instituto superior de engenharia de lisboa

ZCAPNet - Sistema de apoio à Proteção Civil

Gonçalo Dimas

Luis Alves

Orientadores: André Lourenço
Artur Ferreira

Relatório da versão beta do projeto realizado no âmbito de Projecto e Seminário
Licenciatura em Engenharia Informática e de Computadores

Junho de 2025

Instituto Superior de Engenharia de Lisboa

ZCAPNet

48263 Gonçalo Alexandre Rocha Dimas

46974 Luis António Carvalho Alves

Orientadores: Artur Jorge Ferreira

André Ribeiro Lourenço

Relatório da versão beta do projeto realizado no âmbito de Projecto e Seminário
Licenciatura em Engenharia Informática e de Computadores

Junho de 2025

Resumo

A assistência à população em caso de acidente grave ou catástrofe é uma das preocupações dos agentes de Proteção Civil e está definida no âmbito dos Planos de Emergência Nacional, Especial e Municipal. Este visa proporcionar alojamento temporário e seguro aos cidadãos e famílias deslocados em virtude de tais eventos. O apoio é implementado através da ativação, instalação e gestão de Zonas de Concentração e Apoio à População (ZCAP), com a colaboração de diversas entidades, como a Autoridade Nacional de Proteção Civil, Câmaras Municipais, Instituto da Segurança Social, entre outras.

Garantir a existência de um sistema centralizado com uma estrutura de dados uniforme e capacidade de resiliência na eventualidade de falha das infraestruturas de comunicação é um dos objetivos deste projeto.

Por esse motivo, a arquitetura desenhada baseia-se na implementação de uma aplicação desktop autónoma sob o princípio de funcionamento offline-first com uma base de dados local e sincronização posterior com a base de dados remota.

Palavras-chave: Proteção Civil, ZCAP, Alojamento Temporário, Apoio Psicossocial em emergência, *offline-first*.

Lista de Acrónimos

ANEPC Autoridade Nacional de Emergência e Proteção Civil

API Aplication Programing Interface

CRUD Create Read Update Detele

DTO Data Transfer Object

HTTP Hypertext Transfer Protoloc

JPA Jakarta Persistence API

JSON JavaScript Object Notation

JWT JSON Web Token

NISS Numero de identificação da Segurança Social

ORM Object-relational mapping

REST Representational State Transfer

SHA Secure Hash Algorithm

SGO Sistema de Gestão de Operações

SQL Structured Query Language

UI User Interface

ZCAP Zona de Concentração e Apoio à População

Índice

Capítulo 1

Introdução

A assistência à população em caso de acidente grave ou catástrofe é uma das preocupações dos agentes de Proteção Civil e está definida no âmbito dos Planos de Emergência Nacional, Especial e Municipal.

Esta visa proporcionar alojamento temporário seguro, apoio psicossocial em emergência e necessidades básicas aos cidadãos e famílias deslocados antes, durante e após tais eventos (incêndio, inundação, terramoto, pandemia, ...).

O alojamento é implementado através da ativação, instalação e gestão de Zonas de Concentração e Apoio à População (ZCAP)[1], com a colaboração de diversas entidades, como a Autoridade Nacional de Proteção Civil, Câmaras Municipais, Instituto da Segurança Social, entre outras.

Atualmente a gestão da informação nestes edifícios, embora já efetuada com recurso a suporte informático, é realizada de forma precária pois, para além de dispersa e isolada por alojamento, não dispõe de um modelo de dados estruturado e interligado entre as diversas instalações e entidades.

Este formato inibe o cruzamento de dados para fins de relatório diário, estatísticas, gerir os recursos humanos disponíveis ou mesmo para procurar e encontrar parentes desaparecidos.

O objetivo deste projeto é desenvolver um sistema para ambiente Windows desktop, embora sob tecnologia multi-plataforma, com características que permita às entidades gerir e aceder às informações associadas à gestão dos espaços, pessoas e incidentes de forma estruturada, simples e rápida por meio de uma aplicação única, transversal e centralizada.

1.1 Organização do documento

O presente relatório tem como objetivo descrever de que forma foi idealizada e implementada a ideia de solução para o problema apresentado. Descrever-se-á nos capítulos que se seguem os componentes desenvolvidos assim como as escolhas e decisões principais na sua realização. No segundo capítulo é abordada a solução desenvolvida a desenvolver ao longo do semestre. No terceiro é explicado o modelo de dados, assim como a sua estrutura e tipos de abordagem.

No quarto capítulo é abordado o tema da aplicação Frontend, onde os utilizadores serão capazes de interagir com o sistema. No quinto capítulo é abordada a solução de Backend desenvolvida, que permite que o sistema armazene, processe e disponibilize os dados de forma eficiente, garantindo a comunicação entre a base de dados e a interface do utilizador. Por fim, no sexto capítulo são referidas as conclusões, observações, objetivos cumpridos e objetivos por cumprir.

Capítulo 2

Solução a desenvolver

O documento orientador da Instalação e Gestão de Zonas de Concentração e Apoio à População (ZCAP) [1], serviu de base à análise do problema a resolver com este projeto. Este, descreve todos os procedimentos necessários para o funcionamento destas instalações, antes, durante e após uma situação de exceção.

Da análise efetuada considerou-se relevante a necessidade de garantir o funcionamento, mesmo na eventualidade de uma suspensão de serviço das infraestruturas de comunicação, de uma aplicação onde seja possível registar e manter o apoio prestado, e os seus beneficiários.

Assim, foi decidido que a arquitetura da solução teria necessariamente de garantir que, apesar de a informação ser armazenada numa base de dados global e centralizada, servida e exposta por uma Web API, os clientes desta teriam de localmente conter informação suficiente para serem resilientes a esta falha eventual.

2.1 Sistemas semelhantes

A Autoridade Nacional de Emergência e Proteção Civil (ANEPC) utiliza a aplicação web SIGO - Sistema Integrado de Gestão de Operações [2]. Esta aplicação contém funcionalidades de suporte à instalação e gestão de ZCAPs; no entanto, foram identificadas lacunas neste sistema, as quais o projeto visa responder.

- A aplicação é utilizada apenas por agentes da ANEPC e apenas quando é ativada a Fase 4 ou superior do Sistema de Gestão de Operações (SGO) [3]. Nesta fase é possível que já estejam implementadas ZCAPs, as quais são geridas por outras entidades, sem acesso ao software.

- A inexistência de dados das estruturas existentes não permite apurar estatisticamente qual a capacidade de resposta em situação de crise.

- O software é acedido por browser e depende da existência de conectividade com a sua API.

2.2 Arquitetura da solução

A solução proposta segue uma arquitetura **modular e integrada**, composta por três camadas principais: a aplicação Desktop (*Web App*), a API de serviço (*Web API*) e a base de dados central (*Database*).

2.2.1 Web App

A aplicação será desenvolvida em **Flutter**, uma *framework* multi-plataforma que, embora criada para criar aplicações nativas para Android e iOS a partir de um único código-base, suporta da mesma forma o desenvolvimento para Desktop quer seja para Windows, Linux ou MacOs. A escolha inicial de tecnologia foi o *React Native* por ter eficiência e desempenho semelhante ao **Flutter** e a prévia experiência com **JavaScript** facilitaria o ritmo dos trabalhos. No entanto, após algum esforço percebeu-se que o suporte para Desktop e para o armazenamento local não é ainda suportado pela linguagem de forma a permitir a conclusão da mesma em tempo útil pelo que se mudou a abordagem para **Flutter**. Foi também avaliada a opção *Kotlin Multiplatform* tendo sido descartada pela performance relativa ao **Flutter**.

