Índice:

Curso:

- **Engenharia Informática**

Elementos do Grupo:

- **Bernardo Carvalho - 20231441**
- **Thiago Moreirão - 20221437**
- **David Bação - 20230331**

Repositório no GitHub: https://github.com/ThZedd/Zoopolis

Professores

Programação Mobile

Nathan Campos

Projeto Desenvolvimento Móvel

Pedro Rosa

Programação Orientada por Objetos

Fabio Guilherme

Bases de Dados

Miguel Boavida

Competências Comunicacionais

Luiza Bianquini Campos

Matemática Discreta

Paulo Velho

Relatório - Projeto Mobile - Zoopolis

- **Zoopolis** é uma aplicação móvel concebida com o intuito de melhorar a experiência dos visitantes ao zoológico, através do fornecimento de um meio interativo e educativo para explorar as instalações. O seu objetivo é resolver a **falta de informações acessíveis** (por exemplo: a localização dos caixotes do lixo) durante a visita, oferecendo um guia digital, localização em tempo real, exibição de preços, recursos educativos e a **possibilidade de acumular pontos**, de modo a maximizar o aproveitamento da visita.

Palavras-Chave

- Aplicação Móvel, zoológico, interação, educação, turismo, colheita de pontos, guia digital, localização em tempo real.

Objetivos e Motivação

Motivação:

- Com a crescente digitalização dos espaços culturais e educativos, os zoológicos têm a oportunidade de melhorar a experiência dos seus visitantes. A nossa aplicação **Zoopolis** pretende tornar essas visitas mais cativantes e informativas, principalmente para o público jovem, através da combinação de entretenimento e educação.

Objetivos:

- Oferecer uma experiência cativante e interativa, com a colheita de pontos, um guia digital e localização em tempo real.
- Incentivar as visitas recorrentes ao zoológico e o aprendizado sobre as espécies através de um guia digital.
- Facilitar a visita ao zoo com uma interface intuitiva e de fácil acesso.

Público-Alvo

- **Estudantes** interessados na vida animal;
- **Turistas** que visitam o zoológico pela primeira vez;
- **Famílias** com crianças em idade escolar;

- **Amantes de animais** que desejam de aprender mais acerca de animais.

Aplicações semelhantes

Após uma pesquisa acerca de aplicações disponíveis no mercado, deparamo-nos com várias apps para zoológicos, tais como:

1. **ZSL London Zoo App:** Aplicação de guia e mapa interativo com informações sobre os animais e as atrações do zoológico de Londres.
2. **San Diego Zoo:** Oferece informações detalhadas sobre os animais e eventos, além de notificações em tempo real.
3. **Bronx Zoo App:** Além de funcionar como guia interativo, proporciona uma experiência de navegação com mapas detalhados, informações sobre os animais e suporte para planejar visitas, incluindo horários de alimentação e eventos especiais.

Estas aplicações oferecem várias funções básicas, tais como, mapas e guias, mas a Zoopolis pretende diferenciar-se com a colheita de pontos.

Guiões de Teste

Caso de Utilização Principal: Visita Guiada (Core)

1. O utilizador faz login ou cria uma conta;
2. Acede ao menu dos animais e **seleciona o animal** que deseja visitar;
3. Recebe informações detalhadas e curiosidades acerca do animal, e o **trajeto necessário** a efetuar até chegar a esse animal;
4. Após chegar ao recinto desse animal irá se deparar com uma placa com uma foto do animal, algumas informações sobre o mesmo e um código, que futuramente poderá scannear;
5. Ao scannear esse introduzir o código desse animal aparece durante aquela visita como **"Visitado"** e o utilizador **ganha 1 ponto**;
6. Esses pontos podem ser acumulados, e ao fim de juntar um determinado número de pontos poderá levantar num dos kiosks um brinde.

Casos de Utilização Secundários:

- Compra de Bilhetes:

1. O utilizador faz login ou cria uma conta;
2. Acede ao menu dos preços e **seleciona a opção** de "Buy Tickets";
3. Seleciona o tipo de bilhete e o número de entradas;
4. Conclui a compra através de um **pagamento seguro**.

- Pesquisa Informativa:

1. O utilizador pode selecionar a opção de **entrar como convidado**;
2. Seleciona o **menu das atividades;
3. Pesquisa sobre a **atividade que deseja**;
4. Seleciona e obterá informações, como, o horário e uma pequena descrição sobre a mesma.

Descrição da solução

1. Descrição Genérica:

- A solução será criar uma **aplicação móvel** que oferece uma **experiência interativa, educativa e divertida** para os visitantes do zoológico. Inclui funcionalidades como **mapa interativo**, **colheita de pontos**, **compra de bilhetes**, **informação e curiosidades** acerca dos animais do zoológico e **localização em tempo real**.

