🔐 API Livraria com Express.js — Parte 7 (Autenticação de Usuário)

Adicionando autenticação com express-session e bcrypt

👨‍🏫 **Professor:** Fabricio Bizotto

📘 **Disciplina:** Desenvolvimento Web I

🎓 **Curso:** Ciência da Computação

📅 **Fase:** 4ª fase

Roteiro

- Por que autenticação?
- Dependências: `express-session` e `bcrypt`
- Tabela users no SQLite
- Model e repositório de usuário
- Controller e rotas de autenticação
- Middleware de proteção
- Testando endpoints via cURL
- Resumo das mudanças

Por que autenticação?

- Permite login, logout e controle de acesso
- Garante que apenas usuários autenticados possam acessar rotas protegidas
- Armazena sessão do usuário no servidor
- Senhas nunca são salvas em texto puro (hash com bcrypt)

Dependências e instalação

```
npm install express-session bcrypt
```

> `express-session`: Gerencia sessões de usuário no Express.js
> `bcrypt`: Biblioteca para hashing seguro de senhas

Tabela users no SQLite

```
// src/database/sqlite.js (trecho)
function init() {
    // ... Tabela de livros existente
    // Tabela de Usuários
    run(`CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT NOT NULL UNIQUE,
        password_hash TEXT NOT NULL,
        created_at DATETIME DEFAULT CURRENT_TIMESTAMP
    )`);
    // ...
}
```

Configurando express-session

```
// src/config/express.js (trecho)
const session = require("express-session");

app.use(session({
  secret: process.env.SESSION_SECRET || "livraria_secret_key",
  rolling: true, // renova a sessão a cada requisição
  cookie: {
    httpOnly: true,
    secure: false, // true apenas em produção HTTPS
    maxAge: 1000 * 60 * 60 * 2 // 2 horas
  }
}));
```

> Defina SESSION_SECRET no .env em produção. Use o comando openssl rand -base64 32 para gerar um valor seguro.

Model de usuário

```
// src/models/user.model.js
class User {
    constructor({ id = null, username, password = undefined, created_at = undefined }) {
        this.id = id ?? null;
        this.username = String(username || '').trim();
        this.created_at = created_at;
        this.password = password; // opcional (registro/troca)
        this._validar();
    }
    _validar() {
        const erros = [];
        if (!this.username || this.username.length < 3) erros.push('username deve ter pelo menos 3 caracteres');
        if (this.password !== undefined) {
            const pwd = String(this.password);
            if (pwd.length < 6) erros.push('password deve ter pelo menos 6 caracteres');
        }
        if (erros.length) { const e = new Error('Dados de usuário inválidos'); e.statusCode = 400; e.details = erros; throw e; }
    }
    static fromDB(row) { return new User({ id: row.id, username: row.username, created_at: row.created_at }); }
    toJSON() { return { id: this.id, username: this.username, created_at: this.created_at }; }
}
module.exports = User;
```

Repositório de usuários

```
// src/repositories/users.repository.js
const db = require('../database/sqlite');
const User = require('../models/user.model');

class UsersRepository {
    async findById(id) {
        const row = await db.get('SELECT id, username, created_at FROM users WHERE id = ?', [id]);
        return row ? User.fromDB(row) : null;
    }
    async findByUsername(username) {
        const row = await db.get('SELECT id, username, password_hash, created_at FROM users WHERE username = ?', [username]);
        return row || null; // inclui password_hash
    }
    async create({ username, passwordHash }) {
        const result = await db.run('INSERT INTO users (username, password_hash) VALUES (?, ?)', [username, passwordHash]);
        console.log(result);

        const row = await db.get('SELECT id, username, created_at FROM users WHERE id = ?', [result.lastInsertRowid]);
        return User.fromDB(row);
    }
}

module.exports = UsersRepository;
```

