecimentos

Performance

Os requisitos de desempenho têm a ver com aspetos de velocidade, tempo real, capacidade de armazenamento, entre outros.

- O Sistema deverá responder de forma instantânea (menos de meio segundo)
- Percursos sempre atualizados
- A data e hora dos percursos deve ser precisa
- A aplicação deverá funcionar 24 horas por dia
- Não deverão haver perdas de dados
- A aplicação deverá permitir vários utilizadores simultâneos
- A aplicação deverá suportar 1000 utilizadores

- Com o aumento da popularidade da nossa aplicação a mesma deverá aumentar o número de utilizadores concorrentes que ela suporta

Operacionais
Os requisitos operacionais descrevem as características do sistema que permitem que o sistema funcione corretamente.

- A aplicação poderá ser acedida em qualquer dispositivo que tenha acesso à internet
- A aplicação poderá ser acedida em qualquer browser

Manutenção e suporte
Os requisitos de manutenção e suporte podem ser vistos como atributos que permitem a reparação ou a melhoria de funcionalidades do sistema.

- A aplicação deverá ser atualizada 1 vez por mês
- Erros detectados serão corrigidos no espaço de 12 horas
- Serão adicionadas novas viagens disponíveis diariamente
- Atualização da BD em tempo real conforme novos utilizadores/viagens sejam registados

Segurança
Os requisitos de segurança descrevem questões relacionadas com o acesso, confidencialidade, proteção e integridade dos dados.

- Cada utilizador tem de possuir um username exclusivo
- Dados de um utilizador são guardados em BD
- Todas as palavras passes serão guardadas encriptadas para promover a segurança das mesmas

2.2 Diagrama de Use Cases

O processo de desenvolvimento do **Diagrama de Use Cases** apresenta, em conformidade com os requisitos funcionais da aplicação, a distribuição das funcionalidades que cada ator terá na aplicação **Bus Prime**.

O **User** que assumimos como um utilizador que já se encontra com dados de registo na nossa aplicação, tem como funcionalidades **Iniciar/Terminar Sessão** bem como a possibilidade de **Editar Perfil**. Como seria de esperar, o User pode efetuar **Pesquisas** por percurso, bem como utilizar mecanismos de **Filtragem** para facilitar a pesquisa. Um utilizador registado e devidamente autenticado poderá ainda **Adicionar Percursos** e Paragens Favoritas na aplicação, bem como a sua remoção. Adicionalmente, um User poderá consultar uma lista de **Perguntas Frequentes**, onde poderá também Efetuar Pergunta.

Um outro ator identificado na aplicação é o Guest, que é considerado como um utilizador sem dados de sessão autenticado na plataforma. As funcionalidades disponíveis para este tipo de utilizador são mais restringidas, sendo apenas possível fazer a **Pesquisa** e **Filtragem** por percursos, e ainda **Consultar Perguntas Frequentes**.

O último ator identificado é o **Admin** que seria responsável por Responder Perguntas realizadas pelos **User**.

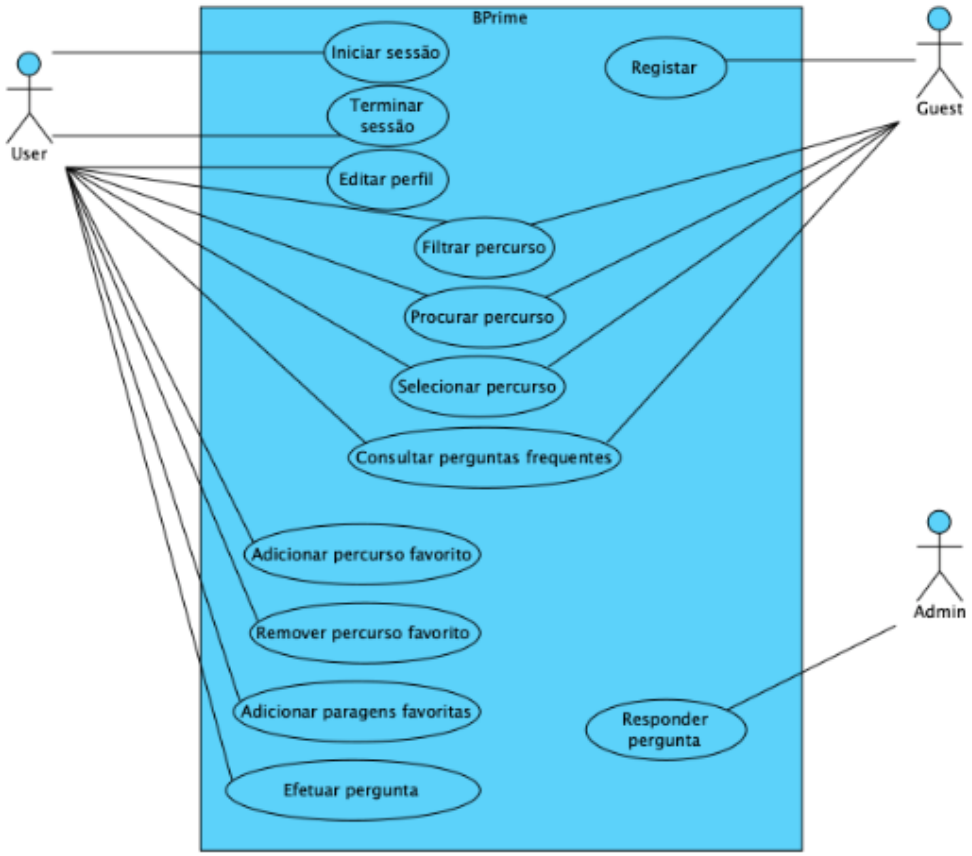

Figura 1: Diagrama de use cases

2.3 Especificação de Use Cases

2.3.1 Use Case: Registar

Descrição: Guest regista a sua conta na aplicação
Ator: Guest
Pré-Condição: Utilizador encontra-se com a aplicação iniciada
Pós-Condição: Utilizador encontra-se registado no Sistema
Fluxo Normal:
1. Utilizador indica que se quer registar
2. Sistema solicita credenciais de acesso (e-mail, nome, password)
3. Utilizador preenche os campos pedidos
4. Utilizador submete campos preenchidos
5. Sistema verifica validade das credenciais
6. Sistema indica que o Utilizador se encontra registado

Fluxo Alternativo 1: [Dados preenchidos encontram-se incompletos] (passo 4)
4.1. Sistema indica que faltam preencher campos obrigatórios
4.2. Volta para 2

Fluxo Alternativo 2: [Dados preenchidos são inválidos] (passo 4)
4.1. Sistema indica que credenciais preenchidas são inválidas
4.2. Volta para 4

2.3.2 Use Case: Iniciar sessão

Descrição: Guest efetua o seu login
Ator: Guest
Pré-Condição: Utilizador encontra-se com a aplicação iniciada
Pós-Condição: Utilizador encontra-se com sessão iniciada
Fluxo Normal:
1. Utilizador indica que se quer iniciar sessão
2. Sistema solicita credenciais de acesso (nome e password)
3. Utilizador preenche os campos pedidos
4. Utilizador submete campos preenchidos
5. Sistema verifica validade das credenciais
6. Sistema indica que o Utilizador se encontra com sessão iniciada

Fluxo Alternativo 1: [Dados preenchidos encontram-se incompletos] (passo 4)
4.1. Sistema indica que faltam preencher campos obrigatórios
4.2. Volta para 2

Fluxo Alternativo 2: [Dados preenchidos são inválidos] (passo 4)
4.1. Sistema indica que credenciais preenchidas são inválidas
4.2. Volta para 4

2.3.3 Use Case: Terminar sessão

Descrição: O utilizador termina sessão na aplicação
Ator: User
Pré-Condição: User encontra-se com sessão iniciada

Pós-Condição: A sessão é terminada
Fluxo Normal:
1. O utilizador solicita o término da sessão
2. O sistema confirma o pedido de término de sessão

2.3.4 Use Case: Procurar percurso

Descrição: Utilizador procura por um percurso que pretende efetuar
Ator: User e Guest
Pré-Condição: Utilizador encontra-se na página de procura de percurso
Pós-Condição: Utilizador encontra o percurso que deseja
Fluxo Normal:
1. Utilizador indica paragem de autocarros de partida
2. Utilizador indica paragem de autocarros de chegada
3. Utilizador insere dia de partida
4. Utilizador indica hora de partida
5. Utilizador clica no botão de procura
6. Sistema procura por percursos correspondentes e disponíveis
7. Sistema apresenta percursos encontrados

