Universidade do Minho
Escola de Engenharia
Mestrado em Engenharia Informática

Unidade Curricular de
Tópicos de Desenvolvimento de Software

Ano Letivo de 2023/2024

Desenvolvimento de uma aplicação móvel híbrida
Parte 2

Gonçalo Pereira	PG53834
Nuno Costa	PG52698
Santiago Domingues	PG54225

17 de junho de 2024

Index

1 Introdução **1**

2 Detalhes de implementação **1**

2.1 Estrutura do projeto . . . 1

2.1.1 /components . . . 1

2.1.2 /constants . . . 1

2.1.3 /redux . . . 1

2.1.4 /styles . . . 2

2.1.5 /utils . . . 2

2.1.6 /views . . . 2

2.2 Soluções de implementação . . . 3

2.2.1 Pedidos à API . . . 4

2.2.2 Armazenamento de Dados . . . 4

2.2.3 Autenticação . . . 4

2.2.4 Funcionamento Offline . . . 4

2.2.5 Navegação no *Google Maps* . . . 5

2.3 Bibliotecas/dependências utilizadas . . . 5

2.4 Padrões de software utilizados . . . 6

2.4.1 Dependency Injection . . . 6

2.4.2 Provider . . . 6

3 Mapa de navegação de GUI **7**

4 Discussão de resultados **8**

4.1 Trabalho realizado . . . 8

4.2 Limitações . . . 8

4.3 Funcionalidades extra . . . 8

5 Gestão de projeto **8**

5.1 Gestão e Distribuição de trabalho 8
5.2 Eventuais metodologias de controlo de versão utilizadas 9
5.3 Reflexão sobre Performance individual 9

6 Conclusão **9**

1 Introdução

No âmbito da unidade curricular de Tópicos de Desenvolvimento de Software, foi-nos proposto o desenvolvimento de um guia turístico na forma de uma aplicação móvel híbrida. Nesta segunda parte do projeto, foi desenvolvida uma aplicação cross-platform através da framework React Native.

A aplicação, denominada de "BraGuia", vai além de ser um simples guia turístico, oferece uma diversidade de roteiros turísticos aos seus utilizadores sem precisarem de um verdadeiro guia. Disponibiliza funcionalidades como localização e navegação geográfica, reprodução de mídia referentes aos diversos pontos de interesse disponíveis, entre outros. Para que seja possível fornecer conteúdo relevante e atualizado aos utilizadores, a BraGuia consome informações de um backend desenvolvido especificamente para este projeto.

2 Detalhes de implementação

2.1 Estrutura do projeto

De forma a obter um projeto organizado e bem estruturado, o grupo decidiu seguir a estrutura aconselhada nas aulas da disciplina e as dicas presentes no enunciado do trabalho prático.

Começamos por criar o projeto através do **React Native Community CLI**, que gerou uma estrutura básica de pastas e ficheiros para a aplicação, incluindo o código nativo para Android e iOS.
De seguida, estruturamos a base do projeto com as seguintes diretorias, para facilitar o desenvolvimento da aplicação:

2.1.1 /components

A diretoria **components** é constituída por diversos componentes reutilizáveis. Com o objetivo de acelerar o processo de desenvolvimento, os componentes que eram utilizados mais do que uma vez, eram implementados aqui. Um exemplo notável de reutilização no código da nossa aplicação, são os botões. Estes apresentam todos o mesmo estilo, uma vez que eram apenas passados como argumentos o texto a mostrar no botão e a sua funcionalidade.

2.1.2 /constants

Nesta diretoria foram guardadas as variáveis constantes utilizadas ao longo do desenvolvimento da aplicação, de forma a centralizar e padronizar os valores usados diferentes locais. Esta abordagem reduz a duplicação de código e facilita a atualização dos valores constantes.

2.1.3 /redux

Na diretoria **redux** está centralizado todo o código relacionado com a gestão dos estados da aplicação. Para auxiliar neste processo utilizamos a biblioteca Redux e seguimos as abordagens recomendadas pela

equipa docente.

