I-on Aplicação Android

Relatório versão final

Autores João Silva, nº44854
Diogo Santos, nº44774

Orientador Prof. Paulo Pereira

20 de Julho de 2020

Conteúdo

Lista de Figuras

Resumo

Existem alguns aspetos do quotidiano académico dos alunos do Instituto Superior de Engenharia de Lisboa que podem ser melhorados. São exemplos: a inexistência de um mapa de exames conciso e adequado a cada aluno (o existente incluí todos os exames de todos os cursos), e; como os horários estão compostos do ponto de vista da organização, o exercício de construção do horário é deixado a cargo de cada aluno (os horários disponibilizados estão organizados por turma e contêm todas as UCs de todos os semestres, incluindo as UCs opcionais).

A iniciativa i-on pretende resolver os problemas referidos anteriormente. A iniciativa é composta por uma família de projetos criada com o objetivo de implementar serviços de apoio à atividade académica.

O subsistema i-on Android pretende viabilizar o acesso às funcionalidades do sistema a partir de dispositivos Android. A aplicação Android resultante é um assistente do aluno concebido para o ajudar no quotidiano académico.

O resultado deste projeto é uma aplicação publicada na Google Play Store. A solução cumpre as recomendações arquiteturais oficiais, e foi desenhada tendo em mente a longevidade da base de código. Essa base de código tem manifestações da preocupação de racionalização do uso de recursos computacionais e da circunstância de ser uma aplicação a ser disponibilizada ao público.

Agradecimentos

Neste projeto, gostaria de agradecer ao meu orientador, Professor Paulo Pereira, pela sua paciência, disponibilidade e por toda a sua ajuda para a realização deste projeto. Queria também agradecer a todos os professores da iniciativa i-on por terem criado esta iniciativa, pelo qual estou bastante orgulhoso por feito parte, e por toda a sua disponibilidade para me ajudarem sempre que precisava de ajuda.

Queria também agradecer ao meu colega de grupo, João Silva, pelo seu convite, pelo seu companheirismo, trabalho em equipa e pela sua confiança na realização deste projeto.

Gostaria também de agradecer a todos os meus colegas do curso por todo o seu suporte, amizade e entre-ajuda nestes últimos 3 anos pelo qual fui aluno, com muito orgulho, do Instituto Superior de Engenharia de Lisboa. E um especial agradecimento à minha família por todo o seu apoio e por terem me dado condições para que pudesse realizar a minha licenciatura.

Julho 2020, Diogo Santos

Começo por agradecer ao nosso orientador, Professor Paulo Pereira, pela sua disponibilidade e apoio ao longo do desenvolvimento deste projeto. Sinto-me lisonjeado por ter consigo um orientador tão entusiasmado que colocou tanto tempo e esforço para nos ajudar no desenvolvimento deste projeto, sem ele este projeto teria sido impossível. Quero também agradecer aos restantes professores na iniciativa i-on que mesmo não sendo orientadores estiveram sempre disponíveis para ajudar.

Deixo o meu agradecimento ao meu colega de grupo, Diogo Santos, pelo companheirismo, e por ter confiado em mim ao longo do desenvolvimento deste projeto. Acho que fazemos uma excelente equipa.

Agradeço à minha namorada, Inês, por todo o apoio e motivação que me deu ao longo desta licenciatura, sem ela teria sido dez vezes mais difícil, e pela paciência que teve ao ajudar-nos na escrita deste relatório.

Concluo agradecendo à minha família por todo o apoio e por me darem a oportunidade de realizar a licenciatura no Instituto Superior de Engenharia de Lisboa.

Julho 2020, João Silva

Capítulo 1

Introdução

O presente capítulo introduz a iniciativa i-on, depois a motivação para este projeto, em seguida, as suas especificações e conclui com uma breve descrição dos restantes capítulos.

1.1 Iniciativa i-on

A iniciativa i-on é uma família de projetos criada com o objetivo de implementar serviços de apoio à atividade académica. A iniciativa é composta por três projetos: i-on Android, i-on Core e i-on Integration. Todos estes projetos compõem o sistema i-on que será detalhado na secção 2.1.

1.2 Motivação

Existem alguns aspetos da atividade académica no âmbito do Instituto Superior de Engenharia de Lisboa (ISEL) que podiam ser melhorados. Um exemplo é o facto de o mapa de exames ser demasiado extenso pois contém todos os exames de todos os cursos, enquanto que a maioria dos alunos só necessita de saber a data dos exames das Unidades Curriculares (UC) em que está inscrito.

Outro exemplo de uma melhoria ao quotidiano académico é que os horários estão compostos do ponto de vista da organização, ou seja, os horários disponibilizados estão organizados por turma e contêm todas as UCs de todos os semestres, incluindo as UCs opcionais. Por causa disto o exercício de construção do horário é deixado a cargo de cada aluno.

Estes são alguns exemplos de melhorias que se pretendem realizar com o sistema i-on. O subsistema i-on Android pretende viabilizar o acesso às funcionalidades do sistema a partir de dispositivos Android.

1.3 Objetivo

O principal objetivo deste projeto é desenvolver uma aplicação móvel que, em conjunto com o resto dos projetos da iniciativa i-on, resolva os problemas identificados anteriormente, proporcionando assim uma experiêncida de utilização que facilita o dia a dia dos elementos da comunidade académica do ISEL. A aplicação é desenvolvida usando a *framework* Android e a linguagem de programação Kotlin.

1.4 Especificações do projeto

Para que esta aplicação seja realmente capaz de facilitar a atividade académica identificaram-se os requisitos mínimos que se apresentam de seguida:

- Guardar as turmas que o utilizador escolha como favoritos;
- Mostrar o calendário escolar com os eventos das turmas favoritas;
- Exportar eventos de turmas;
- Mostrar o seu horário escolar, em função da sua lista de turmas favoritas;
- Barra de pesquisa de turmas;
- Disponibilizar uma interface com o utilizador seguindo as indicações da página *material.io*.

Além destes requisitos ficou também definido o seguinte conjunto de requisitos opcionais que poderão ou não ser cumpridos, dependendo do decorrer do projeto:

- Receber e listar as notificações das turmas que um utilizador escolheu tais como a publicação de um trabalho;
- Apresentar uma interface de autenticação para alunos e docentes;
- Alterar informações no perfil das turmas dependendo do papel do utilizador (aluno ou docente);
- Participação de utilizadores autorizados em eventos de uma turma.

1.5 Estrutura do relatório

No capítulo 2 são identificados os aspetos mais relevantes da arquitetura do sistema i-on e em particular do subsistema i-on Android. No capítulo 3 são apresentados cuidados que se tiveram no desenvolvimento do projeto. O capítulo 4 descreve em detalhe como funciona a navegação dentro da aplicação. A integração com a *Web API* i-on Core é descrita no capítulo 5. Os componentes constituintes da aplicação são descritos no capítulo 6. O capítulo 7 apresenta como foi realizado o tratamento de exceções. No capítulo 8 é apresentada a forma como foram seguidas algumas indicações da página material.io. Por fim, no capítulo 9 é recapitulado o projeto, apresentada uma análise crítica e são sugeridas algumas ideias de como o projeto pode ser continuado em trabalho futuro.

Capítulo 2

Arquitetura

Nesta secção é realizada uma descrição da arquitetura do sistema i-on, o enquadramento do subsistema i-on Android na iniciativa e, por fim, a arquitetura da aplicação Android.

2.1 Sistema i-on

O sistema i-on é um conjunto de peças de *software* que proporcionam soluções para facilitar o dia a dia da atividade académica.

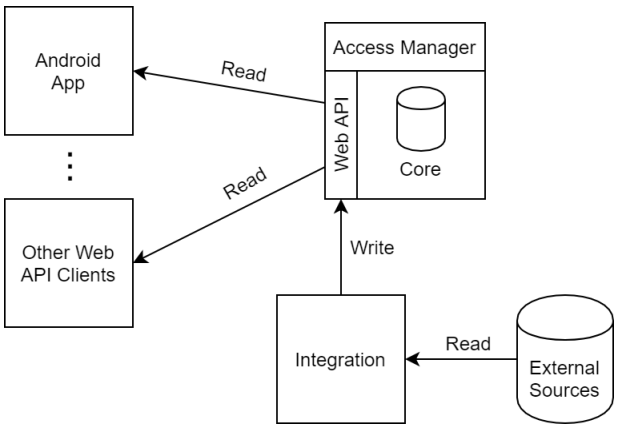


Figura 2.1: Arquitetura do sistema i-on

Na figura 2.1 estão identificados os componentes constituintes do sistema i-on e as relações entre eles. As setas indicam o fluxo da informação dentro do sistema. Na sua base está o projeto i-on Core que tem como funcionalidades guardar informações académicas como os cursos do ISEL ou as datas dos testes de uma unidade curricular, e expor uma interface que permita ler, alterar ou acrescentar esta informação. Esta interface é concretizada numa Web API que permite aceder às representações de um conjunto de recursos sobre os quais se podem executar métodos do *Hypertext Transfer*

Protocol (HTTP). Esta API pode ser separada em duas, a API de leitura e API de escrita. Ainda dentro do subsistema i-on Core existe o módulo *Access Manager* responsável por controlar os acessos dos clientes da Web API. Existe depois o projeto i-on Integration que é responsável por obter informações académicas recorrendo a fontes já existentes, como por exemplo o mapa dos exames, e converter e escrever esta informação via API de escrita. Finalmente existe um conjunto de possíveis clientes da API de leitura: *Web*; Android; IOS e Desktop.

2.2 Subsistema i-on Android

O subsistema i-on Android é uma interface com o utilizador (UI) para o sistema i-on desenvolvida através da *framework* Android. As *frameworks* dispõem de um conjunto de pontos de extensibilidade e a forma como estão definidos caracteriza o modelo de programação resultante e, consequentemente, a forma global da solução. A *framework* Android não é exceção e propõe um conjunto de recomendações (1) para a arquitetura de uma aplicação que foram seguidas no desenvolvimento deste projeto.

