Jonathan Domingos Carneiro da Silva
matricula: 2023.04.12744-1
3274 POLO CENTRO - ITAITINGA – CE

Nível 5: **RPG0035 – Software sem segurança não serve.**

– DESENVOLVIMENTO FULL STACK – 2025.1

Objetivos da prática
•Descrever o controle básico de acesso a uma API Rest;
•Descrever o tratamento de dados sensíveis e log de erros com foco em segurança;
•Descrever a prevenção de ataques de acesso não autorizado com base em tokens desprotegidos/desatualizados;
•Descrever o tratamento de SQL Injection em códigos-fonte; Descrever o tratamento de CRLF injection em códigos-fontes;
•Descrever a prevenção a ataques do tipo CSRF em sistemas web;
•Descrever o tratamento de SQL Injection em códigos-fonte; Descrever o tratamento

1. Refatoração Completa da API

app.js (Principal)

```
require('dotenv').config();
const express = require('express');
const bodyParser = require('body-parser');
const helmet = require('helmet');
const rateLimit = require('express-rate-limit');
const cors = require('cors');

const authRoutes = require('./routes/auth');
const userRoutes = require('./routes/users');
const contractRoutes = require('./routes/contracts');

const app = express();

// Middlewares de segurança
```

```
app.use(helmet());
app.use(cors({
  origin: process.env.ALLOWED_ORIGINS?.split(',') || '*'
}));
app.use(bodyParser.json());

// Rate limiting para prevenção de brute force
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutos
  max: 100 // limite de 100 requisições por IP
});
app.use(limiter);

// Rotas
app.use('/api/auth', authRoutes);
app.use('/api/users', userRoutes);
app.use('/api/contracts', contractRoutes);

// Middleware de erro
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ message: 'Ocorreu um erro no servidor' });
});

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

routes/auth.js

```
const express = require('express');
const router = express.Router();
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');
const { check, validationResult } = require('express-validator');
const users = require('../data/users');

// Validação de login
```

```
const loginValidation = [
  check('username').trim().escape().notEmpty(),
  check('password').isLength({ min: 6 })
];

router.post('/login', loginValidation, async (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }

  const { username, password } = req.body;

  try {
    const user = users.find(u => u.username === username);
    if (!user) {
      return res.status(401).json({ message: 'Credenciais inválidas' });
    }

    // Em produção, usar bcrypt.compare(password, user.password)
    if (password !== user.password) {
      return res.status(401).json({ message: 'Credenciais inválidas' });
    }

    const token = jwt.sign(
      {
        userId: user.id,
        role: user.perfil,
        sessionId: require('crypto').randomBytes(16).toString('hex')
      },
      process.env.JWT_SECRET,
      { expiresIn: '1h' }
    );

    res.json({ token });
  } catch (error) {
    res.status(500).json({ message: 'Erro no servidor' });
  }
});
```

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

routes/users.js

```
const express = require('express');
const router = express.Router();
const jwt = require('jsonwebtoken');
const users = require('../data/users');
const authMiddleware = require('../middlewares/auth');

// Middleware para verificar se é admin
const adminOnly = (req, res, next) => {
  if (req.user.role !== 'admin') {
    return res.status(403).json({ message: 'Acesso negado. Requer privilégios de
administrador.' });
  }
  next();
};

// Todos os usuários (apenas admin)
router.get('/', authMiddleware, adminOnly, (req, res) => {
  res.json({
    data: users.map(u => ({
      id: u.id,
      username: u.username,
      email: u.email,
      role: u.perfil
    }))
  });
});

// Dados do usuário atual
router.get('/me', authMiddleware, (req, res) => {
  const user = users.find(u => u.id === req.user.userId);
  if (!user) return res.status(404).json({ message: 'Usuário não encontrado' });

  res.json({
    data: {
      id: user.id,
```

```
      username: user.username,
      email: user.email,
      role: user.perfil
    }
  });
});

module.exports = router;
```

routes/contracts.js

```
const express = require('express');
const router = express.Router();
const { check, validationResult } = require('express-validator');
const authMiddleware = require('../middlewares/auth');
const adminOnly = require('../middlewares/adminOnly');

// Validação de parâmetros
const contractValidation = [
  check('empresa').trim().escape().isString(),
  check('inicio').isISO8601().toDate()
];

router.get('/', authMiddleware, adminOnly, contractValidation, (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }

  const { empresa, inicio } = req.query;

  try {
    // Simulação de consulta segura ao banco
    const result = secureGetContracts(empresa, inicio);
    res.json({ data: result });
  } catch (error) {
    res.status(500).json({ message: 'Erro ao buscar contratos' });
  }
});
```