Dentro desta diretoria, com o intuito manter o código organizado, criamos outras 4 diretorias, sendo estas:

- **/redux/actions:**
  Nesta diretoria **actions** matemos os ficheiros que definem as ações da aplicação, nomeadamente **appActions.js**, **trailsActions.js** e **userActions.js**. Estes contêm os payloads com as informações enviadas pela aplicação para a store e algumas funções necessárias para a atualização dos estados.
- **/redux/reducers:**
  A diretoria **reducers** contém os ficheiros que especificam como o estado global da aplicação muda em resposta às ações recebidas. O estado global da aplicação é dividido em várias partes, cada uma gerida pelo seu reducer, sendo estes o **appReducer**, o **trailsReducer.js** e o **userReducer.js**. Quando uma ação é "despachada", são os reducers que fazem as mudanças necessárias ao estado para, posteriormente, ser retornado um novo estado da aplicação.
- **/redux/selectors:**
  A diretoria **selectors** é responsável por armazenar as funções responsáveis por extrair os dados necessários pela aplicação de forma eficiente e reutilizável. No nosso caso, definimos apenas um ficheiro `selectors.js` com todas as funções necessárias.
- **/redux/store:**
  Na diretoria **store** é onde configuramos a store do Redux. O ficheiro **configureStore.js** contém o estado global da aplicação.

2.1.4 /styles

Nesta diretoria estão guardados os elementos visuais comuns utilizados, na aplicação, contendo especificações para um botão, um ficheiro para pré-definir as cores, e também informações relativas aos temas, permitindo assim generalizar algumas definições, de modo a tornar o restante código mais simples e evitar repetições.

2.1.5 /utils

A diretoria em questão contém ficheiros com funções que são usadas em várias partes da aplicação para facilitar a interação com a API e armazenamento local. Esta diretoria abstrai operações comuns, permitindo que o código principal da aplicação seja mais limpo e fácil de manter. Esta apenas contém dois ficheiros, um que inclui funções para realizar chamadas à API e obter dados, como a lista de *trails*, e outro que fornece funções para armazenar dados no armazenamento local do dispositivo utilizando *AsyncStorage*, que é útil para persistir dados entre sessões.

2.1.6 /views

Esta diretoria armazena os ficheiro (*.jsx*) relativos às vistas do utilizador, ou seja, diferentes páginas da aplicação. Cada ficheiro aqui presente contém a lógica dos botões presentes na página (como navegação ou funcionalidade, por exemplo), assim como o código que trata da estrutura a nível do *design*. Ao longo do desenvolvimento das diferentes *views*, o grupo reutilizou estilos e componentes sempre que possível.

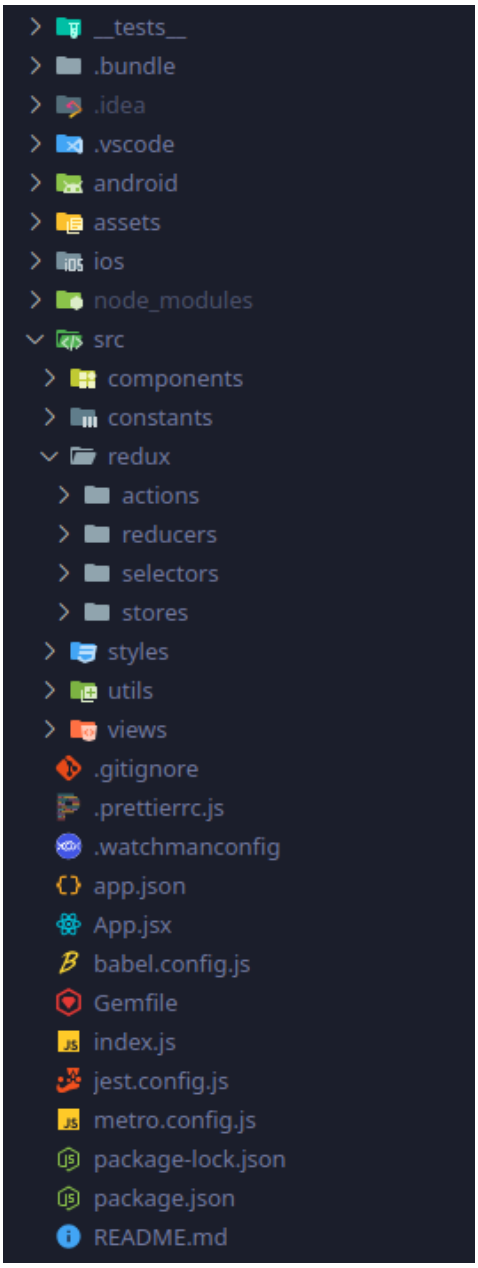


Figura 1: Estrutura do projeto

2.2 Soluções de implementação

Nesta secção serão abordadas as decisões tomadas pelo grupo de forma a conseguir cumprir com os requisitos propostos. Ao longo do desenvolvimento da aplicação foram tomadas inúmeras decisões, sendo que aqui serão detalhadas aquelas que o grupo considera mais relevantes.

