



**Universidade do Minho**  
Escola de Engenharia  
Licenciatura em Engenharia Informática

## **Unidade Curricular de Laboratórios de Informática IV**

Ano Letivo de 2021/2022

### **FunTracker**

Alexandre Flores A93220   Pedro Alves A93272   Mariana Rodrigues A93229  
Matilde Bravo A93246

11 de abril de 2022

**L | 4**

Data de Recepção	
Responsável	
Avaliação	
Observações	

# FunTracker

Alexandre Flores A93220   Pedro Alves A93272   Mariana Rodrigues A93229   Matilde  
Bravo A93246

11 de abril de 2022

## Resumo

Durante esta fase do trabalho foi realizada a implementação de uma proposta de aplicação de um Guia para estabelecimentos noturnos, de acordo com a especificação fornecida. A sua conceção teve sempre em vista o cumprimento dos requisitos funcionais e não funcionais da especificação em questão. Contudo, consideramos que alguns destes se encontravam incompletos ou não poderiam ser implementados sem comprometer a *User Experience*, sendo que quaisquer alterações estão evidenciadas no presente relatório, bem como no de apreciação. Durante a fase de elaboração do relatório de apreciação, tivemos a oportunidade de analisar minuciosamente a especificação da aplicação que viríamos a implementar, a qual se designa por *FunTracker*. Uma vez familiarizados com o produto a desenvolver, procedemos à escolha das ferramentas e linguagens mais apropriadas para o desenvolvimento de uma aplicação *crossplatform*. Este processo de decisão, bem como o desenvolvimento *per se*, foi uma excelente oportunidade para adquirir mais conhecimento no desenvolvimento de uma aplicação de início ao fim. Assim, durante a sua realização, pudemos aprender a utilizar ferramentas de *frontend* e *backend*. Além disso, também tivemos oportunidade de aprender um pouco mais sobre base de dados, área sobre a qual ainda não tivemos formação.

**Área de Aplicação:** Serviço de sugestão de locais noturnos

**Palavras-Chave:** Locais Noturnos, React Native, Node.js, TypesScript, SQLite

# Índice

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Enquadramento geral . . . . .	1
1.2	Caracterização do trabalho a desenvolver e seus objetivos . . . . .	1
1.3	Identificação dos recursos utilizados . . . . .	1
1.4	Plano de desenvolvimento adotado . . . . .	3
<b>2</b>	<b>Desenvolvimento da Aplicação</b>	<b>4</b>
2.1	Estratégia e método de desenvolvimento adotados . . . . .	4
2.2	Identificação e caracterização da aplicação e dos seus serviços . . . . .	4
2.3	Arquitetura geral . . . . .	6
2.3.1	Servidor . . . . .	6
2.3.2	Cliente . . . . .	7
2.4	Definição e caracterização de cada um dos componentes . . . . .	7
2.4.1	Backend . . . . .	7
2.4.2	FunTracker . . . . .	8
2.4.3	Express . . . . .	12
2.4.4	Frontend . . . . .	14
2.4.5	Base de dados . . . . .	15
2.5	Amostras do funcionamento da Aplicação . . . . .	17
<b>3</b>	<b>Conclusões</b>	<b>24</b>
<b>Lista de Siglas e Acrónimos</b>		<b>26</b>
<b>Anexos</b>		<b>27</b>
Apreciação realizada sobre a especificação recebida . . . . .		28

# **Lista de Figuras**

1.1	Diagrama de Gantt . . . . .	3
2.1	Diagrama da arquitetura cliente-servidor . . . . .	7
2.2	Diagrama de Componentes . . . . .	8
2.3	Diagrama de Classes interface IUserDAO . . . . .	9
2.4	Diagrama de Classes User . . . . .	9
2.5	Diagrama de Classes interface IClassificacaoDAO . . . . .	9
2.6	Diagrama de Classes Classificacao . . . . .	10
2.7	Diagrama de Classes interface IEstabelecimentoDAO . . . . .	10
2.8	Diagrama de Classes Estabelecimento . . . . .	10
2.9	Diagrama de Classes interface IIImagensDAO . . . . .	11
2.10	Diagrama de Classes Imagem . . . . .	11
2.11	Diagrama de Componentes . . . . .	14
2.12	Classe AuthContext . . . . .	15
2.13	Modelo da base de dados . . . . .	16
2.14	Login . . . . .	17
2.15	Sign Up . . . . .	17
2.16	Visualizar estabelecimentos . . . . .	18
2.17	Visualizar menu de <i>pop-up</i> . . . . .	18
2.18	Visualizar estabelecimento . . . . .	19
2.19	Visualizar estabelecimento . . . . .	19
2.20	Avaliar estabelecimento . . . . .	20
2.21	Visualizar histórico de avaliações . . . . .	20
2.22	Filtrar estabelecimentos . . . . .	21
2.23	Alterar dados de utilizador . . . . .	21
2.24	Adicionar estabelecimento . . . . .	22
2.25	Adicionar estabelecimento . . . . .	22
2.26	Menu <i>pop-up</i> de um utilizador especial . . . . .	23
2.27	Remover estabelecimento . . . . .	23
3.1	Apreciação realizada sobre a especificação recebida . . . . .	28

# **Lista de Tabelas**

2.1 API User . . . . .	12
2.2 API estabelecimento . . . . .	13
2.3 API Session . . . . .	14

# 1 Introdução

## 1.1 Enquadramento geral

Tendo sempre por base a especificação recebida, a aplicação de *smartphone* a desenvolver visa "*reunir todos os pontos de interesse relacionados com entretenimento noturno em Braga, como bares e discotecas*", permitindo "*proporcionar aos seus utilizadores um rápido acesso a uma lista com todos os locais de interesse perto de si*".

É apresentado como um sistema de auxílio de procura e navegação de estabelecimentos de diversão noturna intitulado **FunTracker**.

## 1.2 Caracterização do trabalho a desenvolver e seus objetivos

Foi proposta a realização de uma aplicação móvel, *FunTracker*, cujo o objetivo é encontrar locais noturnos de interesse ao utilizador. O utilizador deverá poder indicar o critério de preferência na escolha de estabelecimentos e a aplicação deverá apresentá-los de acordo com a mesma. Por exemplo, o utilizador pode pedir para ver os locais ordenados de acordo com a sua avaliação e apenas procurar aqueles que se encontram abertos num dado momento. Além disso, o utilizador poderá registar-se, autenticar-se, especificar a sua localização, avaliar um estabelecimento, bem como visualizar o seu histórico de avaliações. Perante estas especificações foi necessária a implementação de uma forma de registo e autenticação, bem como a implementação de um sistema de filtros.

Durante a análise da especificação, reparámos na falta de indicação de como um estabelecimento é adicionado ou removido. Uma vez que se trata de uma funcionalidade que consideramos fulcral, optámos por conceber um segundo tipo de utilizador, designado por administrador, que terá permissão para adicionar e remover estabelecimentos noturnos.

## 1.3 Identificação dos recursos utilizados

Apesar de apenas ter sido pedida a implementação de uma aplicação *mobile*, decidimos desenvolvê-la de forma a ser *crossplatform*, funcionando, desta forma, em *Android*, *IOS* e até mesmo num *Web Browser*.

- **TypeScript:** A linguagem utilizada para o desenvolvimento da aplicação, que é compilada para *Javascript*. Possui todas as suas funcionalidades com a adição de tipos de dados, o que torna o código mais legível e facilita a resolução de erros que poderiam surgir da ausência do uso de tipos.

- **React Native:** *Framework* e principal ferramenta usada na implementação do *frontend*. Mostrou-se adequada pela sua excelente documentação e facilidade de desenvolvimento, tendo assim uma curva de aprendizagem amigável. Permitiu minizar o tempo de programação da interface. Fornece também uma vasta gama de componentes e ferramentas de design de aplicações desenvolvidas por outros membros e grupos da comunidade, como por exemplo a *framework React Native Elements* que fornece muitos dos componentes visuais utilizados neste projeto. Além disso, esta ferramenta permite que a aplicação funcione tanto em *Android* como *iOS* e em *Web browser*.
- **Node.js:** *Runtime* utilizado para executar código *JavaScript* no *backend* sem a utilização de um *browser*. Facilita o desenvolvimento devido a permitir utilizar a mesma linguagem que é utilizada para o *frontend*.
- **Express.js:** Esta *framework* de **Node.js** é minimalista e robusta, fornecendo todas as ferramentas necessárias para o desenvolvimento de uma *REST API*.
- **SQLite:** Base de dados relacional utilizada para o armazenamento. A sua utilização deve-se principalmente devido ao facto de ser *open-source*, simples de usar e ser suportada na maior parte das plataformas. O facto de ser uma base de dados *lightweight* de fácil integração e não precisar de um servidor externo (existindo apenas como uma biblioteca) foi outro ponto a favor da sua escolha, pois permitiu-nos focar no desenvolvimento da aplicação sem comprometer na qualidade da base de dados.
- **Expo Go:** Este ambiente de desenvolvimento **mobile** foi crucial no decorrer do desenvolvimento da aplicação, permitindo testá-la nos nossos telemóveis bem como no *browser*. Esta ferramenta permite visualizar em tempo real o efeito de uma qualquer alteração efetuada no código (*Fast Refresh*), possibilitando uma maior rapidez na etapa obtenção de resultados e um eficiente design da interface do lado do *frontend*.
- **GitHub e Git:** De modo a facilitar o desenvolvimento de todo o código, sentiu-se a necessidade de se criar um repositório para permitir a colaboração entre toda a equipa.