2.2.2 Web API

A camada de serviços será construída com **Spring Boot**, utilizando a linguagem **Kotlin** para o desenvolvimento do *backend*. A *Web API* será responsável por expor os serviços REST, sendo documentada e especificada através de ferramentas como **Swagger** e **OpenAPI Specification**. A segurança e a autenticação dos utilizadores serão geridas através de mecanismos de autorização e autenticação JWT.

2.2.3 Base de Dados

A camada de persistência de dados será suportada por uma instância **Microsoft SQL Server**, garantindo escalabilidade, fiabilidade e integração com as ferramentas existentes. A comunicação entre a API e a base de dados será realizada com recurso à biblioteca **JPA** com **Hibernate**, simplificando desta forma o mapeamento objeto-relacional.

2.2.4 Base de Dados Local

Um dos requisitos fundamentais da aplicação é a sua resiliência e capacidade de funcionar em modo **offline-first**. Para tal, será implementado um mecanismo de **persistência de dados local** com recurso à biblioteca **Isar**[https://isar.dev/], permitindo que os dados essenciais para o funcionamento da aplicação fiquem armazenados e disponíveis localmente. Desta forma, mesmo em cenários de falha de conectividade com a API, a aplicação poderá continuar a funcionar e registar operações, garantindo a continuidade do serviço. Quando a ligação for restabelecida, os dados serão sincronizados com a base de dados central.

2.2.5 Resumo da Arquitetura

A arquitetura da solução é ilustrada na Figura 2.1, onde se destaca a separação dos seus componentes, assim como a referência à capacidade de armazenamento de dados locais da aplicação.

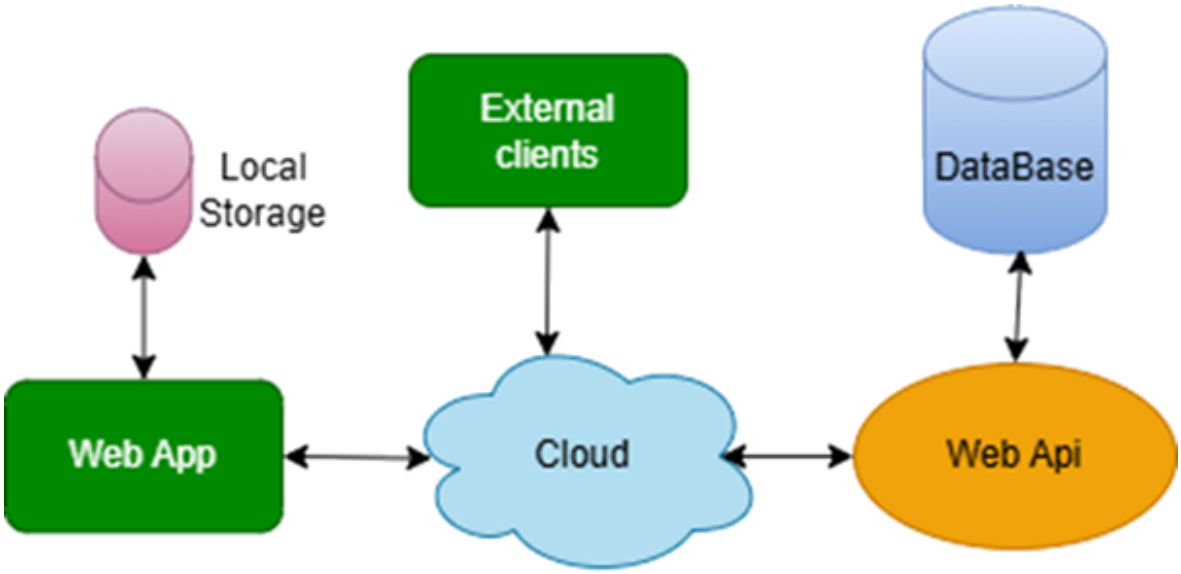

Figura 2.1: Arquitetura da solução proposta.

Capítulo 3

Modelo de Dados

O modelo de dados foi concebido para suportar a gestão e sincronização de zonas de apoio à população (ZCAP), garantindo a integridade e a acessibilidade dos dados em contextos distribuídos e offline.

Neste capítulo é apresentada a abordagem adotada para o modelo de dados, descrição e relação das entidades e análise quanto à extensibilidade e escabilidade.

3.1 Abordagem

Foi utilizada a notação entidade-relacionamento com base na metodologia de Chen. O modelo foi normalizado até à 3ª forma normal para eliminar redundâncias.

A modelação da base de dados foi realizada com base no modelo entidade-relacionamento (ER), tendo como objetivo representar de forma clara e estruturada os elementos essenciais do sistema e as suas inter-relações. Esta abordagem permitiu identificar as entidades, os seus atributos e as relações entre elas, com base nos requisitos funcionais e operacionais definidos para a aplicação.

Após a definição inicial do modelo conceptual, procedeu-se à sua conversão para um modelo relacional, adequado à implementação numa base de dados Microsfot SQL Server. Esta conversão incluiu a definição de chaves primárias e estrangeiras, bem como a aplicação de restrições de integridade e validação de dados.

Para apoiar o processo de modelação e visualização da estrutura da base de dados, foi utilizada a ferramenta SQL Server Management Studio, que facilitou a criação e edição do modelo lógico, bem como a geração automática de scripts SQL para a criação das tabelas.

3.2 Estrutura do modelo

Este modelo de dados organiza-se em cinco grandes grupos: estrutura hierárquica, pessoas, utilizadores, zonas de concentração e apoio à população (ZCAP) e incidentes.

3.2.1 Estrutura Hierárquica

A estrutura hierárquica serve para armazenar dados relacionados com as divisões administrativas, sendo esta elaborada de forma genérica. É possível armazenar uma divisão administrativa, o nível de cada divisão administrativa e ainda guardar detalhes sobre a mesma.

Para o armazenamento de níveis administrativos é utilizada a tabela "treeLevels"representada na Figura 3.1. Esta visa guardar o nível e nome de cada nível de divisão administrativa, sendo 1 o maior nível.

Cada elemento da estrutura hierárquica, é armazenado na tabela "tree"representada na Figura 3.1. Esta tabela visa relacionar o nome de uma região com o seu respetivo nível e, se esta não tiver o nível mais alto estabelecido, relacionar com uma região pai, sendo este outro elemento da estrutura.

Todos os detalhes de cada elemento da estrutura são armazenados na tabela "treeRecordDetails"representada na Figura 3.1. Esta armazena o valor do detalhe, o tipo de detalhe e a que elemento da estrutura hierárquica pertence. O tipo de detalhe é uma chave estrangeira de "treeRecordDetailTypes", que tem como objetivo armazenar os tipos de detalhes existentes no sistema.