Como recomendado na documentação, foi seguido o princípio da separação de preocupações (2), isto é, todo o código não deve ser escrito numa *Activity* (3) ou num *Fragment* (4). Estas classes devem apenas controlar a UI e as interações com o sistema operativo Android.

A figura 2.2 apresenta a arquitetura proposta para aplicações presente na documentação Android adaptada ao contexto do subsistema i-on Android.

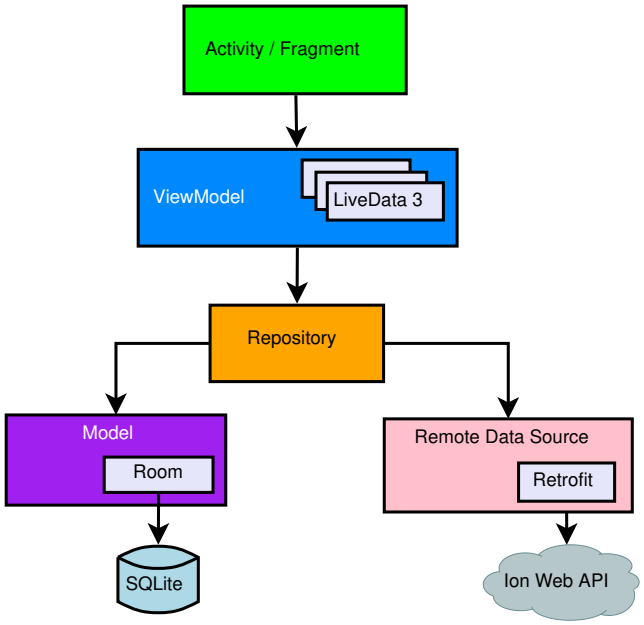


Figura 2.2: Arquitetura da aplicação i-on Android

Para guardar informação é recomendado utilizar um *ViewModel* (5) que está vinculado ao tempo de vida de uma *Activity* ou *Fragment* garantindo sempre a salvaguarda desta informação enquanto a *Activity* ou *Fragment* não for destruída/o pelo sistema operativo Android. No *ViewModel* são guardados objetos do modelo da aplicação como por exemplo a lista de turmas de uma dada disciplina.

Quando um *ViewModel* necessita de procurar informação, acede a um *Repository*, cuja responsabilidade é recolher informação. O *Repository* pode recolher informação efetuando um pedido HTTP a uma Web API (neste caso será a Web API disponibilizada pelo subsistema i-on Core) ou acedendo a uma base de dados (por exemplo, *Room Database*).

Room Database é uma biblioteca de *Object Relational Mapping* (ORM) (6) que permite aceder à base de dados SQLite local à aplicação.

Por fim, para realizar pedidos HTTP à Web API do sistema i-on Core é utilizada a biblioteca Retrofit (7).

Capítulo 3

Aspetos do desenvolvimento

Neste capítulo é introduzido o método de trabalho para o desenvolvimento do projeto e conclui-se com uma breve descrição das convenções de nomes e estilo de escrita de código.

3.1 Método de trabalho

Todo o projeto foi desenvolvido com a ferramenta de controlo de versões Git e o serviço Github. Tirou-se partido do mecanismo de *issues*. Uma *issue* pode ser uma tarefa a realizar para o projeto, uma nova funcionalidade que deve ser implementada, um problema que tem de ser resolvido entre outras coisas. O código relacionado com uma *issue* é colocado num ramo separado do ramo principal (*master*) onde podem ser colocadas consequentes modificações ao código. Este ramo é depois publicado no repositório e caso satisfaça os requisitos delineados na *issue* é criado um *pull-request* para que o código seja revisto pelos orientadores e pelos restantes elementos do grupo. Caso seja aprovado é transferido para o ramo principal. Esta abordagem permite que haja desenvolvimento em paralelo para o projeto.

3.2 Convenções de nomes e estilo de código

Uma aplicação desenvolvida através da *framework* Android contém um conjunto de recursos. Um recurso representa um elemento da aplicação relacionado com a UI, por exemplo a *layout* de uma *Activity*. Com o crescer do projeto o número de recursos também cresce e a manutenção da aplicação torna-se complicada. Além disto, como consequência de existirem várias pessoas a participar no desenvolvimento, tipicamente os nomes dos recursos não ficam uniformes o que dificulta a sua reutilização. Para evitar estes problemas adotou-se uma convenção de nomes de recursos que se encontra no repositório Github (8).

Também para uniformizar e melhorar a legibilidade do código desenvolvido utilizou-se uma ferramenta (9) presente no IDE IntelliJ IDEA que limpa, formata e reorganiza

o código. Por exemplo, elimina todos os *imports* que não estão a ser utilizados. Todo o código que é enviado para o repositório Github tem de primeiro ser limpo e formatado com esta ferramenta.

Capítulo 4

Navegação na aplicação

Neste capítulo são introduzidas possíveis abordagens de navegação numa aplicação móvel. De seguida são identificadas as áreas de contéudo da aplicação para onde é possível navegar. Depois é apresentada e justificada a escolha do mecanismo de navegação mais adequado para os destinos identificados anteriormente. Na secção seguinte é apresentado o método de partilha de informação entre destinos da aplicação e conclui-se o capítulo com alguns aspetos relevantes da implementação da navegação.

4.1 Navegação e tipos de navegação

No contexto de uma aplicação móvel Android a navegação é o ato de deslocação entre duas áreas de conteúdo da aplicação, por exemplo *Activities*. Existem várias abordagens que podem ser adotadas para navegar. Na mais clássica, o conteúdo da aplicação está distribuído por um conjunto de *Activities* sobre as quais se navega. Existe também uma abordagem híbrida em que algumas áreas de conteúdo da aplicação são representadas por *Activities* e outras por *Fragments*. Recentemente foi acrescentado à *framework* Android o componente *navigation* (10) que suporta uma outra forma de navegação onde existe apenas uma *Activity* que contém uma área de ecrã dedicada sobre a qual se navega entre *Fragments*.

4.2 Navegação na aplicação i-on Android

Na fase inicial do projeto foram desenhados *mockups* da interface da aplicação i-on Android para se compreender qual seria a experiência de utilização mais adequada para esta aplicação. Na figura 4.1 está representada a interface escolhida.

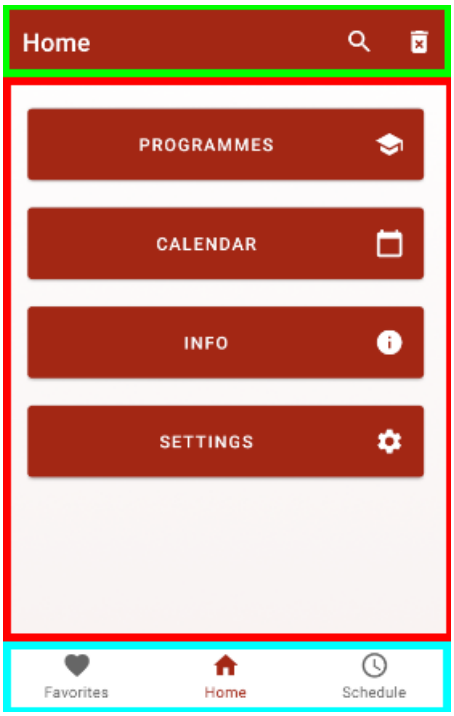


Figura 4.1: Interface da área inicial da aplicação i-on

Na área assinalada a verde está o título da área da aplicação representada, um botão para realizar pesquisas e um botão para eliminar as sugestões de pesquisas recentes. No fundo do ecrã, na secção assinalada a azul, estão três botões que permitem navegar para a lista de turmas favoritas, a página principal e o horário escolar. As secções assinaladas a verde e azul estão sempre visíveis, independentemente de onde o utilizador esteja na aplicação com exceção do horário do aluno, que ocupa o ecrã inteiro. Entre estas secções existe uma área reservada do ecrã assinalada a vermelho onde é apresentado o conteúdo principal, através do uso de *Fragments*, como por exemplo as cadeiras existentes num determinado período de um ano letivo. Cada área de conteúdo apresenta uma forma de navegar para outra área de conteúdo como por exemplo um botão.

Na figura 4.2 está ilustrado um grafo cujos pontos representam os destinos da aplicação e os possíveis caminhos de navegação.

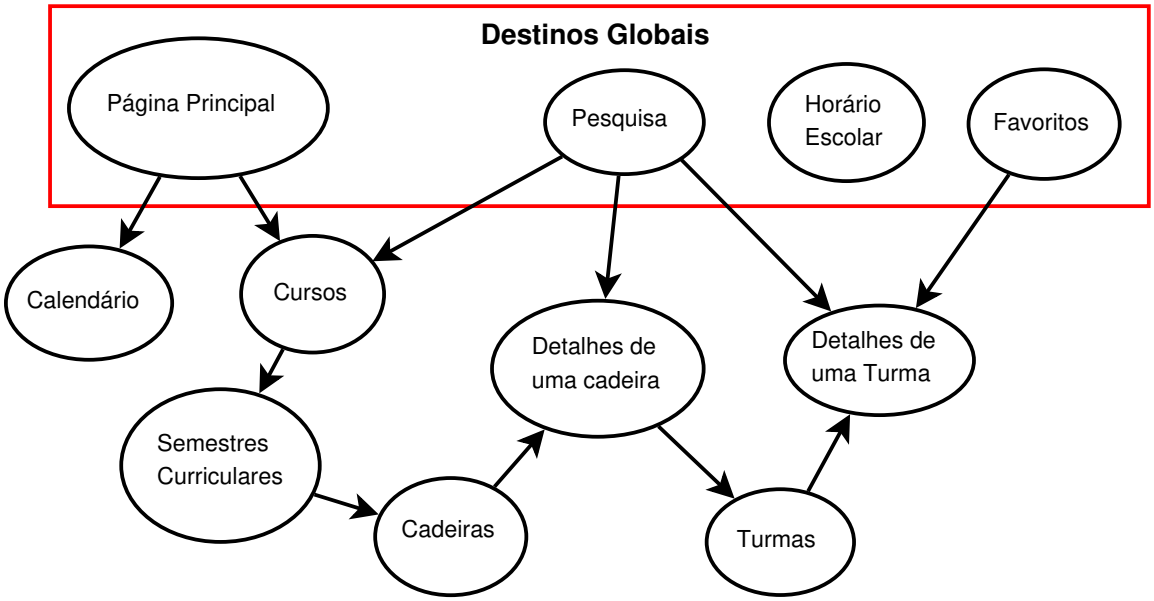


Figura 4.2: Grafo de navegação

4.3 Escolha do mecanismo de navegação

À partida o componente *navigation* pareceu o mais adequado para a aplicação tendo em consideração as características da experiência de navegação mencionadas na secção anterior e as características referidas nos próximos parágrafos.

