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 2: Verificação das Atualizações do App Web PobreFlix

Este relatório apresenta uma análise aprofundada das funcionalidades de segurança desenvolvidas e implementadas no aplicativo web PobreFlix, com ênfase na gestão robusta de sessões e na correção de vulnerabilidades, visando proporcionar uma experiência de usuário mais segura e controlada.

---

1. Implementação da Não-Simultaneidade Estrita de Sessões (Comum e Premium)

A pedra angular desta entrega parcial é a introdução de um sistema de controle de sessões altamente configurável, que impede o uso simultâneo excessivo da plataforma, adaptando-se a diferentes perfis de usuário (Comum e Premium).Arquitetura de Controle de Sessão Detalhada:

- **Endpoint de Registro de Dispositivo Dedicado (`/devices/register`):**
    - **Propósito de Segurança:** Antes mesmo do login, cada dispositivo precisa se registrar. Este endpoint é crucial para emitir uma `api_key` e uma `cripto_key` únicas para cada dispositivo. Isso cria uma camada de identidade para o *dispositivo*, separada da identidade do *usuário*, permitindo um rastreamento mais granular. A ausência dessas chaves ou a tentativa de login sem um registro prévio pode ser um indicador de atividade maliciosa.
    - **Fluxo:** Um novo dispositivo que acessa o PobreFlix pela primeira vez requisita este endpoint. Ele recebe as chaves que serão usadas nas comunicações subsequentes, inclusive no login. A sessão inicial do dispositivo, antes de qualquer autenticação de usuário, é registrada com um `user_id` nulo, indicando que o dispositivo está "desocupado" ou aguardando um login.
- **Validação da API Key (`requireApiKey` Middleware):**
    - **Mecanismo de Segurança:** Este middleware atua como um porteiro para quase todas as requisições à API. Ele verifica se a `X-API-Key` presente no cabeçalho

da requisição é válida, ativa e corresponde a uma sessão de dispositivo legítima em nosso banco de dados.
    - **Impacto na Segurança:** Impede que requisições não autorizadas ou de dispositivos não registrados acessem os recursos da API. Qualquer requisição sem uma `api_key` válida ou com uma `api_key` inativa (por exemplo, após um logout ou desativação manual) é imediatamente rejeitada, protegendo o backend contra acessos indevidos.
- **Lógica de Login (`authController.js`):**
    - **Gerenciamento de Limites:** O `authController.js` agora define dinamicamente os limites de sessões (`MAX_COMMON_SESSIONS` e `MAX_PREMIUM_SESSIONS`). Para usuários comuns, o limite padrão é 1 (não-simultaneidade estrita), enquanto para usuários Premium, pode ser maior (ex: 3), oferecendo um diferencial do serviço.
    - **Forçamento da Não-Simultaneidade:** Após um login bem-sucedido, a função `deviceModel.deactivateExcessUserSessions` é invocada. Esta é a inteligência por trás do controle de sessões: ela identifica e desativa as sessões mais antigas do usuário que excedem o limite permitido. Por exemplo, se um usuário comum já tem uma sessão ativa e tenta logar em um segundo dispositivo, a sessão mais antiga é automaticamente inativada, forçando o logout no primeiro dispositivo.
    - **Inclusão da `deviceApiKey` no JWT:** Para fortalecer a segurança, o JSON Web Token (JWT) agora inclui a `deviceApiKey` no seu payload. Isso significa que, ao validar um JWT para acesso a rotas protegidas, não apenas a assinatura do token é verificada, mas também se a `api_key` do dispositivo que originou aquele token ainda está ativa no banco de dados. Isso mitiga ataques onde um JWT roubado poderia ser usado se a sessão do dispositivo original já tiver sido desativada.
- **Gerenciamento de Sessões (`deviceModel.js`):**
    - **`deactivateExcessUserSessions`:** Esta função é o coração do controle de sessões. Ela consulta o banco de dados para encontrar todas as sessões ativas para um dado `user_id`. Se o número de sessões ativas exceder o `maxAllowedSessions`, ela desativa as sessões mais antigas, garantindo que apenas as sessões mais recentes (e dentro do limite) permaneçam ativas.
- **Processo de Logout:**
    - **Desativação Dupla:** Além de invalidar o JWT (colocando-o em uma blacklist), o processo de logout agora explicitamente desativa a `api_key` do dispositivo associado à sessão que está fazendo logout na tabela `sessao`. Isso garante que mesmo que o JWT não seja imediatamente invalidado (em sistemas sem blacklist ou com latência), a chave do dispositivo associada não possa mais ser usada, reforçando a segurança.

Modificações no Esquema do Banco de Dados (`tabelas_pobreFlix.sql`):

- **Tabela `sessao`:**
    - **`user_id`:** A adição desta coluna é fundamental. Ela estabelece uma relação direta entre uma sessão de dispositivo e um `user_id`. Isso permite que o sistema filtre e conte as sessões ativas por usuário, crucial para a lógica de não-simultaneidade.

