Universidade do Minho

Mestrado em Engenharia Informática

Tópicos de Desenvolvimento de Software

Trabalho Prático - **Parte I**

Guia Turístico: BraGuia

Grupo:

Catarina Gonçalves (PG50180)

Francisco Toldy (PG50379)

Joana Alves (PG50457)

28 de maio de 2023

Conteúdo

1 Introdução

Este relatório foi desenvolvido no âmbito da unidade curricular de Tópicos de Desenvolvimento de *Software* do Mestrado em Engenharia Informática, tendo como objetivo o desenvolvimento de um guia turístico, sob a forma de uma aplicação móvel híbrida.

De acordo com o enunciado do trabalho prático, a **aplicação**, denominada "BraGuia", é uma aplicação para orientação turística que oferece roteiros turísticos aos seus utilizadores, oferecendo funcionalidades de localização e navegação geográfica, reprodução de *media* acerca de pontos de interesse, entre outros. Para além disto, de forma a obter o conteúdo para mostrar ao utilizador, a aplicação consome informação de um *backend* desenvolvido pelo docente para servir especificamente este trabalho prático.

Desta forma, recorrendo ao conhecimento adquirido ao longo do semestre relativo a desenvolvimento nativo e multi-plataforma, o projeto foi separado em duas partes distintas: **parte I** (desenvolvimento com recurso apenas a tecnologia nativa *Android*) e **parte II** (desenvolvimento multi-plataforma recorrendo à tecnologia *React Native*). Assim, o objeto de estudo deste relatório trata-se da **parte I**, onde irão ser apresentados os pormenores relativos à arquitetura, comportamento e decisões tomadas pelo grupo no desenvolvimento da mesma.

2 Detalhes da Implementação

2.1 Estrutura do Projeto

A estruturação do projeto foi baseada na abordagem recomendada, que inclui o armazenamento em base de dados alimentada por comunicação com uma API externa. Desta forma, todos os ficheiros com informação relevante estão organizados nas suas devidas diretorias.

2.1.1 Manifests

Começando pela diretoria *Manifests*, constituída por um ficheiro que foi alterado para englobar todas as novas atividades criadas, entre outras configurações.

2.1.2 Resources

Nesta diretoria estarão todos os recursos que são utilizados na aplicação como um todo:

- **Drawables:** contém todos os *icons*/imagens utilizadas
- **Layout:** contém todos os *layouts* das atividades e de elementos que são listados
- **Values:** contém todos os códigos das cores utilizadas como também strings constantes e a chave da API do *Maps* utilizada pela aplicação.

2.1.3 Gradle Scripts

O *build.gradle* que se encontra em *Gradle Scripts* foi bastante utilizado para adicionar dependências necessárias para permitir a existência de algumas das funcionalidades.

2.1.4 Java

É nesta diretoria onde existe um maior número de incrementos e alterações. Nesta estão primeiramente os ficheiros *java* correspondentes a cada uma das atividades que a aplicação disponibiliza.

Além destas atividades, existem outras 8 diretorias:

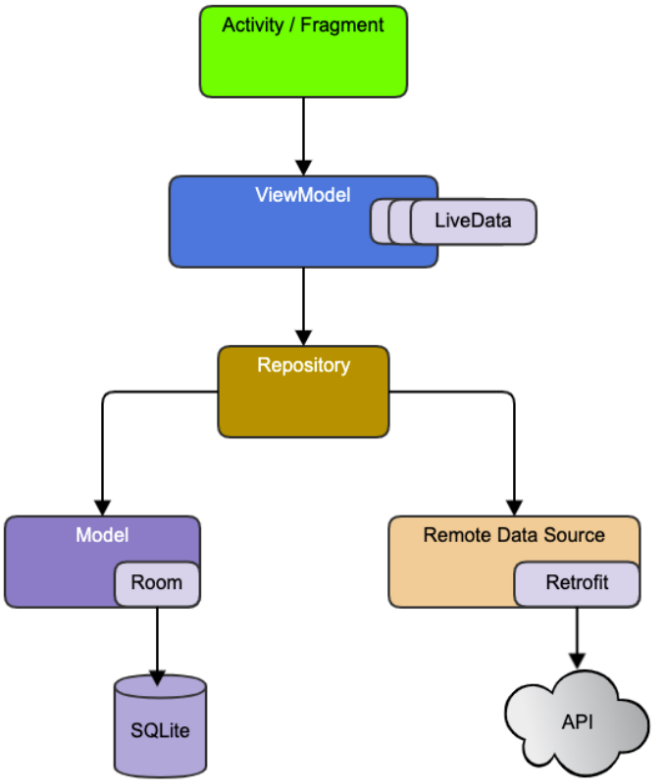
- **Adapters:** Contém todos os adaptadores utilizados nas listagens de diferentes objetos
- **Converters:** Contém apenas um ficheiro onde existe a conversão entre *Date* e *TimeStamp*
- **DAO:** Contém todos os ficheiros *DAO* (um para cada tipo de dados guardado em base de dados) que permitem a execução de *queries* à base de dados para futuro uso nos repositórios
- **Data:** Contém todos os ficheiros que fazem a representação dos dados de cada entidade que é guardada em base de dados
- **BD:** Contém apenas um ficheiro que é utilizado na criação da única instância da base de dados utilizada.

- **Repositories:** Contém um repositório para cada entidade que existe persistida, sendo este um objeto que permite executar o processo de inicialização de cada tipo de dados (busca à Base de dados ou API) como também outras operações que podem ser realizadas na base de dados que vão ser redirecionadas para o seu *DAO* correspondente. O repositório de *trails* é diferente dos restantes no sentido em que existe um sistema de *caching*, que vai obrigar o *fetch* de *trails* após 3 dias passados desde o último *fetch*.

- **Services:** Esta diretoria é dedicada ao *Service* único relacionado com o envio de notificações. Este *Service* único pode ser inicializado por duas atividades diferentes.

- **ViewModel:** Nesta diretoria estão contidos todos os *ViewModels* correspondentes a cada uma das atividades, sendo que cada um irá conter os repositórios necessários para a informação que cada atividade quer disponibilizar.

Assim sendo, a estrutura do projeto vai seguir a abordagem recomendada: cada atividade deverá ter uma instância de um *View Model* único para ela, e este deverá conter os repositórios necessários para poder aceder aos dados que a atividade necessitar a partir de dados do tipo *LiveData*. Cada repositório de cada tipo de dados deverá ter também uma instância do classe *DAO* da entidade correspondente para efetuar as *queries* à base de dados. Assim, segue o formato do seguinte gráfico em exceção à utilização do *retrofit* que foi substituída por **Volley**:



Concluindo, com esta estruturação do projeto resultou o seguinte esquema da base de dados juntamente com o diagrama de classes criado. Devido à extensão do diagrama de classes, o grupo optou por mostrar segmentos do diagrama focados em cada atividade, sendo que o diagrama de classes como um todo será enviado em anexo juntamente com este documento:

