🔐 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
```

Tabela users no SQLite

```
// src/database/sqlite.js (trecho)
run(`CREATE TABLE IF NOT EXISTS users (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  username TEXT NOT NULL UNIQUE,
  email TEXT NOT NULL UNIQUE,
  password TEXT NOT NULL,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP
)`);
run(`CREATE UNIQUE INDEX IF NOT EXISTS idx_users_username ON users(username)`);
run(`CREATE UNIQUE INDEX IF NOT EXISTS idx_users_email ON users(email)`);
```

Model de usuário

```
// src/models/user.model.js
class User {
  constructor({ id = null, username, email, password, created_at = null }) {
    this.id = id;
    this.username = String(username).trim();
    this.email = String(email).trim().toLowerCase();
    this.password = password; // hash
    this.created_at = created_at;
    this._validate();
  }
  _validate() {
    // ...validação de campos
  }
  toJSON() {
    return {
      id: this.id,
      username: this.username,
      email: this.email,
      created_at: this.created_at
    };
  }
  static fromDB(row) {
    return new User(row);
  }
}
```

Repositório de usuários

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

class UsersRepository {
  findById(id) {
    const row = db.get('SELECT * FROM users WHERE id = ?', [id]);
    return row ? User.fromDB(row) : null;
  }
  findByUsername(username) {
    const row = db.get('SELECT * FROM users WHERE username = ?', [username]);
    return row ? User.fromDB(row) : null;
  }
  findByEmail(email) {
    const row = db.get('SELECT * FROM users WHERE email = ?', [email]);
    return row ? User.fromDB(row) : null;
  }
  create({ username, email, password }) {
    const result = db.run(
      'INSERT INTO users (username, email, password) VALUES (?, ?, ?)',
      [username, email, password]
    );
    return this.findById(result.lastInsertRowid);
  }
  listAll() {
    const rows = db.all('SELECT * FROM users ORDER BY id ASC');
    return rows.map(row => User.fromDB(row));
  }
}
```

Configurando express-session

```
// src/config/express.js (trecho)
const session = require("express-session");
app.use(session({
  secret: process.env.SESSION_SECRET || "livraria_secret_key",
  resave: false,
  saveUninitialized: false,
  cookie: {
    httpOnly: true,
    secure: false, // true apenas em produção HTTPS
    maxAge: 1000 * 60 * 60 * 2 // 2 horas
  }
}));
```

Controller de autenticação

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

class AuthController {
  async register(req, res, next) {
    // ...validação, hash da senha, criação do usuário
  }
  async login(req, res, next) {
    // ...validação, verificação do hash, inicia sessão
  }
  async me(req, res, next) {
    // ...retorna dados do usuário logado
  }
  async logout(req, res, next) {
    // ...destroi sessão
  }
}
```

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

```
# Registrar usuário
curl -X POST http://localhost:3000/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{"username": "fulano", "email": "fulano@email.com", "password": "123456"}'

# Login
curl -X POST http://localhost:3000/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username": "fulano", "password": "123456"}'

# Consultar usuário logado
curl -X GET http://localhost:3000/api/auth/me \
  --cookie-jar cookies.txt

# Logout
curl -X POST http://localhost:3000/api/auth/logout \
  --cookie cookies.txt
```

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 validação extra (força de senha, email único)
- Implementar recuperação de senha
- Adicionar roles/permissões (admin, usuário)