2.3.5 Use Case: Filtrar percurso

Descrição: Utilizador coloca filtros de pesquisa
Ator: User e Guest
Pré-Condição: Utilizador encontra-se na página onde estão apresentados os percursos
Pós-Condição: Página onde estão apresentados os percursos encontra-se filtrada
Fluxo Normal:
1. Sistema apresenta percursos encontrados da pesquisa feita
2. Utilizador clica no botão de filtro que pretende
3. Sistema apresenta página com os percursos encontrados de forma filtrada

2.3.6 Use Case: Comprar Viagem

Descrição: Utilizador seleciona percurso que pretende efetuar
Ator: User
Pré-Condição: Utilizador encontra-se na página onde estão apresentadas as viagens
Pós-Condição: Utilizador compra viagem
Fluxo Normal:
1. Sistema apresenta percursos encontrados da pesquisa feita

2. Utilizador clica no botão de comprar de uma viagem
3. Utilizador aceita comprar bilhete para viagem
4. Sistema regista compra de bilhete
5. Sistema apresenta novamente percursos encontrados da pesquisa feita

2.3.7 Use Case: Adicionar percurso favorito

Descrição: Utilizador adiciona um percurso como favorito
Ator: User
Pré-Condição: Utilizador tem sessão iniciada
Pós-Condição: Percurso é adicionado como favorito
Fluxo Normal:

1. Sistema apresenta percursos encontrados da pesquisa feita
2. Utilizador clica no botão de adicionar percurso como favorito
3. Utilizador aceita adicionar percurso como favorito
4. Sistema atualiza a lista de favoritos do Utilizador

2.3.8 Use Case: Remover percurso favorito

Descrição: Utilizador remove um percurso da sua lista de favoritos
Ator: User
Pré-Condição: Utilizador tem sessão iniciada
Pós-Condição: Percurso é removido dos favoritos do utilizador
Fluxo Normal:

1. Utilizador seleciona acesso à sua lista de favoritos
2. Sistema apresenta lista de favoritos o Utilizador
3. Utilizador seleciona um percurso que quer remover
4. Utilizador aceita remover percurso como favorito
5. Sistema atualiza a lista de favoritos do Utilizador

2.3.9 Use Case: Adicionar paragens favoritas

Descrição: Utilizador adiciona uma paragem favorita
Ator: User
Pré-Condição: Utilizador encontra-se na página das paragens favoritas
Pós-Condição: Paragem é adicionada à lista de paragens favoritas
Fluxo Normal:

1. Sistema apresenta lista de paragens favoritas
2. Utilizador seleciona o botão para adicionar paragens favoritas
3. Sistema apresenta a página de adição de paragens favoritas
4. Utilizador insere a paragem que quer introduzir
5. Sistema adiciona a paragem à lista de paragens favoritas do Utilizador

2.3.10 Use Case: Consultar perguntas frequentes

Descrição: Consultar perguntas frequentes
Ator: User e Guest
Pré-Condição: A aplicação está iniciada
Pós-Condição: O utilizador/Guest acede à àrea de Perguntas Frequentes

Fluxo Normal:

1. O utilizador/Guest acede à àrea das Perguntas Frequentes
2. Sistema disponibiliza a lista de perguntas frequentes

2.4 Diagrama de classes

O diagrama de classes é um dos diagrama mais importantes e permite ter uma ideia da arquitetura da aplicação e da forma como os diferentes componentes interagem entre si. Neste diagrama podemos também perceber quais a classes a ser implementadas e quais destas devem ser persistidas na base de dados.

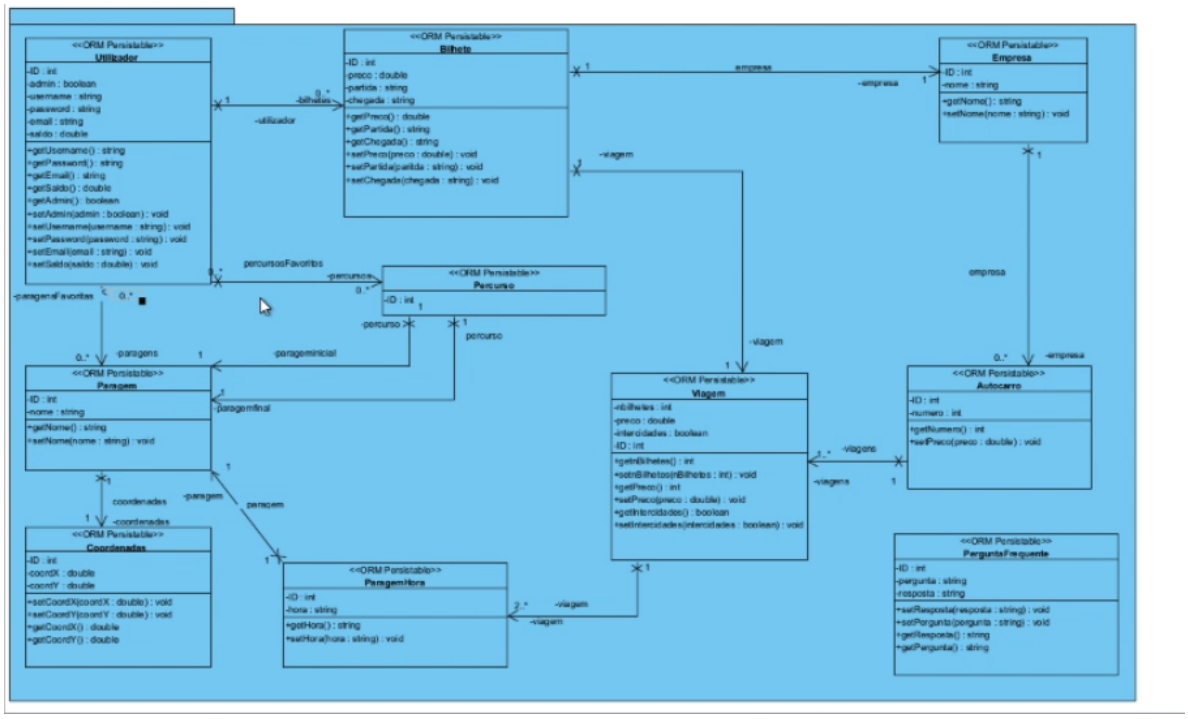

Figura 2: Diagrama de Classes

2.5 Protótipo da interface

Na fase de modelação, elaboramos ainda um protótipo de baixa fidelidade da interface gráfica da aplicação Bus Prime, com recurso à ferramenta *Pencil*.

Nesta fase inicial de prototipagem, conseguimos ter uma noção de como as informações e as funcionalidades seriam apresentadas para que quando for altura de construir a interface gráfica da aplicação esta seja desenvolvida com o máximo grau de usabilidade possível.

De seguida, serão apresentadas as *mockup's* desenvolvidas.

