Laboratório: login em um site estático no Cloudflare Pages

1. Visão geral

Neste laboratório, vamos publicar uma página estática e suas rotas de autenticação em um único projeto do Cloudflare Pages. O conteúdo público será servido em https://NOME-DO-PROJETO.pages.dev. As Pages Functions do mesmo projeto iniciarão o login, receberão as respostas do Google e do GitHub, confirmarão a identidade apresentada por cada provedor e criarão sessões locais em um banco D1.

Essa composição mantém a página e as rotas dinâmicas na mesma origem. Por isso, o navegador poderá enviar um cookie opaco de sessão sem CORS, sem compartilhar cookies entre domínios e sem exigir a compra de um domínio próprio.

O laboratório será realizado pelo navegador. Não será necessário instalar Node.js, `npm`, `npx`, Wrangler ou qualquer biblioteca de pacotes. O GitHub entregará o código ao Cloudflare Pages, enquanto o painel da Cloudflare será usado para criar o projeto, o banco, as ligações e os segredos.

O resultado não tornará privados os arquivos da pasta `public`. Tudo o que for publicado como arquivo estático continuará acessível por sua URL. A sessão protegerá somente as rotas dinâmicas que consultarem e validarem o cookie.

Duração sugerida

Três horas, distribuídas em dois encontros quando a turma precisar criar os registros dos provedores durante a aula.

Organização

Trabalhe em uma dupla. Uma estudante realizará a configuração e a outra conferirá nomes, URLs e evidências. Troquem os papéis depois do primeiro provedor.

Você terá que criar uma página html para o login, e esta página estará, ou será, o novo index.html

Resultado esperado

Ao final, a dupla deverá demonstrar:

1. a publicação do projeto em um endereço `pages.dev`;
2. o início do login com Google e GitHub;
3. o retorno à URL específica de cada provedor;
4. a criação de uma sessão opaca no D1;
5. a consulta autenticada a `/api/me`;
6. a revogação local da sessão por `/oauth/logout`;
7. a rejeição das falhas obrigatórias;

8. a permanência do conteúdo estático como recurso público.

2. Objetivos de aprendizagem

Depois da prática, a estudante deverá conseguir:

- distinguir autenticação, conta local, autorização local e sessão;
- justificar a escolha de um cliente confidencial executado em uma Pages Function;
- explicar por que PKCE complementa o Client Secret em vez de substituí-lo;
- relacionar `state`, `nonce` e `code_verifier` às transações que eles protegem;
- registrar um cliente Web no Google e uma OAuth App no GitHub com URLs de retorno exatas;
- conservar transações e sessões revogáveis em um banco D1;
- validar o token de identidade do Google e confirmar a identidade do GitHub pela API oficial;
- testar o caminho feliz e as falhas que comprovam as fronteiras de segurança.

3. Arquitetura do laboratório

O endereço `https://NOME-DO-PROJETO.pages.dev` terá dois destinos lógicos dentro do mesmo projeto:

Caminho	Destino	Responsabilidade
`/`, CSS, JavaScript e imagens	Cloudflare Pages	Publicação estática e pública
`/oauth/*`	Pages Functions	Início do login, URLs de retorno e saída
`/api/*`	Pages Functions	Recursos que exigem a sessão local

O navegador não receberá access_token, refresh_token nem Client Secret. No fluxo do Google, o id_token chegará à Pages Function e será validado no servidor. No fluxo do GitHub, a Function usará o access_token apenas para consultar o perfil autenticado, revogará a autorização concedida à OAuth App e conservará o identificador numérico estável da conta. Nos dois casos, o navegador receberá apenas um identificador aleatório de sessão local.

Estrutura do repositório

Considere a seguinte estrutura para o seu repositório:

```
oauth-pages-lab/
├── public/
│   ├── index.html
│   ├── app.js
│   └── styles.css
└── functions/
    ├── _shared/
```

```
│   ├── crypto.js
│   ├── cookies.js
│   ├── providers.js
│   └── oidc.js
├── api/
│   ├── health.js
│   └── me.js
└── oauth/
    ├── login/
    │   └── [provider].js
    ├── callback/
    │   └── [provider].js
    └── logout.js
```

A pasta `public` contém os arquivos que qualquer visitante poderá obter. A pasta `functions`, situada na raiz do projeto, contém o código executado na infraestrutura da Cloudflare. Ela não deverá ser colocada dentro de `public`.

Você deve adaptar a estrutura do projeto, para uma estrutura semelhante a esta, ou ter conhecimento total da estrutura que escolher.

