



UC LI4
2019/2020

Ângelo André Castro de Sousa- A80813
Eduardo João Gomes Teixeira da Costa- A85735
José Miguel Fernandes Madeira Pinto- A84590
Luís Francisco Mendes Lopes- A85367
Ricardo Vieira Carvalho- A84261

RESUMO

Serve o presente relatório para demonstrar e explicar as diferentes fases de desenvolvimento de uma aplicação móvel, a CIVERE, cujo objetivo é a centralização de informação para apoio ao turismo e ao turista.

Primeiramente, após um enquadramento e definição de objetivos para este projeto, demonstra-se o trabalho desenvolvido durante o levantamento de requisitos e consequente modelação da aplicação que propusemos desenvolver, assim como se explicitam, brevemente, as tecnologias adotadas para a resolução do mesmo. Durante a fase de modelação, desenvolveram-se vários diagramas e modelos que, neste caso, nos serviram de referência para o decorrer do projeto.

Posteriormente, na secção referente à fase seguinte, explicam-se as diferentes partes da implementação realizada, distinguindo-a entre Base de Dados, *BackEnd* e *FrontEnd*. Em cada uma delas, explicam-se, de forma mais pormenorizada, as várias decisões tomadas, o seu objetivo e de que forma se procedeu.

De seguida, em forma de resultados finais obtidos, apresentam-se as várias interfaces desenvolvidas, assim como uma demonstração, através de diagramas de sequência, do modo de operar das mesmas. Estas primam pela interatividade e simplicidade de uso, tal como idealizamos e estabelecemos, inicialmente, como objetivo.

Por último, em jeito de conclusão, faz-se um balanço de todo o trabalho desenvolvido e daquilo que são as aspirações da equipa relativamente a trabalho futuro e que deverá ser implementado.

ÍNDICE

INTRODUÇÃO.....	4
Enquadramento	4
Objetivos, Motivação e Contribuição do Projeto	5
LEVANTAMENTO DE REQUISITOS E DISCUSSÃO	6
TECNOLOGIAS ADOTADAS	8
<i>Android Studio</i>	8
<i>Stripe Payments.....</i>	8
<i>Adobe Xd.....</i>	9
<i>SQLite</i>	9
<i>Picasso.....</i>	9
<i>REST API</i>	9
<i>JSON</i>	10
<i>JSONWEBTOKEN</i>	10
<i>AZURE.....</i>	10
<i>SQL</i>	10
<i>C#.....</i>	11
<i>Java</i>	11
MODELAÇÃO DA APLICAÇÃO	12
IMPLEMENTAÇÃO DA BASE DE DADOS	16
Tabela <i>Users.....</i>	17
Tabelas <i>Carteira_Guia</i>	17
Tabelas <i>Pedidos_Payout.....</i>	18
Tabela <i>Contactos</i>	18
Tabela <i>Morada</i>	19
Tabelas <i>restaurantes, hoteis e ponto_interesse.....</i>	19
Tabelas <i>Classificacao_Restaurantes, Classificacao_Hoteis e Classificacao_PontoInteresse</i>	20
Tabelas <i>Excursões</i>	21
Tabelas <i>Inscrição.....</i>	21
Tabelas <i>Versoes_BD.....</i>	22
Tabelas <i>Versoes_Tabelas</i>	22
<i>Triggers</i>	23
IMPLEMENTAÇÃO DO BACKEND.....	24
<i>Controllers.....</i>	25

<i>Models</i>	27
IMPLEMENTAÇÃO DO FRONTEND	29
<i>Models</i>	29
<i>View</i>	30
<i>Controller</i>	31
RESULTADOS FINAIS	32
<i>Interfaces</i>	32
Diagramas de Sequência	41
CONCLUSÃO	47
TRABALHO FUTURO	48
REFERÊNCIAS	49
ANEXOS	50
ANEXO A - Diagramas de <i>Gantt</i> da organização quinzenal de tarefas	50
ANEXO B - Modelo lógico final da BD	52
ANEXO C - Especificações dos <i>Use Cases</i>	53
ANEXO D –Protótipos de Interfaces.....	59

ÍNDICE DE FIGURAS

Figura 1 - Diagrama de Classes Inicial	12
Figura 2 - Diagrama FrontEnd Final.....	13
Figura 3 - Modelo lógico da Base de Dados	14
Figura 4 - Diagrama de Use Cases inicial.....	15
Figura 5 - Tabela Users	17
Figura 6 – Tabela da Carteira de guia.....	17
Figura 7 - Tabela de Pedidos de Payout.....	18
Figura 8 - Tabela de Contactos.....	18
Figura 9 - Tabela de Morada	19
Figura 10 - Tabelas das Entidades Ponto de interesse, Hotéis e Restaurantes	19
Figura 11 - Tabelas de Classificações.....	20
Figura 12 - Tabela de Excursões	21
Figura 13 - Tabela de Inscrição.....	21
Figura 14 - Tabela Versoes_BD.....	22
Figura 15 - Tabela Versoes_Tabelas.....	22
Figura 16 - Modelo REST API	24
Figura 20 - Exemplo de Post Request (Talend API Tester)	26
Figura 21 - Exemplo de resposta em formato Json Object (Talend API Tester).....	26
Figura 19 - Métodos de segurança relativos a SQL Injection	27
Figura 20 - Exemplo de Token enviado pelo FrontEnd para ser validado.....	28
Figura 21 - Interfaces Login	32
Figura 22 - Interface registo e Interface registo com campos inválidos	33
Figura 23 - Menu inicial (à esquerda) e Menu inicial com os hotéis selecionados (à direita)	34
Figura 24 - Menu inicial com os restaurantes selecionados (à esquerda) e Menu inicial com os Pontos de Interesse selecionados (à direita)	34
Figura 25 - Menu inicial com Excursões selecionadas (à esquerda) e Menu de Excursões para os guias (à direita).....	35
Figura 26 - Interface adicionar nova excursão	35
Figura 27 - Interface wallet guia.....	36
Figura 28 - Interface escolher cidade	37
Figura 29 - Interface definições e Interface definições com Dark Mode ativado	37
Figura 30 - Interface de alteração de idioma	38
Figura 31 - Interface para trocar de número e Interface troca de número.	38
Figura 32 - Interface informativa de Entidade	39
Figura 33 - Interface de comentários e classificações.....	39
Figura 34 -Interface para adicionar comentário/classificação.....	40
Figura 35 - Interface informativa de excursões e interface pagamento.....	40
Figura 36 - Diagrama de sequência para o algoritmo no FrontEnd que atualiza as tabelas SQLite	41
Figura 37 - Diagrama de sequência para o algoritmo no BackEnd que atualiza as tabelas SQLite	42
Figura 38 - Diagrama de sequência de sistema Login	43
Figura 39 - Diagrama de sequência de sistema Registar.....	44
Figura 40 - Diagrama de sequência de sistema Inserir comentário	44
Figura 41 - Diagrama de sequência de sistema Inscrição em Excursão	45
Figura 42 - Diagrama de sequência de sistema Cashout.....	46

INTRODUÇÃO

Enquadramento

O turismo é uma atividade ampla com grande impacto na economia do nosso país. A atividade turística é uma prática cada vez mais usual na sociedade atual e, como fruto desta nova realidade, várias outras indústrias surgem para a complementar. Deste modo, indústrias dedicadas ao alojamento, à restauração, à animação ou à cultura revelam-se igualmente valorosas na construção de um espaço promissor de boas práticas turísticas. Com estas áreas agregadas e em funcionamento favorável ao turismo, qualquer destino se torna mais procurado, conquistando receitas turísticas superiores. Para os turistas, algumas informações sobre o destino selecionado tornam-se fundamentais, como os possíveis alojamentos ou até mesmo as atrações a visitar, informações estas que, por vezes, requerem numerosas pesquisas. O tempo consumido na recolha de dados turísticos sobre o destino nomeado poderia ser diminuído consideravelmente com uma concentração de todos estes dados numa só plataforma.

Neste seguimento, apresenta-se a CIVERE, uma aplicação móvel que reúne todos os conhecimentos associados às práticas turísticas, expondo todos os serviços de alojamento, restauração ou animação do local desejado. Desta forma torna-se mais cómodo viajar, pois permite aos turistas aceder a informações pertinentes em qualquer parte do mundo. Enquadra-se, assim, numa tentativa de reduzir o tempo e o stress causado pelo planeamento de qualquer viagem. O nome CIVERE surge da complementação de duas palavras do latim - “Cidade” e “Excursão”. Sendo o que melhor define o conceito implementado na aplicação: apresentação de uma cidade e oferta de um serviço de visitas guiadas efetuadas por guias freelancer, sendo prático e proporcionando o crescimento económico no turismo.

Objetivos, Motivação e Contribuição do Projeto

A iniciativa para o desenvolvimento da aplicação surgiu após nos apercebermos de que havia e há um enorme mercado de aplicações focado no turismo que, especialmente em Portugal, não se encontra completamente aproveitado. Deste modo, decidimos tentar colmatar esta falha, também para nosso próprio benefício. E assim, após uma sessão de *brainstorming*, formou-se a ideia original para a CIVERE.

Esta ideia passaria por juntar todas as diferentes componentes ligadas ao turismo da forma mais interativa possível. Tornar-se-ia também um modelo de negócio, tendo em vista a adesão a um serviço de excursões/visitas guiadas pagas, como já referidas anteriormente, efetuadas por guias *freelancer*. No entanto, umas pequenas percentagens dos preços praticados seriam retiradas para manutenção da aplicação e procura de um melhor serviço prestado. Se a motivação no desenvolvimento da aplicação está relacionada com junção das diferentes componentes do turismo, os seus objetivos passam por apresentar os diferentes hotéis, restaurantes e pontos de interesse presentes numa qualquer cidade escolhida, de uma forma apelativa sendo que todos estes têm uma classificação a eles associada e atribuída pelos utilizadores, que têm ainda a possibilidade de efetuar comentários.

Podemos, então, concluir que o desenvolvimento da CIVERE trará aos utilizadores um sítio onde poderão reunir todas as informações relevantes para uma visita turística e dará oportunidade a pessoas que tenham conhecimentos na área turística e procuram ter algum rendimento extra, juntando-se a nós nesta “viagem”.

LEVANTAMENTO DE REQUISITOS E DISCUSSÃO

Antes de começarmos a desenvolver a CIVERE, necessitamos de decidir quais seriam os seus pontos fundamentais, de forma a definir requisitos e funcionalidades mínimos a alcançar antes do lançamento da aplicação para o público, assim como tomar decisões acerca dos mesmos. Sendo assim, nesta secção, explicita-se aquilo que foram os requisitos mínimos inicialmente definidos pela equipa, o que seria necessário para os implementar e as decisões que foram tomadas nesse sentido, não só numa fase inicial, mas ao longo de todo o projeto.

Após o levantamento de requisitos, chegamos à conclusão que a nossa aplicação teria, indubitavelmente, de ser capaz de fornecer aos seus utilizadores a funcionalidade de consultar morada, classificação, fotos e comentários, bem como disponibilizar uma breve história/descrição de pontos de interesse, restaurantes e hotéis de cada cidade.

Do ponto de vista das excursões e visitas turísticas, achamos, inicialmente, que faria sentido permitir que um utilizador obtivesse a localização em tempo real de um guia ativo e, inclusive, reservar um guia através do seu perfil (com informações acerca do mesmo, como horários, por exemplo) ou mostrar a localização de um *tour*. Após algum debate, entendemos que a localização em tempo real de um guia não só não é relevante, como também não nos seria lucrativo, pois os utilizadores poderiam usar a aplicação apenas para se encontrarem com o guia, sem que o pagamento fosse feito através da CIVERE. Entendemos, então, que o mais importante seria permitir que os utilizadores se pudessem inscrever nas excursões, pagando o preço de uma inscrição, definido pelo guia.

De forma a obtermos uma margem de lucro nessas transações, optamos por retirar 10% do valor pago a cada guia e enviar os restantes 90% para a sua *Wallet* (criada aquando do seu registo). Após passar a data de uma excursão, ou seja, após esta ter sido realizada, permitimos ao guia fazer um pedido de *Cashout* do valor pretendido e existente na sua Carteira.

Relativamente a reservas em qualquer tipo de estabelecimento ou local, decidimos que poderíamos facilitar o processo aos utilizadores, enviando automaticamente um *email* para o estabelecimento devido. Mas, nesta fase inicial e de desenvolvimento da CIVERE, apenas disponibilizamos aos utilizadores os contactos necessários para efetuarem eles próprios a reserva da forma que pretendem e esteja disponível, seja por telefone ou por *email*, até porque, como viemos a verificar, nem todos eles possuem serviço de *email* disponível.

Como em qualquer aplicação que envolva uma criação de uma conta, um *user* tem a possibilidade de se registar e a necessidade de fazer *login*. Optamos por permitir a

utilizador já registado editar apenas o seu número de telemóvel e a sua palavra-passe, pois os restantes elementos são identificativos dessa mesma conta.

Outro ponto fundamental da aplicação a desenvolver, passaria por admitir avaliações e comentários a todos as entidades do nosso sistema, numa escala de 0 a 5 e um texto livre, respetivamente, de forma a que esta se tornasse mais interativa e porque, no mundo atual, a opinião isenta e de experiência real é, cada vez mais, um instrumento de decisão na hora de realizar uma atividade.

Em termos da interface que disponibilizamos ao utilizador, decidimos que deveríamos oferecer as opções de *Dark Mode*, um menu de seleção de cidade para onde se pode navegar e disponibilizar a *app* em vários idiomas, por se tratar, acima de tudo, de uma aplicação relativa a turismo. De forma a facilitar o crescimento da nossa aplicação, decidimos que também seria necessário fornecer uma interface de sugestões aos utilizadores, na qual estes seriam redirecionados para o seu e-mail, e que lhes permitisse enviar sugestões de pontos de interesse, hotéis e restaurantes que considerem importante colocar na nossa Base de Dados.

Relativamente ao *FrontEnd*, propusemo-nos a desenvolver uma Base de Dados local que permitisse que parte da aplicação funcionasse sem a utilização de internet e, ao mesmo tempo, garantisse maior fluidez da aplicação devido a diminuir a necessidade de carregamento constante de dados.

À medida que o projeto foi sendo desenvolvimento, fomos, obviamente, alterando e adaptando o que foi inicialmente idealizado, pois, no mundo da programação e, em específico, das aplicações, nunca aquilo que é planeado no início segue um caminho retilíneo até ao final. Deste modo, foram acrescentadas novas funcionalidades, retiradas outras e algumas delas foram adaptadas, conforme nos pareceu mais lógico e prático do ponto de vista do utilizador. Houve, ainda, outras que, apesar de considerarmos relevantes, acabaram por ser adiadas para uma eventual segunda versão da CIVERE, como, por exemplo, consultar informação de serviços de auxílio médico/policial como hospitais e esquadras, como a morada e contactos dos mesmos. Outra funcionalidade que, apesar de considerarmos bastante prática, acabamos por desistir devido à necessidade de implementar uma *Google API*, foi a capacidade de aceder a um mapa interativo da cidade selecionada. No sentido contrário, achamos fundamental acrescentar, relativamente ao plano inicial, a implementação, desde esta fase, de um controlo de versões que permita manter a aplicação atualizada para todos os utilizadores.

TECNOLOGIAS ADOTADAS

Pretende-se, nesta secção, explicitar sucintamente as características e no que consistem as várias tecnologias e plataformas utilizadas ao longo da realização deste trabalho e que irão, ao longo do relatório, ser referidas.

Android Studio

É um ambiente de desenvolvimento integrado (*IDE*) para desenvolver para a plataforma *Android* e encontra-se disponível para as plataformas *Windows*, *Mac OS X* e *Linux*.

A sua utilização tem algumas vantagens como, por exemplo, ser baseada em *IntelliJ* da *JetBrains* e ter compilações em *Gradle*, um assistente baseado em predefinições com designs e com componentes comuns num ambiente *Android*. Existe, ainda, um editor de *layout* bastante rico que permite aos usuários fazer *drag and drop* das componentes de interface se assim o pretendem. Temos, também, a opção de pré-visualizar *layouts* em várias configurações de tela.

De qualquer forma, principal vantagem do *Android Studio* encontra-se no *Android Emulator*, que instala e inicializa aplicativos mais depressa que os dispositivos reais, permitindo criar protótipos e testar aplicativos em todos os tipos de dispositivo *Android*. Para além disso, é ainda possível simular diversos recursos de hardware, como localização por *GPS*, latência de rede, sensores de movimento e interação multi toque.

Stripe Payments

O *Stripe* fornece serviços como proteção contra fraude, taxas fixas independentemente da rede e *API's* que permitem o processamento de cartões de crédito. As *API's* fornecidas são úteis devido à complexidade de uma transação bancária com cartões. Existem múltiplas entidades intervenientes numa transação: os clientes, comerciantes, organizações e instituições financeiras que fornecem as ferramentas para processar e aceitar cartões, processadores que são geralmente grandes instituições financeiras que fornecem a espinha dorsal de todas as transações, organizações de vendas independentes (*ISOs*), emissoras de cartões e redes de cartões. Devido ao processo de transação ser tão complexo, antes da conceção do *Stripe*, a maioria dos negócios apenas aceitava pagamentos físicos. O *Stripe* veio alterar esse paradigma, tornando os pagamentos com cartão virtual acessíveis a todas as *Web apps*.