2.5.1 Protótipo - Página inicial

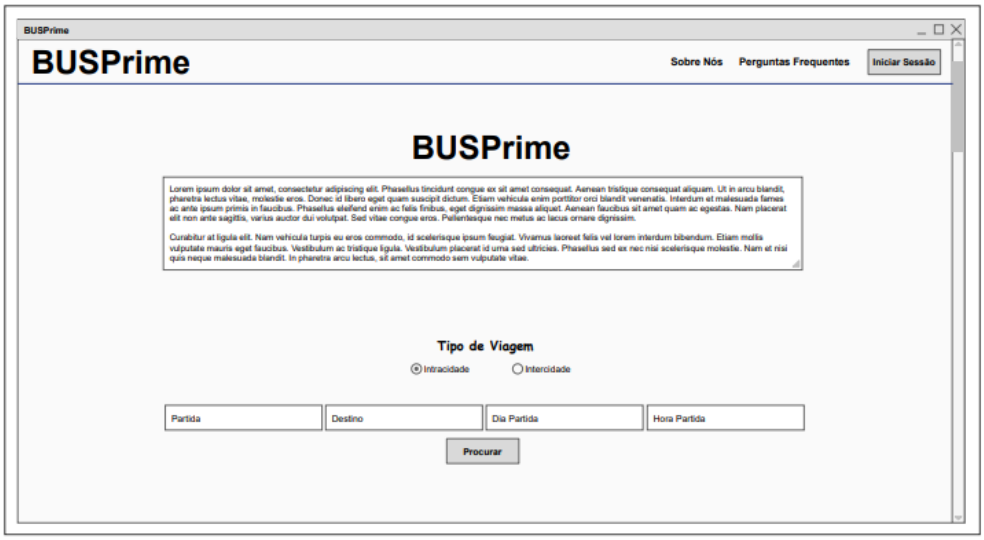

Figura 3: Protótipo - Página inicial

2.5.2 Protótipo - Registar

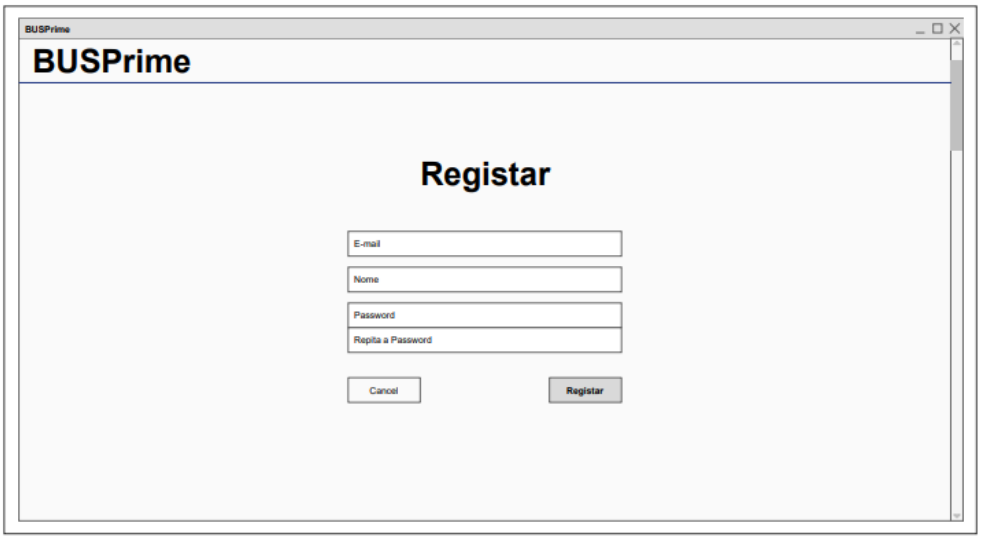

Figura 4: Protótipo - Registar

2.5.3 Protótipo - Login

Figura 5: Protótipo - Login

2.5.4 Protótipo - Página inicial logado

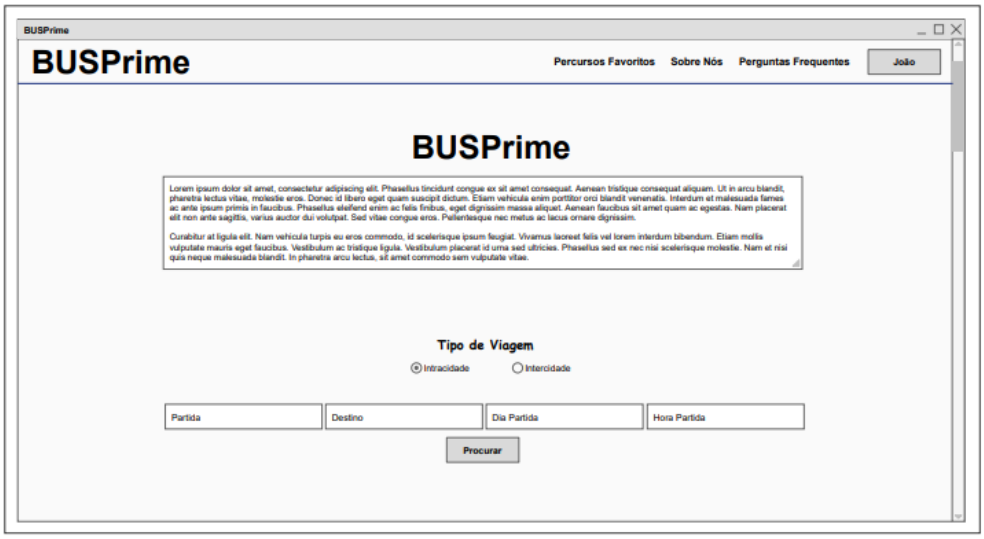

Figura 6: Protótipo - Página inicial logado

2.5.5 Protótipo - Perfil utilizador

Figura 7: Protótipo - Perfil utilizador

2.5.6 Protótipo - Lista de Viagens

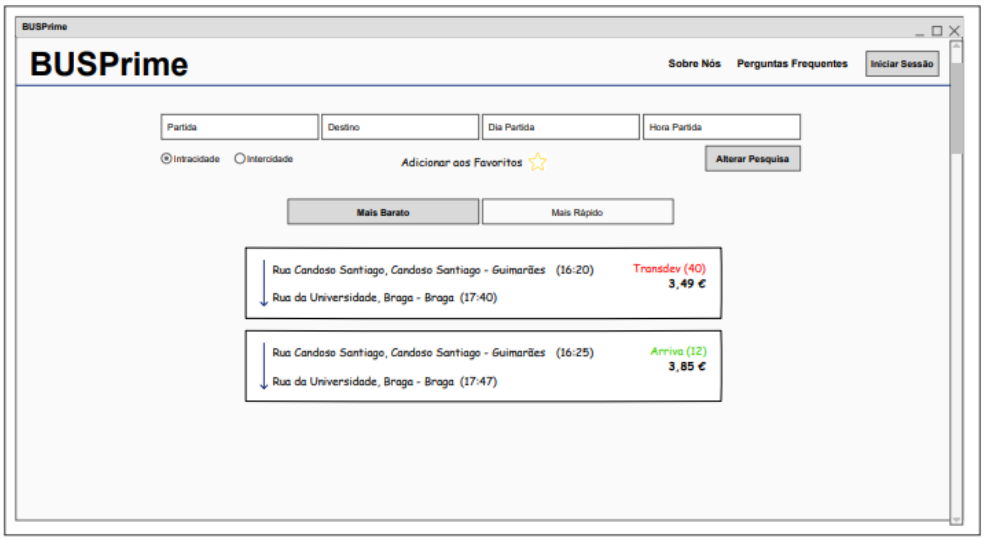

Figura 8: Protótipo - Lista de Viagens

2.5.7 Protótipo - Adicionar Percurso aos Favoritos

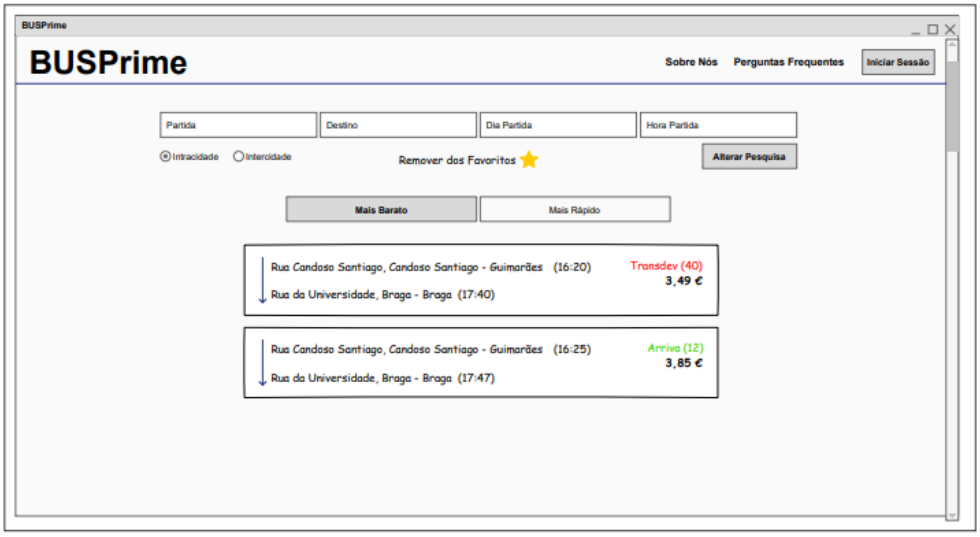

Figura 9: Protótipo - Adicionar Percurso aos Favoritos

2.5.8 Protótipo - Lista de Percursos Favoritos

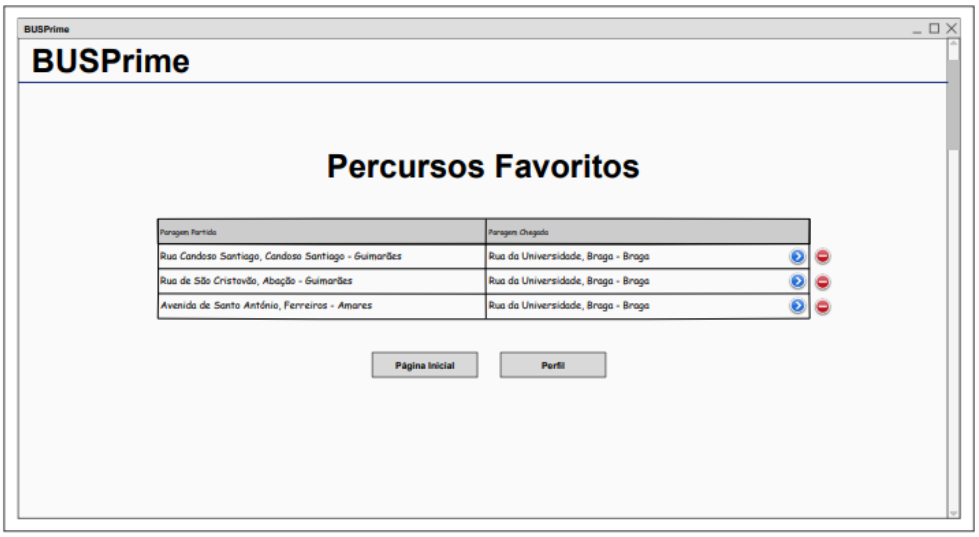

Figura 10

2.5.9 Protótipo - Lista de Paragens favoritas

Figura 11: Protótipo - Lista de Paragens favoritas

2.5.10 Protótipo - Adicionar Paragem aos Favoritos

Figura 12: Protótipo - Adicionar Paragem aos Favoritos

3 Implementação

3.1 Povoamento

3.1.1 Percursos intra-cidades

De modo a povoar a base de dados com uma vasta quantidade de percursos criamos um script em python que permite inserir na base de dados todos os autocarros, circuitos e paragens da TUB (transportes urbanos braga). Este script lê os ficheiros disponibilizados pela TUB e cria o ficheiro insert.sql com todo o código sql necessária para inserir os dados na base de dados.

3.1.2 Percursos inter-cidades

Relativamente a percursos inter-cidade recolhemos percursos de serviços disponibilizados por várias empresas (FlixBus,Rede Expressos).

3.2 Frontend

Para o desenvolvimento da camada da aplicação usamos **Vue.js**. Esta camada é independente das outras pelo que recorre a pedidos à API para obter os dados que necessita de apresentar.

3.2.1 Views

Na realização deste projeto construímos um total de 11 *views* nos quais estão contidas todas as informações necessárias para mostrar ao Utilizador. Para além destas *views* foram criadas ainda duas componentes para a *Navbar*, uma delas para a *navbar* da *Home*, que é interativa com o *scroll* e ainda uma para todas as outras páginas, que será estática.

O Utilizador não logado possui ainda a possibilidade de consultar viagens e usufruir de algumas funcionalidades do sistema sem precisar de estar com a sessão iniciada. Desta forma, estas views adaptam-se conforme o Utilizador está com sessão iniciada ou não.

3.2.2 Pedidos à API

Para efetuar os pedidos à API recorremos ao *package* **axios**, que nos permite definir o *endpoint* requerido, configurar os *headers*, o tipo de método a utilizar (POST, GET,...) e receber assim a resposta (dados) por parte da API.

3.3 Backend

O back-end controla o fluxo entre os pedidos vindos do front-end e que exigem acessos à camada da persistência. Atua assim, como um servidor API REST.

De forma a criar servidores de forma eficaz e robusta, algo bastante relevante para a criação de uma aplicação com as características pretendidas, foi usada a *framework* **Spring** para Java. Para aceder e mapear os objetos da base de dados foi utilizada a *framework* **Hibernate**.

