

Sistema Web de Gerenciamento de Zoológico: Um Guia Completo para Iniciantes

1. Introdução Didática: O Projeto “Zoo Manager”

Este documento é um guia passo a passo, extremamente detalhado e didático, projetado para **iniciantes** no mundo do desenvolvimento web. Nossa objetivo é construir um **Sistema Web de Gerenciamento de Zoológico** (que chamaremos de “Zoo Manager”) utilizando um conjunto de tecnologias modernas e poderosas.

1.1. Objetivo do Sistema e Contexto Acadêmico

O principal objetivo do “Zoo Manager” é permitir que a equipe de um zoológico virtual possa **gerenciar** de forma eficiente as informações sobre seus animais, espécies e habitats. Este projeto simula uma atividade acadêmica comum, onde o aluno deve demonstrar a capacidade de integrar um **frontend** (a interface que o usuário vê) com um **backend** (o servidor que processa a lógica) e um **banco de dados** (onde os dados são armazenados).

O sistema será capaz de realizar as quatro operações fundamentais de qualquer aplicação de gerenciamento de dados, conhecidas como **CRUD**:

- **Create** (Criar/Cadastrar novos animais)
- **Read** (Ler>Listar todos os animais e seus detalhes)
- **Update** (Atualizar/Editar informações de um animal existente)
- **Delete** (Deletar/Remover um animal do sistema)

1.2. Visão Geral da Arquitetura

Nosso sistema segue uma arquitetura de três camadas, que é o padrão para a maioria das aplicações web:

1. **Frontend (Interface do Usuário)**: Construído com **HTML, CSS e JavaScript**. É a parte que roda no navegador do usuário. O JavaScript será responsável por “conversar” com o Backend, enviando e recebendo dados.
2. **Backend (Servidor de Aplicação)**: Construído com **Node.js** e o framework **Express**. É o “cérebro” da aplicação, responsável por receber as requisições do Frontend, aplicar a lógica de negócio (como validar dados) e interagir com o Banco de Dados.
3. **Banco de Dados (Armazenamento de Dados)**: Utilizaremos o **PostgreSQL**, um poderoso banco de dados relacional. O **Sequelize** atuará como uma ponte entre o Backend e o Banco de Dados.

A comunicação entre o Frontend e o Backend é feita através de **APIs REST**, que são um conjunto de regras que permitem que diferentes sistemas troquem informações de forma padronizada, geralmente usando o formato **JSON** (JavaScript Object Notation).

2. Explicação Leiga das Tecnologias Usadas

Para que você, como iniciante, se sinta confortável, vamos desmistificar as ferramentas que usaremos, explicando o que são e qual o papel de cada uma no nosso projeto.

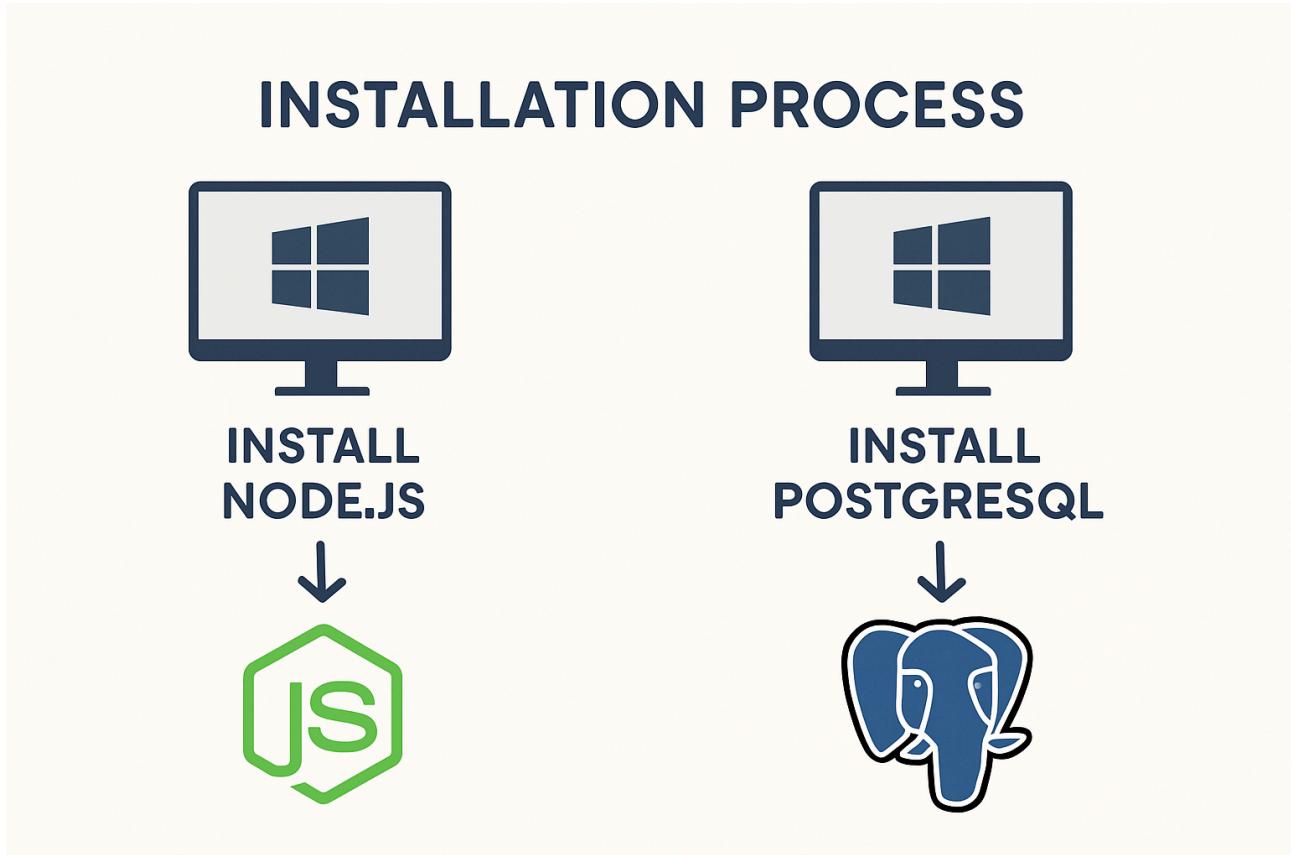
Tecnologia	O que é (Linguagem Simples)	Por que Usamos no Projeto
Node.js	É um ambiente de execução que permite rodar código JavaScript fora do navegador, ou seja, no nosso servidor.	Ele nos permite usar a mesma linguagem (JavaScript) tanto no Frontend quanto no Backend, simplificando o aprendizado e o desenvolvimento.
Express	É um framework (um conjunto de ferramentas e regras) que roda sobre o Node.js. Pense nele como um “kit de construção” para servidores web.	Ele facilita a criação de rotas (os “endereços” da nossa API, como <code>/animais</code> ou <code>/especies</code>) e a organização do nosso código de Backend.
PostgreSQL	É um Banco de Dados Relacional . Imagine-o como um grande armário de arquivos onde os dados são organizados em tabelas (como planilhas), com linhas (registros) e colunas (campos).	É robusto, gratuito e ideal para armazenar dados estruturados, como os detalhes dos nossos animais.
Sequelize	É um ORM (Object-Relational Mapper) . Ele é a “ponte” mágica que traduz o código JavaScript em comandos SQL (a linguagem do banco de dados).	Em vez de escrever SQL complexo, escrevemos código JavaScript simples para criar, ler e atualizar dados. Ele também cria as tabelas no banco de dados automaticamente para nós.
HTML, CSS, JS	São as linguagens básicas do Frontend. HTML estrutura o conteúdo, CSS define a aparência (cores, layout) e JavaScript adiciona interatividade.	O JavaScript, especificamente, usará a função <code>fetch()</code> para enviar e receber dados do nosso Backend (a API REST), garantindo que a interface se atualize dinamicamente.

