**Missão Prática – Mundo 05 – Nível 05**

**Gilvan Pereira de Oliveira – 2023.01.53256-61197**

**Polo Centro – São Lourenço Da Mata – PE**

**RPG0034 – SOFTWARE SEM SEGURANÇA NÃO SERVE! – 9001 – 2025.1**

<https://github.com/GilvanPOliveira/FullStack/tree/main/Mundo05/softwareSeguranca>

**Missão Prática | Software sem segurança não serve**

**Texto

O conteúdo gerado por IA pode estar incorreto.**

**Texto

O conteúdo gerado por IA pode estar incorreto.**

**Texto

O conteúdo gerado por IA pode estar incorreto.**

A implementação correta de mecanismos de segurança é essencial para garantir proteção robusta em APIs modernas, especialmente aquelas que utilizam autenticação baseada em tokens. Neste contexto, a utilização do padrão JSON Web Token (JWT) oferece vantagens como escalabilidade, desempenho e segurança aprimorada, permitindo o gerenciamento eficiente de autenticação e autorização entre cliente e servidor.

Abaixo serão apresentados os principais pontos relacionados à implementação feita a partir da autenticação JWT na API, escolhida para esta missão, destacando estratégias fundamentais adotadas para garantir integridade, controle de acessos e prevenção contra vulnerabilidades comuns, como ataques de SQL Injection:

1. **Autenticação e Geração de Token JWT**

* O endpoint de login (POST /api/autenticar/login) autentica o usuário com base nas credenciais fornecidas;
* Após autenticação bem-sucedida, um token JWT é gerado contendo um payload com informações essenciais (id, username e perfil) e possui validade configurada explicitamente para 1 hora;
* O token JWT é retornado no corpo da resposta ao cliente para uso nas requisições subsequentes.

1. **Envio do Token JWT via Cabeçalho HTTP**

* O cliente deve enviar o token JWT no cabeçalho Authorization das requisições no formato padrão Bearer <token> (ambos nomes utilizados para seguir as melhores práticas da biblioteca utilizada);
* Um middleware (authenticateToken) verifica a existência, formato e validade do token JWT em cada requisição protegida, extraindo e disponibilizando os dados do usuário para uso interno.

1. **Validação Automática do Token**

* Todas as requisições feitas a endpoints protegidos executam o middleware authenticateToken, que garante a integridade do token e verifica sua validade (inclusive a data/hora de expiração);
* Caso o token JWT esteja ausente, incorreto ou expirado, a API retorna uma mensagem clara e um código HTTP apropriado.

1. **Controle de Acesso Baseado em Perfil**

* O middleware requireAdmin assegura que endpoints sensíveis (/api/usuarios e /api/contratos/:empresa/:inicio) sejam acessados exclusivamente por usuários com o perfil administrativo (admin);
* Há também um endpoint dedicado (GET /api/autenticar/usuario) que permite a qualquer usuário autenticado recuperar seus próprios dados sem exigência de perfil especial, facilitando o uso da aplicação para todos os usuários logados.

1. **Prevenção de SQL Injection**

* Nos endpoints que recebem parâmetros de consulta (como contratos), os parâmetros (empresa e inicio) são rigorosamente validados por meio de expressões regulares seguras, permitindo exclusivamente caracteres seguros (alfanuméricos, hífens, underscores, espaços e pontos);
* As consultas ao banco de dados utilizam consultas parametrizadas (simuladas pela classe Repository e pelo método getContracts), eliminando o risco associado à concatenação direta de strings.

1. **Configuração Segura via Variáveis de Ambiente**

* A aplicação utiliza process.env.SECRET\_KEY, com fallback seguro definido ("chaveSecreta"), assegurando maior segurança e flexibilidade entre ambientes (desenvolvimento, homologação e produção);
* A porta da aplicação também é configurada via variáveis de ambiente, garantindo adaptabilidade a diferentes contextos operacionais.

1. **Organização e Melhores Práticas Gerais**

* O uso de middlewares dedicados para autenticação (authenticateToken) e controle de autorização (requireAdmin) torna o código modular, de fácil manutenção e compreensão;
* A separação clara entre os endpoints e a lógica de negócio (como doLogin para autenticação e getContracts para consultas) promove uma arquitetura limpa e organizada, facilitando testes e evolução futura do código.

**Após fazer o que foi proposto na missão resolvi efetuar alguns testes e modifiquei o código-fonte original para incrementar um front e testar de uma forma mais simplificada, seguem os prints dos testes:**

**Interface gráfica do usuário, Texto, Aplicativo

O conteúdo gerado por IA pode estar incorreto.** Interface gráfica do usuário, Texto, Aplicativo

O conteúdo gerado por IA pode estar incorreto. **Interface gráfica do usuário, Texto, Aplicativo

O conteúdo gerado por IA pode estar incorreto.**

**Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto.**  **Interface gráfica do usuário, Texto, Aplicativo

O conteúdo gerado por IA pode estar incorreto.** Interface gráfica do usuário, Texto, Aplicativo

O conteúdo gerado por IA pode estar incorreto.