3.3.1 Controllers (Servlets)

Os controllers são web servlets que atuam sobre rotas REST definidas para cada um. Um servlet é uma classe programada em Java que permite expandir as capacidades de servidores que gerem aplicações que são acedidas através de um modelo request-response. Os servlets contribuem também para a escalabilidade da aplicação, uma vez que usam uma thread por pedido e não por cliente.

Cada uma das rotas realiza operações que necessitam de acesso à base de dados. Este processo devido à criação de inúmeras threads rapidamente esgotaria o número de sessões permitidas pelo hibernate, pelo que é necessário a utilização de Enterprise Java Beans.

Caso o pedido seja bem sucedido é retornado ao front-end em forma de JSON.

3.3.2 Enterprise Java Beans

Os Enterprise Java Beans, permitem a utilização de um conjunto finito de objetos (Beans) evitando que os objetos sejam instanciados desnecessariamente e controlo de concorrência.

Na aplicação apenas foram criados Beans locais (não são utilizados serviços remotos) e stateless, devido às suas inúmeras vantagens. Foram criados assim os seguintes Beans:

- **ParagemBean**: gere operações que envolvem a listagem de paragens
- **PerguntasFreqBean**: gere operações sobre as perguntas frequentes
- **UtilizadorBean**: gere todas as operações relativas aos utilizadores (registar, editar, adicionar/remover/listar percursos favoritos, adicionar/-remover/listar paragens favoritas, comprar bilhete)
- **ViagemBean**: gere operações sobre viagens

3.3.3 Persistência

A *framework* **Hibernate** permite mapear de forma mais simplificada objetos de uma base de dados relacional. Esta gera chamadas SQL e liberta o utilizador de gerir manualmente e converter objetos de um conjunto de resultados.

O mapeamento disponibilizado pelo hibernate foi gerado utilizando Visual Paradigm e permite a utilização de JPA de forma mais simples e direta, tal como a manipulação direta de objetos em memória.

A base de dados relacional utilizada foi o **MySQL** devido à sua adequação à aplicação.

4 Deployment

Depois de termos toda a nossa aplicação completamente funcional no servidor local passamos então ao passo seguinte que seria o *deployment* da mesma para que ela pudesse ser utilizada pelos vários Utilizadores. Assim, começamos por definir a arquitetura que queríamos que a aplicação tivesse.

4.1 Arquitetura Pretendida

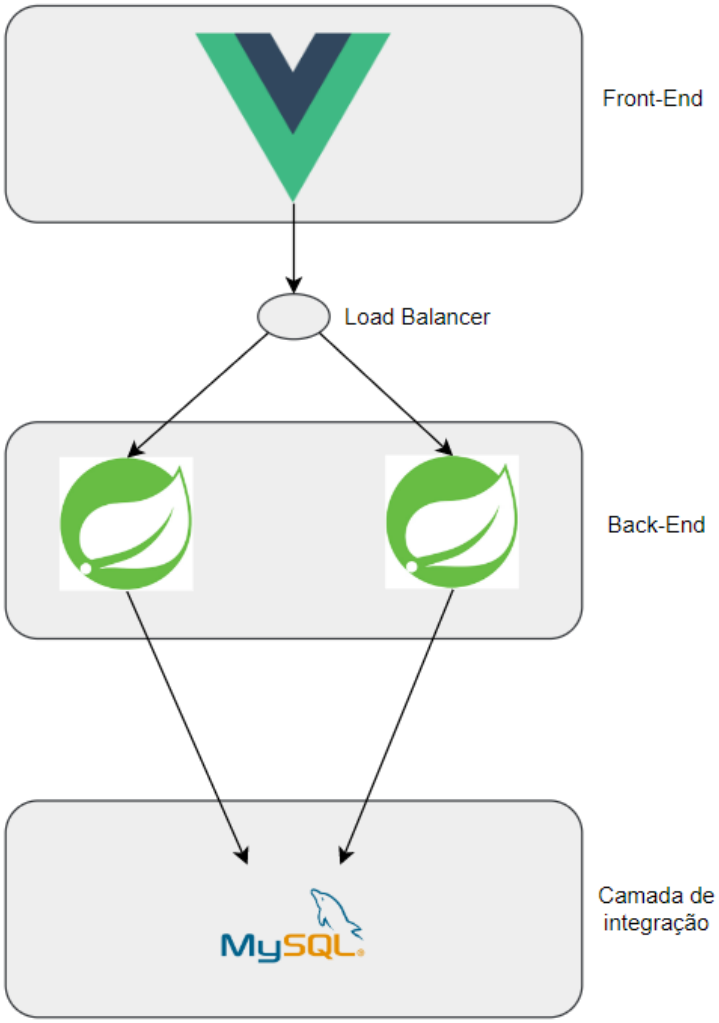

Figura 13: Arquitetura proposta

Assim, propomos utilizar um servidor de *front-End* que o Utilizador irá utilizar e que estará encarregue por mandar pedidos à API, seguido por dois servidores de *Back-End* para que os pedidos possam ser atendidos mais eficientemente (é importante referir que utilizaremos um *load balancer* para garantir que os pedidos são distribuídos corretamente pelos dois servidores de *back-end*

de forma a balancear a carga), e que depois utilizaremos uma única base de dados (pois as bases de dados SQL não possuem escalabilidade horizontal).

