

Material de apoio

Site: [Geração Tech](#)
Curso: Formação em Desenvolvedor Web - Online
Livro: Material de apoio

Impresso por: JOÃO VITOR DE MELO FREITAS
Data: sexta-feira, 2 ago. 2024, 08:44

Índice

1. Middleware

- 1.1. Estrutura do Projeto
- 1.2. Vídeo Aula
- 1.3. Middleware para Autenticação
- 1.4. Vídeo Aula
- 1.5. Implementando JWT para Autenticação
- 1.6. Vídeo Aula
- 1.7. Finalizando a Configuração com Dotenv
- 1.8. Vídeo Aula

1. Middleware

O que é Middleware?

Middleware é um intermediário que fica entre a requisição do cliente e a resposta do servidor.

Ele pode ser usado para diversas finalidades, como autenticação, registro de logs, tratamento de erros, entre outros.

Como Funciona o Middleware?

O middleware funciona interceptando a requisição antes que ela chegue à rota final.


Ele pode realizar operações como verificar se o usuário está autenticado, registrar informações de acesso e muito mais.

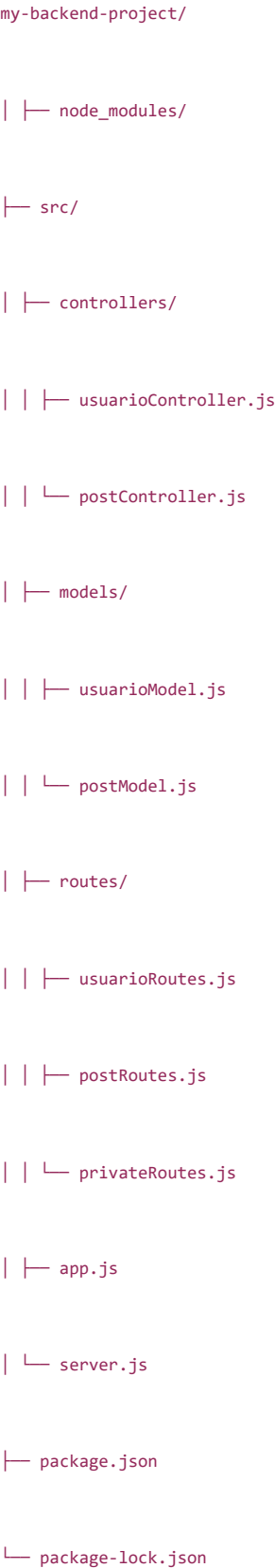
1.1. Estrutura do Projeto

Vamos adicionar uma camada de middleware à nossa aplicação para proteger as rotas privadas.

Estrutura Atual

go

 Copiar código




Criando as Rotas Privadas

privateRoutes.js

Vamos criar um novo arquivo `privateRoutes.js` dentro da pasta `routes`.

javascript

 Copiar código


```
const express = require('express'); const router = express.Router(); const usuarioRoutes = require('./usuarioRoutes'); const postRoutes = require('./postRoutes'); // Middleware de autenticação router.use((req, res, next) => { console.log('Middleware de autenticação'); const autorizado = true; // Lógica de autorização if (autorizado) { next(); } else { res.status(403).json({ mensagem: 'Não autorizado' }); } }); // Rotas privadas router.use('/usuarios', usuarioRoutes); router.use('/posts', postRoutes); module.exports = router;
```

Integrando as Rotas Privadas no Servidor

server.js

Vamos atualizar o arquivo `server.js` para incluir as novas rotas privadas.

javascript

 Copiar código

```
const express = require('express'); const app = express(); const usuarioRoutes = require('./routes/usuarioRoutes'); const postRoutes = require('./routes/postRoutes'); const privateRoutes = require('./routes/privateRoutes'); app.use(express.json()); app.use('/api', privateRoutes); const PORT = 3000; app.listen(PORT, () => { console.log(`Servidor rodando na porta ${PORT}`); });
```