4. Contas e materiais necessários

Cada dupla precisará de:

- uma conta GitHub com acesso ao repositório da prática;
- uma conta Cloudflare no plano gratuito;
- uma conta no Google Cloud;
- permissão para registrar uma OAuth App na conta GitHub usada pela dupla;
- um navegador com ferramentas de desenvolvimento;
- uma janela privativa para as sessões administrativas;
- acesso ao ambiente de entrega de evidências da disciplina.

Não será necessário comprar um domínio. O endereço `pages.dev` fornecido pelo Cloudflare Pages será o endereço canônico da aplicação durante toda a prática.

Atenção à conta GitHub: a OAuth App poderá pertencer à conta pessoal de uma integrante ou a uma organização na qual ela tenha permissão administrativa. O cadastro da aplicação não exige assinatura nem cartão de crédito.

Você não precisa, nem deve, pagar nenhuma assinatura para resolver este exercício. O cliente Web do Google, a OAuth App do GitHub, o projeto Pages e o banco D1 usados nesta prática podem ser configurados sem cadastro de cartão de crédito.

5. Regras de segurança da prática

1. Nunca publique um Client Secret no GitHub, no HTML, em uma captura de tela ou em uma mensagem da equipe.
2. Cadastre os Client Secrets somente como segredos criptografados no painel do Cloudflare

Pages.

3. Não crie arquivos locais para guardar segredos.
4. Não registre códigos de autorização, tokens, cookies, `state`, `nonce` nem o corpo da troca de tokens.
5. Não use um endereço de implantação de prévia como endereço de produção. A URL de retorno cadastrada no provedor deverá permanecer estável e exata.
6. Encerre as sessões administrativas do Google, do GitHub e da Cloudflare ao terminar em um computador compartilhado.

6. Evidências que serão entregues

Guarde as evidências fora do repositório público. A pasta entregue deverá conter:

- `01-pages-configuracao.pdf`, com o nome do projeto, a ramificação de produção e as opções de construção;
- `02-google-retorno.txt`, com a URL de retorno cadastrada no Google;
- `03-github-retorno.txt`, com a página inicial e a URL de retorno cadastradas no GitHub;
- `04-d1-esquema.txt`, com o esquema consultado no console D1;
- `05-inicio-login-google.pdf`, com os cabeçalhos saneados do início do login;
- `06-inicio-login-github.pdf`, com os cabeçalhos saneados do início do login;
- `07-testes-falha.md`, com os casos executados e seus resultados;
- `08-aceitacao.md`, com a lista final assinada pela dupla.

Substitua qualquer valor sensível por `[REMOVIDO]`. Uma captura que mostre apenas o nome de um segredo é aceitável. Uma captura que revele seu valor deverá ser descartada.

Todos os arquivos da entrega precisam estar disponíveis no seu repositório na pasta `public/entrega1`. A primeira parte da avaliação será feita automaticamente nesta pasta. Se não for possível acessar a pasta, ou ela não conter exatamente os arquivos com os nomes e conteúdos descritos acima, a entrega será zerada e o aluno não terá direito a prova de autoria.

7. Etapa 0: preparar o repositório pelo GitHub

Atenção: o passo a passo aqui descrito pode ter sido modificado, no Github, Google, ou no Cloudflare. Considere como a sugestão de um processo que precisa ser validado e atualizado.

7.1 Criar o repositório

No GitHub, selecione **New repository** e crie um repositório com o nome do seu projeto. Mantenha `main` como a ramificação padrão e permita o acesso dos dois integrantes do grupo. Um repositório privado reduz a exposição acidental do trabalho, mas ainda assim não deverá receber segredos. **Já que estamos hospedando no Cloudflare, este será o novo repositório do seu projeto**.

O projeto não deverá conter `package.json`, `package-lock.json`, `node_modules` nem `wrangler.jsonc`. As Pages Functions deste laboratório usarão apenas JavaScript e APIs Web oferecidas pelo ambiente de execução da Cloudflare.

7.2 Conferir a estrutura

Pela interface Web, use **Add file > Create new file**. Uma barra no nome criará a pasta correspondente. Copie `public/` todo o conteúdo do dashboard que estamos usando como modelo de aplicação. Modifique o arquivo `public/index.html` para que ele tenha uma opção de login ou crie `public/index.html` com o conteúdo mínimo:

```
<!doctype html>
<html lang="pt-BR">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Laboratório de autenticação</title>
  </head>
  <body>
    <main>
      <h1>Laboratório de autenticação</h1>
      <p id="status">A sessão ainda não foi consultada.</p>
    </main>
  </body>
</html>
```

Crie também `functions/api/health.js`:

```
export function onRequestGet() {
  return Response.json(
    { status: "ok" },
    { headers: { "Cache-Control": "no-store" } }
  );
}
```

Confirme cada arquivo diretamente em `main`. Ao terminar, `public` e `functions` deverão ser pastas irmãs na raiz.