3. Instruções de Instalação de Todas as Ferramentas

Antes de começar a codificar, precisamos preparar nosso ambiente de trabalho.

3.1. Instalação do Node.js e PostgreSQL

A base do nosso projeto são o ambiente de execução Node.js e o banco de dados PostgreSQL.



- 1. Instalar Node.js:** Acesse o site oficial do Node.js e baixe a versão LTS (Long-Term Support), que é a mais estável. A instalação é um processo simples de “próximo, próximo, finalizar” .
- 2. Instalar PostgreSQL:** Acesse o site oficial do PostgreSQL e baixe o instalador. Durante a instalação, você será solicitado a criar uma **senha** para o usuário `postgres`. **Anote esta senha**, pois ela será crucial para a conexão do nosso sistema. O instalador geralmente já inclui o **pgAdmin**, uma ferramenta gráfica para gerenciar o banco de dados.
- 3. Instalar o VSCode:** O Visual Studio Code é o editor de código mais popular e poderoso. Baixe e instale-o para ter um ambiente de desenvolvimento eficiente.

3.2. Criação do Banco de Dados

Com o PostgreSQL instalado, vamos criar o banco de dados `zoologico`:

1. Abra o **pgAdmin** (geralmente instalado junto com o PostgreSQL).
2. Conecte-se ao servidor usando a senha que você definiu na instalação.
3. Clique com o botão direito em **Databases** e selecione **Create > Database....**
4. No campo **Database**, digite `zoologico`.
5. Clique em **Save**.

Pronto! O Sequelize se conectará a este banco de dados e criará as tabelas para nós.

3.3. Inicialização do Projeto e Instalação de Dependências

Vamos criar a pasta do projeto e instalar os pacotes necessários.

1. Abra o terminal (ou Prompt de Comando/PowerShell) e crie a pasta do projeto:

```
mkdir zoologico-node  
cd zoologico-node
```

2. Inicialize o projeto Node.js. Isso criará o arquivo `package.json`:

```
npm init -y
```

3. Instale as dependências principais:

```
npm install express sequelize pg pg-hstore
```

- `express` : O framework web.
- `sequelize` : O ORM.
- `pg` e `pg-hstore` : Drivers que permitem ao Sequelize se comunicar com o PostgreSQL.