Testando com Insomnia

Configurando Insomnia

1. Listar Posts:

- Método: GET
- URL: `http://localhost:3000/api/posts`

2. Consultar Post por ID:

- Método: GET
- URL: `http://localhost:3000/api/posts/1`

3. Criar Post:

- Método: POST
- URL: `http://localhost:3000/api/posts`
- Body (JSON):

json

 Copiar código

```
{ "id": 3, "titulo": "Novo Post", "conteudo": "Conteúdo do novo post", "userId": 1 }
```

4. Atualizar Post:

- Método: PUT
- URL: `http://localhost:3000/api/posts/1`
- Body (JSON):

json

 Copiar código

```
{ "titulo": "Post Atualizado", "conteudo": "Conteúdo atualizado do post" }
```

5. Deletar Post:

- Método: DELETE
- URL: `http://localhost:3000/api/posts/1`

Implementando a Autenticação Simples

No middleware que criamos em `privateRoutes.js`, podemos implementar uma lógica simples de autenticação.

javascript

 Copiar código

```
// Middleware de autenticação router.use((req, res, next) => { console.log('Middleware de autenticação'); const token = req.headers['authorization']; if (token === 'meu-token-secreto') { next(); } else { res.status(403).json({ mensagem: 'Não autorizado' }); } });
```

Testando com Insomnia

Vamos adicionar um cabeçalho `Authorization` com o valor `meu-token-secreto` para testar a autenticação.

1. Listar Posts:

- Método: GET
- URL: `http://localhost:3000/api/posts`
- Headers: `Authorization: meu-token-secreto`


2. Consultar Post por ID:

- Método: GET
- URL: `http://localhost:3000/api/posts/1`
- Headers: `Authorization: meu-token-secreto`

3. Criar Post:

- Método: POST
- URL: `http://localhost:3000/api/posts`
- Headers: `Authorization: meu-token-secreto`
- Body (JSON):

json


 Copiar código

```
{ "id": 3, "titulo": "Novo Post", "conteudo": "Conteúdo do novo post", "userId": 1 }
```

4. Atualizar Post:

- Método: PUT
- URL: `http://localhost:3000/api/posts/1`
- Headers: `Authorization: meu-token-secreto`
- Body (JSON):

json

 Copiar código

```
{ "titulo": "Post Atualizado", "conteudo": "Conteúdo atualizado do post" }
```

5. Deletar Post:

- Método: DELETE
- URL: `http://localhost:3000/api/posts/1`
- Headers: `Authorization: meu-token-secreto`

Conclusão

Hoje, aprendemos sobre middleware no Express e como utilizá-lo para proteger rotas privadas.

Implementamos um middleware simples de autenticação que verifica um token de autorização antes de permitir o acesso às rotas.

Estudem e pratiquem bastante. Até a próxima aula!

1.2. Vídeo Aula

dia 8 express 1 converted



1.3. Middleware para Autenticação

Vamos continuar nossa aula de back-end e hoje vamos focar em como implementar autenticação utilizando middleware no Express.

Vamos entender como proteger rotas e garantir que apenas usuários autorizados possam acessar determinadas partes da nossa aplicação.

Relembrando Middleware

O que é Middleware?

Middleware é uma função que fica entre a requisição do cliente e a resposta do servidor. Ele pode ser usado para verificar autenticação, registrar logs, manipular dados da requisição, entre outros.

Como Funciona?

Quando uma requisição é feita, o middleware pode interceptá-la, realizar alguma lógica (como verificação de autenticação) e decidir se a requisição pode continuar para a próxima função ou ser bloqueada.

Implementando Middleware para Autenticação

Estrutura do Projeto

Vamos revisar a estrutura atual do projeto:

go



my-backend-project/

| └─ node_modules/

└─ src/

| └─ controllers/

| | └─ usuarioController.js

| | └─ postController.js

| └─ models/

| | └─ usuarioModel.js

| | └─ postModel.js

| └─ routes/ | └─ usuarioRoutes.js

| | └─ postRoutes.js

```
| | └─ privateRoutes.js
```

```
| | └─ publicRoutes.js
```

```
| └─ middlewares/
```

```
| | └─ authMiddleware.js
```

```
| └─ app.js
```

```
| └─ server.js
```

```
└─ package.json
```


```
└─ package-lock.json
```