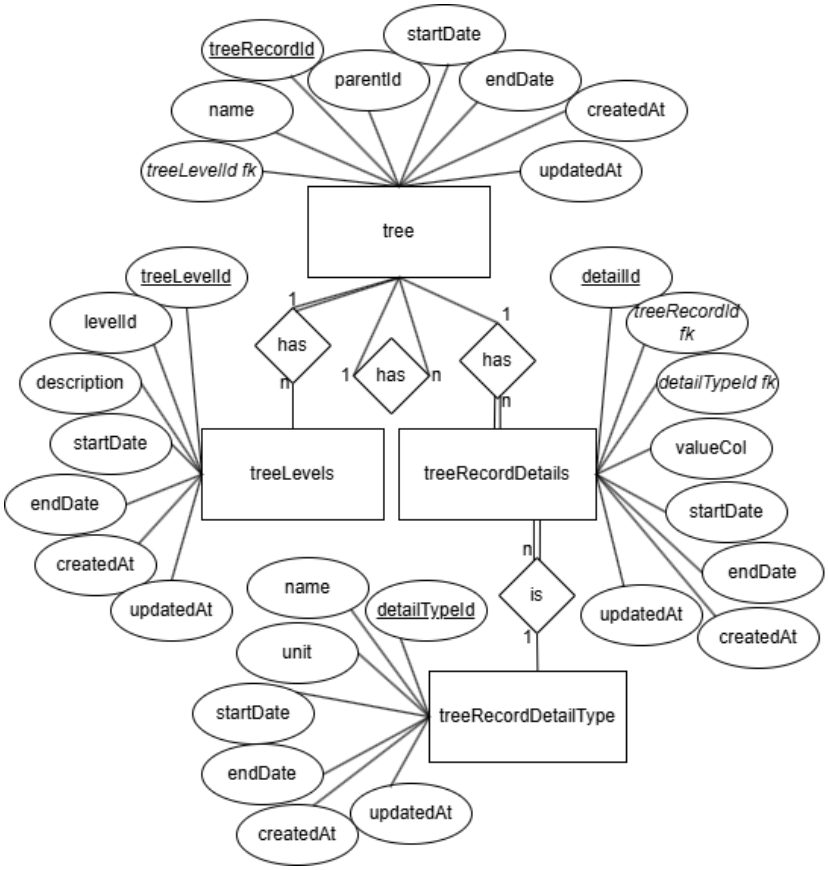

Figura 3.1: Modelo de Dados da estrutura hierárquica.

3.2.2 Pessoas

O modelo das pessoas serve para o registo e gestão de todos os indivíduos ou famílias que, devido a uma emergência ou desastre grave, necessitam ou necessitaram de frequentar uma zona de concentração de apoio à população (ZCAP). Na Figura 3.2 apresentam-se as tabelas necessárias para o funcionamento do sistema, sendo estas as mais relevantes.

No armazenamento e gestão das informações de uma pessoa é possível guardar informações quanto ao nome, idade, contacto, código do país do contacto, local de residência e data de entrada. Estas informações serão armazenadas numa primeira fase de entrada na ZCAP. Numa segunda fase é possível armazenar informação como data de saída, aniversário, nacionalidade, morada, número de identificação de segurança social (NISS), destino de saída da ZCAP e o contacto desse mesmo destino. Diferentemente de muitas outras tabelas, esta não tem registo de "startDate"e "endDate"uma vez que saberemos que esta só estará ativa enquanto não existir registo da data de saída da ZCAP. Estas informações são armazenadas na tabela "persons"como demonstrado na Figura 3.2. Os campos não representados na figura são campos não obrigatórios para o registo.

As pessoas podem ter necessidades especiais (medicamentos, gravidez, ...) ou necessidade de algum tipo de suporte (alojamento, comida, vestuário, ...). As necessidades especiais disponíveis no sistema são armazenadas na tabela "specialNeeds", onde se encontra apenas o nome da necessidade especial. Na tabela "personSpecialNeeds"é onde a relação da pessoa com a necessidade especial é associada. Uma pessoa pode ter várias necessidades especiais e várias pessoas podem ter a mesma necessidade especial. Para a necessidade de suporte é exatamente o mesmo que a necessidade especial. Existe uma tabela "suportNeeded"que armazena todos os tipos de necessidades de suporte aceites pelo sistema e este é associado à pessoa pela tabela "personSupportNeeded".

Por fim, os indivíduos que necessitam de frequentar uma ZCAP, muitas vezes têm familiares. Para armazenamento e gestão dessa informação, o sistema admite os tipos de relação (pai, mãe, primo, tia, ...) entre pessoas na tabela "relationType". Para associação do tipo de relação com as pessoas, é utilizada a tabela "relation", representada na Figura 3.2. Nesta tabela é representado o tipo de relação que a pessoa 1 tem com a pessoa 2.

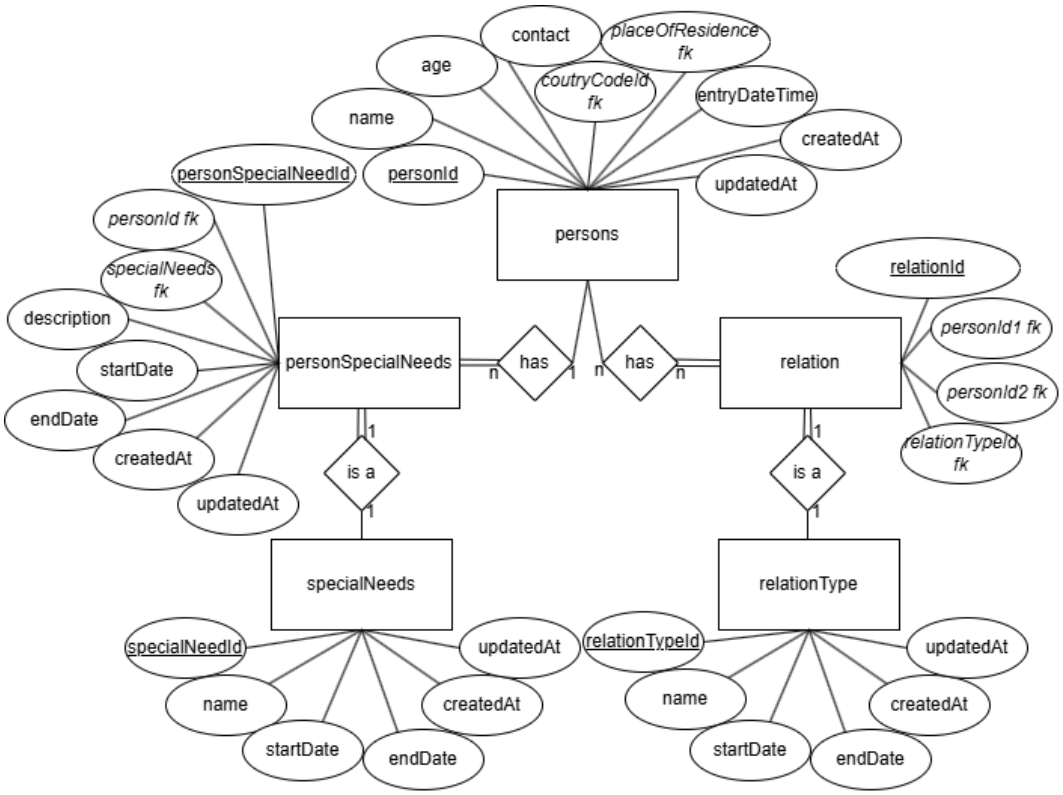

Figura 3.2: Modelo de Dados das pessoas.