2.2.1 Pedidos à API

De modo a ser possível realizar os diversos pedidos à API necessários para o normal funcionamento da aplicação, como *Login*, *Logout* e armazenamento de dados, por exemplo, foi criado um módulo *Api.js* dentro da pasta */utils*. Este módulo visa facilitar a realização de pedidos à API, sendo apenas necessário invocar este módulo e o tipo de pedido que se pretende (*post,get*) com os headers necessários. De notar que todos estes pedidos são executados de forma assíncrona, de modo a não bloquear a *thread* principal e a ser possível ao utilizador continuar com a utilizar a aplicação sem qualquer interferência.

2.2.2 Armazenamento de Dados

Com o intuito de guardar os dados provenientes do *backend* fornecido, o grupo utilizou a biblioteca **Redux**.

Inicialmente, quando a aplicação é iniciada, na *Landing Page* o **Redux** é utilizado para guardar as informações gerais da *app*, que serão, posteriormente, também utilizadas na página dos contactos. De seguida é verificada a existência de *cookies* no armazenamento assíncrono da aplicação, caso estes existam e estejas atualizados, é feito um *fecth* dos dados do utilizador para serem utilizados no perfil do mesmo, no entanto, caso contrário é feito primeiro o processo de autenticação do utilizador. Por fim, quando a **HomePage** carregada é feito um último *fecth*, desta vez dos roteiros para que estes possam ser mostrados ao utilizador.

O grupo optou pela utilização de um sistema **AsyncStorage** para o armazenamento persistente das *cookies* e do histórico de roteiros realizados pelo utilizador no passado.

2.2.3 Autenticação

Na **Landing Page** é feita uma verificação da existência de *cookies* no armazenamento assíncrono e se o estado de **Login** do utilizador se encontra a *true*. Caso o estado de **Login** do utilizador seja *false*, significa que o utilizador não está autenticado e, consequentemente, este será reencaminhado para a página de **Login**. Após o preenchimento dos dados convenientes para a ação de autenticação, será feito um pedido de **Login** à API que, em caso de sucesso, retornará os *cookies* necessários para a realização de um pedido *get (/user)* à API, com o objetivo de obter os dados do utilizador. Após o processo de autenticação os dados do utilizador serão armazenados na *store*, através do auxílio da biblioteca **Redux** e o estado de **Login** do utilizador (*isLoggedIn)* ficará com o valor *true* até que o utilizador realize o processo de **Logout**.

2.2.4 Funcionamento Offline

De forma a possibilitar o carregamento de mídia em ocasiões de pouca ou nenhuma conexão à rede foi disponibilizado um método em que se o utilizador efetuar o *download* das imagens disponíveis, estas são armazenadas no sistema de ficheiros do dispositivo, e posteriormente, se o acesso à aplicação for feito em condições de pouca rede, a imagem será carregada a partir do sistema de ficheiros do dispositivo. De notar, que apesar de diversas tentativas, não foi possível implementar o mesmo método para outros tipos de mídia, como vídeos e áudios.

2.2.5 Navegação no Google Maps

De forma a atingir o objetivo de permitir ao utilizador utilizar a aplicação *Google Maps*, foi criada uma função *handleStartTraill* que é chamada quando o utilizador clica no botão com o texto "iniciar", na página de um *trail*. De forma resumida, esta função começa por fazer o tratamento do tipo de utilizador, ou seja, apenas é executada se o utilizador em questão for do tipo *premium*. Se não for, é lançado um *alert* para notificar que apenas os utilziadores do tipo *premium* podem acessar a esta funcionalidade. De seguida, o *trail* é adicionado ao histórico e as coordenadas dos *pins* são recolhidas para uma lista, removendo-se as coordenadas repetidas. Depois, é utilizada a biblioteca **react-native-get-location**, quer permite obter a localização atual do dispositivo, de forma a adicionar também à rota o trajeto desde a localização atual até ao ponto inicial do *trail*. Por último, é criado o url com as informações necessárias e, através do método **linking**, é aberto o *Google Maps*.

