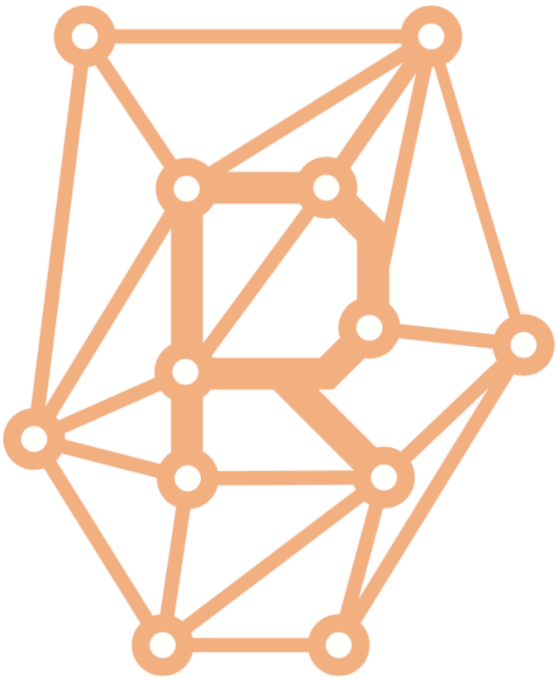
15 DE JUNHO DE 2024



Projeto II

TOMÁS FERREIRA, JOÃO CARVALHOSA
IPVC-ESTG
Engenharia de Redes e Sistemas de Computadores

Índice

Introdução

Este projeto é a continuação do Projeto I, onde aprimoraremos a sua funcionalidade integrando uma API e uma base de dados. O principal objetivo é construir uma API capaz de manipular os dados necessários, tornando o sistema mais dinâmico e robusto. A transição para uma base de dados em PostgreSQL garantirá a persistência e integridade dos dados, substituindo o uso anterior de JSON.

O projeto Hinoportuna, um site de gestão de recursos da tuna, oferece aos utilizadores uma área principal para registo e login. Após autenticação, eles têm acesso a uma área com ícones que direcionam para as tabelas CRUD de instrumentos, acessórios e cordas. No Projeto I, foram utilizadas tecnologias front end como HTML, CSS, Bootstrap, JavaScript e JSON para criar uma interface funcional.

Nesta iteração, além de manter a estrutura de zonas pública e privada, faremos melhorias significativas na zona privada. Utilizando Node.js e Prisma, iremos reconstruir a estrutura de páginas CRUD, proporcionando uma integração mais sólida com a base de dados PostgreSQL. Essa mudança garantirá uma gestão de dados mais eficiente, escalável e segura, permitindo uma maior flexibilidade e expansibilidade do sistema.

Ao implementar Node.js, teremos um ambiente de execução JavaScript no lado do servidor, o que permitirá uma integração mais fluida entre o front end e o back end. O Prisma, como ORM moderno, simplificará a interação com a base de dados, oferecendo uma abstração de alto nível sobre as operações SQL e aumentando a produtividade do desenvolvedor.

Em resumo, esta iteração do projeto visa elevar o Projeto Hinoportuna a um nível superior, proporcionando uma experiência de utilizador aprimorada, uma gestão de dados mais eficiente e uma base sólida para futuras expansões e melhoria.

Estado de Arte

O desenvolvimento de aplicações web tem evoluído significativamente, incorporando tecnologias avançadas que permitem a criação de sistemas mais eficientes, seguros e funcionais. Este projeto, em continuidade ao Projeto I, pretende adicionar uma base de dados e uma API para manipulação dos dados. Utilizando PostgreSQL e Prisma combinado com o HTML, CSS, Bootstrap e JavaScript, o objetivo é criar uma aplicação web robusta com uma interface moderna e funcional.

1. Tecnologias de Frontend

1.1 HTML e CSS

HTML (HyperText Markup Language) é a linguagem padrão para a criação de páginas web, definindo a estrutura do conteúdo. CSS (Cascading Style Sheets) é utilizado para estilizar e definir a apresentação das páginas web. Ambos são essenciais para o desenvolvimento frontend, permitindo a criação de interfaces de utilizador atrativas e funcionais.

