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

Discentes: Igo Quintino Castro Prata

Nº de matrícula: 2020003059

Professor: Rennan Jose Maia Da Silva

Documentos Complementares

Para uma compreensão visual mais aprofundada das implementações de segurança detalhadas neste relatório, existem fluxogramas complementares disponíveis. Estes documentos ilustram os fluxos de trabalho do backend e do frontend:

- **Fluxograma Backend:** `Fluxograma Backend implementações.png`
- **Fluxogramas Frontend:** `Fluxograma Front-End parte 1.png`, `Fluxograma Front-End parte 2.png`, `Fluxograma Front-End parte final.png`

Todos esses arquivos podem ser encontrados na pasta `doc` do projeto `API_REST_PobreFlix`**.** Além disso, estão disponíveis no Google Docs para leitura e comentários.

1. Componentes de Segurança

O projeto PobreFlix incorpora e aprimora os seguintes componentes de segurança, atrelados aos pilares fundamentais da Segurança da Informação:

- **JSON Web Tokens (JWT):**
    - **Descrição:** Utilizados para autenticação e autorização de usuários. Incluem `userId` e `userType` no payload para identificar o usuário e seu papel. O sistema possui um `middleware` **(**`authMiddleware.js`**)** para validar a autenticidade e a validade dos tokens em rotas protegidas. Um mecanismo de `blacklist` é

implementado para invalidar tokens imediatamente no logout, prevenindo o uso indevido de tokens expirados ou roubados.
    - Pilares Atrelados: Autenticação, Autorização, Não Repúdio (ao identificar o usuário da ação).
- **Bcrypt para Hashing de Senhas:**
    - **Descrição:** Empregado para armazenar senhas de forma segura no banco de dados. O Bcrypt é um algoritmo de hashing unidirecional que adiciona um "sal" (salt) aleatório a cada senha antes de hashá-la, tornando-a resistente a ataques de "tabela arco-íris" e garantindo que senhas idênticas tenham hashes diferentes. Durante o login, a senha fornecida pelo usuário é hashada com o mesmo sal e comparada com o hash armazenado.
    - Pilares Atrelados: Confidencialidade (protege a senha original), Integridade (garante que a senha não foi alterada).
- **Controle de Acesso Baseado em Papéis (RBAC):**
    - **Descrição:** Implementa restrições de funcionalidades e acesso a recursos com base no `user_type` (Administrador, Cliente, Premium). Isso garante que apenas usuários com as permissões adequadas possam realizar determinadas operações ou acessar certas áreas do sistema.
    - Pilares Atrelados: Autorização.
- **Variáveis de Ambiente (dotenv):**
    - **Descrição:** Utilizadas para o gerenciamento seguro de configurações sensíveis, como chaves de API, segredos de criptografia e credenciais de banco de dados. Ao manter essas informações fora do código-fonte e em variáveis de ambiente, reduz-se o risco de exposição acidental em repositórios de código.
    - Pilares Atrelados: Confidencialidade.
- **Logs de Acesso e Auditoria:**
    - **Descrição:** Registra operações importantes do sistema, incluindo tentativas de login (bem-sucedidas e falhas), acessos a rotas protegidas e outras atividades relevantes. Esses logs são cruciais para rastreabilidade, monitoramento de segurança, detecção de atividades suspeitas e auditorias futuras.
    - Pilares Atrelados: Não Repúdio, Disponibilidade (para análise e recuperação), Responsabilidade.
- **Validação de Entrada:**

    - **Descrição:** Garante a validade e o formato correto dos dados recebidos de todas as entradas do usuário. Isso previne uma série de ataques comuns, como Injeção SQL, Cross-Site Scripting (XSS) e Buffer Overflow, mantendo a integridade dos dados e a estabilidade do sistema.
    - Pilares Atrelados: Integridade, Disponibilidade.
- **Criptografia de Dados (AES-GCM):**
    - **Descrição:** Utiliza `cryptoHelper` e `encryptionMiddleware` para proteger a confidencialidade do corpo das requisições em trânsito. O AES-GCM é um modo de operação de criptografia autenticada que garante tanto a confidencialidade quanto a integridade dos dados.
    - Pilares Atrelados: Confidencialidade, Integridade.
- **API Key para Dispositivos:**
    - **Descrição:** Um mecanismo inicial para controlar e autenticar o acesso de dispositivos à API. Cada dispositivo registrado recebe uma api_key única, que deve ser apresentada em requisições para rotas protegidas, adicionando uma camada extra de segurança ao acesso do cliente.
    - Pilares Atrelados: Autenticação, Autorização.