2. Enquadramento nas Unidades Curriculares:

- **Programação Mobile:** Desenvolvimento da aplicação móvel através da utilização da aplicação **Android Studio**.
- **Programação Orientada por Objetos:** Interligação da **base de dados** com a app através da utilização do **Spring Boot**.
- **Base de dados:** Armazenamento das **informações dos animais, utilizadores e dos pontos coletados**.
- **Competências Comunicacionais:** Comunicação eficaz na **propaganda e divulgação** do produto em desenvolvimento.

- **Matemática Discreta:** Utilizamos a lógica dada em matemática discreta, de forma a conseguirmos obter dados úteis para analisar e conseguir desenhar os trajetos mais eficientes possíveis, foi utilizada principalmente na recolha destes dados, estando esta

lógica disponível para visualizar na REST API (Mostrar a subárea mais visitada).

3. Requisitos Técnicos:

- **Linguagens de Programação**: **Kotlin**, **Java**, **MySQL**.
- **Plataforma de Desenvolvimento**: Android Studio
- **Base de Dados:** MySQL Workbench.
- **API:** Spring Boot.

4. Arquitetura da Solução:

- **Frontend:** Desenvolvimento da aplicação com Android Studio.
- **Backend:** Utilização de Spring Boot para lidar com as interações entre a base de dados e a aplicação.
- **Base de dados:** Utilização de MySQL Workbench para criar a DB.

5. Tecnologias a utilizar:

- **Frontend:** Kotlin.
- **Backend:** Java.
- **Base de dados:** MySQL.

Project Charter

1. General Project Information

- **Charter Date**: 18 October 2024
- **Project Name**: Zoopolis
- **Project Managers**: Thiago Moreirão, David Bação, Bernardo Carvalho
- **Expected Start Date**: 10 October 2024
- **Expected Completion Date**: 12 January 2025

2. Project Details

- **Zoopolis** is a mobile application designed to improve the experience of zoo visitors by providing an interactive and educational means to explore the facilities.The reason we chose the Zoo is for people to have greater contact with nature, since nowadays,

technology has given us such comfort that we forget how good nature is for us and what it provides us.

3. Key Requirements

1. **Database:** MySQL for data storage, connected to the backend via REST API.
2. **UI/UX Design:** User interface designed in Figma, following Material Design guidelines for a fluid and modern experience.
3. **Mobile Programming:** Developed in Kotlin using the Android SDK, with Retrofit for server communication.
4. **Backend Programming:** Backend developed in Java, with RESTful APIs to manage data and communicate with the database.
5. **Platform:** Native Android app, compatible with devices running Android 5.0 (Lollipop) or higher.

4. Expected Benefits

1. **Improved Visitor Experience:** The application offers interactive features such as a map and information about animals, providing a more educational and engaging visit.
2. **Simplified Access:** Purchasing tickets and planning visits through the app makes the experience more convenient, avoiding queues and offering real-time information.
3. **Increased Engagement:** Multimedia and interactivity features increase visitor engagement, encouraging more frequent visits and greater connection with the zoo.
4. **Ease of Management:** The zoo can monitor visitation patterns, send push notifications and reduce operational costs by digitizing most of its operations.
5. **Sustainability:** Digitizing information reduces the use of paper and helps preserve the environment, aligning with the zoo's conservation objectives.

5. Estimated Costs & Resources

- **Estimated Costs**: $3000
- **Resources**: $450

6. Estimated Milestones

1. **UI/UX Design** - November 20
2. **Database** - December 13
3. **Mobile Programming** - December 20
4. **Backend Programming** - December 29

7. Project Team

- Developers:
  - Bernardo Carvalho;
  - Thiago Moreirão;
  - David Bação;

8. Stakeholders

- **European University - IADE**

9. Overall Project Risk

- **Risks:**
  - Due to the limited time to complete the project, there is a risk that we will not be able to implement all the desired functionalities.
  - Users' lack of familiarity with the new system can result in low use and dissemination, compromising the success of the project.
- **Mitigations:**
  - Conduct development and brainstorming sessions with the team to prioritize essential features and ensure better project planning.
  - Collect feedback from a group of beta users during the testing phase to identify areas for improvement and promote acceptance of the system.

10. Project Success Criteria

- If all the features are working and public acceptance is favorable, we thought about communicating with the Lisbon Zoo to find a possible partnership, where we would publicize our work and implement it in a real situation.

