Regras e Padrões de Uso do Git – Projeto Vought Tech

1. Objetivo

Este documento define as regras, padrões e boas práticas para uso do Git no projeto Vought Tech. O objetivo é facilitar o trabalho colaborativo entre os integrantes do grupo, reduzir conflitos de merge e retrabalho, manter o histórico de commits organizado e padronizar o fluxo de desenvolvimento.

Todas as regras aqui descritas devem ser seguidas por todos os membros da equipe durante o desenvolvimento do projeto.

2. Estrutura de Branches

2.1. Branch principal

- main:

Contém o código estável usado como base para as entregas; apenas código revisado e funcional deve ser mergeado na main.

- 2.2. Branches de desenvolvimento

  Para organizar o trabalho entre frontend, backend e documentação, o projeto utilizará as seguintes branches principais:

- main

  Branch principal do repositório. Armazena a versão estável do projeto e a documentação (ERS, padrões, guias, etc.). Durante o desenvolvimento, o código fonte é mantido nas branches específicas de frontend e backend. Ao final do projeto, será feito o merge das branches frontend e backend na main, integrando toda a solução.

- frontend

  Branch destinada ao desenvolvimento do frontend (React + Vite). Todas as alterações relacionadas à interface do usuário e lógica de front devem ser feitas nesta branch (ou em sub-branches que depois fazem merge em frontend).

- backend

Branch destinada ao desenvolvimento do backend (Node.js + Express + PostgreSQL). Todas as alterações relacionadas à API, regras de negócio e acesso a dados devem ser feitas nesta branch (ou em sub-branches que depois fazem merge em backend).

2.3. Regras gerais de branches

Toda task deve ser feita em branch própria, nunca diretamente na main.

3. Padrões de Commit

3.1. Formato da mensagem

Todos os commits devem seguir o padrão Conventional Commits, com o nome do integrante entre colchetes:

<tipo>: [Nome] descrição curta e objetiva

Exemplos:

- feat: [Marcos] adiciona autenticação de usuário
- fix: [Arthur] corrige cálculo do total do carrinho
- docs: [Diego] atualiza instruções de execução no README

3.2. Tipos de commit permitidos

- feat: nova funcionalidade;
- fix: correção de bug;
- docs: alterações na documentação;
- style: mudanças de formatação/estilo (não alteram lógica);
- refactor: refatoração sem mudar comportamento;
- test: inclusão ou modificação de testes;
- chore: tarefas de infraestrutura, dependências, configurações, etc.

3.3. Boas práticas de commit

- Commits devem ser pequenos e focados em uma única mudança.
- Evitar commits contendo várias alterações diferentes misturadas.
- Sempre rodar os testes relevantes (quando existirem) antes de commitar.

1. **4. Fluxo de Trabalho com Git**
2. O fluxo padrão para desenvolver qualquer tarefa (funcionalidade, correção ou ajuste) é o seguinte:
3. 1. Escolher a branch de trabalho:
4. - `frontend` para alterações de interface e lógica de front-end;
5. - `backend` para alterações de API, regra de negócio e acesso a dados.
6. 2. Atualizar a branch de trabalho local:
7. git checkout frontend
8. git pull origin frontend
9. 3. Implementar as alterações na branch escolhida, fazendo commits frequentes seguindo o padrão definido.

10. Verificar os arquivos alterados e preparar o commit:

11. git status

12. git add <arquivos-modificados>

13. git commit -m "tipo: [Nome] mensagem descritiva"

14. Enviar as alterações para o repositório remoto:

15. git push origin frontend ou backend

16. 6. A branch `main` não deve receber commits diretos durante o desenvolvimento.

17. Ao final do projeto, as branches `frontend` e `backend` serão integradas na `main`, consolidando a versão final estável do sistema.

5. Estrutura de Pastas e Documentação

5.1. Estrutura do repositório

A organização geral do projeto deve seguir:

- frontend/: código do frontend (React + Vite);
- backend/: código do backend (Node.js + Express + PostgreSQL);
- docs/: documentação (ERS, padrões, guias de execução, etc.).

5.2. Padrão dos documentos

- Todo documento deve conter: título claro, pequena descrição do objetivo, conteúdo organizado em seções/tópicos e, quando possível, ser disponibilizado em formato Markdown (.md) e/ou PDF.
- Diagramas devem ser exportados também em imagem (.png, .jpg ou .svg).

6. Arquivo .gitignore

Arquivos temporários, caches e pastas geradas automaticamente devem ser ignorados pelo Git.

- Exemplos:
    - node_modules/
    - .env
    - pastas de build e saída (dist/, build/, etc.).

O .gitignore deve ser mantido atualizado sempre que uma nova ferramenta ou framework gerar arquivos binários ou temporários.

7. Considerações Finais

Todas as regras deste documento valem para qualquer contribuição no repositório Vought Tech. Em caso de dúvida entre fazer mais rápido ou manter o padrão, deve-se priorizar

seguir o fluxo e os padrões definidos aqui. Qualquer alteração significativa nestas regras deve ser discutida pelo grupo e, após aprovada, atualizada neste documento.