Adobe Xd

Adobe *Xd* é uma ferramenta de design para aplicações web e aplicativos móveis, desenvolvida e publicado pela *Adobe Inc.*. Esta ferramenta ajuda a visualizar o resultado do trabalho diretamente em dispositivos móveis.

Foi usada neste projeto pela sua simplicidade de utilização e poder para desenhar interfaces, mas acima de tudo devido ao seu grau de compatibilidade com o *Android Studio*.

SQLite

É uma librarria em C que implementa uma base de dados com todas as funcionalidades, pequena, rápida, independente e consistente. O *SQLite* encontra-se embutido dentro de todos os *smartphones* e na maioria dos computadores. Este tem um formato de ficheiro estável em múltiplas plataformas, que o torna na solução ideal para a aplicação de bases de dados locais.

Picasso

Biblioteca para *download* e *caching* de imagens para *Android*. Esta biblioteca permite o *download* de imagens através de um *URL*.

REST API

Representational State Transfer (REST) é um estilo de arquitetura *software* que define o conjunto de restrições a serem usadas para a criação de *web services*. A *REST API* fornece uma maneira simples e flexível de aceder a serviços *web*, sem haver nenhum processamento.

Os *Web services RESTful* permitem que os sistemas solicitantes acedam e manipulem representações textuais de recursos da *Web*, usando um conjunto uniforme e predefinido de operações sem estado.

As comunicações feitas pela *REST API* usam apenas *HTTP requests*. Este funcionamento acaba por ser feito através de um *request* do cliente para o *server*, através do *URL* como *HTTP GET, POST, PUT* ou *DELETE*. Posteriormente obtemos uma resposta do servidor num formato definido por quem implementa o *BackEnd* da aplicação, tendo à sua disposição formatos diversos como *HTML, XML, Image* ou *JSON*.

JSON

JSON ou JavaScript Object Notation é um formato de estruturação de informação.

Tal como o *XML*, é uma das maneiras de formatar informação. Este formato de informação é usado para as aplicações comunicarem entre si.

O *JSON* é caracterizado por ser *Human-readable* e *writable*, ter um formato leve de troca de dados, baseado em texto e pode ser escrito em qualquer linguagem de programação.

JSONWEBTOKEN

JSON web token(JWT) é um *JSON Object* que é usado para transferir, de forma segura, informação na *Web*. Este pode ser usado para a autenticação do sistema e pode ser usado também para a troca de informações. O *token* gerado é maioritariamente composto por um *header*, *payload* e *signature*. Estas 3 partes são separadas por pontos.

O *header* no *JWT* é usado para descrever as operações criptográficas aplicadas. Também pode conter informação sobre a informação que estamos a enviar.

A parte referente ao *Payload* é onde toda a informação do utilizador é enviada. A informação enviada neste campo encontra-se desprotegida e, por esta razão, não é uma boa prática enviar informação sensível neste campo.

Já a *Signature* é usada para verificar e autenticar o *token*. O *header* e o *payload* são codificados e unidos por um ponto (“.”) e são, depois, “*hashed*” usando um algoritmo de *hashing* definido pela chave secreta de encriptação. Esta assinatura é, posteriormente, adicionada ao *header* e ao *payload* através de um ponto (“.”), formando o *token* final.

AZURE

O *Microsoft Azure* é um conjunto de serviços *cloud* que dá liberdade para criar, gerir e implementar aplicações numa rede global em grande escala, através de um amplo leque de ferramentas e arquiteturas entre as quais podemos escolher.

SQL

É a linguagem padrão das bases de dados que é usada para criar, manter e recuperar informação de uma base de dados relacional. Esta linguagem é usada quando temos a informação estruturada (em forma de tabelas).

C#

É uma linguagem de programação moderna e orientada a objetos. Foi desenvolvida pela *Microsoft* dentro da iniciativa *.NET*. A linguagem *C#* apresenta semelhanças sintáticas com a linguagem *Java*, o que torna a adaptação fácil a utilizadores com conhecimentos em *C*, *C++* e/ou *Java*.

Java

É uma das mais populares linguagens de programação em todo o mundo. É uma linguagem orientada a objetos, mas, no entanto, não é considerada como uma linguagem de objetos pura, uma vez que permite a utilização de tipos de dados primitivos, tais como *char* e *int*.

Java é usada em todos os tipos de aplicações, desde aplicações móveis (*Android* é baseado em *Java*), aplicações *desktop*, aplicações *web*, aplicações *client server*, aplicações *enterprise*, entre outras.

MODELAÇÃO DA APLICAÇÃO

Nas primeiras fases do projeto, após idealização da aplicação a desenvolver, passamos, então, à sua modelação. Nesta fase decidimos desenvolver vários tipos de modelos e diagramas que, apesar de sabermos que não seriam definitivos, devido à dinâmica envolvida no desenvolvimento de uma aplicação (neste caso, móvel), foram, de qualquer forma, bastante úteis como ponto de partida e como base, representando alguns pontos estruturais da mesma, que permaneceram, com ou sem alterações, até à versão final que apresentamos.

O diagrama de classes apresentado na figura abaixo, foi desenvolvido numa fase inicial da modelação, que tinha como objetivo ser usado como um ponto de referência ao longo do desenvolvimento da aplicação, tanto em ambiente *BackEnd* como *FrontEnd*.

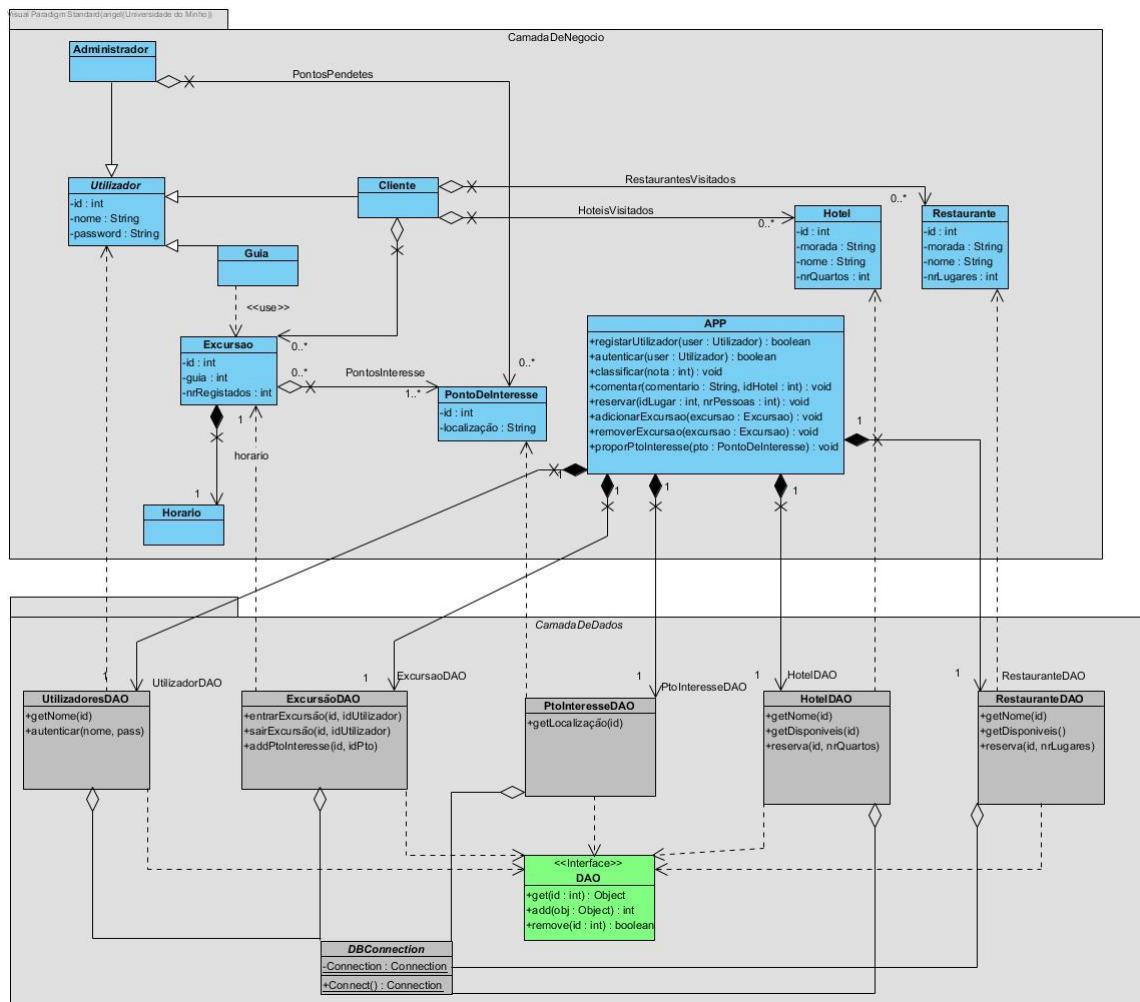


Figura 1 - Diagrama de Classes Inicial

Diversas classes apresentadas no diagrama foram abolidas por completo, devido a simplificações na representação da informação ou por ambiguidade das mesmas, outras foram alteradas e ainda outras foram acrescentadas posteriormente, de forma a garantir que toda a informação presente no sistema fosse representada e manipulada corretamente, garantindo a implementação de todas as funcionalidades do sistema.

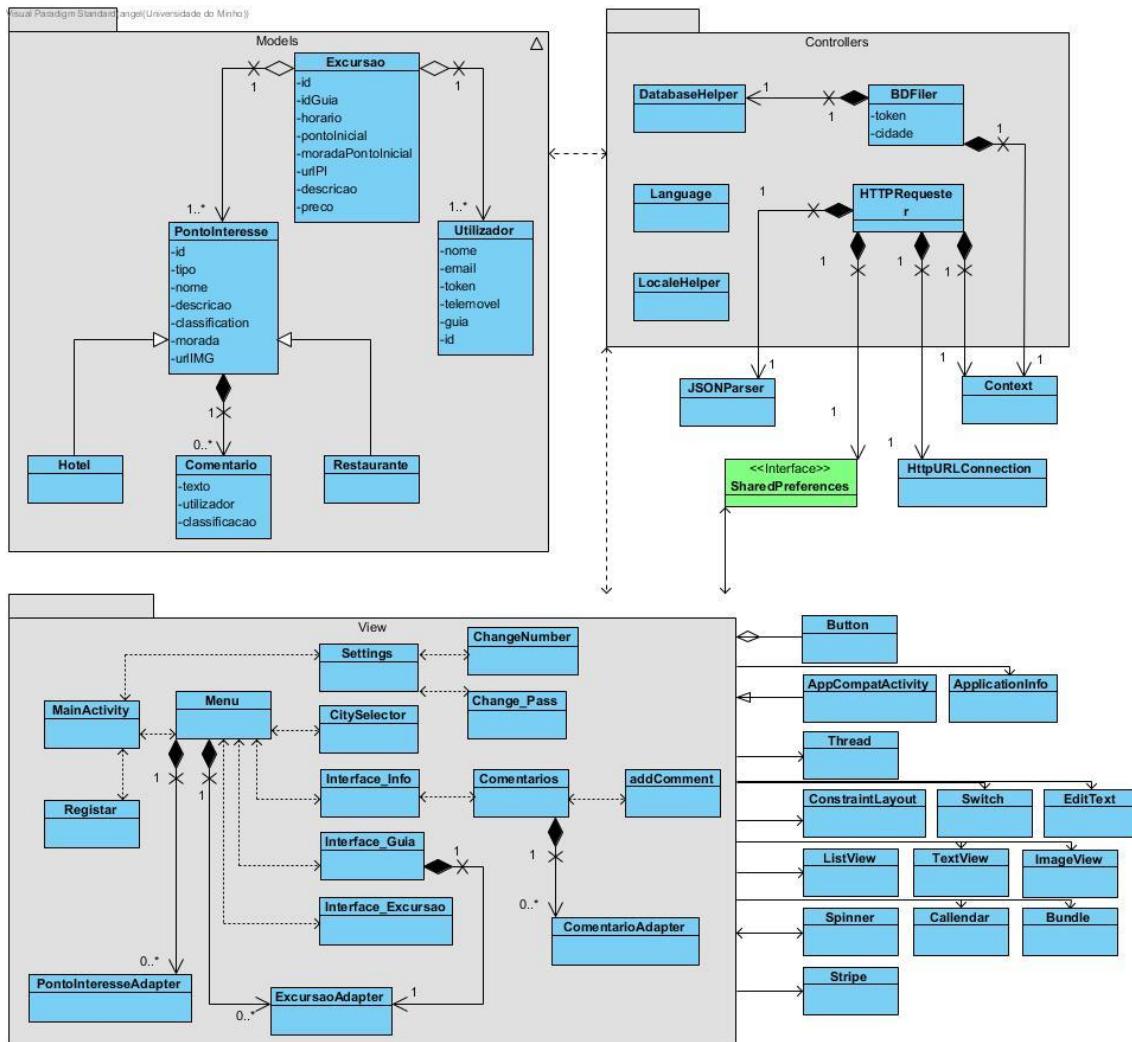


Figura 2 - Diagrama FrontEnd Final

A figura 3, é já uma versão final de um diagrama de classes que representa, neste caso, o ambiente *FrontEnd*. Já se encontram definidas, neste diagrama, todas as classes usadas, assim como um modelo MVC estruturado.

A certo ponto, após definirmos quais as mais importantes entidades do sistema e seus atributos, construímos um modelo lógico da Base de Dados a implementar, que, na altura, consideramos ser o ponto de partida para aquilo que viríamos a implementar, mas sempre cientes que este evoluiria.

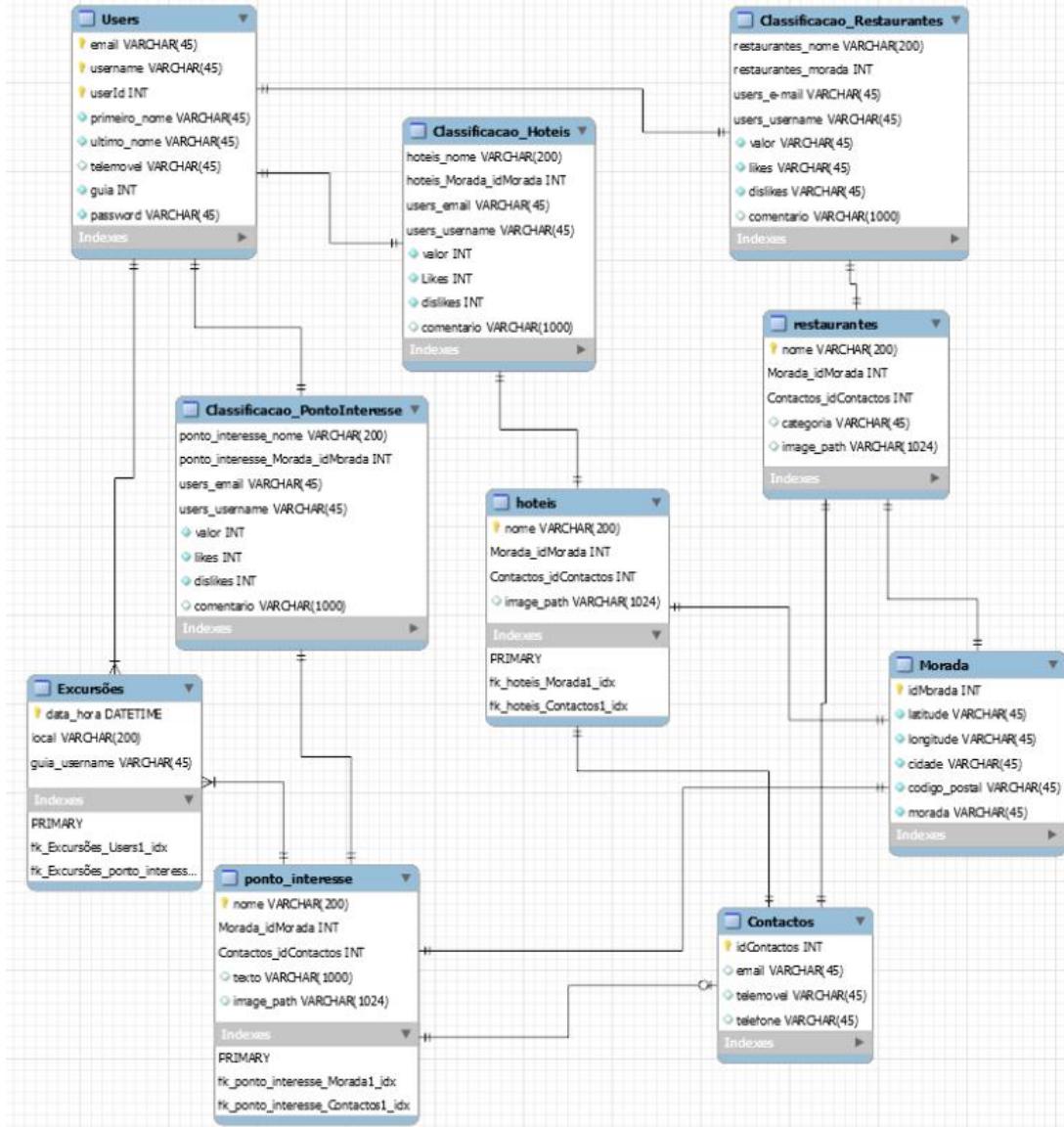


Figura 3 - Modelo lógico da Base de Dados

Para além destes modelos e diagramas, foram, ainda, especificados diversos *Use Cases* (que se encontram nos documentos anexos a este relatório), que descreveriam as funcionalidades do sistema, assim como as condições a que este deveria estar sujeito para que estas pudessem ser aplicadas, e como um utilizador do sistema interagia com o mesmo. Simultaneamente, foi desenvolvido o diagrama de *Use Cases* apresentado na figura abaixo.