O componente *navigation* contém um ficheiro no formato *extensible markup language* (XML) onde está definido o conjunto de destinos da aplicação e as relações entre estes, o que permite visualizar a navegação dentro da aplicação sem que seja necessário recorrer à análise do código fonte. Esta solução coloca em evidência um aspeto fundamental de uma aplicação *Android*: a navegação.

Outra vantagem do uso deste componente é que o *IDE* inclui uma ferramenta para visualização da navegação, designada *navigation editor* (11), que contém uma interface semelhante ao *layout editor* (12).

Do ponto de vista do modelo de programação disponibilizado pelo componente, a navegação entre *Fragments* é realizada através da obtenção de uma instância da classe `NavController` a partir do método de extensão `findNavController` que existe tanto para *Activities* como para *Fragments* e *Views*. Uma vez obtida uma instância da classe `NavController` podem ser chamados métodos com nomes sugestivos como `navigate`. É de notar que o *navigation component* é responsável pela gestão do *backstack* que contém os *Fragments* que vão sendo criados como consequência de navegação, o que simplifica a implementação de mecanismos como *bottom navigation*(13) que proporcionam a experiência de utilização que se pretende para a aplicação. O componente também suporta *deep linking* com recriação de um *backstack* realista.

Tendo em conta as características mencionadas, a experiência de utilização da

aplicação i-on Android é baseada no componente *navigation*. Existe a `MainActivity` que contém o *navigation host* sobre o qual se troca entre *fragments*, a secção de topo utiliza o componente `Toolbar` e a de fundo o componente `BottomNavigationView`.

4.4 Partilha de informação durante a navegação

Existem duas situações de comunicação entre destinos da aplicação: entre a `MainActivity` e os *Fragments*, e entre *Fragments*. A comunicação entre a `MainActivity` e os *Fragments* ocorre quando se realiza uma pesquisa. A caixa de pesquisa pertence à `MainActivity` enquanto que a obtenção dos resultados da pesquisa é feita no `SearchResultsFragment`[1]. Por causa disto é necessário partilhar o texto inserido na caixa de pesquisa com o *Fragment*. A comunicação entre *Fragments* ocorre em várias situações, por exemplo no evento de clique num elemento da lista de UCs de um dado curso. Neste caso o *uniform resource identifier* (URI) que identifica o recurso detalhes de uma UC é partilhado com o *Fragment* que apresenta todos os detalhes dessa mesma disciplina.

O componente *navigation* disponibiliza duas soluções para a passagem de informação entre *Fragments*. Na primeira solução os dados são associados a um *Bundle* que é enviado como parâmetro do método `navigate` da classe `NavController`. Esta solução só deve ser utilizada caso o volume de dados a partilhar seja muito reduzido (14), por exemplo um tipo primitivo. Se um objeto tiver de ser partilhado deve ser adotada a segunda solução, um *ViewModel* partilhado. Como os dados a partilhar nesta aplicação são na maioria objetos, adotou-se esta abordagem. Esta traz algum risco, pois caso existam muitos dados a partilhar, este *ViewModel* pode tornar-se demasiado extenso e assim perder legibilidade. Neste caso será considerada a hipótese de separar o *ViewModel* partilhado em vários *ViewModels* partilhados.

[1] org.ionproject.android.SearchResultsFragment

Capítulo 5

Integração com a Web API i-on Core

O atual capítulo descreve o processo de obtenção de dados vindos da Web API disponibilizada pelo projeto i-on Core. Começa-se por apresentar uma breve descrição acerca dos recursos da Web API. Depois é apresentado o método de obtenção dos dados, seguido pela identificação de alguns problemas na desserialização. Por fim é apresentada uma abordagem que visa reduzir o número de pedidos à Web API mas garantido que os dados se mantêm atualizados.

5.1 Recursos da Web API

Toda a informação académica utilizada dentro da aplicação é obtida a partir da Web API disponibilizada pelo subsistema i-on Core. Esta Web API é do tipo *hypermedia* portanto uma aplicação cliente deve ir obtendo os URIs através de propriedades presentes nos seus recursos. Uma grande vantagem de APIs deste tipo é que tradicionalmente a aplicação cliente de uma Web API tem de guardar um conjunto de URIs e sempre que estes sofrem alterações, a aplicação cliente tem de ser atualizada. Com APIs *hypermedia* só é necessário guardar o URI para o recurso raiz.

A Web API do sistema i-on Core é constituída por vários recursos que contêm informação acerca dos cursos, unidades curriculares e turmas do ISEL. Por exemplo o recurso *Programme* (15) contém os detalhes de uma dado curso do ISEL. Alguns destes recursos têm uma representação sumário presente em coleções, como por exemplo, o recurso *Programme Collection* que contém a lista de cursos do ISEL. Cada curso nesta lista é uma representação sumário que contém um URI que permite obter a sua representação detalhada. O recurso raíz da API está no formato JSON Home (16) e os restantes recursos seguem a especificação *hypermedia* Siren (17). Os eventos estão associados a uma turma ou instância de uma unidade curricular de um dado semestre de calendário.

Para cada tipo de recurso da Web API foi criado uma classe na aplicação. É de notar

que com a especificação Siren a representação de recursos é complexa e contém muitas propriedades para diferentes contextos. Como esta aplicação só tira partido da API de leitura nem todas as propriedades são necessárias como por exemplo a propriedade *actions*. As classes foram criadas com esta particularidade em mente e só contêm exclusivamente a informação necessária para a aplicação.

5.2 Implementação

Na implementação separou-se o processo de realização de um pedido à Web API i-on Core em duas interfaces, uma responsável por realizar o pedido HTTP, `IIonWebAPI` [1] e outra responsável pela desserialização da resposta ao pedido, `IIonMapper` [2]. Desta forma a solução é flexível e possibilita a criação de diferentes implementações, por exemplo, uma *mocked* e outra real, sem ser necessário alterar o resto do código do projeto.

5.2.1 Com dados *mock*

Os subsistemas i-on Android e i-on Core foram desenvolvidos simultaneamente o que implicou que durante uma fase inicial a Web API não estivesse disponível. Durante esta fase foi definida uma versão, ainda que incompleta, da documentação da API para que pudessem ser criados *mocks*. Foram propostas duas abordagens para a construção de *mocks*:

Object mocks São construídos *mocks* dos objetos modelo da aplicação resultantes das operações de desserialização e mapeamento da resposta a um pedido à API. A grande vantagem desta abordagem é que caso as respostas aos pedidos da API sofram alterações, os *mocks* não têm de ser alterados. Por outro lado não permite testar o módulo responsável por realizar a desserialização do corpo de um pedido HTTP.

String mocks São construídas *strings* que são cujo seu conteúdo é igual às representações externas dos recursos vindos da Web API. Com esta abordagem é possível testar desde o início os mecanismos de desserialização da resposta de um pedido. No entanto, sempre que a estrutura de um recurso da API é atualizada, as *strings* têm de ser alteradas.

Foi adotada a segunda abordagem, pois o processo de desserialização de respostas traz alguns desafios, que desta forma ficaram resolvidos mais cedo. Com base nesta escolha foi criada uma implementação da interface `IIonWebAPI`, `MockIonWebAPI` [3], que recebe um URI e retorna um objecto construído através da desserialização das *strings*

[1] org.ionproject.android.common.ionwebapi.IIonWebAPI
[2] org.ionproject.android.common.ionwebapi.IIonMapper
[3] org.ionproject.android.common.ionwebapi.MockIonWebAPI

mock associadas ao URI. A implementação também considera o tempo acrescido que existe num pedido HTTP, para este efeito foi acrescentado um tempo de atraso aleatório nos pedidos à classe `MockWebAPI`.

5.2.2 Com dados reais

O subsistema i-on Core expõe uma interface HTTP para aceder à sua Web API, logo teve de ser escolhida uma biblioteca que permitisse realizar pedidos HTTP. Um aspeto que tem de ser considerado para realizar esta escolha é que qualquer pedido HTTP realizado numa aplicação móvel tem de ser realizado numa *background thread*. Por isso, uma biblioteca que tire partido de um mecanismo de escrita de código assíncrono, como *coroutines* (18), é uma mais valia.

Encontraram-se algumas bibliotecas para a realização de pedidos HTTP das quais se destacam Volley (19) e Retrofit. Inicialmente tinha sido adotada a biblioteca Volley por causa da experiência ganha na UC de PDM e a estrutura do módulo do código responsável pela realização de pedidos HTTP foi desenhada com o modelo de programação da biblioteca Volley em mente. No entanto esta biblioteca ainda não dispõe de nenhuma funcionalidade que tire partido das *coroutines*, logo existiam planos para criar uma extensão à biblioteca que desse suporte a este mecanismo.