4. Modelagem do Banco de Dados

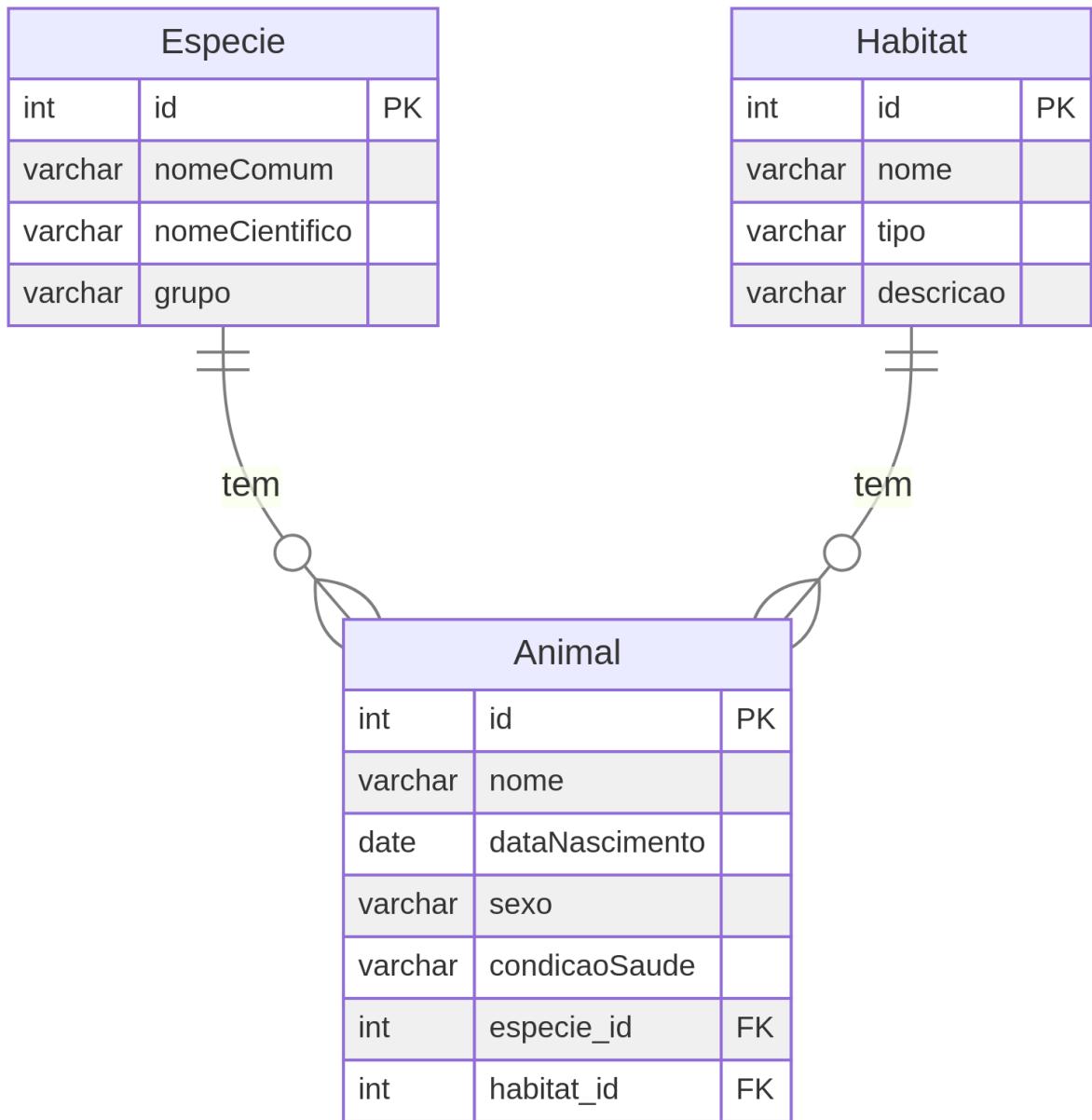
A modelagem de dados é a etapa onde definimos a estrutura das nossas informações. No nosso zoológico, teremos três entidades principais: **Espécie**, **Habitat** e **Animal**.

4.1. Diagrama Entidade-Relacionamento (ER)

O diagrama ER mostra como as tabelas se relacionam. No nosso caso, temos um relacionamento de **Um para Muitos** (1:N):

- Uma **Espécie** pode ter muitos **Animais**.
- Um **Habitat** pode ter muitos **Animais**.
- Um **Animal** pertence a exatamente uma **Espécie** e a exatamente um **Habitat**.

Isso é representado pelas **Chaves Estrangeiras (FK)** na tabela `Animal`.