Ponto de verificação 0

- ☐ o repositório está acessível às duas integrantes;
- ☐ a ramificação `main` existe;
- ☐ a pasta `public` contém a página estática, o dashboard e a pasta `entrega1`;
- ☐ a pasta `functions` está na raiz;
- ☐ não existem arquivos de configuração ou dependências de Node.js;
- ☐ nenhum segredo está versionado.

8. Etapa 1: criar o projeto no Cloudflare Pages

8.1 Conectar o GitHub

No painel da Cloudflare:

1. abra **Workers & Pages**;
2. escolha **Create application > Pages > Connect to Git**;
3. autorize o acesso somente ao repositório da prática, quando essa opção estiver disponível;
4. selecione o repositório da dupla;
5. defina `main` como a ramificação de produção.

8.2 Configurar a publicação

Preencha a construção com estes valores:

Campo	**Valor**
Framework preset	`None`
Build command	deixar vazio
Build output directory	`public`
Root directory	deixar vazio

Selecione **Save and Deploy**. Ao concluir, o painel mostrará uma URL semelhante a:

```
https://oauth-aula-equipe-07.pages.dev
```

Copie exatamente a URL atribuída à equipe. Neste roteiro, ela será chamada de `URL_BASE`. Não acrescente uma barra ao final quando cadastrar variáveis ou comparar o cabeçalho `Origin`.

8.3 Confirmar as Functions

Abra a implantação concluída e confirme que a seção de Functions reconheceu os arquivos da pasta `functions`. Depois, visite:

```
URL_BASE/api/health
```

O projeto-modelo deverá responder `200`. Uma resposta `404` indica que a pasta `functions` está no lugar incorreto ou que a implantação analisada é anterior à inclusão da rota.

Ponto de verificação 1

- ☐ a raiz do site responde por HTTPS;
- ☐ a URL termina em `.pages.dev`;
- ☐ a ramificação de produção é `main`;
- ☐ não existe um comando de construção;

- ☐ o diretório publicado é `public`;
- ☐ `/api/health` responde `200`.

Registre a configuração em `evidencias/01-pages-configuracao.pdf` **sem incluir identificadores privados da conta**. **Qualquer identificador privado disponível no pdf zera o trabalho.**

9. Etapa 2: criar e ligar o banco D1

9.1 Criar o banco pelo painel

No painel da Cloudflare:

1. abra **Storage & Databases > D1 SQL Database**;
2. selecione **Create database**;
3. use o nome `oauth-sessions-EQUIPE`;
4. abra o banco criado e selecione **Console**.

Cole e execute o seguinte esquema:

```
CREATE TABLE oauth_transactions (
  id_hash TEXT PRIMARY KEY,
  provider TEXT NOT NULL CHECK (provider IN ('google', 'github')),
  state_hash TEXT NOT NULL,
  nonce TEXT,
  code_verifier TEXT NOT NULL,
  expires_at INTEGER NOT NULL
);

CREATE INDEX oauth_transactions_expiry
  ON oauth_transactions (expires_at);

CREATE TABLE sessions (
  id_hash TEXT PRIMARY KEY,
  issuer TEXT NOT NULL,
  subject TEXT NOT NULL,
  email TEXT,
  display_name TEXT,
  expires_at INTEGER NOT NULL,
  created_at INTEGER NOT NULL
);

CREATE INDEX sessions_expiry
  ON sessions (expires_at);
```

Execute no console a consulta:

```
SELECT name, type
FROM sqlite_schema
WHERE name NOT LIKE 'sqlite_%'
ORDER BY type, name;
```

O resultado deverá mostrar as duas tabelas e os dois índices.

9.2 Criar a ligação com o projeto

Volte a **Workers & Pages**, abra o projeto e siga para **Settings > Bindings > Add > D1 database bindings**. Use:

```
Variable name: DB
D1 database: oauth-sessions-EQUIPE
```

Salve a ligação para os ambientes de produção e prévia quando o painel oferecer essa escolha. Depois, solicite uma nova implantação da versão atual. As Pages Functions acessarão o banco como `context.env.DB`.

Ponto de verificação 2

- ☐ as tabelas `oauth_transactions` e `sessions` existem;
- ☐ os índices de expiração existem;
- ☐ o projeto possui uma ligação D1 chamada exatamente `DB`;
- ☐ uma nova implantação foi realizada depois da ligação;
- ☐ nenhum identificador do banco foi inserido no código.

Copie somente os nomes e os tipos devolvidos por `sqlite_schema` para `evidencias/04-d1-esquema.txt`.

10. Etapa 3: registrar o cliente no Google

10.1 Configurar o consentimento

Abra o Google Cloud Console em uma janela privativa. Crie ou selecione um projeto e configure a tela de consentimento. Para uma atividade com contas pessoais, mantenha o aplicativo em teste e cadastre as contas da dupla como usuárias de teste.