4.1.1 Front-end

Para fazermos o deployment do nosso servidor de front-end utilizamos a ferramenta ***Vercel*** que é especializada em servidores de Vue.js, fornecendo-nos então imediatamente um servidor operacional e eficiente. Este servidor pode ser acedido por todos os utilizadores através do url: `https://bus-prime-frontend.vercel.app/`.

4.1.2 Back-end

Para implementar o back-end optamos então por utilizar a ferramenta docker através da qual conseguiríamos criar três *containers* independentes (o load balancer e os dois servidores). Para que pudessemos utilizar esta ferramenta tivemos de primeiramente gerar o executável do servidor para que este pudesse ser incluído no dockerFile de cada servidor. Optamos também por utilizar o NGINX como balanceador de carga.

4.1.3 Camada de Integração

Para a camada de integração optamos por utilizar uma instância da *Google Cloud SQL*, na qual corremos o ficheiro que era gerado pela base de dados (o ficheiro que criava as tabelas e as suas relações), obtendo então uma base de dados capaz de responder a todos os pedidos que a API fizesse.

4.2 Resultados

Infelizmente o processo de deployment não foi concluído com sucesso, embora tenhamos conseguido que o servidor de front-end e a base de dados fossem deployed, o executável *.jar* do servidor de back-end não conseguia correr com êxito por causa de problemas de dependências, originando então um obstáculo no deployment que não conseguimos superar. Consideramos no entanto que, caso esse problema tivesse sido resolvido o deployment teria sido efetuado com sucesso e que a aplicação já estaria disponível neste momento para que os Clientes a pudessem utilizar.

5 Análise de Carga

Para avaliar a carga que a nossa aplicação é capaz de aguentar precisamos inicialmente de determinar qual a operação que mais carga adicionará. Como o foco da nossa aplicação é fornecer ao Utilizador a capacidade de comprar um bilhete para um determinado percurso que quer fazer, este será o principal foco no nosso teste (sendo esta a operação mais custosa).

A compra de um bilhete envolve vários pedidos, sendo estes:

- O login na conta (pois não pode comprar se não se encontrar logado);
- A obtenção das paragens que existem na nossa base de dados (para que estas possam ser disponibilizadas para o Utilizador selecionar o percurso);
- A procura das viagens para o percurso que pretende;
- A compra do bilhete.

Utilizaremos então *J-Meter*, uma *framework* que permite realizar testes de *performance* para simular simular a utilização simultânea do número de utilizadores que pretendermos, com a possibilidade de gerar e apresentar gráficos, relatórios estatísticos. De seguida apresentamos os resultados obtidos:

Clientes	**Average**	**Min**	**Max**	**Std.Dev**	**Error %**	**Throughput**
100	941	308	1503	265.77	0.00	61.8
200	3655	2660	4479	441.21	0.00	44.1
300	4454	3399	5263	413.09	0.00	56.7
500	2784	274	3771	528.42	2.60	122.3
1000	4101	106	5569	981.38	15.50	166.9

Tabela 1: Tabela de Resultados - Análise de Carga

Observando os resultados conseguimos concluir que a nossa aplicação não apresenta problemas quando o número de clientes é inferior a 300 mas que observa uma queda de performance para valores superiores a estes (500), e que, quando temos mais de 1000 clientes o erro é superior a 10%, sendo então este erro demasiado grande para que a aplicação possa apresentar um bom comportamento. Concluímos então que a carga máxima que a nossa aplicação aguenta (não danificando muito a sua performance) estará entre 500 e 800 clientes, apresentando comportamento ideal entre 100 a 300 clientes.