Na versão 2.6.0 da biblioteca Retrofit foi adicionado suporte ao modelo das *coroutines* e por este motivo decidiu-se trocar para esta biblioteca. Como a estrutura do módulo do código responsável pela realização de pedidos HTTP foi desenhada com flexibilidade em mente foi fácil trocar de biblioteca, e apenas se teve de implementar uma nova concretização da interface `IIonWebAPI`, `IonWebAPI` [4], tendo o restante código do projeto não sofrido alterações.

5.2.3 Desserialização da resposta de pedidos

Existem várias bibliotecas que permitem realizar a conversão de uma *string* cujo conteúdo está no formato JSON em um objeto Kotlin, das quais se destacam Jackson, Gson e Moshi. Todas disponibilizam um conjunto de mecanismos semelhantes no ponto de vista do modelo de programação. Além disso, todas têm vindo a ser atualizadas e melhoradas. No entanto, como o orientador deste projeto tem mais experiência com a biblioteca Jackson, esta foi a escolhida.

Uma vez decidida a biblioteca, foi necessário construir classes que respeitassem a estrutura das representações externas dos recursos vindos da Web API. À partida foram identificadas duas abordagens para a construção destas classes.

Na primeira, cada representação externa resultaria numa classe, no entanto, como na especificação Siren todas as propriedades com exceção da propriedade *properties* tomam sempre os mesmos nomes e tipos dos seus valores, ter-se-iam de repetir campos nestas classes. No caso do formato JSON *Home* não existe o mesmo problema porque só o recurso raiz é que tem esta representação.

[4] org.ionproject.android.common.ionwebapi.IonWebAPI

Na segunda abordagem em vez de se ter diferentes tipos para cada representação de cada recurso cria-se um tipo que representa a especificação Siren e que contém todas as possíveis propriedades desta representação. Os campos têm de tomar o valor nulo por omissão, pois em algumas representações não são utilizadas todas as propriedades. No caso do recurso raiz manter-se-ia a primeira abordagem.

Foi adotada a segunda abordagem. Na especificação Siren existe a propriedade *entities* que contém uma lista de entidades que podem ser de dois tipos, *Embedded Representation* e *Embedded Link*. Como se adotou a segunda abordagem não se sabe *a priori* qual será o tipo dos elementos da propriedade *entities*, logo houve a necessidade de encontrar uma forma de na desserialização se distinguir entre um e outro. A única forma de realizar esta distinção é pela existência de propriedades. Infelizmente a biblioteca Jackson não traz nenhuma funcionalidade que permita fazer este tipo de distinção o que resultou na necessidade de implementar um desserializador[5] personalizado. A implementação deste desserializador com *Jackson* foi realizada estendendo a classe abstrata `StdDeserializer` (20) e implementando o método `deserialize`. Depois o desserializador é utilizado colocando a anotação `JsonDeserialize` no tipo que o necessita.

Como o processo de desserialização é igual para a versão *mock* e para a versão real da classe responsável por realizar os pedidos à API, só foi necessário criar uma implementação da interface `IIonMapper`, `JacksonIonMapper` [6].

5.3 Supporte *offline*

Nesta secção é introduzido o modo da aplicação que permite lidar com dados académicos vindos da Web API que são raramente alterados, com o objetivo de disponilizar algumas funcionalidades quando o dispositvo não tem conectividade e reduzir o número de pedidos realizados à Web API mas mantendo os dados atualizados.

5.3.1 Problema

A navegação entre os vários *Fragments* e *Activity* da aplicação requer vários pedidos à Web API, que demoram algum tempo a processar (tempo do pedido HTTP + tempo de desserialização). Por exemplo, quando se navega do *Fragment* com a lista das disciplinas, `CoursesFragment` [7], para o *Fragment* com os detalhes de uma turma, `ClassSectionFragment` [8], são realizados 4 pedidos à Web API.

O conteúdo da aplicação depende da informação vinda do subsistema i-on Core o que torna a realização de pedidos à Web API uma acção muito frequente. No entanto, existe alguma informação que quase nunca é alterada, o que levou à procura de uma

[5] org.ionproject.android.common.dto.Siren
[6] org.ionproject.android.common.ionwebapi.JacksonIonMapper
[7] org.ionproject.android.courses.CoursesFragment
[8] org.ionproject.android.class$_{s}ection.ClassSectionFragment$

solução que permitisse guardar a informação que é pouco alterada localmente. Todavia, encontrar uma abordagem que resolva este problema não é trivial, pois existe demasiada informação que raramente é alterada. Por exemplo, como os detalhes de uma disciplina são alterados com pouca frequência e existem muitas disciplinas em vários cursos, teria de ser guardada muita informação, o que não é uma solução viável visto que a memória típica de um telemóvel é limitada. A solução passa por guardar alguma informação localmente.

5.3.2 Solução

Esta solução traz alguns desafios, começando pelo critério de escolha dos dados que ficam guardados localmente. Este critério depende do utilizador, pois cada utilizador necessita de dados diferentes. Por exemplo, um aluno que frequenta a UC de Programação (PG) provavelmente necessitará de ter a informação desta UC sempre o mais atualizada possível. Também é importante considerar as trocas de semestre. Quando um semestre termina deixa de ser necessário guardar a informação das disciplinas desse semestre, pois o mais provável é que o utilizador nunca mais volte necessitar dela. Existem também UCs que o utilizador consulta apenas uma vez, neste caso não devem ser guardados os detalhes localmente. Por fim, para distinguir as UCs que são acedidas com muita frequência das que foram acedidas pontualmente é necessário analisar a frequência dos acessos.

5.3.3 Implementação da solução

A implementação foca-se no uso de *workers* (21) que são um mecanismo disponibilizado pela *framework* Android para executar trabalho sincronamente numa *background thread*. Este mecanismo tem duas grandes vantagens. Em primeiro lugar, as *workers* podem executar trabalho periodicamente, e em segundo lugar, o seu tempo de vida estende-se além do tempo de utilização da UI da aplicação.

No caso concreto desta aplicação, sempre que um utilizador acede a um recurso é lançada uma *worker* à qual se associa um número de trabalhos e o recurso. Enquanto o número de trabalhos for maior que 0 esta *worker* tem de periodicamente verificar se houve alterações ao recurso e caso existam, atualizá-lo. Sempre que o utilizador volta a aceder ao recurso, o número de trabalhos é reiniciado. Quando o número de trabalhos atinge 0 esta *worker* remove-se e ao recurso da base de dados local e termina. Toda esta lógica está ilustrada no *flowchart* da figura 5.1.

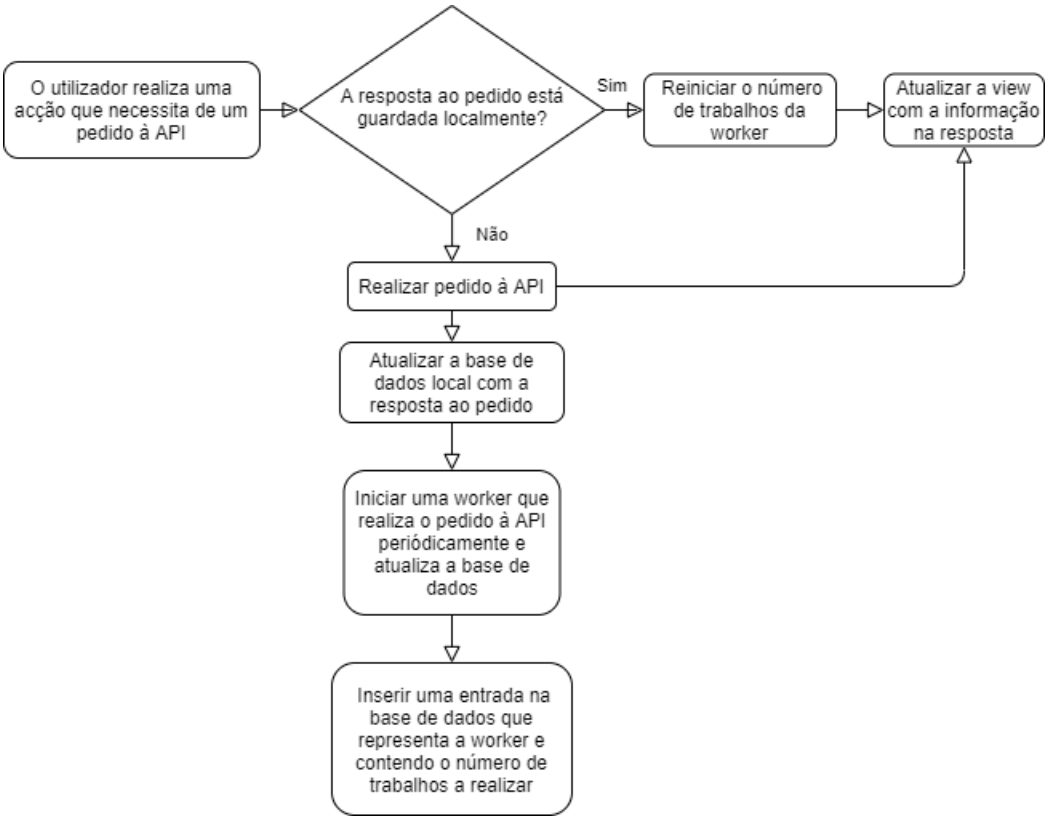


Figura 5.1: *Flowchart* pedidos à Web API

É importante referir que a frequência das *workers* depende do recurso. Por exemplo, é provável que os eventos de uma turma do semestre a decorrer sejam alterados com frequência, ao contrário dos detalhes de uma UC.

Também se teve o cuidado de separar toda a lógica de criação, remoção e contagem do número de trabalhos da *worker* da lógica de manutenção do recurso. Para esse efeito recorreu-se ao padrão *template method* e criou-se uma *worker* abstrata, NumberedWorker[9], que contém o método `doWork` que atualiza o número de trabalhos que a *worker* tem para realizar e caso seja o último trabalho, remove a *worker* da base de dados. Além deste método criaram-se mais dois métodos abstratos, um que é o trabalho normal no qual a *worker* deve pedir o recurso à Web API e comparar com o recurso na base de dados local, e outro, o último trabalho, que deve remover o recurso da base de dados. Para cada tipo de recurso criou-se uma implementação desta *worker*.