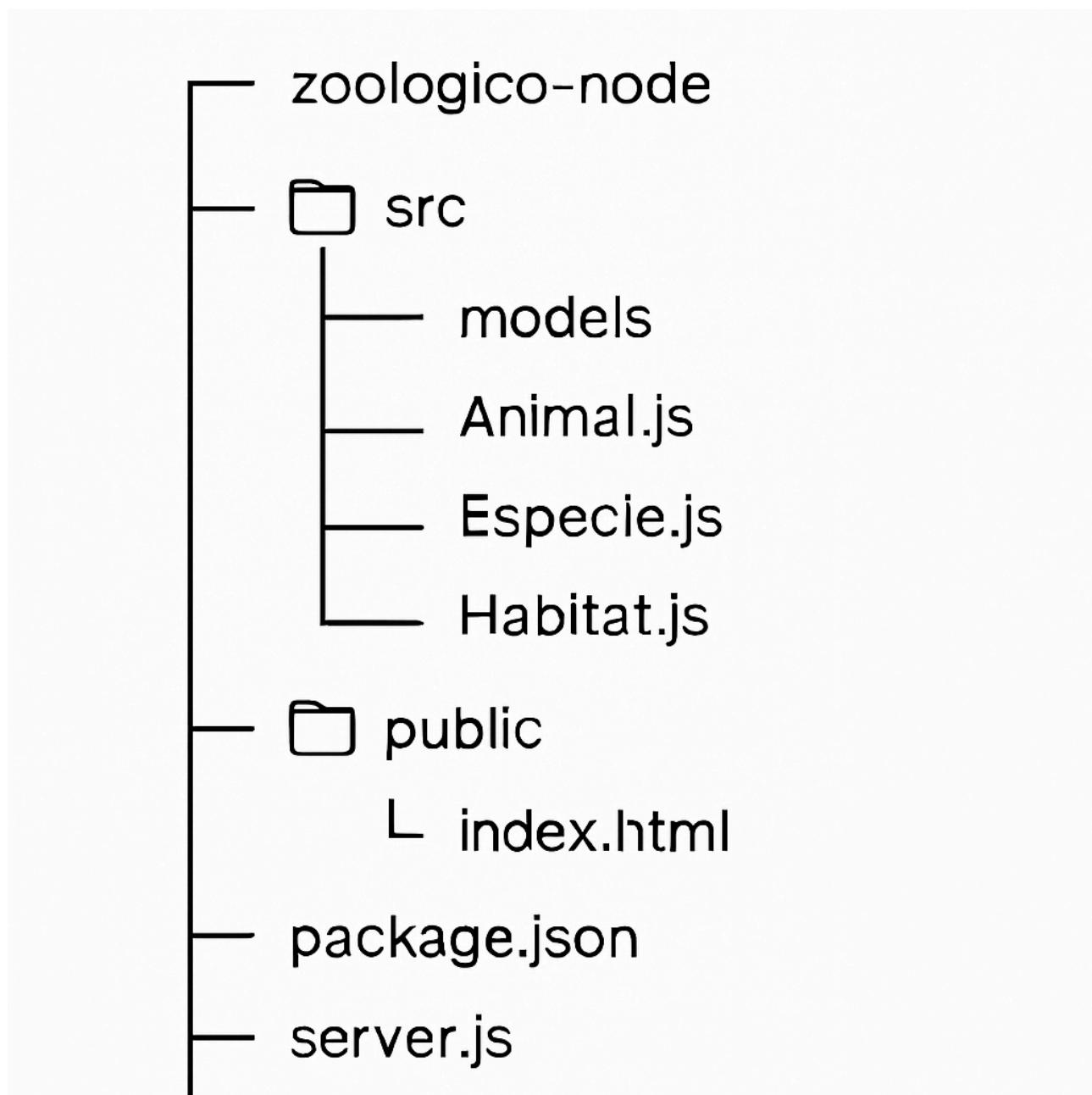
4.2. Estrutura das Tabelas

A tabela a seguir detalha os campos (colunas) de cada entidade:

Tabela	Campo	Tipo de Dado	Descrição	Observação
Especie	<code>id</code>	INTEGER	Chave Primária	Gerado automaticamente pelo Sequelize.
	<code>nomeComum</code>	VARCHAR	Nome popular da espécie.	Ex: “Leão”
	<code>nomeCientifico</code>	VARCHAR	Nome científico.	Ex: “Panthera leo”
	<code>grupo</code>	VARCHAR	Classificação (Mamífero, Ave, Réptil).	
Habitat	<code>id</code>	INTEGER	Chave Primária	Gerado automaticamente.
	<code>nome</code>	VARCHAR	Nome do habitat.	Ex: “Savana Africana”
	<code>tipo</code>	VARCHAR	Tipo de ambiente.	Ex: “Terrestre”
	<code>descricao</code>	VARCHAR	Detalhes sobre o habitat.	
Animal	<code>id</code>	INTEGER	Chave Primária	Gerado automaticamente.
	<code>nome</code>	VARCHAR	Nome do animal.	Ex: “Simba”
	<code>dataNascimento</code>	DATE	Data de nascimento.	
	<code>sexo</code>	VARCHAR	Sexo do animal.	“M” ou “F”
	<code>condicaoSaude</code>	VARCHAR	Status de saúde.	Ex: “Estável”
	<code>especie_id</code>	INTEGER	Chave Estrangeira (FK)	Liga ao <code>id</code> da tabela <code>Especie</code> .
	<code>habitat_id</code>	INTEGER	Chave Estrangeira (FK)	Liga ao <code>id</code> da tabela <code>Habitat</code> .

5. Estrutura do Projeto Node.js

Para manter nosso código organizado, usaremos uma estrutura de pastas modular.



Vamos criar as seguintes pastas e arquivos dentro de `zoologico-node/`:

- `src/` : Contém a lógica principal do Backend.
 - `src/models/` : Onde definiremos os modelos do Sequelize (as tabelas).
- `public/` : Contém os arquivos estáticos do Frontend (HTML, CSS, JavaScript).
- `server.js` : O ponto de entrada do nosso servidor.
- `db.js` : O arquivo de configuração e inicialização do Sequelize.

6. Explicando Cada Arquivo do Backend

6.1. Configuração do Banco de Dados (db.js)

Este arquivo é responsável por configurar a conexão com o PostgreSQL e inicializar o Sequelize.

```
// zoologico-node/db.js

const { Sequelize } = require('sequelize');

// 1. Configurações de Conexão (ADAPTE SUA SENHA AQUI!)
const sequelize = new Sequelize('zoologico', 'postgres', 'SUA_SENHA_AQUI', {
  host: 'localhost',
  dialect: 'postgres', // Indica que estamos usando PostgreSQL
  logging: false, // Desativa logs SQL no console para um ambiente mais
limpo
  pool: {
    max: 5,
    min: 0,
    acquire: 30000,
    idle: 10000
  }
});

// 2. Importação dos Modelos
const db = {};
db.Sequelize = Sequelize;
db.sequelize = sequelize;

// Importa e inicializa os modelos (as tabelas)
db.Especie = require('./src/models/Especie')(sequelize, Sequelize);
db.Habitat = require('./src/models/Habitat')(sequelize, Sequelize);
db.Animal = require('./src/models/Animal')(sequelize, Sequelize);

// 3. Definição dos Relacionamentos (Chaves Estrangeiras)
// Um Animal pertence a uma Espécie
db.Animal.belongsTo(db.Especie, {
  foreignKey: 'especie_id',
  as: 'especie'
});
// Uma Espécie tem muitos Animais
db.EspeciehasMany(db.Animal, {
  foreignKey: 'especie_id',
  as: 'animais'
});

// Um Animal pertence a um Habitat
db.Animal.belongsTo(db.Habitat, {
  foreignKey: 'habitat_id',
  as: 'habitat'
});
```

```
// Um Habitat tem muitos Animais
db.Habitat.hasMany(db.Animal, {
  foreignKey: 'habitat_id',
  as: 'animais'
});

module.exports = db;
```