Solicite somente estes escopos:

```
openid email profile
```

10.2 Criar o cliente Web

Crie um cliente OAuth do tipo **Web application**. Cadastre exatamente:

```
URL_BASE/oauth/callback/google
```

Por exemplo:

```
https://oauth-aula-equipe-07.pages.dev/oauth/callback/google
```

Não acrescente uma barra final e não use um padrão curinga. O laboratório não terá uma URL de retorno para `localhost`, pois todo o teste ocorrerá na implantação HTTPS.

Copie o Client ID e o Client Secret para o painel da Cloudflare na Etapa 5. Não baixe um arquivo de credenciais em um computador compartilhado.

Ponto de verificação 3

- ☐ o cliente é do tipo Web;
- ☐ existe uma única URL de retorno de produção;
- ☐ a URL usa HTTPS e o domínio `pages.dev` da equipe;
- ☐ apenas `openid`, `email` e `profile` serão solicitados;
- ☐ o Client Secret não entrou no repositório nem nas evidências.

Registre somente a URL de retorno em `evidencias/02-google-retorno.txt`.

11. Etapa 4: registrar a OAuth App no GitHub

11.1 Criar a OAuth App

No GitHub, abra Settings > Developer settings > OAuth apps e selecione New OAuth App. O GitHub recomenda GitHub Apps para aplicações de produção com permissões finas e tokens de curta duração. Neste laboratório, vamos usar uma OAuth App porque o segundo provedor servirá somente para identificar a conta durante um fluxo Web.

Preencha Application name com um nome que identifique a equipe. Em Homepage URL, use exatamente URL_BASE. Não ative Device Flow, pois a aplicação terá navegador e uma URL de retorno estável.

11.2 Cadastrar a URL de retorno e as credenciais

Em Authorization callback URL, cadastre exatamente:

```
URL_BASE/oauth/callback/github
```

Registre a aplicação e gere um Client Secret. O pedido de autorização não deverá solicitar repo, user:email nem offline_access. Sem esses escopos, o laboratório poderá consultar o perfil público da conta autenticada e não receberá acesso a repositórios. Como os tokens com expiração estão ativados por padrão em novas OAuth Apps, a resposta da troca também poderá conter um refresh_token. A Function não deverá conservar nem expor nenhum desses tokens.

Ponto de verificação 4

- ☐ a OAuth App foi registrada na conta correta;
- ☐ a página inicial e a URL de retorno usam o domínio pages.dev da equipe;
- ☐ Device Flow permanece desativado;
- ☐ o Client Secret possui responsável pela rotação;
- ☐ não foram solicitados escopos desnecessários.

Registre a página inicial e a URL de retorno em evidencias/03-github-retorno.txt.

12. Etapa 5: cadastrar configurações e segredos no Pages

No projeto do Pages, abra **Settings > Variables and Secrets > Add**. Cadastre as seguintes variáveis como texto simples:

Nome	Valor
`PUBLIC_BASE_URL`	`URL_BASE`, sem a barra final
`GOOGLE_CLIENT_ID`	Client ID do Google
`GITHUB_CLIENT_ID`	Client ID do GitHub

Cadastre separadamente os valores abaixo e selecione **Encrypt** antes de salvar:

Nome do segredo	Valor
`GOOGLE_CLIENT_SECRET`	Client Secret do Google
`GITHUB_CLIENT_SECRET`	Client Secret do GitHub

Configure os valores para produção. Se também forem configurados para implantações de prévia, lembre que essas implantações possuem outros hospedeiros e não funcionarão como URLs de retorno sem registros adicionais nos provedores. Para evitar essa multiplicação, todos os testes avaliados usarão a implantação de produção.

Solicite uma nova implantação depois de salvar as variáveis e os segredos. O código os acessará por `context.env`, mas **seus valores não deverão aparecer em mensagens, respostas ou registros**.

Ponto de verificação 5

- ☐ `PUBLIC_BASE_URL` é exatamente a URL `pages.dev` de produção;
- ☐ os Client IDs do Google e do GitHub estão cadastrados como variáveis;
- ☐ a OAuth App do GitHub usa a URL de retorno exata;
- ☐ os Client Secrets do Google e do GitHub estão criptografados;
- ☐ uma nova implantação foi realizada;
- ☐ nenhuma credencial foi adicionada ao GitHub.

13. Etapa 6: implementar as Pages Functions

Edite os arquivos pelo GitHub ou use os arquivos já fornecidos no projeto-modelo. Cada confirmação em `main` produzirá uma nova implantação. Aguarde o estado **Success** antes de testar a alteração.