5.4 Monitorização da conectividade

A grande maioria das funcionalidades da aplicação i-on Android necessita de conectividade Wi-Fi ou dados móveis para realizar pedidos à Web API i-on Core. A

[9]org.ionproject.android.common.workers.NumberedWorker

conectividade não é um recurso que possa ser dado por garantido, pode não estar disponível. A secção anterior introduziu o conceito de suporte *offline* e neste sentido apresentou uma solução que garante que alguns dados vindos da API ficam guardados localmente de modo a disponibilizar funcionalidades da aplicação mesmo quando não existe qualquer tipo de conectividade. O utilizador deve ser informado, por exemplo na forma de um `Toast` que o seu dispositivo não está conectado e como consequência o conjunto de funcionalidades da aplicação está limitado. Para este efeito é necessário monitorizar a conexão à internet.

5.4.1 Implementação

A framework Android dispõe da classe `ConnectivityManager` (22) que permite monitorizar conexões à rede (Wi-fi, móvel, etc.). Como consequência do avançar das versões da framework Android, esta classe foi sofrendo alterações e alguns dos seus métodos tornaram-se obsoletos. Até à versão 23, inclusive, do Android *software development kit* (SDK), registava-se um BroadcastReceiver que era notificado no evento de alteração do estado de conexão. O estado da conexão era depois obtido através do método `getActiveNetworkInfo` que retorna um objeto do tipo `NetworkInfo`. Um objeto `NetworkInfo` contém um conjunto de métodos que permitem verificar o estado da conexão. Com a introdução da classe `NetworkCallback` (23) na versão 24 do Android SDK a forma de verificar se existe conectividade mudou completamente. Uma classe que estende de `NetworkCallback` pode redefinir o comportamento de métodos como `onLost` que é chamado quando a conexão à internet é perdida. Um objeto desta classe como o próprio nome indica, é um *callback* que deve ser registado sobre uma instância da classe `ConnectivityManager`. Esta instância é responsável por chamar os diferentes métodos de `NetworkCallback` face as alterações do estado da conexão à internet.

A versão mínima do Android SDK definida para a aplicação i-on Android é 21 para abranger um número elevado de dispositivos, logo foi necessário realizar código condicional face à versão do Android SDK presente no dispositivo. Para evitar verificar a versão sempre que se pretende monitorizar a conexão recorreu-se ao padrão de desenho *factory*. Existe o tipo `ConnectivityObservableFactory` [10] responsável por criar e retornar diferentes implementações da interface `IConnectivityObservable` [11] conforme a versão do Android SDK. Esta interface contém vários métodos para observar a conectividade e a sua implementação foi realizada de duas formas. A primeira, `ConnectivityObservable` [12], serve os dispositivos cuja a versão do Android SDK é pelo menos 24 e utiliza `NetworkCallback`. A segunda, `LegacyConnectivityObservable` [13], para dispositivos em que a versão do Android SDK é inferior a 24 utiliza a classe `NetworkInfo` em conjunto com um BroadcastReceiver.

[10] org.ionproject.android.common.connectivity.ConnectivityObservableFactory
[11] org.ionproject.android.common.connectivity.IConnectivityObservable
[12] org.ionproject.android.common.connectivity.ConnectivityObservable
[13] org.ionproject.android.common.connectivity.LegacyConnectivityObservable

Capítulo 6

Componentes da aplicação

O presente capítulo apresenta os componentes da aplicação. Em cada secção existe uma breve introdução onde se apresenta a motivação, características e principais problemas que podem afetar a implementação do componente. As secções concluem referindo os aspetos mais importantes da implementação do componente.

6.1 Favoritos

Para que o utilizador possa aceder aos detalhes de uma turma sem a necessidade de voltar a navegar entre um conjunto de destinos da aplicação, foi proposto que o utilizador possa escolher e guardar qualquer turma numa lista de favoritos.

A área da aplicação que apresenta os favoritos deve dar preferência aos favoritos do semestre atual mas permitindo o acesso aos dos semestres anteriores. Além disso, os favoritos são utilizados pelo calendário para apresentar eventos e pelo horário para apresentar as aulas.

6.1.1 Implementação

Os favoritos ficam guardados na base de dados local à aplicação e todo o acesso a eles é feito através da biblioteca *Room*. Para garantir que é dada prioridade aos favoritos do semestre atual, o *ViewModel* associado ao *Fragment* dos favoritos começa por obter todos os semestres de calendário e ordena-os por ordem decrescente com base no seu ano e estação de ano (Versão ou Inverno). Depois obtêm-se apenas os favoritos que estão associados ao primeiro elemento da lista de semestres de calendário que corresponde ao atual semestre. O *Fragment* de favoritos contém um Spinner (24) que permite trocar o semestre de calendário. Os favoritos ficam guardados numa instância da classe `MediatorLiveData` (25). Como `MediatorLiveData` estende `LiveData` é possível observar alterações que ocorram ao seu conteúdo e sempre que ocorrem atualizar a UI com a nova lista de favoritos. Sempre que o utilizador escolhe um novo semestre de calendário no Spinner, é removida a fonte associada à instância de `MediatorLiveData` e acres-

centada uma nova fonte que corresponde à nova lista de favoritos. Como consequência desta acção os observadores são notificados e atualizam a UI.

6.2 Horário

O horário de um aluno é uma tabela organizada por blocos de meia hora, entre as 8:00 e as 23:30, ao longo de uma semana. Este horário é preenchido com base nas aulas das disciplinas favoritas do semestre atual do utilizador. O horário é apresentado com o ecrã na horizontal para facilitar a visualização semanal. Caso fosse apresentado na vertical, o espaço reservado para os dias da semana ficaria demasiado apertado.

É importante conter o horário do aluno na aplicação, pois tal como já foi referido na motivação deste projeto 1.2, neste momento no ISEL os horários disponibilizados estão organizados por turma e contêm todas as UCs de todos os semestres, incluindo as UCs opcionais. Por causa disto o exercício de construção do horário é deixado a cargo de cada aluno. Com esta funcionalidade a construção do horário passa a ser da responsabilidade da aplicação.

Caso o utilizador tenha favoritos cujo período de aulas coincide, existem UCs que irão ficar sobrepostas no horário. Este problema é importante porque não existem limites no número de disciplinas que um utilizador pode adicionar aos favoritos. Encontraram-se duas soluções: colocar algo, num bloco com aulas a coincidir, em que o utilizador possa carregar para mostrar a lista de aulas que coincidem, por exemplo "..."; ou nos blocos em que aulas coincidem apresentar uma lista com os nomes das UCs todas. Adotou-se a segunda solução porque é mais rápido consultar o horário com os nomes das disciplinas em lista num dado bloco do que carregar num botão para apresentar as disciplinas que coincidem. Esta solução não é ideal, pois caso o utilizador tenha muitas disciplinas que coincidem, o horário fica ilegível. No entanto, o mais provável é que o utilizador comum só tenha como favoritas as disciplinas das turmas que frequenta, tipicamente 5, portanto numa utilização normal o horário continua legível.

6.2.1 Implementação

A implementação do horário tirou partido do componente *RecyclerView* para representar os blocos de aulas da semana. Este é tipicamente utilizado para representar listas, mas também pode ser utilizado para representar uma grelha utilizando o `GridLayoutManager` ao invés do `LinearLayoutManager`. Para gerar os blocos de meia hora criou-se um tipo auxiliar `Moment` [1] que tira partido de alguns mecanismos da linguagem Kotlin, como funções de notação infixa e sobrecarga de operadores, que melhoram a legibilidade do código, como se pode observar no seguinte exemplo:

[1] org.ionproject.android.common.model.Moment

```
(Moment(8, 0) until Moment(23, 0) step Moment.ThirtyMinutes)
    .map {
    it to it + Moment.ThirtyMinutes
}
```

Figura 6.1: Exemplo de código com a classe `Moment`

O troço de código da figura 6.1 gera uma lista de intervalos de meia hora desde as 8:00 até às 23:30. Desta forma a geração de blocos não fica *hardcoded*, o que torna mais simples a alteração da duração ou da data de começo e fim dos intervalos caso seja necessário no futuro.

6.3 Calendário

Neste momento no ISEL não existe um local central onde o utilizador possa consultar as datas de todos os eventos de uma UC, dado que os exames estão no mapa de exames, os testes estão no mapa de testes e os trabalhos poderão estar no moodle ou noutra plataforma como por exemplo o Github. O calendário da aplicação i-on Android traz um local onde os alunos podem consultar os eventos das suas turmas favoritas.

O calendário é uma área da aplicação onde são apresentados os dias de um dado mês. Neste estão contidos botões que permitem avançar ou recuar no tempo. Na área por baixo dos dias do mês, está uma lista com os eventos do dia do mês selecionado.

A *framework* Android disponibiliza o componente CalendarView (26), no entanto este componente não permite a associação de eventos a um dado dia do mês. Existem outras implementações de calendários que contêm esta funcionalidade que podiam ser adotadas. No entanto, como nenhuma destas alternativas é muito popular, se fosse adotada correria-se o risco de no futuro se perder a dependência. Este fator é decisivo para este projeto, pois prevê-se que possa continuar a haver utilização desta aplicação mesmo depois de terminada a UC de Projeto e Seminário.

Por estes motivos ficou decidido desenvolver um calendário personalizado que se adaptasse bem às especificações deste projeto e eventualmente a outros. Este calendário, que ficou com o nome JDCalendar (calendário do João e Diogo), está publicado no repositório público (27) para que possa ser utilizado pela comunidade.

Na figura 6.2 está um exemplo de uma possível UI do JDCalendar sem eventos. Nesta estão presentes os dias do mês de junho e o dia atual está marcado com uma cor mais acentuada.