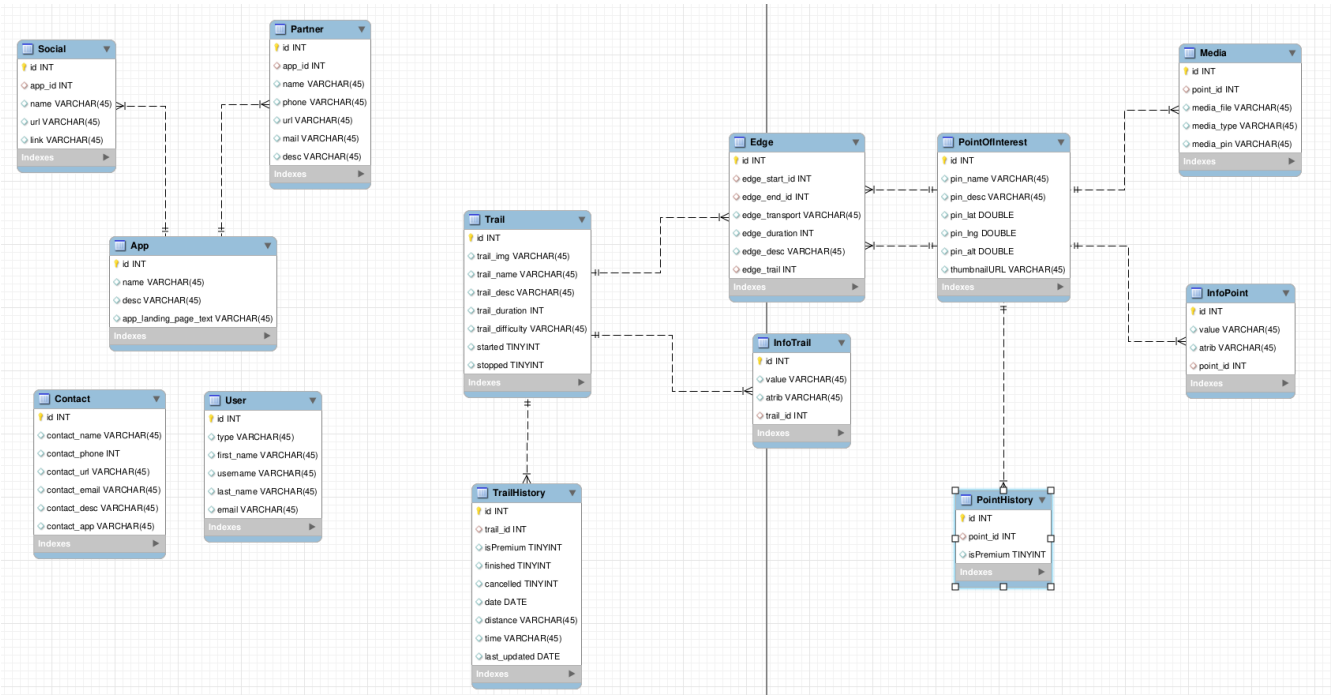


Figura 1: Esquema da base de dados

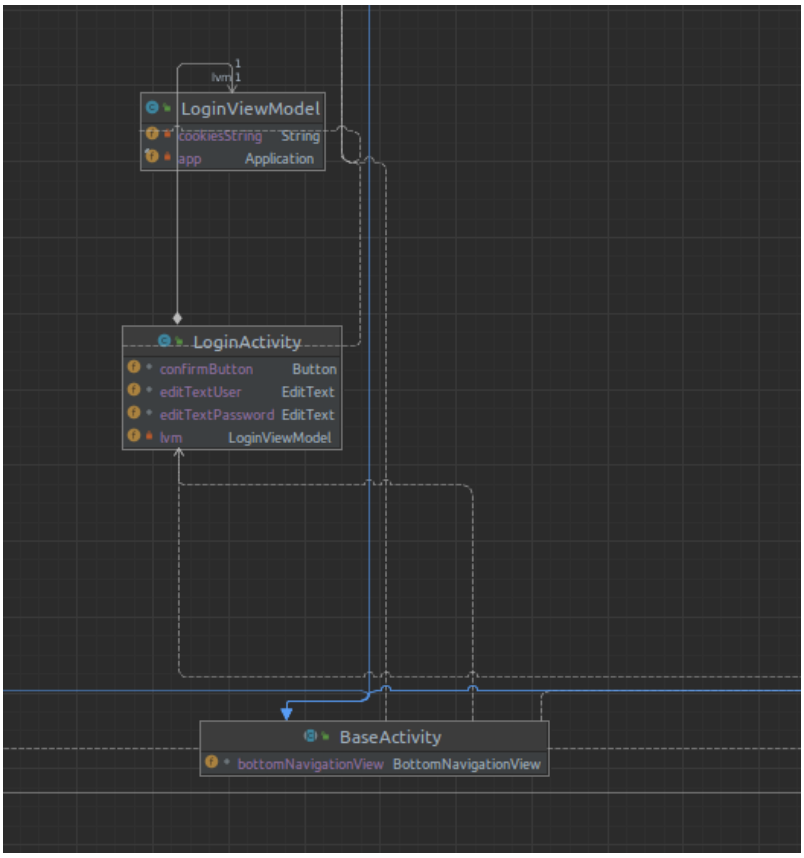


Figura 2: Login Activity

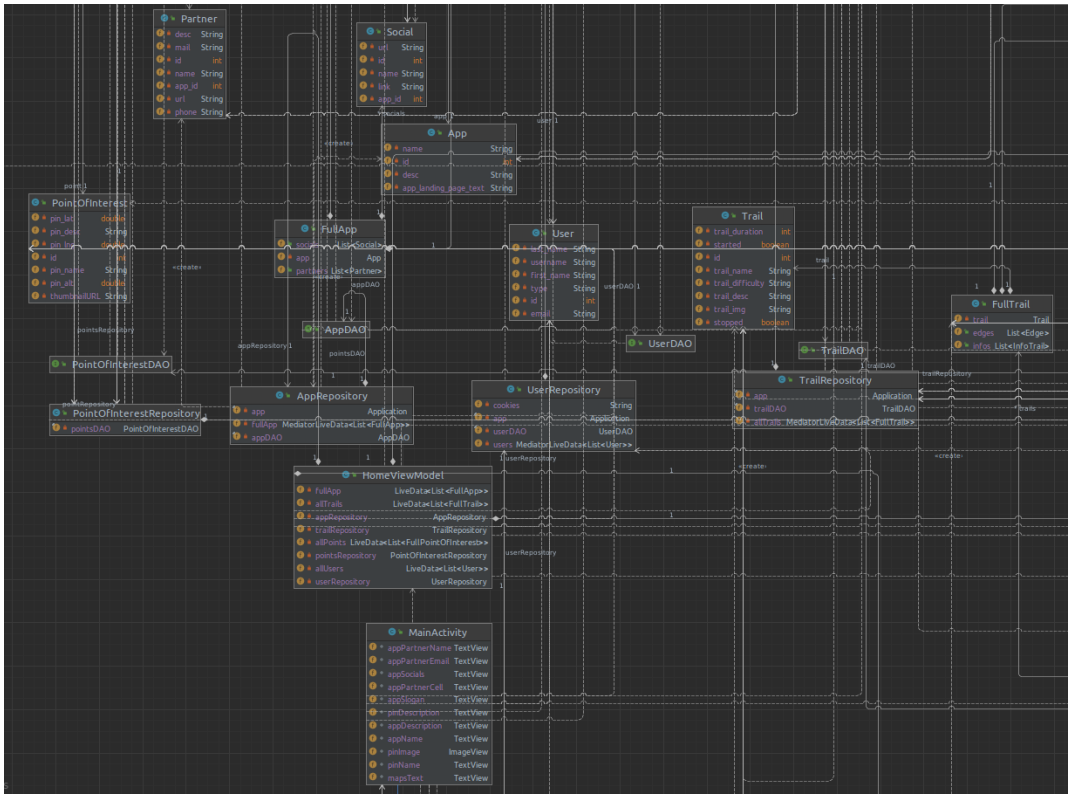


Figura 3: Main Activity

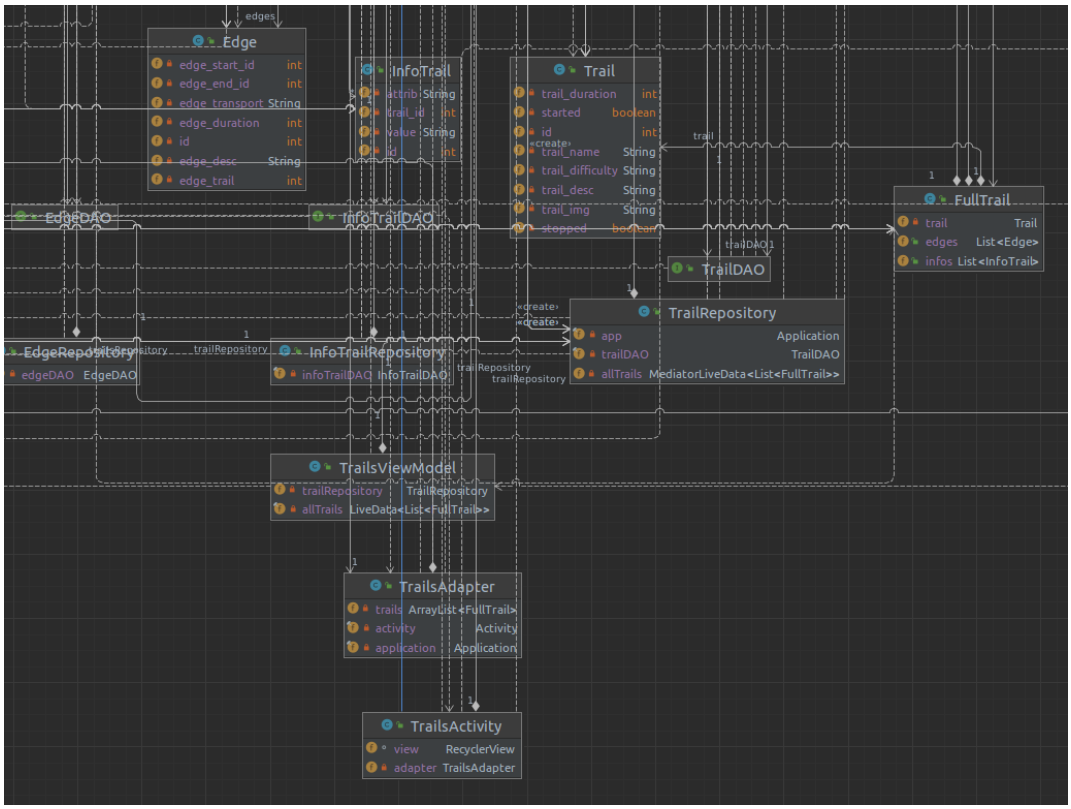


Figura 4: Trails Activity

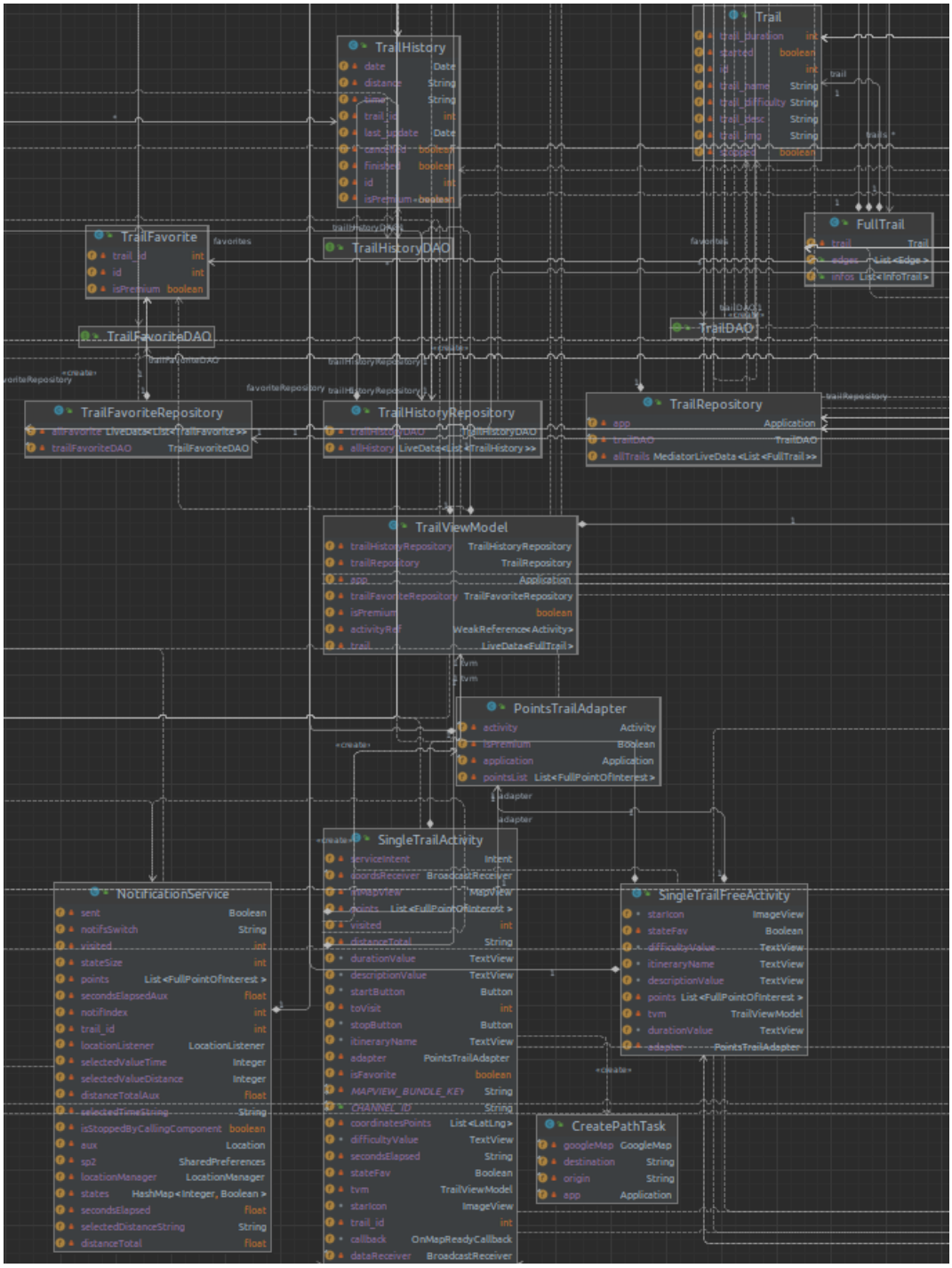


Figura 5: Single Trail Activity,Single Trail Free Activity e NotificationService

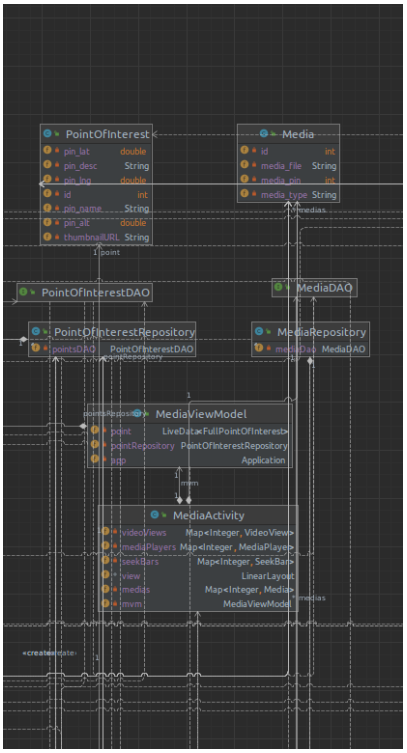


Figura 6: Media Activity

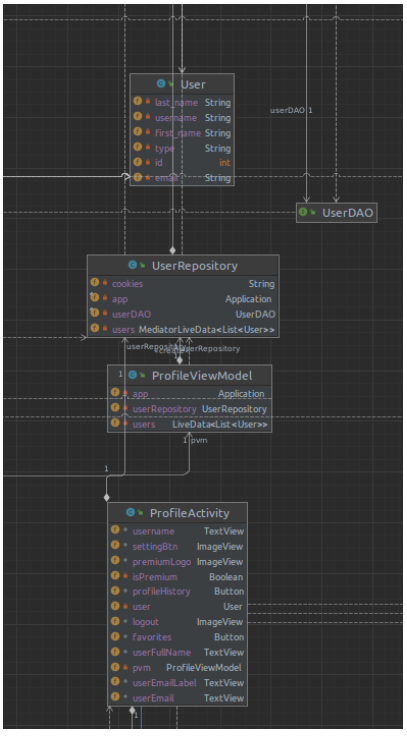


Figura 7: Profile Activity

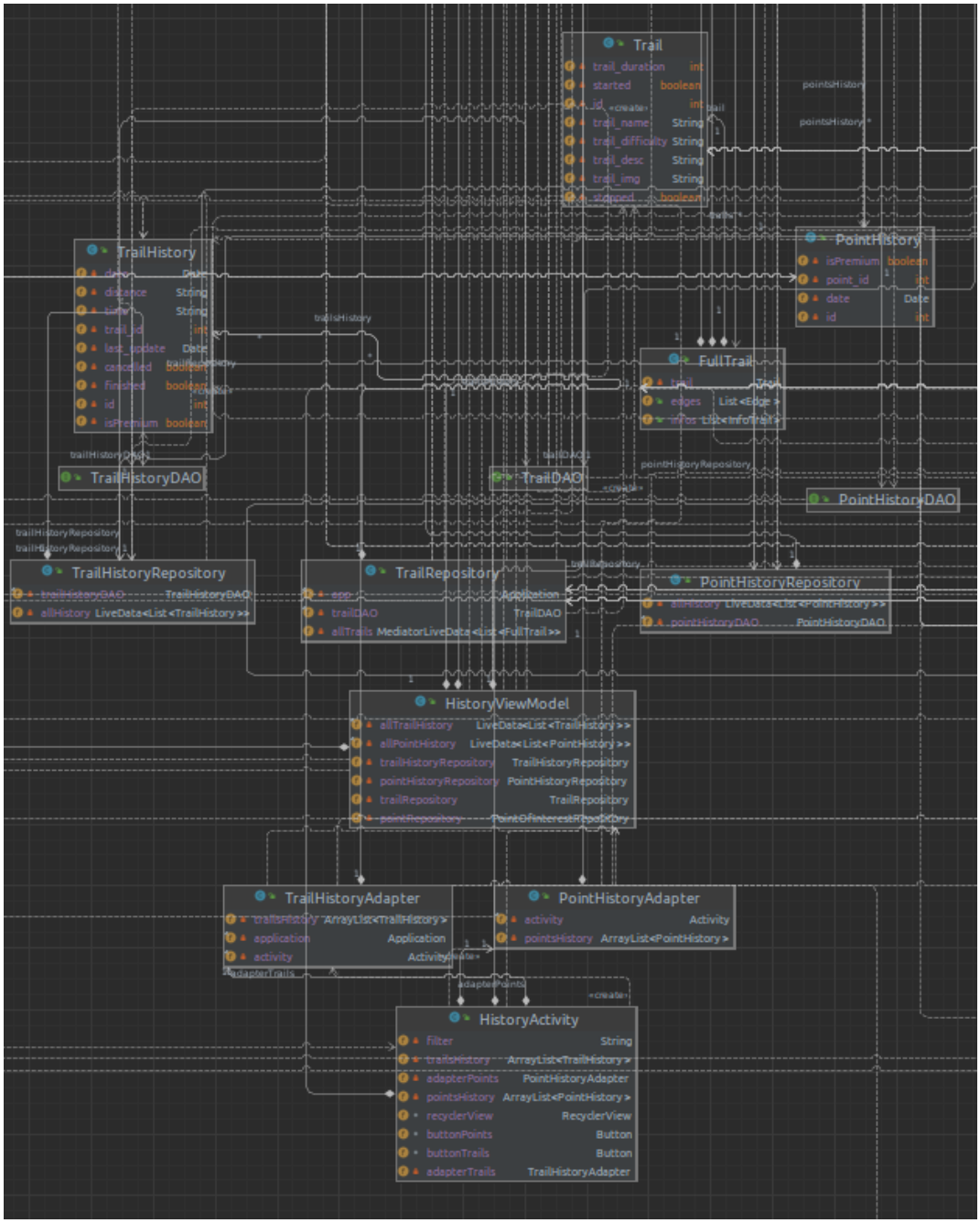


Figura 8: History Activity

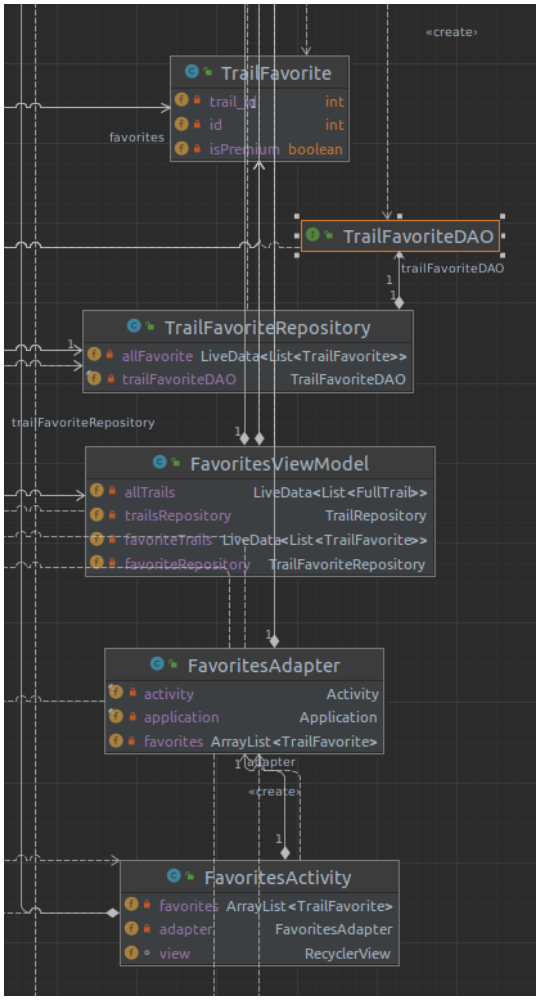


Figura 9: Favorites Activity

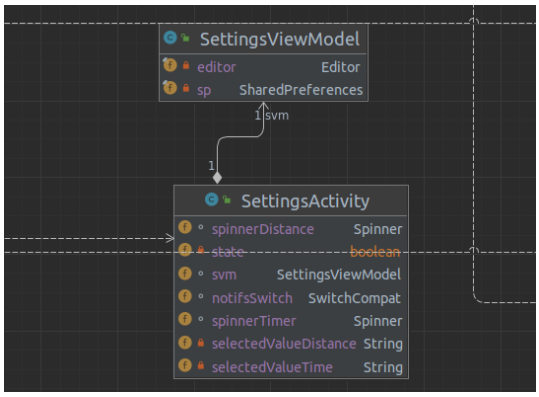


Figura 10: Settings Activity

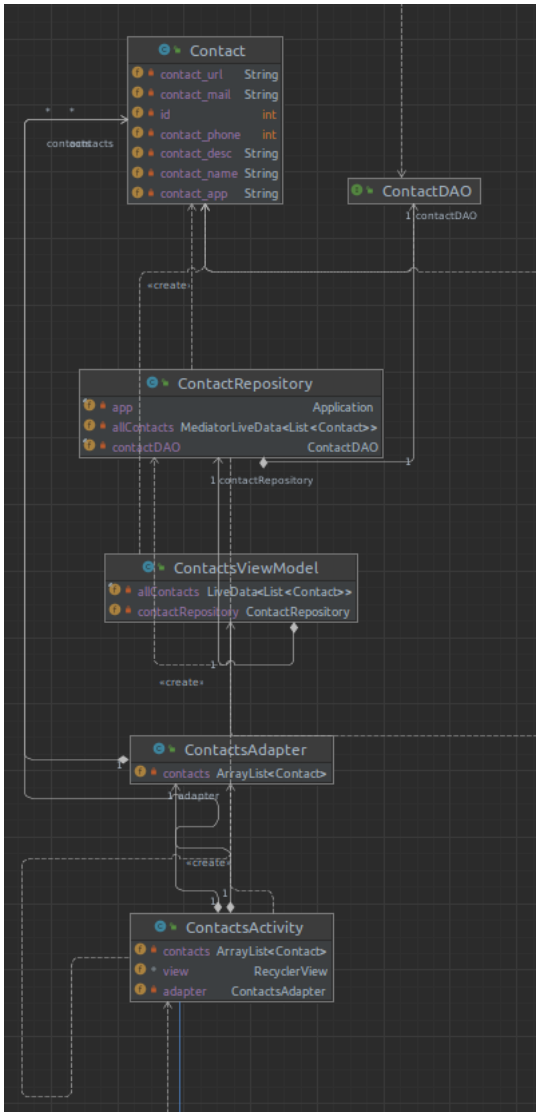


Figura 11: Contacts Activity

2.2 Soluções de Implementação

Nesta secção serão então apresentadas as soluções de implementação de cada serviço ou atividade desenvolvida juntamente com a identificação do requisito que originou a criação dessas mesmas atividades ou funcionalidades extra que a equipa desejou implementar.

2.2.1 Login Activity

- **Requisito:** A aplicação deve permitir efetuar autenticação

Atividade dedicada ao *Login*, recolhe o *input* do utilizador e envia a informação para o seu *ViewModel*, que irá fazer um pedido à API recorrendo ao um *JsonObjectRequest*. Além disso, através também do seu *ViewModel*, irá fazer *parse* da *NetworkResponse* recebida, armazenando os *headers* **sessionid** e **csrftoken** via o método *saveCookies*. Em caso de autenticação correta, irá redirecionar o utilizador para a página inicial.

2.2.2 Base Activity

- **Objetivo:** Automatizar o desenvolvimento das atividades contendo *nav bar*