13.1 Rotas obrigatórias

Método	Caminho	Arquivo	Responsabilidade
`GET`	`/oauth/login/google`	`functions/oauth`	Criar a transação e

Método	Caminho	Arquivo	Responsabilidade
		`/login/[provider].js`	redirecionar ao Google
`GET`	`/oauth/login/github`	`functions/oauth/login/[provider].js`	Criar a transação e redirecionar ao GitHub
`GET`	`/oauth/callback/google`	`functions/oauth/callback/[provider].js`	Validar a resposta do Google e criar a sessão
`GET`	`/oauth/callback/github`	`functions/oauth/callback/[provider].js`	Consultar o perfil no GitHub, revogar a autorização e criar a sessão
`GET`	`/api/me`	`functions/api/me.js`	Resolver a sessão e devolver o perfil mínimo
`POST`	`/oauth/logout`	`functions/oauth/logout.js`	Revogar a sessão local

O parâmetro dinâmico estará disponível como context.params.provider. Qualquer provedor diferente de google ou github deverá produzir 404 sem revelar detalhes internos.

13.2 Valores aleatórios e resumos

Use `crypto.getRandomValues()` para gerar 32 bytes aleatórios e codifique-os em Base64URL sem preenchimento. O resultado terá 43 caracteres e poderá representar o identificador bruto da transação, `state`, `nonce`, `code_verifier` ou o identificador bruto da sessão.

Use `crypto.subtle.digest()` com SHA-256 para calcular:

- o `code_challenge` derivado do `code_verifier`;
- o resumo do cookie de transação;
- o resumo de `state`;
- o resumo do cookie de sessão.

O D1 deverá guardar apenas os resumos dos cookies. Um vazamento do banco, portanto, não entregará imediatamente um cookie reutilizável.

13.3 Início do login

Ao receber `/oauth/login/{provider}`, a Function deverá:

1. aceitar somente google ou github;
2. gerar a transação e gravar os resumos, code_verifier e a expiração no D1. O nonce será gravado somente para o Google;
3. criar o cookie temporário;

4. montar o pedido de autorização com os dados do provedor;
5. responder com um redirecionamento `302`.

O cookie temporário deverá seguir este contrato:

```
Set-Cookie: __Host-oauth-tx=VALOR_ALEATORIO; Path=/; HttpOnly; Secure;
SameSite=Lax; Max-Age=600
```

O pedido de autorização deverá conter:

```
Comuns aos dois provedores:
client_id
redirect_uri
response_type=code
state
code_challenge
code_challenge_method=S256

Somente no Google:
scope=openid email profile
nonce

No GitHub:
omitir scope e nonce
```

O Client Secret e o code_verifier nunca deverão aparecer nessa URL. O nonce pertence ao fluxo OIDC do Google e não será enviado ao GitHub.

13.4 Processar a URL de retorno

Ao receber a resposta do provedor, a Function deverá:

1. recusar `error` ou a ausência de `code` e `state`;
2. exigir o cookie `__Host-oauth-tx`;
3. calcular seu resumo e localizar uma transação não expirada;
4. comparar o resumo de `state` com o valor conservado no D1;
5. apagar a transação antes de concluir o fluxo;
6. trocar o código com o code_verifier e o Client Secret do provedor correto;
7. validar a resposta de identidade conforme o contrato do provedor;
8. criar uma sessão opaca;
9. limpar o cookie temporário;
10. redirecionar para `PUBLIC_BASE_URL`.

Não registre o corpo enviado ao ponto de terminação de tokens nem sua resposta completa.

13.5 Confirmar a identidade sem bibliotecas externas

Os dois provedores devolvem um código de autorização, mas entregam provas diferentes depois da troca. O Google fornece um id_token OIDC. O GitHub fornece um access_token OAuth, que deverá ser usado em uma chamada autenticada à API antes de ser revogado.

1. para o Google, separar as três partes do JWT e recusar qualquer formato diferente;

2. decodificar o cabeçalho e exigir alg igual a RS256;
3. obter o documento de descoberta OIDC do emissor esperado;
4. obter o conjunto de chaves JWKS indicado por jwks_uri;
5. selecionar uma chave pública pelo kid do cabeçalho;
6. importar a JWK com crypto.subtle.importKey();
7. verificar a assinatura com crypto.subtle.verify() e RSASSA-PKCS1-v1_5;
8. validar iss, aud, exp, iat e nonce antes de usar sub, nome e e-mail;
9. para o GitHub, exigir access_token e token_type compatível com Bearer na resposta da troca;
10. chamar GET https://api.github.com/user com Authorization: Bearer, Accept: application/vnd.github+json e X-GitHub-Api-Version: 2026-03-10;

A resposta do GitHub deverá ter estado 200 e um id inteiro. Use esse id como identificador estável, converta-o em texto para subject e use name ou login apenas para apresentação. O e-mail poderá ser nulo porque o laboratório não solicitará user:email.