No que toca a recursos humanos foram necessários quatro programadores. Dois deles ficaram responsáveis pela parte do *frontend*, enquanto que os restantes dois ficaram responsáveis pelo *backend*.

## **1.4 Plano de desenvolvimento adotado**

Tendo realizada uma divisão da nossa equipa nos dois grupos previamente descritos, passámos a uma etapa de delimitação de períodos alocados para tarefas específicas. Esta divisão tem como objetivo dar ao grupo associado a uma dada tarefa tempo suficiente para a concluir com tempo de sobra. Desta forma, se uma dada tarefa demorar mais do que se espera (dentro de limites razoáveis), o plano de desenvolvimento poderá ser seguido na mesma. Para ilustrar tal alocacão de período, concebemos o seguinte diagrama de Gantt.

FUNTRACKER

Grupo 3



Figura 1.1: Diagrama de Gantt

# 2 Desenvolvimento da Aplicação

## 2.1 Estratégia e método de desenvolvimento adotados

Como anteriormente mencionado, para o desenvolvimento da aplicação foram criadas duas equipas, uma para *frontend* e outra para *backend*, cada uma constituída por dois elementos. Perante as funcionalidades a implementar e as especificações fornecidas, acordou-se no tipo de informação a apresentar e armazenar, de modo a permitir que a posterior integração fosse *seamless*.

Ambas as equipas efetuaram o desenvolvimento em paralelo e de forma independente uma da outra, permitindo acelerar significativamente o processo de desenvolvimento. A equipa de *backend* desenvolveu a **API REST**, bem como a forma como os dados são armazenados, enquanto a equipa de *frontend* procurava desenvolver uma forma de apresentar a informação fornecida pelo *backend*. Após a conclusão do desenvolvimento do *backend* procedeu-se à integração que meramente consistiu na adição de requests à *API* desenvolvida, ao invés de utilizar valores *hardcoded*. Durante este processo foram feitos ajustes quer à parte visual, por não serem 100% coerentes com a especificação do programa, quer à parte do *backend* por não se enquadrar à ideia transmitida pela parte visual. Contudo, de forma geral, a integração foi simples e não foram necessárias demasiadas alterações em nenhuma das frentes.

Por fim, procedeu-se à testagem da aplicação, tentando experimentar cada uma das funcionalidades e corrigindo erros que surgiram.

## 2.2 Identificação e caracterização da aplicação e dos seus serviços

Após uma análise cuidadosa da especificação fornecida, notámos que haviam algumas incoerências. Com isso, foi necessário realizar algumas alterações:

- **Adicionar nome aos estabelecimentos**

Aqui é de salientar que nos *mockups* apresentados, no menu de apresentação de um estabelecimento este apresentava um **nome**. No entanto, em mais lado nenhum da especificação é referenciado que um estabelecimento tem como atributo o seu nome.

Perante isso, foi crucial adicionar o atributo **nome** ao estabelecimento, fazendo as devidas alterações à base de dados.

- **Renomear características para categorias**

Ao longo da especificação foi notado que em certos elementos eram referenciados como tendo **características** e outros como **categorias**. Com isto, para ser possível manter a coerência da aplicação foi decidido trocar todas as referências a **características** para **categorias**.

- **Acrescentar uma nova tabela à base de dados, intitulada categorias**

Devido ao facto de que cada **estabelecimento** pode ter mais do que uma **categoria**, foi crucial adicionar à base de dados uma forma de conseguir associar as diferentes **categorias** aos diferentes **estabelecimentos**.

Para tal, foi adicionado uma nova tabela intitulada como **categorias**. Esta contém duas colunas: uma referente aos **ids dos estabelecimentos** e outra ao **nome da categoria**.

É de notar que na especificação dada, a única referência às características na base de dados fornecida é uma coluna intitulada como **Características** na tabela de **Estabelecimento Noturno**, sendo este um **INT**, o que não nos permitia associar mais do que uma **categoria** a um qualquer **estabelecimento**.

- **Adicionar um estabelecimento**

Não se encontra previamente especificado como seriam adicionados os **estabelecimentos** à base de dados, se estes seriam constantes, ou se seria possível ir adicionando ao longo do tempo.

Perante isto, e de modo a manter a aplicação atualizada ao passar do tempo, determinou-se que seria possível adicionar novos **estabelecimentos** à aplicação. Esta funcionalidade adicional será apenas acessível a um utilizador especial (administradores).

- **Adicionar uma utilizador especial (Administrador)**

Para possibilitar a adição de estabelecimentos foi crucial a criação de um tipo de utilizador especial intitulado **admin**. Consoante isso, foi necessário adicionar mais um campo que nos possibilita-se determinar se um dado **utilizador** era **administrador** ou não à tabela de **utilizadores** da base de dados. Por predefinição, a base de dados é iniciada com um utilizador administrar com *username* e *password* "**admin**", de forma a facilitar testagem.

- **Remover um estabelecimento**

Dada a possibilidade de adicionar um **estabelecimento**, decidimos permitir a remoção de um mesmo. Tal como a funcionalidade de adicionar, esta apenas estará disponível aos utilizadores especiais.

- **Realizar o logout**

Funcionalidade extra implementada que permite um dado utilizador autenticado do sistema terminar a sua sessão.

Normalmente, foram implementadas as diferentes funcionalidades originalmente especificadas:

- **Realizar login** - Funcionalidade que permite autenticar um dado utilizador do sistema que se encontre registrado.
- **Criar uma conta** - Funcionalidade que permite criar um utilizador normal ao sistema
- **Alterar dados da conta** - Funcionalidade que permite alterar dados de um dado utilizador, tais como o seu **username** e **password**.
- **Atualizar localização** - Funcionalidade que permite atualizar a localização de um dado utilizador do sistema.
- **Aceder ao mapa** - Funcionalidade que permite a um dado utilizador consultar a localização de um estabelecimento, podendo receber direções para o mesmo.
- **Aceder a um estabelecimento** - Funcionalidade que permite aceder a um dado estabelecimento pedido.
- **Filtrar estabelecimentos** - Funcionalidade que permite apresentar uma listagem de estabelecimentos, consoante os filtros selecionados por um dado utilizador do sistema.
- **Avaliar um estabelecimento** - Funcionalidade que permite um dado utilizador poder fazer uma avaliação a um dado estabelecimento.
- **Aceder ao histórico de avaliações** - Funcionalidade que permite aceder a todas as **classificações** que um dado utilizador autenticado tenha realizado.

## 2.3 Arquitetura geral

A aplicação é constituída por duas principais partes: o **servidor**, implementado em **TypeScript**, fazendo uso da biblioteca **express.js** para criar um servidor HTTP, e o **cliente**, implementado também em **TypeScript** com recurso a **React Native**.

O servidor e o cliente comunicam através de uma *REST API*, utilizando o formato **JSON** para as mensagens. O servidor comunica também com uma base de dados **SQLite**, que usa como a sua camada de armazenamento.

### 2.3.1 Servidor

Como foi acima especificado, o servidor é implementado em **TypeScript** com recurso a uma base de dados **SQLite**. O servidor define vários *endpoints REST*, que são usados pelo **frontend** para apresentar os dados e implementar as várias funcionalidades da aplicação.

Para a autenticação de utilizadores, foram utilizados **JSON Web Tokens**, um formato que consiste em encriptar uma *payload JSON* que contém o *id* do utilizador e se é ou não administrador com uma chave secreta apenas conhecida pelo servidor. Deste modo, eliminamos a necessidade de o servidor manter qualquer estado sobre as sessões, sendo apenas utilizado um simples *middleware* que verifica o *token*.

A comunicação com a base de dados é feita através da classe **FunTracker**, criada quando o servidor é inicializado. Esta classe acede à base de dados através de vários **DAOs**, um para cada tabela.

