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

Gonçalo Pereira	PG53834
Nuno Costa	PG52698
Santiago Domingues	PG54225

11 de maio de 2024

Index

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.

A primeira etapa deste projeto focou-se no desenvolvimento da aplicação com uso exclusivo de tecnologia nativa Android, para a qual optamos por utilizar Java. 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.

A aplicação disponibiliza funcionalidades como localização e navegação geográfica, reprodução de mídia referentes aos pontos de interesse, 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.

O presente relatório visa descrever de forma detalhada a estrutura do projeto, assim como as soluções do grupo para a sua implementação. Serão também enumeradas todas a funcionalidades da aplicação, assim como eventuais limitações, terminando com uma visão acerca da gestão do projeto, e a conclusão.

2 Detalhes de implementação

2.1 Estrutura do projeto

De acordo com a recomendação da equipa docente o grupo decidiu utilizar a arquitetura MVVM *Model - View - ViewModel*. Este padrão ajuda a separar de forma clara a lógica de negócios de uma aplicação da sua interface de utilizador (UI), tal separação facilita na resolução de futuros problemas que ocorram durante o desenvolvimento da aplicação, tornando-a mais fácil de testar, manter e evoluir. Na figura 1 podemos observar uma representação da Arquitetura MVVM e na figura 2 apresentamos a estrutura do nosso projeto baseado nesta mesma arquitetura.

De notar que dentro da pasta *model* o grupo optou por subdividir em 2 pastas, uma *DAO* e outra *Objects*, para ser mais fácil de diferenciar os próprios objetos dos seus *Data Access Object* . A pasta *model* contém também a *RoomDatabase*, denominada **BraguiaDatabase**, uma *API_Service* que contém os métodos de chamadas à API e um *Type_converter* para realizar conversões de tipos para a base de dados.

A pasta *repository* contém os elementos dos repositórios que são utilizados para gerir as várias fontes de dados, por exemplo, fazer pedidos à API (através dos métodos da *API_Service* e armazenar os dados na base de dados (*BraguiaDatabase*), sendo também responsável por consultar os dados armazenados na base de dados. De forma a modularizar esta secção optou-se por criar 3 classes similares de repositórios, uma referente à aplicação, outra referente aos utilizadores e outra referente aos trails.

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.