3.2.3 Utilizadores

O modelo de dados dos utilizadores serve para armazenar e gerir as informações de todos os colaboradores, direta ou indiretamente, das ZCAPs. Para cada utilizador, é necessário haver registo de suas permissões dentro do sistema. O modelo está representado na Figura 3.3.

No armazenamento de um utilizador no sistema, é possível armazenar nome, nome de utilizador, senha, perfil de utilizador e perfil de dados do utilizador (permissão por região).

Os tipos aceites pelo sistema dos tipos de perfis de utilizadores são armazenados na tabela "userProfileDetails". Estes têm informação sobre o tipo de acesso às opções dentro da aplicação e o tipo de permissões CRUD (Create Read Update Delete), sendo estas apenas leitura, apenas escrita ou escrita e leitura. Os perfis de utilizador são armazenados na tabela "userProfiles", que está como chave estrangeira em "users", e armazena um nome e a associação a um tipo de perfil de utilizador.

Para regular as permissões por regiões é necessário armazenar um perfil de dados do utilizador (tabela "userDataProfiles"na Figura 3.3). Este guarda apenas o nome do perfil. Para realizar a associação do perfil com as regiões a qual o utilizador tem acesso é necessário criar e armazenar um detalhe de perfil de dados do utilizador na tabela "userDataProfileDetails". A informação armazenada contém um perfil de dados do utilizador e a que zona territorial (elemento da estrutura hierárquica referido na Subseção 3.2.1) que o perfil tem acesso.

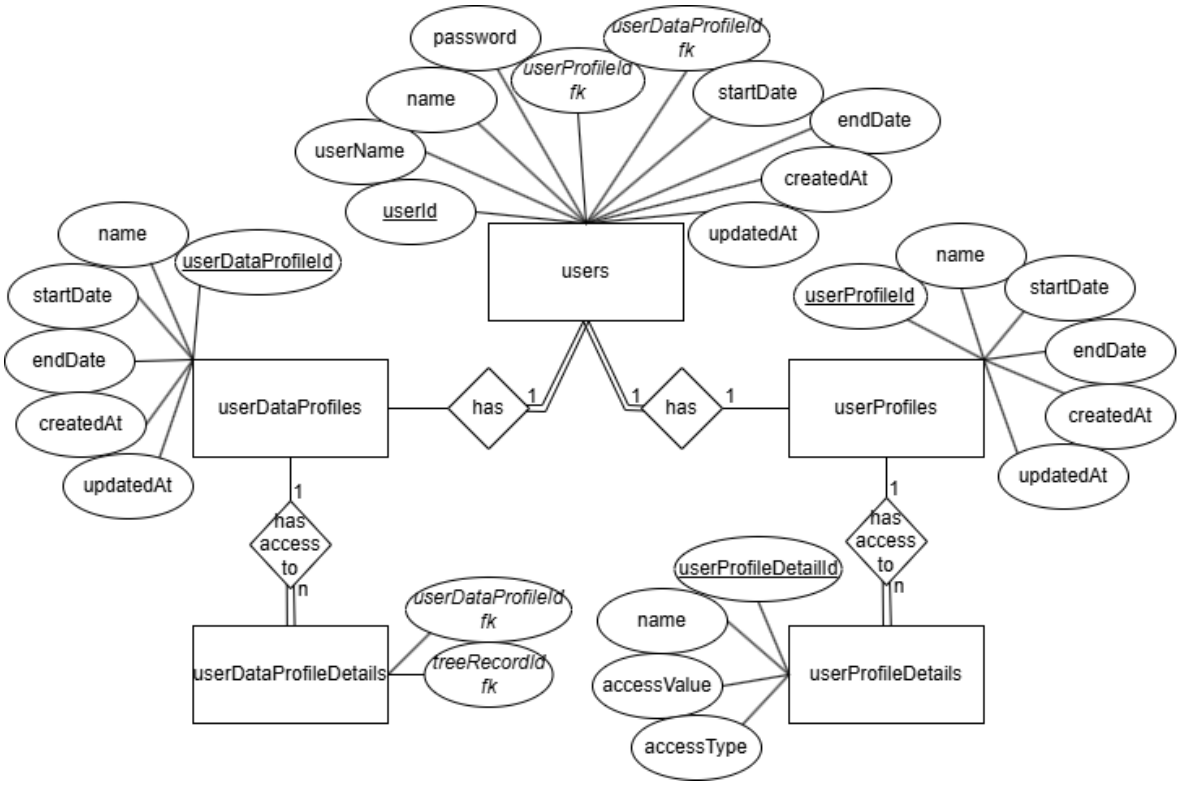

Figura 3.3: Modelo de Dados dos utilizadores.

3.2.4 ZCAPs

O modelo das ZCAPs, que está representado na Figura 3.4, representa todas as informações sobre cada ZCAP e sobre todas as entidades profissionais envolvidas no apoio psicossocial às populações.

As informações gerais de uma ZCAP são armazenadas na tabela "zcaps". Esta tem informações como nome, tipo de edifício, morada, região na estrutura hierárquica e entidade responsável. É possível ainda armazenar valores de latitude e longitude, embora não sejam obrigatórios.

O sistema suporta todos os tipos de edifício armazenados na tabela "buildingTypes"(escola, pavilhão, ...).

A região na estrutura hierárquica é chave estrangeira associada a um elemento da estrutura falado na Subseção 3.2.1). Isto permite distinguir, em termos de permissões dos utilizadores, quem tem acesso à ZCAP pela região.

O sistema aceita qualquer tipo de entidade armazenada na tabela "entityTypes"(Bombeiros, Polícia de Segurança Pública, ...). Quanto às entidades responsáveis, estas são armazenadas na tabela "entities". Estas contêm o tipo de entidade associado, nome e pelo menos um contacto. Também é possível armazenar email e um segundo contacto, se necessário.