2. Funcionamento do Estado Atual das Modificações no Backend

As modificações implementadas no backend do PobreFlix aprimoram significativamente a forma como o sistema lida com as sessões de usuários, especialmente em cenários de múltiplos dispositivos e diferentes tipos de usuário.

2.1. Gerenciamento de Sessões e Dispositivos

- **Registro de Dispositivo Inicial**: Para garantir um controle de acesso mais rigoroso, antes que qualquer usuário possa efetuar login, o dispositivo a ser utilizado deve primeiro se registrar. Isso ocorre através do endpoint `POST /devices/register`. Este processo gera e armazena uma `api_key` e uma `cripto_key` únicas para aquele dispositivo específico na tabela `sessao` do banco de dados. Inicialmente, essas chaves não estão

associadas a um `user_id` (o campo `user_id` é `null`), pois o registro do dispositivo precede a autenticação de um usuário.

- **Validação da API Key:** Um componente crucial do sistema é o middleware `requireApiKey`. Todas as requisições que visam acessar rotas protegidas (incluindo a própria requisição de login) são obrigadas a incluir um cabeçalho `X-API-Key`. Este middleware é responsável por validar se a `api_key` fornecida existe, está ativa e não expirou, garantindo que apenas dispositivos reconhecidos e com sessões válidas possam interagir com o backend.

2.2. Nova Lógica de Login e Controle de Sessões Simultâneas

- **Contagem de Sessões Ativas:** Uma nova função, `deviceModel.countActiveSessionsForUser(userId)`, foi adicionada para permitir ao backend contar o número exato de sessões ativas (registros onde `ativa = TRUE`) que estão atualmente vinculadas a um `user_id` específico na tabela `sessao`. Isso é fundamental para a implementação dos limites de sessão.
- **Limites de Sessão por Tipo de Usuário:** No `authController.login`, após a validação bem-sucedida das credenciais do usuário (e-mail e senha), o sistema verifica o `user_type` do usuário (se é 'Client' ou 'Premium'). Com base nesse tipo, um `maxAllowedSessionsForUser` é definido:
    - Para usuários 'Client' (comuns), o limite é de 1 sessão ativa.
    - Para usuários 'Premium', o limite pode ser de 3 sessões (ou outro valor configurável), permitindo maior flexibilidade.
- **Bloqueio de Login ao Atingir o Limite:** Após a contagem das `activeSessionsCount` para o usuário, o sistema compara esse valor com o `maxAllowedSessionsForUser`. Se `activeSessionsCount` for maior ou igual ao limite permitido, a nova tentativa de login é bloqueada. O backend retorna um status `403 Forbidden` com uma mensagem clara, informando que o limite de sessões foi atingido e que o usuário deve deslogar de outro dispositivo. Importante: As sessões ativas existentes não são deslogadas automaticamente; o novo login é simplesmente impedido.
- **Ativação da Sessão Atual (se permitida):** Se a tentativa de login não for bloqueada (ou seja, há "espaço" para uma nova sessão dentro do limite), a `api_key` do dispositivo que

está efetuando a requisição é atualizada na tabela **`sessao`.** Esta atualização garante que a sessão esteja marcada como **`ativa = TRUE`** e, crucialmente, que esteja associada ao **`user_id`** do usuário que acabou de logar.

- **Geração de JWT:** Um JSON Web Token (JWT) é gerado para o usuário. Este JWT inclui informações essenciais como **`userId`, `userType`** e, de forma vital, a **`deviceApiKey`** da sessão atual. A inclusão da **`deviceApiKey`** no JWT permite que o backend, em requisições futuras para rotas protegidas, verifique não apenas a validade do token, mas também a validade e a associação da sessão do dispositivo.