MockUps:

A aplicação utilizada foi o Figma:

- https://www.figma.com/proto/HtLBeDXc9heSYg3r913Lhk/Projeto?node-id=0-1&t=GbXDCTrreYVTK9H9-1

Planeamento (Gráfico de Gantt):

Utilizamos o site recomendado para a realização do Gráfico de Gantt:

- https://sharing.clickup.com/9012393636/g/8cjwdn4-372/gantt
- https://sharing.clickup.com/9012393636/l/8cjwdn4-332/list

Conclusão:

- O nosso projeto **Zoopolis** tem como objetivo revolucionar as visitas ao zoológico e a forma como as pessoas interagem com o zoo, utilizando tecnologias modernas como **a localização em tempo real**. Queremos criar uma **experiência educativa e cativante**, contribuindo para a preservação da vida selvagem, sensibilizando os visitantes. Ao longo do desenvolvimento da aplicação, **estaremos abertos a críticas construtivas**, de forma a atender às necessidades dos nossos utilizadores, garantindo assim que a nossa aplicação seja **intuitiva, funcional e prática**.

Bibliografia:

- ZSL London Zoo - London Zoo
- Bronx Zoo
- San Diego Zoo App - San Diego Zoo
- Jardim Zoológico de Lisboa
- Figma - Figma, Inc.
- ClickUp
- Kotlin - Jetbrains
- Jetpack Compose - Google
- Android Studio - Google
- Google Maps API - Google
- Java - Oracle
- Spring Boot - VMware Tanzu
- MySQL - Oracle
- Android SDK 28 - Google

Personas

- **Nome:** Lucas Silva
- **Idade**: 27 anos
- **Profissão**: Desenvolvedor de software
- **Localização**: Mora em um apartamento na cidade, próximo ao zoológico
- **Status Familiar**: Solteiro, mas frequentemente visita com amigos ou sobrinhos

Perfil e Comportamento

Lucas é apaixonado por tecnologia e natureza. Cresceu a assistir documentários sobre animais e sempre gostou de explorar lugares que combinam aprendizado com lazer. Ele utiliza aplicativos para maximizar suas experiências e valoriza recursos tecnológicos como localização em tempo real e interação online. Apesar de visitar o zoológico ocasionalmente, ele está sempre em busca de algo novo, como eventos ou experiências exclusivas.

Objetivos ao usar o app

- Descobrir experiências únicas: eventos, habitats imersivos e exposições interativas.
- Interagir com a tecnologia do zoológico, como quiosques digitais, acumulo de pontos e localização em tempo real.
- Compartilhar momentos no zoológico nas redes sociais.

Frustrações e Desafios

- Falta de inovação em passeios típicos.
- Informações desatualizadas sobre eventos ou atrações fechadas.
- Longas filas ou dificuldades em encontrar o caminho no zoológico.

Motivações

- Explorar e aprender sobre animais de forma dinâmica e tecnológica.
- Usar o app para simplificar a visita e evitar contratempos.
- Criar conexões com a natureza e promover a conservação ambiental.

Nome: Ana Clara

- **Idade**: 35 anos
- **Profissão**: Professora de biologia no secundário

- **Localização**: Cidade grande, a 40 km do zoológico
- **Status Familiar**: Casada, mãe de duas crianças (7 e 10 anos)

Perfil e Comportamento

Ana Clara é apaixonada por natureza e está sempre procurando atividades educativas e divertidas para seus filhos. Gosta de planejar passeios antecipadamente e valoriza recursos que tornam a experiência mais interativa e informativa. Usa tecnologia para organizar suas atividades, como aplicativos e sites, mas prefere interfaces simples e intuitivas.

Objetivos ao usar o app

- **Planejar a visita**: Quer saber horários, preços, mapa do zoológico, atrações e eventos especiais.
- **Educar os filhos**: Busca informações sobre os animais que vão visitar, como curiosidades, habitat natural e hábitos alimentares.
- **Interatividade**: Gostaria de interagir com o zoológico mesmo após a visita, como receber atualizações sobre os animais.

Frustrações e Desafios

- Perder tempo com informações desorganizadas ou difíceis de acessar.
- Falta de clareza sobre o que esperar do passeio (exemplo: eventos lotados ou ausência de atrações específicas no dia).
- Dificuldade em manter as crianças engajadas durante o passeio.