2.3 Bibliotecas/dependências utilizadas

A presente secção serve para enumerar as bibliotecas e dependências utilizadas pela aplicação. Começando pelas bibliotecas, através do ficheiro **package.json** selecionamos as mais importantes a mostrar, sendo estas:

- react-native-async-storage/async-storage: 1.23.1
- react-navigation: (bottom-tabs: 6.5.20; native: 6.1.17; native-stack: 6.9.26)
- reduxjs/toolkit: 2.2.5
- redux: 5.0.1
- react-native-get-location: 4.0.1
- react-native-linear-gradient: 2.8.3
- react-native-permissions: 4.1.5
- react-native-safe-area-context: 4.10.1
- react-native-screens: 3.31.1
- react-native-sound: 0.11.2
- react-native-vector-icons: 10.1.0
- react-native-video: 6.2.0
- rn-fetch-blob: 0.12.0

É importante notar que ao longo do desenvolvimento, a implementação de algumas funcionalidade passaram por várias abordagens diferentes, até se obter o melhor resultado, pelo que é possível existir uma ou outra biblioteca na lista que na prática não esteja a ser utilizada.

Por fim, em relação às dependências, no ficheiro *build.gradle* do Andorid foram usadas as seguintes:

- com.android.tools.build:gradle
- com.facebook.react:react-native-gradle-plugin
- org.jetbrains.kotlin:kotlin-gradle-plugin

2.4 Padrões de software utilizados

2.4.1 Dependency Injection

Um dos padrões de software que utilizamos foi a injeção de dependências. Com o objetivo de aumentar a reutilização de código, o grupo decidiu implementar componentes que recebiam dependências na forma de props. Um exemplo claro da utilização deste padrão foi no componente **Button**. Uma vez que pretendíamos manter o mesmo estilo para todos os botões utilizados na aplicação, a utilização de um componente que recebia o texto a mostrar e a função a aplicar quando este era pressionado, simplificou o processo, evitando a duplicação de código.

```
const Button = ({title, onPress}) => {
  return (
    <TouchableOpacity style={styles.buttonContainer} onPress={onPress}>
      <Text style={styles.buttonText}>{title}</Text>
    </TouchableOpacity>
  );
};
```

2.4.2 Provider

Outro padrão utilizado no desenvolvimento da aplicação foi o padrão Provider, essencial para uma gestão eficiente do estado global. Através do Provider do Redux, conseguimos disponibilizar o estado global, gerido pela store, a todos os componentes da aplicação, evitando a necessidade de propagar os dados através de props entre os diferentes níveis da hierarquia de componentes. Desta forma, conseguimos alcançar uma melhor organização e manutenção do código, uma vez que o acesso aos dados se tornou, de certa forma, centralizado.

```
const RNRedux = () => (
  <Provider store={store}>
    <App />
  </Provider>
);

AppRegistry.registerComponent(appName, () => RNRedux);
```

3 Mapa de navegação de GUI

A figura seguinte mostra o gráfico de navegação da aplicação. Através da sua análise conseguimos perceber as diferentes *views* acessíveis através da página principal/inicial, transições essas conseguidas através da barra de navegação. Esta barra de navegação permite transitar entre a página principal, página dos contactos e página do perfil. A partir da página principal é também possível aceder à página de *upgrade* para *premium* (que se trata de uma tela informativa que indica as vantagens das contas *premium*) e a página dos *Trails*. Já no *Trail*, é possível navegar para o fragmento*Pin*. Existe também na página principal o histórico, que apresenta os *trails* cujo utilizador iniciou, e que são também clicáveis para a sua página.

