

1. Criação da estrutura de pastas padrão do projeto

- Separação clara de `pages`, `components`, `styles`, `routes` e `config`.

2. Instalação de dependências iniciais

- React, React Router, Styled Components, Axios, ESLint, Prettier, React Toastify, entre outros essenciais.

2. Configuração do ESLint

- Definidas regras de linting para manter a consistência do código.

3. Configuração do Prettier

- Criado o arquivo `.prettierrc` com as regras de formatação.
- Integrado com `ESLint` para correção automática ao salvar.

4. Criação de estilos globais

- Definidas globais e estilos base no arquivo `GlobalStyles`.
- Cores globais configuradas na pasta `config`.

5. Configuração das rotas

- Implementado `React Router DOM`.
- Estrutura inicial de rotas públicas e protegidas com `MyRoute`.

6. Proteção de rotas privadas

- Middleware de autenticação criado para impedir o acesso a rotas sem login.

COMMITTS E BRANCHS

Os fluxo de commits serão organizados da seguinte maneira:

- `main ou master`: branch de produção (versão estável que vai para o cliente final).
- `develop`: branch de desenvolvimento (onde integramos tudo que está pronto, mas ainda não foi para produção).

- Branches de features (`feature/nome-da-feature`): para desenvolver novas funcionalidades.
- Branches de bugs (`bugfix/nome-do-bug`): para corrigir problemas.
- Branches de release (`release/nome-da-versão`): preparação de uma nova versão para ser enviada para produção.

FLUXO DE TRABALHO

- O primeiro commit será feito na branch principal com as configurações iniciais, após isso os commits não serão feitos mais na `main`.

```
1 git init
2 git add .
3 git commit -m "chore: commit inicial com configurações do projeto"
4 git branch -M main
5 git push origin main
```

- Criação de `develop` (Branch central).

```
1 git checkout -b develop
2 git push origin develop
```

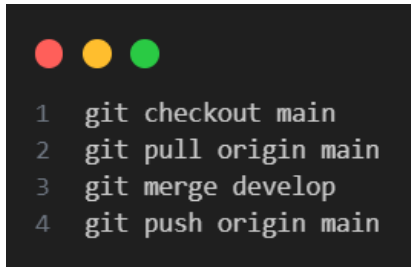
- Sempre comece criando uma branch a partir da `develop`.

```
1 git checkout develop
2 git pull origin develop
3 git checkout -b feature/nome-da-feature
```

- Desenvolva, finalize e abra um `Pull Request` sua branch de feature.

```
1 git add .
2 git commit -m "feat: implementa página de login"
3 git push origin feature/login
```

- Faça o merge na `develop`.



```

1  git checkout main
2  git pull origin main
3  git merge develop
4  git push origin main
  
```

BOAS PRÁTICAS

- Use nomes descritivos nas branches (ex: feature/cadastro-usuário).
- Escreva mensagens de commit padronizadas (ex: feat: adiciona formulário de cadastro).
- Nunca faça commits direto na main após o commit inicial.
- Atualize a develop regularmente com `git pull origin develop`.

Comando	Descrição	Quando Usar
<code>git branch</code>	Lista todas as branches locais	Verificar em quais branches você já trabalhou
<code>git branch <nome></code>	Cria uma nova branch local	Quando quiser iniciar uma nova funcionalidade
<code>git checkout <nome></code>	Troca para a branch especificada	Para mudar de contexto de trabalho
<code>git checkout -b <nome></code>	Cria e já troca para a nova branch	Ao iniciar uma nova branch de forma rápida
<code>git pull origin <branch></code>	Atualiza a branch local com o conteúdo remoto	Antes de criar uma nova branch ou começar a trabalhar
<code>git push origin <branch></code>	Envia a branch para o repositório remoto	Após criar/atualizar uma branch local
<code>git merge <branch></code>	Mescla a branch especificada na branch atual	Quando quiser unir o trabalho de outra branch
<code>git branch -d <branch></code>	Deleta uma branch local que já foi mesclada	Para limpar branches antigas
<code>git branch -D <branch></code>	Força a exclusão de uma branch local	Quando a branch não foi mesclada, mas você quer apagar
<code>git fetch</code>	Atualiza referências remotas sem fazer merge	Quando quer ver o que mudou no remoto sem afetar o local
<code>git pull</code>	Faz fetch + merge da branch atual	Para atualizar seu código com o servidor

git push --set-upstream origin <branch>	Associa a branch local com a remota	Quando cria uma nova branch e vai empurrar pela primeira vez
git switch <branch>	Alternativa moderna a checkout	Para mudar de branch de forma mais intuitiva
git switch -c <nome>	Cria e troca para a nova branch	Igual ao checkout -b, mas mais legível
git rebase <branch>	Reaplica os commits da branch atual sobre outra	Para manter um histórico linear (avançado)