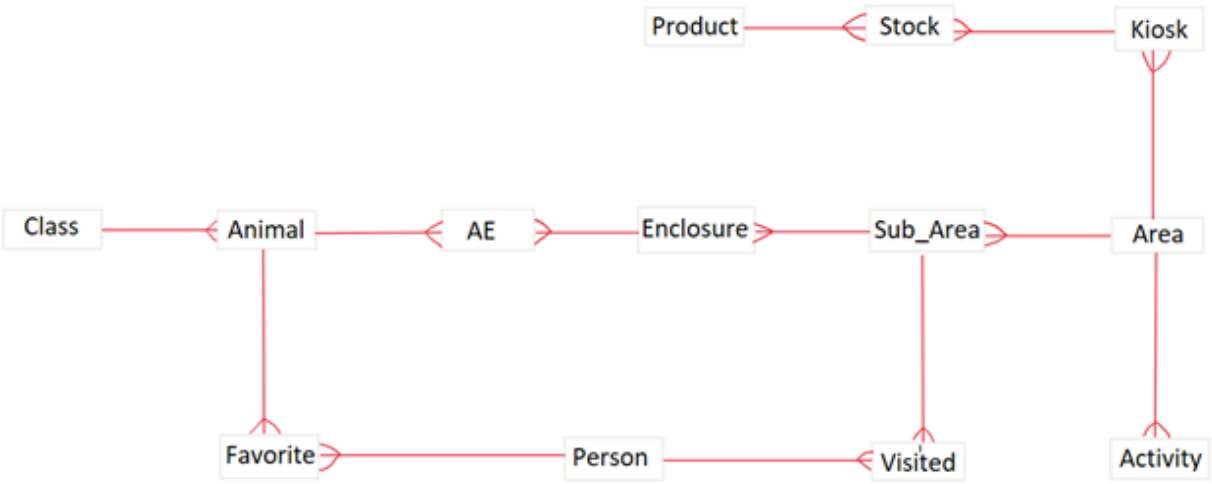
Motivações

- Ensinar aos filhos a importância da conservação da natureza.
- Proporcionar momentos de lazer e conexão familiar.
- Descobrir novidades no zoológico e compartilhar com a comunidade escolar.

Diagrama de Classes

Dicionário de dados

Modelo Entidade-Relação



O relacionamento com a pessoa (Person) é central no modelo. Cada pessoa pode visitar diferentes subáreas do local (Sub Area), registrando suas preferências e comportamentos durante a sua visita. Algumas dessas pessoas podem indicar também um animal favorito (Favorite), estabelecendo um vínculo que ajuda na personalização da experiência e na análise de tendências do interesse do público.
Os animais (Animal) estão organização em diferentes classes (Class), como mamíferos, aves ou répteis, permitindo uma categorização eficiente de acordo com suas características biológicas. Esses animais também estão alojados em recintos específicos (Enclousures), que atendem às necessidades de cada espécime. Cada recindo também está localizado em uma subárea (SubArea), que é uma parte de uma área maior que é o zoológico, permitindo a organização do espaço em setores bem definidos. As áreas (Area) também desempenhas outras funções importantes, pois incluem diferentes tipos de atividades (Activity) realizadas em diversos locais, como passeios, brincadeiras e interações com os animais. Além disso, às áreas contêm quiosques (Kiosks), que oferecem produtos para os visitantes e também conta com o sistema derecompensa por pontos adquiridos na visita. Esses quiosques mantêm um controle de estoque (Stock) de produtos, como alimentos, bebidas e brinquedos. Com esse modelo, não é apenas fácil de organizar como também gerir o zoológico, já que podemos saber todas as informações sobre os locais mais visitados além de gerir bem os produtos e serviços prestados em cada local.

Documentos de referência

Dicionário de dados

Guia de dados

REST API

REST API

Introdução

A API do Zoopolis oferece acesso eficiente a informações sobre animais, os pontos dos usuários e localização dos recintos, permitindo consultas, visualizações e recolhas de informações. Desenvolvida para integração com a aplicação móvel, a API fornece os dados atualizados e é compatível várias outras plataformas.

Os dados são entregues em formato JSON, garantindo respostas consistentes e facilitando a integração com diversos sistemas. A estrutura dos dados e os Endpoints são flexíveis, projetados para suportar futuras atualizações e melhorias contínuas na aplicação.

Endpoints

- Mostrar Animais
    - **URL:**

`/animals`

    - **METHOD:**

`GET`

- **SUCESS RESPONSE:**

```
[
  {

        "id": [integer],
        "name": [string],
        "ciName": [string],
        "description": [string],
        "weight": [float],
        "height": [float],
        "length": [float],
        "classe": {
              "id": [integer],
              "name": [string],
              "order": [string],
```

```
            "family": [string]
        },
    "imageUrl": [string]
  },
]
```

- **ERROR RESPONSE:**

```
{
    "status": 500,
    "message": "An unexpected error occurred.",
    "timestamp": [datetime]
}
```

- **SAMPLE CALL:**

```
@GET("animalsDTO")
suspend fun getAnimals(): List<AnimalDTO>
```

- Mostrar Animais por ID
    - **URL:**

  `/animals/:id`

    - **METHOD:**

  `GET`

    - **URL Parameters:**
        - Required:

      `id: [integer]`
- **SUCESS RESPONSE:**

```
[
  {

    "id": [integer],
    "name": [string],
```

```
        "ciName": [string],
        "description": [string],
        "weight": [float],
        "height": [float],
        "length": [float],
        "classe": {
              "id": [integer],
              "name": [string],
              "order": [string],
              "family": [string]
            },
        "imageUrl": [string]
          },

]
```

- **ERROR RESPONSE:**

```
{

        "status": 404,
        "message": "Animal with id {id} not found.",
        "timestamp": [datetime]

}
```