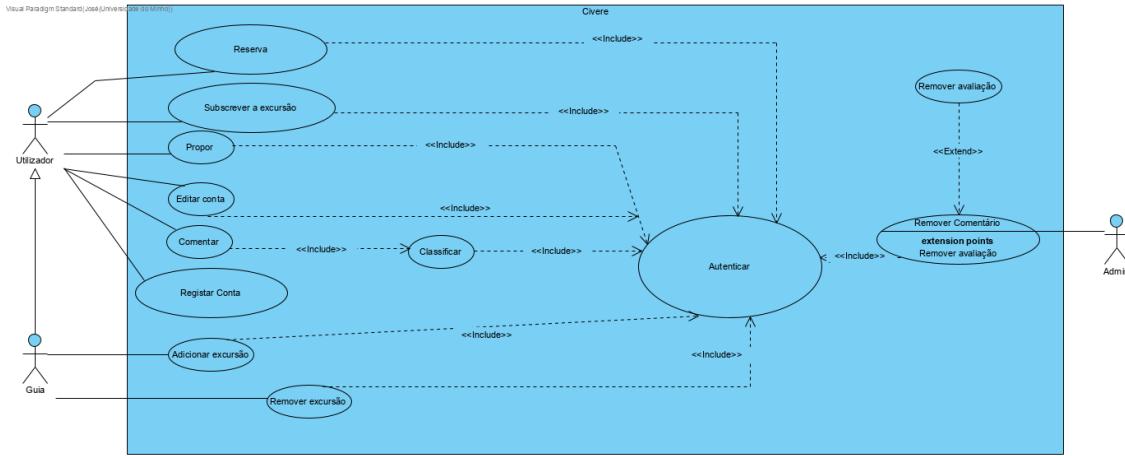


Figura 4 - Diagrama de Use Cases inicial

É de notar que as funcionalidades disponíveis no sistema desenvolvido, apesar de serem fortemente baseadas nestas especificações, não correspondem, na globalidade, à sua especificação inicial.

Após a modelação inicial das camadas lógicas do sistema, começaram também a ser modeladas as interfaces gráficas da aplicação através das ferramentas já anteriormente mencionadas, Adobe XD e Picasso, que são ambas muito compatíveis com o tipo de aplicação que viríamos a desenvolver. Esta modelação gráfica (ANEXO D), serviu não só como um excelente ponto de partida para o desenvolvimento final das interfaces gráficas usadas na aplicação, que poderão ser vistas com mais detalhe na seção dos resultados finais, mas também numa melhor compreensão de como um utilizador da aplicação viria a interagir com o sistema.

IMPLEMENTAÇÃO DA BASE DE DADOS

Umas das decisões essenciais do nosso projeto passava pela forma como seriam guardados os dados da aplicação, relativos a atores do sistema da mesma e todas as suas funcionalidades, de forma a cumprir com os objetivos a que nos propusemos, nomeadamente a centralização da informação.

Para o efeito, aplicando conhecimento adquirido previamente acerca do assunto e através de pesquisa feita para esta situação em específico, começamos por pesar os prós e os contras entre utilizar bases de dados *SQL* e *NoSQL*. Apesar das bases de dados não relacionais serem altamente eficientes para ambientes nos quais são intensamente explorados relacionamentos entre entidades, este não seria o caso da aplicação a desenvolver, pelo que, rapidamente, nos “inclinamos” para as relacionais, também por estas serem mais usadas no “mundo real”. Pesou também na nossa decisão, o facto de estas serem reconhecidas por funcionarem bem com dados previsíveis e poderem ser organizadas de forma tabelar, com *queries* simples, como previmos inicialmente que se iria verificar no nosso projeto (e se viria a confirmar), assim como o facto da linguagem *SQL* estar disponível para *Linux*, *Windows* e *Mac* e ter conectores para linguagens como *Ruby*, *C#*, *C++*, *Java*, *Python*, *PHP*, *Kotlin*, etc., que nos abriria muitas possibilidades. Outro fator importante foi a nossa ideia, desde o início, da necessidade de guardar imagens na base de dados, que, segundo as nossas pesquisas, seria consideravelmente mais simples numa base de dados relacional em relação a outra não-relacional, como, por exemplo, em *Neo4J*. Por motivos de rotina e hábito, optamos por utilizar a plataforma *MySQL*.

Partindo da modelação inicial, a base de dados sofreu muitas alterações ao longo do projeto, acompanhando a evolução dos seus restantes componentes. Inicialmente, e como mostrado anteriormente, começamos por implementar uma base de dados com os elementos essenciais do sistema e, a partir dela, fomos realizando alterações e/ou adições consoante a necessidade de cada funcionalidade a implementar. Como nenhuma tabela construída inicialmente na modelação do projeto foi removida ao longo do desenvolvimento da aplicação (apenas sofreram alterações), passar-se-á, de seguida, a explicar os motivos da implementação de cada uma delas na sua forma final, contextualizando-as.

Tabela *Users*

Como decidido inicialmente, a aplicação teria, obviamente, a possibilidade de criação de contas de utilizador, pelo que foi imperativo a implementação de uma tabela para o efeito. Estes utilizadores podem ser, ou não, guias, o que irá, mais à frente, alterar as suas possibilidades de utilização dentro da aplicação. Nesta tabela, como se pode ver na figura abaixo, colocamos todos os atributos que definem um utilizador e são pedidos aquando do seu registo, à exceção do inteiro *user_id*, que nos facilita na manipulação dos dados, através da atribuição desse identificador único; e do atributo binário *guia*, que, para o valor 0, identifica contas de utilizador normal e, no caso de ser 1, as contas dos guias.

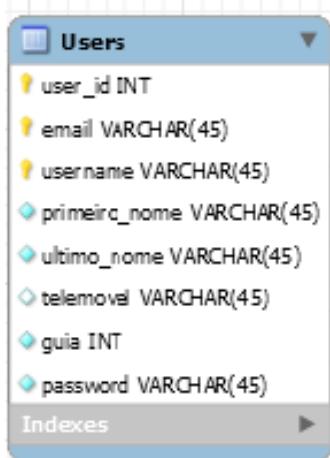


Figura 5 - Tabela *Users*

Tabelas *Carteira_Guia*

No caso de um utilizador ser também um guia turístico, é-lhe automaticamente atribuída uma “carteira” que contém o seu saldo por levantar, resultante das excursões por ele realizadas.



Figura 6 – Tabela da *Carteira de guia*

Tabelas *Pedidos_Payout*

Como a CIVERE permite que um guia levante esse saldo que se encontra na sua Carteira, implementamos uma tabela que guarda informação acerca desses pedidos de levantamento, nos quais nos são fornecidos o valor a levantar e o IBAN para o qual devemos transferir o dinheiro. Isso, de momento, é feito manualmente, assim como a passagem do inteiro *Pago* de 0 para 1, conforme a transferência estiver concluída ou não, respetivamente, de forma a termos um melhor controlo das transações feitas.

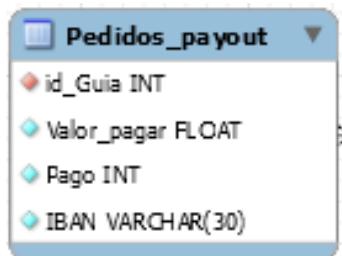


Figura 7 - Tabela de Pedidos de Payout

Tabela *Contactos*

Como meio de generalização da informação, achamos por bem criar uma tabela que contivesse toda a informação relativa aos contactos dos elementos do nosso sistema, nomeadamente restaurantes, hotéis e pontos de interesse das cidades, como se mostra mais à frente. Nesta tabela, incluímos alguns dados básicos, para além de um identificador único *idContactos*, que nos permite relacioná-la com a devida entidade.

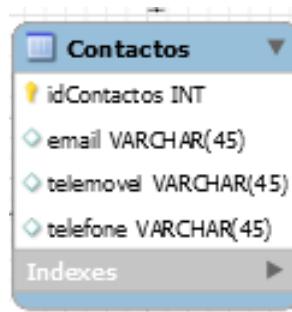


Figura 8 - Tabela de Contactos

Tabela Morada

À semelhança da tabela anterior, também foi criada uma tabela relativa aos dados da morada para esse mesmo tipo de entidades. Nesta tabela, para além de um identificador único *idMorada*, que, tal como no caso anterior, nos permite relacioná-la com a devida entidade, incluímos informação acerca das suas coordenadas, cidade, código postal e morada específica.

The screenshot shows the MySQL Workbench interface with the 'Morada' table selected. The table has the following structure:

!	<i>idMorada</i> INT
◆	<i>latitude</i> VARCHAR(45)
◆	<i>longitude</i> VARCHAR(45)
◆	<i>cidade</i> VARCHAR(45)
◆	<i>codigo_postal</i> VARCHAR(45)
◆	<i>morada</i> VARCHAR(45)

Below the table structure, there is a section labeled 'Indexes'.

Figura 9 - Tabela de Morada

Tabelas *restaurantes*, *hoteis* e *ponto_interesse*

Enquanto elementos fundamentais do nosso projeto, construímos tabelas relativas aos Hotéis, Pontos de interesse e Restaurantes, todos eles com um identificador único (*id*). Como atributos, cada um tem alguns elementos básicos, dos quais se destacam a morada e os contactos, identificados através de um identificador como explicitado anteriormente, uma classificação média calculado através de um *trigger* e também, por exemplo, um *image_path*, referente a uma imagem descriptiva desse elemento, guardada na forma de *URL*.

The screenshot shows the MySQL Workbench interface with three tables side-by-side: 'ponto_interesse', 'hoteis', and 'restaurantes'. Each table has the following structure:

!	<i>id</i> INT
◆	<i>nome</i> VARCHAR(200)
◆	<i>classificacao_media</i> FLOAT
◆	<i>Morada_idMorada</i> INT
◆	<i>Contactos_idContactos</i> INT
◆	<i>image_path</i> VARCHAR(1024)
◆	<i>descricao</i> VARCHAR(1024)

Below each table structure, there is a section labeled 'Indexes'.

Figura 10 - Tabelas das Entidades Ponto de interesse, Hotéis e Restaurantes

A nossa ideia inicial seria construir uma tabela que representasse simultaneamente os 3 casos, distinguindo-os através de um inteiro, por exemplo. Rapidamente abandonamos essa abordagem, porque vimos vantagens em tratá-los separadamente, no sentido de nos permitir ter mais identificadores únicos disponíveis e de, como podemos ver na figura, possuírem atributos distintos entre si, facilitando também, assim, trabalho futuro que venha a requisitar mais atributos distintos para algum dos casos.

Tabelas *Classificacao_Restaurantes*, *Classificacao_Hoteis* e *Classificacao_PontoInteresse*

A única desvantagem advinda da decisão de separar as tabelas anteriores, para cada um dos casos, apareceu quando foi necessário relacionar, a cada uma delas, um conjunto de classificações dadas pelos utilizadores da CIVERE. Assim, construímos 3 tabelas iguais, uma para cada caso, que permitem guardar um valor classificativo de 1 a 5 e um comentário (opcional), feito por um utilizador acerca de um elemento específico, reconhecidos através dos seus identificadores únicos.

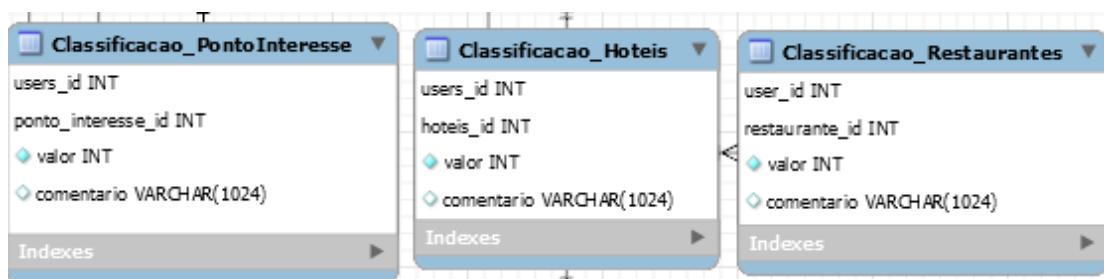


Figura 11 - Tabelas de Classificações

Tabelas Excursões

Tal como explicado anteriormente, uma das funcionalidades mais interessantes do nosso programa é a disponibilização de excursões/visitas guiadas dentro de uma cidade, feitas por um utilizador guia. Nesse sentido, foi necessário guardar os dados relativos a essas atividades, atribuindo-lhes, também, um identificador único, um horário e data de início, o local de partida ou a visitar, um preço correspondente e uma descrição feita pelo guia, na qual pode incluir observações que considere pertinentes. Cada excursão está também relacionada ao guia que a criou, através do seu identificador único.

O diagrama mostra a estrutura da tabela 'Excursões'. A estrutura é a seguinte:

id	INT
data_hora	DATETIME
ponto_interesse_id	INT
guia_id	INT
preço	FLOAT
descrição	VARCHAR(1024)

Na parte inferior, há uma barra azul com o rótulo 'Indexes' e um ícone de seta apontando para a direita.

Figura 12 - Tabela de Excursões

Tabelas Inscrição

Obviamente, e por motivos de interatividade, a aplicação permite que qualquer utilizador se inscreva numa excursão que ainda não tenha iniciado. Para que tal informação seja guardada, construímos uma pequena tabela que relaciona, também através dos seus identificadores inteiros únicos, uma excursão com um utilizador, assim como um identificador alfanumérico gerado pelo *Stripe*, respetivo ao pré pagamento realizado.

O diagrama mostra a estrutura da tabela 'Inscrição'. A estrutura é a seguinte:

excursão_id	INT
user_id	INT
pagamento_id	VARCHAR(45)

Na parte inferior, há uma barra azul com o rótulo 'Indexes' e um ícone de seta apontando para a direita.

Figura 13 - Tabela de Inscrição

Tabelas *Versoes_BD*

Uma das capacidades mais complexas da nossa aplicação foi o controlo de versões que passou por distingui-lo em 2 tipos diferentes: controlo de versão estrutural da base de dados e controlo das alterações/inserções de dados em alguma tabela. Nesse sentido, quanto ao primeiro, decidimos que na Base de Dados seria apenas necessário guardar, manualmente, a informação relativa ao nome de cada versão, assim como a sua data de lançamento, verificando-se, ao iniciar, se as versões locais da aplicação são compatíveis com a última versão presente nesta tabela.

Versoes_BD	
Yellow icon	Versao VARCHAR(25)
Blue diamond icon	ReleaseDateTime DATETIME
Indexes	

Figura 14 - Tabela *Versoes_BD*

Tabelas *Versoes_Tabelas*

Quanto ao segundo tipo de controlo de versões, respetivo a cada tabela, a nossa implementação apenas necessitou de guardar informação referente à última adição/alteração de dados que nessa tabela tenha ocorrido, como podemos ver na figura abaixo. Esta tabela tem apenas uma linha, mas que é constantemente alterada, como explicado mais à frente.

Versoes_Tabelas	
Blue diamond icon	Users DATETIME
Blue diamond icon	Morada DATETIME
Blue diamond icon	Contactos DATETIME
Blue diamond icon	restaurantes DATETIME
Blue diamond icon	ponto_interesse DATETIME
Blue diamond icon	hoteis DATETIME
Blue diamond icon	Excursões DATETIME
Blue diamond icon	Inscrição DATETIME
Blue diamond icon	Classificacao_Restaurantes DATETIME
Blue diamond icon	Classificacao_PontoInteresse DATETIME
Blue diamond icon	Classificacao_Hoteis DATETIME
Blue diamond icon	Carteira_Guia DATETIME
Blue diamond icon	Pedidos_Payout DATETIME

Figura 15 - Tabela *Versoes_Tabelas*

Triggers

De forma a automatizar alguns processos referentes à base de dados, decidimos implementar alguns *triggers* que nos pareceram relevantes. Ao longo da evolução do projeto foram surgindo novas ideias e funcionalidades, que foram acompanhadas, como explicado atrás, pela evolução da Base de Dados. Nesse sentido, foram realizados 3 tipos de *triggers*.

O primeiro, aquando da disponibilização da funcionalidade de comentar e classificar os vários locais presentes na aplicação, para que fosse recalculada a classificação média de uma entidade específica, sempre que surja uma nova entrada nas tabelas de classificação. Como estas são 3, uma para cada um dos tipos de locais considerados, foi, então, implementado um *trigger* para cada uma delas.

Posteriormente, após decidirmos os tipos de controlo de versões a concretizar, ficou clara, para o controlo das versões das tabelas, a necessidade de verificar todas as ocorrências de entradas e alterações de dados nas mesmas. Para tal, construímos 2 *triggers* para cada tabela (um para alterações, outro para inserções) que atualizem a linha da tabela *Versoes_Tabelas* na devida coluna, correspondente à tabela alterada, como mostrado na secção anterior.