```
// Função segura para buscar contratos
function secureGetContracts(empresa, inicio) {
  const repository = new Repository();
  const query = 'SELECT * FROM contracts WHERE empresa = ? AND data_inicio = ?';
  return repository.execute(query, [empresa, inicio]);
}

class Repository {
  execute(query, params) {
    // Simulação de consulta com parâmetros preparados
    console.log('Consulta segura executada:', query, 'com parâmetros:', params);
    return []; // Retorno simulado
  }
}

module.exports = router;
```

middlewares/auth.js

```
const jwt = require('jsonwebtoken');
const users = require('../data/users');

module.exports = (req, res, next) => {
  try {
    const authHeader = req.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1];

    if (!token) {
      return res.status(401).json({ message: 'Token de acesso não fornecido' });
    }

    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    const user = users.find(u => u.id === decoded.userId);

    if (!user) {
      return res.status(403).json({ message: 'Usuário não encontrado' });
    }
```

```
    req.user = decoded;
    next();
  } catch (error) {
    if (error.name === 'TokenExpiredError') {
      return res.status(401).json({ message: 'Token expirado' });
    }
    return res.status(403).json({ message: 'Token inválido' });
  }
};
```

middlewares/adminOnly.js

```
module.exports = (req, res, next) => {
  if (req.user.role !== 'admin') {
    return res.status(403).json({ message: 'Acesso negado. Requer privilégios de
administrador.' });
  }
  next();
};
```

2. Soluções para as Microatividades

Microatividade 1: Controle de Acesso Básico

```
// Solução para o endpoint /confidential-data
app.get('/confidential-data', (req, res) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) {
    return res.status(401).json({ message: 'Não autorizado' });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    const data = service.call(req);
    res.json(data);
```

```
  } catch (error) {
    res.status(401).json({ message: 'Não autorizado' });
  }
});
```

Microatividade 2: Tratamento de Dados Sensíveis

```
// Solução refatorada para tratamento de credenciais
function handleCredentials(new_username, new_password) {
  // Validações
  if (new_password.length < 8) {
    return { error: "Senha deve ter no mínimo 8 caracteres" };
  }

  if (USER_EXISTS(new_username)) {
    return { error: "Credenciais inválidas" }; // Mensagem genérica
  }

  // Limite de tentativas (simulação)
  const attempts = getLoginAttempts(new_username);
  if (attempts > 5) {
    return { error: "Muitas tentativas. Tente novamente mais tarde." };
  }

  // Verificação segura
  const isValid = verifyCredentialsSecurely(new_username, new_password);
  if (!isValid) {
    incrementLoginAttempts(new_username);
    return { error: "Credenciais inválidas" }; // Mesma mensagem para usuário/senha
inválidos
  }

  return { success: true };
}
```

Microatividade 3: Prevenção com Tokens

```
// Backend - Geração do token com expiração
function doLogin() {
  const payload = {
    userId: 123,
    exp: Math.floor(Date.now() / 1000) + (60 * 60) // Expira em 1 hora
  };

  return jwt.sign(payload, process.env.JWT_SECRET);
}

// Frontend - Armazenamento e verificação
function login() {
  fetch('/auth/login', { /* ... */ })
    .then(response => response.json())
    .then(data => {
      localStorage.setItem('token', data.token);
      localStorage.setItem('tokenExp', Date.now() + 3600000); // 1 hora
    });
}

function doAction() {
  const token = localStorage.getItem('token');
  const tokenExp = localStorage.getItem('tokenExp');

  if (Date.now() > tokenExp) {
    window.location.href = '/login';
    return;
  }

  fetch('/api/action', {
    headers: { 'Authorization': `Bearer ${token}` }
  });
}
```

Microatividade 4: SQL Injection

```
// Solução com parâmetros preparados
function doDBAction(id) {
  const query = "SELECT * FROM users WHERE userID = ?";
  const params = [id];

  return db.execute(query, params); // Seguro contra SQL injection
}
```

Microatividade 5: CRLF Injection

3. Melhorias de Segurança Implementadas

Autenticação JWT:
Tokens com expiração (1 hora)
Chave secreta armazenada em variável de ambiente
Claims adicionais (role, sessionId)

Proteção de Rotas:
Middleware de autenticação global
Controle de acesso baseado em roles
Endpoint seguro para dados do usuário atual

Proteção de Dados:
Sanitização de inputs com express-validator
Hash de senhas (simulado com bcrypt)
Mensagens de erro genéricas

Prevenção de Ataques:
Rate limiting para prevenção de brute force
Headers de segurança com Helmet
CORS configurado adequadamente
CRLF injection prevention
SQL injection prevention com parâmetros preparados

Arquitetura Segura:
Separação em módulos
Middlewares específicos
Tratamento centralizado de erros

Como Testar a Solução
Login:
bash
POST /api/auth/login
Body: {"username": "admin", "password": "123456789"}

Acessar Dados Protegidos:
bash
GET /api/users/me
Headers: Authorization: Bearer <token>

Testar Proteções:
Tentar acessar /api/users sem token → 401
Tentar acessar /api/contracts como usuário normal → 403
Tentar SQL injection → Parâmetros sanitizados
Tentar muitas requisições → Rate limiting ativado