2.3. Gerenciamento de Logout

- No processo de logout, a **`api_key`** específica do dispositivo que está realizando a ação de deslogar é marcada como **`ativa = FALSE`** na tabela **`sessao`**. Além disso, o JWT correspondente a essa sessão é adicionado a uma **`blacklist`** para garantir sua invalidação imediata, impedindo qualquer uso posterior.

2.4. Modificações na Estrutura do Banco de Dados

- **Tabela `users`**: A coluna **`user_type`** foi atualizada para permitir o valor 'Premium', além dos já existentes 'Administrator' e 'Client'. Isso suporta a diferenciação de serviços e limites de sessão.
- **Tabela `sessao`:** Esta tabela agora contém as **`api_key` e `cripto_key`** dos dispositivos, seu status (**`ativa`**), data de expiração e, de forma crucial, o **`user_id`** associado. Essa ligação direta é fundamental para o rastreamento e gerenciamento de sessões por usuário.
- **Tabela `log`:**
    - A coluna **`id_usuario`** foi tornada **`nullable`**, permitindo o registro de logs para eventos que não possuem um usuário logado no momento (ex: o registro inicial de um dispositivo).
    - Os tamanhos das colunas **`operacao`** e **`descricao`** foram aumentados (**`VARCHAR(50)`** e **`VARCHAR(255)`** respectivamente) para acomodar mensagens de log mais detalhadas e descritivas.

- **Script `seedData.js`:** Este script, responsável por popular o banco de dados com dados iniciais e garantir a estrutura das tabelas, foi atualizado para assegurar que as definições de tabela recriadas (incluindo a tabela `users` com o tipo 'Premium' e as modificações na tabela `sessao` e `log`) estejam sempre consistentes com o esquema mais recente do banco de dados.

2.5. Registro Detalhado de Logs

- Uma nova operação de log, `LOGIN_BLOCKED_SESSION_LIMIT`, é registrada sempre que uma tentativa de login é bloqueada devido ao limite de sessões atingido para um usuário. Isso aprimora significativamente a rastreabilidade e a capacidade de auditoria de segurança, permitindo identificar e analisar tentativas de acesso que excedem as políticas de sessão.

3. Fluxo Explicado do Backend

O fluxo de segurança do backend controla o acesso de usuários e dispositivos:

1. **Registro do Dispositivo:** Um novo dispositivo obtém `api_key` e `cripto_key` via `/devices/register`.
2. **Tentativa de Login:** O usuário tenta logar. O backend valida a `api_key` do dispositivo e as credenciais do usuário.
3. **Controle de Limite:** Verifica o número de sessões ativas do usuário. Se o limite for atingido, o login é bloqueado. Caso contrário, a sessão do dispositivo é ativada para o usuário, e um JWT é emitido.
4. **Acesso Contínuo:** Requisições futuras são autenticadas com o JWT e a `api_key`.
5. Logout: O usuário desloga, invalidando o JWT e desativando a `api_key` no banco de dados.

4. Resumo das Modificações de Segurança no Backend

O backend do PobreFlix agora possui um sistema de segurança robusto e granular. Ele gerencia sessões simultâneas com base no tipo de usuário, bloqueia logins que excedem o limite e

mantém logs detalhados para auditoria. As modificações no banco de dados suportam essa nova lógica, garantindo maior controle de acesso e rastreabilidade.

5. Implementações no Frontend

As modificações no frontend visam aprimorar a segurança, a consistência e a robustez do fluxo de autenticação e da comunicação com o backend em diversas páginas da aplicação.

5.1. Configuração Inicial e Armazenamento Local

- **Gerenciamento de Credenciais:** As chaves de dispositivo (`api_key`, `cripto_key`) e as informações de sessão do usuário (`token`, `userId`, `user`) são agora gerenciadas de forma consistente no `localStorage` do navegador. Isso permite a persistência da sessão entre diferentes acessos e recarregamentos da página.
- **Base de URL da API:** Foi introduzida uma constante `API_BASE_URL` para centralizar e facilitar a gestão dos endpoints do backend em todos os scripts JavaScript do frontend, tornando a manutenção e futuras alterações mais eficientes.

