Universidade do Minho
Departamento de Informática

Tópicos de Desenvolvimento de Software

Desenvolvimento Android
Aplicação BraGuia

Grupo:
José Carvalho - PG53975
José Barbosa - PG52689
Miguel Silva - PG54097

7 de maio de 2024

Conteúdo

1 Introdução

Neste documento relatamos as decisões por trás do processo de implementação da primeira fase da aplicação BraGuia. Esta aplicação foi desenvolvida para a unidade curricular de Tópicos de Desenvolvimento de Software do primeiro ano do Mestrado em Engenharia Informática da Universidade do Minho.

Os objetivos do projeto encontram-se em detalhe no enunciado, mas resumidamente, temos de implementar uma aplicação *Android* que permite aos seus utilizadores iniciar trilhos por toda a cidade de Braga.

O relatório está dividido em diversas secções, nestas abordamos pormenores da implementação, como a estrutura, bibliotecas e padrões de software utilizados. Apresentamos também um mapa de navegação, uma lista de funcionalidades e uma discussão de resultados. Por fim, discutimos a gestão do projeto e terminamos com uma conclusão e análise crítica ao trabalho desenvolvido nesta fase.

2 Detalhes de implementação

Durante o desenvolvimento desta fase do projeto tivemos de tomar várias decisões a nível da implementação. Nesta secção explicamos as nossas opções, mas antes disso é importante relatar que muitas destas decisões são limitadas pela estrutura de *backend* existente.

2.1 Estrutura do projeto

Antes de abordar a estrutura em si do projeto, é importante salientar que tentamos seguir a estrutura base que a equipa docente forneceu.

Na pasta *com.ruirua.sampleguideapp*, o projeto contém 6 diretorias base, sendo elas a *model*, a *notifications*, a *repositories*, a *sensors*, a *ui* e por último a *viewModel*. Para além disso, a diretoria base contém dois ficheiros *Java*. Um deles representa uma instância da nossa aplicação (*MyApplication.java*), em que a sua utilidade é só para inicializar certas classes no nosso programa. O outro é relativo a gestão de permissões (*Permissions.java*).

A pasta *model*, contém todas as nossas classes base para o projeto. Esta diretoria está dividia em sub-diretorias, sendo cada uma responsável por uma entidade. Por exemplo, a pasta *pin*, contém a própria classe *Pin*, todas as classes associadas à mesma (*Media* e *RelPins*) e também contém duas interfaces, uma associada ao acesso à base de dados e outro associada às chamadas à *Rest API* disponibilizada pela equipa docente. As restantes pastas seguem a mesma metodologia desta, destacando apenas que a pasta *general* refere-se ao conteúdo genérico da aplicação, isto é, informações da mesma como nome, descrição, parceiros, etc.

A diretoria *notifications*, tal como o nome sugere, contém uma classe que contém métodos estáticos para a geração de notificações.

Em relação à *repositories*, esta diretoria apenas contém classes que são responsáveis por gerir o acesso a dados. Estes dados tanto podem ser obtidos através do próprio dispositivo ou através de chamadas à *API*. Dentro desta pasta, existe uma diretoria denominada de *utils* que contém duas classes, uma que contém métodos estáticos que são genéricos e usados por múltiplos repositórios, e uma outra classe que é útil a simplificar o código quando fazemos pedidos à *Rest API*.

Tal como a *notifications*, a pasta *sensors* contém dois ficheiros, relativos ao sensor de localização, que é útil para o envio de notificações.

A diretoria *ui* contém todas as classes relativas a atividades e fragmentos da nossa aplicação. Tal como a pasta *model*, esta diretoria está sub dividida em sub diretorias, que têm nomes sugestivos. A pasta *utils*, apenas contém algumas classes com código reutilizado por múltiplos fragmentos e atividades.

Por último a pasta *viewModel*, apenas contém dois ficheiros java, que são úteis para a visualização dos dados dos Pins e dos Trilhos.