1.2 Bootstrap

Bootstrap é um framework CSS amplamente utilizado para o desenvolvimento de interfaces web responsivas e móveis. Ele oferece uma vasta gama de componentes prontos, como botões, formulários, e layouts de grelha, facilitando a criação de interfaces consistentes e modernas.

1.3 JavaScript

JavaScript é uma linguagem de programação essencial para o desenvolvimento web, permitindo a criação de interatividade e dinamismo nas páginas web. No Projeto I, JavaScript foi utilizado para a implementação de um sistema de login utilizando JSON. No Projeto II, o seu papel será ampliado para integrar-se com a API e a base de dados, proporcionando uma experiência de utilizador mais rica e interativa.

2. Tecnologias de Backend

2.1 PostgreSQL

PostgreSQL é um sistema de gestão de bases de dados objeto-relacional poderoso e de código aberto. É conhecido pela sua robustez, escalabilidade e conformidade com padrões SQL. A utilização de PostgreSQL no Projeto II permitirá uma gestão eficiente e segura dos dados, suportando operações complexas e garantindo a integridade dos dados.

2.2 Prisma

Prisma é um ORM (Object-Relational Mapping) moderno para Node.js, que simplifica o acesso e a manipulação de bases de dados. Ele oferece uma abstração de alto nível sobre as operações SQL, facilitando a interação com a base de dados e aumentando a produtividade dos desenvolvedores. No Projeto II, o Prisma será utilizado para definir e manipular o esquema da base de dados, executar consultas e garantir a consistência dos dados.

3. Implementação de APIs

3.1 Node.js

Node.js é um ambiente de execução JavaScript server-side, permitindo a construção de aplicações escaláveis e eficientes. No Projeto II, Node.js será utilizado para implementar a API que servirá como intermediário entre o frontend e a base de dados, manipulando as requisições HTTP gerindo a lógica da aplicação web.

3.2 Estrutura de Rotas

A implementação de rotas em Node.js permite a definição clara e organizada dos endpoints da API. Cada rota corresponderá a uma operação específica (por exemplo, criação, leitura, atualização, exclusão) sobre os dados, garantindo uma comunicação eficaz entre o frontend e a base de dados.

4. Autenticação e Segurança

A segurança é um aspecto crucial no desenvolvimento de aplicações web. No Projeto II, a autenticação será implementada desta vez de uma forma segura, para garantir que apenas utilizadores autorizados possam aceder à zona privada da aplicação.

A continuação do Projeto I com a implementação de uma base de dados PostgreSQL e a utilização de Prisma para manipulação dos dados representa um avanço significativo na criação de uma aplicação web robusta e funcional. A combinação destas tecnologias, juntamente com uma estrutura de frontend moderna e segura, permitirá a construção de um sistema eficiente, escalável e seguro, atendendo às necessidades dos utilizadores e garantindo uma experiência de utilizador satisfatória.

Com este estado da arte, estabelecemos a base teórica e tecnológica para o desenvolvimento do Projeto II, focando em práticas modernas de desenvolvimento web e integração de tecnologias que suportam a criação de aplicações complexas e de alto desempenho.

Caso de Uso

Fluxo:

1. O utilizador interage com a interface do utilizador.
2. O utilizador faz login através do formulário de login.
3. O front end envia uma requisição de autenticação à API.
4. A API valida as credenciais e gera um token JWT se forem válidas.
5. O token JWT é armazenado no cliente para autenticação de futuras requisições.
6. O utilizador autenticado pode aceder as páginas CRUD.
7. As páginas CRUD enviam requisições à API para operações de criação, leitura, atualização e exclusão na base de dados.
8. A API usa Prisma para interagir com o PostgreSQL e realizar as operações solicitadas.

