UNIVERSIDADE FEDERAL DO OESTE DO PARÁ INSTITUTO DE ENGENHARIA E GEOCIÊNCIAS PROGRAMA DE COMPUTAÇÃO

Discente: Igo Quintino Castro Prata

Nº de Matrícula: 2020003059

Professor: Rennan Jose Maia Da Silva

Projeto Final - Entrega Parcial 1: Análise dos componentes de segurança do App web PobreFlix

Componentes de segurança do projeto PobreFlix

A aplicação web PobreFlix utiliza diversas medidas de segurança da informação, com direcionamento em autenticação , autorização e proteção de dados. a seguir o detalhamento de componentes de segurança identificados no código fonte do mesmo:

---

1. JSON Web Tokens (JWT)

- **Onde estão:**
  - `src/utils/auth.js`: Contém as funções `generateToken` (para criar o token) e `verifyToken` (para verificar o token).
  - `src/controllers/authController.js`: Utiliza o JWT para login e logout de usuários.
  - `src/middlewares/authMiddleware.js`: O middleware authenticate intercepta as requisições para validar o JWT presente no cabeçalho Authorization.
- **Como é aplicado:**
  - **Autenticação**: Após um login bem-sucedido, um JWT é gerado e retornado ao cliente. Este token inclui o `userId` e o `userType` do usuário.
  - **Autorização**: Para acessar rotas protegidas, o cliente deve enviar o JWT no cabeçalho Authorization como Bearer `seu_token_aqui`. O middleware authenticate verifica a validade e a autenticidade do token.
  - **Gerenciamento de Sessão**: O token tem um tempo de expiração definido (`JWT_EXPIRES_IN`). No logout, o token é adicionado a uma "blacklist" para invalidá-lo imediatamente, mesmo antes de sua expiração natural.

2. Bcrypt para Hashing de Senhas

- **Onde estão:**
    - src/models/userModel.js: As funções addUser e createPublicUser usam bcrypt.hash para criptografar as senhas antes de armazená-las no banco de dados.
    - src/controllers/authController.js: A função login utiliza bcrypt.compare para verificar a senha fornecida pelo usuário com a senha hasheada armazenada.
    - package.json e package-lock.json: bcrypt é listado como uma dependência do projeto.
- **Como é aplicado:**
    - **Proteção de Senhas**: Em vez de armazenar senhas em texto puro, o bcrypt é usado para gerar um hash unidirecional. Isso protege as senhas mesmo que o banco de dados seja comprometido, pois é extremamente difícil reverter o hash para a senha original. O custo do hash é definido em 10 (bcrypt.hash(password, 10)).
    - **Verificação Segura**: Durante o login, o bcrypt compara o hash da senha fornecida com o hash armazenado, garantindo que a comparação seja feita de forma segura sem expor a senha em texto puro.

3. Middleware de Autenticação e Autorização

- **Onde estão**:
    - src/middlewares/authMiddleware.js: Contém o middleware authenticate e as funções para gerenciar a blacklist de tokens.
- **Como é aplicado**:
    - **authenticate**: Esta função é usada em várias rotas para garantir que apenas usuários com um JWT válido e não expirado (e não na blacklist) possam acessá-las. Ele extrai o userId e userType do token e os anexa ao objeto req para uso posterior nos controladores.
    - **Blacklist de Tokens**: Implementa um mecanismo para invalidar tokens JWT imediatamente após o logout. Isso impede que um token roubado continue sendo usado até sua expiração natural.

4. Controle de Acesso Baseado em Papéis (RBAC)

- **Onde estão:**
    - src/models/userModel.js: A tabela users possui a coluna user_type com os valores 'Administrator' e 'Client'.
    - src/controllers/userController.js, src/controllers/catalogController.js, src/controllers/historyController.js, src/controllers/logAccessController.js, src/controllers/externalApiController.js: Diversas verificações de creatorUserType são realizadas para restringir o acesso a certas funcionalidades apenas a administradores.
- **Como é aplicado:**
    - **Tipos de Usuário**: O sistema diferencia usuários por seus tipos (Administrator ou Client).

    - **Restrição de Funções**: Ações sensíveis, como listar todos os usuários, criar administradores, adicionar/editar/remover conteúdo do catálogo, visualizar histórico geral e logs de acesso, são restritas a usuários do tipo `Administrator`.
    - **Criação de Usuários**: A criação de novos usuários `Administrator` só pode ser feita por um `Administrator` já existente, enquanto usuários `Client` podem se registrar publicamente.