- **Tabela `log`:**
    - **`id_usuario` (Nullable):** A coluna `id_usuario` foi tornada *nullable* para permitir o registro de eventos de log que não estão diretamente associados a um usuário logado (ex: tentativas de login falhas, registro de dispositivos antes do login). Isso melhora a rastreabilidade geral do sistema.
    - **Aumento de Tamanho de Colunas (`operacao`, `descricao`):** O aumento do tamanho de `operacao` e `descricao` evita truncamento de informações importantes nos logs, garantindo que detalhes críticos sobre eventos de segurança e sistema sejam completamente registrados para auditoria e depuração.
- **Tabela `users`:**
    - **Atualização do `CHECK` para `user_type`:** A inclusão de 'Premium' na restrição CHECK da coluna `user_type` garante a integridade dos dados e permite a categorização e o gerenciamento de diferentes níveis de usuários, habilitando a diferenciação de limites de sessão.

Protocolo de Testes no Postman:

- **Abrangência:** O guia de testes no Postman é essencial para validar a eficácia do sistema. Ele simula cenários do mundo real, como múltiplos logins de um mesmo usuário em diferentes "dispositivos" (simulados via chaves de API distintas).
- **Verificação de Comportamento:** Os testes confirmam que as sessões mais antigas são desativadas conforme o esperado para usuários comuns e que usuários premium podem manter o número configurado de sessões ativas simultaneamente, provando a aderência aos requisitos de não-simultaneidade.

---

2. Resolução de Erros Durante a Implementação

A correção dos erros a seguir demonstra a atenção à robustez e funcionalidade do sistema:

- **"connect is not defined" no `authController.js`:** Erro de escopo ou importação que impedia a conexão com o banco de dados. A resolução garante que o controlador de autenticação possa interagir corretamente com o BD para validar credenciais e gerenciar sessões.
- **"INSERT has more target columns than expressions" no `deviceModel.js`:** Erro comum de SQL que indica uma inconsistência entre o número de colunas e valores fornecidos em uma instrução INSERT. A correção assegura que os dados de sessão sejam inseridos corretamente, incluindo o crucial `user_id`.
- **"value too long for type character varying(20)" no log:** Este erro destacou uma limitação na capacidade de registro de logs. A correção (aumento do tamanho das colunas `operacao` e `descricao`) é vital para a segurança, pois garante que mensagens de log detalhadas, incluindo possíveis alertas de segurança, não sejam

truncadas, permitindo uma análise forense mais completa.
- **"violates check constraint users_user_type_check" ao semear dados:** Essencial para o ambiente de desenvolvimento e testes. A correção na semeadura de dados (`src/config/seedData.js`) assegura que o banco de dados seja populado corretamente com usuários de ambos os tipos (Comum e Premium), permitindo testar a lógica de sessões para ambos os perfis.

Explicação do Fluxo:

1. **Início e Verificação de Dispositivo:**
    - O cliente verifica se possui uma `api_key` salva.
    - Se não tiver, envia requisição para `/devices/register` para obter novas `api_key` e `cripto_key`. A sessão inicial tem `user_id` como `null`.
    - Se tiver, recupera as chaves salvas.
2. **Tentativa de Login:**
    - Com a `api_key` do dispositivo, o cliente envia uma requisição POST para `/auth/login` com email, password, `X-API-Key` e, se criptografado, `X-Encryption-Key-Id`.
3. **Processamento no Backend:**
    - `requireApiKey` (Middleware): Valida a `X-API-Key`. Se inválida, a requisição é recusada. Se válida, a sessão do dispositivo é anexada a `req.deviceSession`.
    - `decryptRequest` (Middleware): Decifra o corpo da requisição se criptografado.
    - `authController.login`:
        - Valida credenciais.
        - Determina `user_type` (Client ou Premium) e `maxAllowedSessions`.
        - Chama `deviceModel.deactivateExcessUserSessions()` para desativar sessões antigas que excedam o limite.
        - A sessão do dispositivo atual é atualizada para `ativa=TRUE` e associada ao `user_id`.
        - Um novo JWT é gerado, contendo `userId`, `userType` e `deviceApiKey`.
        - A resposta de sucesso (200 OK) é enviada ao cliente.
4. **Acesso Concedido e Requisições Protegidas:**
    - Com o JWT e a `api_key` do dispositivo, o cliente faz requisições a rotas protegidas.
    - Para cada requisição, `requireApiKey` valida a `X-API-Key`.
    - `authenticate` valida o JWT e verifica se a `deviceApiKey` no payload do JWT ainda está ativa no banco de dados. Se desativada, a requisição é negada.
5. **Processo de Logout:**
    - Uma requisição para `/auth/logout` é enviada.
    - No backend, o JWT é adicionado a uma *blacklist*.
    - A `api_key` do dispositivo que fez o logout é explicitamente desativada na tabela `sessao`.

Conclusão:

O aprimoramento do PobreFlix com este sistema de gerenciamento de sessões representa um salto significativo na segurança e na adequação do aplicativo