6 Interface

6.0.1 Interface - Home Page Visitante

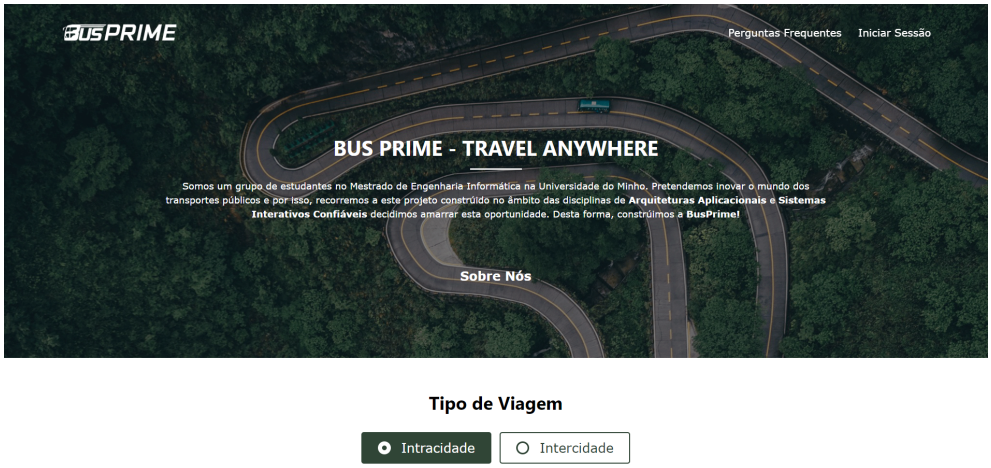

Figura 14: Interface - Home Page Visitante

6.0.2 Interface - Home Page

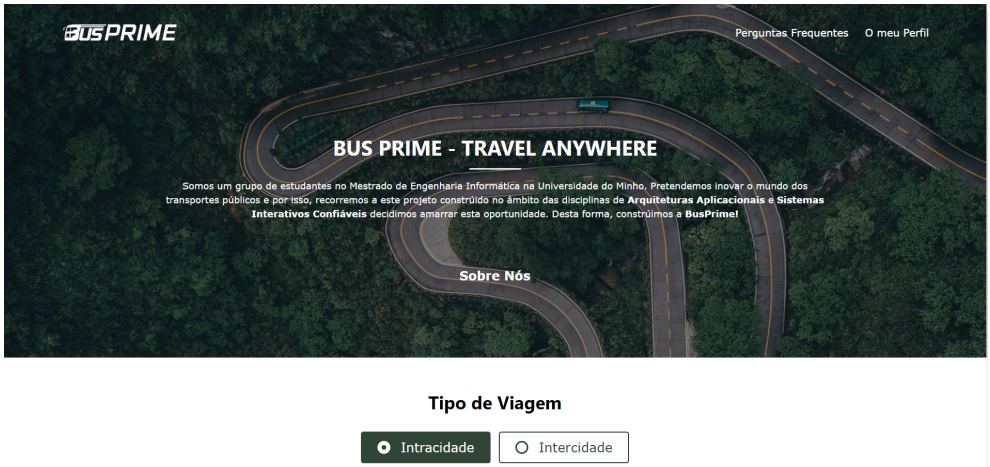

Figura 15: Interface - Home Page

6.0.3 Interface - Home Page Continuação

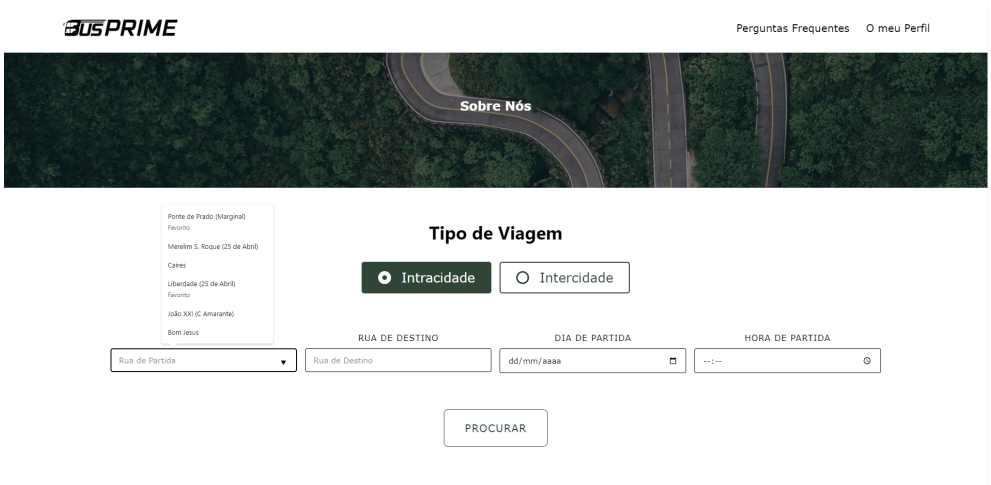

Figura 16: Interface - Home Page Continuação

6.0.4 Interface - Login

Figura 17: Interface - Login

6.0.5 Interface - Login Fail

Figura 18: Interface - Login Fail

6.0.6 Interface - Login Erro

Figura 19: Interface - Login Erro

6.0.7 Interface - Registar

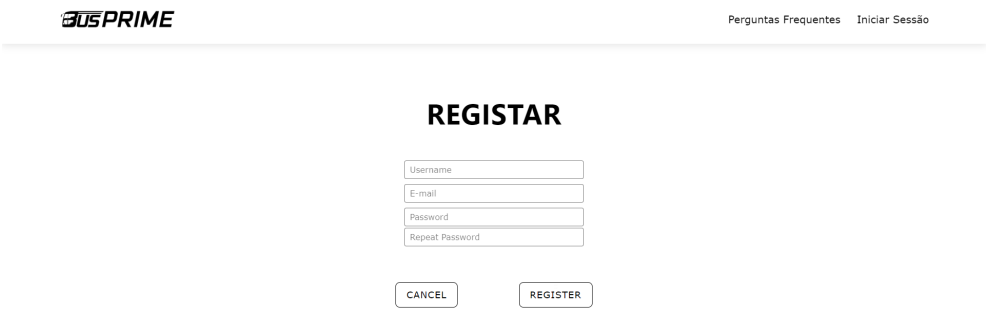

Figura 20: Interface - Registar

6.0.8 Interface - Registar Fail

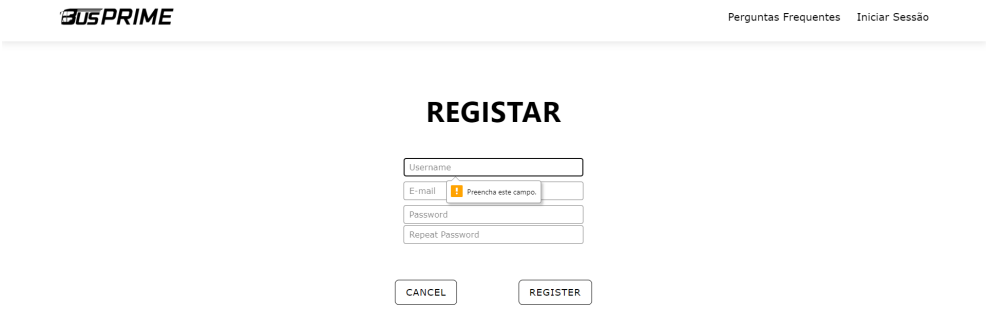

Figura 21: Interface - Registar Fail

6.0.9 Interface - Registar Erro

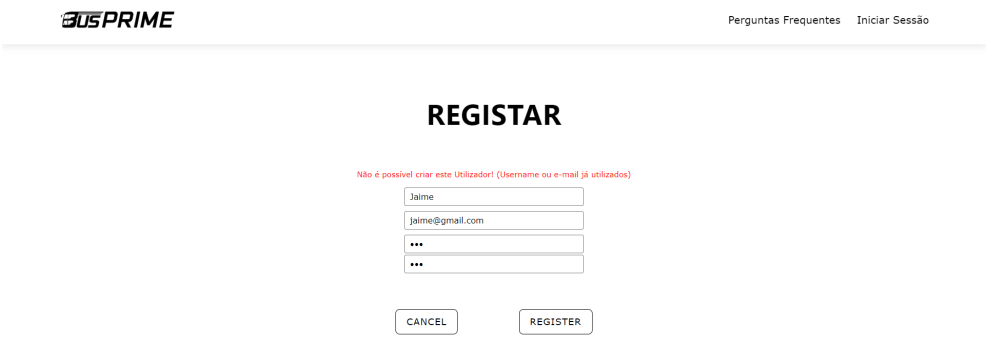

Figura 22: Interface - Registar Erro

6.0.10 Interface - Perfil

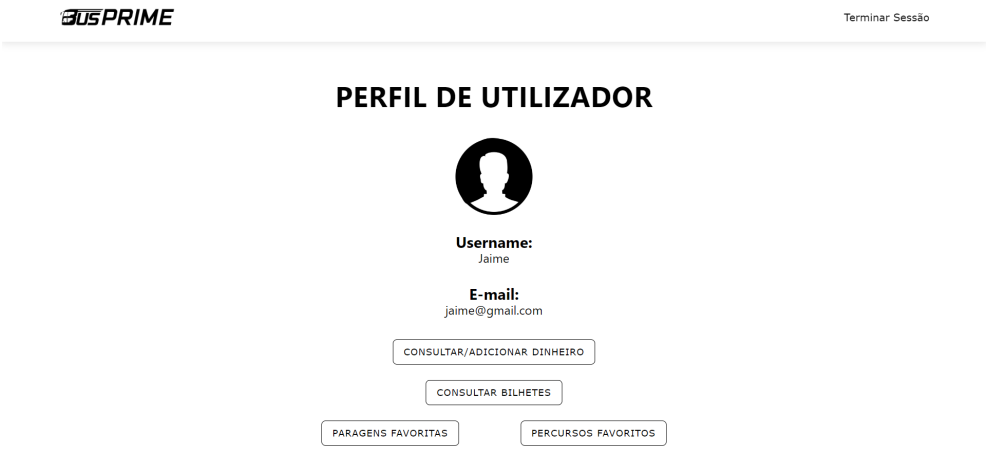

Figura 23: Interface - Perfil

6.0.11 Interface - Terminar Sessão

Figura 24: Interface - Terminar Sessão

6.0.12 Interface - Carteira

Figura 25: Interface - Carteira

6.0.13 Interface - Acrescentar Dinheiro

Figura 26: Interface - Acrescentar Dinheiro

6.0.14 Interface - Acrescentar Dinheiro Fail

Figura 27: Interface - Acrescentar Dinheiro Fail

6.0.15 Interface - Bilhetes

Figura 28: Interface - Bilhetes

6.0.16 Interface - Bilhetes Informação

Figura 29: Interface - Bilhetes Informação

6.0.17 Interface - Paragens Favoritas

Figura 30: Interface - Paragens Favoritas

6.0.18 Interface - Acrescentar Paragens Favoritas

Figura 31: Interface - Acrescentar Paragens Favoritas

6.0.19 Interface - Acrescentar Paragens Favoritas Fail

Figura 32: Interface - Acrescentar Paragens Favoritas Fail

6.0.20 Interface - Eliminar Paragens Favoritas

Figura 33: Interface - Eliminar Paragens Favoritas

6.0.21 Interface - Percursos Favoritos

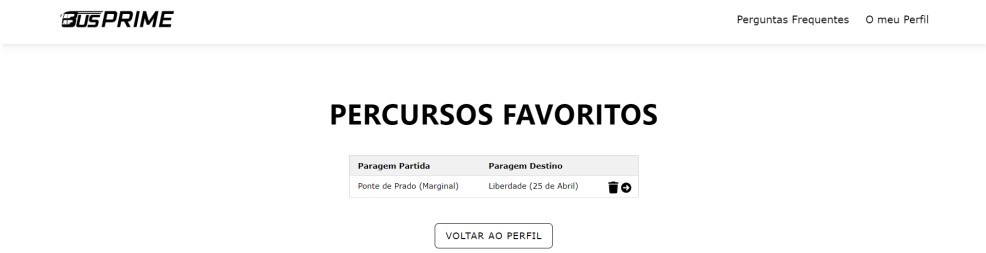

Figura 34: Interface - Percursos Favoritos

6.0.22 Interface - Eliminar Percursos Favoritos

Figura 35: Interface - Eliminar Percursos Favoritos