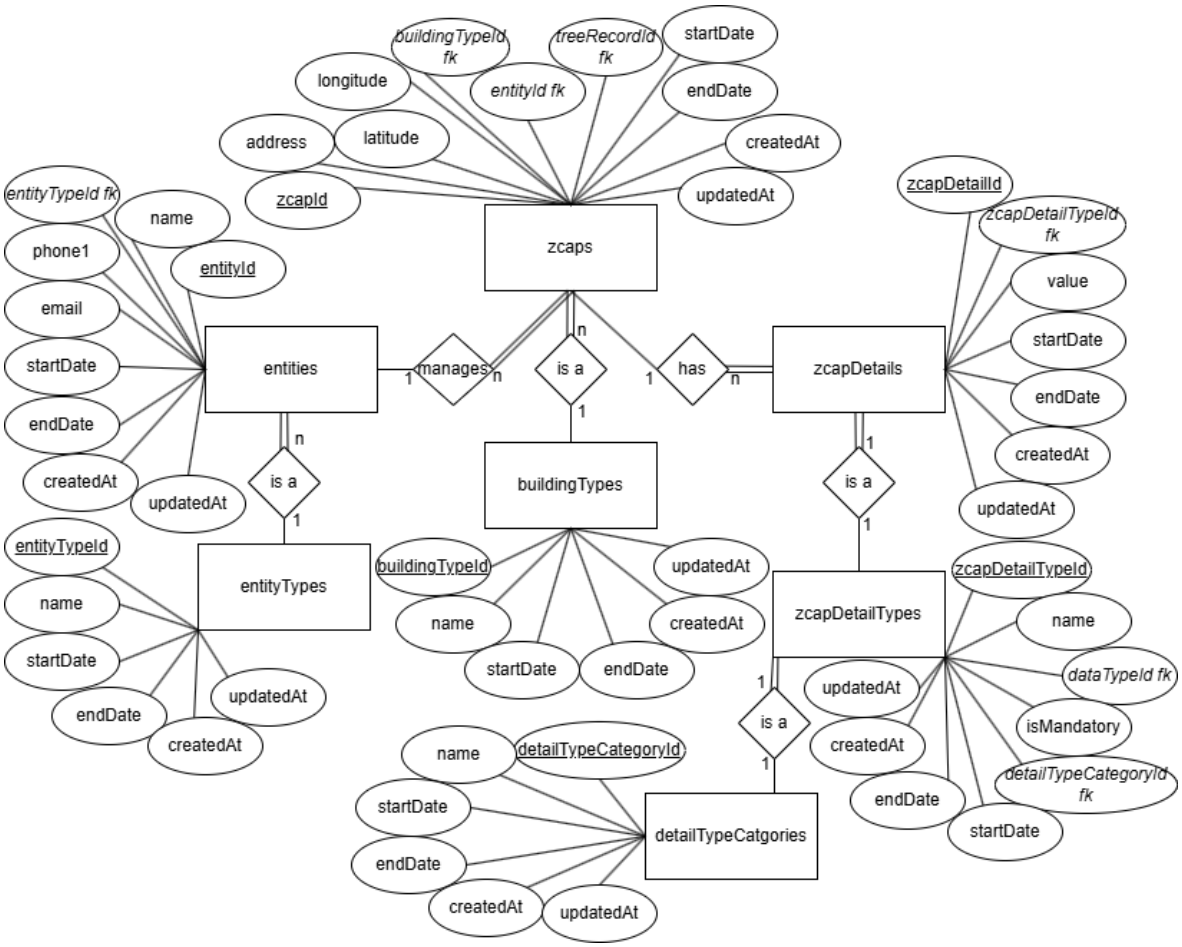

Figura 3.4: Modelo de Dados das ZCAPs.

3.2.5 Incidentes

O modelo dos incidentes serve para armazenar informações sobre todos os acidentes graves ou catástrofes que existem.

O sistema só aceita os tipos de incidentes armazenados na tabela "incidentTypes", representada na Figura 3.5. Este armazena o nome de todos os tipos de incidentes existentes no sistema (Terramoto, Pandemia, etc.).

A tabela "incidents"armazena cada incidente existente. Este armazena o tipo de incidente e o período de ativação.

A tabela "incidentZcaps"armazena a associação de um incidente com uma ZCAP(Subseção 3.2.4) e a entidade responsável(Subseção 3.2.4). Isto permite saber qual ou quais incidentes originaram a abertura da respetiva ZCAP.

Na tabela "incidentZcapPersons"é associada uma entrada da tabela "incidentZcaps"com uma pessoa. Assim, facilmente conseguimos obter informação de qual ZCAP em que a pessoa está e o porquê dessa ZCAP estar ativa (incidente).

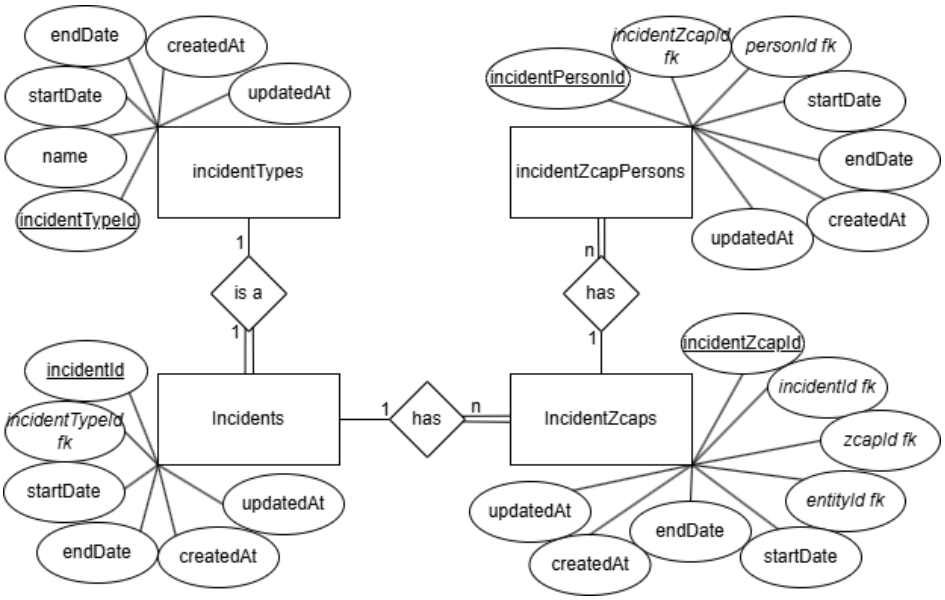

Figura 3.5: Modelo de Dados das ZCAPs.

3.3 Extensibilidade e Escabilidade

O modelo de dados foi concebido com foco na extensibilidade e escalabilidade, de forma a garantir a sua adaptabilidade a novos requisitos e a sua capacidade de suportar volumes crescentes de informação sem degradação significativa de desempenho.

Em termos de extensibilidade, a estrutura foi organizada de forma modular, com separação clara entre diferentes domínios de dados, como utilizadores, zonas territoriais (árvore hierárquica), ZCAPs, pessoas e incidentes. Esta organização facilita a introdução de novas entidades, atributos ou relações, sem a necessidade de alterações disruptivas nas tabelas existentes. Por exemplo, a inclusão futura de novos tipos de incidentes, edifícios, utilizadores, relações ou níveis hierárquicos poderá ser efetuada com alterações mínimas ao modelo atual.

A utilização de chaves estrangeiras bem definidas e de relações normalizadas até à terceira forma normal contribui também para a robustez do modelo face a alterações e ampliações, promovendo a consistência dos dados e a facilidade de manutenção.

Quanto à escalabilidade, o modelo foi implementado sobre um sistema de gestão de base de dados relacional (Microsoft SQL Server), que oferece mecanismos eficientes de indexação, consulta e gestão de dados em larga escala. Foram consideradas práticas como a utilização de índices em campos de pesquisa frequente e a separação lógica de responsabilidades entre tabelas, o que permite otimizar o desempenho em cenários com elevado volume de dados ou múltiplos utilizadores em simultâneo.

Capítulo 4

Frontend

4.1 Introdução

Para o *frontend* designadamente a aplicação para Windows Desktop a desenvolver foi selecionada a *framework* **Flutter** pelas suas características de desempenho e capacidade de suportar multi-plataformas a partir de um único código-base.