Foi definido um conjunto de aspetos personalizáveis como a cor de fundo, entre outros, que estão detalhados na documentação do repositório. Todos podem ser alterados a partir dos atributos no ficheiro XML, onde o calendário está contido.

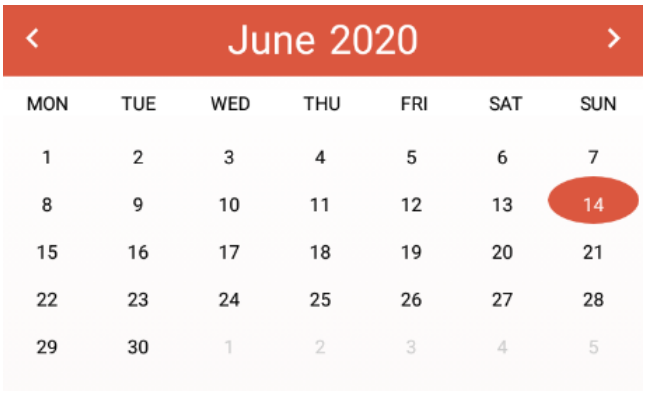


Figura 6.2: Calendário

6.3.1 Implementação

A implementação do calendário foi realizada seguindo os passos da documentação Android para a implementação de uma *View* personalizada (28).

Layout personalizável

Em primeiro lugar, para satisfazer o requisito de personalizar o calendário a partir de atributos XML, como o resto dos elementos visuais disponibilizados pela framework Android, identificou-se um conjunto de atributos que fariam sentido ser personalizáveis. Depois construiu-se um ficheiro XML de recursos com um item declare-stylable que contém todos estes atributos. A este ficheiro ficou associada a *View* que representa o calendário, `JDCalendar`. Desta forma os novos atributos ficam acessíveis quando o elemento `JDCalendar` é adicionado a um *layout*.

Modelo

Para obter o dia atual e realizar operações como avançar ou recuar no calendário recorreu-se a uma biblioteca Java designada por `Calendar` (29). Isto é possível porque a linguagem de programação Kotlin é interoperável com Java. O tipo `Calendar` contém um conjunto de operações que permitem avançar ou recuar num calendário, tais como `add(int field, int amount)` que soma ou subtrai uma certa quantidade de tempo ao calendário.

O modelo de programação da classe `Calendar` está desatualizado portanto criou-se um conjunto de métodos de extensão com nomes sugestivos para melhorar a sua legibilidade. A implementação destes métodos utiliza uma abordagem mais funcional onde, ao invés de se afetar uma instância de `Calendar` com métodos destrutivos como `add(int field, int amount)`, criaram-se métodos como `daysFromNow(days: Int)` [2] que retornam novas instâncias de `Calendar` sempre que são chamados.

[2] org.ionproject.android.calendar.JDCalendar.CalendarExtensions

Desta forma, as instâncias de `Calendar` criadas como consequência do uso destes novos métodos tornam-se imutáveis. É importante referir que apesar desta abordagem trazer mais trabalho ao *garbage collector* torna o código mais expressivo e perde-se a necessidade de ir consultar o código fonte, como no caso dos métodos tradicionais de `Calendar`.

Adapter personalizável

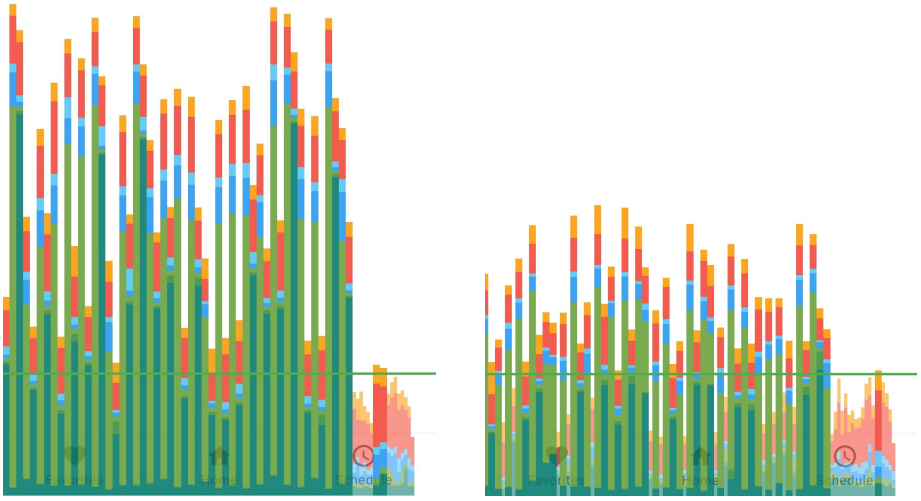
No componente *RecyclerView* disponibilizado pela *framework* Android é fácil controlar o processo de instanciação e ligação das *Views* ao modelo, sendo apenas necessário estender as classes `RecyclerView.Adapter` e `RecyclerView.ViewHolder` e implementar alguns métodos, entre os quais se destacam `onCreateViewHolder` e `onBindViewHolder`. O primeiro deve instanciar uma *View* que representa um elemento da lista e colocá-la num contentor, `ViewHolder`. O segundo deve preencher os elementos visuais da *View* com os valores do modelo. Esta abordagem traz flexibilidade à implementação da *RecyclerView*.

Decidiu-se adotar uma abordagem semelhante na implementação do `JDCalendar`, onde a única diferença é que no método `onBindViewHolder` é também passado como parâmetro o dia do mês que está a ser apresentado. Quem implementa o método tem a liberdade de ligar aquele dia do mês ao `ViewHolder` da forma que achar mais indicada para o seu contexto, inclusive pode aceder a um `ViewHolder` para verificar se naquele dia existe algum evento e mudar o aspeto visual da *View* contida no `ViewHolder` caso haja.

6.3.2 Otimizações

Uma vez que o calendário foi disponibilizado para a comunidade decidiu-se procurar melhorias ao seu desempenho. Para este efeito recorreu-se a uma ferramenta (30) que quando ativada permite observar um conjunto de barras no eixo vertical. O tamanho de uma barra vertical indica o tempo que a aplicação demora a processar uma *frame*. Além disso cada barra contém um conjunto de cores em que cada cor representa o tempo necessário para uma dada operação. Por exemplo, a cor laranja indica o tempo que o CPU está à espera que a GPU termine de realizar operações.

No eixo horizontal existe uma linha verde que representa os 16 milissegundos. Caso as barras estejam abaixo desta linha, a aplicação está a processar as *frames* a uma velocidade superior a 60 *frames* por segundo. Esta velocidade é a ideal para que a aplicação seja fluída, especialmente se contiver animações.



(a) Calendário antes de otimizações (b) Calendário depois de otimizações

Em ambas as figuras 6.3a e 6.3b foi realizada a ação de navegação por vários meses no calendário, o que resultou no aparecimento de um conjunto de barras ao longo do eixo vertical. Na figura 6.3a foi utilizada uma versão do componente `JDCalendar` sem otimizações e consegue-se identificar um conjunto de problemas.

Em primeiro lugar consegue-se observar que a maioria das barras têm uma grande parte da sua área pintada de verde escuro, indicando que a UI *thread* está a realizar demasiado processamento. Em segundo lugar algumas das barras têm uma grande parte da sua área pintada com um verde mais claro o que pode indicar que a aplicação está a demorar muito tempo com animações, está a executar muito código dentro de *callbacks* de eventos de *input* ou está a demorar muito tempo nos métodos, `onLayout` e `onMeasure` que são responsáveis por processar *layouts*.

A origem do primeiro problema deve-se ao facto de que sempre que o calendário carrega um novo mês é necessário obter a lista com os dias desse mês. Para isso tem de ser realizado um conjunto de operações demoradas que estavam a ser executadas sobre a UI *thread*. A solução foi passar a executar estas operações dentro de *coroutines* que utilizam *threads* de *background* libertando assim a UI *thread*. O segundo problema deve-se ao facto de se estar a utilizar *ConstraintLayout* em vários locais do calendário. Este tipo de *layout* tem o melhor desempenho em muitas situações mas quando se está a listar *Views* sem posicionamentos complexos, como é o caso, é mais adequado utilizar *LinearLayout*.

A figura 6.3b apresenta o resultado do uso da ferramenta depois das otimizações. Como se pode observar, houve uma descida considerável no tempo que a aplicação demora a processar as *frames*. Ainda não está abaixo da linha verde, no entanto, como não são animações, não é necessário obter uma velocidade de processamento de 60 *frames* por segundo.

6.4 Exportação de eventos

A aplicação i-on Android oferece um calendário, no entanto, podem existir utilizadores que preferem utilizar outras aplicações de calendário, como por exemplo o calendário da Google. Para satisfazer estes casos a aplicação oferece a opção de exportar eventos de uma turma.

6.4.1 Implementação

A implementação recorre a *Calendar intents* (31). Um *Calendar intent* é um *intent* implícito, logo não é especificado o nome da aplicação calendário, mas sim a ação de inserção de evento e um conjunto de valores que irá preencher o formulário de inserção de um evento em qualquer aplicação de calendário que suporte a *Calendar Provider API* (32).

6.5 Pesquisa

Existe uma secção no topo da interface da aplicação que permite realizar pesquisas. Esta pesquisa serve principalmente como uma forma de acesso mais rápido a conteúdos da aplicação. À medida que o utilizador insere caracteres, a aplicação apresenta sugestões que podem ser selecionadas para completar a pesquisa. Existem vários tipos de sugestões, tais como sugestões com base em pesquisas recentes, em informação guardada localmente ou remotamente. As sugestões apresentadas na aplicação i-on Android são geradas com base nas pesquisas recentes do utilizador. Assim que a pesquisa é submetida é realizado um pedido à Web API disponibilizada pelo subsistema i-on Core que irá responder com um conjunto de resultados relacionados com o texto pesquisado. Existem três tipos de resultados de pesquisa, *programmes*, *class sections* e *courses*. Consoante o tipo de resultado de pesquisa que for selecionado a aplicação leva o utilizador a uma área diferente da aplicação. Esta funcionalidade melhora bastante a experiência de utilização pois com a pesquisa, o utilizador deixa de necessitar de percorrer toda a aplicação até chegar à sua turma. Por exemplo, sem pesquisa, para chegar à turma LI61D de DAW é necessário realizar o seguinte percurso:

Área inicial → Cursos → LEIC → DAW → LI61D DAW.