Finalmente, decidimos automatizar a criação de contas, através de um *trigger* ativado após a criação de um novo utilizador, que verifica se este pretende ser um guia ou não. Em caso afirmativo, é criada automaticamente uma Carteira com o seu identificador único e com um saldo inicial nulo, obviamente.

IMPLEMENTAÇÃO DO BACKEND

Devido à sua enorme compatibilidade, decidimos desenvolver o nosso *BackEnd* numa tecnologia *Microsoft* e optamos por desenvolver esta parte fulcral do nosso trabalho na linguagem *C#*.

Para tal, usamos o *VisualStudio* e desenvolvemos um projeto baseado em *Asp.net core web Application*, usando **REST API**. Escolhemos esta tecnologia, pois permite fazer a ligação *FrontEnd* ↔ *BackEnd* através de pedidos *HTTP*, e cada *HTTP* contém toda a informação necessária para funcionar, não havendo necessidade do lado do servidor nem do lado do cliente de ter conhecimento das conexões anteriores. Outra das razões, entre outras, que nos levou a escolher *REST API*, é a separação que esta tecnologia faz entre o cliente e o servidor, pois separa completamente os utilizadores do local onde se armazenam os dados.

O *REST* também tem a vantagem de permitir sempre uma mais fácil evolução do design da *API* e esta foi uma das funcionalidades que mais apreciamos neste projeto, visto que apenas fomos desenvolvendo *models* e *controllers* à medida que fomos precisando deles. No entanto, sempre tivemos o *BackEnd* disponível para ser utilizado através de *controllers* como *login* e *registar*, que já se encontravam implementados aquando da primeira vez que demos *deploy* ao *BackEnd*.

Outra das razões que nos levou a escolher este tipo de tecnologia, foi o facto de permitir desenvolver vários *Controllers*, que possibilitam responder às diversas necessidades do *FrontEnd*.

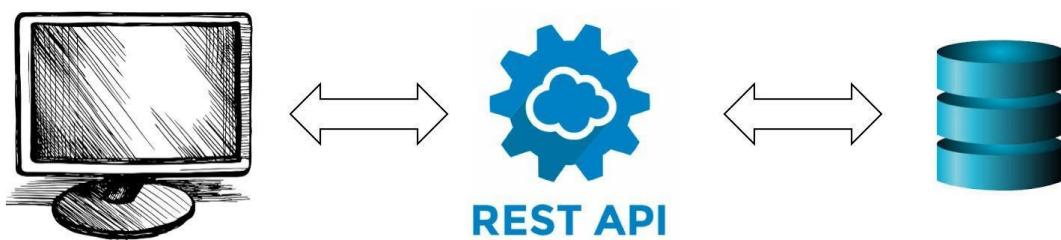


Figura 16 - Modelo REST API

Em todos os nossos controladores, apenas usamos o método *POST*, contendo como mensagem, tanto do *FrontEnd* para o *BackEnd* como no sentido inverso, **Json Objects**. Optamos por apenas utilizar este método por ser mais seguro do que, por exemplo, um *GET*.

De seguida, passa-se a explicitar a arquitetura do *BackEnd*, que se encontra dividida em *Controllers* (que lidam diretamente com os pedidos dos clientes) e *Models* (parte do *BackEnd* que contém os algoritmos, inclusive os de acesso à Base de Dados).

Controllers

- *Register*: Este controlador recebe toda a informação necessária para a criação de um novo utilizador, devolvendo 1 caso tudo corra corretamente. Caso não seja possível criar um utilizador, é devolvido ao *FrontEnd* um 0;
- *Login*: Sempre que um utilizador se quiser autenticar, este é o controlador a contactar. Vai receber a informação de *login* e, se esta estiver correta, vai devolver parte da informação desse utilizador, junto com um *token* e um inteiro representativo da sua validade;
- *EditAccount*: Tal como o nome indica, este controlador permite alterar a conta de um utilizador, sendo que este pode alterar apenas a sua palavra-passe ou o seu número de telemóvel. Retorna 1 se bem sucedido;
- *Images*: Este controlador devolve, no formato de *Json Object*, um inteiro que representa a validade do *token*, um novo *token* caso este seja válido e um *array* representativo de Hotéis/Pontos de Interesse/Restaurantes, ou seja, um array que contém as informações de todos estes, desde que de uma certa cidade. Neste controlador, é feito o controlo de versões das tabelas, verificando a *timestamp* que se encontra na tabela *SQLite* do *user*, sendo que apenas é devolvida a informação requerida caso esta esteja desatualizada. Caso contrário, é apenas enviado um elemento no *array*, que contém um sinal no campo *date*, informando o *FrontEnd* de que já tem a versão atualizada daquilo que pediu ao *BackEnd*;
- *Excursões*: Criamos este controlador para os guias criarem excursões, devolvendo um 1 para confirmar a inserção da excursão. Aos guias é permitido escolher onde vai ser a excursão, o preço e o horário em que se vai realizar;
- *GetExcursões*: Método que fornece, em formato *Json Object*, todas as excursões ainda por realizar de uma cidade;
- *Payment*: Método de inscrição nas excursões. Neste *controller*, é primeiro realizado o pagamento da excursão e só depois é feita a inserção na tabela de inscrições da Base de Dados. Devolve um inteiro -3 no caso de o pagamento ser mal sucedido e 1 se for bem sucedido e a inscrição for realizada. Optamos por não permitir que um utilizador, após inscrito, não possa desistir de uma excursão, de forma a proteger os guias;

- *Payout*: Método de pedido de *cashout* de um guia. O guia fornece o seu *IBAN* e um valor a levantar. É verificado se é possível levantar esse montante e, se sim, fica registado um pedido de *cashout* na Base de Dados para posterior processamento.
- *RemoveExcursão*: Controlador que permite a um guia remover uma excursão, removendo também todos os inscritos da excursão a ser retirada, efetuando o *refund* do valor pago a cada utilizador, usando o *id* do pagamento de cada um dos inscritos;
- *Saldo*: Este controlador serve para atualizar o saldo de um guia, atualizando-o na Base de Dados. Verifica-se todas as excursões de um guia que já foram realizadas e calcula-se o valor a acrescentar ao seu saldo, conforme os inscritos na mesma. Ao calcular este valor, multiplica-se por 0.9, de forma a obtermos um lucro de 10% de cada inscrição realizada pela aplicação. Após efetuar este cálculo, são retiradas as inscrições consideradas da Base de Dados, de forma a que não existam repetições em cálculos futuros;
- *newComment*: Este controlador existe para que os utilizadores tenham a opção de realizar um comentário num Hotel/Ponto de Interesse/Restaurante. Sempre que o mesmo quiser realizar um segundo comentário na mesma entidade, este vai substituir o comentário anterior por ele feito, de forma a impedir interferências indevidas no cálculo das médias de classificação;
- *Comments*: Método que devolve todos os comentários de um determinado Hotel/Ponto de Interesse/Restaurante, em formato *Json Object*;

```
{
    "Email" : "email_teste@gmail.com",
    "Pass" : "teste"
}
```

Figura 17 - Exemplo de Post Request (Talend API Tester)

```
[
  valido: "1",
  token: "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOiE1OTM2OTAyMjN9.jyoXOT5SZ_xbEzkqVwMpfPXUiDiX7fnz0mg8wovl4J8",
  Userinfo: {
    Nome: "escuteiro",
    Email: null,
    Primeiro_nome: "Arnaldo",
    Segundo_nome: "Silva",
    Telemovel: "912345567",
    Pass: null,
    Guia: "1",
    Token: null,
    valido: 1,
    id: "1"
  }
]
```

Figura 18 - Exemplo de resposta em formato *Json Object* (Talend API Tester)

De forma a garantir mais segurança para a nossa aplicação, quando cada utilizador faz *login*, é lhe fornecido um *token*. Após isto, em todas as interações *FrontEnd* ↔ *BackEnd*, o utilizador vai enviar, juntamente com a sua mensagem, esse *token*, cuja validade terá sempre a duração de uma hora. Sempre que um utilizador fizer um *request* de um conjunto de Hotéis/Pontos de Interesse/Restaurantes ou sempre que for realizado um comentário, este *token* vai ser renovado, garantindo mais uma hora de utilização. Quando um *token* surgir inválido, de forma a uniformizar o *BackEnd*, reservamos o valor 0, ou seja, sempre que um controlador retornar um « “válido”: “0” » (formato *Json*) ou um 0 literal, o *FrontEnd* vai ter necessariamente de pedir um novo *login*.

Models

Os *models* são uma das partes fulcrais do *BackEnd*, pois, enquanto os *controllers* recebem os pedidos dos *users*, os *models* vão ser os algoritmos que vão conseguir responder a estes pedidos e, em relação aos *models*, há um que se destaca dos demais, pois é nesse que é feita a ligação com a nossa Base de Dados, o *model BD*.

Model BD

Sempre que este *model* é inicializado, é aberta uma ligação com a Base de dados, usando ***MySqlConnection Class***. Esta classe, permite-nos aceder e manipular todos as entradas da nossa base de dados. Sempre que é aberta uma ligação à *BD*, é porque há alguma função deste *model* a ser usada e, por esse motivo, no fim de cada uma das funções que possa ser chamada por um *controller*, a ligação com a *BD* é fechada, o que leva a que em certos *controllers* exista mais do que uma ligação à *Base de Dados*. Isto acontece, geralmente, pois um pedido pode necessitar de duas conexões, devido ao facto de a execução da segunda ligação ser dependente dos resultados obtidos na primeira. Outra das nossas preocupações foi impedir ***SQL Injections***. Para este efeito, damos sempre *match* a cada campo que seja fornecido pelo utilizador, usando ***MySqlCommand Paramaters***.

```
String sql = "INSERT INTO app.users(username, `email`,password," +
" primeiro_nome, ultimo_nome, guia,telemovel) VALUES(@Nome,@Email,@Pass,@N1,@N2,@Guia,@Tel)";
MySqlCommand cmd = new MySqlCommand(sql, conn);
cmd.Parameters.AddWithValue("@Nome", nome);
cmd.Parameters.AddWithValue("@Email", mail);
cmd.Parameters.AddWithValue("@Pass", pass);
cmd.Parameters.AddWithValue("@N1", n1);
cmd.Parameters.AddWithValue("@N2", n2);
cmd.Parameters.AddWithValue("@Guia", guia);
cmd.Parameters.AddWithValue("@Tel", tel);
```

Figura 19 - Métodos de segurança relativos a SQL Injection

Quanto aos restantes *models*, com exceção do *Jwtmodel*, todos os *models* acabam por ser apenas estruturas onde armazenamos, recebemos e enviamos dados.

Model Jwtmodel

Como já referimos acima, *JSONWEBTOKEN(JWT)* é usado para transferir, de forma segura, informação na *Web*, e portanto, surgiu a necessidade de desenvolver este *model* que, embora apenas disponibilize as funções *jwtcheck* e *jwtcreate*, fornece uma maior segurança à aplicação e aos dados dos nossos *users*.

A função *jwtcreate* é chamada sempre que um utilizador faz *login*, fornecendo ao *user* um *token* com validade de uma hora e, também, sempre que é acedido conteúdo da *BD* com um *token* válido, e neste caso, a *jwtcreate* faz a revalidação do *token*.

Por sua vez, a função *jwtcheck* vai verificar se o *token* é válido, isto é, se foi gerado pela nossa aplicação e se ainda se encontra dentro da sua hora de validade.

```
"token" : "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOiE1OTM3Nzk4ODN9.Y6F4ELIb5apeuhAc07WjmSONAtOFF6xWYK1G3iIMTdM"
```

Figura 20 - Exemplo de Token enviado pelo FrontEnd para ser validado

Deployment

De forma a permitir um acesso global à nossa aplicação, demos *deploy* tanto do *BackEnd* como da Base de dados para a plataforma do *Azure*, tirando proveito do crédito inicial oferecido pela *Microsoft* para o efeito.

Escolhemos esta opção, pois o *Azure* garante uma enorme segurança, escala global que nos pode permitir chegar a vários países, integração com *Visual Studio* e permite dar *deploy* de projetos juntamente com as suas Bases de Dados. Outra das razões que nos levou a implementar a nossa aplicação com *Azure* foi capacidade de poder ter um poder de computação dinâmico, logo, neste momento, estamos com as versões base do servidor e da base de dados pois, temos poucos utilizadores e uma base de dados relativamente pequena. No entanto, pensamos que este projeto pode vir a crescer e, se isso acontecer, podemos aumentar a velocidade de resposta do *server* e da Base de Dados e aumentar a capacidade da *BD*.

IMPLEMENTAÇÃO DO FRONTEND

Para que os utilizadores possam interagir com os dados do sistema, surgiu a necessidade de criação de uma camada de apresentação, o *FrontEnd*. Assim decidimos desenvolver uma aplicação móvel *Android*, que concluímos ser o mais adequado, com base nos requisitos do nosso sistema, com recurso à utilização do *AndroidStudio*.

Passamos, agora, à explicação da arquitetura implementada no nosso sistema, que segue um modelo *MVC (Models View Controllers)*.

Models

- **Utilizador:** Classe que representa um utilizador do sistema e toda a informação representativa do mesmo;
- **PontoInteresse:** Classe que representa um ponto de interesse presente no sistema;
- **Hotel:** Classe que representa um hotel no sistema, deriva diretamente de PontoInteresse;
- **Restaurante:** Classe que representa um restaurante no sistema, deriva diretamente de PontoInteresse;
- **Excursao:** Classe que representa uma excursão no sistema e toda a respetiva informação da mesma;
- **Contactos:** Classe que representa os contactos de um utilizador ou ponto de interesse do sistema. Estes podem ser Email, telemóvel e telefone;
- **Comentario:** Classe que representa um comentário efetuado por um utilizador do sistema relativamente a um ponto de interesse;

View

- **MainActivity:** Classe que representa a *main activity* da app. Esta é a primeira a aparecer no ecrã quando a *app* é iniciada. Sendo que a primeira interface a aparecer permite ao utilizador fazer *login*, se não existir uma sessão guardada com *token* válido, caso contrário, a interface *login* não é mostrada no ecrã e passa-se imediatamente para a *activity menu*. É também na *main activity* que é iniciada a base de dados *SQLite* e onde é criada uma *Thread* com a classe *BDFiller*, responsável pelo carregamento de informações do *server* para a Base de Dados. É importante frisar que, mesmo quando passamos de imediato para a *activity menu*, a Base de Dados é iniciada e a *Thread* com a *BDFiller* percorrida;
- **Menu:** Classe onde é iniciado o menu inicial. É aqui onde se encontram os botões que permitem navegar entre os hotéis, restaurantes, pontos de interesse e excursões;
- **AddComment:** Classe onde se encontra o método responsável por enviar um novo comentário para o servidor;
- **Change_Pass:** Classe onde se encontra o método responsável por enviar um pedido de troca de password ao servidor;
- **ChangeNumber:** Classe onde se encontra o método responsável por enviar um pedido de troca de número de telemóvel ao servidor;
- **interface_Excursao:** Classe onde se encontram os métodos de pagamento e gestão das excursões. A função responsável por enviar o pedido de pagamento é a função *pay*. Temos, ainda nesta classe, a função *cartao* com a finalidade de processar os dados do cartão introduzido pelo cliente. Para além disso, existe a função *remove*, que envia um pedido de remoção de excursão e a função *getExcursao* que trata do processo de inscrição numa excursão;
- **CitySelector:** Classe composta por botões que representam as cidades suportadas pela aplicação. Sendo que, ao clicar num destes botões, são eliminados os dados da cidade previamente carregada e todos os dados posteriormente carregados são da cidade escolhida;
- **Comentarios:** Classe que carrega os comentários de um hotel, restaurante ou pontos de interesse e que, consoante o botão clicado pelo cliente, inicia a *activity* de adicionar um novo comentário, ou retrocede para a *activity* anteriormente visitada (menu inicial com hotéis, restaurantes, pontos de interesse ou excursões selecionadas);

- **interface_Guia:** Classe que inicia a *activity interface_guia*. Dentro desta classe, encontram-se botões que permitem passar para outras funcionalidades, como adicionar uma excursão. Para além disso, temos também botões que permitem interagir com *widgets*, tais como a *Wallet*;
- **Interface_Info:** Classe que inicia a *activity_info* e que, consoante o botão clicado, carrega os comentários ou retorna à *activity* previamente visitada;
- **PontoInteresseAdapter:** Classe com os métodos necessários para criar uma *row*. Esta *row* criada é uma *view* feita para representar um ponto de interesse, um hotel ou um restaurante;
- **ExcursaoAdapter:** Classe idêntica ao *MyAdapter*, mas que, em vez de receber um ponto de interesse, hotel ou restaurante, recebe uma excursão;
- **Registar:** Classe onde se encontra o método de registo e o método de envio de uma tentativa de registo para o servidor;
- **Settings:** Classe que inicia a interface *activity_settings*, onde se encontram as diferentes configurações da aplicação e o botão que permite aos utilizadores dar sugestões.

Controller

- **BDFiler:** É a classe responsável por carregar e atualizar toda a informação disponível na base de dados SQL Lite;
- **DatabaseHelper:** É a classe responsável pela criação das tabelas da base de dados SQLite, assim como da inserção de nova informação nas tabelas;
- **HTTPRequester:** É a classe que trata de estabelecer a conexão entre os clientes e o servidor do sistema, assim como de tratar e realizar todos os pedidos HTTP;
- **Language:** É a classe responsável por definir que linguagem é usada na aplicação.
- **LocaleHelper:** Esta classe, juntamente com a classe *Language*, é responsável pela alteração da linguagem na aplicação.