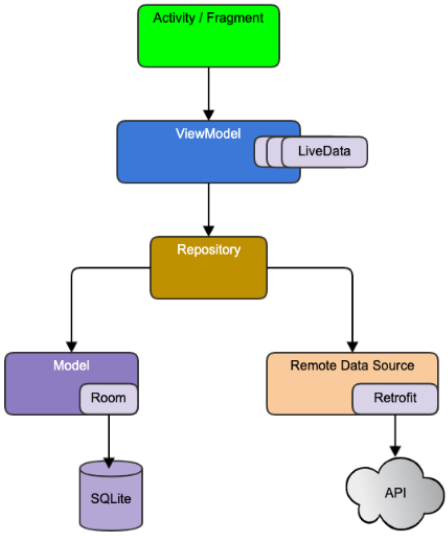


Figura 1: Arquitetura MVVM

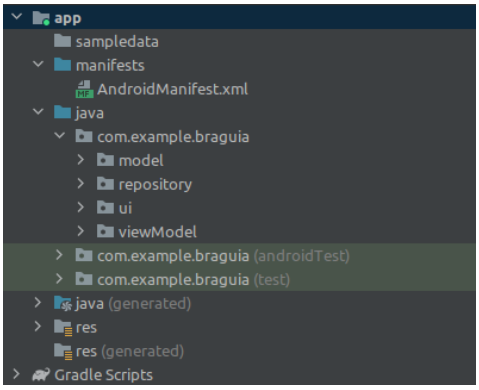


Figura 2: Estrutura do projeto

2.2.1 Navegação entre fragmentos

Para navegar entre os diferentes fragmentos da aplicação foi utilizado um **NavController**. Este componente é responsável por gerir a transição entre os diferentes fragmentos ou atividades da aplicação, gerindo a *stack* de navegação. Para além da sua utilização ser recomendada pela Google, como sendo uma boa prática, este componente oferece inúmeras vantagens, entre elas, a compatibilidade com a arquitetura utilizada (MVVM) e a simplicidade da implementação da navegação, tornando o desenvolvimento do código mais intuitivo. A seguinte porção de código mostra a utilização do NavController para navegar entre os diferentes fragmentos da barra de navegação.

```
NavController navController = Navigation.findNavController(this, R.id.nav_host_fragment);
    binding.bottomNavigationView.setOnItemSelectedListener(item -> {
        int id = item.getItemId();
        if (id == R.id.home) {
            navController.navigate(R.id.HomeFragment);
        } else if (id == R.id.profile) {
            navController.navigate(R.id.ProfileFragment);
        } else if (id == R.id.contacts) {
            navController.navigate(R.id.ContactFragment);
        }
        return true;
    });
```

2.2.2 Recolha e armazenamento de dados

O método utilizado pelo grupo para a recolha e armazenamento de dados passa pelo uso de um **API_Service**, que define os métodos para interagir com a backend da aplicação. Além disso, são utilizados os repositórios **AppRepository**, **TrailRepository** e **UserRepository**, que servem como intermediários entre a API e a base de dados *ROOM*. Esses repositórios não só definem os métodos para inserção dos dados na base de dados local, como gerem a obtenção desses dados.

A classe **BraguiaDatabase** representa a base de dados local, utilizando a biblioteca ROOM para armazenar os dados de forma persistente. Ela define as entidades da base de dados e fornece os DAOs para a manipulação das mesmas.

Resumidamente, quando é necessário obter os dados, eles são solicitados aos repositórios. Estes decidem se irão obtê-los da base de dados local ou da API, dependendo do contexto e da disponibilidade dos dados. Destaca-se ainda que as chamadas à API são feitas utilizando a biblioteca **Retrofit**, simplificando e tornando mais eficiente a comunicação entre a aplicação e a backend.

```
public void getAppAPI(){
    Call<List<App>> call = app_api.getApp();
    call.enqueue(new Callback<List<App>>() {
        @Override
        public void onResponse(Call<List<App>> call, Response<List<App>> response) {

            if (response.isSuccessful()){

                insert(response.body().get(0));
            }


public static BraguiaDatabase getInstance(Context context){
    if (INSTANCE == null){
        synchronized (BraguiaDatabase.class){
            if (INSTANCE == null){
                INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                                                BraguiaDatabase.class,DATABASE_NAME)
                                .fallbackToDestructiveMigration()
                                .build();
            }
        }
    }
    return INSTANCE;
}


public static void populateDbAsync(BraguiaDatabase catDatabase) {
    executorService.execute(() -> {
        UserDAO userDAO = catDatabase.userDAO();
        TrailDAO trailDAO = catDatabase.trailDAO();
        AppDAO appDAO = catDatabase.appDAO();
```

```
        userDAO.deleteAll();
        trailDAO.deleteAll();
        appDAO.deleteAll();
    });
}
```

2.2.3 Serviço de localização

Tal como pretendido, a aplicação tem um serviço de localização em *background*. Para a sua elaboração, foi criado um *Service* que, pedindo as devidas permissões ao utilizador, através de um **LocationManager** faz esta monitorização. Quando este serviço está ativo (sendo possível desativá-lo nas definições da aplicação), o utilizador tem uma notificação do tipo *setOnGoing* para a sua informação. Para além da monitorização da localização atual do dispositivo/utilizador, este serviço implementa também a funcionalidade de enviar um notificação ao utilizador quando está próximo de um ponto de interesse. As coordenadas dos pontos de interesse são conseguidas através da criação de uma instância do *ViewModel* dos trails. Note-se que tanto a atualização da localização como o raio do *geofence* podem são adaptáveis (no código-fonte). O código apresentado a seguir mostra uns dos constituintes principais deste serviço.

```
public int onStartCommand(Intent intent, int flags, int startId) {
    startForeground(NOTIFICATION_ID, buildNotification());

    if (ActivityCompat
            .checkSelfPermission(this,
                Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED
                && ActivityCompat.checkSelfPermission(this,
                Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED)
        stopSelf();
    }

    locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
                                           60000, 0, locationListener);

    return START_STICKY;
}

private class MyLocationListener implements LocationListener {

    @Override
    public void onLocationChanged(@NonNull Location location) {
        Log.d("LocationService", "Latitude: "
                + location.getLatitude()
                + ", Longitude: "
                + location.getLongitude());

        System.out.println("Latitude: "
                           + location.getLatitude()
                           + ", Longitude: "
                           + location.getLongitude());
```

```
            checkForNearPOI(location);
        }
    }
```

