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

Conteúdo

1 Introdução

Esta secção contextualiza o problema ambiental que o KeepMyPlanet visa endereçar e apresenta a solução proposta.

1.1 Contextualização do Problema Ambiental

Portugal, à semelhança do resto do mundo, enfrenta um desafio ambiental considerável no que toca à gestão de resíduos e à poluição. Segundo dados da Agência Portuguesa do Ambiente, em 2023, a produção de resíduos urbanos em Portugal atingiu os 5,3 milhões de toneladas, um acréscimo de 0,28% face ao ano precedente [1]. Preocupantemente, apenas 32% destes resíduos são encaminhados para reutilização e reciclagem, um valor significativamente abaixo da meta europeia de 55% fixada para 2025 [1].

Globalmente, a situação é igualmente séria. Anualmente, são gerados aproximadamente 2,24 mil milhões de toneladas de resíduos urbanos, prevendo-se um aumento de 73%, o que poderá totalizar 3,88 mil milhões de toneladas até ao ano de 2050 [2]. A poluição decorrente destes resíduos não se limita às áreas de deposição, estendendo-se à contaminação de espaços naturais, urbanos e, de forma crítica, às zonas costeiras [3]. Nestas últimas, estima-se que cerca de 80% do lixo marinho seja constituído por plásticos [3].

Apesar de uma crescente sensibilização para as questões ambientais e de um notório interesse da população em participar ativamente em iniciativas de limpeza, constata-se uma carência significativa de ferramentas eficazes para a identificação de áreas poluídas e para a coordenação otimizada destes esforços. A organização de ações de limpeza ambiental depara-se, atualmente, com obstáculos como a dispersão de informação, dificuldades na coordenação de voluntários e a ausência de mecanismos robustos para monitorizar o progresso e o impacto das intervenções realizadas.

1.2 A Solução Proposta: KeepMyPlanet

É neste contexto que se insere o ***KeepMyPlanet***, um sistema multiplataforma concebido com recurso a *Kotlin Multiplatform (KMP)* [4]. Esta plataforma tem como objetivo principal permitir a identificação e o mapeamento de zonas poluídas, facilitando simultaneamente a organização e a participação em eventos comunitários de limpeza. O sistema disponibiliza uma interface interativa através da qual os voluntários podem sinalizar geograficamente áreas que se encontrem poluídas. Adicionalmente, os utilizadores podem partilhar fotografias e descrições detalhadas das condições ambientais observadas nessas zonas. A plataforma permite ainda que os utilizadores criem novas iniciativas de limpeza para as áreas identificadas ou que adiram a eventos já organizados por outros membros da comunidade. Desta forma, o ***KeepMyPlanet*** ambiciona ser um ponto de convergência para a ação cívica e ambiental, oferecendo uma resposta tecnológica a um problema concreto, cada vez mais premente e relevante na sociedade atual.

2 Análise e Design do Sistema

Nesta parte, explicamos os requisitos que orientaram a criação do KeepMyPlanet. Também mostramos como foi pensada a arquitetura e o modelo de dados que fazem funcionar as partes já prontas do sistema.

2.1 Funcionalidades Implementadas e Requisitos Atendidos

O sistema foi concebido para ajudar em várias tarefas importantes na gestão do ambiente pela comunidade. As funcionalidades que já existem cobrem os usos mais importantes que identificámos, respondendo tanto a requisitos funcionais como não funcionais.

2.1.1 Gestão de Utilizadores, Zonas e Eventos

A gestão de utilizadores é completa, permitindo o registo, a autenticação segura com tokens JWT (JSON Web Tokens) [10], e a gestão de perfis, incluindo alteração de palavra-passe (protegida por hashing PBKDF2 [11])) e exclusão de conta, com verificações de dependências. No que concerne à identificação de zonas poluídas, o backend está preparado para registar estas áreas, incluindo a sua geolocalização, descrição, nível de severidade e associação a fotografias (via IDs). O sistema permite listar, visualizar, atualizar e excluir zonas, com autorização baseada no utilizador que reportou a zona ou organizador de evento associado à mesma. A plataforma oferece também uma gestão abrangente de eventos de limpeza. Organizadores podem criar novos eventos (associados a uma zona, cujo estado pode ser atualizado para "Limpeza Agendada"), especificando título, descrição, período e máximo de participantes. Utilizadores consultam detalhes e gerem a sua participação, enquanto organizadores podem cancelar ou concluir eventos, o que, por sua vez, atualiza o estado da zona associada. Para facilitar a coordenação, cada evento possui um chat em tempo real, utilizando Server-Sent Events (SSE) [14], onde participantes e organizador podem trocar mensagens de texto, com histórico mantido.