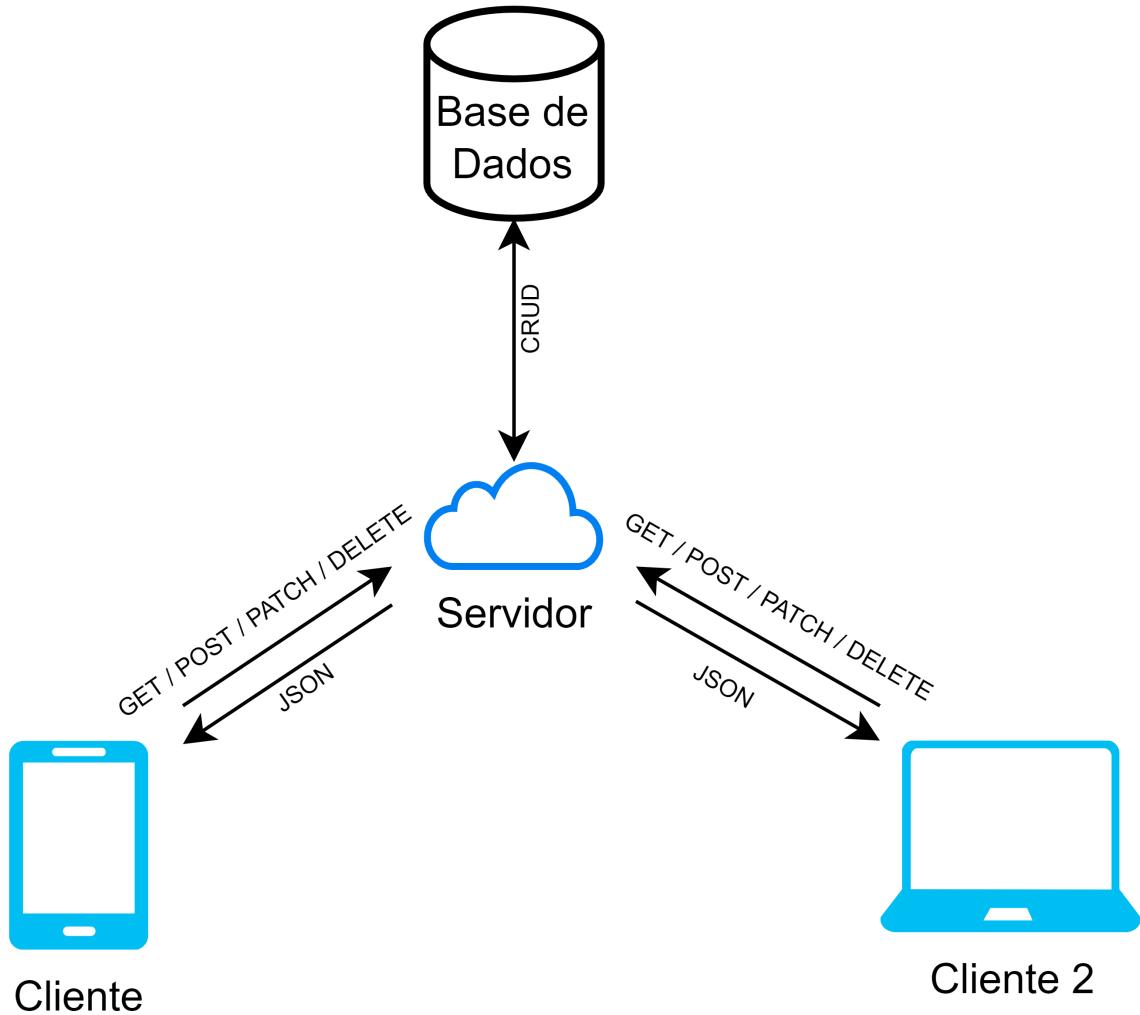


Figura 2.1: Diagrama da arquitetura cliente-servidor

### 2.3.2 Cliente

A arquitetura do cliente é relativamente simples, pois toda a computação e processamento de dados é feita pelo servidor. A aplicação de cliente é, portanto, pouco mais do que uma interface gráfica que pede e submete dados, comunicando com o servidor através da sua API.

## 2.4 Definição e caracterização de cada um dos componentes

### 2.4.1 Backend

Para melhor descrever a nossa implementação do *backend* da aplicação, elaboramos o seguinte diagrama de componentes.

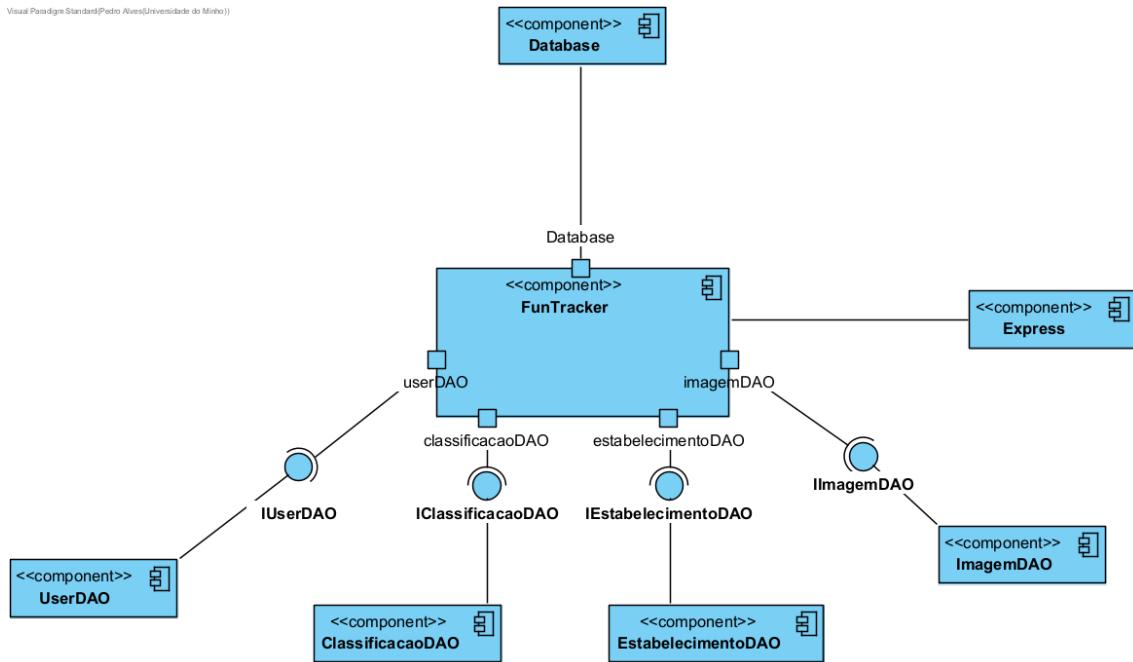


Figura 2.2: Diagrama de Componentes

No **backend**, temos dois componentes principais: **FunTracker** e o **Express**.

## 2.4.2 FunTracker

O **FunTracker** é responsável pelo tratamento e armazenamento de todos os dados presentes no nosso sistema.

Para ser possível escalar a aplicação, permitindo esta ser utilizada por diversos **utilizadores**, tiramos proveito da classe **Promise**. Com isto, maior parte das funções desenvolvidas usam o **async**, conseguindo garantir com isto concorrência e gestão de variáveis partilhadas no nosso sistema.

O **FunTracker** encontra-se composto por outros cinco componentes:

- **PromisedDatabase** - responsável pelo tratamento dos vários pedidos à base de dados.

- **UserDAO** - responsável pelo tratamento e armazenamento de todos os utilizadores presentes no sistema.

Esta classe implementa a interface **IUserDao**.

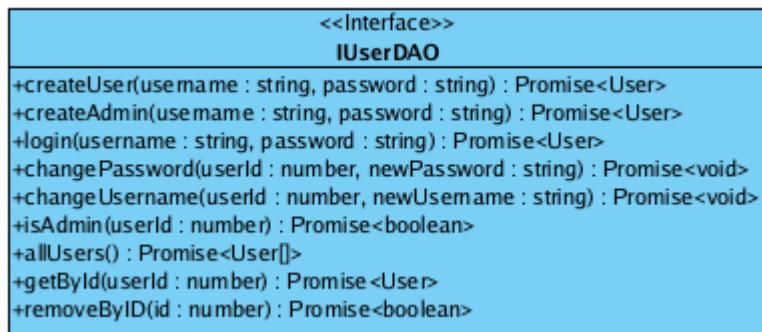


Figura 2.3: Diagrama de Classes interface IUserDAO

Cada **utilizador** do sistema será representado da seguinte forma:

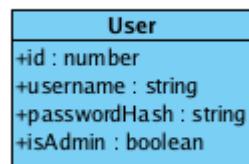


Figura 2.4: Diagrama de Classes User

É de salientar que todas as **passwords** armazenadas usam **bcrypt**.