5.2. Fluxo de Registro de Dispositivo (Página de Login - `login.html`)

- **Primeiro Acesso ou Chaves Ausentes:** Na primeira vez que o aplicativo é acessado em um dispositivo, ou sempre que as chaves de dispositivo (`api_key` e `cripto_key`) não estão presentes no `localStorage` (por exemplo, após uma limpeza do navegador ou logout completo), a função `registerDevice()` no script `js/login/script_login_Public.js` é automaticamente executada.
- **Requisição de Registro:** Esta função envia uma requisição **POST** para o endpoint `http://localhost:3000/devices/register` do backend, incluindo um `nome_Dispositivo` (como "Navegador: [User Agent]").
- Armazenamento das Chaves: O backend responde com as `api_key` e `cripto_key` geradas, que são então armazenadas de forma segura no `localStorage` do navegador, preparando o dispositivo para futuras interações autenticadas.

5.3. Fluxo da Tela de Login (`POST /auth/login`)

- Comunicação Padronizada: O formulário de login (`login.html`) agora utiliza uma função padronizada (`sendRequest` ou `sendAuthenticatedRequest` em outros contextos) para se comunicar com o backend.
- Inclusão de Cabeçalhos de Dispositivo: As requisições de login obrigatoriamente incluem os cabeçalhos `X-API-Key` e `X-Encryption-Key-Id`, obtidos do `localStorage`, para autenticar o dispositivo antes mesmo da validação das credenciais do usuário.
- Tratamento de Erros Aprimorado:
    - `401 Unauthorized` (Credenciais Inválidas): Se o backend retornar um status 401 com uma mensagem específica de credenciais inválidas, o frontend exibe a mensagem "Email ou senha inválidos. Por favor, tente novamente.".
    - Controle de Tentativas: Um contador de `loginAttempts` é implementado no frontend. Após 3 tentativas inválidas, o botão de login é desabilitado por 5 minutos, e uma contagem regressiva é exibida ao usuário, prevenindo ataques de força bruta.
    - `403 Forbidden` (Limite de Sessões/API Key Inválida): Para respostas `403 Forbidden`, a mensagem de erro exata retornada pelo backend (ex: "Limite de sessões (X) atingido.") é exibida diretamente na tela, fornecendo feedback claro ao usuário.
    - Outros Erros: Para outros erros (como `500 Internal Server Error` ou problemas de rede), uma mensagem genérica como "Ocorreu um erro inesperado. Por favor, tente novamente mais tarde." é exibida.
- Login Bem-Sucedido: Em caso de login bem-sucedido, o token JWT e os dados do usuário são salvos no `localStorage`, e quaisquer contadores de tentativas (`loginAttempts`, `lockoutTime`) são removidos. O usuário é então redirecionado para `homepage_user.html`.

5.4. Requisições a Rotas Protegidas (`sendAuthenticatedRequest` Global)

- **Função Padronizada:** A função `sendAuthenticatedRequest` foi padronizada e implementada em scripts globais (como `js/admin/script.js`, `js/painel/script.js`, `js/setting/script.js` e `js/homepage/script.js`).
- **Cabeçalhos Obrigatórios:** Todas as chamadas a rotas protegidas agora incluem obrigatoriamente os cabeçalhos `Authorization: Bearer [seu_token_jwt]`,

`X-API-Key: [sua_api_key]` e `X-Encryption-Key-Id: [sua_cripto_key]`, garantindo que a autenticação e a identificação do dispositivo sejam sempre enviadas.

- **Tratamento Global de Sessão Expirada/Inválida:** Se qualquer requisição feita através de `sendAuthenticatedRequest` receber um status `401 Unauthorized` ou `403 Forbidden` do backend, a função automaticamente:
    1. Exibe uma mensagem de alerta ao usuário (ex: "Sua sessão expirou ou foi encerrada. Faça login novamente.").
    2. Limpa todas as credenciais (`token`, `userId`, `user`, `api_key`, `cripto_key`) do `localStorage`.
    3. Redireciona o usuário para login.html, forçando um novo ciclo de autenticação.