Configurando o Middleware

Vamos criar um middleware para autenticação que verificará se o token é válido antes de permitir o acesso às rotas privadas.

authMiddleware.js

javascript

 Copiar código

```
const authMiddleware = (req, res, next) => { const token = req.headers['authorization']; if (token === 'meu-token-secreto') { next(); } else { res.status(403).json({ mensagem: 'Não autorizado' }); } }; module.exports = authMiddleware;
```

Configurando Rotas Privadas

Vamos configurar as rotas privadas para utilizar o middleware de autenticação.

privateRoutes.js

javascript

 Copiar código

```
const express = require('express'); const router = express.Router(); const authMiddleware = require('../middlewares/authMiddleware'); const usuarioRoutes = require('./usuarioRoutes'); const postRoutes = require('./postRoutes'); // Middleware de autenticação router.use(authMiddleware); // Rotas privadas router.use('/usuarios', usuarioRoutes); router.use('/posts', postRoutes); module.exports = router;
```

Configurando Rotas Públicas

Vamos criar rotas públicas para login e autenticação.

publicRoutes.js

javascript

 Copiar código

```
const express = require('express'); const router = express.Router(); const authController = require('../controllers/authController'); // Rota pública para login router.post('/login', authController.login);
```

```
module.exports = router;
```

authController.js

javascript

 Copiar código

```
const usuarios = [ { id: 1, nome: 'Admin', login: 'admin', senha: '123456' }, { id: 2, nome: 'User', login: 'user', senha: '123456' } ]; exports.login = (req, res) => { const { login, senha } = req.body; const usuario = usuarios.find(user => user.login === login && user.senha === senha); if (usuario) { res.json({ token: 'meu-token-secreto' }); } else { res.status(401).json({ mensagem: 'Login ou senha incorretos' }); } };
```

Configurando o Servidor

Vamos configurar o servidor para utilizar as rotas públicas e privadas.

server.js

javascript

 Copiar código

```
const express = require('express'); const app = express(); const publicRoutes = require('./routes/publicRoutes'); const privateRoutes = require('./routes/privateRoutes'); app.use(express.json()); app.use('/api', publicRoutes); app.use('/api', privateRoutes); const PORT = 3000; app.listen(PORT, () => { console.log(`Servidor rodando na porta ${PORT}`); });
```

Testando com Insomnia

Configurando Insomnia

1. Login:

- Método: POST
- URL: `http://localhost:3000/api/login`
- Body (JSON):

json

 Copiar código

```
{ "login": "admin", "senha": "123456" }
```

- Resposta esperada:

json

 Copiar código

```
{ "token": "meu-token-secreto" }
```


2. Listar Usuários (Autenticado):

- Método: GET
- URL: `http://localhost:3000/api/usuarios`
- Headers: `Authorization: meu-token-secreto`

3. Criar Post (Autenticado):

- Método: POST
- URL: `http://localhost:3000/api/posts`
- Headers: `Authorization: meu-token-secreto`
- Body (JSON):

json

 Copiar código

```
{ "titulo": "Novo Post", "conteudo": "Conteúdo do novo post" }
```

4. Listar Posts (Autenticado):

- Método: GET
- URL: `http://localhost:3000/api/posts`
- Headers: `Authorization: meu-token-secreto`

Conclusão

Hoje, aprendemos a implementar autenticação usando middleware no Express.

Criamos rotas públicas e privadas, configuramos middleware para autenticação e testamos nossa aplicação usando Insomnia.

Estudem e pratiquem bastante. Até a próxima aula!

1.4. Vídeo Aula

dia 8 express 2 converted



1.5. Implementando JWT para Autenticação

Hoje vamos continuar nossa aula de back-end e focar na implementação do JWT para autenticação.

O JWT (JSON Web Token) é amplamente utilizado para comunicação segura entre cliente e servidor, especialmente em autenticação.

O que é JWT?

JWT significa JSON Web Token.

É um padrão aberto (RFC 7519) que define uma forma compacta e segura de transmitir informações entre as partes como um objeto JSON.

Essas informações podem ser verificadas e confiáveis porque são assinadas digitalmente.

