# Relatório de Desenvolvimento do Projeto Backend - Creatus

Rodrigo Sandler

Julho 2024

# Contents



# 1 Introdução

Este relatório descreve detalhadamente o desenvolvimento do projeto backend para a entrevista na Creatus. O projeto inclui funcionalidades de autenticação, CRUD de usuários e geração de relatórios. O objetivo deste documento é esclarecer todas as decisões técnicas tomadas durante o desenvolvimento.

# 2 Descrição do Projeto

Este é um projeto backend desenvolvido para a empresa Creatus, focado no gerenciamento de usuários. O projeto utiliza Node.js com TypeScript e Prisma ORM para interação com um banco de dados MongoDB. As principais funcionalidades incluem autenticação JWT, operações CRUD para usuários e geração de relatórios em formato CSV.

# 3 Estrutura do Projeto

```
creatus-backend/
 node_modules/
 src/
    config/
       db.ts
    controllers/
       authController.ts
       userController.ts
    middleware/
       auth.ts
    prisma/
       client.ts
       schema.prisma
    routes/
       authRoutes.ts
```

```
      userRoutes.ts
   types/
      AuthenticateRequest.ts
      express.d.ts
      User.ts
   utils/
      csvGenerator.ts
   app.ts
   server.ts
.env
.gitignore
package-lock.json
package.json
README.md
tsconfig.json
```

# 4 Tomada de Decisões

## 4.1 Organização das Rotas

Para melhorar a organização e a manutenibilidade do código, decidi dividir as rotas em arquivos separados. Isso permite uma melhor modularização e facilita futuras expansões ou modificações.

- **authRoutes.ts**: Contém as rotas relacionadas à autenticação.
- **userRoutes.ts**: Contém as rotas relacionadas ao CRUD de usuários e à geração de relatórios.

## 4.2 Hashing de Senhas

Para garantir a segurança das senhas dos usuários, implementei hashing utilizando a biblioteca `bcryptjs`. Dessa forma, apenas o hash da senha é armazenado no banco de dados, protegendo as senhas em caso de vazamento de dados.

Listing 1: Hashing de Senhas

```
const salt = await bcrypt.genSalt(10);
const hashedPassword = await bcrypt.hash(password, salt);
```

## 4.3 Configuração do Prisma e Conexão com o Banco de Dados

O Prisma foi configurado para facilitar a interação com o banco de dados MongoDB. A configuração está no arquivo `schema.prisma` e a conexão é gerenciada no arquivo `client.ts`.

Listing 2: Configuração do Prisma

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

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

## 4.4 Middleware de Autenticação

O middleware de autenticação foi implementado para proteger rotas que requerem autenticação. Ele verifica a validade do token JWT e adiciona os dados do usuário à requisição.

Listing 3: Middleware de Autenticação

```
const authMiddleware = (req: AuthenticatedRequest, res: Response, next: NextFunction) =>
  const token = req.header('x-auth-token');
  if (!token) {
    return res.status(401).json({ msg: 'Nenhum token, autoriza  o negada' });
  }
  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET) as UserPayload;
    req.user = decoded.user;
    next();
  } catch (err) {
    res.status(401).json({ msg: 'Token inv lido' });
  }
};
```

## 4.5 Geração de Relatórios

A geração de relatórios em formato CSV foi implementada utilizando a biblioteca `json2csv`. Apenas usuários com nível de acesso adequado podem gerar relatórios.

Listing 4: Geração de Relatórios

```
const generateReport = async (req: AuthenticatedRequest, res: Response) => {
  if (!req.user || req.user.level < 4) {
    return res.status(403).json({ msg: 'Acesso negado' });
  }
  const users = await prisma.user.findMany();
  const csv = generateCSV(users);
  res.header('Content-Type', 'text/csv');
  res.attachment('report.csv');
  res.send(csv);
};
```

## 4.6 Descrição Detalhada dos Arquivos

### 4.6.1 Config

**db.ts** Este arquivo configura a conexão com o banco de dados MongoDB usando Prisma. Ele importa o cliente Prisma e o configura para uso em toda a aplicação.

**Funções/Métodos**:

- Configuração da conexão com o Prisma.