6.2. Definição dos Modelos (src/models/)

Os modelos são a representação das nossas tabelas em código JavaScript.

src/models/Especie.js

```
// zoologico-node/src/models/Especie.js

module.exports = (sequelize, DataTypes) => {
  const Especie = sequelize.define('Especie', {
    nomeComum: {
      type: DataTypes.STRING,
      allowNull: false
    },
    nomeCientifico: {
      type: DataTypes.STRING,
      allowNull: false,
      unique: true
    },
    grupo: {
      type: DataTypes.STRING
    }
  });

  return Especie;
};
```

src/models/Habitat.js

```
// zoologico-node/src/models/Habitat.js

module.exports = (sequelize, DataTypes) => {
  const Habitat = sequelize.define('Habitat', {
    nome: {
      type: DataTypes.STRING,
      allowNull: false,
      unique: true
    },
    tipo: {
      type: DataTypes.STRING
    },
    descricao: {
      type: DataTypes.TEXT
    }
  });

  return Habitat;
};
```

src/models/Animal.js

Este modelo inclui as chaves estrangeiras (especie_id e habitat_id).

```
// zoologico-node/src/models/Animal.js

module.exports = (sequelize, DataTypes) => {
  const Animal = sequelize.define('Animal', {
    nome: {
      type: DataTypes.STRING,
      allowNull: false
    },
    dataNascimento: {
      type: DataTypes.DATEONLY, // Apenas a data, sem hora
      allowNull: false
    },
    sexo: {
      type: DataTypes.ENUM('M', 'F'), // Restringe a 'M' ou 'F'
      allowNull: false
    },
    condicaoSaude: {
      type: DataTypes.STRING,
      defaultValue: 'Estável'
    },
    // As chaves estrangeiras (FKs) serão adicionadas automaticamente pelo
    // Sequelize
    // devido aos relacionamentos definidos em db.js
  });

  return Animal;
};
```

6.3. O Servidor Principal (`server.js`)

Este arquivo configura o Express, define as rotas da API e inicia o servidor.

```
// zoologico-node/server.js

const express = require('express');
const db = require('./db'); // Importa a configuração do banco de dados
const app = express();
const PORT = 8080;

// Configuração do Middleware
app.use(express.json()); // Permite que o Express leia JSON no corpo das
requisições
app.use(express.urlencoded({ extended: true })); // Permite ler dados de
formulários
app.use(express.static('public')); // Serve os arquivos estáticos (HTML,
CSS, JS) da pasta 'public'

// -----
// 1. Definição das Rotas da API (CRUD)
// -----


// Rota para LISTAR todos os animais
app.get('/api/animais', async (req, res) => {
  try {
    const animais = await db.Animal.findAll({
      // Inclui os dados de Espécie e Habitat no resultado
      include: [{ model: db.Especie, as: 'especie' }, { model: db.Habitat,
      as: 'habitat' }]
    });
    res.status(200).json(animais);
  } catch (error) {
    res.status(500).json({ message: 'Erro ao buscar animais', error:
    error.message });
  }
});

// Rota para CADASTRAR um novo animal
app.post('/api/animais', async (req, res) => {
  try {
    const novoAnimal = await db.Animal.create(req.body);
    res.status(201).json(novoAnimal);
  } catch (error) {
    res.status(500).json({ message: 'Erro ao cadastrar animal', error:
    error.message });
  }
});
```

```
// Rota para ATUALIZAR um animal por ID
app.put('/api/animais/:id', async (req, res) => {
  try {
    const [updated] = await db.Animal.update(req.body, {
      where: { id: req.params.id }
    });
    if (updated) {
      const animalAtualizado = await db.Animal.findByPk(req.params.id);
      return res.status(200).json(animalAtualizado);
    }
    throw new Error('Animal não encontrado');
  } catch (error) {
    res.status(500).json({ message: 'Erro ao atualizar animal', error: error.message });
  }
});

// Rota para DELETAR um animal por ID
app.delete('/api/animais/:id', async (req, res) => {
  try {
    const deleted = await db.Animal.destroy({
      where: { id: req.params.id }
    });
    if (deleted) {
      return res.status(204).send("Animal deletado com sucesso");
    }
    throw new Error('Animal não encontrado');
  } catch (error) {
    res.status(500).json({ message: 'Erro ao deletar animal', error: error.message });
  }
});

// -----
// 2. Inicialização do Servidor e Sincronização do Banco
// -----


// Sincroniza os modelos com o banco de dados (cria as tabelas se não existirem)
db.sequelize.sync({ force: true }) // Use { force: true } apenas em desenvolvimento para recriar as tabelas
  .then(() => {
    console.log('Banco de dados sincronizado. Tabelas criadas!');
    // Inicia o servidor Express
    app.listen(PORT, () => {
      console.log(`Servidor rodando em http://localhost:${PORT}`);
    });
  });

```

```
    });
  })
  .catch(err => {
    console.error('Erro ao sincronizar o banco de dados:', err);
  });
}
```

7. Frontend: A Interface do Usuário

O Frontend é a parte que o usuário interage. Usaremos HTML para a estrutura, CSS para o estilo e JavaScript para a lógica de comunicação com o Backend.

7.1. Mockup da Interface

A interface será simples, focada na funcionalidade CRUD.

O mockup apresenta a interface do Zoo Management System. No topo, uma barra azul contém o título "ZOO MANAGEMENT SYSTEM". Abaixo, uma seção para "Animal Registration" contém campos para "Name" (campo de texto), "Sex" (campo dropdown com opções "Male" e "Female"), "Date of Birth" (campo com máscara YYYY-MM-DD) e "Species" (campo dropdown com lista suspenso). Um botão "REGISTER" em destaque em tons de verde e azul está posicionado no centro. Abaixo, uma tabela mostra os detalhes de quatro animais: Leo (Lion, Savannain), Ellie (Elephant, Grossland), Manny (Tortoise, Desart) e Tina (Tiger, Jungle). Cada linha da tabela inclui botões "Edit" (em verde) e "Remove" (em vermelho).