2.1.2 Aspetos Não Funcionais e Qualidade do Sistema

Diversos aspetos não funcionais foram cruciais. A escolha do Kotlin Multiplatform (KMP) [4] foi fundamental para maximizar a partilha de código (lógica de domínio, DTOs, serviços) entre as plataformas (Android, Desktop/JVM, WebAssembly) através do módulo `shared`, promovendo consistência e eficiência. O desempenho foi uma preocupação central. O backend Ktor [5], assíncrono, e índices otimizados na base de dados PostgreSQL [7] para colunas frequentemente consultadas visam acelerar as operações e garantir boa responsividade. A segurança da aplicação foi tratada com hashing PBKDF2 [11] para palavras-passe e JSON Web Tokens (JWT) [10] para controlo de acesso à API. A interface do utilizador, feita com Compose Multiplatform [6], foi pensada para ser intuitiva e de fácil utilização, seguindo boas práticas de design.

2.2 Arquitetura da Solução e Modelo de Dados

A arquitetura do KeepMyPlanet foi desenhada para ser modular, escalável e de fácil manutenção, aproveitando as capacidades do Kotlin Multiplatform [4], e é suportada por um modelo de dados robusto.

2.2.1 Estrutura Geral e Módulos Principais (KMP)

A solução está dividida em três módulos principais: `shared`, `server`, e `composeApp`. O módulo `shared` contém a lógica de negócios comum partilhada. O módulo `server`, implementado com Ktor [5], constitui o backend, fornecendo a API RESTful e gerindo a lógica do servidor. O módulo `composeApp` é a aplicação cliente multiplataforma, desenvolvida com Jetpack Compose Multiplatform [6], que consome a API do servidor.

2.2.2 Arquitetura Detalhada do Backend e Frontend

O backend Ktor [5] organiza-se em camadas: API (endpoints RESTful), Serviço (lógica de negócio, validações, autorizações) e Repositório (acesso a dados via SQLDelight [8]), utilizando plugins Ktor para autenticação (JWT) [10], serialização (kotlinx.serialization) [9], tratamento de erros (StatusPages) e SSE [14]. O frontend Compose Multiplatform [6] adota um padrão semelhante ao MVVM. As `ViewModel`s gerem o estado da UI (com `StateFlow`) e interações, comunicando com o backend através de Serviços cliente que utilizam Ktor HTTP [5]. A navegação é controlada por um sistema de rotas customizado.

2.2.3 Persistência de Dados com PostgreSQL e SQLDelight

A informação da aplicação é guardada numa base de dados PostgreSQL [7]. O schema, definido nos scripts de arranque e replicado no SQLDelight [8], inclui tabelas para as entidades principais (`users`, `photos`, `zones`, `events`, `messages`) e tabelas de junção, com chaves e constraints para integridade. A interação com a base de dados é feita através do SQLDelight, que gera código Kotlin type-safe a partir de queries SQL. `ColumnAdapter`s específicos asseguram a correta conversão de tipos de domínio Kotlin para tipos SQL.

3 Tecnologias Utilizadas

Para criar o KeepMyPlanet, usámos um conjunto de ferramentas e tecnologias modernas, escolhidas para que o sistema funcione da melhor forma em várias plataformas, sendo rápido e fácil de manter. Cada uma desempenhou um papel importante na construção do sistema.

3.1 Plataforma de Desenvolvimento e Frameworks Base

A base de todo o desenvolvimento foi o Kotlin Multiplatform (KMP) [4], que permitiu escrever muito do código uma só vez e usá-lo tanto no servidor como nas diferentes aplicações para o utilizador (Android, computador e WebAssembly) através do módulo `shared`. Para o desenvolvimento do backend e para a comunicação no cliente, usámos a framework Ktor [5]. No servidor, o Ktor forneceu uma forma leve e rápida de criar a API RESTful, com funcionalidades como routing, serialização JSON, autenticação e Server-Sent Events (SSE) [14] para o chat. Na aplicação do utilizador (`composeApp`), o Ktor Client foi usado para fazer pedidos à API do servidor. A interface que o utilizador vê foi construída com Jetpack Compose Multiplatform [6], um toolkit moderno que permitiu criar interfaces ricas, reativas e partilhar a lógica da UI entre plataformas.

3.2 Gestão de Dados e Persistência

Para guardar os dados da aplicação, como informações de utilizadores, zonas e eventos, escolhemos o PostgreSQL [7], um sistema de gestão de base de dados relacional robusto. A ligação entre o servidor Ktor e a base de dados foi feita com o SQLDelight [8], uma ferramenta que transforma comandos SQL em código Kotlin seguro, ajudando a evitar erros e simplificando o acesso aos dados. Para converter os objetos Kotlin para o formato JSON (e vice-versa) nas comunicações da API, usámos a biblioteca `kotlinx.serialization` [9].