6.0.23 Interface - Go To Percurso Favorito

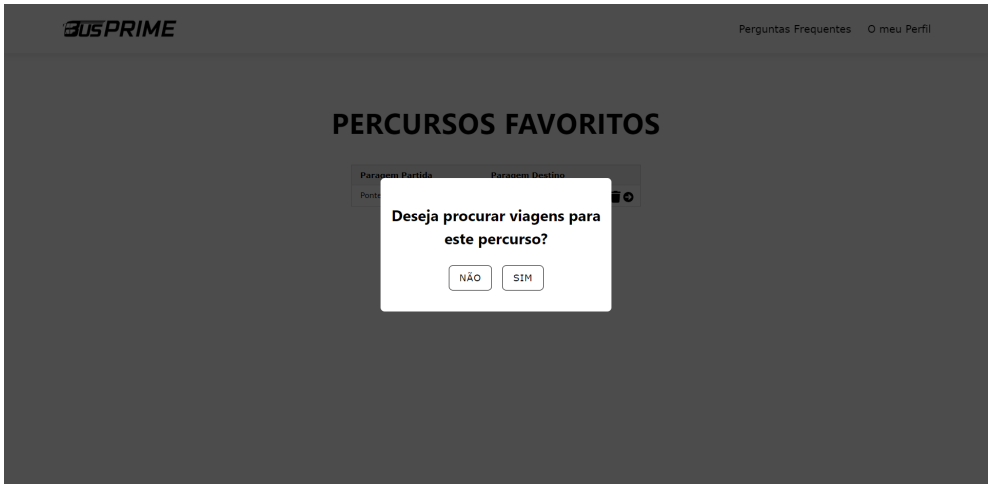

Figura 36: Interface - Interface - Go To Percurso Favorito

6.0.24 Interface - Consulta de Viagens

Figura 37: Interface - Consulta de Viagens

6.0.25 Interface - Consulta de Viagens Sem Percurso Favorito

Figura 38: Interface - Consulta de Viagens Sem Percurso Favorito

6.0.26 Interface - Adicionar Percursos Favoritos

Figura 39: Interface - Adicionar Percursos Favoritos

6.0.27 Interface - Consulta de Viagens Visitante

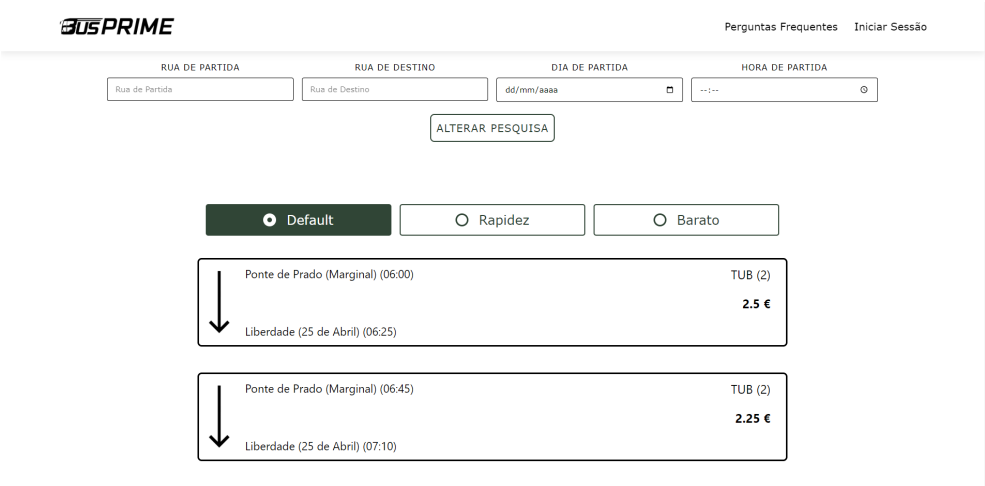

Figura 40: Interface - Consulta de Viagens Visitante

6.0.28 Interface - Consulta de Viagens Filtro

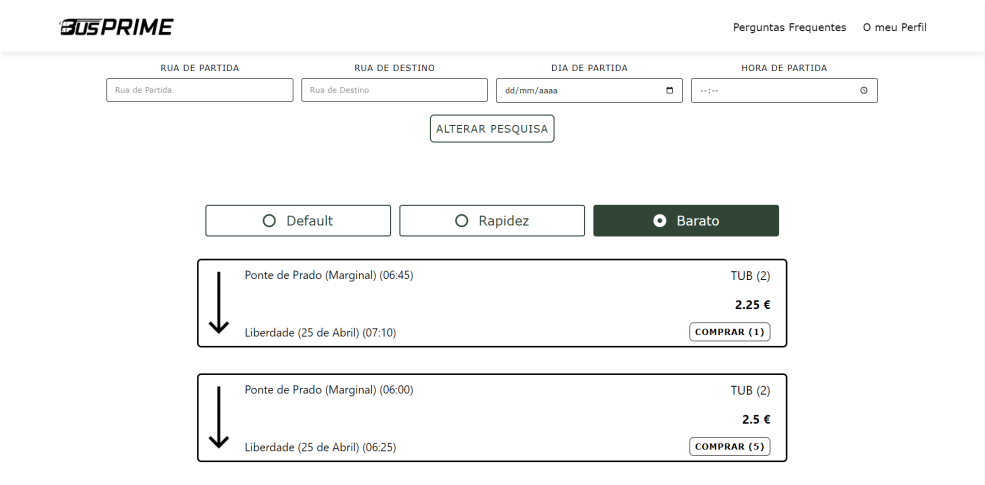

Figura 41: Interface - Consulta de Viagens Visitante

6.0.29 Interface - Comprar Viagem

Figura 42: Interface - Comprar Viagem

6.0.30 Interface - About Us

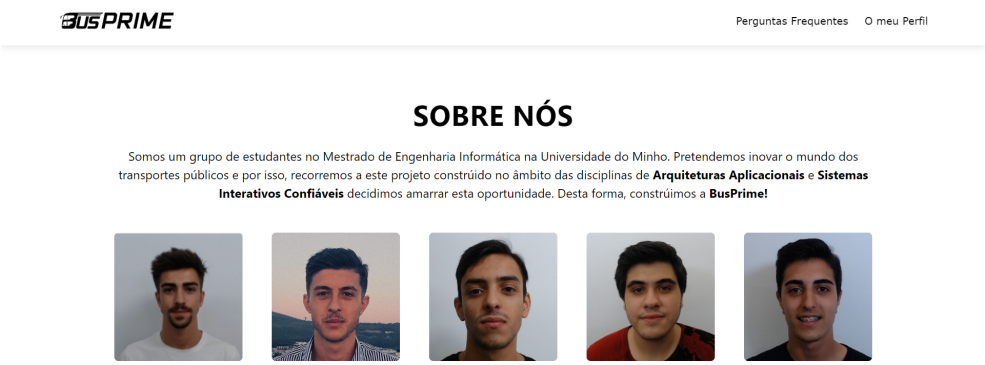

Figura 43: Interface - About Us

6.0.31 Interface - About Us Hover

Figura 44: Interface - About Us Hover

6.0.32 Interface - Perguntas Frequentes

Figura 45: Interface - Perguntas Frequentes

7 Análise de usabilidade

7.1 Princípios de usabilidade

Os princípios de usabilidade definem a usabilidade de uma peça de software e tem como objetivo proporcionar o máximo de usabilidade possível aos utilizadores.

Estes, devem ser tidos em conta desde o início do design da interface gráfica e devem ser revisitados várias vezes ao longo do processo de desenvolvimento.

7.1.1 Learnability

Este princípio diz respeito à facilidade com que novos utilizadores podem iniciar uma interação eficaz e alcançar máxima performance. Este princípio é divisível em outros 5: **Predictability**,**Synthesizability**,**Familiarity**, **Generalizability**,**Consistency**.

Em relação à **Predictability**, a aplicação tem campos como escolher rua de partida e rua destino da viagem que basicamente são *inputs* de um *datalist* possível de paragens registadas na aplicação. No caso da **Familiarity** temos a data e hora das viagens apresentados em *Date/Hour picker*, um formato bastante familiar a novos utilizadores.