5. Variáveis de Ambiente (`dotenv`)

- **Onde estão:**
    - `.env`: Arquivo para armazenar variáveis de ambiente sensíveis.
    - `src/server.js`: Carrega as variáveis do arquivo `.env`.
    - `src/utils/auth.js`: Utiliza `process.env.JWT_SECRET` e `process.env.JWT_EXPIRES_IN`.
- **Como é aplicado:**
    - **Gerenciamento Seguro de Configurações**: Informações sensíveis como a `JWT_SECRET` e a string de conexão do banco de dados (`CONNECTION_STRING`) são armazenadas em variáveis de ambiente em vez de serem codificadas diretamente no código. Isso evita que credenciais sejam expostas publicamente no repositório. O `.gitignore` garante que o arquivo `.env` não seja versionado.

6. Logs de Acesso e Auditoria

- **Onde estão:**
    - `src/models/logModel.js`: Define a estrutura e a lógica para adicionar registros de log ao banco de dados.
    - `src/services/logService.js`: Serviço responsável por encapsular a lógica de registro de logs, coletando informações da requisição como IP e user agent.
    - `src/controllers/authController.js`: Implementa o registro de logs para tentativas de login (sucesso, falha, erro).
    - `src/controllers/deviceController.js`: Implementa o registro de logs para o registro de novos dispositivos.
    - `doc/new-cria-banco-postgres.sql`: Define a tabela `log` no banco de dados.
- **Como é aplicado:**
    - **Rastreabilidade**: Registra operações importantes do sistema, como tentativas de login (bem-sucedidas e falhas), logouts e registro de dispositivos.
    - **Monitoramento de Segurança**: Ajuda a identificar atividades suspeitas ou tentativas de acesso não autorizado, fornecendo detalhes como IP de origem, user agent, operação, status e ID do usuário.

7. Validação de Entrada

- **Onde estão:**
  - src/controllers/authController.js: Validação de email e password obrigatórios.
  - src/controllers/userController.js: Validação de campos obrigatórios (name, email, password) na criação de usuário e formato de email na atualização.
  - src/controllers/catalogController.js: Validação de campos obrigatórios (title, content_type, video_url) e do tipo de conteúdo (filme ou serie).
  - src/models/catalogModel.js: Validação de campos obrigatórios e do tipo de conteúdo.
  - src/models/userModel.js: Verifica se o email já está em uso na criação de usuário.
- **Como é aplicado:**
  - **Prevenção de Ataques**: Garante que os dados recebidos da requisição estejam no formato esperado e sejam válidos, prevenindo ataques como injeção SQL (embora o pg já ajude com parametrização de queries) e outros tipos de entradas maliciosas ou inesperadas.
  - **Integridade dos Dados**: Ajuda a manter a consistência e a validade dos dados armazenados no banco.

8. Criptografia de Dados (AES-GCM)

- **Onde estão:**
  - src/utils/cryptoHelper.js: Contém a função decrypt que implementa a descriptografia AES-256-GCM.
  - src/middlewares/encryptionMiddleware.js: O middleware decryptRequest usa cryptoHelper.decrypt para descriptografar o corpo da requisição antes que ele chegue aos controladores.
  - src/routes/authRoutes.js e src/routes/catalogRoutes.js: Algumas rotas utilizam decryptRequest como middleware.
  - src/models/deviceModel.js: Gera cripto_key para a sessão do dispositivo.
  - src/testeCripto.html: Um arquivo HTML de exemplo para testar a criptografia no lado do cliente.
- **Como é aplicado:**
  - **Confidencialidade da Requisição**: Permite que o corpo das requisições POST/PATCH seja criptografado no cliente e descriptografado no servidor. Isso adiciona uma camada de segurança para proteger a confidencialidade dos dados em trânsito, mesmo que o canal de comunicação não seja HTTPS (embora HTTPS seja sempre recomendado).
  - **Chave Única por Dispositivo**: Uma cripto_key única é gerada e associada a cada sessão de dispositivo registrada, aumentando a segurança ao limitar o uso da chave a um dispositivo específico.
  - **Integridade e Autenticidade**: O uso de AES-GCM (Galois/Counter Mode) não apenas criptografa, mas também fornece autenticação dos dados (via authTag), garantindo que os dados não foram adulterados em trânsito.

9. API Key para Dispositivos

- **Onde estão:**
  - src/models/deviceModel.js: Responsável por registrar novos dispositivos e gerar api_key e cripto_key.
  - src/middlewares/apiKeyMiddleware.js: Middleware requireApiKey valida a API Key presente no cabeçalho X-API-Key.
  - src/routes/authRoutes.js e src/routes/catalogRoutes.js: Rotas que exigem a API Key.
  - doc/new-cria-banco-postgres.sql: Cria a tabela sessao para armazenar as informações do dispositivo, incluindo api_key e cripto_key.
- **Como é aplicado:**
  - **Acesso de Dispositivos Confiáveis**: Permite que apenas dispositivos previamente registrados com uma API Key válida e ativa acessem certas funcionalidades da API, adicionando uma camada de segurança antes mesmo da autenticação do usuário.
  - **Controle de Acesso Fino**: A API Key é validada para garantir que não está inativa ou expirada

No geral, a PobreFlix aborda as camadas de segurança, combinando métodos robustos como por exemplo hashing de senhas e JWTs para autenticação, com algumas estratégias adicionais como validação de entrada, logs e criptografia de dados em trânsito para proteger a integridade e a confidencialidade das informações.