Com a pesquisa o percurso fica substancialmente mais curto:

Área inicial → Resultados de pesquisa → LI61D DAW.

6.5.1 Implementação da pesquisa

A implementação da pesquisa tira partido do elemento *Search Widget* (33). Este é um de dois componentes recomendados pela *framework* Android para a realização

de pesquisa. Este componente pode ser colocado em qualquer lugar na aplicação, mas como está recomendado na documentação foi colocado na secção de topo.

Os resultados de pesquisa vindos da API i-on Core seguem a especificação Siren como a maioria dos recursos da API. Um resultado é caracterizado pela sua classe, que identifica o tipo de recurso associado, *programme*, *class section*, *course*, *calendar term* ou *class*. Além da classe todos os resultados contêm um id, um nome e um URI que identifica o recurso associado.

A aplicação i-on Android só tira partido dos resultados *programme*, *class section* e *course* porque não existem áreas na aplicação que apresentem exclusivamente tanto *calendar terms* como *classes*. É possível excluir estes tipos de resultados pois o URI *search* suporta a *query string types* que permite especificar os tipos de resultados de pesquisa que se pretende.

Para a representação dos tipos de resultado de pesquisa recorreu-se ao padrão *template method*. Para este efeito implementou-se uma classe base abstrata `SearchResult` [3] que contém um conjunto de campos comuns aos diferentes tipos de resultados e um único método abstrato, `navigateToResource`. Na implementação deste método deve ser realizado o conjunto de ações que permitem navegar para o *Fragment* adequado que representa o recurso associado ao resultado. Cada tipo de resultado é representado através de uma classe que estende `SearchResult`. Quando o utilizador carrega num resultado de pesquisa é chamado o seu método `navigateToResource` que por consequência navega para a área de conteúdo adequada.

Quando é realizada uma pesquisa, existe a possibilidade do número de resultados ser elevado, por exemplo, quando o texto de pesquisa é uma única letra. Além disto, como a área do ecrã é limitada, por vezes não é possível apresentar todos os resultados de uma vez. Neste caso, o utilizador tem de realizar a ação de deslizar sobre a lista para ver o resto dos resultados. Por causa disto não é adequado pedir todos os resultados de uma vez, mas sim à medida da necessidade do utilizador. Para resolver este problema é necessário haver suporte para paginação tanto na aplicação Android como na API. Felizmente, o URI *search* da API i-on Core suporta as *query strings limit* e *page* e a framework Android contém a *Paging library* (34) que traz o componente *Paged List*, uma lista que suporta paginação.

6.5.2 Implementação das sugestões de pesquisa

Na implementação das sugestões de pesquisa foram seguidas as orientações sugeridas pela documentação Android (35). Sempre que o utilizador insere um caractere é realizada uma chamada ao *content provider* (36) que é responsável por gerir as sugestões que são apresentadas. Neste contexto, o *content provider* acede à base de dados local para obter as sugestões cujos caracteres iniciais correspondem aos caracteres que estão a ser introduzidos pelo utilizador.

[3] org.ionproject.android.common.model.SearchResult

Capítulo 7

Tratamento de exceções

Nos capítulos anteriores foram apresentadas diversas funcionalidades da aplicação, no entanto não foi referido como a aplicação reage face a um erro nestas funcionalidades. Este capítulo começa por apresentar o comportamento que a *framework* Android tem por omissão e o comportamento que se pretende para a aplicação i-on Android. De seguida é explicado como se atingiu o pretendido, começando pelo serviço crashlitycs e depois com a implementação do mecanismo de tratamento de exceções.

7.1 Comportamento pretendido

Caso não exista qualquer tipo de tratamento de exceções numa aplicação Android, o comportamento pré-definido é fechar a aplicação, o que do ponto de vista da experiência de utilização não é agradável. No caso da aplicação i-on Android, pretende-se que quando ocorre uma exceção, dependendo do tipo da exceção e da área da aplicação onde esta ocorre, haja um comportamento diferente. Na seguinte lista estão presentes os comportamentos que se pretendem para a aplicação:

***Activities* e HomeFragment** uma exceção que ocorra numa *Activity* ou no HomeFragment deve resultar no seu fecho e deve ser iniciada uma nova *Activity* que apresenta a mensagem de erro.

Schedule Fragment a aplicação deve voltar para o *Fragment* anterior mas garantindo que o ecrã volta para a orientação vertical e tanto a secção de topo como de fundo estão visíveis. Finalmente deve notificar o utilizador da ocorrência do erro.

Resto dos *Fragments* a aplicação deve voltar para o *Fragment* anterior e notificar o utilizador acerca da ocorrência do erro.

Comum aos *Fragments* e *Activities* caso a exceção esteja relacionada com a falta de conectividade à rede móvel ou Wi-Fi, a mensagem mostrada ao utilizador deve informar que o dispositivo está sem conectividade.

Além destes comportamentos, também se pretende que todas as exceções que ocorram em qualquer dispositivo sejam registadas para que seja possível detetar erros, mesmo quando a aplicação já está disponível para o público.

7.2 Registo de exceções com Crashlitycs

De forma a cumprir o requisito de registar as exceções que ocorrem na aplicação, recorreu-se ao serviço *Firebase Crashlytics* (37). Este serviço é um dos produtos disponíveis na plataforma *Firebase* (38).

O serviço *Firebase Crashlytics* permite ter uma visão clara e em tempo real de relatórios de erros que ocorrem durante a execução da aplicação num determinado dispositivo. Isto permite detetar o tipo de erro, a razão do seu surgimento e as características do dispositivo onde surgiu esse erro.

O relatório dos erros pode ser consultado através da plataforma *Firebase* onde foi criado um projeto associado à aplicação. Desta forma é possível detetar eventuais falhas que estejam a prejudicar utilizadores.

7.3 Implementação da função de tratamento global de exceções

Como foi mencionado na secção 7.1 pretende-se que diferentes áreas da aplicação tenham diferentes comportamentos face à ocorrência de uma exceção.

Para atingir este objetivo implementou-se uma função de tratamento global de exceções, `GlobalExceptionHandler` [1]. Esta função captura qualquer exceção que ocorre na aplicação e responde com uma de três funções ordenadas por ordem de prioridade, a atual (`currExceptionHandler`), a base (`baseExceptionHandler`) e a predefinida (`defaultExceptionHandler`). Face à captura de uma exceção a função de tratamento global de exceções verifica se existe uma função atual definida. Caso exista executa-a, caso contrário executa a função base. Todas as exceções capturadas são registadas na plataforma Crashlitycs. Tanto a função atual como a base podem variar conforme a área da aplicação, por exemplo, a função atual registada pelo `ScheduleFragment` tem como comportamento navegar para o *Fragment* anterior, voltar a colocar o ecrã na vertical e colocar visível tanto a secção de topo como a de fundo.

Uma vez que a resposta que se pretende à ocorrência de uma exceção é igual em quase todos os *Fragments* criou-se um *Fragment* base, `ExceptionHandlingFragment`, [2] que regista uma função atual de tratamento de exceções no `GlobalExceptionHandler` num método virtual. Os *Fragments* estendem da classe `ExceptionHandlingFragment` e caso necessitem de um tratamento de exceções diferente só têm de redefinir o comportamento

[1] org.ionproject.android.error.GlobalExceptionHandler
[2] org.ionproject.android.ExceptionHandlingFragment

do método da classe base. Pela mesma razão também se criou uma *Activity* base, `ExceptionHandlingActivity` [3].

Também é necessário considerar as exceções que são lançadas quando a UI da aplicação não está visível, nas *Workers*. O `GlobalExceptionHandler` não abrange estas exceções portanto é necessário capturá-las de outra forma.

A solução que se encontrou foi envolver o método `doWork` presente na classe `CoroutineWorker` num bloco *try catch* que captura todas as exceções e regista-as na plataforma Crashlitycs. É também importante referir que no bloco *catch* se deve certificar que tanto a *Worker* como o recurso associado foram removidos da base de dados local, caso contrário esta fica com dados que só serão apagadas caso o utilizador desinstale a aplicação.

[3] org.ionproject.android.ExceptionHandlingActivity

Capítulo 8

Design e experiência de utilização

No desenvolvimento da aplicação seguiram-se algumas indicações da página material.io para que se obtenha uma interface elegante que proporcione uma boa experiência de utilização. A cor primária e secundária foram retiradas diretamente da página do ISEL e aplicadas ao longo da UI seguindo os princípios *consistent, distinct, intentional* (39). Na secção de navegação no fundo do ecrã colocaram-se mais de 2 destinos e menos de 6 destinos como indicado (40). Adicionou-se também o gesto de voltar para trás, *swipe right*. A operação de eliminar um favorito também seguiu as indicações (41) presentes na página material.io. Quando o utilizador desliza o dedo sobre um favorito, este mostra a imagem de um caixote para que o utilizador compreenda a consequência da acção antes da sua conclusão. Por fim, sempre que o utilizador realiza uma ação que resulta na mudança de *Fragment* é apresentada uma animação na transição.

8.1 Implementação do gesto *swipe right*

Na implementação do gesto *swipe right* também se seguiram recomendações (42) da página material.io. Implementou-se a transição de forma a que à medida que o fragmento é arrastado, este vai ganhando cada vez mais transparência, acompanhe o dedo do utilizador e suporte cancelamento, isto é, caso o utilizador deslize o dedo para o sentido oposto a transição é cancelada.

