Implementação de OAuth 2.0 (PKCE) em SPA Segura

Disciplina:

Segurança

Objetivo: Implementar uma SPA (Single Page Application) segura, utilizando o fluxo **OAuth 2.0 Authorization Code com PKCE**, e demonstrar controle de autorização através de **Escopos (Scopes)**.

1. Descrição do Cenário e Configuração

1.1 Escolha do Cenário

Seu grupo deve escolher **um** dos cenários abaixo. O cenário define qual API Externa (Resource Server) sua aplicação irá acessar e quais permissões (Scopes) serão necessárias.

Cenário	Perfil A (Viewer)	Perfil B (Manager)	API Externa	Ação
1. GitHub Repo Manager	`read:user`	`repo:write`	GitHub API	Listar repos. (A) ou Criar repos. (B)
2. Google Calendar	`calendar.readonly`	`Calendars.ReadWrite`	Google Calendar API	Ver eventos (A) ou Criar eventos (B)
3. Microsoft Outlook	`Mail.Read`	`Mail.Send`	Microsoft Graph API	Listar 5 emails (A) ou Enviar email de teste (B)

4. Azure DevOps/Projetos	`vso.code_read`	`vso.code_manage`	Azure DevOps API	Ver código/issues (A) ou Criar/Gerenciar PRs (B)
5. Sistema Financeiro	`account.read`	`payment_transfer`	API de Pagamentos (Mock)	Ver extrato/saldo (A) ou Iniciar transferência (B)
6. Player de Mídia (Spotify)	`user-read-playback-state`	`user-modify-playback-state`	Spotify API	Ver o que está tocando (A) ou Botões Play/Pause/Skip (B)

1.2 O Cliente (SPA no GitHub Pages)

- **Tecnologia:** A aplicação deve ser desenvolvida em **JavaScript/React/Vue/Svelte** (Front-end puro, sem servidor backend próprio).
- **Hospedagem:** O deployment final deve ser feito no **GitHub Pages**.
- **Função:** Sua SPA é um **Cliente Público**.

2. Requisitos de Implementação do Fluxo

O foco da atividade está na correta implementação do fluxo de segurança.

Requisito A: Fluxo PKCE (Proof Key for Code Exchange)

O JavaScript da sua aplicação deve executar o fluxo PKCE para a troca segura de tokens, provando que o Cliente é legítimo sem usar um `client_secret` (que seria inseguro em uma SPA).

Ações de Implementação:

1. **1. Pré-Redirecionamento:** Gerar um `code_verifier` (segredo criptograficamente aleatório) e seu `code_challenge` (HASH SHA256 do `verifier`). Armazenar o `code_verifier` no `sessionStorage`.

2. **2. Redirecionamento:** Redirecionar para o Authorization Server com `response_type=code`, `code_challenge`, `code_challenge_method=S256` e o `state`.
3. **3. Troca de Token:** No *callback*, fazer um POST para o endpoint `/token` do provedor, enviando o `code` obtido da URL **e** o `code_verifier` resgatado do `sessionStorage`.

Requisito B: Controle de Autorização (Scopes)

A interface do usuário deve mudar drasticamente baseada nas permissões.

1. **Validação de Escopo:** Após a obtenção do `access_token`, você deve **decodificar o JWT** (se disponível) ou usar o endpoint `/userinfo` para verificar quais escopos (`scp` ou `scope` claim) foram efetivamente concedidos.
2. **Renderização Condicional:**
    - **Usuário Viewer:** Mostrar apenas as funcionalidades (botões/links) associadas ao Perfil A (`read:*`).
    - **Usuário Manager:** Mostrar todas as funcionalidades, incluindo as associadas ao Perfil B (`write:*` ou `manage:*`).
    - **Deve ser escolhido apenas um perfil para implementar.**
3. **Prova de Uso:** Pelo menos **um** botão (um de Leitura ou de Escrita/Gestão) devem funcionar: ao serem clicado, ele deve usar o `access_token` em um cabeçalho `Authorization: Bearer` para fazer uma chamada real à API externa escolhida no seu Cenário.

Requisito C: Segurança e Armazenamento

Mecanismos de Segurança Essenciais:

- **Proteção contra CSRF**
    - Gerar o `state` aleatório, enviá-lo na requisição de login e **validar** se o `state` retornado na URL é o mesmo.
- **Armazenamento de Tokens**
    - O `access_token` **NÃO DEVE** ser armazenado em `localStorage`. Use apenas a **memória da aplicação** (variável JavaScript) ou, no máximo, `sessionStorage`.
- **Logout Seguro**
    - Limpar o token da memória/sessão local **E** redirecionar o usuário para o `end_session_endpoint` do Provedor para encerrar a sessão remota.

3. Requisito de DevOps e Segurança de Credenciais (GitHub Actions)

O projeto deve demonstrar a prática de não *hardcodear* chaves, mesmo as públicas.

1. **Gerenciamento de Segredo:** O `CLIENT_ID` da sua aplicação OAuth deve ser configurado como um **Secret** (ex: `AUTH_CLIENT_ID`) no seu repositório GitHub.
2. **Workflow de CI/CD:** O grupo deve criar o arquivo `.github/workflows/deploy.yml` que:
    - Garanta que o `CLIENT_ID` seja **injetado como variável de ambiente** no processo de *build* da SPA.
    - Faça o deploy automático dos arquivos estáticos para o **GitHub Pages**.

4. Critérios de Avaliação e Entregáveis

Critério	Detalhes da Avaliação	Entregáveis
1. Fluxo de Segurança (PKCE/State)	Implementação correta da geração, armazenamento (`sessionStorage`), e validação do `code_verifier` e do `state`.	Código-fonte (`.js`/`.jsx`) e página online (Link).
3. Segurança e DevOps	Uso correto do **GitHub Actions** para injetar o `CLIENT_ID`. Token armazenado corretamente.	Arquivo `.github/workflows/deploy.yml`.
4. Documentação Técnica	Explicando as decisões técnicas e de segurança (Por que PKCE? Como funciona a validação de `state`?).	Manuscrito

Ponto de Partida: O ponto de partida é a página pública com o botão de login. O ponto de chegada é uma dashboard privada que funciona de maneira diferente para o Usuário Viewer e o Usuário Manager.