Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e Computadores

KeepMyPlanet: Sistema Kotlin Multiplatform para Limpeza Ambiental

Diogo Ribeiro

Telefone: +351 911 889 669
E-mail: a47207@alunos.isel.pt

Rafael Pegacho

Telefone: +351 913 464 971
E-mail: a49423@alunos.isel.pt

Orientador:

Eng. Pedro Pereira
E-mail: pedro.pereira@isel.pt

2 de junho de 2025

1 Estrutura Geral do Projeto

A diretoria raíz agrega todos os componentes do projeto, desde a configuração do sistema de controlo de versões Git, ficheiros de licença e documentação geral, até ao próprio código fonte da aplicação Kotlin Multiplatform (KMP).

1.1 Diretoria Raiz do Repositório

No nível superior do repositório, encontramos ficheiros cruciais para a gestão e compreensão do projeto. O ficheiro `.gitignore` especifica os ficheiros e diretorias que devem ser ignorados pelo Git. O ficheiro `LICENSE` contém o texto integral da GNU General Public License Versão 3, que rege a utilização e distribuição do software. O ficheiro `README.md` oferece uma introdução ao projeto KeepMyPlanet, descrevendo o seu propósito, a tecnologia KMP utilizada, os autores e o orientador.

2 Projeto Kotlin Multiplatform: KeepMyPlanet

Dentro do diretório raiz do repositório Git, encontra-se a pasta principal da App, `KeepMyPlanet/`. Esta pasta contém a estrutura do projeto Kotlin Multiplatform.

2.1 Configuração do Projeto KMP

A configuração geral do projeto KMP é gerida por vários ficheiros. `.gitignore` especifica ficheiros a ignorar dentro do próprio projeto KMP. `README.md` fornece uma descrição técnica da estrutura do projeto, detalhando os módulos `composeApp`, `server` e `shared`. Ficheiros de configuração do Gradle, como `build.gradle.kts` (a nível do projeto), `settings.gradle.kts` (que define os módulos incluídos) e `gradle.properties` (que define propriedades globais do Kotlin, Gradle e Android), são cruciais para a compilação e gestão de dependências. A pasta `gradle/wrapper/` com o ficheiro `gradle-wrapper.properties` assegura uma versão consistente do Gradle. O ficheiro `docker-compose.yml` define a configuração para executar uma base de dados PostgreSQL num contentor Docker, incluindo a imagem, variáveis de ambiente, portas e volumes para persistência de dados e scripts de inicialização.

2.2 Módulo `shared`

O módulo ***`shared`*** desempenha um papel fundamental na arquitetura do sistema, funcionando como um repositório centralizado de componentes reutilizáveis e transversais a várias camadas da aplicação. Este módulo foi concebido para promover a coesão e a reutilização de código, agrupando elementos que são comuns a diferentes partes do sistema. Dentro do módulo ***`shared`***, encontram-se várias subpastas organizadas por responsabilidade:

- ***`/domain`*** - Contém os tipos de domínio que representam as entidades do sistema. Estes tipos encapsulam a lógica de negócio e as regras associadas às entidades, garantindo a consistência e a integridade dos dados ao longo do sistema.
- ***`/dto`*** - Inclui os ***`Data Transfer Objects`*** (DTOs), que são utilizados para transferir dados entre diferentes camadas ou componentes do sistema. Os DTOs permitem uma comunicação eficiente e segura, abstraindo os detalhes internos dos tipos de domínio e expondo apenas as informações necessárias.
- ***`/errors`*** - Reúne as exceções personalizadas criadas para o sistema. Estas exceções permitem uma gestão de erros mais clara e específica, facilitando a identificação e o tratamento de situações anormais durante a execução da aplicação.

- */mapper* - Contém as funções responsáveis por converter os tipos de domínio em *Data Transfer Objects* e vice-versa. Estas funções asseguram que os dados são transformados corretamente entre as diferentes representações, mantendo a separação de responsabilidades entre as camadas.
- */util* - Inclui funções utilitárias que fornecem funcionalidades auxiliares, como a formatação de datas e hora e a gestão de *password hashing*. Estas funções são projetadas para serem reutilizadas em várias partes do sistema, reduzindo a duplicação de código.