No total, ficamos com 19 diretorias e 63 ficheiros.

Na figura 1, temos 4 imagens que ilustram a nossa estrutura, sendo as caixas brancas ficheiros, a vermelho as diretorias principais e a verde as sub diretorias.

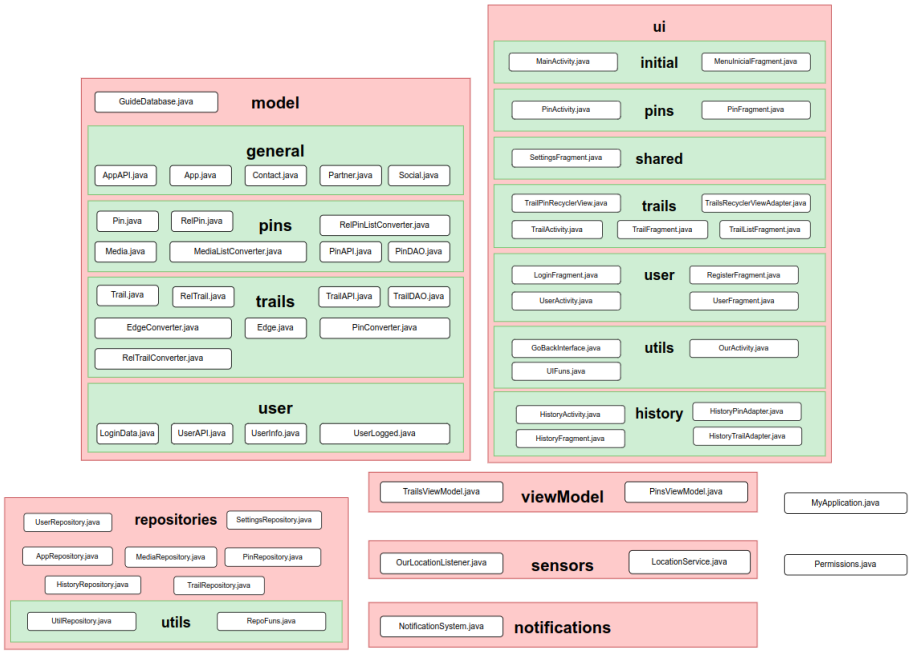


Figura 1: Estrutura do Projeto

2.2 Soluções de implementação

Neste projeto, como vamos falar mais abaixo nas funcionalidades, conseguimos implementar todos os requisitos pedidos. Nesta secção vamos falar daquilo que fizemos para realizar os objetivos propostos.

Antes de falarmos dos ecrãs em concreto, vamos abordar algo que criamos para facilitar bastante todo o nosso trabalho quer a nível de organização quer ao nível de aplicar funcionalidades a todas as atividades sem termos de repetir muito código. Dentro da pasta *utils*, existem 2 classes (e uma interface) que iremos abordar agora.

A primeira destas é a classe *OurActivity*, que acaba por ser uma superclasse para todas as restantes atividades do nosso programa. O que esta classe implementa são funcionalidades que nós tencionamos que estejam presentes em todos os pontos dentro da nossa aplicação. Por exemplo, a configuração do tema é feita aqui.

Também é garantido que outras ações da aplicação como envio de notificações e outras funcionalidades que iremos referir mais à frente ao falar de outras partes da aplicação.

Esta classe implementa também a interface *GoBackInterface*, sendo que o seu objetivo é a implementação de um botão de regresso universal. Este está presente na *template* de todas as páginas, e sempre que é pressionado a aplicação retorna ao fragmento ou atividade anterior. Também nesta *template*, mas do lado oposto, está implementado outro botão que permite ao utilizador ligar a um número de emergência.