- **ClassificacaoDAO** - responsável pelo tratamento e armazenamento de todas as classificações realizadas pelos utilizadores presentes no sistema.

Esta classe implementa a interface **IClassificacaoDao**.



Figura 2.5: Diagrama de Classes interface IClassificacaoDAO

Cada **classificação** do sistema será representado da seguinte forma:

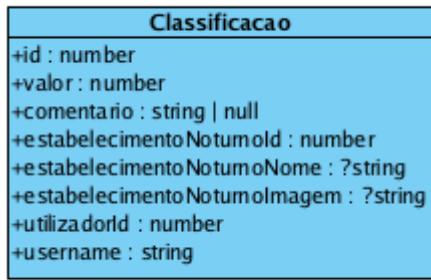


Figura 2.6: Diagrama de Classes Classificacao

- **EstabelecimentoDAO** - responsável pelo tratamento e armazenamento de todos os estabelecimentos presentes no sistema.

Esta classe implementa a interface **IEstabelecimentoDao**.

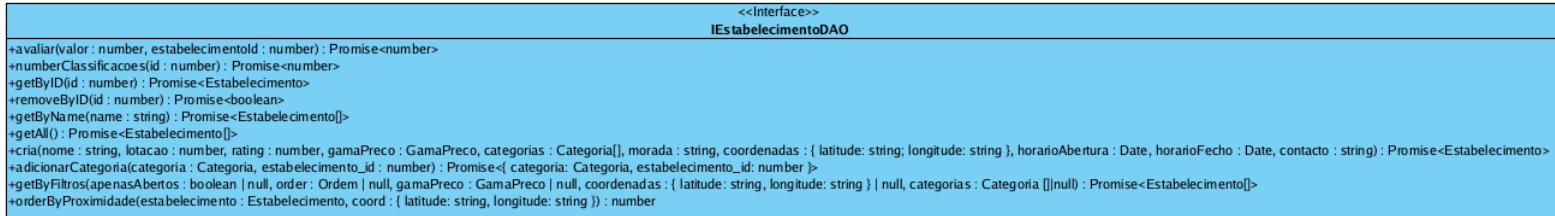


Figura 2.7: Diagrama de Classes interface IEstabelecimentoDAO

Cada **estabelecimento** do sistema será representado da seguinte forma:

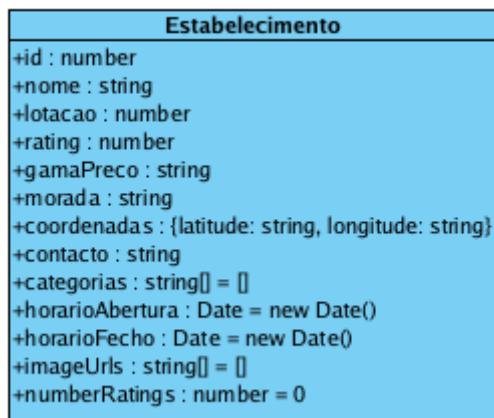


Figura 2.8: Diagrama de Classes Estabelecimento

- **ImagensDAO** - responsável pelo tratamento e armazenamento de todas as imagens associadas aos estabelecimentos presentes no sistema.

Esta classe implementa a interface **IImagensDao**.

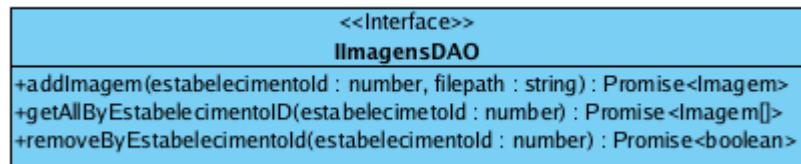


Figura 2.9: Diagrama de Classes interface IImagensDAO

Cada **imagem** do sistema será representado da seguinte forma:

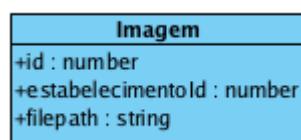


Figura 2.10: Diagrama de Classes Imagem

## 2.4.3 Express

Para implementarmos a comunicação entre a aplicação de *smartphone* e o servidor em *backend*, foi necessário configurar *endpoints* em *Express*. Nas seguintes tabelas está especificado o funcionamento das diversas *routes* desenvolvidas para manipulação e obtenção de dados. Todos os *URLs* especificados nas tabelas são prefixados com o nome da tabela e ainda com '/api/v1' para o seu uso nas *requests* (por exemplo, para a mudança da password do utilizador de *id* 0, utilizariamos o o método *POST* e o *URL* '/api/v1/user/0/password').

- API User

/user				
Método	URL	Parâmetros	Valores	Descrição
POST	/	username	string	Registo de um utilizador normal
		password	string	
POST	/criarAdmin	username	string	Registo de um administrador
		password	string	
PATCH	/{id}/password	{id}		Mudança da password do utilizador identificado
		password	string	
PATCH	/{id}/username	{id}		Mudança do username do utilizador identificado
		username	string	
GET	/			Obtenção de todos os utilizadores
GET	/{id}	{id}		Obtenção do utilizador identificado
GET	/{id}/historico	{id}		Obtenção do histórico de avaliações do utilizador identificado

Tabela 2.1: API User

## ▪ API Estabelecimento

/estabelecimento				
Método	URL	Parâmetros	Valores	Descrição
GET	/{{id}}	{id}		Obtenção o estabelecimento identificado
POST	/	nome	string	Registo de um novo estabelecimento, cujos detalhes estão especificados nos parâmetros
		lotacao	number	
		gamaPreco	'\$' '\$'\$' '\$ \$\$'	
		morada	string	
		coordenadas	{string, string}	
		horarioAbertura	string	
		horarioFecho	string	
		contacto	string	
		categorias	string[]	
		image	string	
		fileMime	string	
DELETE	/{{id}}	{id}		Eliminação do estabelecimento identificado
GET	/	categorias	string	Obtenção de todos os estabelecimentos que cumprem os filtros especificados pelos parâmetros, sendo que a latitude e longitude são fornecidos para eventual ordenação de estabelecimentos por proximidade
		aberto	boolean	
		order	'Proximidade'   'Preco'   'Criticas'	
		precos	'\$' '\$'\$' '\$ \$\$'	
		nome	string	
		latitude	string	
		longitude	string	
GET	/{{id}}/allImagens	{id}		Obtenção de todas as imagens do estabelecimento identificado
POST	/{{id}}/adicionarImagem	{id}		Associação da imagem fornecida ao estabelecimento identificado
POST	/{{id}}/addCategoria	{id}		Associação de uma categoria ao estabelecimento identificado
GET	/{{id}}/classificacoes	{id}		Obtenção de todas as classificações (críticas) associadas ao estabelecimento identificado
GET	/{{id}}/classificacoes/num	{id}		Obtenção do número de classificações (críticas) associadas ao estabelecimento identificado
POST	/{{id}}/classificacoes	{id}		Associação da classificação/crítica fornecida ao estabelecimento identificado
		valor	number	
		comentario	string	

Tabela 2.2: API estabelecimento

- API Session

/session				
Método	URL	Parâmetros	Valores	Descrição
POST	/	username	string	Autenticação de um utilizador na aplicação
		password	string	

Tabela 2.3: API Session

## 2.4.4 Frontend

No *frontend*, temos dois componentes principais: o da apresentação (**App**) e o da persistência de dados de autenticação (apenas persiste o *JSON Web Token*, daí chamar-se **Auth-Context**).

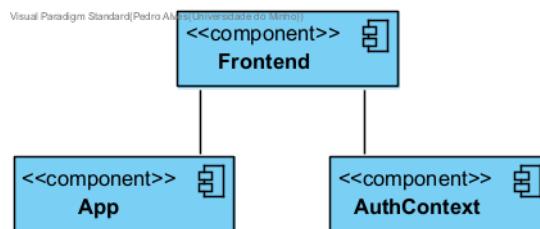


Figura 2.11: Diagrama de Componentes

### App

O componente de apresentação é um simples componente de **React Native**. Define algumas informações básicas (como cores, etc.) e todos os ecrãs visíveis ao longo da aplicação (Login, Sign up, Listar estabelecimentos, Filtrar listagem, Ver um estabelecimento, Avaliar um estabelecimento, Adicionar um estabelecimento, Histórico de avaliações e Opções de conta). Screenshots destes ecrãs podem ser vistas na próxima secção.

### AuthContext

Esta secção é responsável pela gestão do contexto de autenticação do utilizador. É implementada como um *context* de React, para poder ser acedida a partir de qualquer ecrã da aplicação.

Este contexto contém várias informações sobre o utilizador atualmente autenticado:

**Nota:** Em TypeScript, ao contrário de Java, é comum utilizar apenas propriedades públicas para armazenar informação, ao invés de *getters* e *setters*, pois é possível definir que o acesso a uma propriedade seja um método. Devido a isto, decidimos não usar *getters* e *setters*.