Name	Date of Birth	Sex	Species	Habitat	
Leo	2010-06-15	Male	Lion	Savannain	<button>Edit</button>
Ellie	2009-09-33	Female	Elephant	Grossland	<button>Remove</button>
Manny	2015-00-02	Male	Tortoise	Desart	<button>Edit</button>
Tina	2017-07-29	Tiger	Tiger	Jungle	<button>Remove</button>

7.2. Estrutura HTML (public/index.html)

O HTML define o formulário de cadastro e a tabela que listará os animais.

```
<!-- zoologico-node/public/index.html -->
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Zoo Manager - Gerenciamento de Animais</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">
        <h1>Zoo Manager</h1>
        <h2>Cadastro de Animal</h2>

        <form id="animal-form">
            <input type="hidden" id="animal-id">

            <label for="nome">Nome:</label>
            <input type="text" id="nome" required>

            <label for="dataNascimento">Data de Nascimento:</label>
            <input type="date" id="dataNascimento" required>

            <label for="sexo">Sexo:</label>
            <select id="sexo" required>
                <option value="M">Macho</option>
                <option value="F">Fêmea</option>
            </select>

            <label for="especie_id">Espécie:</label>
            <!-- Este select será preenchido via JavaScript -->
            <select id="especie_id" required></select>

            <label for="habitat_id">Habitat:</label>
            <!-- Este select será preenchido via JavaScript -->
            <select id="habitat_id" required></select>

            <button type="submit" id="submit-button">Cadastrar
                Animal</button>
        </form>

        <h2>Animais Cadastrados</h2>
        <table id="animais-table">
            <thead>
                <tr>
```

```
<th>Nome</th>
<th>Espécie</th>
<th>Habitat</th>
<th>Nascimento</th>
<th>Saúde</th>
<th>Ações</th>
</tr>
</thead>
<tbody>
    <!-- Linhas de animais serão inseridas aqui pelo JavaScript
-->
</tbody>
</table>
</div>

<script src="script.js"></script>
</body>
</html>
```

7.3. Lógica JavaScript (public/script.js)

O JavaScript é o motor do Frontend. Ele usa a função `fetch()` para se comunicar com o Backend.

```
// zoologico-node/public/script.js

const API_URL = '/api/animais';
const form = document.getElementById('animal-form');
const tableBody = document.querySelector('#animais-table tbody');
const submitButton = document.getElementById('submit-button');

// Função para carregar e preencher os selects de Espécie e Habitat
async function loadDependencies() {
    // Simulação: Em um projeto real, teríamos rotas separadas para Espécies
    // e Habitats
    // Para simplificar, vamos usar dados mockados para a demonstração
    // didática
    const especies = [
        { id: 1, nomeComum: 'Leão' },
        { id: 2, nomeComum: 'Elefante' },
        { id: 3, nomeComum: 'Tartaruga' },
        { id: 4, nomeComum: 'Tigre' }
    ];
    const habitats = [
        { id: 1, nome: 'Savana' },
        { id: 2, nome: 'Floresta' },
        { id: 3, nome: 'Deserto' }
    ];

    const especieSelect = document.getElementById('especie_id');
    const habitatSelect = document.getElementById('habitat_id');

    especies.forEach(e => {
        const option = document.createElement('option');
        option.value = e.id;
        option.textContent = e.nomeComum;
        especieSelect.appendChild(option);
    });

    habitats.forEach(h => {
        const option = document.createElement('option');
        option.value = h.id;
        option.textContent = h.nome;
        habitatSelect.appendChild(option);
    });
}

// Função para carregar e exibir a lista de animais (READ)
async function loadAnimais() {
```

```
const response = await fetch(API_URL);
const animais = await response.json();

tbody.innerHTML = '' // Limpa a tabela

animais.forEach(animal => {
  const row = tbody.insertRow();
  row.insertCell().textContent = animal.nome;
  // Acessa os dados relacionados (incluídos pelo Sequelize)
  row.insertCell().textContent = animal.especie ?
    animal.especie.nomeComum : 'N/A';
  row.insertCell().textContent = animal.habitat ? animal.habitat.nome
    : 'N/A';
  row.insertCell().textContent = animal.dataNascimento;
  row.insertCell().textContent = animal.condicaoSaude;

  // Botões de Ação
  const actionsCell = row.insertCell();
  const editBtn = document.createElement('button');
  editBtn.textContent = 'Editar';
  editBtn.onclick = () => editAnimal(animal);

  const deleteBtn = document.createElement('button');
  deleteBtn.textContent = 'Remover';
  deleteBtn.onclick = () => deleteAnimal(animal.id);

  actionsCell.appendChild(editBtn);
  actionsCell.appendChild(deleteBtn);
});

// Função para submeter o formulário (CREATE e UPDATE)
form.addEventListener('submit', async (e) => {
  e.preventDefault();

  const animalId = document.getElementById('animal-id').value;
  const method = animalId ? 'PUT' : 'POST';
  const url = animalId ? `${API_URL}/${animalId}` : API_URL;

  const animalData = {
    nome: document.getElementById('nome').value,
    dataNascimento: document.getElementById('dataNascimento').value,
    sexo: document.getElementById('sexo').value,
    especie_id: document.getElementById('especie_id').value,
    habitat_id: document.getElementById('habitat_id').value,
    // condicaoSaude pode ser adicionado aqui se houver um campo no form
  }
})
```

```
};

const response = await fetch(url, {
    method: method,
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(animalData)
});

if (response.ok) {
    alert(animalId ? 'Animal atualizado com sucesso!' : 'Animal cadastrado com sucesso!');
    form.reset();
    document.getElementById('animal-id').value = '';
    submitButton.textContent = 'Cadastrar Animal';
    loadAnimais();
} else {
    alert('Erro na operação.');
}
});

// Função para carregar dados no formulário para edição (UPDATE)
function editAnimal(animal) {
    document.getElementById('animal-id').value = animal.id;
    document.getElementById('nome').value = animal.nome;
    document.getElementById('dataNascimento').value = animal.dataNascimento;
    document.getElementById('sexo').value = animal.sexo;
    document.getElementById('especie_id').value = animal.especie_id;
    document.getElementById('habitat_id').value = animal.habitat_id;
    submitButton.textContent = 'Atualizar Animal';
}

// Função para deletar um animal (DELETE)
async function deleteAnimal(id) {
    if (confirm('Tem certeza que deseja remover este animal?')) {
        const response = await fetch(`${API_URL}/${id}`, {
            method: 'DELETE'
        });

        if (response.status === 204) {
            alert('Animal removido com sucesso!');
            loadAnimais();
        } else {
            alert('Erro ao remover animal.');
        }
    }
}
```

```
// Inicialização
document.addEventListener('DOMContentLoaded', () => {
    loadDependencies();
    loadAnimais();
});
```