No canto superior direito, colocamos outro botão que dá acesso às definições da aplicação. O menu das definições é acessível através de qualquer menu da aplicação, quer esteja o utilizador autenticado ou não. Nesta página, podemos mudar a aplicação para modo claro ou escuro, ligar e desligar a localização, definir se se pretende ou não receber notificações da aplicação, podemos também definir o intervalo de tempo que se espera para ir buscar informações sobre a localização atual e a precisão da mesma localização. A implementação deste menu é realizada no *SettingsFragment*.

Entremos agora na explicação da classe *UIFuns*. Esta classe é muito importante, pois é nela que estão definidas imensas funções que construimos, que depois são chamadas em diversos pontos da aplicação. Por exemplo, é aqui que estão definidas as funções necessárias para realizar a mudança de fragmento ou de atividade de forma segura, sem causar erros nas transições que ocorrem ao longo da utilização da nossa solução. Também é aqui que estão definidas, funções e métodos bastante úteis no que toca à integração com o *GoogleMaps*, à demonstração de multimédia, seja vídeo, áudio ou imagem e define também a chamada ao número de emergência referido no parágrafo anterior.

No realizar deste trabalho, começamos pela criação da pagina inicial. Na *MainActivity* realizamos as alterações necessárias para que se começasse no ecrã inicial, encaminhando no *onCreate* para o *MenuInicialFragment*.

No panorama da autenticação, existem dois botões, que reencaminham para duas páginas, uma de registar e uma de *login*. Ao selecionar um destes botões

mudamos para a *UserActivity*, que reencaminhará para o fragmento desejado. O de registar é meramente estético, uma vez que devido às condições estipuladas por este trabalho não conseguimos adicionar nenhum utilizador novo à base de dados. Contudo consideramos que adicionar o menu torna a aplicação mais completa.

No que toca ao ato de*login*, o processo de implementação é realizado no *LoginFragment* (com ajuda do *UserRepository*). Quando o utlizador clica no botão de login a aplicação manda o pedido ao *backend* que retorna os *tokens* associados ao utilizador. Assim, conseguimos autenticar tanto utilizadores *Standard* e *Premium*.

A autenticação sofreu algumas melhorias, após a implementação inicial. Implementamos posteriormente, a capacidade de um utilizador fazer *logout* caso assim o pretenda. Conseguimos também manter o estado da sessão do utilizador, ou seja se um utilizador tiver realizado o *login*, o seu estado é guardado e se fechar a aplicação sem realizar o *logout* ao abrir a aplicação de novo, o utilizador continua com a autenticação feita sem ter de colocar as suas credenciais de novo.

O ponto principal da aplicação é a navegação com o auxilio de trilhos. No total existem pré-definidos do *backend* trilhos, com os seus respetivos pins, que contém os pontos de interesse que existe em cada um dos mesmos. Esta parte a nível de implementação envolve imensas classes pelo que vamos explicar o que cada uma delas faz de forma concisa.

A classe *TrailRepository* pertence à diretoria *repositories* cujas funcionalidades já foram explicadas. Esta, ajuda-nos assim a manter uma constante atualização dinâmica e em tempo real das informações conforme o trilho que o utilizador está a realizar, muitas vezes com ajudas visuais.

Para a visualização dos trilhos, é utilizado o *TrailListFragment*, que apresenta uma lista dos trilhos disponíveis. Cada item da lista exibe informações básicas sobre o trilho que podem ser úteis para o utilizador, como nome e dificuldade, permitindo que este selecione um trilho específico para visualizar detalhes adicionais e de seguida decidir se o tenciona realizar ou não. A classe *TrailsRecyclerViewAdapter* é responsável pela interface onde aparecem algumas informações destes itens.

Ao visualizar um trilho específico, com a ajuda da classe *TrailFragment*, os utilizadores têm acesso a informações mais detalhadas sobre o trilho. Além disso, são apresentados os pontos de interesse associados a este roteiro, permitindo que os utilizadores os explorem ao longo do percurso. A classe *TrailPinRecyclerView* é responsável por exibir os *pins* numa lista, facilitando a navegação e visualização dos pontos de interesse, funciona de forma análoga à classe *TrailsRecyclerViewAdapter*.