Controller de autenticação

```
// src/controllers/auth.controller.js
const bcrypt = require('bcrypt');
const UsersRepository = require('../repositories/users.repository');

class AuthController {
    constructor() {
        this.usersRepo = new UsersRepository();
    }

    async register(req, res, next) {
        try {
            const { username, email, password } = req.body;
            if (!username || !email || !password) {
                return res.status(400).json({ erro: 'Preencha todos os campos obrigatórios.' });
            }
            if (this.usersRepo.findByUsername(username)) {
                return res.status(409).json({ erro: 'Usuário já existe.' });
            }
            if (this.usersRepo.findByEmail(email)) {
                return res.status(409).json({ erro: 'Email já cadastrado.' });
            }
            const hash = await bcrypt.hash(password, 10);
            const user = this.usersRepo.create({ username, email, password: hash });
            req.session.userId = user.id;
            res.status(201).json({ mensagem: 'Usuário registrado com sucesso!', user: user.toJSON() });
        } catch (err) {
            next(err);
        }
    }
}

module.exports = AuthController;
```

Controller de autenticação (continuação)

```
async login(req, res, next) {
    try {
        const { username, password } = req.body;
        const user = this.usersRepo.findByUsername(username);
        if (!user) {
            return res.status(401).json({ erro: 'Usuário ou senha inválidos.' });
        }
        const valid = await bcrypt.compare(password, user.password);
        if (!valid) {
            return res.status(401).json({ erro: 'Usuário ou senha inválidos.' });
        }
        req.session.userId = user.id;
        res.status(200).json({ mensagem: 'Login realizado com sucesso!', user: user.toJSON() });
    } catch (err) {
        next(err);
    }
}
```

Controller de autenticação (continuação)

```
async me(req, res, next) {
    try {
        if (!req.session.userId) {
            return res.status(401).json({ erro: 'Não autenticado.' });
        }
        const user = this.usersRepo.findById(req.session.userId);
        if (!user) {
            return res.status(404).json({ erro: 'Usuário não encontrado.' });
        }
        res.status(200).json({ user: user.toJSON() });
    } catch (err) {
        next(err);
    }
}

async logout(req, res, next) {
    req.session.destroy(() => {
        res.status(200).json({ mensagem: 'Logout realizado com sucesso.' });
    });
}
```

Rotas de autenticação

```
// src/routes/auth.routes.js
const express = require('express');
const router = express.Router();
const AuthController = require('../controllers/auth.controller');
const { requireAuth } = require('../middlewares/auth');

const authController = new AuthController();

router.post('/register', (req, res, next) => authController.register(req, res, next));
router.post('/login', (req, res, next) => authController.login(req, res, next));
router.get('/me', requireAuth, (req, res, next) => authController.me(req, res, next));
router.post('/logout', requireAuth, (req, res, next) => authController.logout(req, res, next));

module.exports = router;
```

Middleware de proteção

```
// src/middlewares/auth.js
function requireAuth(req, res, next) {
  if (!req.session.userId) {
    return res.status(401).json({ erro: 'Acesso não autorizado. Faça login.' });
  }
  next();
}
module.exports = { requireAuth };
```

Testando endpoints via cURL

```
# subir o servidor
npm run dev

# registrar (salva cookie)
curl -i -c cookies.txt -H "Content-Type: application/json" \
  -d '{"username":"alice","password":"secret123"}' \
  http://localhost:3000/api/auth/register

# usuário atual (usa cookie)
curl -b cookies.txt -c cookies.txt http://localhost:3000/api/auth/me

# logout
curl -b cookies.txt -X POST http://localhost:3000/api/auth/logout

# login
curl -i -c cookies.txt -H "Content-Type: application/json" \
  -d '{"username":"alice","password":"secret123"}' \
  http://localhost:3000/api/auth/login
```

Resumo das mudanças

- Tabela `users` criada no SQLite
- Model, repositório, controller e rotas de autenticação
- Middleware de proteção para rotas privadas
- Sessão de usuário via express-session
- Senhas protegidas com bcrypt
- Testes via cURL

Próximos passos / Desafios

- Proteger rotas de CRUD de livros para usuários autenticados
- Adicionar novos campos (email, nome completo) no registro