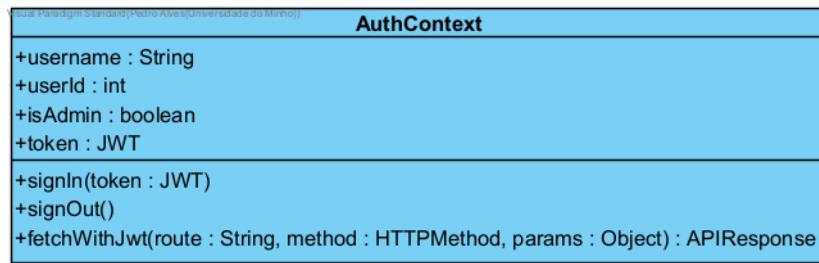


Figura 2.12: Classe AuthContext

Os método **signIn** e **signOut** simplesmente preenchem ou apagam os dados sobre o utilizador atualmente autenticado (como o uso de JWTs permite autenticação *stateless* do lado do servidor, não é necessária nenhuma comunicação com este para encerrar a sessão).

O método **fetchWithJwt** é um método de conveniência, que utiliza a API nativa de JavaScript **fetch** para enviar um pedido à API com o formato correto, incluindo o JWT se este existir. Interpreta também a resposta da API, facilitando consideravelmente o seu uso.

## 2.4.5 Base de dados

Tal como referido anteriormente, a camada de armazenamento é implementada recorrendo a uma base de dados SQLite. Esta base de dados, em geral, segue o esquema que nos foi fornecido, com a exceção da adição de um campo que indica se um utilizador é administrador e de uma tabela para suportar a adição de várias categorias.