- **SAMPLE CALL:**

```
@GET("animalsDTO/{id}")
suspend fun getAnimalsById(@Path("id") id: Int): AnimalDTO
```

Listar Usuários

- **URL:**

  `/api/persons`

- **METHOD:**

  `GET`
- **SUCESS RESPONSE:**

```
[
  {
    "id": [integer],
    "name": [string],
    "email": [string],
    "points": [integer]
  }
]
```

- **ERROR RESPONSE:**

```
{
  "status": 500,
  "message": "An unexpected error occurred.",
  "timestamp": [datetime]
}
```

- **SAMPLE CALL:**

```
@Get("persons")
suspend fun getPersons(): List<Person>
```

Obter Usuário por ID

- **URL:**

  `/api/persons/{id}`
- **METHOD:**

  `GET`
- **URL Parameters:**
  - Required:

`id: [integer]`

- **SUCESS RESPONSE:**

```
{
  "id": [integer],
  "name": [string],
  "email": [string],
  "points": [integer]
}
```

- **ERROR RESPONSE:**

```
{
  "status": 404,
  "message": "User not found.",
  "timestamp": [datetime]
}
```

- **SAMPLE CALL:**

```
@Get("persons/{id}")
suspend fun getPersonById(id: Int): Person
```

Registrar Usuário

- **URL:**

  `/api/persons/register`

- **METHOD:**

  `POST`

- **REQUEST BODY:**

```
{
  "name": [string],
  "email": [string],
```

```
  "password": [string]
}
```

- **SUCESS RESPONSE:**

```
{
  "id": [integer],
  "name": [string],
  "email": [string],
  "points": 0
}
```

- **ERROR RESPONSE:**

```
{
  "status": 409,
  "message": "Email already in use.",
  "timestamp": [datetime]
}
```

- **SAMPLE CALL:**

```
@POST("persons/register")
suspend fun register(@Body person: Person): Person
```

Login

- **URL:**

  `/api/persons/login`

- **METHOD:**

  `POST`

- **REQUEST BODY:**

```
{
  "email": [string],
```

```
  "password": [string]
}
```

- **SUCESS RESPONSE:**

```
{
  "id": [integer],
  "token": [string]
}
```

- **ERROR RESPONSE:**

```
{
  "status": 401,
  "message": "Invalid password.",
  "timestamp": [datetime]
}

{
  "status": 404,
  "message": "User not found.",
  "timestamp": [datetime]
}
```

- **SAMPLE CALL:**

```
@POST("persons/login")
suspend fun login(@Body loginRequestDTO: LoginRequestDTO): LoginResponseDTO
```

Validar Token

- **URL:**

  `/api/persons/validate`

- **METHOD:**

  `GET`

- **HEADERS:**
  - Authorization: `Bearer [token]`
- **SUCESS RESPONSE:**

```
"Token is valid"
```

- **ERROR RESPONSE:**

```
{
  "status": 401,
  "message": "Invalid or expired token.",
  "timestamp": [datetime]
}
```

- **SAMPLE CALL:**

```
@Get("persons/validate")
suspend fun validateToken(@Header("Authorization") token: String): String
```

Adicionar Pontuação

- **URL:**

  `/api/persons/{id}/add-point`
- **METHOD:**

  `POST`
- **URL Parameters:**
  - Required:

    `id: [integer]`
- **SUCESS RESPONSE:**

```
{
  "id": [integer],
  "name": [string],
  "email": [string],
```

```
  "points": [integer]
}
```

- **ERROR RESPONSE:**

```
{
  "status": 404,
  "message": "User not found.",
  "timestamp": [datetime]
}
```

- **SAMPLE CALL:**

```
@POST("persons/{id}/add-point")
suspend fun addPointToUser(@Path("id") id: Int): Person
```

Remover Pontuação

- **URL:**

  `/api/persons/{id}/remove-point`
- **METHOD:**

  `POST`
- **URL Parameters:**
  - Required:

    `id: [integer]`
- **SUCESS RESPONSE:**

```
{
  "id": [integer],
  "name": [string],
  "email": [string],
  "points": [integer]
}
```

- **ERROR RESPONSE:**

```
{
  "status": 404,
  "message": "User not found or points are already at 0.",
  "timestamp": [datetime]
}
```

- **SAMPLE CALL:**

```
@POST("persons/{id}/remove-point")
suspend fun removePointToUser(@Path("id") id: Int): Person
```

Listar Visitas

- **URL:**

  `/api/visited`
- **METHOD:**

  `GET`
- **SUCESS RESPONSE:**

```
[
  {
    "id": [integer],
    "person": {
      "id": [integer],
      "name": [string],
      "email": [string]
    },
    "subArea": {
      "id": [integer],
      "name": [string]
    },
    "dtime": [datetime]
```

```
  }
]
```