Figura 1 - Caso de Uso

Estrutura do Site

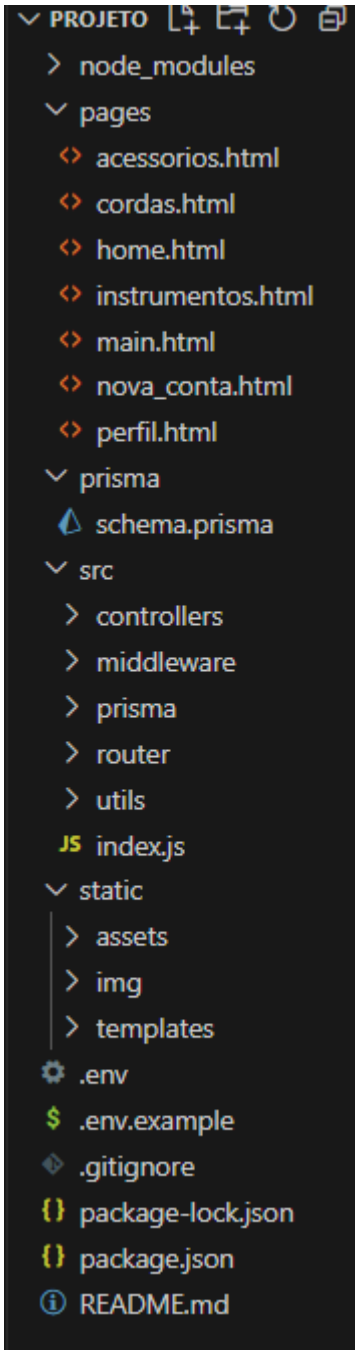


Figura 2 - Sistema de Ficheiros

O diretório node_modules contém todas as dependências e pacotes instalados via npm (Node Package Manager), necessários para o funcionamento da aplicação. No diretório pages, encontram-se as várias páginas HTML da aplicação.

O diretório prisma contém ficheiros relacionados ao Prisma, uma ferramenta de ORM (Object-Relational Mapping), sendo o schema.prisma o arquivo que define o esquema da base de dados.

O diretório src é o principal diretório do código-fonte da aplicação e está subdividido em várias partes:

- controllers: Diretório que contém a lógica de receber pedidos HTTP e os redirecionar para os serviços.
- middleware: Inclui a parte de autenticação e autorização de login de utilizadores.
- prisma: Contém a configuração e inicialização do Prisma.
- router: Inclui as rotas da aplicação.
- utils: Contém funções que verificam as palavras-passe.
- index.js: O ficheiro de entrada para a aplicação.

O diretório static contém ficheiros estáticos como imagens e estilos CSS, e está subdividido em:

- assets: Contém ficheiro json, estilos css e uns scripts em java utilizados ao longo do projeto.
- img: Contém imagens.
- templates: Contém templates HTML.

O ficheiro. env é utilizado para a configuração de variáveis de ambiente. O package-lock.json regista as versões exatas das dependências instaladas, garantindo que a instalação seja reproduzível, e o package.json contém metadados sobre o projeto, incluindo dependências e scripts de execução, entre outras configurações. Por fim, o **README.md** é um ficheiro de documentação que contém informações sobre a aplicação.