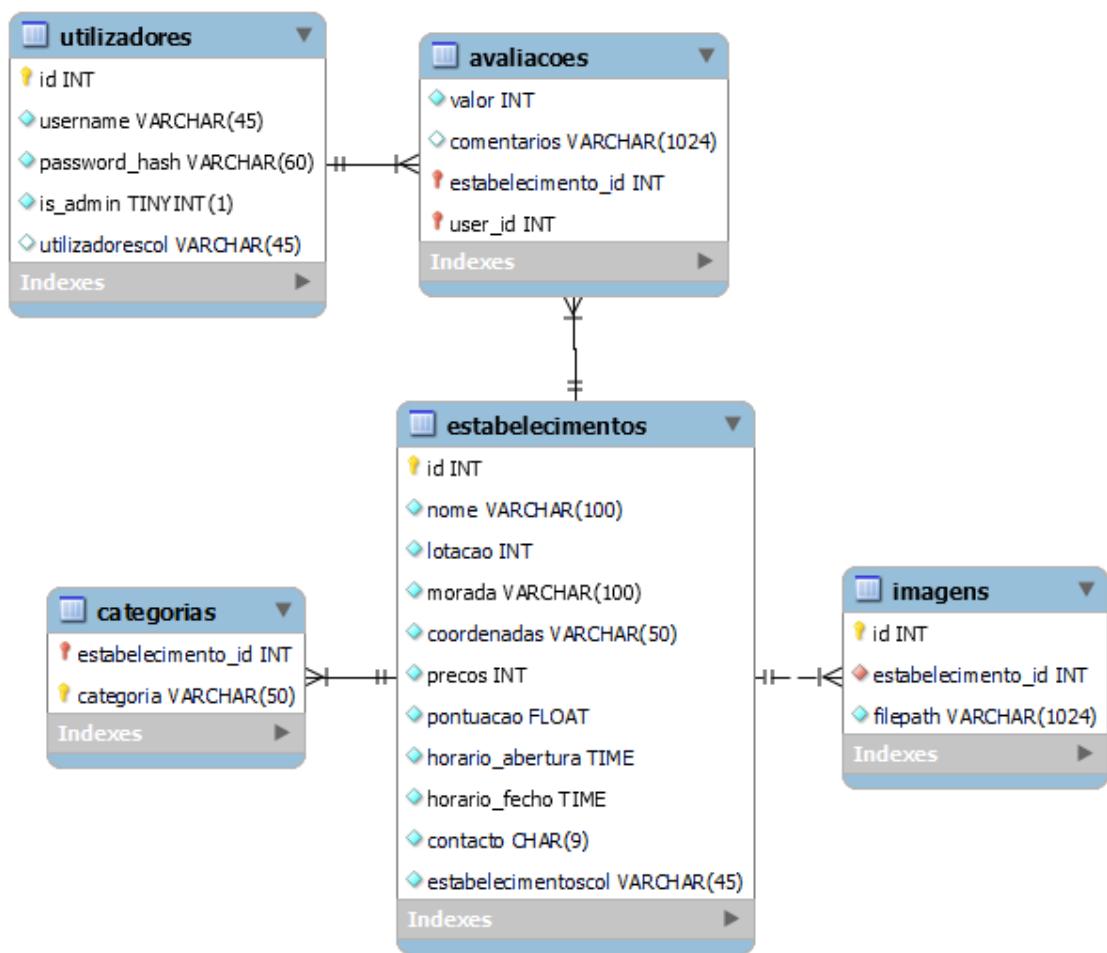


Figura 2.13: Modelo da base de dados

## 2.5 Amostras do funcionamento da Aplicação

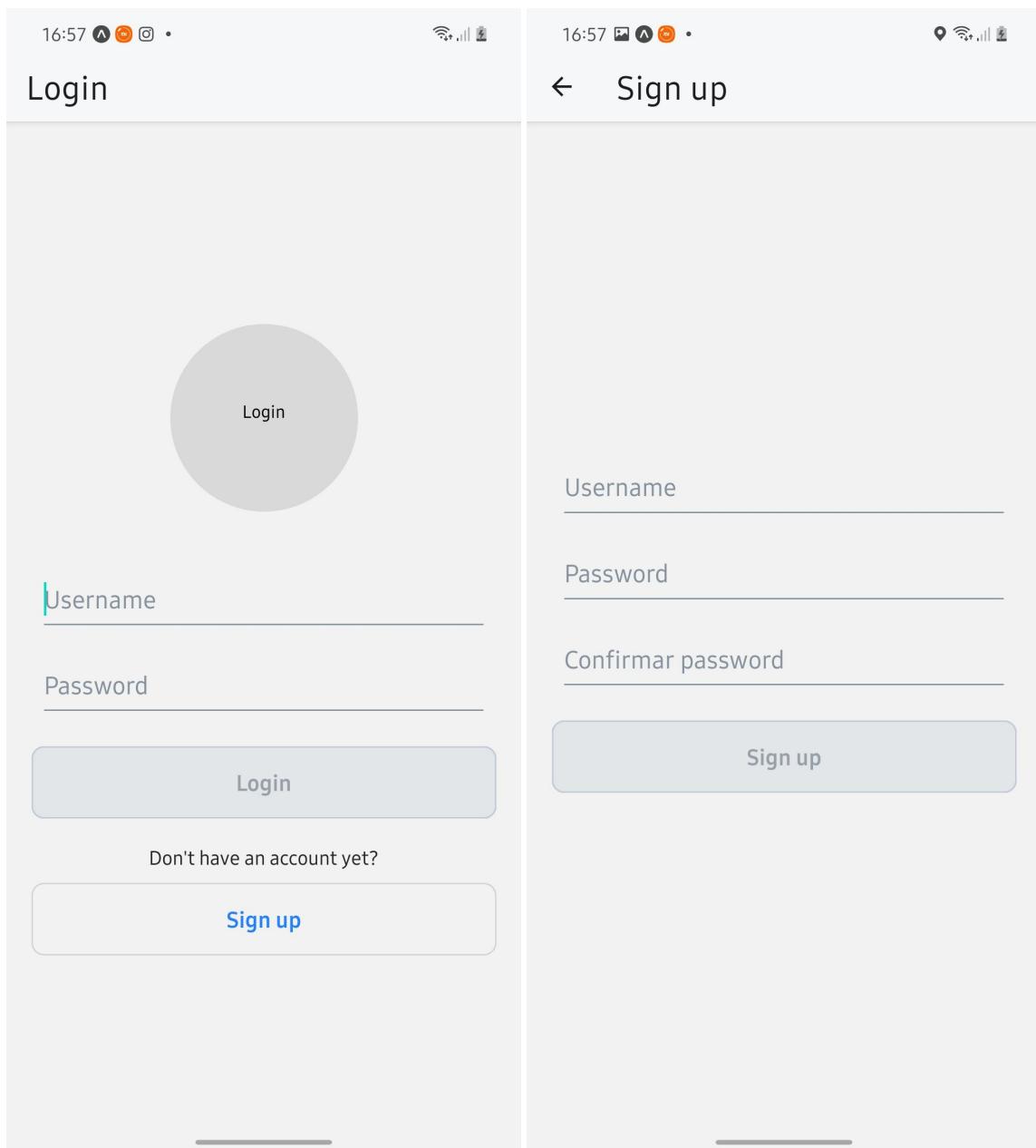


Figura 2.14: Login

Figura 2.15: Sign Up

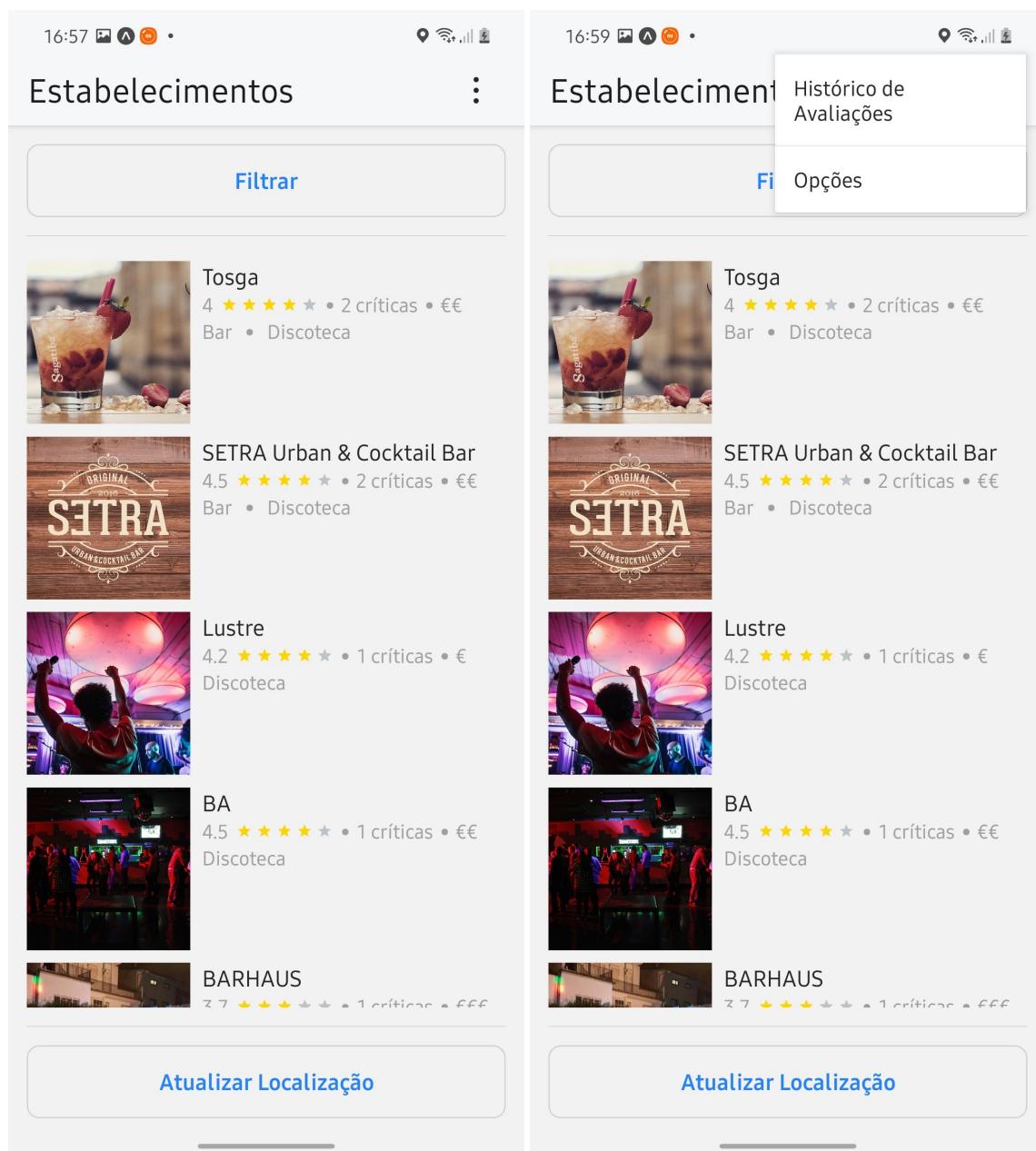


Figura 2.16: Visualizar estabelecimentos

Figura 2.17: Visualizar menu de *pop-up*

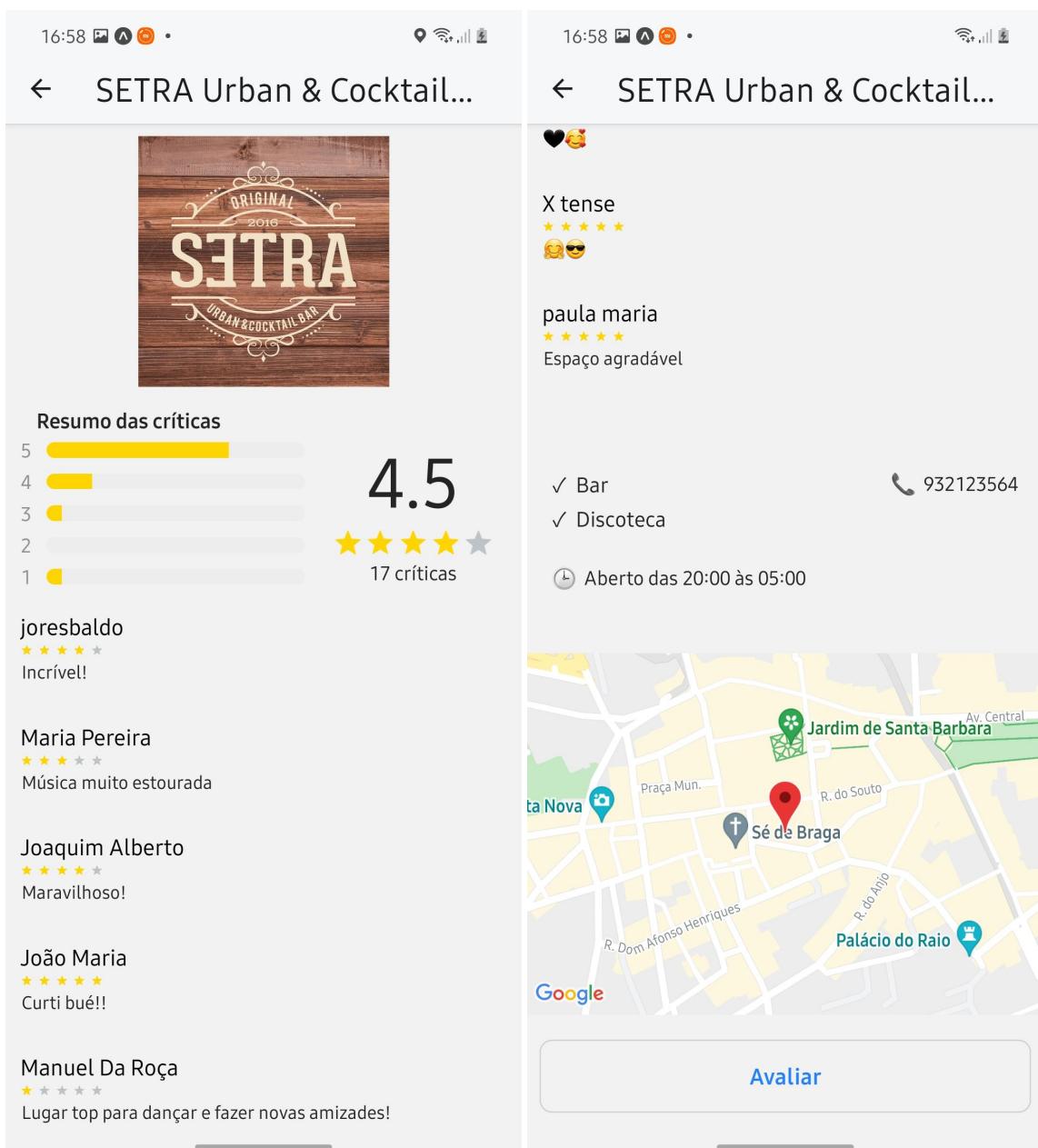


Figura 2.18: Visualizar estabelecimento

Figura 2.19: Visualizar estabelecimento

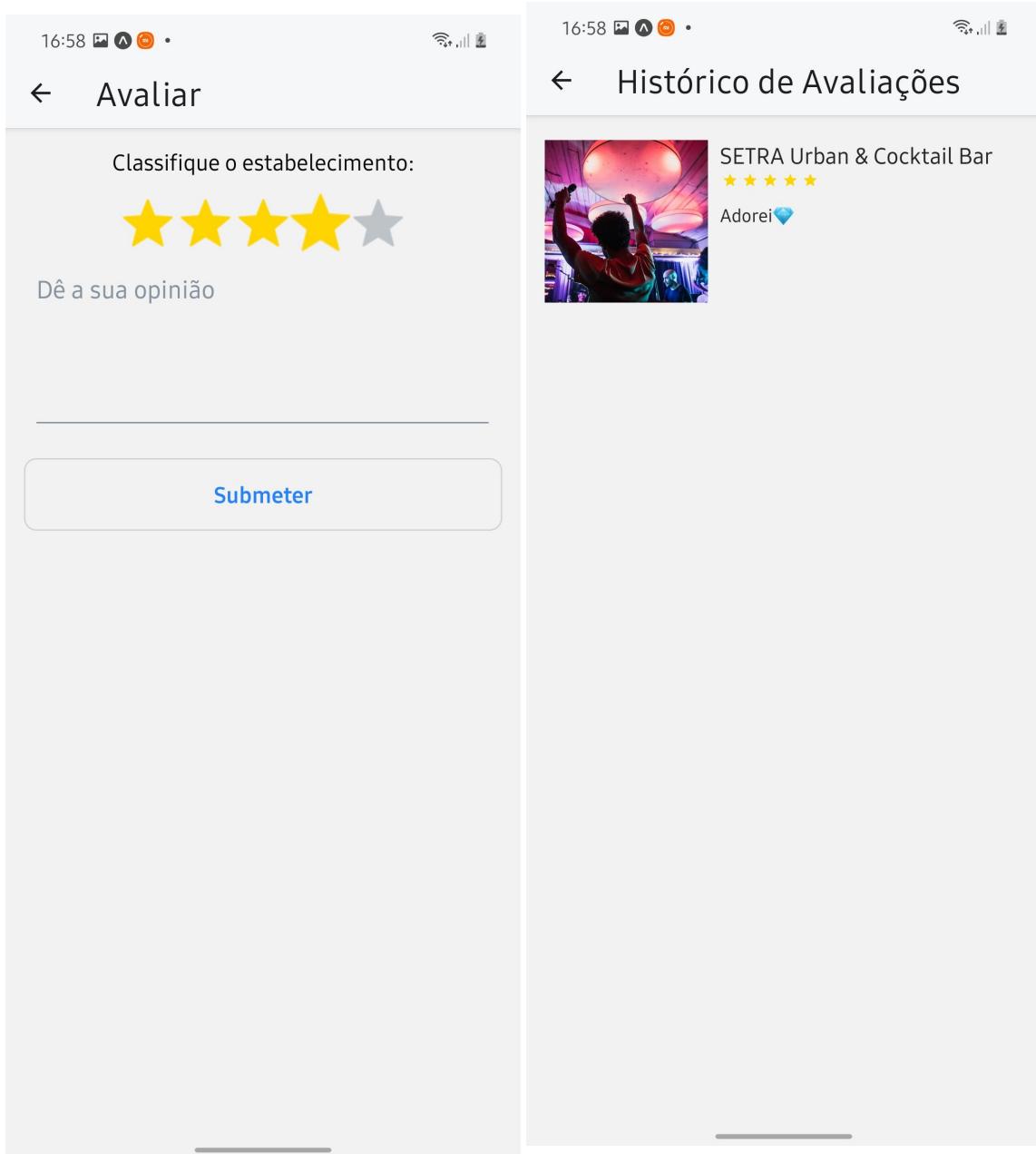


Figura 2.20: Avaliar estabelecimento

Figura 2.21: Visualizar histórico de avaliações

**Filtros**

Ok

Abertos de momento

Bar

Discoteca

Selecionar uma ordem

Nenhuma | Proximidade | **Custo** | Críticas

Selecionar uma gama de preço

Nenhuma | **€** | €€ | €€€

Nome

Pesquisar por nome

**Alterar Credenciais**

Mariii\_01

Alterar Username

Password nova

Repetir password nova