3.3 Segurança e Ferramentas de Construção

Para saber quem está a usar a aplicação e para proteger os acessos à API, usámos JSON Web Tokens (JWT) [10], com o servidor Ktor a gerar e validar estes tokens. Para proteger as palavras-passe dos utilizadores, usámos a biblioteca `dev.whyoleg.cryptography` [11], especificamente o algoritmo PBKDF2. O sistema que usámos para construir o projeto foi o Gradle [12], com Kotlin DSL, que tratou das dependências e da compilação dos diferentes módulos. Para simplificar a configuração da base de dados no ambiente de desenvolvimento, usámos o Docker [13] e o Docker Compose..

4 Implementação Detalhada

Nesta parte, explicamos como foram feitas as diferentes partes do sistema KeepMyPlanet, mostrando as principais ideias de como o sistema foi desenhado e os componentes mais importantes de cada módulo, aproveitando a modularidade do Kotlin Multiplatform [4].

4.1 Módulo `shared`: Componentes Comuns da Aplicação

O módulo `shared` é a base da aplicação, contendo a lógica principal e os modelos de dados usados pelo servidor e clientes. Foram criados modelos de dados sólidos com classes Kotlin e `value class` (`Id`, `Description`, etc.) para segurança de tipos e validação. As entidades principais (`User`, `Zone`, etc.) são "data classes". Data Transfer Objects (DTOs), anotados com `@Serializable` [9], facilitam a comunicação com a API, com funções de mapeamento (`toResponse()`) para conversão. Utilitários como o `PasswordHasher` (com PBKDF2 da biblioteca `cryptography` [11]) e o `DateTimeUtil` (`kotlinx.datetime` [15]) são partilhados. Um sistema de exceções personalizado (`AppException` e subclasses) ajuda no tratamento de erros.

4.2 Módulo `server`: Lógica e Serviços do Backend

O módulo `server` trata de toda a lógica do servidor usando Ktor [5]. O servidor foi configurado com plugins Ktor para `Authentication` (JWT) [10], `ContentNegotiation` (JSON com `kotlinx.serialization` [9]), `StatusPages` (tratamento de erros), `SSE` [14] (chat em tempo real) e `CallLogging`. A API RESTful, organizada em ficheiros `*WebApi.kt`, expõe endpoints para autenticação (`/auth/login`), utilizadores (`/users` - registo, perfil, etc.), zonas (`/zones` - reporte, busca, gestão de fotos), eventos (`/events` - criação, participação, etc.) e mensagens de chat (`/events/{eventId}/chat`, incluindo stream SSE). A camada de serviço (`AuthService`, `UserService`, etc.) contém a lógica de negócio, validações e autorizações, orquestrando os repositórios. A camada de repositório (`Database*Repository`) usa SQLDelight [8] para acesso type-safe à base de dados PostgreSQL [7]. O `JwtService` gera tokens, e o `ChatSseService` difunde mensagens SSE para o chat.

4.3 Módulo `composeApp`: Interface do Utilizador Multiplataforma

O módulo `composeApp` cria a interface do utilizador com Jetpack Compose Multiplatform [6] e comunica com o backend Ktor [5]. A arquitetura segue um padrão MVVM-like, com um `AppViewModel` central e `ViewModel`s específicas por funcionalidade (`LoginViewModel`, etc.) que gerem o estado da UI (`StateFlow`) e eventos (`Channel`). Foram implementados ecrãs para Login/Registo, Home, Listagem/Detalhes/Criação de Eventos, Chat de Evento e Perfil do Utilizador, com componentes reutilizáveis como `LoadingButton`. A comunicação com o backend é feita por um cliente Ktor HTTP configurado para JWT e JSON, com a lógica de chamada à API encapsulada em serviços cliente. O projeto compila para Android, Desktop (JVM) e WebAssembly (WasmJs), com motores HTTP específicos e recursos de plataforma adequados. A estrutura iOS e o seu cliente HTTP estão funcionais.

4.4 Configuração e Inicialização da Base de Dados

A base de dados PostgreSQL [7] é configurada através de scripts SQL na pasta `db_init_scripts`: `01_schema.sql` (para a estrutura de tabelas), `02_indexes.sql` (para otimizar com índices) e `03_seed_data.sql` (com os dados iniciais). O Docker Compose [13] (`docker-compose.yml`) gere o serviço PostgreSQL no desenvolvimento, executando estes scripts automaticamente.

5 Resultados e Demonstração das Funcionalidades

Apresentam-se os resultados da implementação do KeepMyPlanet e o funcionamento das suas funcionalidades principais, visando uma plataforma intuitiva para gestão ambiental comunitária.