Desenho das Páginas

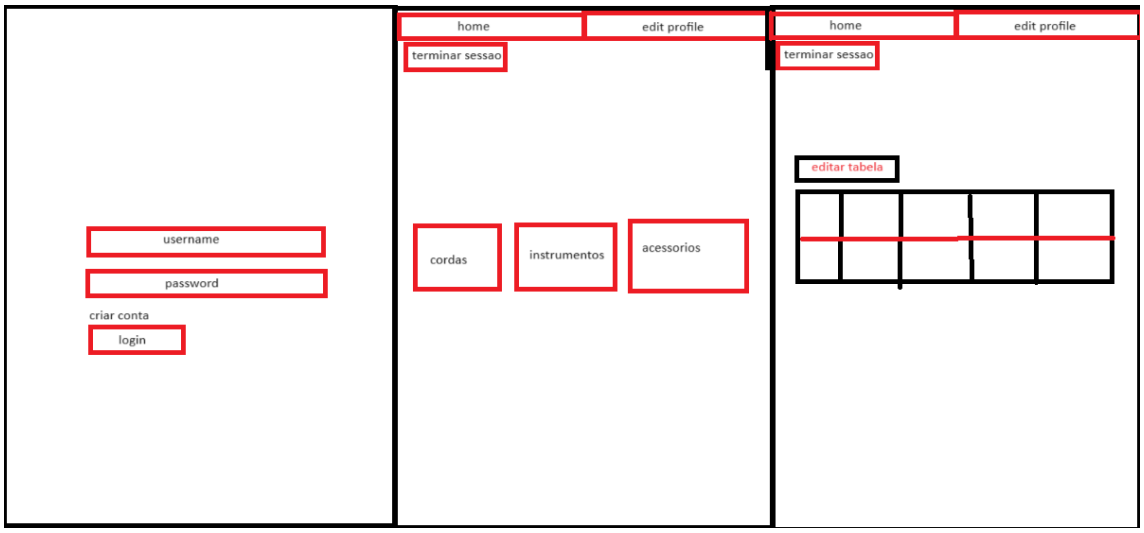


Figura 3 - Desenho Inicial de Páginas

Base de Dados

Modelo Entidade Relacional

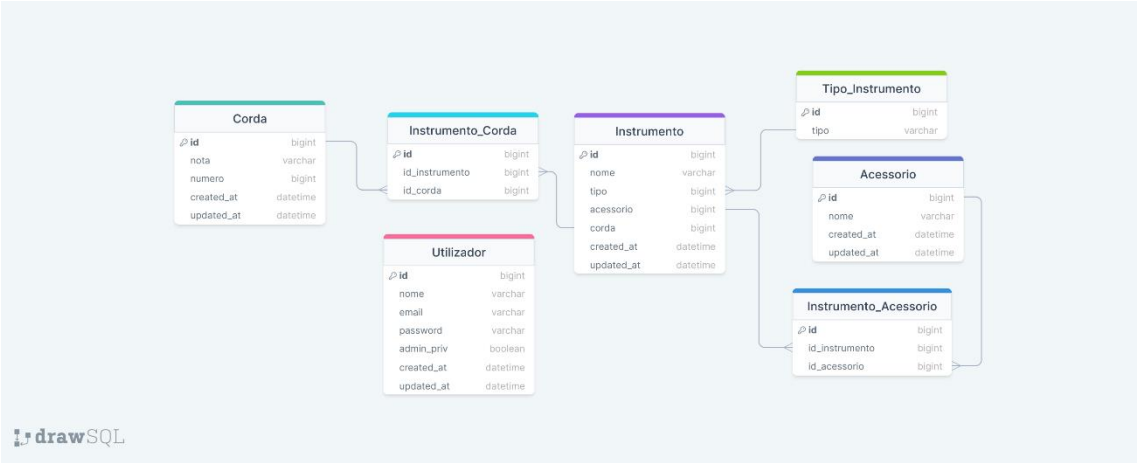


Figura 4 Modelo Entidade Relacional da Base de Dados.

Prisma- Criação de Tabelas

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}
```

```
model Tipo_Instrumento {
  instrumento_id Int    @id @default(autoincrement())
  nome           String @unique

  instrumento Instrumento[]

  @@map("tipo_instrumento")
}

model Acessorio {
  id        Int      @id @default(autoincrement())
  nome      String
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  instrumento Tipo_Acessorio[]

  @@map("acessorio")
}

model Tipo_Acessorio {
  instrumento_id Int
  acessorio_id   Int

  acessorio   Acessorio   @relation(fields: [acessorio_id], references: [id])
  instrumento Instrumento @relation(fields: [instrumento_id], references: [id])

  @@id([instrumento_id, acessorio_id])
  @@map("tipo_acessorio")
}
```

```
model Utilizador {
  id         Int      @id @default(autoincrement())
  nome       String
  email      String   @unique
  password   String
  admin_priv Boolean  @default(false)
  createdAt  DateTime @default(now())
  updatedAt  DateTime @updatedAt

  @@map("utilizador")
}

model Corda {
  id        Int      @id @default(autoincrement())
  nota      String
  numero    Int
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  instrumento Instrumento_Corda[]

  @@map("corda")
}

model Instrumento_Corda {
  instrumento_id Int
  corda_id       Int

  instrumento Instrumento @relation(fields: [instrumento_id], references: [id])
  corda       Corda       @relation(fields: [corda_id], references: [id])

  @@id([instrumento_id, corda_id])
  @@map("instrumento_corda")
}

model Instrumento {
  id                    Int                @id @default(autoincrement())
  nome                  String
  tipo                  Int
  createdAt             DateTime           @default(now())
  updatedAt             DateTime           @updatedAt
  corda                 Instrumento_Corda[]
  tipo_instrumento      Tipo_Instrumento   @relation(fields: [tipo], references: [instrumento_id])
  acessorio_instrumento Tipo_Acessorio[]