Esta classe surgiu como uma necessidade de automatizar o desenvolvimento de atividades que utilizassem a barra de navegação, isto é, atividades que correspondessem aos elementos selecionados na barra de navegação. Desta forma, esta classe faz *extend* à **AppCompatActivity**, implementando o método *onCreate* e incluindo métodos abstratos para definição tanto do conteúdo do ecrã como do item selecionado na *nav bar*. Assim, uma atividade que necessite de possuir a *nav bar* no seu *layout* apenas tem de fazer *extend* a esta mesma classe e implementar os métodos abstratos.

2.2.3 Main Activity

- **Requisitos:**
  - A aplicação deve possuir uma página inicial onde apresenta as principais funcionalidades do guia turístico, descrição, etc
  - A aplicação deve assumir que o utilizador tem o Google Maps instalado no seu dispositivo (e notificar o utilizador que este software é necessário)
- **Funcionalidade Extra:** Apresentação do ponto de interesse do Dia

Esta atividade, tal como referido nos requisitos, trata-se da página inicial da aplicação onde são apresentadas as principais funcionalidades da mesma, assim como uma breve descrição e os seus contactos. Para além disto, o grupo decidiu adicionar uma secção na página para colocar em destaque um ponto de interesse presente na aplicação, de modo a facilitar a utilização a um utilizador menos experiente ao fornecer um primeiro ponto de interação.

Além disso, de forma a garantir recuperação do roteiro após fecho abrupto da aplicação, esta atividade vai também verificar se o serviço de notificação deveria estar a correr ou não, e caso devesse e não esteja, aciona o serviço *NotificationService*.

2.2.4 Trails Activity

- **Requisito:** A aplicação deve mostrar num ecrã, de forma responsiva, uma lista de roteiros disponíveis
- **Funcionalidade Extra:** Os trails devem ser atualizados de 3 em 3 dias após a primeira atualização na base de dados