2.2.4 Navegação utilizando Google Maps

No que toca à navegação utilizando a aplicação da Google, toda esta operação é realizada na função **setGoogleMap**. Esta função coleciona as coordenadas dos *pins* para uma lista, através de percorrer as *edges* do trail em questão. Após esta coleção, são removidas as repetidas, e retirados os pontos inicial e final. Após isto, utiliza-se um *StringBuilder* para construir o uri que vai ser utilizado no momento de abrir o *Maps*, sendo a origem o ponto inicial referido anteriormente, o destino o ponto final, e os *waypoints* os restantes elementos da lista. Por último, verifica-se se o utilizador deu as permissões necessárias e, através da criação do *mapIntent* (no qual se utiliza o uri), é então aberta a aplicação com o trajeto pronto a iniciar. O grupo optou também por acrescentar o percurso desde a localização atual do utilizador até ao ponto inicial do percurso, para o caso do utilizador decidir iniciar a navegação e ainda não se encontrar no ponto inicial.

2.3 Bibliotecas/dependências utilizadas

2.3.1 Permissões utilizadas

Para garantir o bom funcionamento e o acesso aos recursos necessários, a aplicação requer certas permissões do dispositivo. A seguir, serão descritas as permissões requeridas e as respetivas finalidades na aplicação.

- **INTERNET:** Permite que a aplicação tenha acesso à Internet.

```
<uses-permission android:name="android.permission.INTERNET" />
```

- **Localização:** Estas permitem que a aplicação tenha acesso à localização do dispositivo. Enquanto a FINE fornece uma localização mais precisa, utilizando GPS, a COARSE fornece uma localização menos precisa, utilizando, por exemplo, redes Wi-Fi.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

- **Localização em primeiro e segundo plano:** A permissão FOREGROUND_SERVICE_LOCATION permite que a aplicação tenha acesso à localização do dispositivo quando está em primeiro plano. Por outro lado, a permissão ACCESS_BACKGROUND_LOCATION permite o acesso à localização do dispositivo quando esta corre em segundo plano.

```
<uses-permission android:name="android.permission.FOREGROUND_SERVICE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION"/>
```

- **ACCESS_NETWORK_STATE:** Permite que a aplicação tenha acesso a informações sobre a rede.

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

- **FOREGROUND_SERVICE:** Permite a utilização de diversos serviços em primeiro plano.

```
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
```

- **Armazenamento Externo:** Permitem a leitura e escrita no armazenamento externo do dispositivo.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
    android:maxSdkVersion="32" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
    android:maxSdkVersion="32" />
```

- **Leitura de mídia:** Permitem que a aplicação leia imagens, vídeos e áudios do dispositivo.

```
<uses-permission android:name="android.permission.READ_MEDIA_IMAGES"/>
<uses-permission android:name="android.permission.READ_MEDIA_VIDEO" />
<uses-permission android:name="android.permission.READ_MEDIA_AUDIO" />
```

- **Notificações:** Permite que a aplicação envie notificações.

```
<uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
```

2.3.2 Bibliotecas utilizadas

Para atender às diversas necessidades da aplicação, foram incorporadas algumas bibliotecas que facilitam o desenvolvimento e melhoram a experiência do utilizador. A seguir, serão brevemente descritas as principais bibliotecas utilizadas e as respetivas funcionalidades.

- **Retrofit:** Bibliotecas utilizadas para facilitar na comunicação e consumo da API fornecida pela equipa docente.

```
implementation 'com.google.code.gson:gson:2.8.6'
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
```

- **Picasso:** Bibliotecas utilizadas para fazer download e cache das imagens usadas na aplicação. Fornece uma melhor gestão de imagens garantindo uma experiência fluída ao utilizador.

```
implementation 'com.squareup.picasso:picasso:2.8'
implementation 'com.jakewharton.picasso:picasso2-okhttp3-downloader:1.1.0'
```

- **Navigation:** Bibliotecas utilizadas para navegar de forma simplificada entre os vários fragmentos e atividades da aplicação.