  @@map("instrumento")
}
```

Figura 5 - Criação de tabelas no Prisma

PostgresSQL

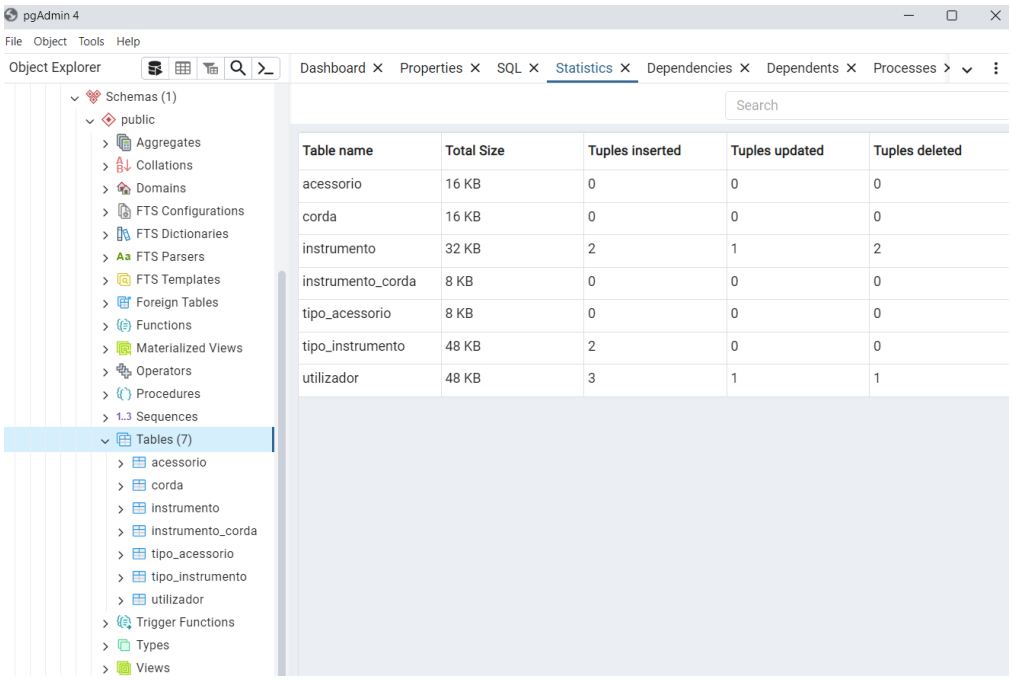


Figura 6 - Base de Dados no pgAdmin4

Páginas web

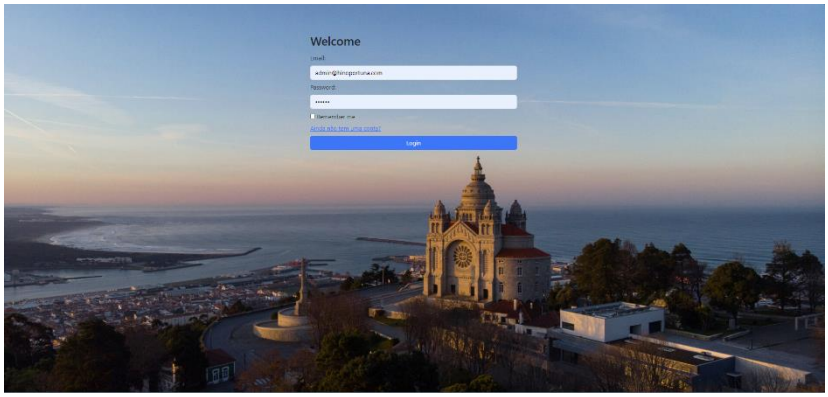


Figura 8 - Página de login

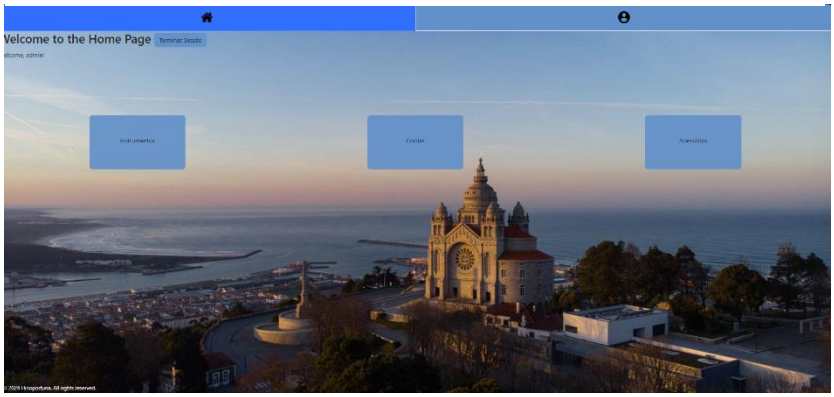


Figura 9 - Página principal

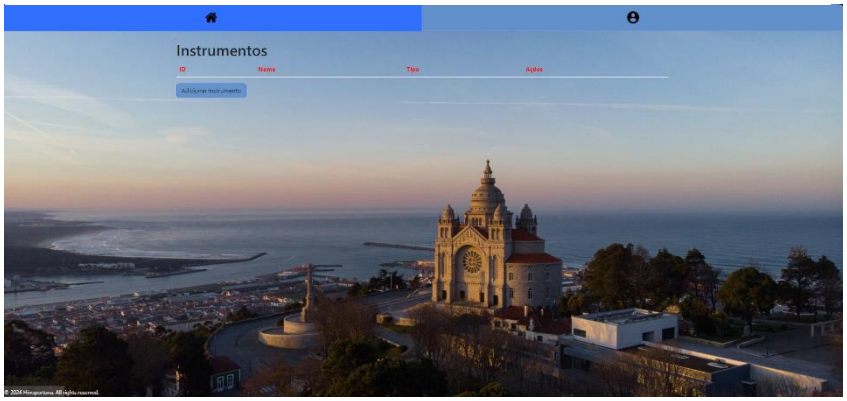


Figura 10 - Página da tabela CRUD

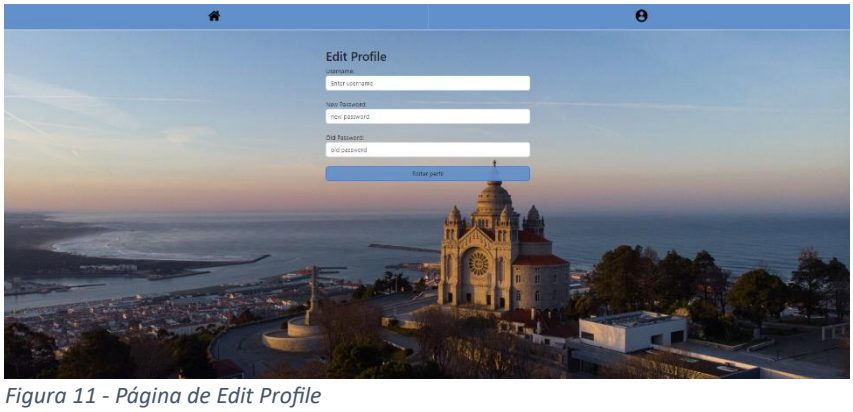


Figura 11 - Página de Edit Profile

API e as suas rotas

A nossa API é composta por várias rotas que desempenham diversas funções essenciais. As rotas relacionadas a acessórios, cordas e instrumentos são bastante semelhantes na sua estrutura, diferenciando-se apenas por certos campos específicos. O seu propósito é permitir a criação, atualização e exclusão de novos objetos. É importante notar que todas estas operações exigem não apenas a autenticação do utilizador, mas também privilégios de administrador para serem executadas com sucesso.

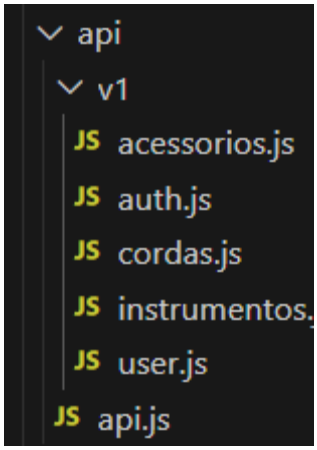


Figura 12 - Ficheiros da API

Para gerir a autenticação dos utilizadores, a nossa API conta com a rota "auth.js", que desempenha um papel crucial. Além de fornecer funções para login, registro e logout, esta rota garante a segurança do sistema, verificando as credenciais dos utilizadores e concedendo acesso conforme necessário.

Em complemento, temos a rota de "user.js", dedicada à gestão das informações dos utilizadores. Aqui, os utilizadores podem visualizar e atualizar os seus dados pessoais conforme necessário, proporcionando uma experiência de utilizador completa e personalizada.