De modo a ser possível a apresentação de uma lista de todos os roteiros disponíveis, foi utilizado um *RecyclerView* que permite, de uma forma mais eficiente, apresentar uma lista de elementos. Assim, esta atividade consiste simplesmente na criação do *adapter* correspondente e o envio da lista de roteiros para esse mesmo *adapter*.

Esse mesmo *adapter* ("TrailsAdapter") vai fazer a atribuição dos dados da classe *Trail* ao *layout* definido para um roteiro na interface. Além disso, vai permitir a apresentação da imagem to *trail* em modo *offline* graças à cópia local das *thumbnails* de cada roteiro.

2.2.5 Single Trail Activity

- **Requisitos:**
  - A aplicação deve suportar 2 tipos de utilizadores: utilizadores *standard* e utilizadores *premium*
  - A navegação proporcionada pelo *Google Maps* deve poder ser feita de forma visual e com auxílio de voz, de modo a que possa ser utilizada por condutores
  - A aplicação deve possuir a capacidade de iniciar um roteiro
  - A aplicação deve possuir a capacidade de interromper um roteiro
  - A aplicação deve mostrar, numa única página, informação acerca de um determinado roteiro: galeria de imagens, descrição, mapa do itinerário com pontos de interesse e informações sobre a *media* disponível para os seus pontos
- **Funcionalidade Extra:** Adicionar/Retirar dos Favoritos

Esta atividade tem como objetivo disponibilizar um roteiro completo a um utilizador *premium* da aplicação. Para tal, além de mostrar as diversas informações básicas associadas ao roteiro (*thumbnail*, nome, descrição, entre outros), vai também disponibilizar um mapa dentro da atividade que mostra os vários pontos de interesse bem como uma *PolyLine* representativa do itinerário que o utilizador irá seguir. Na parte de cima existem 2 botões, para iniciar e parar um roteiro. O botão Stop encontra-se bloqueado até o utilizador iniciar o roteiro.