```
implementation 'androidx.navigation:navigation-fragment:2.4.0'
implementation 'androidx.navigation:navigation-ui:2.4.0'
```

- **Google Services:** Bibliotecas utilizadas para integrar serviços do Google, como mapas e localização, na aplicação.

```
implementation 'com.google.android.gms:play-services-maps:18.2.0'
implementation 'com.google.android.gms:play-services-location:21.2.0'
```

2.4 Padrões de software utilizados

2.4.1 Padrão arquitetural

Tal como mencionado e explicado no capitulo 2, o padrão arquitetural recomendado pela equipa docente e utilizado pelo grupo foi o **MVVM** (*Model-View-ViewModel*). A utilização deste padrão permitiu ao grupo definir uma boa divisão entre as duas principais camadas, a lógica de negócios e a interface do utilizador.

2.4.2 Creational Pattern: Singleton

Outro padrão utilizado durante o desenvolvimento da aplicação foi o Singleton na classe **BraguiaDatabase**, este garante apenas uma única instância de uma classe, útil quando precisamos de um único controlador, como no caso, uma única instância de base de dados. O método **getInstance(Context context)**, é o ponto de acesso global que permite outras classes na aplicação obterem a instância única da base de dados. Quando chamado, se já existir uma instância da base de dados, ela é retornada, caso contrário, uma nova é criada e retornada. Isto é executado de forma sincronizada para evitar que múltiplas instâncias sejam criadas se, por exemplo, várias threads invocarem o método simultaneamente.

```
public abstract class BraguiaDatabase extends RoomDatabase {

    private static volatile BraguiaDatabase INSTANCE = null;

    public static BraguiaDatabase getInstance(Context context){
        if (INSTANCE == null){
            synchronized (BraguiaDatabase.class){
                if (INSTANCE == null){
                    INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                                                    BraguiaDatabase.class,
                                                    DATABASE_NAME)
                        .fallbackToDestructiveMigration()
                        .build();
```

```
                }
            }
        }
        return INSTANCE;
    }

}
```

2.4.3 Structural Pattern: Adapter

Outro padrão bastante utilizado durante o desenvolvimento da aplicação foi o padrão estrutural Adapter, permitindo que classes com interfaces incompatíveis trabalhem juntas. Foi utilizado para fosse possível usar a RecyclerView com algumas classes que criamos e assim obter listas otimizadas de, por exemplo, **Roteiros**, **Pontos de Interesse**, **Contactos**, entre outros. Esta abordagem para além de melhorar o desempenho da aplicação, também contribuiu para a modularidade e a manutenção do código, permitindo uma separação entre a classe dos dados e a sua visualização.

2.4.4 Behavioral Pattern: Observer

Um padrão que também desempenhou um papel crucial no desenvolvimento da aplicação foi o padrão comportamental Observer. Este permite que um objeto (Observer) seja automaticamente notificado quando ocorrem mudanças de estado noutro objeto (Observed). No contexto da nossa aplicação, este padrão foi utilizado para atualizar a interface do utilizador sempre que os dados sofriam uma alteração.

Na classe HomeFragment, mais precisamente no método **onCreateView()**, o ViewModel **trailsViewModel** é observado e quando os dados dos trails mudam, a interface do utilizador é atualizada automaticamente. Isto acontece através do método **observe()**, que aceita um LifecycleOwner e um Observer.

```
@Override
public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container,
                                                            Bundle savedInstanceState) {

    View view = inflater.inflate(R.layout.fragment_home, container, false);

    ...

    trailsViewModel.getAllTrails().observe(getViewLifecycleOwner(), trails -> {
        TrailsRecyclerViewAdapter adapter =
            new TrailsRecyclerViewAdapter(trails,
                                new TrailsRecyclerViewAdapter.TrailClickListener() {

            @Override
            public void onItemClick(int position) {
                ...
            }
```

```
        });
        recyclerView.setAdapter(adapter);
    });

    ...