5.5. Função de Logout Consistente (`POST /auth/logout`)

- **Padronização:** A lógica de logout foi padronizada em todas as páginas autenticadas (ex: `admin.html`, `homepage_user.html`, `config.html`).
- **Limpeza Imediata do `localStorage`: Ao clicar em "Sair", o `localStorage` é imediatamente limpo no frontend, removendo todas as credenciais do navegador.**
- **Requisição ao Backend: Em seguida, uma requisição `POST` é enviada para `http://localhost:3000/auth/logout`. Esta requisição ainda inclui os cabeçalhos `Authorization`, `X-API-Key` e `X-Encryption-Key-Id` (utilizando os valores que foram capturados antes da limpeza do localStorage), garantindo que a sessão seja invalidada também no backend.**
- **Redirecionamento: Independentemente do sucesso da requisição de logout ao backend, o usuário é então redirecionado para `login.html`, permitindo um novo ciclo de registro de dispositivo e login.**

5.6. Painel de Administração (`admin_painel.html` e `js/painel/script.js`)

- **Nova Seção de Logs:** Uma nova seção de "Log de Atividades do Sistema" foi adicionada ao `admin_painel.html`, permitindo que administradores visualizem as atividades do sistema.

- **Carregamento e Exibição de Logs:** No script `js/painel/script.js`, as funções `loadLogActivities()` e `renderLogActivities()` foram criadas especificamente para buscar e exibir os dados da tabela log do backend, utilizando o endpoint `http://localhost:3000/logAccess/All/logs`. Erros no carregamento dos logs são exibidos na própria tabela para feedback.

5.7. Página de Configurações (`config.html` e `js/setting/script.js`)

- **Uso de `sendAuthenticatedRequest`:** As funcionalidades de atualização de nome de usuário e senha, e a exclusão de conta, agora utilizam a função `sendAuthenticatedRequest` aprimorada. Isso garante que a `api_key` e a `cripto_key` sejam sempre incluídas nas requisições e que os erros de autenticação/autorização sejam tratados globalmente, seguindo o padrão estabelecido.

6. Fluxo Explicado das Atividades da Implementação no Frontend

O frontend gerencia o acesso do usuário e a comunicação com o backend através dos seguintes fluxos:

1. **Início/Primeiro Acesso:** Ao carregar a página de login, verifica se as chaves do dispositivo estão no `localStorage`. Se não, registra o dispositivo no backend e as armazena.
2. **Login do Usuário:** O usuário envia credenciais. O frontend inclui as chaves do dispositivo. Em caso de sucesso, salva o token e dados. Em caso de falha (credenciais inválidas, limite de sessões), exibe mensagens e pode bloquear tentativas.
3. Requisições Protegidas: Em páginas autenticadas, verifica as credenciais. Se ausentes, redireciona para o login. Todas as requisições para o backend incluem JWT e chaves do dispositivo. Erros de autenticação/autorização (401/403) resultam na limpeza do `localStorage` e redirecionamento.
4. **Logout:** Ao clicar em "Sair", o `localStorage` é limpo e uma requisição de logout é enviada ao backend para invalidar a sessão, seguida de redirecionamento para o login.

5. **Sessões Concorrentes:** Se o backend desativa uma sessão (ex: login em outro dispositivo), o dispositivo antigo é deslogado na próxima requisição, limpando o `localStorage` e redirecionando.
6. **Logs de Atividades:** O painel de administração busca e exibe logs do sistema.

7. Resumo das Implementações no Frontend

O frontend agora oferece um fluxo de autenticação robusto, com gerenciamento consistente de credenciais, registro de dispositivo, tratamento detalhado de erros (incluindo bloqueio de tentativas e mensagens de limite de sessão) e um mecanismo global para lidar com sessões expiradas. A padronização do logout e a integração de novas funcionalidades, como o log de atividades, aprimoram a segurança e a experiência do usuário.

8. Resumo Total do Projeto PobreFlix

O projeto PobreFlix foi significativamente aprimorado em segurança e gerenciamento de sessões. O backend agora controla sessões simultâneas por tipo de usuário, bloqueia novos logins que excedam o limite e mantém logs detalhados. O frontend complementa isso com um fluxo de autenticação robusto, gerenciamento seguro de credenciais, tratamento abrangente de erros e um mecanismo global para sessões. Juntos, esses aprimoramentos aumentam a segurança, flexibilidade e capacidade de auditoria do sistema.