Use issuer e subject como chave conceitual da identidade. Para o Google, subject receberá sub. Para o GitHub, issuer será https://github.com e subject receberá o id numérico convertido em texto. Depois da consulta a /user, envie DELETE para https://api.github.com/applications/{client_id}/grant com autenticação Basic formada pelo Client ID e pelo Client Secret e com o access_token no corpo JSON. Exija a resposta 204 antes de criar a sessão local. Essa revogação elimina a autorização e todos os tokens associados à OAuth App para a conta. O nome, o login e o e-mail servem para apresentação e não deverão conceder privilégios por comparação textual.

13.6 Criar e revogar a sessão

A sessão final durará oito horas. O cookie terá este contrato:

```
Set-Cookie: __Host-session=VALOR_ALEATORIO; Path=/; HttpOnly; Secure;
SameSite=Strict; Max-Age=28800
```

A rota `/api/me` calculará o resumo do cookie, procurará uma sessão ainda válida e devolverá apenas o perfil mínimo. A resposta deverá usar `Cache-Control: no-store`.

A rota `/oauth/logout` deverá:

1. aceitar somente `POST`;
2. exigir um cabeçalho `Origin` exatamente igual a `PUBLIC_BASE_URL`;
3. remover a linha da sessão no D1;
4. expirar o cookie;
5. responder com `Cache-Control: no-store`.

O logout revogará a sessão local. Ele não encerrará a sessão global da usuária no Google nem no GitHub.

Ponto de verificação 6

- ☐ todas as rotas obrigatórias foram reconhecidas pelo Pages;
- ☐ a aplicação usa `context.env.DB`;
- ☐ nenhuma variável global mutável conserva dados de uma requisição;
- ☐ os valores aleatórios e os resumos usam Web Crypto;
- ☐ a validação do Google e a consulta ao GitHub não dependem de pacotes externos;
- ☐ a transação é apagada antes da conclusão do retorno;
- ☐ a sessão só é criada depois da confirmação completa da identidade;
- ☐ erros, retornos e respostas de sessão usam `Cache-Control: no-store`.

14. Etapa 7: ligar a página estática

Em `public/index.html`, inclua os links:

```
<a href="/oauth/login/google">Entrar com Google</a>
<a href="/oauth/login/github">Entrar com GitHub</a>
```

Em `public/app.js`, consulte a sessão na mesma origem:

```
fetch("/api/me", { credentials: "same-origin" })
  .then((response) => (response.ok ? response.json() : null))
  .then((user) => {
    const status = document.getElementById("status");
    status.textContent = user
      ? `Sessão de ${user.email ?? user.displayName}.`
      : "Nenhuma sessão neste navegador.";
  });
```

Use um formulário `POST` para a saída:

```
<form method="post" action="/oauth/logout">
  <button type="submit">Sair</button>
</form>
```

Depois de confirmar as alterações em `main`, aguarde a implantação e abra a URL de produção. Ocultar um link depois da consulta a `/api/me` não torna seu arquivo de destino privado. Qualquer recurso colocado em `public` continuará acessível pela URL correspondente.

15. Etapa 8: testar a implantação pelo navegador

15.1 Conferir a implantação

No painel do Pages, abra a implantação mais recente e confirme:

- ☐ o estado é **Success**;
- ☐ a implantação pertence a `main`;

- ☐ as Functions foram compiladas;
- ☐ não existem mensagens sobre uma ligação `DB` ausente;
- ☐ os registros não exibem tokens, códigos ou segredos.

15.2 Inspecionar o início do login

Abra as ferramentas de desenvolvimento do navegador, selecione **Network** e ative **Preserve log**. Visite `/oauth/login/google` sem concluir o login imediatamente. Localize a resposta da rota inicial e confira:

- ☐ o estado HTTP é `302`;
- ☐ a resposta cria `__Host-oauth-tx`;
- ☐ `Location` aponta para o Google;
- ☐ `response_type=code` está presente;
- ☐ `code_challenge_method=S256` está presente;
- ☐ a URL de retorno é a URL `pages.dev` exata;
- ☐ Client Secret e `code_verifier` estão ausentes.

Repita a inspeção para /oauth/login/github. No pedido ao GitHub, confirme que nonce e os escopos repo, user:email e offline_access estão ausentes. Antes de salvar as evidências, oculte o valor do cookie, state e code_challenge.

15.3 Executar o caminho feliz

No navegador:

1. abra `URL_BASE`;
2. inicie o login com Google;
3. conclua a autenticação no domínio do Google;
4. confirme o retorno para `URL_BASE`;
5. confirme que `/api/me` devolve o perfil mínimo;
6. confira que o armazenamento local e o armazenamento de sessão não contêm tokens;
7. execute o logout e confirme que `/api/me` passa a responder `401`;
8. repita o fluxo com GitHub e confirme que /api/me funciona mesmo quando a conta não expõe um e-mail público.