O mapa mostrado é preenchido com informação recorrendo a informação disponível sobre o roteiro. Começam por ser adicionados os marcadores dos pontos do roteiro e depois é executada uma *Task* que vai, de forma assíncrona, fazer um pedido à API do *GoogleMaps* de forma a receber a *Polyline* correspondente ao itinerário entre os pontos, e desenhar essa *Polyline* no mapa.

Quando o utilizador inicia o roteiro, é iniciado o *service* de Notificações - *NotificationService*. Esse serviço começa por fornecer a localização atual do utilizador de forma a ser executado um redirecionamento para a aplicação *GoogleMaps*, com o pedido de itinerário já preenchido com cada uma das paragens relativas aos pontos de interesse. Além deste redirecionamento, o roteiro também é adicionado ao histórico do utilizador.

O botão de Stop vai parar o *Service*, sendo que depois um *Receiver* vai processar a mensagem final do *Service*. Esse processamento irá incluir a criação de uma caixa de diálogo que informa o utilizador de algumas estatísticas do roteiro recém-terminado.

Por último, esta atividade disponibiliza ao utilizador uma lista dos pontos de interesse por via de um *adapter* que irá adicionar os pontos de interesse no lugar respetivo, indicando também ao *adapter* se se trata de um utilizador *premium* ou não. Como funcionalidade adicional, o utilizador pode adicionar o roteiro ou retirar o roteiro dos favoritos no ícone correspondente.

2.2.6 Single Trail Free Activity

- **Requisito:** A aplicação deve suportar 2 tipos de utilizadores: utilizadores *standard* e utilizadores *premium*
- **Funcionalidade Extra:**Adicionar/Retirar dos Favoritos

Atividade com o objetivo de apresentar a versão da página de um dado roteiro para utilizadores gratuitos. Assim, foram retirados os botões de Start e Stop bem como o mapa interno à aplicação, sobrando apenas a informação básica do roteiro, juntamente com a listagem dos pontos de interesse. Tal como para a versão *premium*, esta atividade também tem a funcionalidade extra de poder adicionar/retirar o roteiro aberto dos favoritos.

2.2.7 Media Activity

- **Requisitos:**
  - Para utilizadores *premium*, a aplicação deve possibilitar a capacidade de navegação, de consulta e descarregamento de *media*
  - A aplicação deve ter a capacidade de apresentar e produzir 3 tipos de *media*: voz, imagem e vídeo
  - A aplicação deve possuir a capacidade de descarregar *media* do *backend* e alojá-la localmente, de modo a poder ser usada em contextos de conectividade reduzida
- **Funcionalidade Extra:** Ficheiros de áudio com controlo de *playback*

Esta atividade apresenta todos os ficheiro de *media* de um ponto de interesse em específico, dando a possibilidade de interagir com os mesmos (no caso de áudio e vídeo) e, por fim, realizar o seu *download*, sendo apenas possível executar *download* coletivo, uma vez que o grupo concluiu que a funcionalidade de descarregar um elemento de *media* individualmente não seria tão utilizada como um *download* geral.

2.2.8 Point Of Interest Activity

- **Requisitos:**
  - A aplicação deve suportar 2 tipos de utilizadores: utilizadores *standard* e utilizadores *premium*
  - A aplicação deve possuir uma página que mostre toda a informação disponível relativa a um ponto de interesse: localização, galeria, *media*, descrição, propriedades, etc

- **Funcionalidade Extra:** Redirecionamento da localização do ponto para o *Google Maps*

Esta atividade apresenta toda a informação de um ponto de interesse, como o seu nome, descrição e propriedades. Incluímos uma imagem do ponto de interesse de modo a apresentar algo representativo do mesmo, sendo substituída por uma imagem genérica da cidade de Braga no caso em que o ponto não possua imagens na sua *media*. Para além disto, ao fazermos *display* da sua localização, incluímos um *link* que irá redirecionar o utilizador para a aplicação *Google Maps* apontando para morada do ponto.

2.2.9 Profile Activity

- **Requisitos:**
  - A aplicação deve suportar 2 tipos de utilizadores: utilizadores *standard* e utilizadores *premium*
  - A aplicação deve possuir uma página de informações acerca do utilizador atualmente autenticado
- **Funcionalidade Extra:** Redirecionamento para a página de Favoritos

Esta atividade apresenta toda a informação do utilizador autenticado e disponibiliza o acesso às funcionalidades de consulta de *trails* favoritos e *log out* para ambos os tipos de utilizador, sendo o utilizador redirecionado para as páginas devidas. Para além disto, no caso do utilizador *premium*, são também apresentados ícones para acesso às definições e histórico de *trails* e pontos de interesse visitados.

2.2.10 History Activity

- **Requisito:** A aplicação deve guardar (localmente) o histórico de roteiros e pontos de interesse visitados pelo utilizador
- **Funcinalidade Extra:** A aplicação apresenta ao utilizador cada roteiro com um identificador do seu estado: iniciado/cancelado/finalizado

Semelhante ao método utilizado para ser apresentado a lista de *trails*, o mesmo método é utilizado nas listagens do histórico. No entanto, vai ser possível ao utilizador selecionar através de 2 botões quais listagens deseja visualizar: a de roteiros ou de pontos de interesse.

Para cada uma das listagens existirá um *adapter* correspondente ("TrailHistoryAdapter"e "PointHistoryAdapter") em que ambos têm o mesmo objetivo: colocar os dados do Roteiro/Ponto no *layout* correspondente.

2.2.11 Favorites Activity

- **Funcionalidade Extra:** A aplicação apresenta ao utilizador uma lista de roteiros declarados como favoritos

O método utilizado na apresentação da lista de *trails* é utilizado de forma semelhante nas listagens dos favoritos, juntamente com um *adapter* que permite a inserção de dados do roteiro favorito no *layout* respetivo.

2.2.12 Contacts Activity

- **Requisito:** A aplicação deve possuir a capacidade de efetuar chamadas para contactos de emergência da aplicação através de um elemento gráfico facilmente acessível na aplicação

O método utilizado na apresentação da lista de trails é utilizado semelhantemente nas listagens dos contactos, juntamente com um *adapter* que permite a inserção de dados do contacto no *layout* respetivo. Além disso, foi adicionado um ícone que permite o redirecionamento para as chamadas já com o contacto apresentado. Isto foi possível através da utilização de *ACTION_DIAL*.

2.2.13 Settings Activity

- **Requisitos:**
  - A aplicação deve possuir um menu com definições que o utilizador pode manipular
  - A aplicação deve possuir a capacidade de ligar, desligar e configurar os serviços de localização

Em conformidade com os requisitos, esta atividade permite ao utilizador premium alterar diversas definições relativas às notificações e à localização, nomeadamente:

- Distância ao ponto de interesse para gerar notificação
- Intervalo de tempo entre medições da localização
- Ligar e desligar as notificações

De forma a poder mostrar as definições selecionadas no momento em que a *activity* é aberta, é feito um *GET* dos valores armazenados. Após uma alteração de qualquer uma das definições, estas são automaticamente guardadas através do *ViewModel* respetivo, via o método *saveSettings*. A alteração das definições é feita via um *Switch* no caso de ligar/desligar as notificações e *Spinners* nas restantes definições.

2.2.14 Notification Service

- **Requisitos:**
  - A aplicação deve possuir a capacidade de emitir uma notificação quando o utilizador passa perto de um ponto de interesse
  - A notificação emitida quando o utilizador passa pelo ponto de interesse deve conter um atalho para o ecrã principal do ponto de interesse

De forma a proporcionar um sistema de notificações ao utilizador durante um roteiro, foi criado o serviço *NotificationService*. Este serviço é inicializado num de dois cenários:

- **Cenário A:** o utilizador *premium* iniciou um roteiro na página do roteiro
- **Cenário B:** A aplicação foi aberta após ter sido encerrada abruptamente a meio de um roteiro

Após o roteiro ser iniciado, este serviço *Foreground* vai fazer verificações periódicas (consoante o definido pelo utilizador) da localização, comparando com as localizações de todos os pontos de interesse de forma a tomar uma decisão sobre emitir uma notificação ou não.

Caso o roteiro seja parado pelo utilizador, o serviço emite uma *Broadcast* para a atividade do Roteiro, enviando os dados necessários para a caixa de diálogo com estatísticas do roteiro interrompido/terminado. Caso o roteiro seja interrompido por um fecho propositado/abrupto, o serviço vai armazenar nas *SharedPreferences* informação necessária para continuação do roteiro no momento em que a aplicação volta a ser aberta, deixando ao utilizador a hipótese habitual de encerrar o roteiro. Nessa situação, o serviço é iniciado na página principal após reabertura da aplicação, retomando imediatamente a contagem da distância percorrida e tempo do roteiro.

Esse cálculo de distância é feito ao somar as distâncias entre cada medição que é feita, sendo um raciocínio análogo utilizado no cálculo de tempo do roteiro, somando os intervalos de medição.