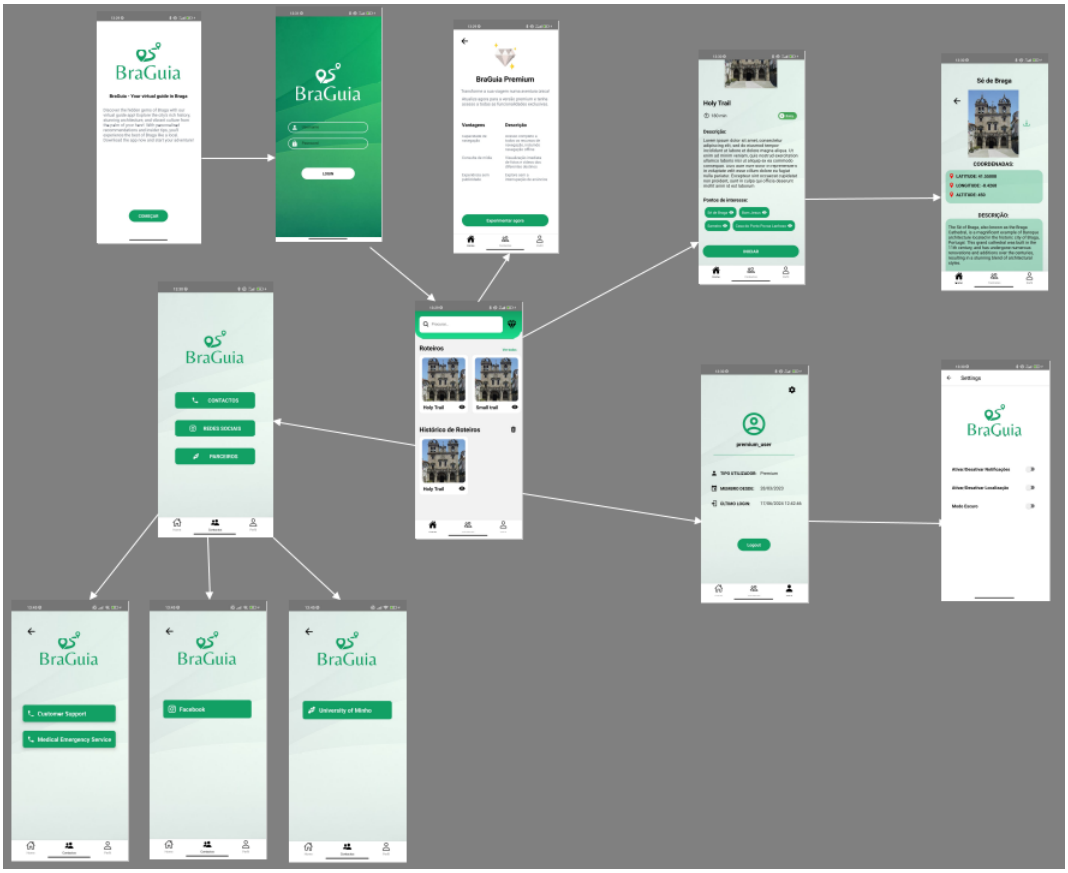


Figura 2: Mapa de navegação de GUI

4 Discussão de resultados

4.1 Trabalho realizado

Analisando cuidadosamente a lista de requisitos proposta pela equipa docente, presente no enunciado, é crucial informar que o grupo não conseguiu cumprir com um dos requisitos mais importantes, o serviço de localização em *background*. Existiram inúmeras tentativas e abordagens para esta implementação, porém o objetivo não foi alcançado. À exceção deste, todos os restantes requisitos foram cumpridos e a aplicação mostra-se a funcionar corretamente, sendo a sua utilização e interface bastante intuitivas. O grupo procurou cumprir também com dos princípios de qualidade definidos pela *Google*, sendo que muitos deles foram cumpridos com sucesso, mas não todos.

Note-se que em anexo encontram-se *printscreens* nas diferentes páginas da aplicação.

4.2 Limitações

Apesar dos esforços para desenvolver uma aplicação robusta e funcional, é importante reconhecer que esta apresenta algumas limitações, sendo esta também uma análise importante para ter conhecimento de algumas melhorias a implementar no futuro.

- Serviço de localização em *background* - A maior limitação da aplicação desenvolvida é a não concretização deste requisito. Tal como explicado no capítulo do trabalho realizado, este requisito mostra-se importantíssimo, porém não foi conseguido.
- Guardar mídia localmente - Este requisito foi parcialmente atingido, visto ser possível guardar e carregar imagens de um sistema local, no entanto o mesmo não acontece com vídeos e áudio.

4.3 Funcionalidades extra

Relativamente a funcionalidades extra, foi implementado na página de definições um *switch* para alterar para o modo escuro. Apesar desta implementação, alguns elementos visuais não permitem a tradução automática para este modo, como imagens, por exemplo, pelo que é um dos objetivos do grupo aplicar melhorias a este modo, no futuro. Apesar disto, porém este modo está funcional (na medida do que é possível ser alterado automaticamente).