Vantagens do JWT


- **Compacto:** Pode ser enviado através de URL, parâmetros POST ou dentro de um cabeçalho HTTP.
- **Autenticado:** Assegura que a carga útil do token não foi alterada.
- **Seguro:** Pode usar um algoritmo de assinatura (HMAC ou RSA) para proteger os dados.

Implementação do JWT

Instalando a Biblioteca JWT

Vamos começar instalando a biblioteca `jsonwebtoken`:

bash

 Copiar código

```
npm install jsonwebtoken
```

Configurando o JWT na Aplicação

Vamos configurar nossa aplicação para utilizar JWT. Primeiro, vamos importar a biblioteca e gerar um token.

authController.js

javascript

 Copiar código


```
const jwt = require('jsonwebtoken'); const usuarios = [ { id: 1, nome: 'Admin', login: 'admin', senha: '123456' }, { id: 2, nome: 'User', login: 'user', senha: '123456' } ]; exports.login = (req, res) => { const { login, senha } = req.body; const usuario = usuarios.find(user => user.login === login && user.senha === senha); if (usuario) { // Gerando o token const token = jwt.sign( { id: usuario.id, nome: usuario.nome }, 'seu-segredo-seguro', // Chave secreta para assinatura { expiresIn: '1h' } // Token expira em 1 hora ); res.json({ token }); } else { res.status(401).json({ mensagem: 'Login ou senha incorretos' }); } };
```

Verificando o JWT nas Rotas Privadas

Agora vamos modificar nosso middleware para verificar o token.

authMiddleware.js

javascript

 Copiar código

```
const jwt = require('jsonwebtoken'); const authMiddleware = (req, res, next) => { const token = req.headers['authorization']; if (!token) { return res.status(403).json({ mensagem: 'Token não fornecido' }); } try { const
```

```
decoded = jwt.verify(token, 'seu-segredo-seguro'); req.usuarioId = decoded.id; next(); } catch (err) {  
  res.status(403).json({ mensagem: 'Token inválido' }); } } module.exports = authMiddleware;
```

Configurando o Servidor

Vamos configurar o servidor para utilizar as rotas públicas e privadas.

server.js

javascript

 Copiar código

```
const express = require('express'); const app = express(); const publicRoutes = require('./routes/publicRoutes'); const  
privateRoutes = require('./routes/privateRoutes'); app.use(express.json()); app.use('/api', publicRoutes); app.use('/api',  
privateRoutes); const PORT = 3000; app.listen(PORT, () => { console.log(`Servidor rodando na porta ${PORT}`); });
```

Testando com Insomnia

Configurando Insomnia

1. Login:

- Método: POST
- URL: `http://localhost:3000/api/login`
- Body (JSON):

json

 Copiar código

```
{ "login": "admin", "senha": "123456" }
```

- Resposta esperada:

json

 Copiar código

```
{ "token": "seu-token-gerado-aqui" }
```

2. Listar Usuários (Autenticado):

- Método: GET
- URL: `http://localhost:3000/api/usuarios`
- Headers: `Authorization: seu-token-gerado-aqui`

3. Criar Post (Autenticado):

- Método: POST
- URL: `http://localhost:3000/api/posts`
- Headers: `Authorization: seu-token-gerado-aqui`
- Body (JSON):

json

 Copiar código

```
{ "titulo": "Novo Post", "conteudo": "Conteúdo do novo post" }
```

4. Listar Posts (Autenticado):

- Método: GET
- URL: `http://localhost:3000/api/posts`
- Headers: `Authorization: seu-token-gerado-aqui`

Conclusão

Hoje, aprendemos a implementar autenticação usando JWT no Express.

Criamos tokens JWT para autenticação de usuários e configuramos middleware para verificar esses tokens nas rotas privadas.

Estudem e pratiquem bastante.

Até a próxima aula!

Se houver qualquer dúvida ou questão, estarei por aqui para ajudar. Continuem se dedicando e bons estudos!

1.6. Vídeo Aula

dia 8 express 3 converted



1.7. Finalizando a Configuração com Dotenv

Na última aula, trabalhamos com JWT para autenticação.