2.3 Bibliotecas/Dependências Utilizadas

Nesta secção vamos apresentar uma listagem de todas as dependências do projeto presentes no ficheiro ***build.gradle***, explicando o contexto de utilização de cada uma:

- **Constraint Layout:** Utilizado para criar *layouts* grandes e complexos com uma *flat view hierarchy*, possibilitando ter uma UI responsiva.
- **Material Design:** Utilizado para desenvolvimento de componentes pré-feitos que suportam as melhores práticas de *design* da interface do utilizador.
- **Picasso:** Esta biblioteca foi utilizada para carregar mais facilmente as imagens provindas de URLs para as diversas *ImageView* necessárias.
- **Diversas dependências relacionadas com localização/Maps:** Para poder acomodar o uso da API do GoogleMaps na aplicação, bem como aceder à localização do utilizador ao longo da utilização
- **Room**: Necessário para a criação dos componentes Room para permitir a persistência

2.4 Padrões de *Software* Utilizados

Os *design patterns* são **soluções** para problemas comuns em *design* de *software* para melhorar a reutilização, compreensão, expansibilidade e manutenção do código. Assim, nesta secção, vamos apresentar todos os *design patterns* utilizados no desenvolvimento da aplicação, dividindo os mesmos pelos três tipos: criação, estruturais e comportamentais.

2.4.1 Padrões de Criação

Tal como referido acima, começamos pelos padrões de criação que, por definição, representam abstração e uniformização do processo de criação do objeto. Assim, apresentamos os vários padrões de criação encontrados no projeto:

- **Singleton:** Utilizado na criação da base de dados, permitindo assim que apenas uma seja criada e que essa seja sempre retornada quando os repositórios a utilizarem:

```
public static RoomDB getDatabase(final Context context) {
    if (INSTANCE == null) {
        *** create database ***
    }
    return INSTANCE;
}
```

- **Builder:** Utilizado na criação de Notificações com a utilização da classe *NotificationBuilder*, na criação da caixa de diálogo e, finalmente, na criação da base de dados:

```
INSTANCE = Room.databaseBuilder(context.getApplicationContext(), ...)
                        .fallbackToDestructiveMigration()
                        .build();
```

2.4.2 Padrões Estruturais

Os padrões estruturais especificam como compor e montar diferentes componentes de *software*, definindo interoperabilidade dos mesmos. Assim, passamos a apresentar os diversos padrões estruturais implementados no projeto:

- **Adapter:** O *recyclerViewAdapter* representa um *Adapter* na forma em que serve como ponte entre a *RecyclerView* e os dados que vão ser apresentados. Para isto acontecer, basta que o *adapter* implemente os métodos requisitados:

```
@NonNull
@Override
public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    ... *** Utilizado para representar um novo objeto da listagem ***
}

@Override
public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
    ... *** Atribuição de dados de um item ao layout correspondente ***
}


@Override
public int getItemCount() {
    ... *** Retorna o número de itens da listagem ***
}
```

- **Composite:** Este padrão foi utilizado de maneira indireta, uma vez que as *Android Views* utilizadas no projeto seguem este padrão.

2.4.3 Padrões Comportamentais

Por último, os padrões comportamentais servem para definir padrões de comunicação comuns entre objetos, abstraindo algoritmos e responsabilidades. Assim, apresentamos os padrões comportamentais identificados:

- **Observer:** Este padrão é aplicado para obter os dados através do *ViewModel* da atividade correspondente, onde aplicamos um *observer* nos dados (*LiveData*) retornados.

```
HomeViewModel avm = new ViewModelProvider(this).get(HomeViewModel.class);
LiveData<Data> livePoint = avm.getPoint();
livePoint.observe(this, new Observer<FullPointOfInterest>() {
    @Override
    public void onChanged(FullPointOfInterest point) {
        if (point != null) setPointInfo(point);
    }
});
```

3 Mapa de Navegação de GUI

Nesta secção serão apresentados dois mapas de navegação da aplicação, um para cada tipo de utilizador. Os mapas demonstram as opções de navegação que o utilizador *Premium* e o utilizador *Standard* podem recorrer, sendo que as indicações das opções de navegação resultantes da *nav bar* foram apenas demonstradas no ecrã principal para reduzir a redundância. Os mapas também estarão disponibilizados nos anexos:

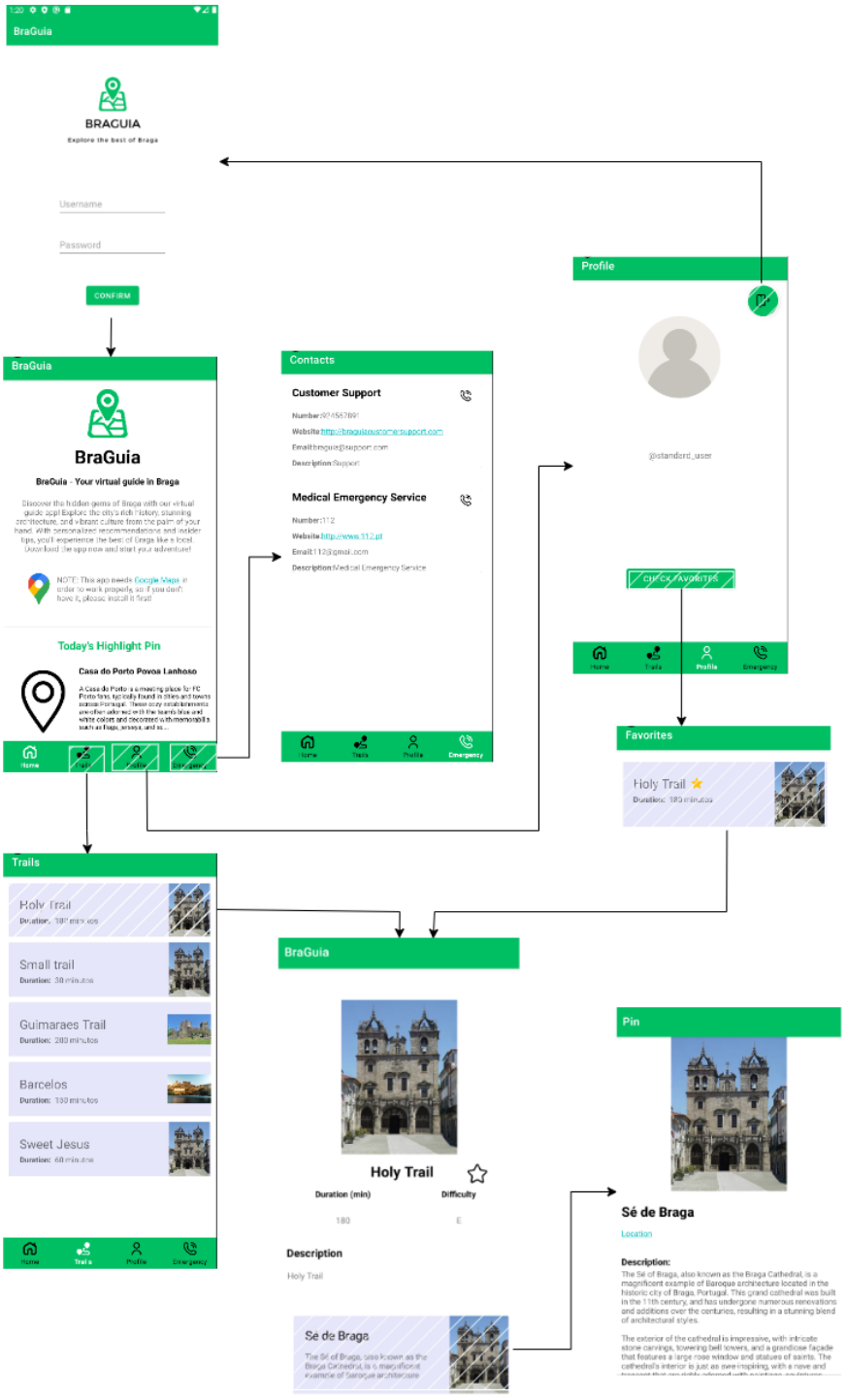


Figura 12: Mapa de Navegação utilizador Standard

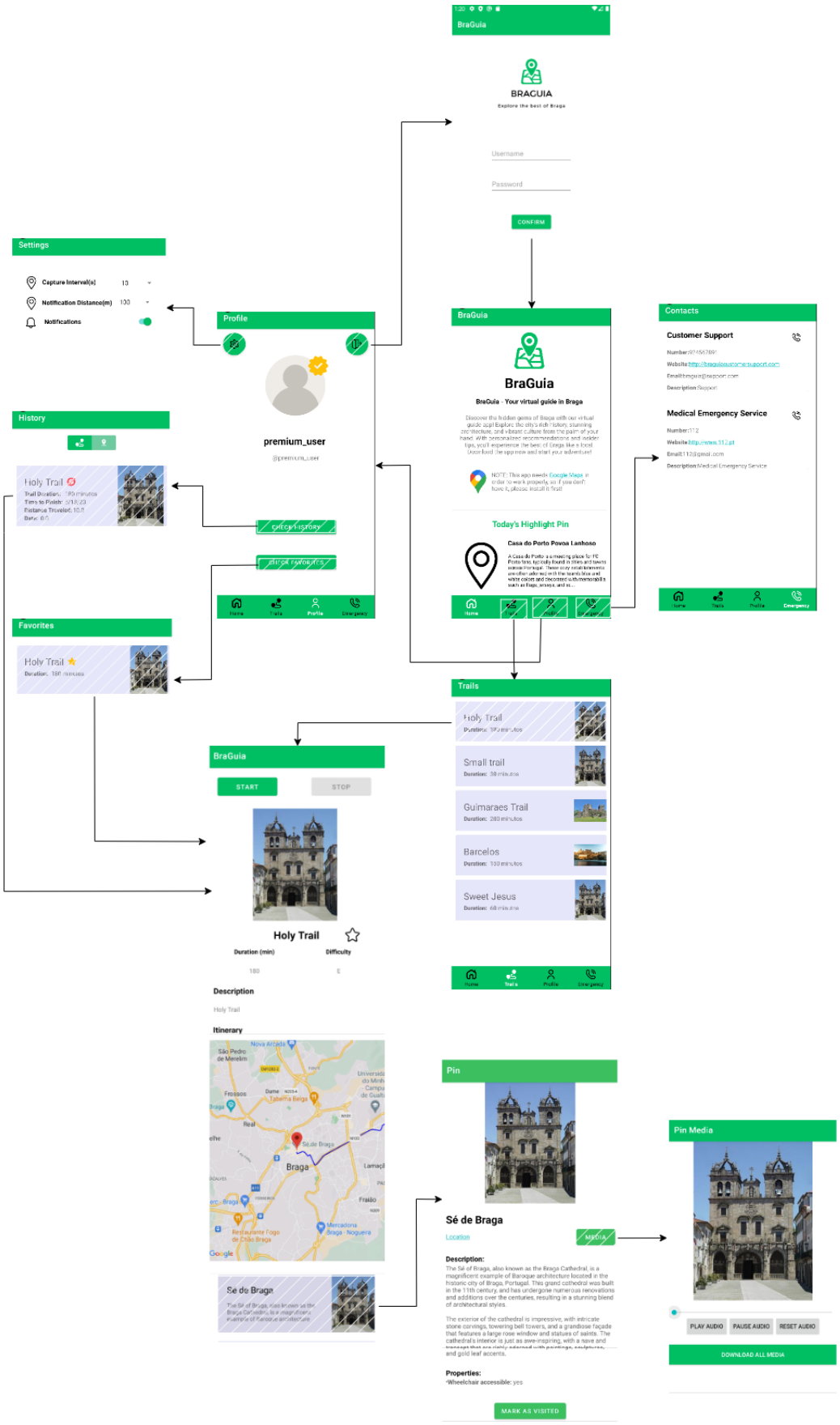


Figura 13: Mapa de Navegação utilizador Premium

4 Funcionalidades

Nesta secção iremos apresentar as diversas funcionalidades da aplicação, podendo estas ser funcionalidades obrigatórias provindas dos requisitos do enunciado, assim como extras implementados pelo grupo. Desta forma, começamos pelas funcionalidades específicas de um utilizador *standard*, notando que o utilizador *premium* é uma extensão deste, isto é, herda as suas funcionalidades.

4.1 Funcionalidades Utilizador *Standard*

Para os utilizadores *standard*, isto é, os utilizadores com **menos** privilégios na aplicação, são oferecidas as seguintes funcionalidades, notando que todas as funcionalidades exceto a autenticação são apenas acessíveis a utilizadores previamente **autenticados**:

- **Autenticação:** Os utilizadores poderão realizar a sua autenticação, com os seus dados correspondentes.
- **Informações da Aplicação:** Na página inicial, qualquer utilizador irá poder visualizar toda a informação da aplicação, assim como um ponto de interesse do dia, escolhido aleatoriamente.
- **Listagem de *Trails*:** Todos os utilizadores têm acesso a uma listagem de todos os *trails* do sistema, incluindo o seu nome, imagem representativa e duração esperada (em minutos)
- **Informação de um *Trail*:** Todos os utilizadores conseguem visualizar a informação de um *trail*, a sua imagem representativa, nome, duração, dificuldade, descrição e pontos de interesse do mesmo
- **Informação de um Ponto de Interesse:** Todos os utilizadores, através da página de um *trail*, conseguem aceder à página de um ponto de interesse em específico, onde conseguem visualizar a sua imagem (caso a possua), nome, descrição, localização.
- **Informações de Perfil:** Todos os utilizadores têm acesso à informação do seu perfil, contendo atalhos de acesso aos seus favoritos e executar *log out*.
- **Favoritos:** Todos os utilizadores conseguem aceder e gerir os seus *trails* favoritos.
- **Contactos de Emergência:** Todos os utilizadores têm acesso a uma página com todos os contactos de emergência da aplicação.

4.2 Funcionalidades Utilizador *Premium*

Para os utilizadores *premium*, e lembrando que estes são uma extensão dos utilizadores *standard*, a aplicação oferece as seguintes funcionalidades:

- **Informação de um *Trail*:** Para além de todas as informações disponíveis referidas acima, a aplicação adiciona a apresentação interativa de todo o percurso do *trial*, assim como a possibilidade de iniciar e parar esse mesmo percurso, obtendo estatísticas sobre o mesmo.
- **Informação de um Ponto de Interesse:** Para além das referidas acima, para um utilizador *premium* é ainda adicionada a funcionalidade de visualização de *media* do ponto de interesse e possibilidade de marcar o mesmo como visitado.

- **Download Media:** Na página de *media* de um ponto de interesse, os utilizadores *premium* têm a possibilidade de fazer *downlaod* da mesma, permitindo uma melhor experiência em uso *offline*.
- **Histórico:** Os utilizadores *premium* têm acesso ao seu histórico de *trails* (sejam estes completados com sucesso ou não) e de pontos de interesse visitados.
- **Definições:** Os utilizadores *premium* conseguem, também, aceder às definições da aplicação de modo a definir certos parâmetros e permissões como: intervalo de tempo para a verificação da distância do utilizador aos vários pontos de interesse um *trail*, distância à qual o utilizador quer ser notificado da sua proximidade aos vários pontos de interesse e, por último, a permissão de envio de notificações.

5 Testes

Nesta secção vamos apresentar todos os testes desenvolvidos para o projeto, dividindo os mesmos nos seus derivados e propósitos. No entanto, antes de iniciarmos este processo, decidimos aplicar uma ferramenta de **análise estática** para verificar todo o código da aplicação: **Lint**. Assim, foi-nos retornado um relatório contendo todos os potenciais *smells* e problemas presentes no código de forma a auxiliar a simplificação ou eliminação destes. Depois desta primeira etapa concluída, passamos para a fase de teste propriamente dita que inclui Monkey Testing e UI e Unit Testing.

5.1 Monkey Testing

Monkey Testing trata-se da criação randomizada de testes que envolvem interações sintéticas com a aplicação. Assim, foram feitos algumas iterações de testes de modo a principalmente afirmar que não existiam interações que devolvessem erros ANR. Exemplo de um dos testes feitos foi o seguinte:

```
adb shell monkey -p com.example.fase1 -s 123456
    --throttle 100 1000
    --pct-touch 35
    --pct-motion 30
    --pct-nav 30
    --pct-syskeys 5
```

5.2 Unit and UI Testing

De modo a permitir a testagem correta da aplicação no ambiente em que ela se insere, foi necessário tomar algumas limitações nos testes, isto é, devido à eliminação de dados do emulador na gravação de testes espresso implicava que todos os testes eram iniciados com o *login*. No entanto, num ambiente normal da aplicação, após feito o *login*, este não será mais necessário até feito um *logout*, logo, antes de cada teste deverá ser feito o *login* de algum dos *users* e só depois correr os testes correspondentes a cada tipo de utilizador.

Em termos da diversidade de testes, existem testes de navegação entre atividades e testes de comportamento. Assim sendo, existem os seguintes testes:

- **LoginActivityPremiumTest e LoginActivityStandardTest:** Estes testes foram desenvolvidos de forma a testar o correto Login por parte de ambos os tipos de utilizador. Assim, após inicio de sessão, verificam o nome do utilizador na página de perfil.
- **MainTest:** Este teste verifica o correto redirecionamento para a página do Ponto de Interesse ao carregar na imagem do ponto do dia em destaque na página inicial da aplicação.
- **TrailsTest:** Neste ficheiro existe apenas um teste a ser feito, que é a navegação da atividade de *trails* para um único *trail*.
- **TestStartStop:** Este ficheiro é relativo ao teste de Iniciar e Parar um roteiro.

- **PinTest:** Este teste serve para verificar o correto redirecionamento para a página de um ponto de interesse tendo em conta o tipo de utilizador autenticado. Assim, para o caso do utilizador *premium* é também verificada a visibilidade dos botões de acesso à *media* do mesmo e de adicionar ao histórico.

- **MediaTest:** Este teste, apenas disponível para o utilizador *premium*, verifica o correto redirecionamento para a página de *media* de um determinado ponto de interesse.

- **ProfileTest:** Este teste, utilizado para ambos os tipos de utilizador, verifica o correto redirecionamento para a página de informações de perfil de cada utilizador através da seleção na barra de navegação. No caso do utilizador *standard* é verificada a visibilidade do botão de Favoritos e *log out*. Já no caso do utilizador *premium*, é verificada a visibilidade dos botões de histórico e acesso a definições.