5 Gestão de projeto

5.1 Gestão e Distribuição de trabalho

A alocação de tarefas foi dinâmica, ocorrendo à medida que o trabalho progredia. Cada membro avaliava as necessidades e restante trabalho e selecionava uma tarefa para realizar, notificando os restantes elementos do grupo, ou seja, não houve pré-distribuição das tarefas. Existiu apenas, numa fase final do

projeto, uma divisão daquilo que ainda não estava concluído, onde se tentou atribuir a cada elemento as tarefas que estes realizaram na primeira parte do projeto.

5.2 Eventuais metodologias de controlo de versão utilizadas

Durante o desenvolvimento do projeto, a nossa metodologia de controlo de versão foi baseada em várias funcionalidades associadas a um repositório do **GitHub**. Esta estratégia permitiu-nos gerir de forma eficiente as alterações realizadas ao longo da evolução da aplicação.

À medida que cada membro do grupo implementava uma funcionalidade, a mesma era registada no repositório através de um *commit*. Desta forma todas modificações ficaram documentadas, permitindo-nos acompanhar a evolução da aplicação e facilitar a identificação e correção de possíveis erros futuros.

Além disso, sempre que mais do que um elemento do grupo trabalhava na mesma funcionalidade, criávamos uma *branch* separada da principal. Isto permitiu-nos trabalhar em paralelo sem interferir nas alterações principais. Após a conclusão do trabalho em cada *branch*, realizávamos um *merge* para aplicar as alterações na *branch* principal.

5.3 Reflexão sobre Performance individual

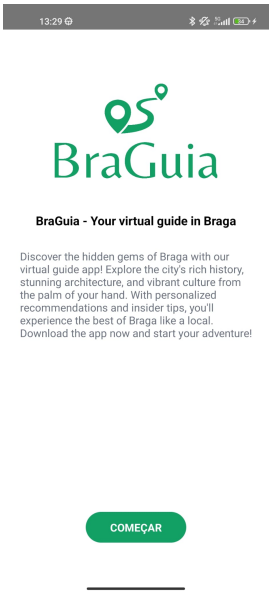
- Gonçalo Pereira (PG53834) - Um dos requisitos, o serviço de localização em *background*, não ficou implementado. Apesar de várias tentativas por parte do grupo, surgiram sempre dificuldades que acabaram por tornar a implementação deste requisito inviável.
- Nuno Costa (PG52698) - Apesar de já ter tido a oportunidade de trabalhar com React Native anteriormente, esta foi uma experiência diferente e mais completa. Embora o resultado final não tenha alcançado todas as expectativas, uma vez que a funcionalidade de carregamento de mídia local não ficou completamente funcional e a implementação do serviço de localização não teve sucesso, acredito que o grupo realizou um bom trabalho e, em geral, obteve um bom produto final.
- Santiago Domingues (PG54225) - Na parte de carregamento de mídia local em caso de pouca conexão à rede ficou a faltar a implementação para vídeos e áudios, apesar de diversas tentativas e pesquisas em documentação e páginas web, nas quais muitas introduziam o facto de não ser sequer possível a leitura de vídeos e áudios a partir do sistema de ficheiros do dispositivo. Apesar disso ainda foram realizadas tentativas para conseguir essa implementação, todas sem sucesso.

6 Conclusão

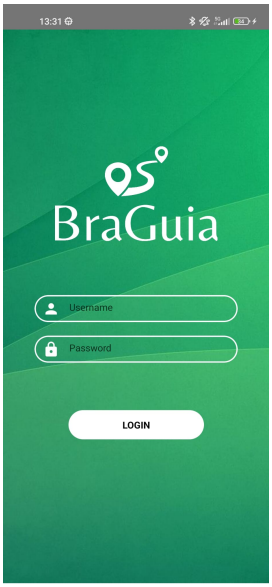
Tal como na primeira parte do projeto, o grupo contactou pela primeira vez com uma ferramenta, desta vez o **React Native**, pelo que é notória a quantidade de conhecimentos adquiridos ao nível do desenvolvimento de aplicações na mesma. Esta segunda parte do projeto permitiu também perceber as etapas do desenvolvimento multi-plataforma, ao invés de ser apenas para Android como foi o caso da primeira, sendo possível ver as diferenças entre as diferentes plataformas (Android e iOS, neste caso), o que nos prepara para eventuais projetos futuros.