Os dois fluxos deverão voltar para o mesmo hospedeiro que iniciou a autenticação. Uma implantação de prévia com outro endereço não satisfará esse contrato. O fluxo do GitHub deverá usar o token apenas para obter /user e revogar a autorização antes de criar a sessão local.

16. Etapa 9: executar os testes de falha

Registre cada caso em `evidencias/07-testes-falha.md` com quatro campos: preparação, pedido enviado, resultado esperado e resultado observado.

Caso 1: retorno sem cookie temporário

Inicie o login em uma janela comum e pare na página do provedor. Copie a URL de autorização para uma janela privativa que não possua `__Host-oauth-tx` e conclua o login nessa segunda janela. A rota de retorno deverá recusar a resposta e não criar uma sessão.

Caso 2: `state` alterado

Inicie outro login e pare na página do provedor antes de fornecer as credenciais. Na barra de endereço, altere um único caractere do parâmetro `state` e prossiga. A rota de retorno deverá recusar a resposta antes de trocar o código.

Não registre a URL modificada, pois ela contém valores transitórios.

Caso 3: reutilização da transação

Depois de uma conclusão bem-sucedida, localize a requisição de retorno no painel **Network** e use **Copy URL**. Abra a URL novamente. A transação já terá sido removida e a repetição deverá falhar.

Caso 4: sessão expirada

Depois de criar uma sessão de teste, abra o console D1 e execute:

```
UPDATE sessions
SET expires_at = 0;
```

Recarregue a página. `/api/me` deverá responder `401`. Esse banco pertence apenas ao laboratório, portanto a alteração poderá encerrar todas as sessões da equipe.

Caso 5: origem inválida na saída

Com uma sessão válida aberta em `URL_BASE`, abra outra origem, como `https://example.com`, e execute no console do navegador:

```
fetch("URL_BASE/oauth/logout", {
  method: "POST",
  credentials: "include"
});
```

A rota deverá recusar a operação. Volte à aba de `URL_BASE` e confirme que a sessão original permanece válida. O teste comprova a conferência de origem mesmo quando o navegador também impõe suas próprias regras de cookies e CORS.

Caso 6: reutilização do cookie revogado

Em uma sessão exclusiva do laboratório, copie temporariamente o valor do cookie `__Host-session` pelas ferramentas de desenvolvimento. Execute o logout, tente restaurar o mesmo valor e consulte `/api/me`. Como a linha correspondente foi removida do D1, a resposta deverá ser `401`.

Apague a cópia imediatamente. Nunca coloque esse valor na evidência entregue.

17. Critérios de aceitação

Copie esta lista para `evidencias/08-aceitacao.md`:

- ☐ o site é servido pelo endereço `pages.dev` atribuído à equipe;
- ☐ os arquivos estáticos e as Functions compartilham a mesma origem;
- ☐ o projeto foi publicado por integração com GitHub;
- ☐ a equipe não instalou nem executou Node.js, `npm`, `npx` ou Wrangler;
- ☐ cada provedor usa uma URL de retorno própria e exata;
- ☐ os pedidos de autorização usam código e PKCE `S256`;
- ☐ a Function apresenta o Client Secret correto somente na troca de tokens;
- ☐ o retorno recusa uma transação ausente, expirada, alterada ou reutilizada;
- ☐ o id_token do Google só produz uma sessão depois da validação criptográfica e semântica;
- ☐ o access_token do GitHub é usado somente para consultar /user e a autorização é revogada antes da criação da sessão;
- ☐ o cookie de sessão é opaco, `Secure`, `HttpOnly`, `SameSite=Strict` e não possui `Domain`;
- ☐ o D1 guarda o resumo do cookie, não seu valor bruto;
- ☐ `/api/me` devolve somente o perfil necessário;
- ☐ o logout confere `Origin`, remove a sessão e expira o cookie;
- ☐ um cookie revogado não restaura a sessão;
- ☐ tokens e segredos não aparecem no HTML, nas URLs salvas, no armazenamento Web ou nos registros;
- ☐ a dupla consegue explicar por que os arquivos estáticos permanecem públicos;
- ☐ as sessões administrativas foram encerradas no computador compartilhado.