4.2 Estrutura do Projeto

O projeto segue uma organização modular e escalável, baseada numa estrutura por *features*, que promove a separação de responsabilidades e facilita a manutenção e evolução da aplicação. A estrutura geral de pastas é a seguinte:

```
assets/
  images/
core/
  constants/
  services/
  utils/
features/
  home/
    model/
    screen/
  login/
  settings/
    entityTypes/
    entities/
    treeLevels/
    tree/
    users/
```

```
    ...
shared/
widgets/
```

Onde as pastas refletem a separação de responsabilidades com o seguinte objetivo:

- `core/`: contém elementos transversais à aplicação, como definições globais (*constants*), modelos de dados genéricos (*models*), serviços de acesso a dados e sincronização (*services*) e utilitários (*utils*).
- `features/`: organiza o código por funcionalidade (*home*, *login*, *settings*, ...), cada uma com o seu próprio modelo de dados *(model/)* e ecrãs *(screen/)*.
- `shared/`: contém elementos partilhados entre diferentes funcionalidades.
- `widgets/`: contém elementos de interface reutilizáveis, como botões, *date pickers*, *list views*, *snack bars*, etc por forma a manter o aspecto e funcionalidade consistente ao longo das diversas funcionalidades.

4.2.1 Dependências Principais

A aplicação recorre a várias dependências para suportar funcionalidades específicas, nomeadamente:

- `window_size` - usado para gerir o tamanho e posicionamento da janela na versão *desktop*. Em dispositivos móveis o tamanho da janela não é adaptável e não causa problemas de posicionalmento dos componentes mas nas versões Desktop podem surgir comportamentos indesejados pelo que se limitou a dimensão minima da janela.
- `http` - usado para realizar as chamadas HTTP à API de backend.
- `connectivity_plus` - permite a utilização de funcoes de verificação de conectividade da rede.
- `crypto` - disponibiliza funções de criptografia (encriptação de password SHA-256).
- `jwt_decoder` - contém as funcionalidades necessárias para descodificar e validar os tokens JWT recebidos da API.
- `shared_preferences` - para armazenar dados simples de forma persistente (ex. dimensão da janela na última utilização ou o darkMode).
- `intl` - fornece suporte à formatação de datas, horas e outros elementos de internacionalização.
- `isar` e `isar_flutter_libs` - base de dados nosql para a persistência de dados offline.

- `path_provider` - para aceder a diretórios do sistema de ficheiros.
- `dropdown_search` - permite criar campos de lista de seleção avançados.
- `cupertino_icons` - disponibiliza os ícones compatíveis com o estilo iOS.
- `provider` - para a gestão de estado reativa da aplicação.

4.2.2 Desenvolvimento

O processo de desenvolvimento é coadjuvado por ferramentas como o `build_runner` e o `isar_generator`, principalmente para a geração do código associado aos modelos e esquemas de dados da base de dados Isar.

4.2.3 Modo Offline-First e persistencia de Dados

Um dos objetivos principais da aplicação é a capacidade de operar na forma **offline-first**, permitindo aos utilizadores aceder e registar informações mesmo na ausência de ligação ao *backend*. Para que isso aconteça, os dados são armazenados localmente utilizando a base de dados `Isar`, e posteriormente sincronizados com o servidor através da API (se e quando esta estiver disponível), garantindo a integridade e a consistência dos dados.

Capítulo 5

Backend

O Backend consiste em desenvolver uma Web API com framework Spring Boot [4] utilizando a linguagem Kotlin [5], uma vez que é a linguagem mais conhecida pelos intervenientes.

No mapeamento das entidades para uma base de dados relacional, é usado Jakarta JPA [6].

Na ajuda do armazenamento e gestão na base de dados, é utilizado Spring Data JPA [7], que fornece uma camada de abstração por cima do JPA, facilitando muito o desenvolvimento.

Para serviços de autenticação e autorização, é usado JWT (JSON Web Token) [8], por ser um formato compacto e seguro de *tokens*.

Neste capítulo, serão abordados temas como estrutura e organização, explicando cada parte da estrutura, os *endpoints*, assim como as respetivas estruturas de resposta, e a documentação, explicando ferramentas usadas.

5.1 Estrutura e Organização

A arquitetura do sistema foi dividida em camadas bem definidas: Controladores (Controllers), Serviços (Services), Repositórios (Repositories) e Entidades (Entities). A estrutura é a seguinte:

```
api/
    exceptions/
    login/
    incidents/
    persons/
    supportTables/
    ...
/domain
    incidents/
    persons/
    ...
```

```
repository/
    dto/
    models/
services/
    incidents/
    persons/
    ...
```

- `domain/` - contém os elementos que representam o modelo de dados mapeado para a base de dados relacional.

- `repository/` - contém os modelos de entrada e saída de dados (dto/) e os modelos JPA de cada tabela da base de dados.

- `services/` - contém a implementação da lógica do sistema.

- `api/` - contém os *handlers* de exceções da API (exceptions/) e os *controllers* (login/, incidents/, persons/, ...) que expõe todos os *endpoints*, que são abordados na secção "Endpoints".

5.1.1 Entidades

A diretoria domain/ define o modelo de dados da aplicação. Esta diretoria está também subdividida por áreas funcionais, como incidents/, persons/, entre outros, que incluem todos os arquivos de cada entidade presente no modelo de dados.

Nas diretorias por baixo da domain/, encontram-se as entidades JPA que serão mapeadas para entidades da base de dados relacional. Para facilitar este mapeamento, o Jakarta JPA [6] fornece anotações para mapear classes (@Entity, @Table, etc.), sabendo assim que classe será mapeada para o modelo relacional, e para os mapeamentos de atributos (@Column, @JoinColumn, etc.), relações (@OneToMany, @ManyToOne, etc.) e chaves primárias (@Id), ilustrado na Figura 5.1.

```kotlin
@Entity
@Table(name = "users")
data class User (
  @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
  val userId: Long = 0,
  val userName: String = "",
  val name: String = "",
  val password: String = "",

  @ManyToOne
  @JoinColumn(name = "userProfileId")
  val userProfile: UserProfile = UserProfile(),

  @ManyToOne
  @JoinColumn(name = "userDataProfileId")
  val userDataProfile: UserDataProfile = UserDataProfile(),

  val startDate: LocalDate = LocalDate.now(),
  val endDate: LocalDate? = null,
  val createdAt: LocalDateTime = LocalDateTime.now(),
  val updatedAt: LocalDateTime = LocalDateTime.now(),
)
```

Figura 5.1: Exemplo da entidade relacional dos utilizadores.

