Desenvolvimento de Backend para a aplicação Fidelis

Thiago Freire Cavalcante, Ivan Barroca Neto

January 15, 2025

Unidade 3

Contents

1 Back-end

1.1 server.js

Depois de criar o arquivo .env, que contém as variáveis de ambiente necessárias para o funcionamento da aplicação, vamos criar o arquivo principal server.js.

```
const express = require('express');
const cors = require('cors');
const mongoose = require('mongoose');
require('dotenv').config();
```

- express: Framework para criar e gerenciar o servidor HTTP.

- cors: Middleware para habilitar o CORS (Cross-Origin Resource Sharing), permitindo que o backend aceite requisições de diferentes origens.

- mongoose: Biblioteca para trabalhar com MongoDB em JavaScript, oferecendo um modelo orientado a objetos.

- dotenv: Carrega as variáveis do arquivo .env para process.env.

```
const authRoutes = require('./routes/auth');
const lojaRoutes = require('./routes/loja');
const userRoutes = require('./routes/user');
```

Aqui, são importados arquivos de rotas localizados na pasta routes. Essas rotas gerenciam diferentes funcionalidades:

- authRoutes: Gerencia autenticação, como login e registro.

- lojaRoutes: Gerencia operações relacionadas às lojas.

- userRoutes: Gerencia operações relacionadas aos usuários.

```
const app = express();
```

Cria uma instância do Express, que será usada para configurar e iniciar o servidor.

```
app.use(cors());
app.use(express.json());
```

- cors(): Permite que o backend aceite requisições de outros domínios (útil em APIs públicas).

- express.json(): Middleware para processar o corpo das requisições no formato JSON.

```
mongoose.connect(process.env.MONGODB_URI)
  .then(() => console.log('Conectado ao MongoDB'))
```

```
3         .catch(err => console.error('Erro ao conectar ao
              MongoDB:', err));
```

- mongoose.connect: Conecta o backend ao banco de dados MongoDB, usando a URI fornecida na variável de ambiente $MONGODB_URI$ do arquivo .env.
- .then() e .catch(): Gerenciam o sucesso ou falha da conexão.

```
1       app.use('/api/auth', authRoutes);
2       app.use('/api/lojas', lojaRoutes);
3       app.use('/api/users', userRoutes);
```

Define as rotas da aplicação: - /api/auth: Redireciona as requisições relacionadas à autenticação para o authRoutes.
- /api/lojas: Redireciona requisições relacionadas às lojas.
- /api/users: Redireciona requisições relacionadas aos usuários.

```
1       const PORT = process.env.PORT || 3000;
2       app.listen(PORT, () => {
3         console.log(`Servidor rodando na porta ${PORT}`);
4       });
```

- PORT: Define a porta do servidor, priorizando a porta especificada na variável de ambiente PORT. Caso não esteja definida, usa a porta 3000 como padrão.
- app.listen: Inicia o servidor e exibe uma mensagem de sucesso no console.

1.2 Modelo da coleção Loja

Vamos criar o modelo de dados para a coleção de lojas.

```
1       const mongoose = require('mongoose');
```

- mongoose: Biblioteca para trabalhar com MongoDB em JavaScript, oferecendo um modelo orientado a objetos.

```
1           const lojaSchema = new mongoose.Schema({
2       nome: {
3           type: String,
```

```
        required: true
    },
    categoria: {
        type: String,
        required: true
    },
    localizacao: {
        type: String,
        required: true
    },
    descricao: String,
    programas: [{
        nome: String,
        progresso: Number,
        maxCarimbos: Number
    }],
    banner: String,
    logo: String,
    horario: String,
    proprietario: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User'
    }
    }, {
    timestamps: true
    });
```

1.3 Controller da coleção Loja

Vamos criar o controller para manipular as operações relacionadas às lojas no backend, usando o modelo Loja previamente definido. Esses controladores lidam com as ações de criação, consulta, atualização e manipulação de dados relacionados às lojas.

```
        exports.criarLoja = async (req, res) => {
    try {
        const loja = new Loja({
        ...req.body,
        proprietario: req.user._id
        });
        await loja.save();
        res.status(201).json(loja);
    } catch (error) {
        res.status(400).json({ message: error.message });
    }
    };
```

Objetivo: Criar uma nova loja e armazená-la no banco de dados. Como funciona:

- Recebe os dados da loja no corpo da requisição (req.body).
- Adiciona o ID do proprietário ($req.user._id$) ao documento da loja. O proprietário deve ser o usuário autenticado.
- Salva a loja no banco de dados com loja.save().
- Retorna o documento criado em resposta com o código HTTP 201 (Criado).
- Em caso de erro, retorna um código 400 (Requisição inválida) com a mensagem do erro.

```
1          exports.buscarLojas = async (req, res) => {
2      try {
3          const lojas = await Loja.find();
4          res.json(lojas);
5      } catch (error) {
6          res.status(500).json({ message: error.message });
7      }
8      };
```

Objetivo: Buscar todas as lojas cadastradas. Como funciona:

- Usa Loja.find() para recuperar todas as lojas do banco de dados.
- Retorna as lojas em formato JSON.
- Em caso de erro, retorna um código 500 (Erro interno do servidor) com a mensagem do erro.

```
1          exports.buscarLojaPorId = async (req, res) => {
2      try {
3          const loja = await Loja.findById(req.params.id);
4          if (!loja) {
5          return res.status(404).json({message: 'Loja n o
               encontrada'});
6          }
7          res.json(loja);
8      } catch (error) {
9          res.status(500).json({ message: error.message });
10     }
11     };
```

Objetivo: Buscar uma loja específica pelo ID. Como funciona: - Usa Loja.findById(req.params.id)
para recuperar a loja com o ID fornecido.

- Se a loja não for encontrada, retorna um código 404 (Não encontrado).
- Retorna a loja encontrada em formato JSON.

- Em caso de erro, retorna um código 500 (Erro interno do servidor) com a mensagem do erro.

```
exports.atualizarProgresso = async (req, res) => {
    try {
      const { lojaId, programaId, novoProgresso } = req.body;
      const loja = await Loja.findById(lojaId);

      if (!loja) {
        return res.status(404).json({ message: 'Loja não encontrada' });
      }

      const programa = loja.programas.id(programaId);
      if (!programa) {
        return res.status(404).json({ message: 'Programa não encontrado' });
      }

      programa.progresso = novoProgresso;
      await loja.save();

      res.json(programa);
    } catch (error) {
      res.status(400).json({ message: error.message });
    }
  };
```

Objetivo: Atualizar o progresso de um programa específico em uma loja. Como funciona: - Recebe o ID da loja, o ID do programa e o novo progresso no corpo da requisição (req.body).

- Usa Loja.findById(lojaId) para recuperar a loja com o ID fornecido.
- Verifica se a loja existe.
- Usa loja.programas.id(programaId) para encontrar o programa com o ID fornecido dentro da loja.
- Atualiza o progresso do programa com o novo valor fornecido.
- Salva a loja atualizada com loja.save().
- Retorna o programa atualizado em formato JSON.
- Em caso de erro, retorna um código 400 (Requisição inválida) com a mensagem do erro.