    return view;
}
```

3 Mapa de navegação de GUI

Relativamente ao mapa de navegação, este pode ser gerado pelo próprio *Android Studio*. A figura seguinte mostra então o respetivo gráfico. Através da sua análise conseguimos perceber os diferentes fragmentos acessíveis através da *MainActivity*, transições essas conseguidas através da barra de navegação. Esta barra de navegação permite transitar entre *Home*, *Contact* e *Profile*. A partir de *Home*, é possível também aceder aos fragmentos *Update* (que se trata de um fragmento informativo que indica as vantagens das contas *premium*) e *Trail*. Já no *Trail*, é possível navegar para o fragmento*Pin*. Existe também o fragmento do histórico (*TrailHistory*), acessível através do fragmento do perfil.

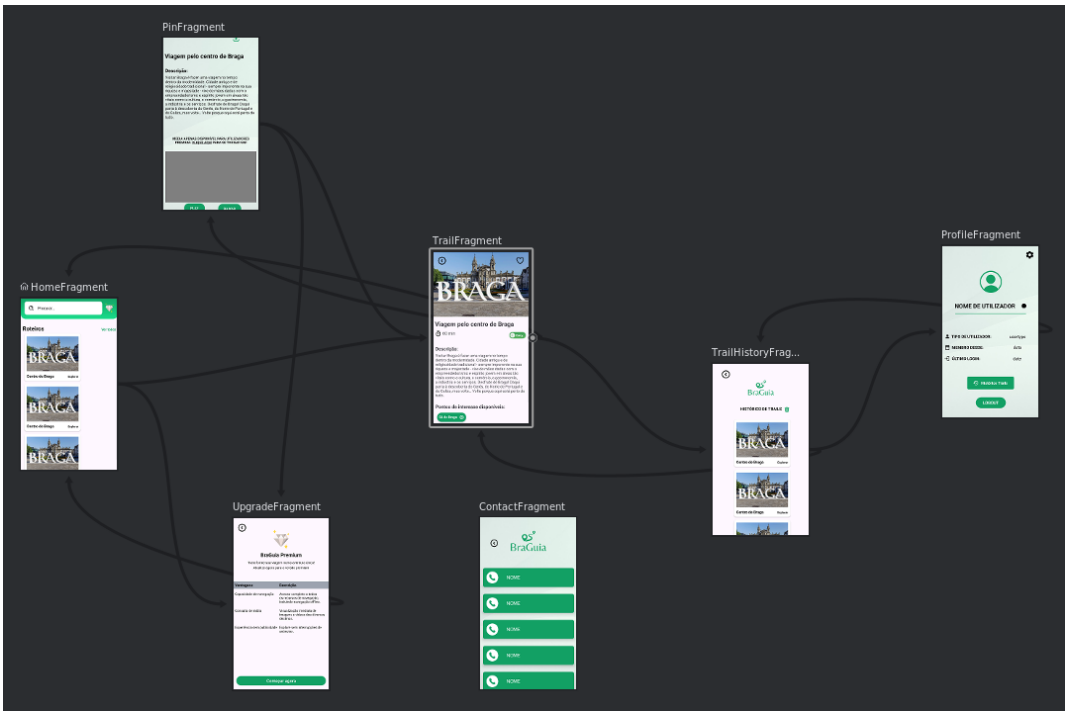


Figura 3: Mapa de navegação de GUI

4 Discussão de resultados

4.1 Trabalho realizado

Analisando cuidadosamente a lista de requisitos proposta pela equipa docente, presente no enunciado, o grupo considera que desenvolveu uma boa aplicação na medida em que praticamente todos os requisitos foram cumpridos. Existe porém a exceção, na parte dos requisitos funcionais, que é o facto de a aplicação não permitir armazenar a mídia automaticamente (localmente) para quando o utilizador se encontra sem acesso à internet. O grupo também não conseguiu realizar a etapa de testes à aplicação, devido a problemas técnicos no serviço da backend. Excluindo os dois pontos mencionados, a aplicação encontra-se a funcionar corretamente cumprindo com os restantes requisitos. 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.