5.1.2 DTOs (Data Transfer Objects) e Repositórios

Na arquitetura do *backend*, foi adotado o padrão de Data Transfer Object (DTO) para promover uma separação clara entre o modelo de dados interno da aplicação (entidades JPA) e os dados expostos pela API ou recebidos a partir do *frontend*. Um exemplo de um DTO exposto pela API (Figura 5.2a) e um DTO para dados recebidos (Figura 5.2b) estão ilustrados na Figura 5.2

Os DTOs ajudam a impedir que campos sensíveis (por exemplo, *password* encriptada) ou que dados internos sejam acidentalmente expostos, permitir regras de validação nos dados recebidos, antes de serem convertidos para entidades persistentes e permitir modelos distintos de entrada e saída de dados, adaptados às necessidades específicas de cada *endpoint*.

```kotlin
data class UserOutputModel (
  val userId: Long,
  val userName: String,
  val name: String,

  val iserProfile: UserProfile,
  val userDataProfiel: UserDataProfile,

  val startDate: LocalDate,
  val endDate: LocalDate,
  val createdAt: LocalDateTime,
  val updatedAt: LocalDateTime,
)
```

(a) DTO de saída do utilizador

```kotlin
data class UserInputModel (
  val userName: String,
  val name: String,
  val password: String,
  val startDate: LocalDate,
  val endDate: LocalDate,
)
```

(b) DTO de entrada do utilizador

Figura 5.2: Exemplos dos DTOs dos utilizadores

Quanto ao modelo de repositório foi usado o JPA em conjunto com Spring Data JPA para realizar o mapeamento objeto-relacional (ORM) entre as entidades JPA e as entidades relacionais. As entidades JPA são classes Kotlin que utilizam anotação Jakarta JPA, referidas na sub-secção anterior, que representam tabelas da base de dados. Na diretoria repository/models/, cada entidade possui uma interface repositório que estende JpaRepository, como está ilustrado na Figura 5.3. Este tipo que a interface estende permite aceder a métodos como findAll(), save(), deleteById(), entre outros, sem necessidade de implementar consultas SQL manualmente, reduzindo significativamente o código repetitivo.

Ainda foi discutido o uso da interface Entitymanager para implementar consultas SQL manualmente, para ter mais controlo sobre as transações, mas a biblioteca do Spring Data JPA acabou por ser a escolha óbvia pelo pouco código e muito expressivo, pelo suporte a consultas personalizadas, paginação e pela integração automática com transações e facilita testes e manutenção.

```kotlin
@Repository
@Transactional
interface UserRepository : JpaRepository<User, Long>{
  fun findByUserName(userName: String): List<User>
  @Query(
    """
  SELECT u FROM User u
  WHERE u.startDate <= :targetDate
   AND (u.endDate IS NULL OR u.endDate >= :targetDate)
    """
  )
  fun findValidOnDate(@Param("targetDate") targetDate: LocalDate): List<User>
}
```

Figura 5.3: Exemplo do Repositório dos utilizadores.

5.1.3 Serviços

A diretoria services/ implementa a lógica da aplicação. Cada serviço (classe com anotação genérica @Component, mas poderia ser anotado com @Service) comunica com os repositórios para aceder a dados e com os DTOs para tratar os dados de entrada/saída, que são enviados pelos controladores. Aqui é verificado se os dados foram enviados corretamente pelo utilizador ou se as entidades pedidas existem.

5.1.4 Controladores

Os controladores constituem a camada da aplicação responsável por expor os Endpoints da Web API REST, permitindo a comunicação entre os clientes (*frontend*) e a lógica da aplicação de *backend*. Cada controlador está organizado por domínio funcional(*incidents*, *persons*, etc.) refletindo as principais entidades e operações da aplicação de gestão de Zonas de Concentração e Apoio à População (ZCAP). Cada classe é anotada com @RestController e @RequestMapping, e os métodos usam anotações como @GetMapping, @PostMapping, @PutMapping e @DeleteMapping para mapear pedidos HTTP aos métodos correspondentes. Esta abordagem, baseada em convenções RESTful, proporciona uma interface clara, estruturada e facilmente utilizável tanto por aplicações cliente como por ferramentas de documentação automática como o Swagger UI (Seção 5.4).

5.2 *Endpoints*

A API RESTful desenvolvida para esta aplicação define um conjunto de *endpoints* organizados por domínio funcional, permitindo operações de consulta, criação e atualização, sendo eliminação, apenas para algumas exceções, uma vez que o objetivo é reter todo o histórico, em caso de consulta futura. Estas operações são sobre os principais recursos da aplicação: utilizadores, zonas de concentração e apoio à população (ZCAPs), estrutura hierárquica, pessoas, incidentes e *login/logout*.

As seguintes figuras ilustram a organização dos *Endpoints* (Figura 5.4) e um exemplo mais específico(Figura 5.5).

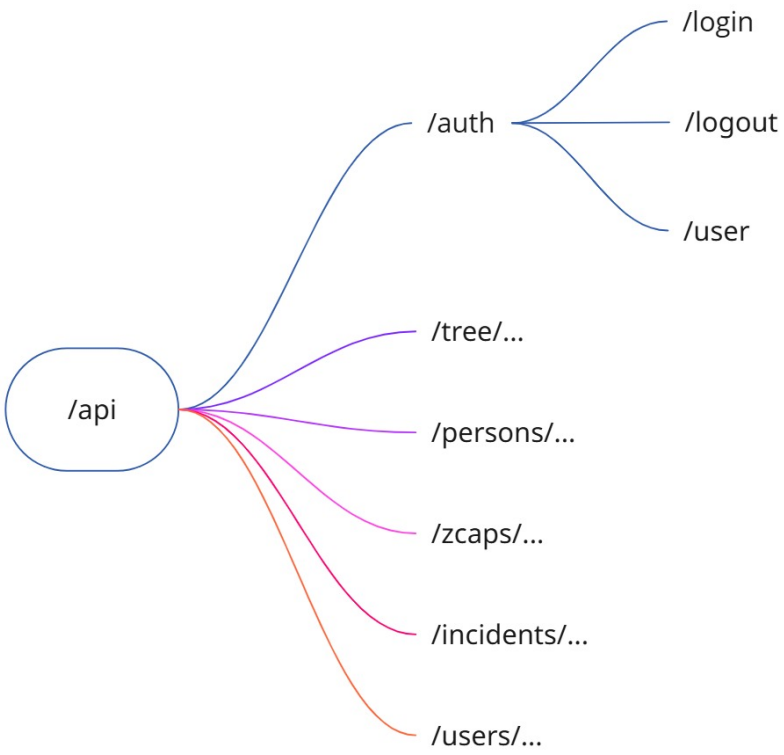

Figura 5.4: Organização dos *Endpoints*.

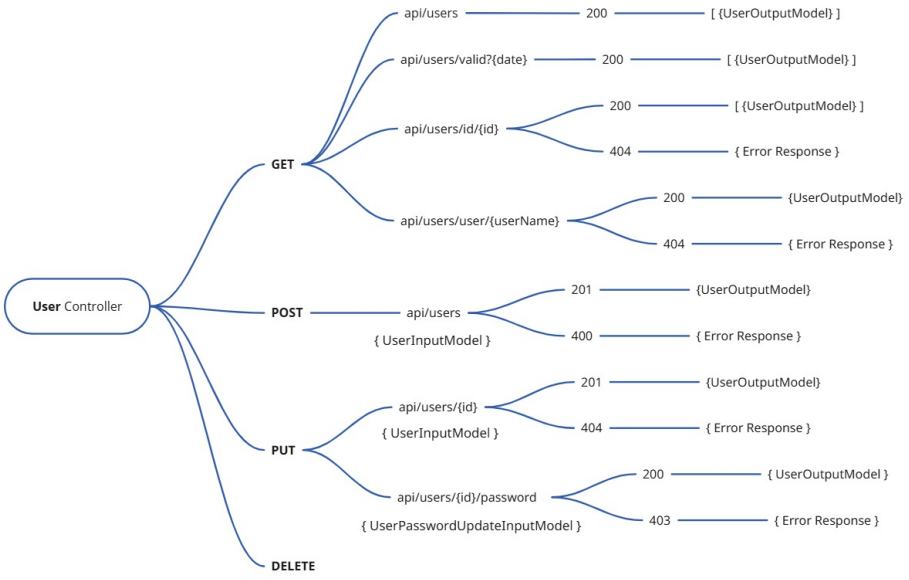

Figura 5.5: Exemplo do controlador dos utilizadores.