RESULTADOS FINAIS

Interfaces

Ao longo do projeto, uma das principais preocupações foi o grau de dificuldade de utilização da aplicação. Desta forma, todas as interfaces realizadas estão pensadas para tornar a aplicação o mais apelativa e interativa possível. Desse modo, explicita-se, nesta secção, as interfaces desenvolvidas, assim como alguns diagramas que permitem mostrar o modo de operação da CIVERE.

De seguida, apresentam-se as interfaces de *Login*, onde o utilizador se pode autenticar com as suas credenciais ou aceder à interface de registo.

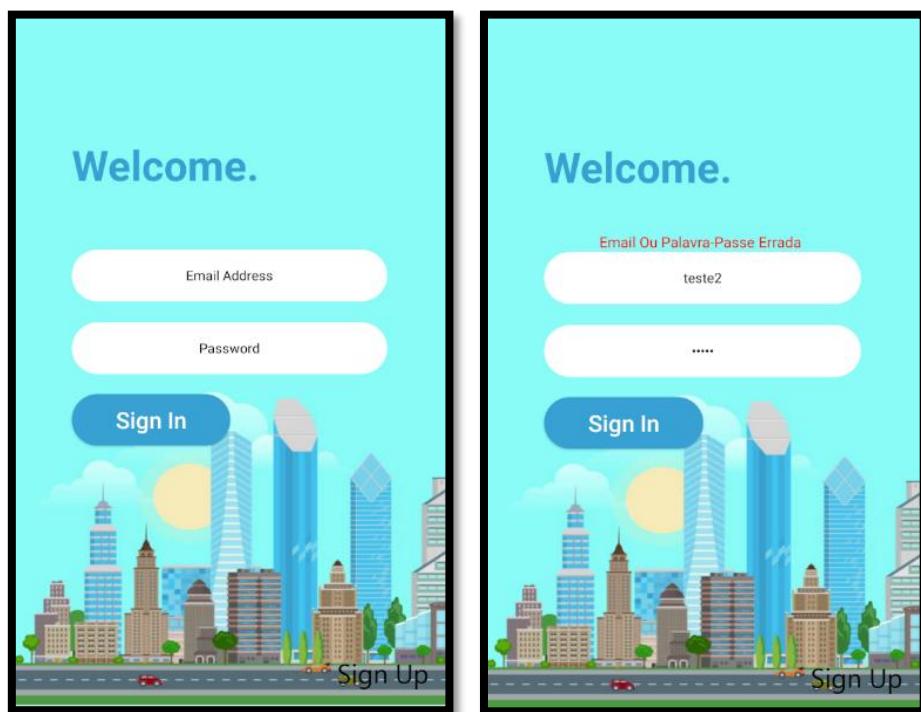


Figura 21 - Interfaces Login

Caso o utilizador tente autenticar-se, acontece uma, de duas situações possíveis: ou é notificado que o *email* ou *password* estão errados, ou a autenticação é feita com sucesso.

Para a que um utilizador possa fazer *login*, é necessário que este se registe e, para isso, desenvolvemos a interface de registo, onde o utilizador coloca as suas informações para criação de conta, inclusive se pretende ser, ou não, um guia.

The figure consists of two side-by-side screenshots of a mobile application's sign-up screen. Both screens have a black header bar with the text "Sign Up" in white. Below the header are seven input fields, each with a blue rounded rectangle background and a thin blue border. The fields are labeled: "First Name", "Last Name", "Nick Name", "Email Address", "Password", "Confirm Password", and "Phone Number". At the bottom of the screen are two buttons: a white button with a black outline containing the text "Want to be Guide?" and a blue button with white text containing the word "CONFIRM".

The left screenshot shows a standard sign-up form. The right screenshot shows the same form but with validation errors. Above the "Password" field, there is a red error message that reads "Palavras passas não correspondem" (Passwords do not match). The "Want to be Guide?" checkbox is checked in the bottom-left corner of the right screenshot.

Figura 22 - Interface registo e Interface registo com campos inválidos

Caso alguma informação colocada seja inválida, o utilizador é notificado, como se pode ver no exemplo da direita.

Após um *login* bem sucedido, é apresentado um menu inicial onde se encontram botões que nos permitem navegar entre hotéis, restaurantes, pontos de interesse e excursões (barra inferior), menu de definições (canto superior direito) e o selecionador de cidades (canto superior esquerdo).

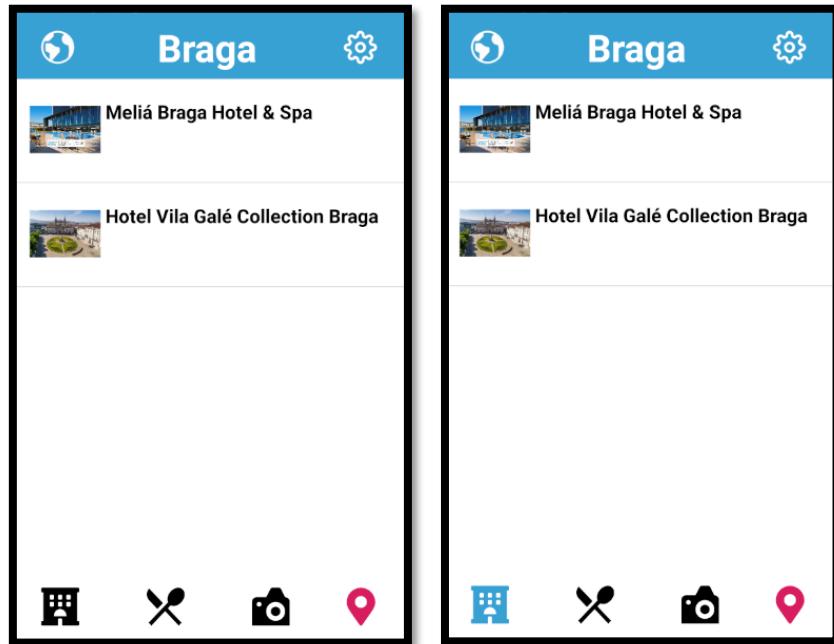


Figura 23 - Menu inicial (à esquerda) e Menu inicial com os hotéis selecionados (à direita)

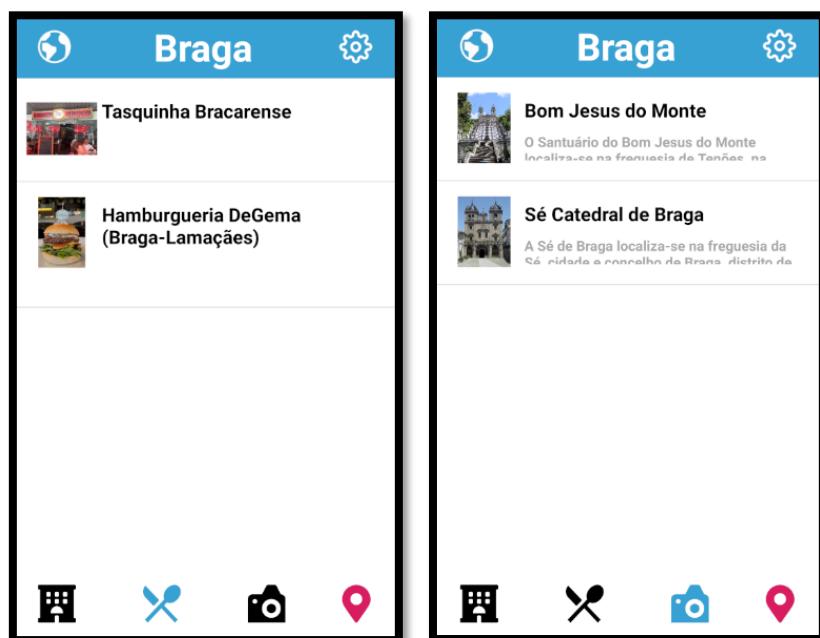


Figura 24 - Menu inicial com os restaurantes selecionados (à esquerda) e Menu inicial com os Pontos de Interesse selecionados (à direita)

A interface das excursões é diferente, consoante se trata de um utilizador normal ou de um guia. Aqui, o guia tem acesso às suas excursões, tendo a possibilidade de as editar, remover ou adicionar. Para além disso, é também nesta interface que o guia tem acesso à sua *Wallet*.

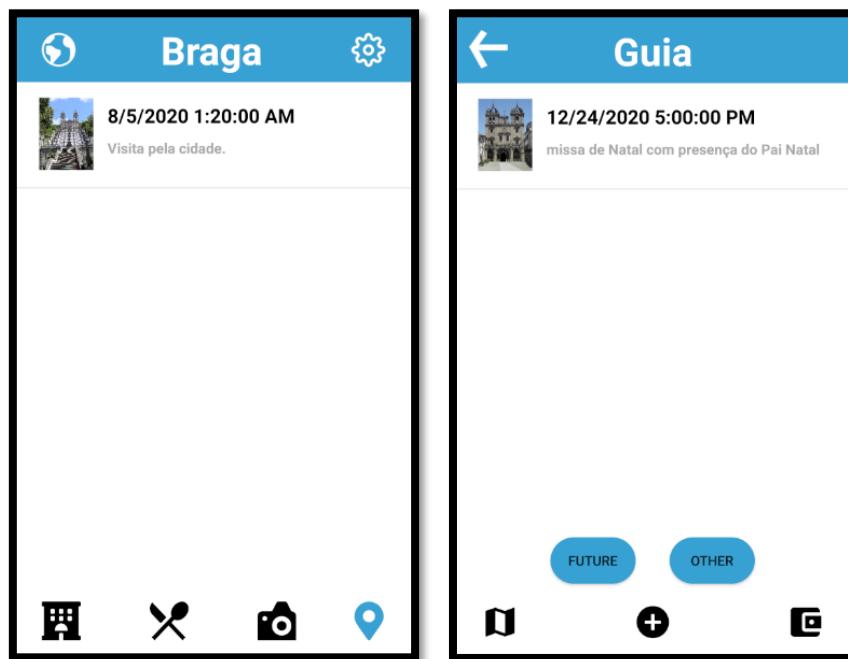


Figura 25 - Menu inicial com Excursões selecionadas (à esquerda) e Menu de Excursões para os guias (à direita)

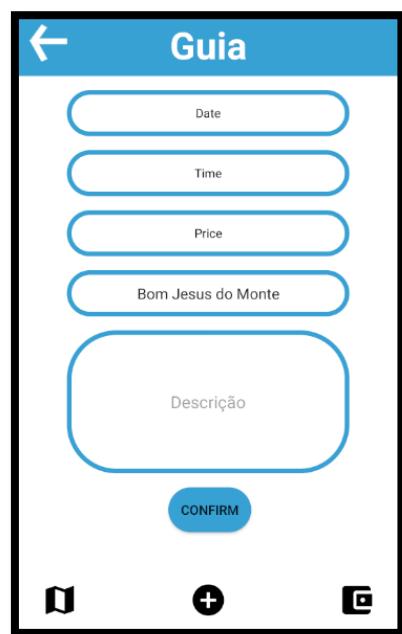


Figura 26 - Interface adicionar nova excursão

Selecionando o ícone do canto inferior direito, pode-se aceder à interface da *Wallet* e verificar o saldo disponível e, eventualmente, fazer um pedido de *cashout*. As excursões do guia continuam visíveis, por baixo.

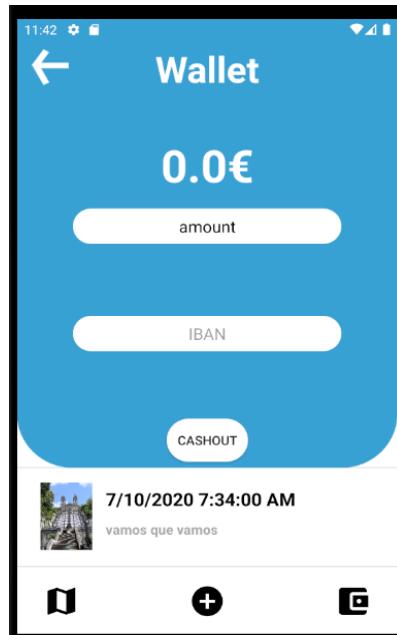


Figura 27 - Interface wallet guia

Para que o guia possa fazer checkout deverá introduzir o valor que quer retirar, não sendo este superior ou valor da sua carteira e fornecer, também, o seu IBAN. Estes valores serão armazenados numa tabela na Base de Dados para posterior processamento.

Como já foi referido, selecionando o ícone do globo, no canto superior esquerdo, podemos aceder ao menu de seleção da cidade, que mostrará as opções disponíveis, como se pode ver no exemplo abaixo.



Figura 28 - Interface escolher cidade

Por outro lado, selecionando o ícone do canto superior direito, pode-se aceder ao menu de definições, que permite alternar entre modos e idiomas, alterar a palavra passa e o número de telemóvel associado à conta e fazer sugestões.

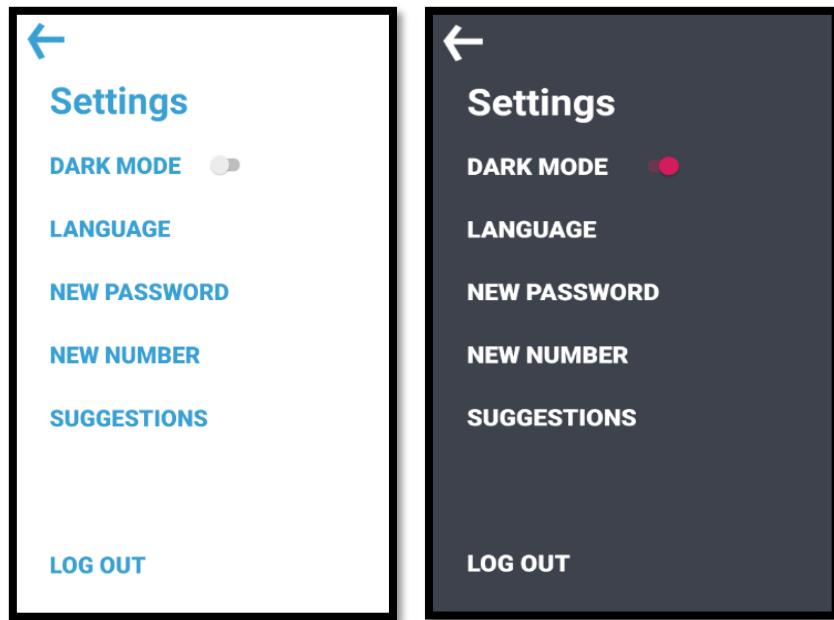


Figura 29 - Interface definições e Interface definições com Dark Mode ativado

Caso se pretenda trocar de idioma, são disponibilizadas as seguintes opções.

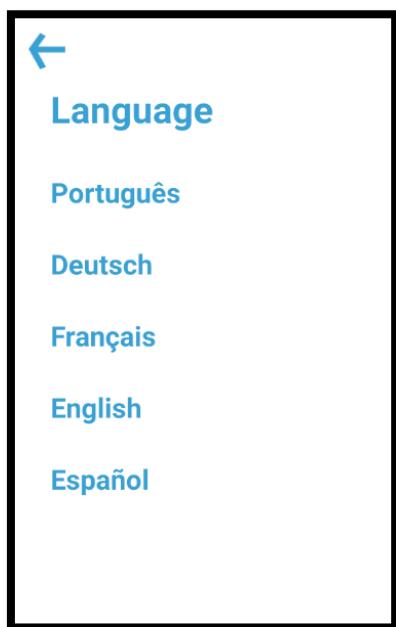


Figura 30 - Interface de alteração de idioma

Se, por outro lado, se pretender editar a conta (alterar *password* ou o número de telemóvel), apresentar-se-ão as seguintes interfaces, respetivamente.

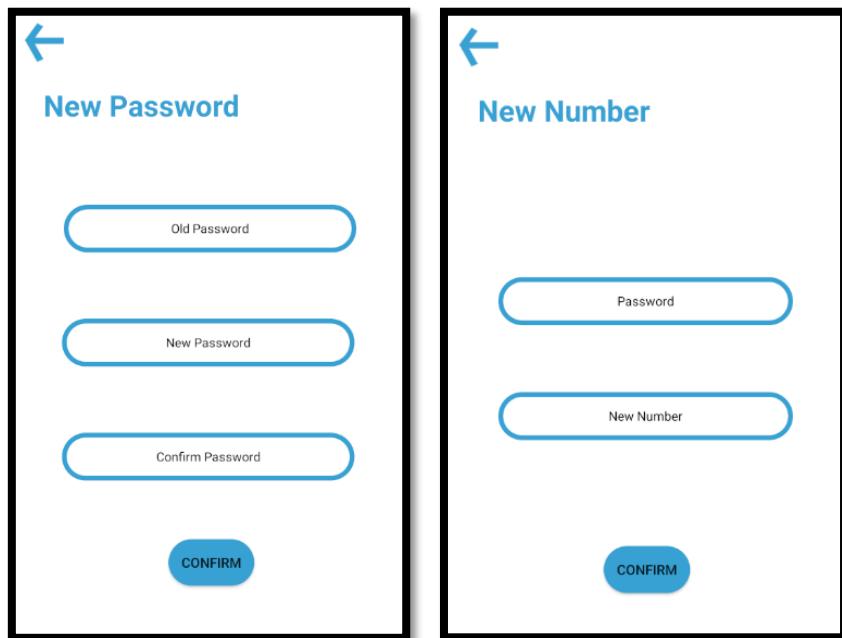


Figura 31 - Interface para trocar de número e Interface troca de número.