Em relação aos pontos de interesse, estes contêm atributos como, nome, descrição e até coordenadas. Para além disto, estes podem estar associados a outros pins, o que cria uma espécie de rede de pontos conectados, que podem ser observados sequencialmente.

A integração propriamente dita, é gerida pela classe *PinRepository*, que tal como o seu equivalente para os trilhos que já foi mencionado, atua como uma

ponte entre a aplicação e o *backend* proporcionado.

Com a ajuda do *PinFragment*, conseguimos implementar estes detalhes que referimos e também a multimédia associadas a cada um deles, seja vídeos, imagens ou áudios, que o *backend* nos proporciona.Convém frisar que nem todos os pins possuem os três tipos, havendo alguns sem qualquer conteúdo multimédia. Este facto é algo que também foi tido em consideração e que está implementado de forma a que o utilizador consiga facilmente perceber que tipos de multimédia possui cada um dos pontos de interesse.

Resta nos falar da implementação do histórico da aplicação, ou seja da possibilidade do utilizador conseguir aceder à lista de trilhos e de pontos de interesse que já iniciou ou visitou. O modo como isto foi implementado segue uma lógica relativamente simples em que quando um utilizador começa um trilho esse é adicionado a um *Set* que depois serve como histórico, a mesma lógica é aplicada a pontos de interesse. O modo como estes são visualizados é num próprio menu à parte, cuja ideia base é feita no *HistoryFragment*, sendo que a lógica pela qual se apresenta as listas com os roteiros e pontos de interessa já visualizados estão implementados nas classes *HistoryTrailAdapter* e *HistoryPinAdapter*, respetivamente, estas são semelhantes às classes *TrailsRecyclerViewAdapter* e *TrailPinRecyclerView*.

2.3 Bibliotecas/dependências utilizadas

Em relação às bibliotecas utilizadas, o grupo não fugiu muito para além daquelas que já estavam a ser utilizadas na base do projeto. Sendo assim utilizamos as seguintes:

- ***Retrofit***: Utilizada para efetuar pedidos *HTTPS* para uma *Rest API*.
- ***room***: Biblioteca nativa android para efetuar operações sobre Base de Dados.
- ***Gson***: Auxiliar na serialização e deserialização de objetos no formato *json*.

2.4 Padrões de software utilizados

Em relação a padrões de software utilizados, chegamos a usar padrões de todos os tipos, isto é, padrões de criação (*Singleton*), padrões de estrutura (*Adapter*) e padrões de comportamento (*Memento*, *Template Method*).

O ***Singleton***, tal como o próprio nome indica, é usado quando queremos ter apenas uma instância de uma certa entidade na nossa aplicação. Tendo isso em conta algumas das classes presentes na pasta *repositories* seguem este padrão, nomeadamente *UserRepository*, *MediaRepository*, *SettingsRepository*, *HistoryRepository* e *AppRepository*.

No caso do *UserRepository* utilizamos este padrão de forma a centralizar o estado da sessão, não correndo o risco de termos múltiplos estados de sessões. Já a classe *MediaRepository*, mantém o estado atual de qual o conteúdo multimédia

está guardado no sistema, não fazendo sentido também termos múltiplos estados. Na classe *SettingsRepository*, o padrão faz sentido pois assim garantimos que só há uma combinação para as definições em cada instante na aplicação. A *HistoryRepository* contém este padrão sobretudo de forma a centralizar o processo de guardar e ler o histórico de pontos / trilhos visitados pelo utilizador. Por último, o *AppRepository* segue o padrão *Singleton*, pois assim conseguimos guardar apenas num só sitio os dados genéricos da aplicação, em vez de estar a fazer diversos pedidos ao servidor de *backend*.