2.3 Módulo `server`

O módulo *server*, *backend*, é responsável por implementar a lógica central da aplicação, conectando as diferentes camadas do sistema e expondo os serviços através de uma interface de comunicação. Este módulo organiza-se em várias subpastas, cada uma com responsabilidades bem definidas:

- */api* - Contém os *endpoints* e os *handlers* (ou *controllers*), que são responsáveis por receber e processar os pedidos HTTP dos clientes. Esta camada atua como a interface pública do sistema, traduzindo as chamadas externas em operações internas.
- */service* - Reúne os serviços que encapsulam a lógica de negócio da aplicação. Estes serviços realizam validações, verificações de autorização e implementam as regras específicas do domínio, garantindo a integridade e consistência das operações.
- */repository* - Implementa a camada de acesso a dados, contendo interfaces de repositórios e as suas implementações, tanto em memória como para acesso a bases de dados. Esta camada abstrai os detalhes de persistência, permitindo que o restante do sistema interaja com os dados de forma desacoplada.
- */plugins* - Inclui configurações específicas e *status pages* criadas para monitorizar e personalizar o comportamento do sistema. Esta pasta centraliza os elementos que estendem ou configuram funcionalidades adicionais.
- */util* - Contém funções utilitárias que fornecem suporte ao módulo *server* como um todo.

2.4 Módulo `composeApp`

O módulo *composeApp* é responsável pela implementação do cliente da aplicação, com foco numa abordagem multiplataforma. Através do uso de tecnologias como o *Jetpack Compose Multiplatform*, o cliente é desenvolvido uma única vez e reutilizado em todas as plataformas suportadas. A diretoria */commonMain* concentra a lógica e os componentes principais, garantindo a consistência e a reutilização do código.

Dentro de *commonMain*, destacam-se as seguintes subpastas:

- `/data`
  - */api* - Inclui as configurações do *Ktor* e a criação do *HttpClient*, que é utilizado para realizar comunicações HTTP com o servidor.
  - */model* - Reúne classes como *EventInfo* e *UserInfo*, etc, que representam os dados manipulados no cliente. Estas classes são utilizadas para modelar as informações trocadas entre o cliente e o servidor.
  - */service* - Implementa os "serviços"do lado do cliente, contendo funções que realizam pedidos HTTP para os *endpoints* definidos no servidor. Estes serviços encapsulam a lógica de comunicação com o *backend*.

- */navigation* - Contém objetos que representam as vistas da aplicação. Esta estrutura facilita a navegação entre diferentes ecrãs, promovendo uma gestão clara e centralizada das rotas.
- */ui* - Abrange os componentes visuais da aplicação, incluindo ecrãs, componentes reutilizáveis e *viewModels*. Os *viewModels* são responsáveis por gerir o estado e a lógica de apresentação, conectando a interface do utilizador com os dados e serviços.

2.5 Scripts de Base de Dados (`db_init_scripts`)

A pasta `db_init_scripts/` contém os ficheiros SQL necessários para inicializar a base de dados PostgreSQL. `01_schema.sql` define o esquema das tabelas. `02_indexes.sql` cria os índices para otimizar as consultas. `03_seed_data.sql` popula as tabelas com dados iniciais para desenvolvimento e teste.

3 Documentação (`docs`)

A diretoria `docs/` contém a documentação do projeto. Inclui os diagramas `ClassDiagram.puml`, `UseCaseDiagram.puml` e `ScreenNavigationDiagram.png` (ficheiros PlantUML para geração de diagramas). `main.tex` é o ficheiro fonte LaTeX para a documentação formal do projeto, que pode gerar um PDF como `ppG33.pdf`. O logótipo `logo_isel.png` é utilizado nesta documentação.