- Guardar mídia localmente - Tal como mencionado no capítulo anterior, o objetivo de conseguir guardar os diferentes ficheiros de mídia de cada ponto de interesse localmente não foi alcançado. Esta é uma funcionalidade importante a implementar no futuro, de modo a que o utilizador tenha uma melhor experiência no uso da aplicação.

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.

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.

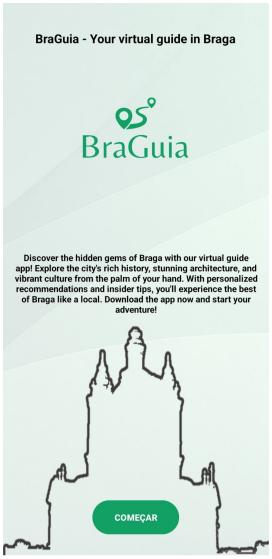
5.3 Reflexão sobre Performance individual

- Gonçalo Pereira (PG53834) - Relativamente ao serviço de localização, considero que poderia ter implementado nas definições espaço para alterar a distância das geofences e colocado um mecanismo de forma a não repetir uma notificação desse tipo previamente enviada. Gostaria de ter implementado também uma página de trails favoritos.
- Nuno Costa (PG52698) - Quando foi anunciada a primeira parte do trabalho prático de desenvolver uma aplicação nativa Android, admito que, inicialmente, não estava muito entusiasmado. No entanto, decidi encarar este desafio como uma oportunidade para aprender uma nova tecnologia e, contrariamente ao que esperava, gostei e fiquei interessado em continuar a explorar no futuro. Em relação ao resultado final, gostei do trabalho que realizámos enquanto equipa, mas acredito que poderíamos ter ido mais além e ter implementado algumas funcionalidades extras, como uma página com os roteiros favoritos do utilizador, um modo escuro funcional, entre outras.