5.1 Interação do Utilizador com a Aplicação

A aplicação KeepMyPlanet (Android, Desktop/JVM, WebAssembly) oferece funcionalidades integradas para interação eficaz.

5.1.1 Registo, Login e Gestão de Perfil

Novos utilizadores criam conta no ecrã de registo (nome, email, senha segura). O sistema valida e regista, fazendo hashing da senha (PBKDF2) [11]. Utilizadores registados acedem via login, recebendo um token JWT [10] para autenticação. No perfil, podem ver/editar dados e alterar senha. A exclusão de conta é possível, condicionada a responsabilidades ativas.

5.1.2 Criação, Participação e Comunicação em Eventos

Utilizadores autenticados visualizam uma lista de eventos. Ao selecionar, acedem a detalhes. Organizadores criam novos eventos (título, descrição, período, zona, máximo de participantes). Utilizadores inscrevem-se ou cancelam inscrição. Organizadores cancelam ou concluem eventos. Cada evento tem um chat dedicado para comunicação em tempo real (via SSE) [14] entre participantes e organizador.

5.1.3 Funcionalidades de Zonas Poluídas

O backend e modelo de dados suportam o reporte e gestão de zonas (coordenadas, descrição, severidade, IDs de fotos). O sistema permite listar, ver detalhes, e a API suporta busca por proximidade. Dados de zonas podem ser atualizados com autorização.

5.2 Validação do Cumprimento dos Requisitos

O KeepMyPlanet, no estado atual, cumpre significativamente os requisitos funcionais e não funcionais. Gestão de utilizadores, criação/participação em eventos e chat estão funcionais. Sinalização e gestão de zonas são suportadas pelo backend. Requisitos não funcionais (KMP [4], segurança com hashing e JWT [10], desempenho com índices de BD) foram implementados. A arquitetura e tecnologias (Ktor [5], Compose Multiplatform [6]) demonstram um sistema robusto. Funcionalidades como mapas interativos no cliente e upload de fotos são próximos passos.

6 Conclusão

Referências Bibliográficas

Referências

[1] Agência Portuguesa do Ambiente (APA). (2023). *Relatório Anual de Resíduos Urbanos 2023*. Disponível em: https://apambiente.pt/sites/default/files/_Residuos/Producao_Gest%C3%A3o_Residuos/Dados%20RU/2023/raru_2023.pdf. Acedido em 10/03/2025.

[2] World Bank Group. (2022). *Solid Waste Management*. Disponível em: https://www.worldbank.org/en/topic/urbandevelopment/brief/solid-waste-management. Acedido em 10/03/2025.

[3] International Union for Conservation of Nature (IUCN). (2024). *Plastic Pollution*. Disponível em: https://iucn.org/resources/issues-brief/plastic-pollution. Acedido em 10/03/2025.

[4] JetBrains. *Kotlin Multiplatform*. Kotlin Foundation. Disponível em: https://kotlinlang.org/lp/multiplatform/. Acedido em 24/05/2024.

[5] JetBrains. *Ktor Framework*. Ktor.io. Disponível em: https://ktor.io/. Acedido em 24/05/2024.

[6] JetBrains. *Compose Multiplatform*. JetBrains. Disponível em: https://www.jetbrains.com/lp/compose-multiplatform/. Acedido em 24/05/2024.

[7] The PostgreSQL Global Development Group. *PostgreSQL: The World's Most Advanced Open Source Relational Database*. Disponível em: https://www.postgresql.org/. Acedido em 24/05/2024.

[8] Cash App. *SQLDelight*. Square, Inc. Disponível em: https://cashapp.github.io/sqldelight/. Acedido em 24/05/2024.

[9] JetBrains. *Kotlinx.serialization*. Kotlin Foundation. Disponível em: https://github.com/Kotlin/kotlinx.serialization. Acedido em 24/05/2024.

[10] Auth0. *JSON Web Tokens*. Disponível em: https://jwt.io/. Acedido em 24/05/2024.

[11] whyoleg. *Cryptography*. Disponível em: https://whyoleg.github.io/cryptography-kotlin/. Acedido em 24/05/2024.

[12] Gradle, Inc. *Gradle Build Tool*. Disponível em: https://gradle.org/. Acedido em 24/05/2024.

[13] Docker, Inc. *Docker*. Disponível em: https://www.docker.com/. Acedido em 24/05/2024.

[14] JetBrains. *Ktor Server-Sent Events*. Ktor.io. Disponível em: https://ktor.io/docs/sse.html. Acedido em 24/05/2024.

[15] JetBrains. *Kotlinx-datetime*. Kotlin Foundation. Disponível em: https://github.com/Kotlin/kotlinx-datetime. Acedido em 24/05/2024.