18. Diagnóstico rápido

Sintoma	Fronteira provável	Primeira conferência
`redirect_uri_mismatch`	Cadastro do provedor	URL `pages.dev` exata e plataforma Web
`invalid_client`	Pages Function e provedor	Client ID, Client Secret e expiração

Sintoma	Fronteira provável	Primeira conferência
`invalid_grant`	Transação PKCE	Código, verificador, expiração e URL de retorno
`state` inválido	Navegador e D1	Cookie temporário e transação correta
Emissor inválido no Google	Validação OIDC do Google	Documento de descoberta, `iss`, chave e relógio
Audiência inválida no Google	Validação OIDC do Google	Client ID do provedor correto
`/api/me` responde `401`	Sessão local	Cookie, resumo no D1 e expiração
`/api/health` responde `404`	Implantação do Pages	Pasta `functions` na raiz e implantação mais recente
`DB` é indefinido	Ligação do Pages	Nome `DB`, ambiente e nova implantação
A página abre sem login	Hospedagem estática	Comportamento esperado para arquivos públicos

Uma configuração recém-salva pode exigir uma nova implantação. Esperar não corrige uma URL de retorno diferente, um segredo expirado, uma audiência errada ou uma ligação com outro nome.

19. Encerramento e limpeza

Antes de deixar o laboratório:

1. encerre as sessões administrativas no Google, no GitHub e na Cloudflare;
2. feche a janela privativa;
3. remova das evidências qualquer cookie, código, token, segredo, `state`, `nonce` ou `code_challenge`;
4. confirme que nenhum segredo aparece no histórico do GitHub;
5. confirme que os Client Secrets continuam marcados como criptografados no Pages;
6. registre quem ficará responsável pela rotação dos Client Secrets;
7. entregue somente a pasta de evidências saneada.

Não apague o projeto nem o banco durante a aula. Quando a professora autorizar a limpeza definitiva, remova primeiro os registros dos provedores e depois os recursos de laboratório no painel da Cloudflare.

20. Referências

CLOUDFLARE. **Bindings**. Cloudflare Pages Docs, 2026. Disponível em: https://developers.cloudflare.com/pages/functions/bindings/. Acesso em: 13 set. 2026.

CLOUDFLARE. **Functions: get started**. Cloudflare Pages Docs, 2026. Disponível em: https://developers.cloudflare.com/pages/functions/get-started/. Acesso em: 13 set. 2026.

CLOUDFLARE. **Getting started**. Cloudflare D1 Docs, 2026. Disponível em: https://developers.cloudflare.com/d1/get-started/. Acesso em: 13 set. 2026.

CLOUDFLARE. **Git integration**. Cloudflare Pages Docs, 2026. Disponível em: https://developers.cloudflare.com/pages/get-started/git-integration/. Acesso em: 13 set. 2026.

CLOUDFLARE. **Pages Functions**. Cloudflare Pages Docs, 2026. Disponível em: https://developers.cloudflare.com/pages/functions/. Acesso em: 13 set. 2026.

GITHUB. Authorizing OAuth apps. GitHub Docs, 2026. Disponível em: https://docs.github.com/en/apps/oauth-apps/building-oauth-apps/authorizing-oauth-apps. Acesso em: 14 set. 2026.

GITHUB. Best practices for creating an OAuth app. GitHub Docs, 2026. Disponível em: https://docs.github.com/en/apps/oauth-apps/building-oauth-apps/best-practices-for-creating-an-oauth-app. Acesso em: 14 set. 2026.

GITHUB. Creating an OAuth app. GitHub Docs, 2026. Disponível em: https://docs.github.com/en/apps/oauth-apps/building-oauth-apps/creating-an-oauth-app. Acesso em: 14 set. 2026.

GITHUB. REST API endpoints for OAuth authorizations. GitHub Docs, 2026. Disponível em: https://docs.github.com/en/rest/apps/oauth-applications. Acesso em: 14 set. 2026.

GITHUB. REST API endpoints for users. GitHub Docs, 2026. Disponível em: https://docs.github.com/en/rest/users/users. Acesso em: 14 set. 2026.

GOOGLE. **OpenID Connect**. Google for Developers, 2026. Disponível em: https://developers.google.com/identity/openid-connect/openid-connect. Acesso em: 13 set. 2026.

IETF. **RFC 7636: Proof Key for Code Exchange by OAuth Public Clients**. 2015. Disponível em: https://www.rfc-editor.org/rfc/rfc7636. Acesso em: 13 set. 2026.

IETF. **RFC 9700: Best Current Practice for OAuth 2.0 Security**. 2025. Disponível em: https://www.rfc-editor.org/rfc/rfc9700. Acesso em: 13 set. 2026.

IETF. **RFC 10017: OAuth 2.0 for Browser-Based Applications**. 2026. Disponível em: https://www.rfc-editor.org/rfc/rfc10017. Acesso em: 13 set. 2026.

OPENID FOUNDATION. **OpenID Connect Core 1.0 incorporating errata set 2**. 2023. Disponível em: https://openid.net/specs/openid-connect-core-1_0.html. Acesso em: 13 set. 2026.