Implementação: Sessão Ativa Única por Usuário no PobreFlix

Objetivo

Garantir que **apenas um dispositivo por vez** mantenha uma sessão ativa para o mesmo usuário. Ou seja, ao fazer login em um novo dispositivo, a sessão anterior é automaticamente encerrada.

Visão Geral da Implementação

A lógica de **não-simultaneidade de sessões** foi construída em etapas, envolvendo:

- **Alterações no banco de dados**
- **Controle de dispositivos por sessão**
- **Autenticação combinada com gerenciamento de chaves de dispositivo (API Key)**

Modificações no Banco de Dados

1. Adição da Coluna user_id na Tabela sessao

- **Arquivo**: doc/new-cria-banco-postgres.sql
- **O que foi feito**: A tabela sessao agora inclui a coluna user_id (INT) com *foreign key* para a tabela users.
- **Objetivo**: Permitir a associação de cada dispositivo a um usuário.
    - Inicialmente, user_id é NULL.
    - Após login com sucesso, a sessão é associada ao user_id.

2. Tornar a Coluna id_usuario da Tabela log Opcional (Nullable)

- **Arquivo**: doc/new-cria-banco-postgres.sql
- **O que foi feito**: Remoção da restrição NOT NULL da coluna id_usuario.
- **Objetivo**: Permitir que logs sejam registrados mesmo **sem** um usuário logado, como no caso:
    - Registro de um novo dispositivo.
    - Erros de sistema antes do login.

Fluxo da Sessão: Como Funciona na Prática

🔹 **1. Registro de Dispositivo – POST /devices/register**

- **Descrição**: Todo dispositivo precisa se registrar antes de interagir com a API.
- **Retorno**: api_key e crypto_key exclusivas para o dispositivo.
- **Estado inicial**: A sessão criada **ainda não tem user_id**, pois o login ainda não ocorreu.

🔹 **2. Middleware requireApiKey**

- **Função**: Primeira camada de proteção. Verifica:
    - Se o cabeçalho X-API-Key existe.
    - Se a chave está ativa e válida no banco.
- **Aplicação**: Antes de rotas como /auth/login.

🔹 **3. Login com Exclusividade de Sessão – authController.login**

1. **Autenticação do usuário**: Verificação de email e senha.
2. **Desativação de sessões anteriores**:
    - Chamada: deviceModel.deactivateAllUserSessions(user.id)
    - Resultado: Todas as outras api_keys ativas para o mesmo usuário são desativadas.
3. **Ativação da sessão atual**:
    - A api_key atual é marcada como ativa = TRUE.
    - É associada ao user_id no banco.
4. **Geração de JWT**:
    - Contém a deviceApiKey no payload.
    - Permitirá, futuramente, verificar se a chave da requisição ainda é válida.

🔹 **4. Middleware authenticate**

- **Função**: Executado após requireApiKey em rotas protegidas.
- **Verificações**:
    - Validade do JWT.
    - Se a deviceApiKey ainda está ativa no banco.
- **Caso inválida**: A sessão é rejeitada, forçando re-login.

🔹 **5. Logout – POST /auth/logout**

- **Ações**:
    - O JWT é adicionado à blacklist.
    - A api_key usada é marcada como inativa no banco.

Papel das Tabelas no Banco de Dados

Tabela sessao

- **Campos principais**: api_key, crypto_key, ativa, data_expiracao, user_id.
- **Função**: Gerenciar sessões e dispositivos conectados.

Tabela log

- **Campos principais**: Inclui id_usuario (agora opcional).
- **Função**: Registrar:
    - Tentativas de login (com sucesso ou falha).
    - Registro de dispositivos.
    - Revogação de sessões antigas.
    - Outros eventos importantes.

Conceito Central

> **"Uma sessão ativa por usuário, por vez."**

Isso é feito com:

- Associações exclusivas entre api_key e user_id.
- Desativação automática de sessões anteriores ao novo login.
- JWT incluindo referência à sessão do dispositivo.

Conclusão

A modificação centralizou-se nas tabelas sessao e log, com a finalidade de:

- Habilitar controle de sessão por dispositivo.
- Permitir login exclusivo por usuário.
- Garantir registros detalhados, mesmo em contextos sem login ativo.

Tudo isso resulta em um **sistema robusto e seguro**, com autenticação eficiente e gerenciamento de sessões em conformidade com boas práticas de segurança.