A implementação deste gesto foi realizada na função de extensão do tipo `View`, `addSwipeRightGesture` [1], para que qualquer componente que estenda a classe `View` pudesse recorrer a esta experiência de utilização caso necessitasse. À medida que o utilizador desloca o dedo, a propriedade `LayoutParams` presente numa instância da classe `View` vai sendo atualizada para que a sua representação na UI realize a translação para a direita. A propriedade `LayoutParams` indica a posição da *View* relativamente à *layout* onde está contida. Na implementação do gesto *swipe right* também se considerou a velocidade a que o utilizador realiza o gesto. Caso seja muito rápido, não é apresentada a transição e o *Fragment* troca de imediato. Para este efeito recorreu-se ao tipo

[1] org.ionproject.android.common.ViewGesturesExtensions

`VelocityTracker` que permite determinar a velocidade a que o utilizador realiza uma ação sobre o ecrã, foi definido um limite de velocidade que caso fosse ultrapassado, não é realizada a transição.

Esta implementação não chega para o gesto funcionar sobre *RecyclerViews*, pois verificou-se que um *RecyclerView* consome os eventos de toque que são realizados sobre a sua área no ecrã. Para garantir que isto não acontece foi criada uma classe que estende `LinearLayout`, `SwipeRightLinearLayout` [2], que intercepta os gestos de *swipe right* e não os passa ao resto dos elementos filhos que o compõem. Todas as áreas da aplicação que contêm um *RecyclerView* utilizam este novo tipo de *layout* para que o gesto *swipe right* funcione corretamente.

[2] org.ionproject.android.common.SwipeRightLinearLayout

Capítulo 9

Conclusão e trabalho fututo

Este último capítulo começa por recapitular o trabalho desenvolvido e apresentar os principais objetivos atingidos. Finalmente são indicadas algumas ideias acerca de como este projeto pode ser estendido e melhorado em trabalho futuro.

9.1 Conclusão

Hoje em dia, no ISEL, existe um conjunto de aspetos da atividade académica que podiam ser melhorados. A iniciativa i-on aparece com o objetivo de melhorar alguns destes aspetos e os seus projetos, i-on Android, i-on Integration e i-on Core, são o primeiro passo nessa direção. Poderão haver outros projetos no âmbito da iniciativa que deverão estender as funcionalidades destes projetos.

Os atuais projetos da iniciativa i-on já proporcionam um conjunto de funcionalidades que melhoram o quotidiano académico, tais como o horário do aluno que agora é construído pela aplicação i-on Android através da informação obtida do sistema i-on Core. Esta informação que já é obtida pelo sistema i-on Integration a partir de documentos do ISEL.

O principal desafio que este projeto trouxe foi o facto do desenvolvimento da aplicação estar a ser realizado em paralelo com desenvolvimento da Web API i-on Core. Mesmo com as reuniões que se tiveram em conjunto com os outros membros da iniciativa i-on, houveram algumas dificuldades na integração com a Web API. Por exemplo, como a documentação da API foi sofrendo alterações ao longo do seu desenvolvimento, trouxe consequências contraprodutivas tais como blocos de código que tiveram de ser reescritos.

Apesar destas dificuldades a versão final da aplicação i-on Android já está disponível na Google Play Store e disponibiliza um conjunto de funcionalidades que cumprem o seu objetivo inicial de facilitar o quotidiano de um aluno no ISEL.

9.2 Trabalho futuro

Por fim são apresentas algumas ideias para eventual trabalho futuro:

Contas para os utilizadores Seria muito interessante adicionar contas para utilizadores ao sistema i-on. Um utilizador podia criar uma conta e associar-lhe as suas turmas favoritas e se instalasse a aplicação noutro dispositivo continuar com as mesmas turmas. Desta forma perde-se a necessidade de selecionar todas as turmas favoritas novamente sempre que se volta instalar a aplicação noutro dispositivo.

Notificações acerca de eventos Sempre que ocorre uma alteração em algum recurso importante, como por exemplo numa data de um exame, deveria ser gerada uma notificação para o aluno. Esta notificação podia também ficar guardada na base dados local para que o aluno possa mais tarde ir consultá-la numa área dedicada às notificações.

Tirar partido do protocolo HTTP para melhorar a *cache* Nesta versão da aplicação já são guardadas algumas respostas da Web API i-on Core. No futuro esta funcionalidade deve ser estendida, tirando partido dos mecanismos de `cache` presentes no protocolo HTTP. Como por exemplo, realizar pedidos condicionais face à frescura do recurso já guardado localmente, levando à diminuição do peso computacional para o efeito.

Bibliografia

[1] *Guide to app architecture*, Android Developers. [Online]. Available: https://developer.android.com/jetpack/docs/guide

[2] *Guide to app architecture/separation of concerns*, Android Developers. [Online]. Available: https://developer.android.com/jetpack/docs/guide#separation-of-concerns

[3] *Introduction to Activities*, Android Developers. [Online]. Available: https://developer.android.com/guide/components/activities/intro-activities

[4] *Fragments*, Android Developers. [Online]. Available: https://developer.android.com/guide/components/fragments

[5] *ViewModel Overview*, Android Developers. [Online]. Available: https://developer.android.com/topic/libraries/architecture/viewmodel

[6] *Room overview*, Android Developers. [Online]. Available: https://developer.android.com/training/data-storage/room

[7] *Retrofit:A type-safe HTTP client for Android and Java*, Square. [Online]. Available: https://square.github.io/retrofit

[8] *Github*, IT service management company. [Online]. Available: https://github.com/i-on-project/android/wiki/Resource-naming-conventions

[9] *Reformat and rearrange code*, jetbrains. [Online]. Available: https://www.jetbrains.com/help/idea/reformat-and-rearrange-code.html

[10] *Navigation*, Android Developers. [Online]. Available: https://developer.android.com/guide/navigation

[11] *Navigation Editor*, Android Developers. [Online]. Available: https://developer.android.com/guide/navigation/navigation-getting-started#nav-editor

[12] *Layout Editor*, Android Developers. [Online]. Available: https://developer.android.com/studio/write/layout-editor

[13] *Bottom Navigation*, Material Design. [Online]. Available: https://material.io/components/bottom-navigation

[14] *Pass data between destinations*, Android Developers. [Online]. Available: https://developer.android.com/guide/navigation/navigation-pass-data

[15] *Programme*, i-on Core read API Docs. [Online]. Available: https://github.com/i-on-project/core/blob/master/docs/api/read/programme.md

[16] *Home Documents for HTTP APIs*. [Online]. Available: https://mnot.github.io/I-D/json-home/

[17] *Siren: a hypermedia specification for representing entities*. [Online]. Available: https://github.com/kevinswiber/siren

[18] *Coroutines for asynchronous programming and more*, Kotlin docs [Online]. Available: https://kotlinlang.org/docs/reference/coroutines-overview.html

[19] *Volley overview*, Android Developers. [Online]. Available: https://developer.android.com/training/volley

[20] *StdDeserializer*, Jackson Docs. [Online]. Available: https://fasterxml.github.io/jackson-databind/javadoc/2.9/com/fasterxml/jackson/databind/deser/std/StdDeserializer.html

[21] *Workers*, Android Developers. [Online]. Available: https://developer.android.com/reference/androidx/work/Worker

[22] *ConnectivityManager.NetworkCallback*, Android Developers. [Online]. Available: https://developer.android.com/reference/android/net/ConnectivityManager.NetworkCallbac

[23] *ConnectivityManager.NetworkCallback*, Android Developers. [Online]. Available: https://developer.android.com/reference/android/net/ConnectivityManager.NetworkCallback

[24] *Spinners*, Android Developers. [Online]. Available: https://developer.android.com/guide/topics/ui/controls/spinner

[25] *MediatorLiveData*, Android Developers. [Online]. Available: https://developer.android.com/reference/androidx/lifecycle/MediatorLiveData

[26] *Calendar View*, Android Developers. [Online]. Available: https://developer.android.com/reference/android/widget/CalendarView

[27] *Ion Android Repositoy*. [Online]. Available: https://github.com/i-on-project/android-calendar

[28] *Creating a view class*, Android Developers. [Online]. Available: https://developer.android.com/training/custom-views/create-view

[29] *Calendar*, Android Developers. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/util/Calendar.html

[30] *Inspect GPU rendering speed and overdraw*, Android Developers. [Online]. Available: https://developer.android.com/topic/performance/rendering/inspect-gpu-rendering

[31] *Calendar intents*, Android Developers. [Online]. Available: https://developer.android.com/guide/topics/providers/calendar-provider#intents

[32] *Calendar provider overview*, Android Developers. [Online]. Available: https://developer.android.com/guide/topics/providers/calendar-provider#top_of_page

[33] *Using the Search Widget*, Android Developers. [Online]. Available: https://developer.android.com/guide/topics/search/search-dialog#UsingSearchWidget

[34] *Paging library overview*, Android Developers. [Online]. Available: https://developer.android.com/topic/libraries/architecture/paging

[35] *Adding Custom Suggestions*, Android Developers. [Online]. Available: https://developer.android.com/guide/topics/search/adding-custom-suggestions

[36] *Content providers*, Android Developers. [Online]. Available: https://developer.android.com/guide/topics/providers/content-providers

[37] *Firebase Crashlytics*, Google. [Online]. Available: https://firebase.google.com/docs/crashlytics

[38] *Firebase*, Google. [Online]. Available: https://firebase.google.com

[39] *Applying color do UI: principles*, Material.io. [Online]. Available: https://material.io/design/color/applying-color-to-ui.html#usage

[40] *Bottom navigation: When to use*, Material.io. [Online]. Available: https://material.io/components/bottom-navigation#usage

[41] *Gestures: Show what gestures do*, Material.io. [Online]. Available: https://material.io/design/interaction/gestures.html#properties

[42] *Gestures: Provide realistic response*, Material.io. [Online]. Available: https://material.io/design/interaction/gestures.html#properties