- **ERROR RESPONSE:**

```
{
  "status": 500,
  "message": "An unexpected error occurred.",
  "timestamp": [datetime]
}
```

- **SAMPLE CALL:**

```
@GET("visited")
suspend fun getAllVisits(): List<Visited>
```

Obter Visita por ID

- **URL:**

  `/api/visited/{id}`
- **METHOD:**

  `GET`
- **URL Parameters:**
  - Required:

    `id: [integer]`
- **SUCESS RESPONSE:**

```
{
  "id": [integer],
  "person": {
    "id": [integer],
    "name": [string],
    "email": [string]
  },
  "subArea": {
```

```
    "id": [integer],
    "name": [string]
  },
  "dtime": [datetime]
}
```

- **ERROR RESPONSE:**

```
{
  "status": 404,
  "message": "Visit not found.",
  "timestamp": [datetime]
}
```

- **SAMPLE CALL:**

```
@GET("visited/{id}")
suspend fun getVisitById(@Path("id") id: Int): Visited
```

Criar Visita

- **URL:**

  `/api/visited`
- **METHOD:**

  `POST`
- **REQUEST BODY:**

```
{
  "personId": [integer],
  "animalId": [integer]
}
```

- **SUCESS RESPONSE:**

```
{
  "id": [integer],
  "person": {
    "id": [integer],
    "name": [string],
    "email": [string]
  },
  "subArea": {
    "id": [integer],
    "name": [string]
  },
  "dtime": [datetime]
}
```

- **ERROR RESPONSE:**

```
{
  "status": 400,
  "message": "Invalid person ID.",
  "timestamp": [datetime]
}
```

- **SAMPLE CALL:**

```
@POST("visited")
suspend fun createVisit(
    @Query("personId") personId: Int,
    @Query("animalId") animalId: Int
): Visited
```

Atualizar Visita

- **URL:**

  `/api/visited/{id}`
- **METHOD:**

`PUT`

- **REQUEST BODY:**

```
{
  "dtime": [datetime],
  "person": {
    "id": [integer]
  },
  "subArea": {
    "id": [integer]
  }
}
```

- **SUCESS RESPONSE:**

```
{
  "id": [integer],
  "person": {
    "id": [integer],
    "name": [string],
    "email": [string]
  },
  "subArea": {
    "id": [integer],
    "name": [string]
  },
  "dtime": [datetime]
}
```

- **ERROR RESPONSE:**

```
{
  "status": 404,
  "message": "Visit not found.",
  "timestamp": [datetime]
}
```

Deletar Visita

- **URL:**

  `/api/visited/{id}`
- **METHOD:**

  `DELETE`
- **URL Parameters:**
  - Required:

    `id: [integer]`
- **SUCESS RESPONSE:**

```
{
  "message": "Visit deleted successfully."
}
```

- **ERROR RESPONSE:**

```
{
  "status": 404,
  "message": "Visit not found.",
  "timestamp": [datetime]
}
```

Obter Sub-área Mais Visitada

- **URL:**

  `/api/visited/most-visited-subarea`
- **METHOD:**

  `GET`
- **SUCESS RESPONSE:**

```
{
  "subArea": [string],
```

```
  "visitCount": [integer]
}
```

- **ERROR RESPONSE:**

```
{
  "status": 500,
  "message": "An unexpected error occurred.",
  "timestamp": [datetime]
}
```

Get All Favorites

- **URL:**
  `/api/favorite`
- **METHOD:**
  `GET`
- **SUCESS RESPONSE:**

```
[
  {
    "id": [integer],
    "personId": [integer],
    "animalId": [integer]
  },
  ...
]
```

- **ERROR RESPONSE:**

```
{
  "status": 500,
  "message": "Internal Server Error.",
  "timestamp": [datetime]
}
```

- **SAMPLE CALL:**

```
@GET("favorite")
suspend fun getFavorites(): List<Favorite>
```

Get Favorite by ID

- **URL:**
  `/api/favorite/{id}`
- **METHOD:**
  `GET`
- **URL Parameters:**
  - Required:
    `id: [integer]`
- **SUCESS RESPONSE:**

```
{
  "id": [integer],
  "personId": [integer],
  "animalId": [integer]
}
```

- **ERROR RESPONSE:**

```
{
  "status": 404,
  "message": "Favorite not found.",
  "timestamp": [datetime]
}
```

- **SAMPLE CALL:**

```
@GET("favorite/{id}")
suspend fun getFavorite(@Path("id") id: Int): Optional<Favorite>
```

Get Favorite Animals by Person

- **URL:**
  `/api/favorite/person/{personId}`
- **METHOD:**
  `GET`
- **URL Parameters:**
  - Required:
    `personId: [integer]`
- **SUCESS RESPONSE:**

```
[
  {
    "id": [integer],
    "name": [string],
    "species": [string]
  },
  ...
]
```