7.1.2 Flexibility

Este princípio diz respeito à multiplicidade de formas com que o utilizador e o sistema trocam mensagens. Este princípio divide-se em cinco subcategorias: **Dialogue initiative**,**Multithreading**,**Task migratability**,**Substitutivity**, **Customizability**. Relativamente à **Customizability** o utilizador pode adicionar/remover percursos ou paragens favoritas.

7.1.3 Robustness

Este princípio diz respeito ao nível de suporte fornecido ao utilizador na determinação do êxito e na avaliação do comportamento direcionado a objetivos. Este princípio de usabilidade divide-se em: **Observability**,**Recoverability**, **Responsiveness**,**Task conformance**.

A **Observability** pode ser comprovada por todas a páginas identificarem claramente o seu propósito. Relativamente a **Recoverability** a interface disponibiliza mensagens de erro e mensagens de confirmação.

7.2 Heurísticas de Nielsen

As heurísticas de Nielsen, são um método analítico que nos ajuda a perceber qual o grau de usabilidade de uma interface gráfica. Tratam-se assim 10 heurísticas para se criar uma boa interface. De salientar que estas heurísticas foram revisitadas várias vezes ao longo do processo de design e construção da interface.

Neste momento, chegamos à solução final e, por isso, procederemos em seguida à avaliação sumativa da solução desenvolvida. Desta forma, vamos

visitar cada uma das heurísticas de forma a avaliar se estão a ser cumpridas, ou se efetivamente se adequam à solução desenvolvida.

7.2.1 Visibility of system status

Esta heurística diz respeito à visibilidade do estado do sistema e se este mantém o utilizador informado sobre o mesmo.

Relativamente a esta heurística antes de comprar um bilhete é possível observar o número de bilhetes disponível (Figura 37.

7.2.2 Match between system and the real world

Nesta heurística procura-se garantir que existe uma correspondência entre o sistema e o mundo real.

A nossa interface cumpre esta heurística uma vez que a aplicação se baseia no funcionamento de redes de autocarros reais e permite a compra de bilhetes para viagens reais. A inserção da data e hora das viagens cumpre também este requisito.

7.2.3 User control and freedom

Esta heurística tenta garantir que o utilizador se sinta livre e em controlo ao usar a interface. Desta forma, é importante que se forneça saídas, *undo* e *redo*, e que se guie o utilizador pela navegação da interface sem ele praticamente se aperceber.

Neste aspeto, a interface disponibiliza mensagens de confirmação (figura 24, 33, 35 , 36, 39 e 42). O utilizador tem também a possibilidade de remover/adicionar percursos e paragens favoritas.

7.2.4 Consistency and standards

Ao construir uma aplicação web é importante seguir os standards e ser consistente, uma vez que os utilizadores já estão habituados a diversos tipos de abordagem. Desta forma, a aplicação disponibiliza operações de login, logout e registo que são bastante frequentes e também disponibiliza um *qr-code* para cada bilhete comprado, algo baste popular atualmente.

7.2.5 Error prevention

Antes de permitir que o utilizador cometa erros, é importante prevenir erros para que este não tenha de que repetir a ação novamente.

De modo a prevenir erros são apresentadas mensagens a indicar o que é pretendido em cada campo, como podemos ver nas figuras 18, 21 e 27.

7.2.6 Recognition rather than recall

Nesta heurística, pretende-se que o utilizador se lembre das coisas porque as vê e reconhece. Assim, o utilizador não deve ter de memorizar informação de uma interface para outra.

Esta heurística é cumprida, por exemplo, quando o utilizador escolhe um percurso favoritos e o percurso escolhido volta a ser listado juntamente com os preço do mesmo ao quando o utilizador escolhe uma das paragens favoritas.

7.2.7 Flexibility and efficiency of use

A flexibilidade e eficiência é outro princípio a ter em conta. É importante que os utilizadores consigam acelerar a interação, para isso, a interface deve fornecer atalhos com acesso direto aos itens mais utilizados.

A interface permite comprar bilhetes mais rapidamente selecionando uma viagem da lista de viagens frequentes. Ao pesquisar por uma viagem é também possível escolher a rua de acordo com as rua que dão *match* com as letras já inseridas tornando este processo mais eficiente.

7.2.8 Aesthetic and minimalist design

Uma interface gráfica deve ter uma boa estética e deve ser minimalista. Assim, os diálogos não devem conter informação irrelevante e a informação deve ser apresentada de forma simples e compreensível.

As interfaces foram construídas seguindo sempre este princípio, tendo a predominância de 2 cores e apresentando sempre toda a informação relevante e de forma simples.

7.2.9 Help users recognize and recover from errors

Esta heurística diz respeito à possibilidade de os utilizadores reconhecerem e recuperarem dos erros. As mensagem devem ser simples e indicar o problema ocorrido.

Como podemos ver, as figuras 19, 22 e 32 são exemplos onde as mensagens de erro identificam o erro ocorrido e são simples.

7.2.10 Help and documentation

Por fim, a aplicação *web* deve ainda oferecer ajuda e documentação para guiar as pessoas para o sítio certo.

O grupo não considerou diretamente nenhum plano para esta heurística, uma vez que a aplicação **Bus Prime** é bastante simples e tem um número limitado de funcionalidades, não sentimos a necessidade de fornecer um manual para auxiliar a utilização da mesma.

8 Conclusão

Neste trabalho foi-nos proposto desenvolver uma aplicação tendo em conta todos os passos do seu desenvolvimento e tendo como foco o utilizador. No inicio do trabalho, começamos por definir qual seria o principal utilizador do nosso sistema bem como todas as principais funcionalidades que este deverá ter disponível. Modelamos também como deveriam ser executadas as interações do utilizador com o sistema para que posteriormente, fosse só programar as mesmas em vez de termos de pensar como as mesmas se fariam. Desenvolvemos também um protótipo que nos permitiu focar nos objetivos na aplicação e também pensar como a interface deveria ser de modo a que fosse intuitiva para o utilizador. Optamos também por realizar um diagrama de classes para que depois, através deste, conseguíssemos gerar a base de dados e o código *hibernate* que seria capaz de aceder à mesma.

Após termos o código que acedia à base de dados passamos então para a implementação do *back-end* e do *front-end*. Neste passo, começamos por implementar o *back-end* tendo em mente todos os pedidos aos quais a API deveria responder (usando o código *hibernate* sempre que precisávamos de alterar/aceder à base de dados). Passamos depois à programação do HTML e CSS do *front-end*, permitindo então, ajustar também o *back-end* pois já tínhamos em mente os pedidos que deveriam receber. Nesta fase de codificação, a principal dificuldade foi em juntar a interface e o *back-end* (operação que fizemos utilizando *vue.js*). A *framework vue.js* era bastante complexa e apresentava um mecanismo de erros difícil de interpretar e de encontrar os erros que iríamos cometendo ao longo da codificação.

Depois de implementarmos localmente a aplicação, passamos então ao *deployment*. Esta foi sem duvida a parte mais difícil de todo o trabalho, uma vez que encontramos múltiplos erros de dependências ao gerar o ficheiro executável, erros estes que não encontrávamos quando corríamos localmente (pois já tínhamos certas dependências instaladas que não estavam no ficheiro *pom-xml*). Esta foi também uma fase que não conseguimos executar completamente. Embora tenhamos conseguido dar *deployment* do *front-end* e da base de dados, não o conseguimos para o *back-end* resultando então numa falha do *deployment* da própria aplicação.

Posteriormente avaliamos também a carga que a nossa aplicação conseguia suportar enquanto mantinha o seu bom comportamento para sabermos quantos utilizadores a mesma conseguiria aguentar caso fosse lançada neste preciso momento. Neste teste tivemos em mente a operação mais custosa e que seria mais utilizada pelos utilizadores (a operação de comprar o bilhete) e todas as operações que ele precisava de executar anteriormente, para estimar a percentagem de erro que iria acontecer quando aumentávamos o número de clientes simultâneos.

Por fim, realizamos também uma avaliação da interface que tínhamos implementado para referir em que aspetos esta respeitava os princípios de usabilidade e as heurísticas e tinha como fator principal o Utilizador, de modo a que mesmo um Utilizador inexperiente conseguisse usufruir da nossa aplicação imediatamente.

Concluímos então que conseguimos atingir a maior parte dos objetivos deste trabalho adquirindo conhecimento mais aprofundado acerca de todos os assuntos abordados no trabalho.