7.4. Estilo CSS (public/style.css)

Um CSS básico para dar uma aparência limpa e profissional.

```
/* zoologico-node/public/style.css */

body {
    font-family: Arial, sans-serif;
    background-color: #f4f7f6;
    color: #333;
    margin: 0;
    padding: 20px;
}

.container {
    max-width: 1000px;
    margin: 0 auto;
    background-color: #fff;
    padding: 30px;
    border-radius: 8px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}

h1, h2 {
    color: #007bff;
    border-bottom: 2px solid #eee;
    padding-bottom: 10px;
    margin-top: 20px;
}

/* Estilo do Formulário */
#animal-form label {
    display: block;
    margin-top: 10px;
    font-weight: bold;
}

#animal-form input[type="text"],
#animal-form input[type="date"],
#animal-form select {
    width: 100%;
    padding: 10px;
    margin-top: 5px;
    margin-bottom: 15px;
    border: 1px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box;
}
```

```
#animal-form button {
    background-color: #28a745;
    color: white;
    padding: 12px 20px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    font-size: 16px;
    transition: background-color 0.3s;
}

#animal-form button:hover {
    background-color: #218838;
}

/* Estilo da Tabela */
#animais-table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
}

#animais-table th, #animais-table td {
    border: 1px solid #ddd;
    padding: 12px;
    text-align: left;
}

#animais-table th {
    background-color: #007bff;
    color: white;
}

#animais-table tr:nth-child(even) {
    background-color: #f2f2f2;
}

#animais-table button {
    padding: 5px 10px;
    margin-right: 5px;
    border: none;
    border-radius: 3px;
    cursor: pointer;
}

#animais-table button:first-of-type {
```

```
background-color: #ffc107; /* Editar */
color: #333;
}

#animais-table button:last-of-type {
background-color: #dc3545; /* Remover */
color: white;
}
```

8. Seed (Dados Iniciais)

Para testar o sistema, precisamos de dados iniciais nas tabelas `Especie` e `Habitat`. Vamos adicionar uma função de *seeding* (população inicial) ao nosso `server.js`.

Modificação no `server.js`:

Adicione a função `seedDatabase` e chame-a após a sincronização do banco de dados.

```
// zoologico-node/server.js (Trecho modificado)

// ... (código anterior)

// Função para popular o banco de dados com dados iniciais
async function seedDatabase() {
    console.log('Populando o banco de dados...');

    // 1. Inserir Espécies
    const especiesData = [
        { nomeComum: 'Leão', nomeCientifico: 'Panthera leo', grupo: 'Mamífero' },
        { nomeComum: 'Elefante Africano', nomeCientifico: 'Loxodonta africana', grupo: 'Mamífero' },
        { nomeComum: 'Arara Azul', nomeCientifico: 'Anodorhynchus hyacinthinus', grupo: 'Ave' },
        { nomeComum: 'Cobra Píton', nomeCientifico: 'Python regius', grupo: 'Réptil' },
        { nomeComum: 'Tartaruga Marinha', nomeCientifico: 'Chelonioidea', grupo: 'Réptil' }
    ];
    const especies = await db.Especie.bulkCreate(especiesData, {
        ignoreDuplicates: true
    });

    // 2. Inserir Habitats
    const habitatsData = [
        { nome: 'Savana', tipo: 'Terrestre', descricao: 'Grandes planícies abertas.' },
        { nome: 'Floresta Tropical', tipo: 'Terrestre', descricao: 'Ambiente denso e úmido.' },
        { nome: 'Aquário', tipo: 'Aquático', descricao: 'Tanque de água salgada.' },
        { nome: 'Viveiro', tipo: 'Aéreo', descricao: 'Grande gaiola para aves.' }
    ];
    const habitats = await db.Habitat.bulkCreate(habitatsData, {
        ignoreDuplicates: true
    });

    // 3. Inserir Animais (25 exemplos)
    const animaisData = [];
    const leaoId = especies.find(e => e.nomeComum === 'Leão').id;
    const elefanteId = especies.find(e => e.nomeComum === 'Elefante Africano').id;
    const araraId = especies.find(e.nomeComum === 'Arara Azul').id;
    const savanaId = habitats.find(h => h.nome === 'Savana').id;
```

```
const viveiroId = habitats.find(h => h.nome === 'Viveiro').id;

for (let i = 1; i <= 25; i++) {
    let especie_id, habitat_id;
    if (i % 3 === 0) {
        especie_id = araraId;
        habitat_id = viveiroId;
    } else if (i % 2 === 0) {
        especie_id = elefanteId;
        habitat_id = savanaId;
    } else {
        especie_id = leaoId;
        habitat_id = savanaId;
    }

    animaisData.push({
        nome: `Animal ${i}`,
        dataNascimento: `2020-01-${i < 10 ? '0' + i : i}`,
        sexo: i % 2 === 0 ? 'F' : 'M',
        condicaoSaude: i % 5 === 0 ? 'Em Tratamento' : 'Estável',
        especie_id: especie_id,
        habitat_id: habitat_id
    });
}

await db.Animal.bulkCreate(animaisData);
console.log('Dados iniciais inseridos com sucesso!');
}

// ... (Trecho final do server.js)

// Sincroniza os modelos com o banco de dados (cria as tabelas se não existirem)
db.sequelize.sync({ force: true }) // Use { force: true } apenas em desenvolvimento para recriar as tabelas
.then(() =>
    console.log('Banco de dados sincronizado. Tabelas criadas!');
    return seedDatabase() // Chama a função de população
)
.then(() => {
    // Inicia o servidor Express
    app.listen(PORT, () => {
        console.log(`Servidor rodando em http://localhost:${PORT}`);
    });
})
.catch(err => {
```

```
        console.error('Erro fatal ao iniciar o sistema:', err);
    });
}
```