Por fim temos a rota de "api.js" que organiza e delega as requisições HTTP para diferentes rotas. Isto permite uma estrutura modular e organizada para o tratamento de diferentes partes da API.

Autenticação e Autorização

A nossa API implementa uma série de middlewares que desempenham funções cruciais no controle de acesso e autenticação dos utilizadores. Cada middleware possui um papel específico e contribui para a segurança e funcionalidade geral do sistema.

Figura 13 - Ficheiros do middleware

O "onlyAdminMiddleware" é responsável por garantir que apenas utilizadores com privilégios de administrador tenham acesso a determinadas rotas. Caso um utilizador sem as devidas permissões tentar aceder a essas rotas, o middleware responde com um código de erro 403, indicando a falta de autorização para a ação solicitada.

O "authenticationMiddleware" desempenha um papel fundamental na verificação da autenticidade dos tokens JWT (JSON Web Tokens) enviados com as requisições. Além de validar os tokens, este middleware também anexa as informações do utilizador autenticado ao pedido, permitindo que a API tenha conhecimento do contexto do utilizador. Se a verificação do token falhar, o middleware responde com um código de erro 401, indicando falta de autenticação, ou redireciona o utilizador para a página inicial.

Por fim, o "noAuthenticationMiddleware" é responsável por verificar se um token JWT está presente em determinadas rotas que devem ser acessíveis apenas por utilizadores não autenticados (por exemplo, páginas para convidados). Se um token válido for encontrado, o middleware redireciona o utilizador para uma página específica ou responde com um código de erro 401, indicando que a autenticação não é permitida para a ação solicitada.

Código JS