Alterar Password

Figura 2.22: Filtrar estabelecimentos

Figura 2.23: Alterar dados de utilizador

17:00

← Adicionar Local



[Adicionar Imagem](#)

---

Nome do estabelecimento

---

Telefone

---

Lotação máxima

---

Bar

Discoteca

Selecionar uma gama de preço

€€€€€€

Aberto das 0:00 às 23:59

17:00

← Adicionar Local

---

Lotação máxima

---

Bar

Discoteca

Selecionar uma gama de preço

€€€€€€

Aberto das 0:00 às 23:59

---

Braga



[Adicionar](#)

Figura 2.24: Adicionar estabelecimento

Figura 2.25: Adicionar estabelecimento

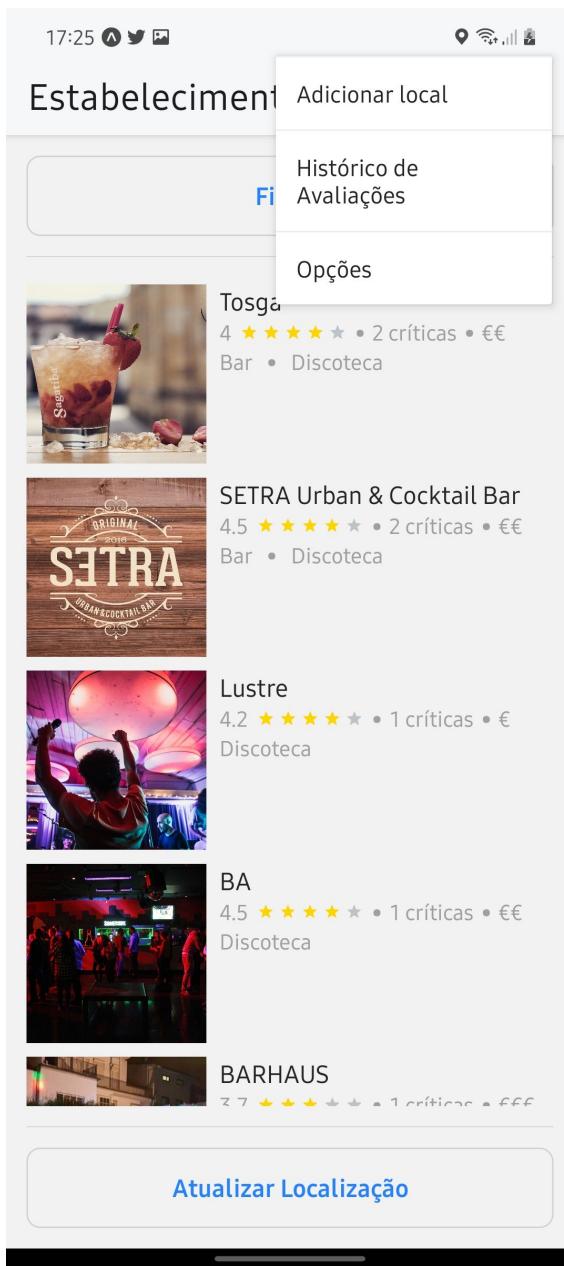


Figura 2.26: Menu *pop-up* de um utilizador especial

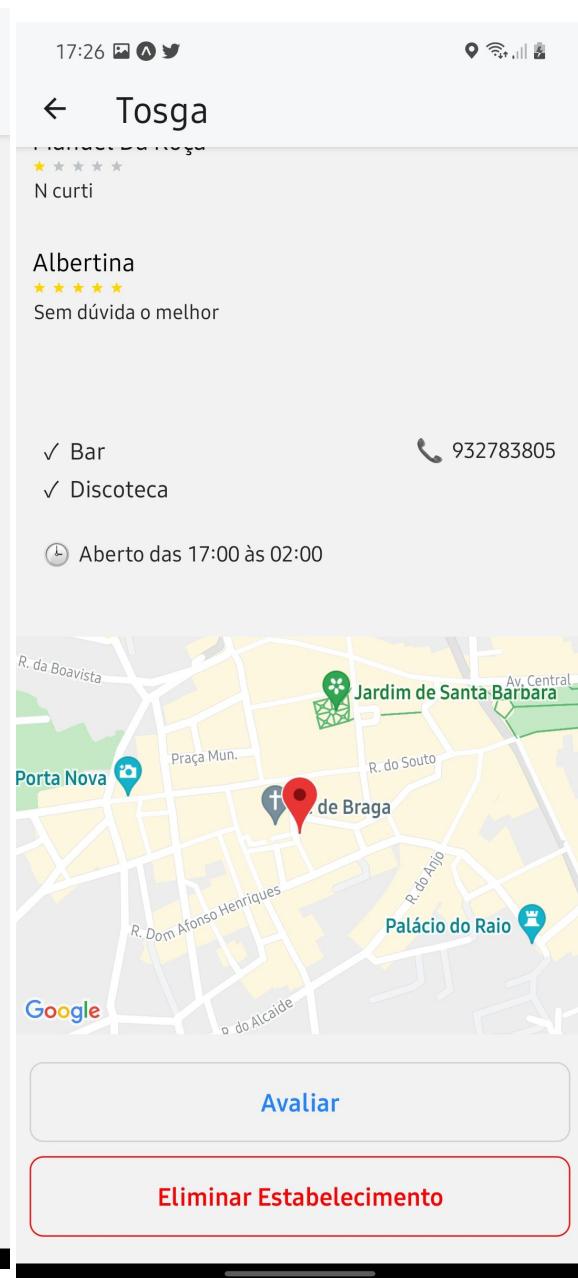


Figura 2.27: Remover estabelecimento

## 3 Conclusões

Após a finalização deste projeto, podemos tirar determinadas conclusões acerca do processo de especificação da aplicação e como este afeta a implementação.