9. Execução do Sistema Completo

Com todos os arquivos criados, o sistema está pronto para rodar.

1. Rodar o Servidor: No terminal, dentro da pasta `zoologico-node/`, execute:

```
node server.js
```

Você verá as mensagens de log:

```
Banco de dados sincronizado. Tabelas criadas!
Populando o banco de dados...
Dados iniciais inseridos com sucesso!
Servidor rodando em http://localhost:8080
```

2. Abrir o Navegador: Abra seu navegador e acesse: <http://localhost:8080>

Você verá a interface do Frontend e a tabela já preenchida com os 25 animais que acabamos de inserir (o *seed*).

10. Testes Funcionais

Agora, vamos testar as funcionalidades CRUD e algumas consultas avançadas.

10.1. Testes CRUD (Interface Web)

- **Cadastro de Animal (CREATE):** Preencha o formulário com um novo animal (ex: “Pingo”, Espécie: “Leão”, Habitat: “Savana”) e clique em **Cadastrar Animal**. O novo animal deve aparecer na tabela.

- **Edição (UPDATE):** Clique no botão **Editar** ao lado de um animal. Os dados dele devem ser carregados no formulário. Mude o nome ou a condição de saúde e clique em **Atualizar Animal**. A tabela deve refletir a mudança.
- **Remoção (DELETE):** Clique no botão **Remover** ao lado de um animal. Confirme a exclusão. O animal deve desaparecer da tabela.

10.2. Consultas Avançadas (Desafio de Código)

Para realizar consultas mais complexas, você precisaria adicionar novas rotas no `server.js` e funções no `script.js`.

Consulta	Lógica no Sequelize (Exemplo)
Filtro por Espécie	<code>db.Animal.findAll({ where: { especie_id: id_da_especie } })</code>
Filtro por Habitat	<code>db.Animal.findAll({ where: { habitat_id: id_do_habitat } })</code>
Animal Mais Velho	<code>db.Animal.findAll({ order: [['dataNascimento', 'ASC']], limit: 1 })</code>
Média de Idade por Espécie	Usar <code>db.Animal.findAll({ attributes: [[db.sequelize.fn('AVG', db.sequelize.fn('AGE', db.sequelize.col('dataNascimento'))), 'mediaIdade']], include: [{ model: db.Especie, as: 'especie' }], group: ['especie.id'] })</code> (Esta é uma consulta mais avançada que requer o uso de funções de agregação do Sequelize/PostgreSQL).

11. Conclusão e Próximos Passos

Parabéns! Você construiu um sistema web completo, do zero, integrando Frontend, Backend e Banco de Dados.

11.1. O que Foi Aprendido

- **Full-Stack:** Você viu como as três camadas (Frontend, Backend, DB) trabalham juntas.

- **APIs REST:** Entendeu como o Frontend usa `fetch()` para se comunicar com o Backend via rotas.
- **ORM:** Aprendeu a usar o Sequelize para manipular o banco de dados sem escrever SQL complexo.
- **Estrutura de Projeto:** Dominou a organização de um projeto Node.js modular.

11.2. Sugestões de Melhorias e Evolução do Projeto

Para evoluir este projeto e torná-lo uma aplicação de nível profissional, você pode implementar:

- **Autenticação e Autorização (Auth):** Adicionar telas de login e registro para garantir que apenas usuários autorizados possam acessar e modificar os dados.
- **Rotas de Dependência:** Criar rotas `/api/especies` e `/api/habitats` para que o Frontend possa carregar os *selects* dinamicamente, em vez de usar dados *mockados*.
- **Relatórios e Gráficos:** Implementar as consultas avançadas (como a média de idade por espécie) e usar bibliotecas de Frontend (como Chart.js) para exibir os resultados em gráficos.
- **Validação de Dados:** Adicionar validações mais robustas no Backend para garantir a integridade dos dados antes de salvar no banco.

Data: 23 de Novembro de 2025 **Revisão:** 1.0 [1] Sequelize ORM.
Documentação Oficial. <https://sequelize.org/> [2] Express.js. *Documentação Oficial.* <https://expressjs.com/> [3] PostgreSQL. *Documentação Oficial.* <https://www.postgresql.org/> [4] BezKoder. *Node.js Express & PostgreSQL: CRUD Rest APIs example with Sequelize.* <https://www.bezkoder.com/node-express-sequelize-postgresql/>