### 4.6.2 Controllers

**authController.ts** Este arquivo contém a lógica para autenticação dos usuários. Ele lida com a rota de login, verifica as credenciais do usuário e gera um token JWT.

**Métodos**:

- **login**: Verifica as credenciais e gera um token JWT.

**userController.ts** Este arquivo contém a lógica para as operações CRUD de usuários e a geração de relatórios. Ele lida com as rotas de criação, leitura, atualização, exclusão de usuários e geração de relatórios em CSV.

**Métodos**:

- **createUser**: Cria um novo usuário.
- **getUsers**: Retorna todos os usuários.
- **getUserById**: Retorna um usuário específico pelo ID.

- **updateUser**: Atualiza um usuário específico.
- **deleteUser**: Exclui um usuário específico.
- **generateReport**: Gera um relatório CSV de todos os usuários.

#### 4.6.3 Middleware

**auth.ts** Este arquivo contém o middleware de autenticação que verifica a validade do token JWT e adiciona os dados do usuário à requisição.
**Funções/Métodos**:

- **authMiddleware**: Middleware para verificar o token JWT.

#### 4.6.4 Prisma

**client.ts** Este arquivo configura e exporta o cliente Prisma para ser usado em toda a aplicação.
**Funções/Métodos**:

- Configuração e exportação do cliente Prisma.

**schema.prisma** Este arquivo define o esquema do banco de dados utilizando Prisma.
**Modelo**:

- **User**: Modelo de usuário com campos id, name, email, password, e level.

#### 4.6.5 Routes

**authRoutes.ts** Este arquivo define as rotas relacionadas à autenticação.
**Rotas**:

- **POST /api/auth/login**: Rota para login do usuário.

**userRoutes.ts** Este arquivo define as rotas relacionadas ao CRUD de usuários e à geração de relatórios.
**Rotas**:

- **POST /api/users**: Rota para criar um novo usuário.
- **GET /api/users**: Rota para listar todos os usuários.
- **GET /api/users/:id**: Rota para obter detalhes de um usuário específico.
- **PUT /api/users/:id**: Rota para atualizar um usuário específico.
- **DELETE /api/users/:id**: Rota para deletar um usuário específico.
- **GET /api/users/report**: Rota para gerar um relatório em CSV dos usuários.

#### 4.6.6 Types

**AuthenticateRequest.ts** Este arquivo define o tipo de requisição autenticada, que inclui os dados do usuário.
**Tipos**:

- **AuthenticatedRequest**: Extensão da interface Request que inclui o usuário.

**express.d.ts** Este arquivo adiciona o tipo de usuário à interface de Request do Express.
**Tipos**:

- Adição do tipo usuário à interface Request do Express.

**User.ts** Este arquivo define a interface para o modelo de usuário.
**Interface**:

- **User**: Interface que define os campos id, name, email, password, e level.

#### 4.6.7 Utils

**csvGenerator.ts** Este arquivo contém a função que gera um arquivo CSV a partir de uma lista de usuários.

**Funções/Métodos**:

- **generateCSV**: Gera um arquivo CSV a partir de uma lista de usuários.

#### 4.6.8 Arquivos Principais

**app.ts** Este arquivo inicializa a aplicação Express e configura os middlewares.

**Funções/Métodos**:

- Configuração de middlewares (cors, express.json).
- Definição das rotas principais (authRoutes, userRoutes).

**server.ts** Este arquivo inicia o servidor na porta especificada.

**Funções/Métodos**:

- Conexão ao banco de dados.
- Inicialização do servidor.

### 4.7 Outros Aspectos Relevantes

Além das decisões mencionadas acima, também utilizei variáveis de ambiente para armazenar informações sensíveis, como a URL do banco de dados e a chave secreta do JWT. Isso aumenta a segurança do sistema ao evitar que essas informações sejam expostas no código-fonte.

## 5 Conclusão

Este relatório detalha todas as decisões técnicas tomadas durante o desenvolvimento do projeto backend para a Creatus. Espero que este documento esclareça quaisquer dúvidas sobre o projeto. Agradeço pela oportunidade de participar do processo seletivo e estou à disposição para quaisquer perguntas adicionais.