Pode-se aceder à interface informativa relativa a um hotel, restaurante ou ponto de interesse, onde são agrupadas todas as suas informações. É também aqui que é possível selecionar o botão *Comentários*.



Figura 32 - Interface informativa de Entidade

Selecionando esse botão, temos, então, acesso aos comentários e classificações relativos ao item em que nos encontramos.

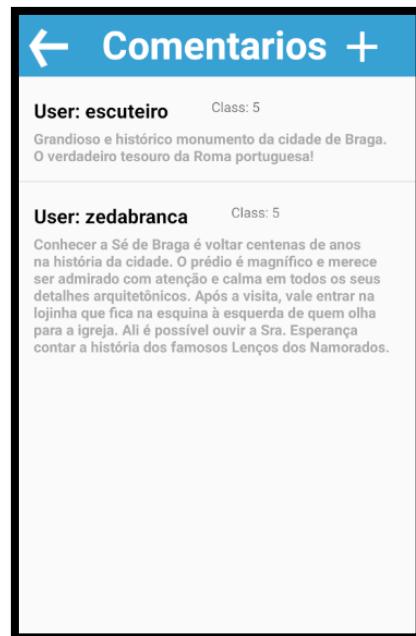


Figura 33 - Interface de comentários e classificações.

É nesta interface, através do botão +, que podemos adicionar um novo comentário e classificação.



Figura 34 -Interface para adicionar comentário/classificação

Quanto às excursões, após a seleção de uma à escolha, é exibida uma interface informativa, semelhante à do exemplo abaixo. É aqui que os utilizadores se podem inscrever. Caso o cliente proceda à sua inscrição numa excursão, a interface *pop-up* à direita aparece, de forma a recolher os dados de pagamento.

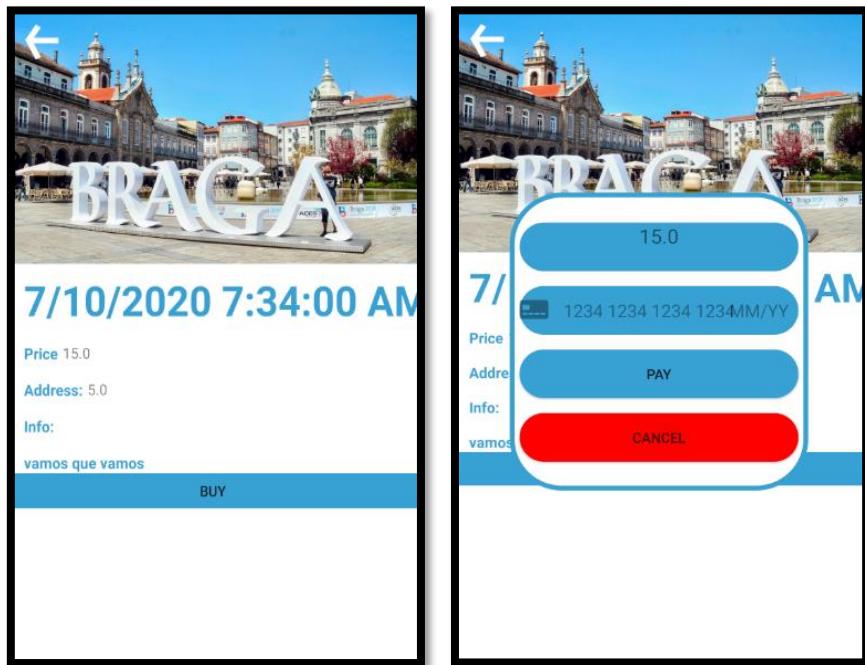


Figura 35 - Interface informativa de excursões e interface pagamento

Diagramas de Sequência

Primeiramente, encontram-se 2 diagramas de sequência que visam clarificar o funcionamento do algoritmo de controlo de versões de atualização das tabelas das Bases de Dados, sendo que o caso do exemplo abaixo é a atualização da tabela dos hotéis.

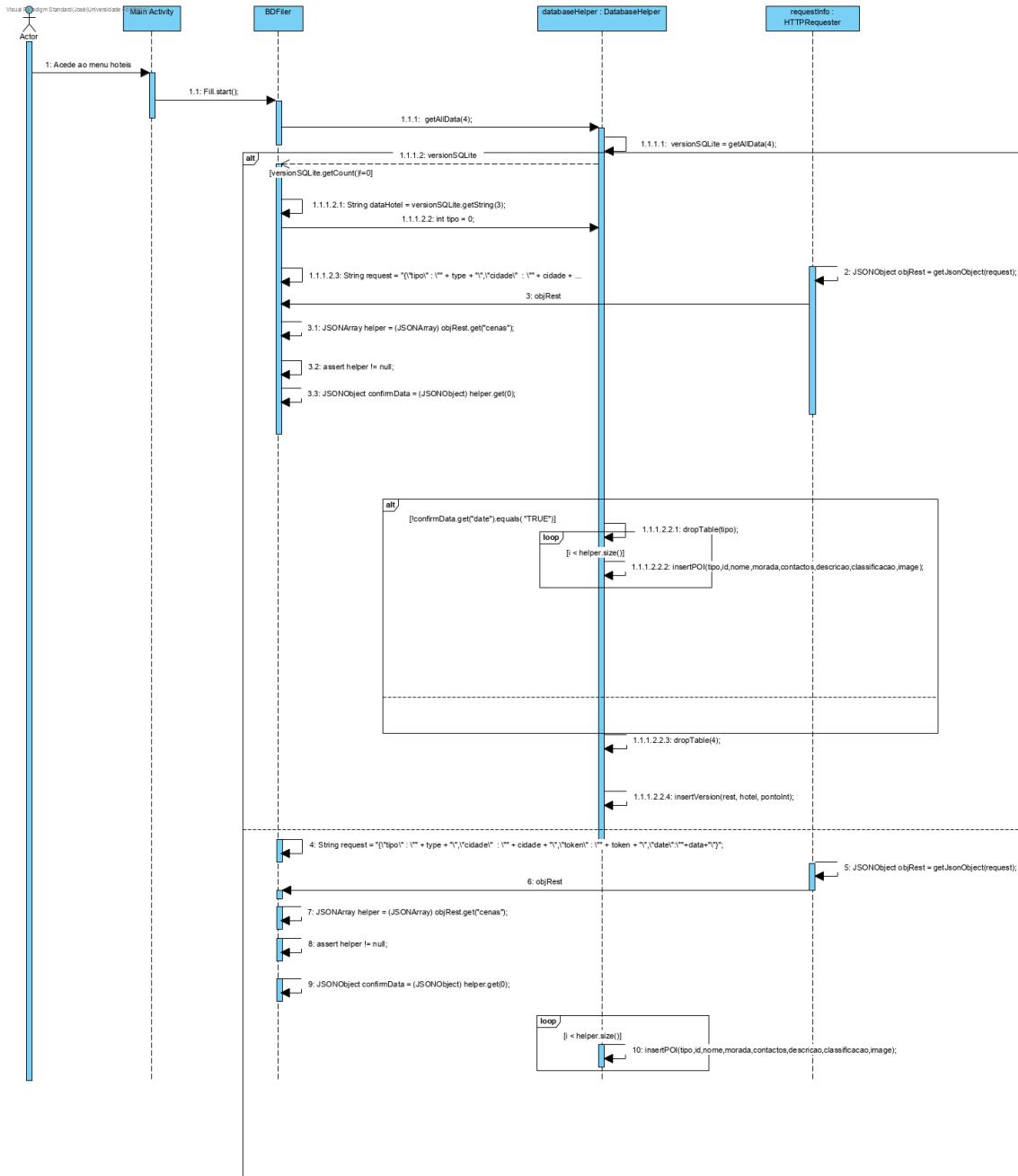


Figura 36 - Diagrama de sequência para o algoritmo no FrontEnd que atualiza as tabelas SQLite

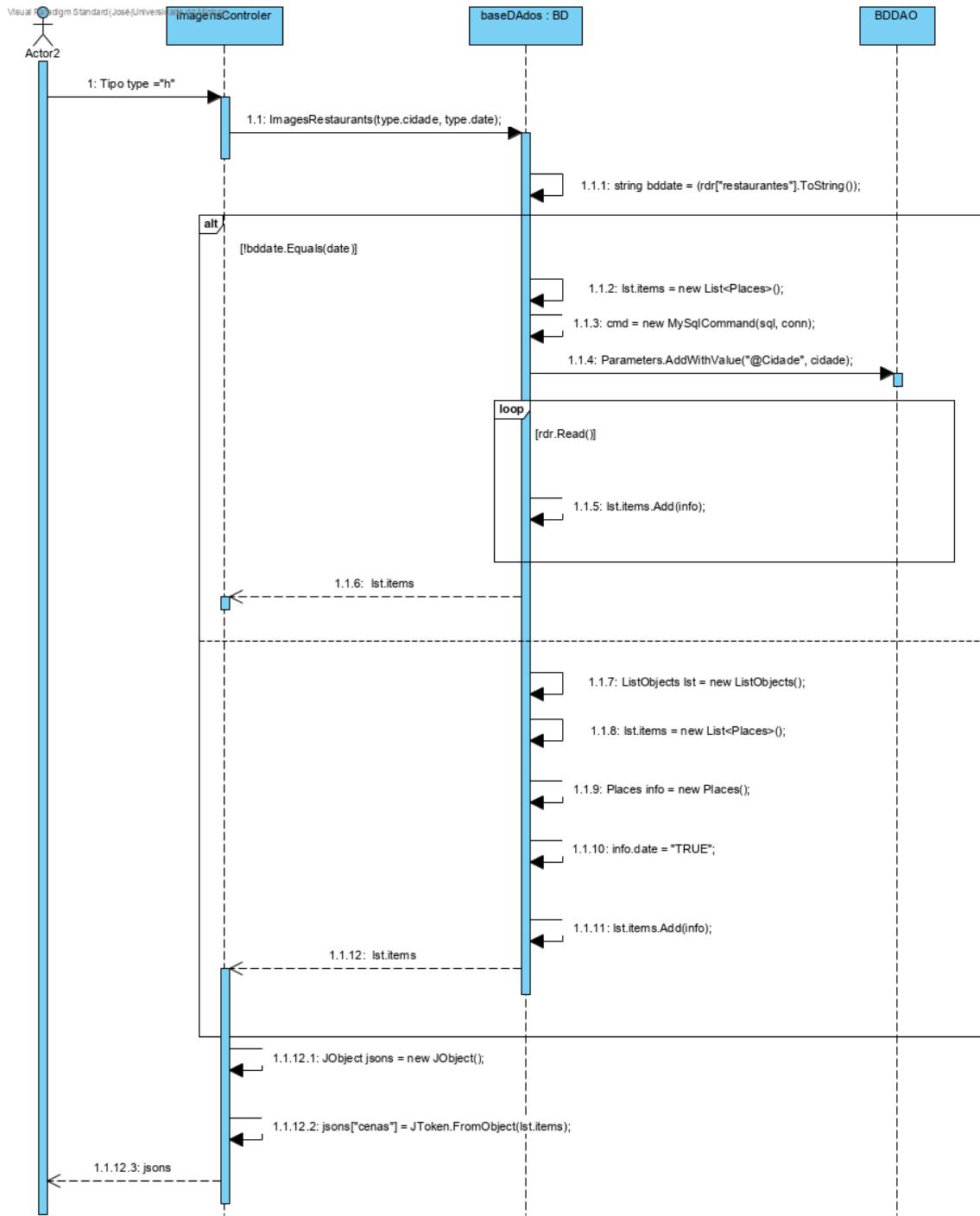


Figura 37 - Diagrama de sequência para o algoritmo no BackEnd que atualiza as tabelas SQLite

Esta foi uma das principais problemáticas do projeto, pois trata-se do carregamento da informação da Base de Dados até ao *FrontEnd*. Para tal, foi construído um algoritmo que pretende minimizar carregamentos desnecessários da base de dados. Quando um utilizador abre pela primeira vez a aplicação, a sua Base de Dados local encontra-se vazia. Por esta razão, é necessário carregar toda a informação dos hotéis, restaurantes, pontos de interesse e excursões.

No entanto, se não for a primeira vez que o utilizador se liga à aplicação, a sua Base de Dados local já se encontrará preenchida. A problemática passa por saber se a informação que o utilizador tem está, ou não, atualizada. De forma a resolver este problema, as Bases de Dados locais têm todas uma tabela apenas com *timestamps*, uma referente a cada tabela. Assim, quando um utilizador abre a aplicação, já com a sua base de dados preenchida, são enviadas as suas *timestamps* e o servidor verifica se é preciso actualizar, ou não, as suas tabelas. Caso seja necessário, é feito *drop* à tabela do utilizador e é guardada a tabela atualizada.

É importante referir que este método de controlo de versões de tabelas não é usado para as excursões. Estas são apenas carregadas para a memória e são sempre actualizadas quando a sua interface é acedida. Deste modo, evitam-se potenciais casos de tentativa de compra de excursões desatualizadas.

Outro desafio encontrado foi a autenticação dos utilizadores, bem como a criação de novas contas. De seguida, encontram-se diagramas de sequência de sistema com o *modus operandi* do sistema nestas situações.

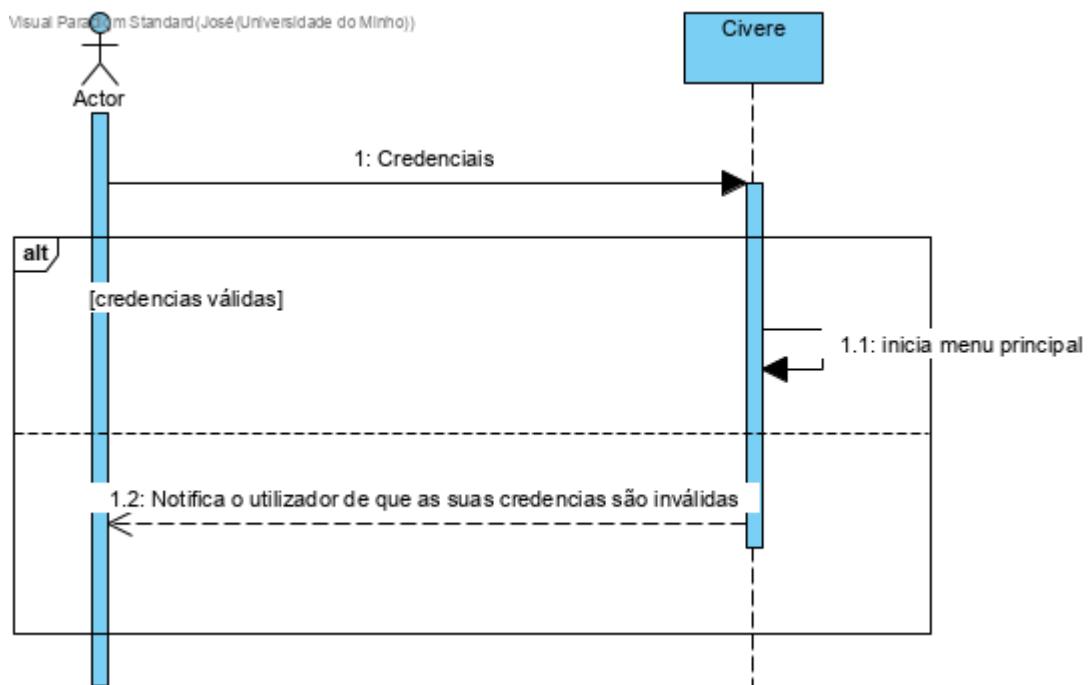


Figura 38 - Diagrama de sequência de sistema Login

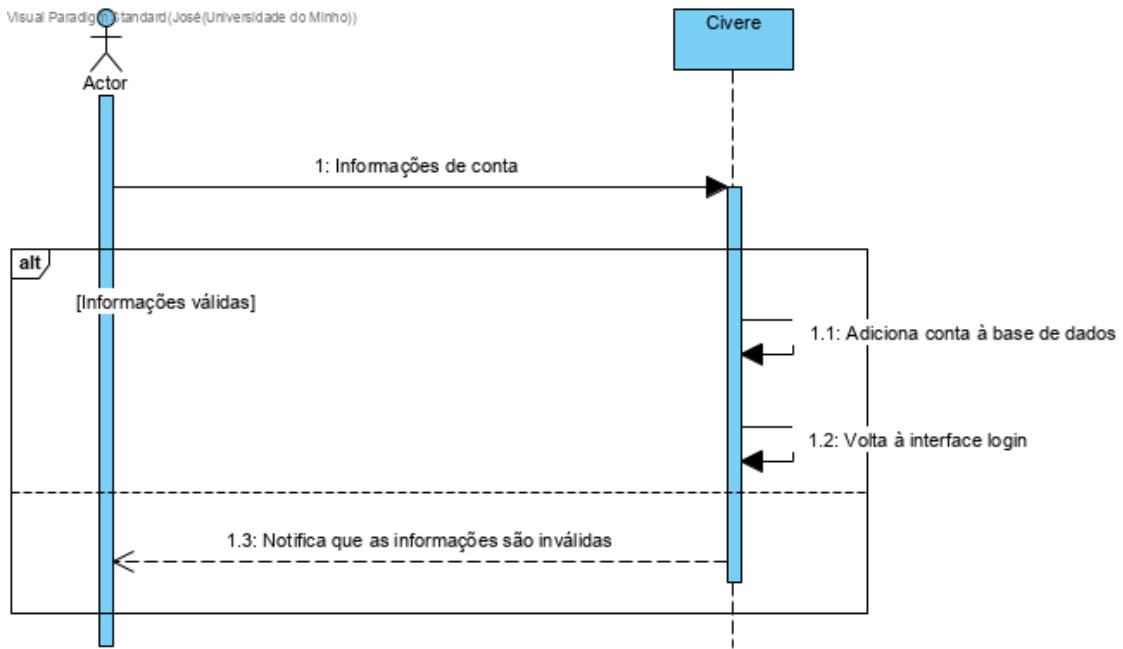


Figura 39 - Diagrama de sequência de sistema Registrar

Quanto às avaliações e comentários, foi necessário ter em conta as situações em que um utilizador tentasse fazer mais que uma avaliação. Nestas situações, a nova avaliação é rejeitada, como se mostra abaixo, no diagrama de sequência de sistema para a inserção de um comentário.