- **ProfileBtnSettingsTest:** Este teste é apenas aplicado no caso do utilizador *premium* e serve para verificar o correto redirecionamento a partir da página de perfil do utilizador para a página de *settings*.

- **HistoryTest:** Neste ficheiro existe apenas um teste a ser feito, que é a navegação da atividade de histórico para um único trail.

- **AddFavoriteTest**: Este ficheiro adiciona o primeiro *trail* aos favoritos ao premir o *icon* de estrela na atividade do mesmo. Depois, verifica se este foi adicionado aos favoritos.

- **FavoritesTest:** Neste ficheiro também só existe um teste a ser feito, que é a navegação da atividade dos favoritos para um único *trail*.

- **LogOutTest:** Este teste verifica a correta desautenticação de ambos os tipos de utilizador, refazendo o processo de autenticação no final de forma a facilitar o processo de testagem dos restantes, uma vez que a maior parte assume a devida autenticação.

- **ContactsTest:** Neste ficheiro só existe um teste para a navegação necessária da atividade de contactos. Para a criação deste teste, não foi utilizado o espresso recorder devido à verificação de troca de ecrã para o Dial do telemóvel. No entanto, é verificado então que na seleção do ícon de um contacto, este vai ser redirecionado para o Dial com o número correspondente.

- **ChangeSettingsTest:** Este teste para funcionalidade do utilizador *premium*, altera várias definições na atividade de definições, saindo e voltando a entrar na atividade de forma a verificar que os valores foram corretamente alterados.

6 Discussão de Resultados

6.1 Trabalho Realizado

Finalizado assim o desenvolvimento do projeto, verificamos que todas as funcionalidades requisitadas foram realizadas com sucesso, juntamente com uma estrutura do projeto que seguia o padrão recomendado. Em termos de requisitos não funcionais, estes também foram cumpridos aos olhos da equipa, visto que existe subjetividade em "experiência de utilizador intuitiva". Assim, encontramo-nos satisfeitos com o produto resultante mas também ansiosos para a seguinte fase.

6.2 Limitações

Em termos de limitações deste produto pode-se dizer que existem algumas previstas, principalmente o facto de que a aplicação não funcionará completamente em modo *offline*, visto que algumas das *features* implementadas necessitam da conexão à Internet e outras que não funcionarão *offline* visto que não foram requisitados, exemplo disto seria a apresentação dos ficheiros multimédia de um ponto sem ser feito o respetivo *download*.

A nível do serviço implementado, os pedidos de localização são feitos no *timing* indicado pelo utilizador, no entanto, devido a limitações do *Android*, para que isso fosse possível, foi necessário estabelecer o serviço como um serviço *Foreground*. Isto foi necessário porque recentemente o *Android* definiu limites para o número de pedidos de localização num dado intervalo de tempo por parte de serviços *Background* por razões de poupança de bateria. Assim, ao contornar isso, a aplicação pode ser dispendiosa a nível de bateria. No entanto, uma vez que a localização será (teoricamente) usada pela aplicação do *Google Maps* em simultâneo durante a realização de um roteiro, esse gasto de bateria já seria feito de qualquer das formas. Outra limitação que o serviço pode ter está no cálculo da distância percorrida pelo utilizador durante um roteiro: uma vez que o cálculo da distância é resultante do somatório das distâncias entre cada uma das medições feitas, quanto maior o intervalo definido pelo utilizador para cada medição, menos exato o valor resultante.

6.3 Funcionalidades Extra

Ao longo do desenvolvimento do projeto, para além das funcionalidades obrigatórias, o grupo decidiu inovar alguns aspetos do mesmo ao disponibilizar aos utilizadores funcionalidades extra. Deste modo, passamos a enumerar as mesmas:

- Apresentação de um **ponto de interesse do dia** (destaque) na página inicial
- Ficheiros mídia do tipo **áudio** com controlo de *playback*
- Redirecionamento para a **localização** de um determinado **ponto de interesse** através do *Google Maps* a partir da página do mesmo
- Possibilidade de adicionar *trails* aos **Favoritos**, sendo depois acedidos através da página de perfil
- Apresentação do **estado** (iniciado/cancelado/finalizado) de cada *trail* na página de Histórico.

7 Gestão do Projeto

Nesta secção iremos apresentar a metodologia seguida para a gestão e distribuição de tarefas ao longo de todo o desenvolvimento do trabalho de modo a otimizar o mesmo, assim como uma pequena avaliação ou reflexão coletiva de todos os elementos do grupo sobre as suas performances.

7.1 Gestão e Distribuição de Trabalho

A gestão e distribuição do trabalho realizado foi bastante uniforme ao longo do desenvolvimento, que envolveu igualmente todos os elementos do grupo através de métodos de distribuição de tarefas e responsabilidades a cada elemento.

Assim, o projeto iniciou-se pela **leitura** e análise extensiva do enunciado, seguida da criação de **mockups** que permitiriam à equipa obter uma ideia melhor sobre o funcionamento da aplicação, como também o *design* que gostaríamos que fosse aplicado. Este processo foi feito através da distribuição de requisitos pelos elementos, cada um criando a(s) *mockup*(s) que achassem necessárias para cumprir os requisitos.

Seguidamente foi feita uma **análise** de todas as *mockups* construídas, juntamente com correções que a equipa achou necessárias. Finalmente, todas as *mockups* foram **distribuídas** igualmente pelos elementos. Após essa distribuição, todos os elementos conseguiram facilmente comunicar entre si graças à separação de responsabilidades, o que permitiu o desenvolvimento de diferentes atividades com uma transição suave.

7.2 Metodologias de Controlo de Versão Utilizadas

Ao longo do semestre, evoluímos o nosso conhecimento acerca de ferramentas de controlo de versões e possíveis metodologias a seguir nas mesmas de forma a auxiliar o processo de desenvolvimento do projeto. Estas metodologias e ferramentas têm como principal objetivo agilizar o processo de desenvolvimento paralelo de *software* ao tentar evitar possíveis conflitos de *deployment* ou de sincronização no caso de um projeto com uma equipa de grande escala. No entanto, este projeto era deveras de uma dimensão mais reduzida e daí não surgiu a necessidade de utilizar métodos como *Trunk-based Development*. Assim, cada contribuidor dava *commit* às suas *features* e correções de *bugs* diretamente no ramo principal do projeto. No entanto, ocorreram situações em que seria necessário um distanciamento do *branch* principal devido a erros de maior escala, e daí foram criados *branches* que permitiam um maior isolamento.

7.3 GitHub Actions

Para além das metodologias acima referidas, desenvolvemos, também, conhecimento sobre *GitHub Actions* - uma plataforma de integração e entrega contínuas (CI/CD) que permite automatizar o pipeline de compilação, teste e *deployment*. Desta forma, e com a ajuda dos exercícios das aulas práticas, implementamos três *workflows* no repositório, passando a enumerar cada um:

- **createRelease.yml:** Este *workflow* trata a geração do ficheiro APK resultante da compilação do código-fonte do projeto e seu respetivo *upload* utilizando para tal *actions* pré-definidas, sendo, por isso, apenas preciso passar os parâmetros necessários de cada uma. Assim, o *workflow* é iniciado sempre que é executado um *push* para a *branch main*.

- **downloadArtifact.yml:** Este *workflow* trata o *download* do ficheiro APK previamente criado e *uploaded* no *workflow* anterior. Desta forma, este *workflow* é iniciado sempre que o *createRelease.yml* terminar (com sucesso), utilizando também *actions* pré-definidas para o efeito.

- **uploadRelease.yml:** Por último, este *workflow* trata da criação de uma versão de *release* na plataforma *GitHub*, utilizando o ficheiro APK anteriormente descarregado, atribuindo uma *tag* de *release* apropriada, sendo, por isso, iniciado quando o *workflow downloadArtifact.yml* termina. No entanto, apesar de ser esta a sua intenção, a versão das *tags* não é incrementada e o ficheiro APK não é corretamente adicionado, sendo apenas feito o *release* do código-fonte. Assim, apesar das várias tentativas de correção do grupo, este *workflow* encontra-se **incompleto**.

7.4 Reflexão sobre Performance Individual

Ao longo do desenvolvimento do projeto todos os elementos tiveram uma participação ativa tanto na idealização como no desenvolvimento, permitindo um rápido desenvolvimento sem problemas de comunicação. Assim, todos os elementos estariam com uma avaliação igual em relação à sua performance.

8 Conclusão

Após a realização desta primeira parte do projeto, o grupo encontra-se satisfeito com o trabalho desenvolvido, tendo sido alcançados todos os objetivos estabelecidos quer pelo docente, como pelo próprio grupo. No entanto, algumas abordagens a certos aspetos poderão vir a ser alteradas na fase seguinte.

Dado que nenhum membro do grupo tinha experiência prévia em desenvolvimento *mobile*, foi um processo de desenvolvimento com as suas dificuldades devido à novidade que era trabalhar neste ambiente. Consideramos que isso, tal como era de esperar, atrasou partes do desenvolvimento. No entanto, essas dificuldades foram ultrapassadas ou contornadas, sendo entregue um produto final completo.

Em suma, aprofundamos o conhecimento lecionado nas aulas da unidade curricular em relação à linguagem de programação *Java*, tecnologias nativas *Android* e, por fim, do ambiente de desenvolvimento *Android Studio*.