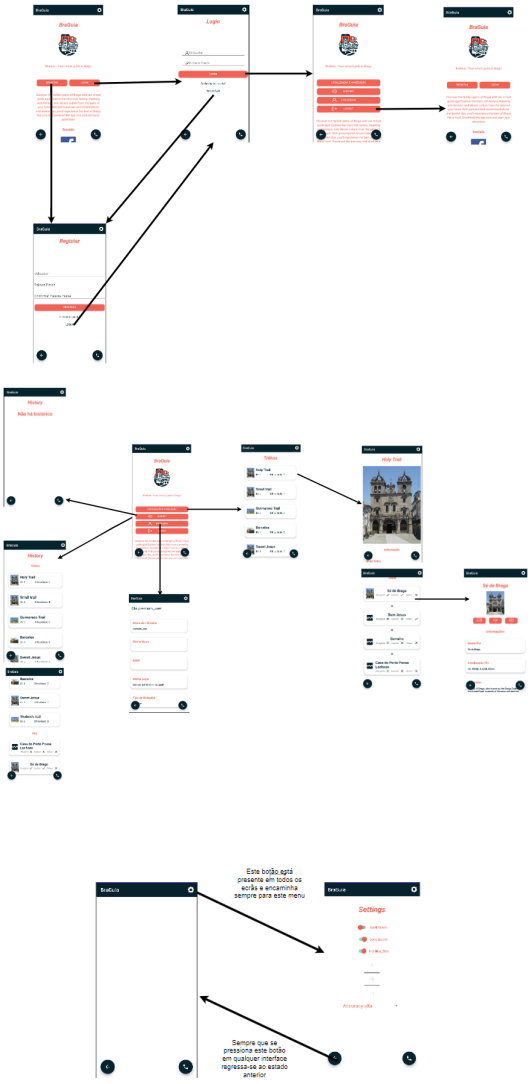
Outro padrão utilizado foi o ***Memento***. Este padrão é útil quando temos uma classe / entidade que precisa de guardar o seu estado em disco, de forma a não perder o tal estado quando reiniciamos a aplicação. Este padrão foi usado nas classes *UserRepository*, *SettingsRepository* e *HistoryRepository*.

A classe *UserRepository* precisa deste padrão, pois assim damos a possibilidade de quando reiniciamos a aplicação, podermos restaurar o último estado da sessão. No caso da classe *SettingsRepository*, faz sentido usar este padrão de forma a guardar e restaurar as últimas definições que o utilizador definiu. Por último, a classe *HistoryRepository* utiliza este padrão de forma a guardar o histórico do utilizador.

Um outro padrão que utilizamos foi o ***Template Method***. Este padrão deve ser usado quando queremos definir certas funcionalidades de um certo algoritmo que é partilhado por diversas classes, numa super classe. No nosso caso, não foi nenhum algoritmo, mas sim as atividades. A classe *OurActivity* é uma super classe de todas as nossas atividades e nela fazemos a renderização da *template* base de todas as páginas, bem como também definimos certos comportamentos por defeito, como o retroceder nas páginas.

Estes foram os padrões que utilizamos na nossa aplicação, mas é importante destacar que o padrão ***Adapter*** é utilizado quando recorremos às bibliotecas *Retrofit* e *Gson*, pois ambas estas classes tratam da parte de fazer pedido à API e converter a resposta em objetos *Java*.

3 Mapa de navegação de GUI



4 Funcionalidades

As funcionalidades fornecidas pela nossa implementação BraGuia surge em resposta aos requisitos presentes no enunciado. Ao longo deste projeto conseguimos implementá-los todos. Obtemos então o seguinte:

- Um menu inicial através do qual se consegue aceder à grande maioria das funcionalidades e que contém informações sobre a aplicação como os seus parceiros e redes sociais.
- A aplicação mostra num ecrã, de forma responsiva, uma lista dos roteiro disponíveis.
- Capacidade de um utilizador se autenticar na aplicação.
- Foram implementados dois tipos de utilizadores diferentes: *standard premium*.
- A aplicação, numa única pagina, mostra a informação existente para cada trilho, desde uma galeria de imagens, a descrição, um mapa do itinerário com pontos de interesse e informações sobre a mídia disponível para os seus pontos.
- Todos os utilizadores, após efetuarem o Login ganham acesso a uma pagina onde podem visualizar informações sobre a própria conta.
- Os utilizadores *premium*, tal como pretendido, têm acesso a mais funcionalidades como a capacidade de navegação.
- A aplicação assume que o *Google Maps* está instalado e notifica o utilizador que este é necessário de modo à navegação poder ser feita de forma visual e com auxílio de voz, desta forma pode ser ser utilizada por condutores.
- Existe um menu de definições, que pode ser acedido em qualquer sitio da aplicação.
- Dentro da aplicação é possível um utilizador ligar, desligar e configurar os serviços de localização.
- É possível iniciar e interromper um trilho na aplicação.
- A aplicação guarda localmente um histórico dos trilhos e dos pontos de interesse visitados pelo utilizador.
- Quando um utilizador passa por um ponto de interesse a aplicação emite uma notificação sobre o dito ponto.
- Se um utilizador decidir aceder à notificação, que foi referida no tópico anterior, será reencaminhado para o ecrã principal do ponto.
- Cada um dos pontos de interesse possui um ecrã descritivo sobre o mesmo, que contém várias informações sobre ele, como localização, mídia, descrição e propriedades.
- A aplicação consegue apresentar e produzir 3 tipos de mídia: voz, imagem e video.
- A aplicação possui a capacidade de descarregar mídia do backend e alojala localmente, de modo a poder ser usada em contextos de conectividade reduzida.
- É possível efetuar chamadas para contactos de emergência da aplicação através de um elemento que é acessível em qualquer ponto da aplicação.

5 Discussão de resultados

Ao longo deste projeto, conseguimos cumprir todos requisitos estabelecidos inicialmente, com algumas funcionalidades extra também conseguidas. Nesse aspeto

como grupo estamos contentes com o progresso obtido, se bem que houveram certos problemas e limitações que encontramos e que iremos abordar também.

5.1 Trabalho realizado

No geral entendemos que o nosso trabalho correu de forma adequada, pois conseguimos implementar todos os objetivos levantados pelos requisitos, assim como listamos na secção anterior. Para além disso implementamos também modo escuro com cores alternativas.

Mesmo assim, apesar de estarmos satisfeitos com o nosso desempenho, achamos que podíamos ter feito melhor, nomeadamente em aspetos como estrutura e legibilidade do código, pois acabamos por utilizar certos excertos de código que o IDE assinala como *deprecated* e em retrospetiva consideramos que a nossa estrutura, embora não errada podia ser mais clara.

Outra questão na qual gostávamos de ter tido mais controlo foi na questão artística. O que queremos dizer com isto é que devido a não termos experiência com design gráfico, precisamos de recorrer a ferramentas como o *Copilot* da *Microsoft* para a geração do logótipo da aplicação e ao banco de imagens Veryicon para obter imagens *SVG* para situações que o *Android Studio* não consiga cobrir com o gerador de recursos vetoriais.

Por fim, gostaríamos de referenciar que alguns extras que gostávamos de ter incorporado seriam:

- Uma *navbar* no fundo do ecrã que facilitasse a navegação entre os fragmentos.
- Um sistema que permitisse os utilizadores marcarem certos trilhos como favoritos.
- Mais informação disponível na consulta do histórico, listando dados como: tempo do trilho, data de inicio e de fim, número de vezes realizado e possivelmente análise desses mesmos dados para estudar a evolução do utilizador.
- *Google Maps* dentro da aplicação e não apenas externamente, não o conseguimos fazer pois requeria a *API* paga da *Google*.
- Sistema que permitisse utilizadores avaliarem com uma nota um trajeto ou um ponto de interesse, ou até mesmo deixar comentários.
- Multimédia apresentada sobre a forma de uma galeria navegável, mais intuitiva do que a que incorporamos.