```
const jwt = require("jsonwebtoken");

const JWT_SECRET = process.env.JWT_SECRET || "jwt_secret";

const ttl = 60 * 60 * 24; // 1 day
const ttlMs = ttl * 1000;

function generateToken(payload) {
  return jwt.sign(payload, JWT_SECRET, { expiresIn: ttl });
}

function verifyToken(token) {
  return jwt.verify(token, JWT_SECRET);
}

module.exports = {
  generateToken,
  verifyToken,
  ttl,
  ttlMs,
};
```

Figura 14 – JWToken generator

Esta código permite a criação e verificação de tokens JWT de forma segura, utilizando uma chave secreta para assinatura e verificação. Os tokens gerados possuem um tempo de vida definido, garantindo que expiram após um determinado período de tempo.

```
const SALT_ROUNDS = 10;

function hashPassword(password) {
  return bcrypt.hashSync(password, SALT_ROUNDS);
}

function comparePassword(password, hash) {
  return bcrypt.compareSync(password, hash);
}

module.exports = {
  hashPassword,
  comparePassword,
};
```

Figura 15 - Password hashing

Estas funções são úteis para garantir a segurança das senhas dos utilizadores numa aplicação web, utilizando técnicas de hashing fornecidas pela biblioteca `bcryptjs`.. A função `comparePassword` é útil para verificar se uma senha fornecida corresponde ao hash armazenado no banco de dados durante o processo de autenticação do usuário.

```
require("dotenv").config(); // Configurar as Variáveis Ambiente
const cors = require("cors"); // Buscar o pacote "cors"
const morgan = require("morgan"); // Buscar o pacote "morgan"
const express = require("express"); // Buscar o pacote "express"
const cookieParser = require("cookie-parser"); // Buscar o pacote "cookie-parser"
const router = require("./router/router"); // Buscar o ficheiro "router.js"

const app = express(); // Inicialização da aplicação express

app.use(express.json()); // Receber e enviar dados em formato JSON
app.use(cookieParser()); // Utilizar o cookie-parser na aplicação express
app.use(morgan("tiny")); // Utilizar o morgan na aplicação express
app.use(cors()); // Utilizar o cors na aplicação express

app.use(router);

const port = process.env.SERVER_PORT || 3000; // Buscar "SERVER_PORT" ao ficheiro .env ou utiliza a 3000 por predefinição