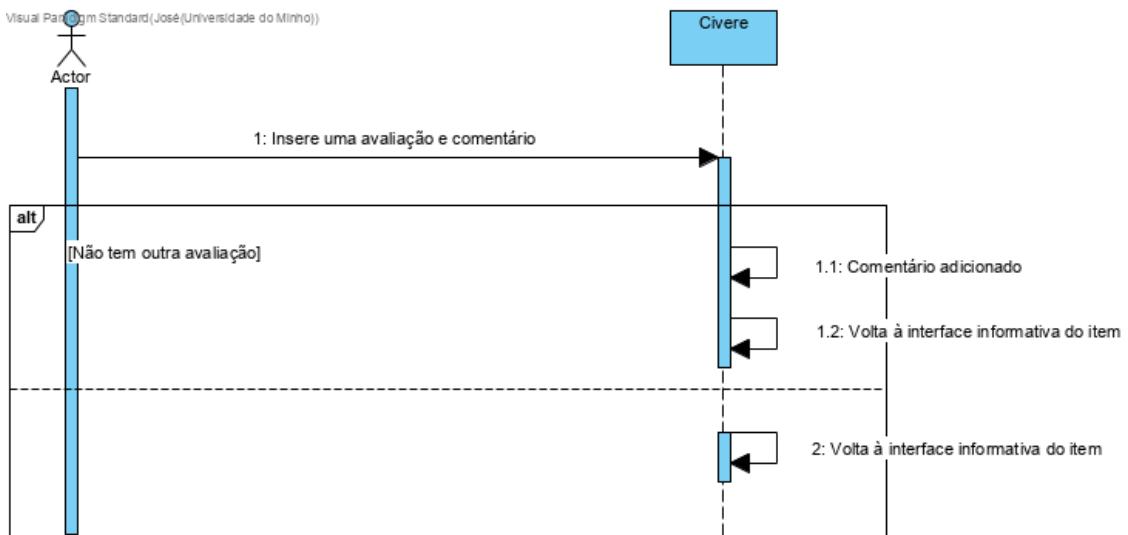


Figura 40 - Diagrama de sequência de sistema Inserir comentário

No momento de inscrição é necessário considerar se o utilizador já se encontra, ou não, inscrito na excursão e se o pagamento foi bem sucedido. Para isso, o algoritmo para a inscrição em excursões encontra-se desenvolvido segundo o diagrama abaixo.

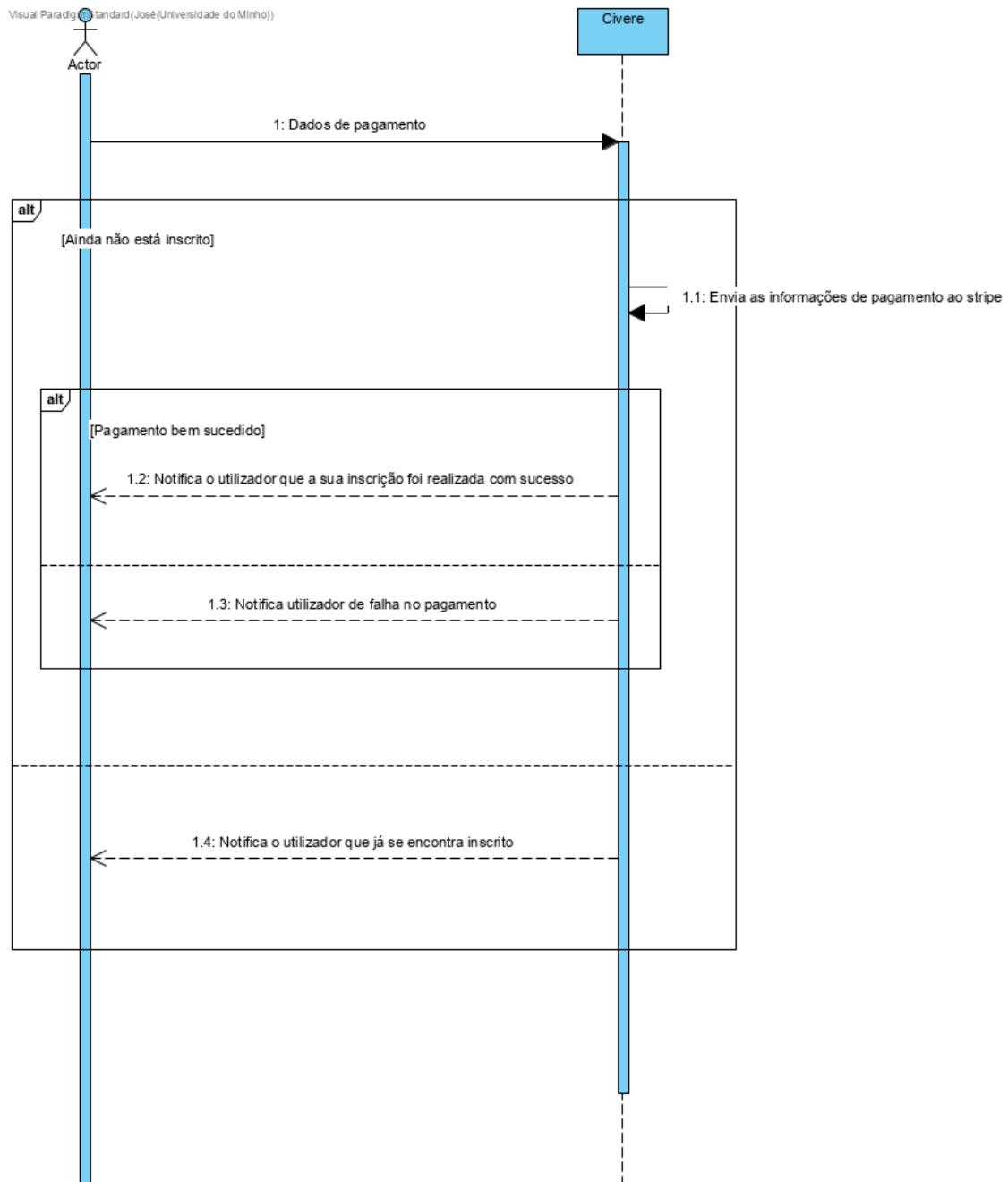


Figura 41 - Diagrama de sequência de sistema Inscrição em Excursão

No caso dos pagamentos aos guias, não pudemos seguir a mesma estratégia, pelo que optamos por receber e guardar o *IBAN* de cada guia com o valor que pretende transferir e, posteriormente, os administradores da aplicação ficam encarregues de realizar essa transferência e notificar o guia, por *email*, do sucesso ou insucesso da mesma.

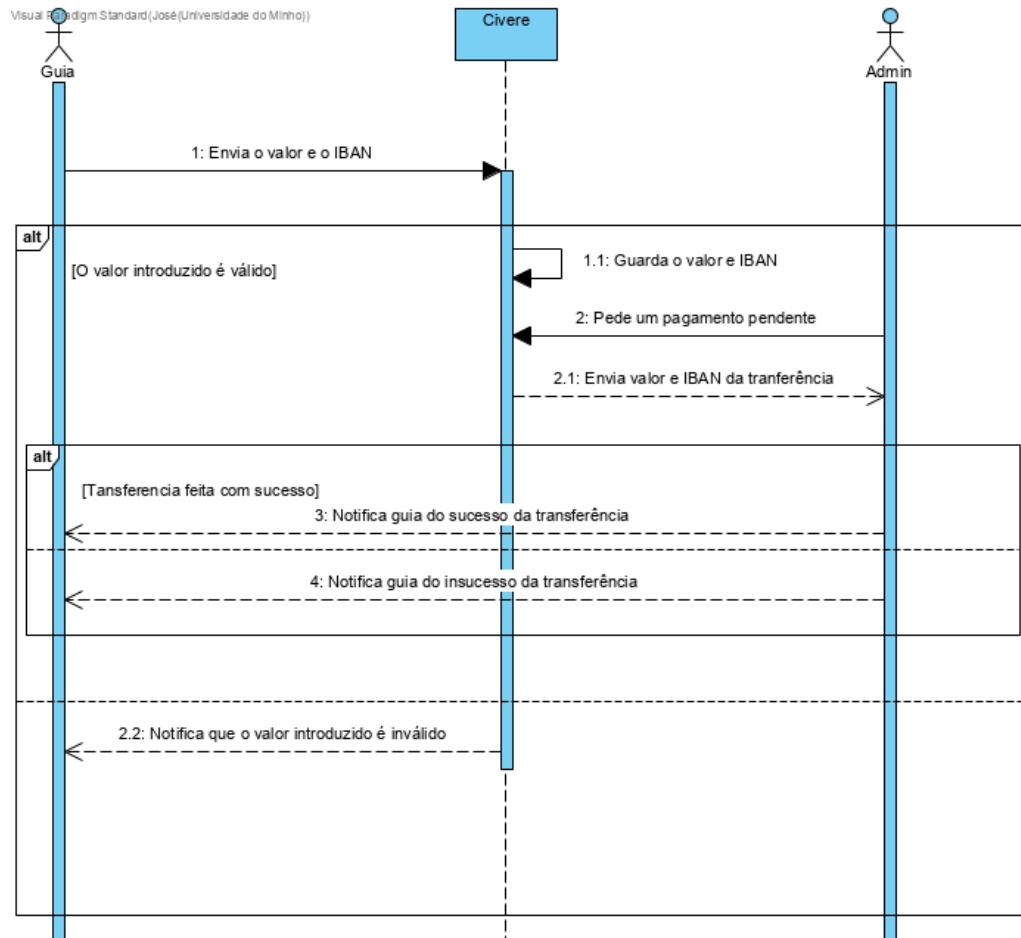


Figura 42 - Diagrama de sequência de sistema Cashout

CONCLUSÃO

Finalmente, a equipa atenta que o balanço geral deste projeto é muito positivo, pois o produto final é uma aplicação funcional, interativa e com muito potencial. Temos orgulho de apresentar a CIVERE, desde o pensamento inicial até ao resultado que se mostra neste relatório.

Consideramos ter atingido os objetivos a que nos propusemos, uma vez que foi feito um delineamento de uma estratégia holística e flexível de desenvolvimento, aplicado um desenvolvimento iterativo e incremental ao longo do projeto e um desenvolvimento ágil, progressivo e melhorado do produto, com coesão e melhoria progressiva da dinâmica da equipa e da sua comunicação interna diária, bem como a adoção de uma metodologia própria de gestão do projeto para discussão quinzenal. No entanto, ao longo do projeto, fomos encontrando vários desafios que, na sua maioria, foram ultrapassados com sucesso.

Após a definição e contextualização do problema e de toda a modelação que consideramos necessária ter sido desenvolvida, o principal desafio com que deparamos foi a escolha e adaptação das melhores tecnologias a usar de forma a alcançar os objetivos a que nos propusemos no início do projeto. Fomos obrigados, por vezes, a recuar e repensar a nossa estratégia de implementação, bem como optar por diferentes ferramentas tecnológicas, alterando ou adaptando o plano inicial, mas nunca desviando profundamente do plano traçado. Contudo, estes recuos e pequenos desvios permitiram tornar a aplicação mais coerente e consistente.

Em conclusão, pensamos que, acima de tudo, enriquecemos os nossos conhecimentos em várias vertentes, para além de aplicarmos uma imensidão de conhecimentos adquiridos previamente, desde a modelação do projeto, passando pela implementação, até à construção do produto final.

TRABALHO FUTURO

Apesar de tratado como finalizado, nunca dessa forma a ele nos podemos referir, pois a dinâmica do desenvolvimento de aplicações, em especial, não o permite. Uma aplicação nunca estará terminada, pois existem sempre melhorias a fazer, novas funcionalidades a implementar e outras a retirar, para que esta não se torne nunca obsoleta nem ultrapassa, deverá sempre evoluir juntamente com a necessidade dos seus utilizadores. Não obstante, sabemos que, mesmo tendo orgulho da versão da aplicação que aqui apresentamos, a equipa considera que alguns pontos menos fortes poderiam ter sido aprimorados, como, por exemplo, o método de *Cashout*, automatizando-o, a criação de um sistema de *rating* e comentário dos guias e até a disponibilização de um perfil do guia, com as suas informações, classificação individual e de cada uma das suas excursões.

Pensando, então, em termos de evolução, é importante referir que, ao longo do projeto, a equipa teve várias ideias sobre como melhorar e expandir a aplicação, mas que, por vários motivos, não puderam ser implementados, desde dificuldades técnicas até prioridade de implementação de funcionalidades. Pensamos, por exemplo, em construir um algoritmo inteligente que exibisse ao utilizador, primeiro, os itens que coincidem com os seus gostos ou até em introduzir a *API* da *Google Maps*, de forma a que fosse possível aceder ao mapa de uma cidade, o que permitiria o cálculo automática de trajetos até um local pretendido.

Por outro lado, pensamos também que, no futuro, poderíamos tornar a aplicação ainda mais prática com a introdução de novos métodos de criação de conta, através das contas da *Google* ou do *Facebook*, novos métodos de pagamento, como *MBWay*, a adição de outros serviços importantes nas cidades, como hospitais, farmácias, cinemas, etc. e até uma pequena notificação no ecrã, quando o utilizador não tivesse acesso à *internet* e tentasse aceder a uma funcionalidade que dela necessite.

Quanto aos guias, de um ponto de vista legal, seria, possivelmente, necessária a entrega de documentos, como por exemplo, o cartão de cidadão, mas, nesta fase, não consideramos esse caso.

Da mesma forma que o mundo se encontra sempre em desenvolvimento e evolução, a nossa aplicação nunca estará acabada, pois esse será o seu fim. Aquele que deixar de evoluir nunca será verdadeiramente grande.