Quanto à aplicação, apesar da falha num dos requisitos acima mencionados, o grupo mostra-se satisfeito com o trabalho desenvolvido, criando uma aplicação intutiva, apelativa e útil, que tem espaço para ser melhorada e ser tornada numa aplicação com mais qualidade e funcionalidades.

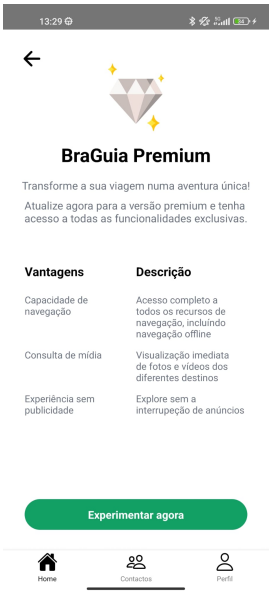
Anexo



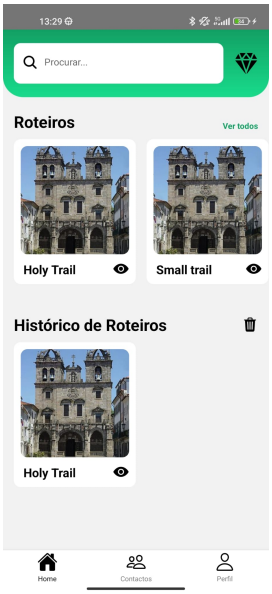
(a) Landing page.



(b) Login

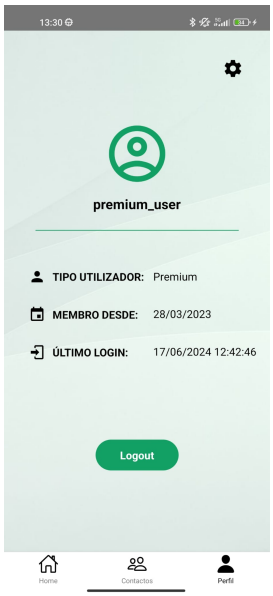


(c) Premium



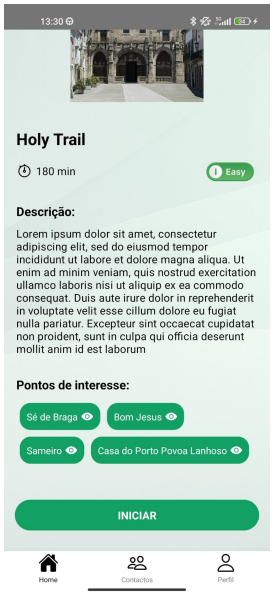
(d) Home Page

(a) Contactos



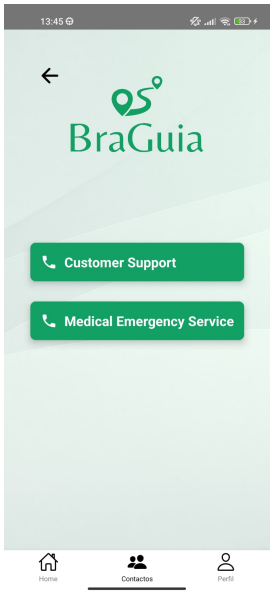
(b) Perfil

(c) Definições

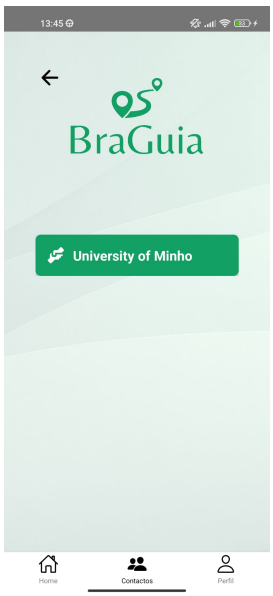


(d) Trail

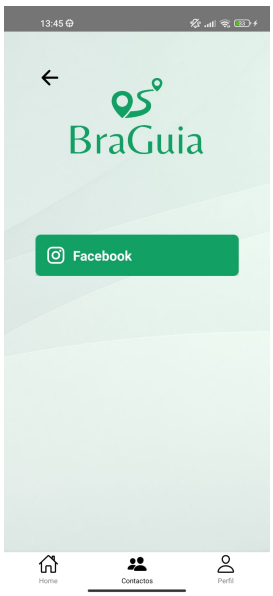
(a) Pin



(b) Suporte



(c) Parceiros



(d) Redes sociais