- **ERROR RESPONSE:**

```
{
  "status": 404,
  "message": "No favorite animals found for the person.",
  "timestamp": [datetime]
}
```

- **SAMPLE CALL:**

```
@GET("favorite/person/{personId}")
suspend fun getFavoriteAnimalsByPerson(@Path("personId") personId: Int):
List<AnimalDTO>
```

Check if Animal is Favorite

- **URL:**
  `/api/favorite/isFavorite`
- **METHOD:**
  `GET`
- **QUERY Parameters:**
  - Required:
    `personId: [integer]`
    `animalId: [integer]`
- **SUCESS RESPONSE:**

```
true
```

- **ERROR RESPONSE:**

```
{
  "status": 400,
  "message": "Bad Request.",
  "timestamp": [datetime]
}
```

- **SAMPLE CALL:**

```
@GET("favorite/isFavorite")
suspend fun isFavorite(
    @Query("personId") personId: Int,
    @Query("animalId") animalId: Int
): Boolean
```

Add Favorite

- **URL:**
  `/api/favorite/add`
- **METHOD:**
  `POST`
- **QUERY Parameters:**

    - Required:
      `personId: [integer]`
      `animalId: [integer]`
- **SUCESS RESPONSE:**

```
"Animal added to favorites successfully."
```

- **ERROR RESPONSE:**

```
{
  "status": 409,
  "message": "Animal is already a favorite for this person.",
  "timestamp": [datetime]
}
```

- **SAMPLE CALL:**

```
@POST("favorite/add")
suspend fun addFavorite(
    @Query("personId") personId: Int,
    @Query("animalId") animalId: Int
): String
```

Remove Favorite

- **URL:**
  `/api/favorite/remove`
- **METHOD:**
  `DELETE`
- **QUERY Parameters:**
    - Required:
      `personId: [integer]`
      `animalId: [integer]`
- **SUCESS RESPONSE:**

```
"Animal removed from favorites successfully."
```

- **ERROR RESPONSE:**

```
{
  "status": 404,
  "message": "Animal is not a favorite for this person.",
  "timestamp": [datetime]
}
```

- **SAMPLE CALL:**

```
@DELETE("favorite/remove")
suspend fun removeFavorite(
    @Query("personId") personId: Int,
    @Query("animalId") animalId: Int
): String
```

Retrieve All Enclosures

URL:

`/api/enclosuresDTO`

METHOD:

`GET`

SUCCESS RESPONSE:

```
[
  {
    "id": [integer],
    "name": [string],
    "animalClass": [string],
    "mapsId": [integer],
    "supportedAmount": [integer],
```

```
    "latitude": [float],
    "longitude": [float]
  }
]
```

ERROR RESPONSE:

- **No specific error responses defined for this endpoint.**

SAMPLE CALL:

```
@GET("enclosuresDTO")
suspend fun getAllEnclosures(): List<EnclosureDTO>
```

Retrieve Enclosure by ID

URL:

`/api/enclosuresDTO/{id}`

METHOD:

`GET`

URL Parameters:

- Required:

  `id: [integer]`

SUCCESS RESPONSE:

```
{
  "id": [integer],
  "name": [string],
  "animalClass": [string],
  "mapsId": [integer],
  "supportedAmount": [integer],
  "latitude": [float],
```

```
  "longitude": [float]
}
```

ERROR RESPONSE:

```
{
  "status": 404,
  "message": "Enclosure not found",
  "timestamp": [datetime]
}
```

SAMPLE CALL:

```
@GET("enclosuresDTO/{id}")
suspend fun getEnclosureById(@Path("id") id: Int): EnclosureDTO
```

Obter Todos os Registros AE

- **URL:**

  `/api/ae`
- **METHOD:**

  `GET`
- **SUCCESS RESPONSE:**

```
[
  {
    "id": [integer],
    "dateIn": [string], // Date in ISO format
    "dateOut": [string], // Date in ISO format
    "code": [string],
    "animalDTO": {
      "id": [integer],
      "name": [string],
      "ciName": [string],
      "description": [string],
```

```
      "imageUrl": [string]
    },
    "enclosureDTO": {
      "id": [integer],
      "name": [string],
      "animalClass": [string],
      "mapsId": [string],
      "supportedAmount": [integer],
      "latitude": [double],
      "longitude": [double]
    }
  }
]
```

- **SAMPLE CALL:**

```
@GET("ae")
suspend fun getAllAE(): List<AEDTO>
```

Obter Registro AE por ID

- **URL:**

  `/api/ae/{id}`
- **METHOD:**

  `GET`
- **URL Parameters:**
  - Required:

    `id: [integer]`
- **SUCCESS RESPONSE:**

```
{
  "id": [integer],
  "dateIn": [string], // Date in ISO format
  "dateOut": [string], // Date in ISO format
  "code": [string],
```

```
  "animalDTO": {
    "id": [integer],
    "name": [string],
    "ciName": [string],
    "description": [string],
    "imageUrl": [string]
  },
  "enclosureDTO": {
    "id": [integer],
    "name": [string],
    "animalClass": [string],
    "mapsId": [string],
    "supportedAmount": [integer],
    "latitude": [double],
    "longitude": [double]
  }
}
```

- **ERROR RESPONSE:**

```
{
  "status": 404,
  "message": "AE with id [id] not found",
  "timestamp": [datetime]
}
```

- **SAMPLE CALL:**

```
@GET("ae/{id}")
suspend fun getAEById(@Path("id") id: Int): AEDTO
```

Obter Registros AE por ID do Animal

- **URL:**

  `/api/ae/animal/{animalId}`
- **METHOD:**

`GET`

- **URL Parameters:**
  - Required:

    `animalId: [integer]`
- **SUCCESS RESPONSE:**

```
[
  {
    "id": [integer],
    "dateIn": [string], // Date in ISO format
    "dateOut": [string], // Date in ISO format
    "code": [string],
    "animalDTO": {
      "id": [integer],
      "name": [string],
      "ciName": [string],
      "description": [string],
      "imageUrl": [string]
    },
    "enclosureDTO": {
      "id": [integer],
      "name": [string],
      "animalClass": [string],
      "mapsId": [string],
      "supportedAmount": [integer],
      "latitude": [double],
      "longitude": [double]
    }
  }
]
```

- **SAMPLE CALL:**

```
@GET("ae/animal/{animalId}")
suspend fun getAEByAnimalId(@Path("animalId") animalId: Int): List<AEDTO>
```

URL:

`/api/images/{imageName}`

METHOD:

`GET`

URL Parameters:

- Required:
  - `imageName: [string]`
    - The name of the image file to retrieve.

SUCCESS RESPONSE:

- **Code:** 200 OK
- **Content:**
  - Returns the requested image file as a binary stream.
  - Headers include `Content-Disposition` to suggest a download with the original filename, and `Content-Type` set to the appropriate media type (e.g., `image/jpeg`).

ERROR RESPONSE:

- **Code:** 404 NOT FOUND
- **Content:**

```
No content returned.
```

Manual do Utilizador

Manual do Utilizador

Autoavaliação do Projeto

O Zoopolis é um projeto que se destaca pela iniciativa, tendo em atenção os detalhes e pela dedicação em construir uma aplicação funcional, segura, intuitiva e interativa. Segue uma análise mais detalhada das áreas mais relevantes da aplicação:

API e Back-End

A API foi desenvolvida seguindo as boas práticas de arquitetura e design de dados. Os endpoints estão bem estruturados, e o tratamento de erros é robusto, com mensagens explicitas e informativas, que auxiliam no processo de debugging, facilitando os desenvolvedores. Além disso, o sistema mostrou-se flexível, pronto para futuras atualizações.

Front-End

O front-end foi desenvolvido focado na utilização e experiência do usuário. A interface é limpa, intuitiva e simples, adaptando-se a diferentes dispositivos e tamanhos de ecrã. A integração com o Google Maps API foi bem executada, proporcionando uma navegação suave e eficiente. Em especial, destacam-se:

- **Mapa Personalizado**, integrado com Google Maps API, que efetua o tratamento dos dados de forma limpa e eficaz .

Base de Dados

A base de dados foi projetada de forma eficiente, com uma estrutura bem normalizada e otimizada para diversos tipos de consultas. O dicionário de dados e o guia de dados fornecem informações detalhadas sobre todas as tabelas, campos e relacionamentos, facilitando a compreensão e manutenção da base de dados. As queries foram bem elaboradas, com uso adequado de índices e cláusulas de forma a garantir uma melhor performance e integridade dos dados.

- **Dados georreferenciados** para os recintos, permitindo consultas rápidas e integração fluida com o mapa personalizado no frontend.

Funcionalidades Implementadas

O Zoopolis apresenta um conjunto diverso de funcionalidades:

- **Sistema de busca de animais**, essencial para lidar com grandes quantidades de dados sobre os animais.
- **Login e registro de utilizadores**, com autenticação segura.
- **Menu de Favoritos**, com os animais favoritos de cada utilizador.
- **Mapa Personalizado** facilitando o percurso no Zoo para os utilizadores.

- **Sistema de pontuação**, permitindo assim uma experiencia mais divertida e única no Zoológico.

Conformidade com a Proposta

O projeto atende à proposta inicial, entregando todas as funcionalidades principais.

Opinião e Nota Final

O Zoopolis é um projeto completo, tecnicamente sólido, bem pensado e construído por apenas uma pessoa. A integração do back-end com a base de dados e front-end é eficiente, somada á usabilidade e escalabilidade, demonstra o compromisso com a entrega de um produto de alta qualidade.

Nota sugerida: 18/20