Apenas os *Endpoints* de /auth são públicos. Para o resto dos *Endpoints* é utilizado JWT para autenticação e autorização, uma vez que é *Stateless*, ou seja, não requer sessão no servidor, compacto e transportável, fácil de verificar autenticidade com chave secreta e pode

conter dados úteis, como informações sobre o utilizador.

5.3 Estrutura de Respostas

As respostas seguem o formato JSON, com campos bem definidos, mensagens de erro padronizadas e códigos de estado HTTP apropriados (200 OK, 201 Created, 400 *Bad Request*, 401 *Unauthorized*, etc.), facilitando o tratamento no lado do cliente.

5.3.1 Respostas de Sucesso

Em caso de sucesso, apenas será retornado o DTO do elemento pedido ou a lista desses mesmos DTO em formato JSON. Geralmente, tem o seguinte formato:

```
{
    id: 1,
    name: Exemplo1,
    startDate: "2025-01-01",
    endDate: "2027-01-01",
    createdAt: "2025-01-01T12:12:12.12",
    updatedAt: "2025-01-01T12:12:12.12",
}
```

As respostas possíveis em caso de sucesso são:

- `200 Ok` - quando é retornado o/s elemento/s pedidos.
- `201 Created` - quando um elemento é armazenado com sucesso, retornando o próprio elemento.
- `204 No Content` - quando um elemento é eliminado com sucesso.

5.3.2 Respostas de erro

Em caso de erro é lançado uma exceção tratada pelo respetivo *handler*, retornando um objeto genérico designado "ErrorMessage"como está no exemplo:

```
{
    errorCode: "ENTITY_NOT_FOUND",
    errorMessage: "The entity with the given ID was not found.",
    details: "Person with ID 99 not found."
}
```

As respostas possíveis em caso de erro são:

- `400 Bad Request` - quando os dados de entrada estão no formato errado.

- `401 Unauthorized` - quando o utilizador não tem as credencias corretas para o uso dos *endpoints*.
- `404 Not Found` - quando um elemento não é encontrado.
- `409 Conflict` - quando um elemento já existe na base de dados.
- `500 Internal Server Error` - quando algum erro é retornado da base de dados.

5.4 Documentação

Para documentação, é utilizado *Swagger* [9], mais especificamente Swagger UI, que fornece uma interface web interativa que permite aos utilizadores visualizar, testar e explorar *Endpoints* de uma API REST de forma intuitiva, diretamente no navegador. Utiliza a especificação OpenAPI [9] como base e gera automaticamente uma documentação dinâmica, onde é possível:

- Visualizar todos os *endpoints* da API com os respetivos métodos (GET, POST, etc.).
- Consultar os parâmetros de entrada, cabeçalhos, corpo da requisição e respostas esperadas.
- Executar chamadas diretamente pela interface (útil para testes e *debug*).
- Ver mensagens de erro, *status codes* e exemplos de *request/response*.

Capítulo 6

Conclusão

6.1 Trabalho desenvolvido

Ao longo deste projeto, foi desenvolvida uma aplicação para gestão de Zonas de Concentração e Apoio à População (ZCAP) em cenários de emergência, com o objetivo de facilitar a coordenação e o apoio a populações deslocadas. A solução é composta por um *backend* desenvolvido em Kotlin com Spring Boot, que expõe uma Web API RESTful, e um *frontend* desenvolvido com Flutter, compatível com dispositivos móveis e ambientes *desktop*. A persistência local foi assegurada através da utilização da base de dados Isar, enquanto o *backend* utiliza um modelo relacional com JPA. Foram ainda integrados mecanismos de autenticação baseados em JWT e uma documentação interativa da API via Swagger/OpenAPI.

6.2 Observações e conclusões

Ao longo do desenvolvimento do projeto, foi possível criar uma solução funcional e adaptável para a gestão de Zonas de Concentração e Apoio à População (ZCAP), com foco na operação em contextos de emergência. A arquitetura modular permite a separação clara entre o *frontend* e o *backend*, facilitando o desenvolvimento e os testes de forma independente.

Foram ultrapassados diversos desafios técnicos, nomeadamente a sincronização de dados entre o *frontend* e o *backend* em cenários de conectividade intermitente, a gestão da persistência local com Isar e a configuração segura do sistema de autenticação baseada em JWT.

A integração da documentação interativa via Swagger/OpenAPI revelou-se útil tanto para testes como para futuras manutenções ou integrações externas. Globalmente, o projeto contribui com uma base sólida para sistemas de apoio à população em situações críticas, com potencial para evoluir e ser adaptado a diferentes contextos operacionais.

6.3 Trabalho futuro

Até à data limite é expectável alcançar os restantes objetivos:

- Criação e gestão de pessoas desalojadas(em duas fases). Uma primeira fase para um registo rápido e breve sobre cada pessoa, priorizando a segurança das mesmas, e uma segunda fase para obter informações mais detalhadas sobre cada pessoa.
- Utilizadores com papéis diferentes para diferentes níveis hierárquicos na organização da ZCAP. Isto é, cada utilizador tem acesso a diferentes funcionalidades, dependendo da sua função.
- Criação e gestão de todas as estruturas base do sistema.

Referências

[1] Autoridade Nacional de Emergência e Proteção Civil (ANEPC). Caderno técnico prociv 32 – instalação e gestão de zonas de concentração e apoio à população (zcap). https://prociv.gov.pt/media/uqtisk2t/ct_prociv32_zcap.pdf, 2025. Acesso em Fevereiro de 2025.

[2] Autoridade Nacional de Emergência e Proteção Civil (ANEPC). Sigo - sistema integrado de gestão de operações. https://app.sigo-online.pt, 2025. Acesso em Fevereiro de 2025.

[3] Presidência do Conselho de Ministros. Despacho n.º 4067/2024. https://diariodarepublica.pt/dr/detalhe/despacho/4067-2024-860479489, 2024. Publicado no Diário da República, 2.ª série, n.º 86, de 2 de Maio de 2024. Acesso em Fevereiro de 2025.

[4] Spring Team. Spring boot reference documentation. https://docs.spring.io/spring-boot/docs/current/reference/html/, 2025. Acesso em: maio de 2025.

[5] JetBrains. Kotlin language documentation. https://kotlinlang.org/docs/home.html, 2025. Acesso em: maio de 2025.

[6] Jakarta EE. Jakarta persistence api specification. https://jakarta.ee/specifications/persistence/, 2025. Acesso em: maio de 2025.

[7] Spring Team. Spring data jpa documentation. https://docs.spring.io/spring-data/jpa/docs/current/reference/html/, 2025. Acesso em: maio de 2025.

[8] Baeldung. Guide to jwt with spring security. https://www.baeldung.com/spring-security-oauth-jwt, 2025. Acesso em: maio de 2025.

[9] OpenAPI Initiative. Openapi specification. https://swagger.io/specification/, 2025. Acesso em: maio de 2025.