“It is not the strongest of the species that survives, nor the most intelligent that survives. It is the one that is most adaptable to change.” – Charles Darwin

REFERÊNCIAS

<https://geekflare.com/sql-vs-nosql/> - SQL

<https://developer.android.com/> - Android Studio

<https://www.geeksforgeeks.org/android-running-your-first-android-app/> - Android Studio

<https://www.geeksforgeeks.org/json-web-token-jwt/> - JWT

<https://jwt.io/> -JWT

<https://stripe.com/pt-br-pt> - Stripe

<https://www.geeksforgeeks.org/rest-api-introduction/> - REST

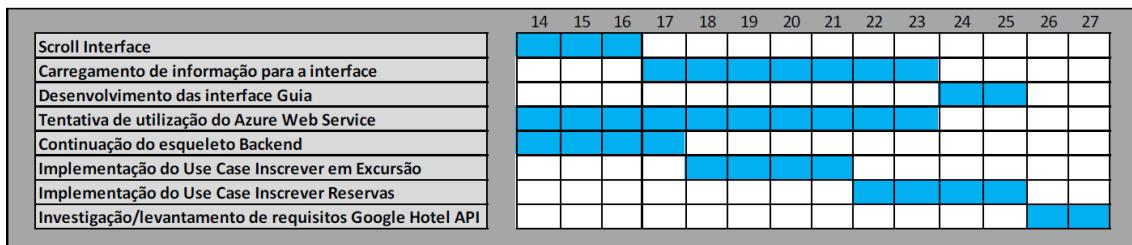
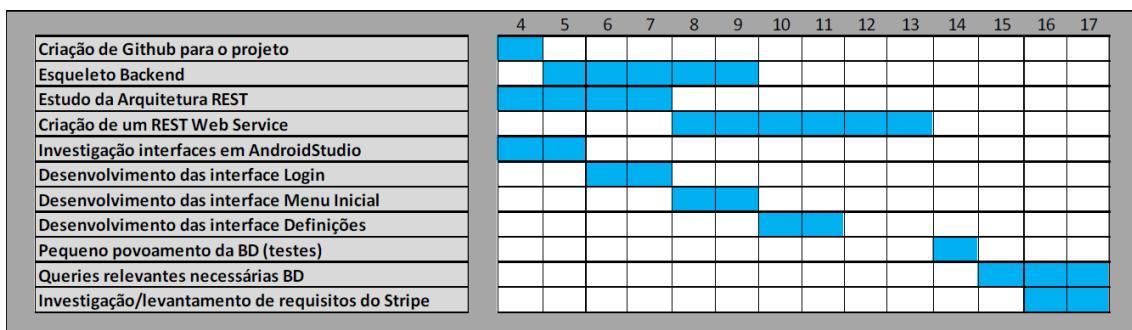
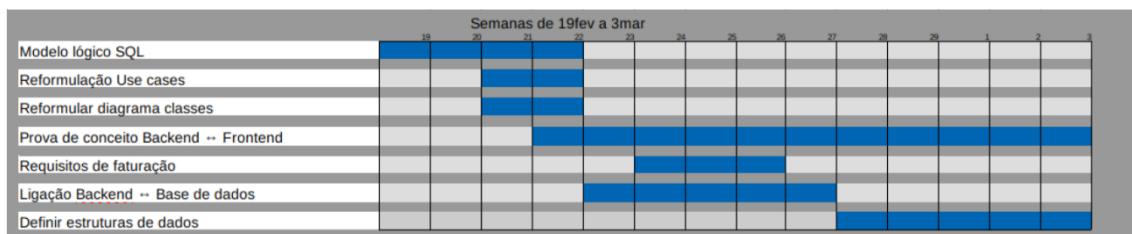
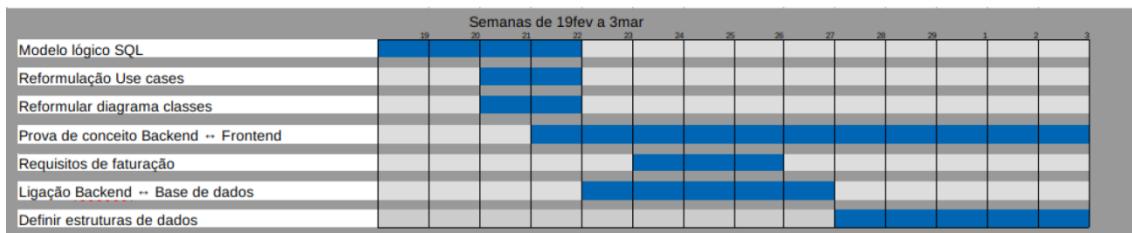
<https://www.sqlite.org/index.html> - SQLite

<https://www.json.org/json-en.html> - JSON

<https://azure.microsoft.com/pt-pt/> - Azure

ANEXOS

ANEXO A - Diagramas de Gantt da organização quinzenal de tarefas



	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Guardar imagens na base de dados														
Carregar imagens da base de dados até ao front-end														
Alterações à base dados														
Pesquisa SQL lite														
Continuar o desenvolvimento da interface guial														
Interface stripe														
Pesquisa oauth2 e bearer														
Pesquisa JSONWEBTOKEN														

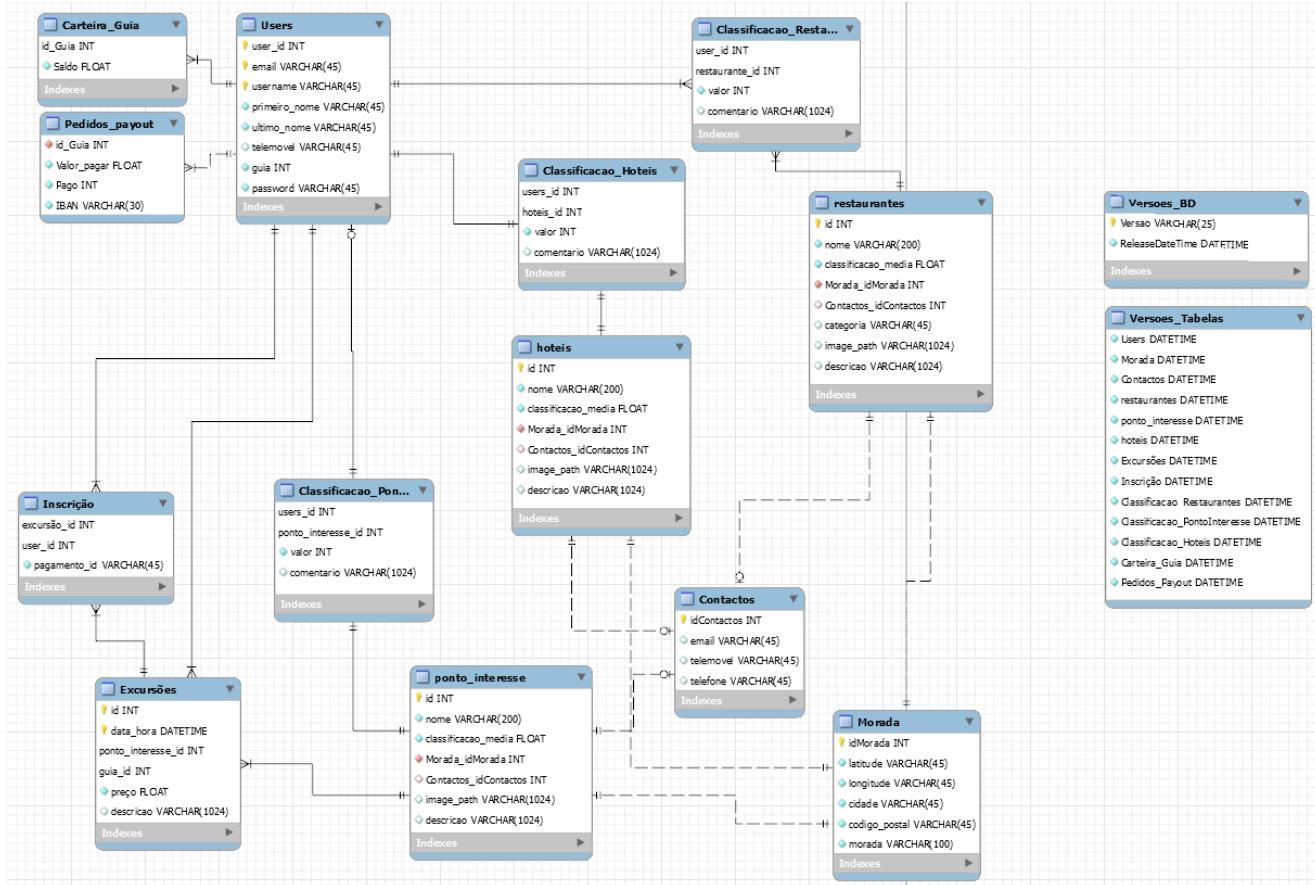
	22	23	24	25	26	27	28	29	30	1	2	3	4	5
Implementação da inicialização da base de dados SQLite														
Implementação do método de inserção de dados na base de dados SQLite														
Introdução à implementação de JSON Object														
Tentativa da implementação do Stripe Front-End														
Tentativa da implementação do Stripe Back-End														
Introdução à implementação do JSONWebToken (OAuth2.0) Front-End														
Introdução à implementação do JSONWebToken (OAuth2.0) Back-End														

	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Correção de dependências Stripe														
Carregamento de dados no SQLite														
Display informação guardada em SQLite nas interfaces														
Checkout Stripe														
Pagamentos Stripe														
Alteração de estruturas para utilizar JSONObject														
Carregamento de URLs (front-end)														
Criar excursões (back-end)														
Pedido de criação de excursões (front-end)														
Pedido de remoção de excursões (front-end)														

	20	21	22	23	24	25	26	27	28	29	30	31	1	2
Alteração Base de Dados para Registar Excursão														
Registrar Excursão - BackEnd														
Registrar Excursão - FrontEnd														
Checkout Stripe correção dependências														
Implementação de dapper - BackEnd														
Finalização de inserção de dados SQLite (<i>timestamp</i>)														
Fix's nos URLs - FrontEnd														
Preenchimento da tabela das excursões														
Disponibilização do serviço de excursões														

	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Introduzir token nos pedidos ao backend														
Pedido de refresh do token														
Alterações na BD para controlo de versões														
Algoritmo de controlo de versões backend														
Finalização algoritmo de versões frontend														
Retirar excursão backend														

ANEXO B - Modelo lógico final da BD



ANEXO C - Especificações dos *Use Cases*

Registrar Utilizador:

Descrição: Utilizador cria uma conta na aplicação.

Cenários: Pablo quer criar uma conta para si na aplicação.

Pré-Condição: - .

Pós-Condição: A nova conta é criada com sucesso.

Fluxo Normal:

1: O utilizador seleciona opção de se registrar.

2: A aplicação pede que o utilizador introduza o seu primeiro e último nome, um *username*, uma palavra-passe e a sua data de nascimento.

3: O administrador introduz as informações pedidas.

4: O sistema regista a nova conta do utilizador.

Fluxo Exceção 1: [username já existir] (4)

4.1: O sistema informa que o *username* escolhido já existe.

4.2 O sistema volta ao menu inicial.

Editar conta:

Descrição: utilizador edita a sua conta na aplicação.

Cenário: o Zeferino quer editar os dados da sua conta na ‘Civere’.

Pré-condição: ter conta criada na aplicação e ter sido autenticado.

Pós-condição: edição concluída como desejado pelo utilizador.

Fluxo Normal:

1: O utilizador seleciona a opção ‘editar conta’.

2: O utilizador insere palavra-passe.

3: A aplicação valida a palavra-passe.

4: O utilizador entra no menu ‘editar conta’.

Fluxo Exceção 1: [palavra-passe errada]

3.1: A aplicação não valida a palavra-passe.

Autenticar Utilizador:

Descrição: utilizador autentica-se na aplicação.

Cenário: o Gervásio inicia sessão na aplicação.

Pré-condição: ter conta criada na aplicação.

Pós-condição: ficar autenticado.

Fluxo Normal:

1: O utilizador seleciona a caixa de credenciais.

2: O utilizador insere e-mail e palavra-passe.

3: A aplicação valida os dados.

Fluxo Exceção 1: [opção ‘recuperar palavra-passe’] (1)

1.1: O utilizador escolhe opção ‘recuperar palavra-passe’.

1.2: A aplicação apresenta o menu ‘recuperar palavra-passe’.

1.3: O utilizador insere o e-mail.

1.4: O Sistema envia um e-mail para recuperar palavra-passe ao utilizador.

1.5: 1.

Fluxo Exceção 2: [dados errados] (3)

3.1: A aplicação não valida os dados inseridos.

3.2: A aplicação avisa que houve erro na validação da conta.

Classificar:

Descrição: o utilizador classifica um elemento do sistema.

Cenário: o Aboubakar quer classificar a sua interação um elemento do sistema.

Pré-condição: estar autenticado.

Pós-condição: classificação feita pelo utilizador.

Fluxo Normal:

1: O utilizador seleciona a opção ‘classificar’.

2: O utilizador escolhe tipo do elemento a classificar.

3: O utilizador escolhe o elemento a classificar.

4: O utilizador classifica o elemento.

5: O sistema regista a classificação.

Fluxo Exceção 1:[utilizador seleciona opção ‘cancelar’](1)

1.1: O utilizador seleciona a opção ‘cancelar’.

1.2: O sistema volta ao menu inicial.

Comentar:

Descrição: o utilizador comenta um elemento do sistema.

Cenário: o Abel Xavier quer comentar a sua experiência com um elemento do sistema.

Pré-condição: estar autenticado.

Pós-condição: comentário feita pelo utilizador.

Fluxo Normal:

1: O utilizador seleciona a opção ‘comentar’.

2: O utilizador escolhe tipo do elemento a comentar.

3: O utilizador escolhe o elemento a comentar.

4: O utilizador comenta o elemento.

5: O sistema regista o comentário.

Fluxo Exceção 1:[utilizador seleciona opção ‘cancelar’](_)

_1: O utilizador seleciona a opção ‘cancelar’.

_2: O sistema volta ao menu inicial.

Inscrever em Excursão:

Descrição: o utilizador quer inscrever-se numa excursão.

Cenário: o Mourinho quer inscrever-se numa excursão pelo centro de Braga.

Pré-condição: ter sido autenticado.

Pós-condição: pedido de subscrição de excursão efetuado.

Fluxo Normal:

1: O utilizador seleciona a opção ‘Excursões’.

2: O utilizador escolhe excursão a inscrever.

3: O utilizador escolhe horário pretendido.

4: O utilizador insere os dados da inscrição.

5: O sistema envia os dados da inscrição para o guia da excursão.

6: A subscrição é concluída.

Fluxo Alternativo 1: [opção contactos] (4)

4.4: O utilizador escolhe obter os contactos do guia e contacta com este diretamente.

4.2: 6.

Fluxo Exceção 1: [dados errados] (5)

5.1: O sistema avisa que os dados são inválidos.

5.2: O sistema volta ao menu inicial.

Fluxo Exceção 2: [horário não disponível]

5.1: O sistema avisa que o horário selecionado não se encontra disponível.

5.2: O sistema volta ao menu inicial.

Reservas:

Descrição: o utilizador quer reservar um elemento do sistema.

Cenário: o Eliseu quer reservar um restaurante em Braga.

Pré-condição: ter sido autenticado.

Pós-condição: pedido de reserva efetuado.

Fluxo Normal:

1: O utilizador seleciona a opção ‘Reservas’.

2: O utilizador escolhe tipo do elemento a reservar.

3: O utilizador escolhe o elemento a reservar.

4: O utilizador insere os dados da reserva.

5: O sistema envia os dados da reserva para o elemento.

6: O pedido de reserva é concluído.

Fluxo Alternativo 1 [utilizador escolhe obter dados do elemento] (4):

4.1: O utilizador escolhe obter os dados do elemento e contacta com este diretamente.

4.2: 6.

Fluxo Exceção 1 [dados inválidos] (5):

5.1: O sistema avisa que os dados são inválidos.

5.2: O sistema volta para o menu inicial.

Propor:

Descrição: O utilizador propõe um ponto de interesse.

Cenários: o utilizador José propõe uma determinada localização como monumento.

Pré-Condição: o utilizador tem a sua conta autenticada

Pós-Condição: A proposta de ponto de interesse é aceite

Fluxo Normal:

1. O utilizador seleciona propor local de interesse.

2. O sistema mostra os locais de interesse propostos por outros utilizadores.

3. O utilizador seleciona a opção “Novo ponto de interesse”.
4. O utilizador adiciona os dados sobre o local.
5. O sistema valida os dados do local adicionado.
6. A proposta de ponto de interesse é aceite.

Fluxo Alternativo 1 [*utilizador seleciona ponto de interesse já adicionado*] (3):

- 3.1 O utilizador seleciona um dos pontos de interesse adicionados previamente.
- 3.2 O sistema incrementa os pedidos de proposta para aquele lugar.
- 3.3 6.

Fluxo Exceção 1 [*os dados introduzidos são inválidos*] (5):

- 5.1. O sistema rejeita os dados do local adicionado.
- 5.2 O sistema informa que a proposta não foi aceite.
- 5.3 O sistema volta ao menu inicial.

Adicionar excursão:

Descrição: O guia adiciona uma excursão.

Cenários: o guia Luís adiciona uma excursão para todos os sábados às 11h.

Pré-Condição: o guia tem a sua conta autenticada.

Pós-Condição: A excursão é adicionada ao sistema.

Fluxo Normal:

1. O guia escolhe a opção adicionar excursão.
2. O guia adiciona os dados da excursão.
3. O sistema pede confirmação dos dados.
4. O guia confirma os dados.
5. A excursão é adicionada.

Fluxo Exceção 1 [*guia rejeita dados*] (4):

- 4.1. O guia rejeita os dados.
- 4.2. O sistema confirma o cancelamento do processo.
- 4.3 O sistema volta ao menu inicial.

Remover excursão:

Descrição: O guia remove uma das suas excursões.

Cenários: O guia remove a sua excursão de segunda às 9h.

Pré-Condição: O guia tem a sua conta autenticada.

Pós-Condição: A excursão passa a estar indisponível e é retirada do sistema.

Fluxo Normal:

1. O guia escolhe a opção remover excursão.
2. O guia seleciona uma das suas excursões.
3. O sistema pede confirmação.
4. O guia confirma que pretende remover a excursão.
5. A excursão é retirada do sistema

Fluxo Exceção 1 [o guia rejeita a remoção] (4):

- 4.1. O guia rejeita o pedido de remoção.
- 4.2. O sistema confirma o cancelamento do processo.
- 4.3 O sistema volta ao menu inicial.

Remover Comentário:

Descrição: O administrador pretende remover um comentário.

Cenários: O administrador remove um comentário ofensivo.

Pré-Condição: O administrador tem de estar autenticado.

Pós-Condição: O comentário é também removido.

Fluxo normal:

1. O administrador seleciona a opção remover num comentário.
2. O sistema pergunta se pretende também remover a avaliação
3. O administrador diz que não.
4. O sistema pede confirmação.
5. O administrador confirma.
6. Comentário removido.

Fluxo alternativo 1 [remover avaliação] (3):

- 3.1.** O administrador diz que sim.
- 3.2.** O sistema pede confirmação.
- 3.3.** O administrador confirma.
- 3.4.** A avaliação é removida.
- 3.5.** 6.

Fluxo Exceção 1 [confirmação comentário rejeitada] (5):

- 5.1.** O administrador rejeita.
- 5.2.** Operação é cancelada.

Fluxo Exceção 1 [confirmação avaliação rejeitada] (3.3):

- 3.3.1** O administrador rejeita.
- 3.3.2** Operação cancelada.

ANEXO D –Protótipos de Interfaces