Agora, vamos abordar como gerenciar variáveis de ambiente com a biblioteca `dotenv`, garantindo mais segurança para nossa aplicação.

O que é Dotenv?


Dotenv é uma biblioteca que carrega variáveis de ambiente de um arquivo `.env` para o ambiente `process.env`.

Isso é especialmente útil para armazenar informações sensíveis, como chaves de API e senhas, fora do código-fonte.

Instalação do Dotenv

Vamos instalar o `dotenv`:

bash

 Copiar código

```
npm install dotenv
```

Configuração do Dotenv

1. Crie um arquivo `.env` na raiz do seu projeto:


plaintext

 Copiar código

```
APP_KEY_TOKEN=suachavemuitosegura DB_PASSWORD=minhasenha secreta
```

2. Carregue o Dotenv no início do seu arquivo `server.js`:

javascript

 Copiar código

```
require('dotenv').config();
```

Uso das Variáveis de Ambiente

Vamos modificar nossa aplicação para usar essas variáveis de ambiente.

authController.js

Atualize o arquivo `authController.js` para usar a chave do token a partir do `.env`:

javascript

 Copiar código

```
const jwt = require('jsonwebtoken'); const usuarios = [ { id: 1, nome: 'Admin', login: 'admin', senha: '123456' }, { id: 2, nome: 'User', login: 'user', senha: '123456' } ]; exports.login = (req, res) => { const { login, senha } = req.body; const usuario = usuarios.find(user => user.login === login && user.senha === senha); if (usuario) { // Gerando o token const token = jwt.sign( { id: usuario.id, nome: usuario.nome }, process.env.APP_KEY_TOKEN, // Usando a chave do .env { expiresIn: '1h' } ); res.json({ token }); } else { res.status(401).json({ mensagem: 'Login ou senha incorretos' }); } };
```

Validando o JWT

Vamos atualizar o middleware para validar o token usando a chave do `.env`.

authMiddleware.js

javascript

 Copiar código


```
const jwt = require('jsonwebtoken'); const authMiddleware = (req, res, next) => { const token = req.headers['authorization']; if (!token) { return res.status(403).json({ mensagem: 'Token não fornecido' }); } try { const decoded = jwt.verify(token, process.env.APP_KEY_TOKEN); req.usuarioId = decoded.id; next(); } catch (err) { res.status(403).json({ mensagem: 'Token inválido' }); } }; module.exports = authMiddleware;
```

Atualizando as Rotas

server.js

Certifique-se de carregar o `dotenv` no início do seu `server.js`:

javascript

 Copiar código


```
require('dotenv').config(); const express = require('express'); const app = express(); const publicRoutes = require('./routes/publicRoutes'); const privateRoutes = require('./routes/privateRoutes'); app.use(express.json()); app.use('/api', publicRoutes); app.use('/api', privateRoutes); const PORT = 3000; app.listen(PORT, () => { console.log(`Servidor rodando na porta ${PORT}`); });
```

Testando com Insomnia

1. Login:

- Método: POST
- URL: `http://localhost:3000/api/login`
- Body (JSON):

json

 Copiar código

```
{ "login": "admin", "senha": "123456" }
```

- Resposta esperada:

json

 Copiar código

```
{ "token": "seu-token-gerado-aqui" }
```

2. Listar Usuários (Autenticado):

- Método: GET
- URL: `http://localhost:3000/api/usuarios`
- Headers: `Authorization: seu-token-gerado-aqui`

3. Criar Post (Autenticado):

- Método: POST
- URL: `http://localhost:3000/api/posts`
- Headers: `Authorization: seu-token-gerado-aqui`
- Body (JSON):

json

 Copiar código

```
{ "titulo": "Novo Post", "conteudo": "Conteúdo do novo post" }
```

Conclusão

Hoje, aprendemos a usar o `dotenv` para gerenciar variáveis de ambiente, tornando nossa aplicação mais segura ao armazenar informações sensíveis fora do código-fonte.

Estudem e pratiquem bastante. Até a próxima aula!

Se houver qualquer dúvida ou questão, estarei por aqui para ajudar. Continuem se dedicando e bons estudos!

1.8. Vídeo Aula

dia 8 express 4 converted