Acabamos por não implementar estes extras porque nos focamos nos requisitos bases do projeto.

5.2 Limitações

Embora tenhamos cumprido os requisitos, a nossa aplicação apresenta algumas limitações que foram identificadas por nós ao longo da realização do projeto, ainda que tenhamos encontrado soluções para as mesmas.

Após uma consulta sobre certos requisitos base de qualquer aplicação *Android*, o grupo optou por implementar tanto a funcionalidade de dar a opção do utilizador escolher entre modo claro e modo escuro, como a funcionalidade de retroceder nas páginas. As nossas limitações estão associadas a estas duas funcionalidades.

Relativamente ao modo claro / escuro, o principal problema que encontramos foi que as alterações não eram logo visíveis na template base das páginas (cabeçalho da página + dois botões em baixo). A solução para este problema foi meter as mesmas cores tanto em modo escuro como em modo claro.

Em relação à opção de retroceder, o problema surge quando o grupo optou por criar diversas atividades na nossa aplicação, seguindo a metodologia de, páginas com contextos diferentes, resulta em atividades diferentes. O nosso problema surge quando queremos retroceder de uma atividade para uma anterior em certos casos específicos. Um exemplo disso é quando estamos na atividade *trails*, mais especificamente na página de visualização de um trilho, e queremos visualizar um ponto de interesse do mesmo. Inicialmente, criávamos a atividade associado ao *pin*, passando os respetivos argumentos. Quando estamos na página deste *pin* e queremos retroceder, com base na nossa implementação, a aplicação vai ver que já não havia mais fragmentos associados à nova atividade, finalizando a atividade atual e reiniciando a anterior, ou seja, reiniciando a atividade *trails*. Só que inicialmente a atividade *trails* está associada ao fragmento que ilustra todos os trilhos, em vez de ir para o trilho especifico que o utilizador já esteve. A solução para este problema foi, em vez de criar uma atividade *pin*, simplesmente mudamos o fragmento em si e a aplicação comporta-se tal como era suposto.

5.3 Funcionalidades extra

Embora o foco do grupo tenha sido bastante orientado para os requisitos funcionais presentes no enunciado, o grupo conseguiu implementar funcionalidades extra, mais relacionados com certos requisitos não funcionais.

Essas funcionalidades focam-se nas recomendações de aplicações *Android*, que podem ser consultados em: Princípios de qualidade. Os princípios que o grupo implementou foram:

- Possibilidade de alternar entre modo escuro e claro.
- Ter a possibilidade de retroceder em qualquer ponto da página.
- A preservação do estado do utilizador, por exemplo, se o utilizador estiver autenticado e reiniciar a aplicação, ele continua autenticado.
- As notificações que a aplicação lança para o utilizador, também seguem as diretrizes.

Para além disso, o grupo fez uma página de registo de utilizador, que ao momento, não faz nada, pois para isso seria necessário adaptar o servidor de *backend* para adicionar novos utilizadores.

6 Gestão de projeto

6.1 Gestão e Distribuição de trabalho

Em relação à distribuição de trabalho acabamos os três por focar em partes distintas, sendo que íamos fornecendo ajuda uns aos outros quando alguém se encontrava com dificuldades.

O José Carvalho focou maioritariamente na camada da lógica de negócio, tendo feito grande maioria do código para essa parte. Ajudou também a desenvolver duas páginas no que toca à interface e corrigiu também vários erros e problemas que iam aparecendo aos seus colegas.

O José Barbosa focou maioritariamente nas camadas superiores do projeto, tendo implementado ou ajudado a implementar maioria das interfaces e do código dos níveis mais visíveis do projeto. Deu também sugestões e tentou ajudar e dar suporte aos seus colegas.

O Miguel Silva foi o principal responsável pela implementação do funcionamento dos trilhos e dos respetivos pins tendo também desenhado a respetiva interface e lógica de negócio. Deu também apoio e tentou sempre ajudar e dar sugestões aos colegas.