// Iniciar o Servidor Express na porta
app.listen(port, () => {
  console.log(`Server open at: http://localhost:${port}`);
});
```

Figura 16 - Configuração do servidor

Este último código serve para configurar o servidor Express (node.js) com os middlewares necessários para analisar solicitações HTTP, gerar logs, lidar com cookies e habilitar o CORS, além de definir e utilizar rotas para gerenciar requisições http.

Implementação da aplicação web no Vercel

O Vercel é uma plataforma de Hosting e implementação de projetos web, focada principalmente em frameworks e bibliotecas JavaScript como Next.js, React, Vue.js e outros. É conhecido por permitir que os desenvolvedores façam deploy de suas aplicações de forma rápida e fácil. Ele oferece otimizações de desempenho automáticas, como renderização estática e caching, para garantir que as aplicações sejam rápidas e responsivas. O Vercel é fácil de integrar com repositórios GitHub, GitLab e Bitbucket, facilitando o fluxo de trabalho de desenvolvimento e deployment. Para cada pull reques o Vercel gera automaticamente uma URL de visualização, permitindo que equipas revejam as mudanças antes de as publicar. Infelizmente a nossa tentativa de implementação terminou em fracasso.

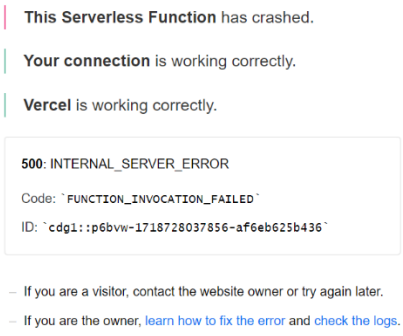


Figura 17 - tentativa de implementação no Vercel

Conclusão

Ao longo deste trabalho, exploramos diversas tecnologias e metodologias fundamentais para o desenvolvimento de aplicações web modernas e funcionais.

A integração de uma API e uma base de dados PostgreSQL no projeto Hinoportuna representou um passo significativo em direção à construção de um sistema mais robusto e dinâmico. A transição para uma base de dados relacional funcional proporcionou maior segurança e escalabilidade, substituindo o uso anterior de JSON. Por meio da utilização de tecnologias como Node.js, Prisma e Express(node.js), conseguimos redefinir e aprimorar a estrutura do projeto, garantindo uma gestão de dados mais eficiente e segura.

A implementação de rotas e middlewares para autenticação e autorização na API permitiu um melhor controlo sobre o acesso às funcionalidades do sistema. A utilização de tokens JWT e a definição de middlewares específicos contribuíram para uma experiência de utilizador mais segura e personalizada, adaptada às necessidades de autenticação e privacidade dos utilizadores.

Além disso, a exploração de tecnologias frontend como HTML, CSS, Bootstrap e JavaScript, em conjunto com as práticas de desenvolvimento de interfaces modernas, proporcionou uma base sólida para a criação de uma interface de utilizador atrativa e funcional. A integração dessas tecnologias com a API e a base de dados permitiu uma comunicação fluida entre o frontend e o backend, proporcionando uma experiência de utilizador coesa e integrada.

Este relatório, juntamente com o trabalho desenvolvido ao longo deste projeto, representa um marco importante na esperamos nós conclusão da disciplina, mas também uma base sólida para futuros projetos e aprendizagens no campo do desenvolvimento de aplicações web. Com a conclusão deste trabalho, estamos preparados para enfrentar novos desafios e continuar a crescer e evoluir como alunos e futuros profissionais.

Referências

1. Lira Fernandes, A. 2024. Apresentações Programação Web [Material de aula]. Moodle. https://elearning.ipvc.pt/ipvc2023/course/view.php?id=1395
2. Prisma. (s.d.). Prisma Documentation. Retirado de https://www.prisma.io/docs Acedido a 7 de junho de 2024.
3. PostgreSQL. (s.d.). PostgreSQL Documentation. Retirado de https://www.postgresql.org/docs/ . Acedido a 7 de junho de 2024.
4. DigitalOcean. How To Build a REST API with Prisma and PostgreSQL. Recuperado de https://www.digitalocean.com/community/tutorials/how-to-build-a-rest-api-with-prisma-and-postgresql Acedido a 8 de junho de 2024.