Encontramos certos aspectos previamente descritos que não se adequavam a uma boa implementação ou que não descreviam em suficiente detalhe um dado aspecto da mesma. Alguns destes aspectos foram delineados e criticados no relatório de apreciação elaborado anteriormente, e outros foram encontrados posteriormente durante o processo de implementação. No entanto, continuamos a considerar a especificação suficiente para descrever a ideia do projeto e como esta deve ser implementada de um modo satisfatório no geral.

Em relação à nossa implementação desta especificação, estamos satisfeitos com o produto final. Conseguimos desenvolver uma aplicação que vai de encontro às funcionalidades descritas pelo nosso grupo par, cumprindo os requisitos por eles especificados. Pudemos também, após uma análise das especificações e das críticas levantadas pelos docentes às mesmas, construir novas funcionalidades que consideramos essenciais, sem sacrificar as funcionalidades básicas. Desta forma, desenvolvemos um produto mais coeso e útil.

# Referências

- [1] **React Native** Disponível em: <<https://reactnative.dev/>>.
- [2] **React Native Elements** Disponível em: <<https://reactnativeelements.com/>>.
- [3] **Javascript - documentação** Disponível em: <<https://developer.mozilla.org/en-US/docs/Web/javascript>>.
- [4] **Express.js Guide** Disponível em: <<https://expressjs.com/en/guide/routing.html>>.
- [5] **Expo** Disponível em: <<https://expo.dev/>>.
- [6] **HTML Erros** Disponível em: <<https://kb.iu.edu/d/bfrc>>.

# Listas de Siglas e Acrónimos

**LI4** Laboratórios de Informática IV

**GPS** *Global Positioning System*

**MySQL** *Open-source relational database management system (RDBMS)*

**DAO** *Data Access Object*

**API** *Application Programming Interface*

**JSON** *JavaScript Object Notation*

**REST API** *Representational State Transfer Application Programming Interface*



# Anexos

## Apreciação realizada sobre a especificação recebida

Universidade do Minho  
Licenciatura em Engenharia Informática  
Mestrado Integrado em Engenharia Informática  
Laboratórios de Informática IV  
2021/2022 - 1º Semestre

Avaliação da Fundamentação e Especificação do Sistema de Software

Grupo	3		
<b>Relatório Recebido</b>			
Grupo Nr:	4		
Título:	RanTracker		
<b>Grade de Avaliação</b>			
Elemento	Descrição	Pontuação (0,5)	Anotações
<b>A1 Definição e Fundamentação</b>			
A1	O sistema foi bem contextualizado? Foram apresentados elementos suficientes para se conhecer a motivação, objetivos, medidas de sucesso e o utilizador do sistema? O plano de desenvolvimento foi bem organizado e as reuniões necessárias identificadas e explicadas?	3	Consideramos que na parte da contextualização foi dado demasiado foco à situação pandémica e aos seus aspectos positivos e negativos, ao invés de apenas ser referido, de forma objetiva, que a pandemia afetou negativamente a economia, tendo causado maior impacto nos estabelecimentos noturnos. Contudo, de forma geral, a contextualização, motivação e objetivos estavam claros. Quanto à identidade da aplicação, não foi referida a falta既り de serem o público alvo da mesma. O plano de desenvolvimento está claro, bem estruturado e consideramos que corresponde a uma estratégia inequívoca. A maior parte dos recursos necessários, tanto humanos como não-humanos, são referidos, exceptuando a base de dados. Por último, consideramos que as medidas de sucesso são bastante irrealistas, uma vez que esta aplicação tem como utilizadores principais os habitantes de Braga, particularmente jovens, e é bastante improvável que os utilizadores constituam só jovens. No entanto, a descrição das medidas de sucesso é clara e bem estruturada. Em certas partes da fundamentação são referidas como utilizadores da aplicação os amigos dos estabelecimentos e, noutras partes, são os frequentadores dos espaços noturnos, pelo que se torna um pouco confuso entender quem consideram, efectivamente, os utilizadores/clientes da aplicação.
<b>A2 Levantamento e Análise de Requisitos</b>			
A2	O processo de requisitos foi bem desenvolvido? Foi revelada a forma como foi realizado o seu levantamento? Os requisitos estão claros, bem organizados e detalhados?	3	É referido como foi efetuado o levantamento dos requisitos, no entanto, achamos que o grupo deveria ter sido mais explícito no processo em si, explicando melhor em que consistiram as entrevistas que realizaram. Aparar na secção dos aspetos comportamentais ser referido que o utilizador deve poder "alterar dados da conta" e "verificar as informações de cada estabelecimento", tais funcionalidades não são apresentadas como requisitos funcionais. Além disso, consideramos que o grupo fez demasiado em explicar como os requisitos devem ser cumpridos, indicando passo a passo, quase como uma especificação de usucaso, ao invés de indicar apenas as funcionalidades que o sistema deve ter.
<b>A3 Especificação UML</b>			
A3	A especificação UML está organizada e completa? Acompanha de perto os requisitos previamente estabelecidos? As principais classes de uso estão bem identificadas e detalhadas, bem como os actos dos envolvidos? Os objectos do sistema a desenvolver e as suas relações entre estão bem descritas?	3	No modelo de domínio, achamos que a componente "Histórico de Avaliações", possuída pelo Utilizador, é redundante, pois este Histórico é composto pelas várias Avaliações realizadas pelo Utilizador, elemento já representado no modelo. Quanto aos use cases, o que diz respeito aos utilizadores "normais" da aplicação, achamos que a especificação das suas ações está completa. No entanto, não temos uma falha na especificação de use cases para o lado mais administrativo da aplicação, ou seja, adicionais locais e imagens, apesar que não têm qualquer use case. Para além disso, como já foi referido na secção anterior, alguns use cases são referentes a requisitos que não foram especificados. No entanto, todos os use cases especificados estão bem definidos e permitem a sua implementação sem ambigüidade.
<b>A4 Sistema de Dados</b>			
A4	Os sistemas de dados estão apresentados de forma clara e sustentada? Os dados apresentados foram devidamente enquadrados na especificação UML realizada?	5	O Sistema de dados está bastante claro e simples. Os nomes utilizados são descriptivos e os tipos de dados estão bem explicados. Apenas consideramos que a tabela "Espaço Noturno" não deveria ter espaço no nome, pois é má prática e inconveniente de utilizar.
<b>A5 Interfaces</b>			
A5	Os interfaces (mockups) do sistema não claros e foram devidamente explicados? É possível identificar a sua utilização no processo de especificação UML que foi realizada?	4	Os mockups de interface são em grande parte suficientemente claros e intuitivos para permitir a sua implementação. Estes estão identificados e não deviam ser referenciados nas especificações dos Use Cases, facilitando a sua compreensão no contexto da aplicação. Consideramos as seguintes críticas: - Não foi elaborado um mockup para descrever a via de acesso às interfaces das telas 8 e 9. No entanto, a presença de um "hamburger button" no canto superior esquerdo da tela 3 sugere que estes seriam acessados via uma "ideia" a partir do menu principal dos estabelecimentos (tela 2). - A presença de imagens de perfil de utilizadores nas avaliações de um estabelecimento é contraditória à especificação da informação de um utilizador, que foi descrito como tendo apenas um ID, um username e uma password.
<b>A6 Implementação</b>			
A6	Com base na especificação realizada e na documentação disponibilizada é possível fixar e implementar o sistema requerido?	4	Na nossa opinião, a especificação e documentação disponibilizada permite-nos realizar uma boa implementação do sistema desejado, com a excepção do lado administrativo do sistema, isto é, a gestão de estabelecimentos e de imagens, pois não foi fornecida nenhuma especificação para essa parte.
Total :			
22,00			
3,07			

Figura 3.1: Apreciação realizada sobre a especificação recebida