6.2 Eventuais metodologias de controlo de versão utilizadas

Para controlo de versão recorremos à plataforma *GitHub*. Aqui criamos um repositório no qual fomos colocando o código que fomos desenvolvido.

Acabamos por não utilizar *Branches* pois estivemos constantemente em contacto uns com os outros, marcando reuniões frequentemente. Mesmo assim, admitimos que o podíamos ter feito, pois reduzem o número de conflitos de atualizações, sendo que recorremos frequentemente a essas funcionalidades noutros projetos. Apenas não o fizemos neste projeto, pois já nos conhecemos bem e conseguimos conciliar o nosso trabalho sem termos tido conflitos.

6.3 Reflexão sobre Performance individual

José Carvalho

Eu, José Carvalho, penso que contribui bastante para o desenvolvimento do projeto. Acabei por abordar todas as áreas de desenvolvimento, tanto ao nível da interface como na camada da lógica de negócio. No desenvolvimento de interfaces, acabei por ser o elemento que menos contribuiu, pois acabei por desenvolver apenas duas páginas, mas tive sempre a dar a minha opinião ou sugestões para os meus colegas. Em compensação, acabei por fazer a grande maioria do código da camada da lógica de negócio, como já foi referido na secção 6.1.

José Barbosa

Eu, José Barbosa, penso que tive uma influência positiva no desenvolvimento do projeto, sendo que abordei maioritariamente as camadas superiores do mesmo. Acredito que no desenvolvimento das interfaces acho que fui quem tratou mais desse aspeto, tendo mexido, organizado e dado sugestões para o desenvolvimento da maioria destas. Por contraste, fui o elemento que menos ajudou na parte da camada de lógica de negócio.

Miguel Silva

Eu, Miguel Silva, considero que tive um bom contributo neste projeto. Fiquei responsável por definir a estética da interface final, a nível de cores e formulação das páginas, atuei bastante nas partes relacionadas com os trilhos, fazendo tanto a parte da lógica de negocio relativa a estes componentes como as interfaces onde estes aparecem. Tirando isso, tentei prestar suporte aos colegas sempre que consegui.

7 Conclusão

Na nossa opinião, esta etapa do trabalho prático foi um sucesso. Conseguimos implementar todas as funcionalidades, embora admitidamente não da forma mais estética.

Podemos dizer também que este projeto deu-nos conhecimentos práticos relevantes de desenvolvimento *Android*, que inclusive foram úteis em particular para dois elementos do grupo que também tiveram de recorrer a esta tecnologia para um projeto de outra unidade curricular.

Achamos também que todos os elementos contribuíram de forma relevante para o projeto e como tal consideramos que o grupo funcionou bem, dividindo de forma adequada as tarefas.

A nível de criticas negativas, podemos criticar a nossa estética da aplicação que podia melhorar com algo como uma *navbar* no fundo da página com atalhos para as restantes páginas da aplicação, ou ter o *Google Maps* implementado dentro da aplicação, tal como listamos na secção relativa ao trabalho realizado, onde também apresentamos outros extras que na nossa opinião tornariam o nosso projeto mais rico, mas assim como já afirmamos neste relatório, focamo-nos em implementar todos os requisitos base.

Aproveitamos também esta secção para referir que o servidor de *backend* esteve constantemente ir a baixo, devido ao limite da largura de banda disponível ser atingido. Apesar disso, na parte final do projeto não fomos muito afetados, pois tínhamos a maioria dos dados guardados localmente na base de dados, não necessitando de fazer pedidos ao servidor. Mesmo assim, gostaríamos de ter tido algo mais estável e menos volátil. Para além disso achamos que os dados estavam armazenados de forma confusa e repetitiva, pois por exemplo, a rota '*/trails*' apresenta os dados todos relativos tanto aos trilhos como aos

pins, sem ser necessário autenticação. Isto torna, outras rotas honestamente desnecessárias.