- Santiago Domingues (PG54225) - Na parte do histórico penso que o modo como os trails são armazenados podia ser otimizado, visto que ficam armazenados num objeto *SharedPreferences*. Também era interessante ter sido implementada uma página de favoritos de modo a ser mais fácil para o cliente visualizar os seus trails preferidos com mais facilidade.

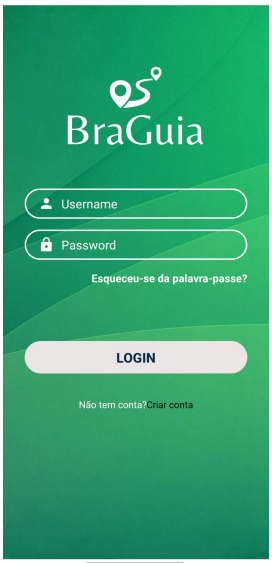
6 Conclusão

Considerando que foi a primeira vez que o grupo contactou com desenvolvimento nativo e neste caso, mais concretamente em **Android**, são notórios os conhecimentos adquiridos. Através dos desafios enfrentados e das soluções encontradas. Esta experiência não permitiu apenas explorar as complexidades do desenvolvimento para uma plataforma tão amplamente utilizada, mas também permitiu ganhar inúmeras competências, preparando-nos para projetos futuros. Como sendo uma plataforma tão utilizada, é também bastante interessante perceber o que está por detrás das diversas aplicações que utilizamos no quotidiano, obtendo assim uma nova visão das mesmas.

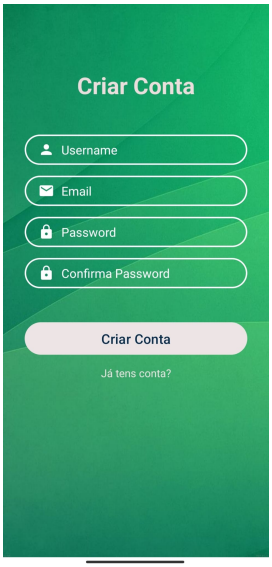
Anexo



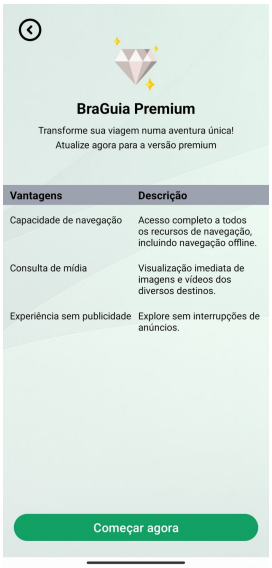
(a) Landing page.



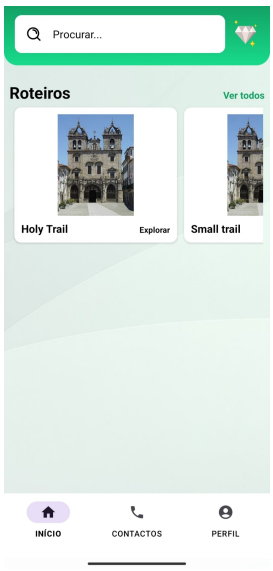
(b) Login



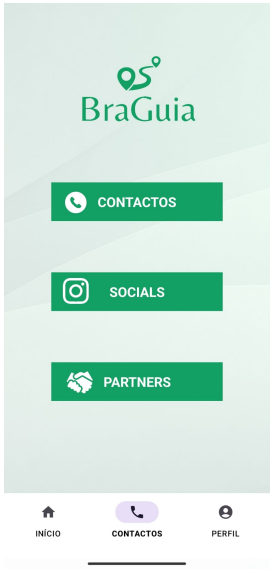
(c) Register



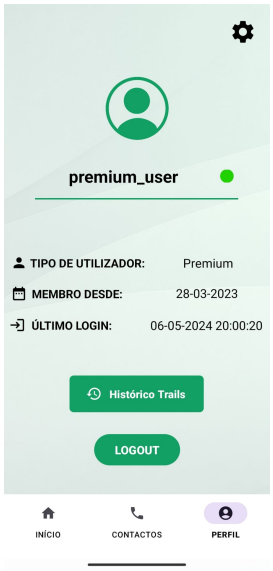
(d) Premium



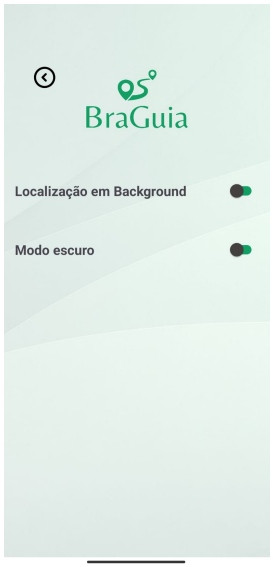
(a) Home page



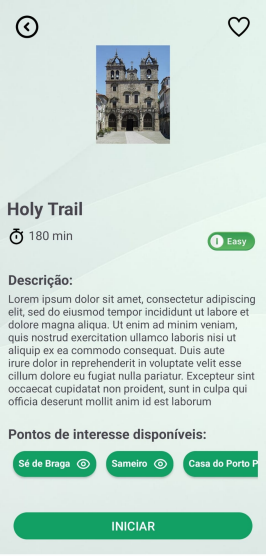
(b) Contacts



(c) Profile

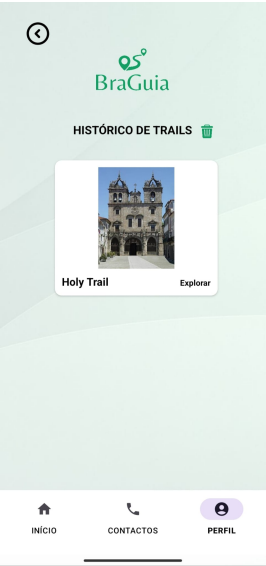


(d) Settings

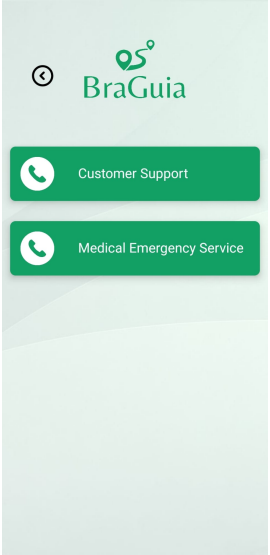


(a) Trail

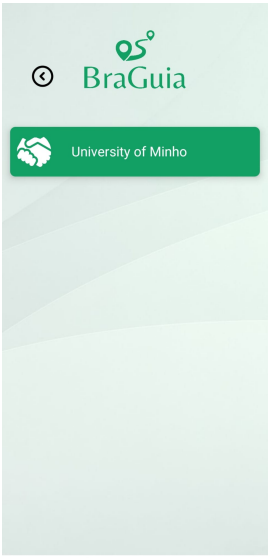
(b) Pin



(c) History



(d) Support



(a) Partners

(b) Socials