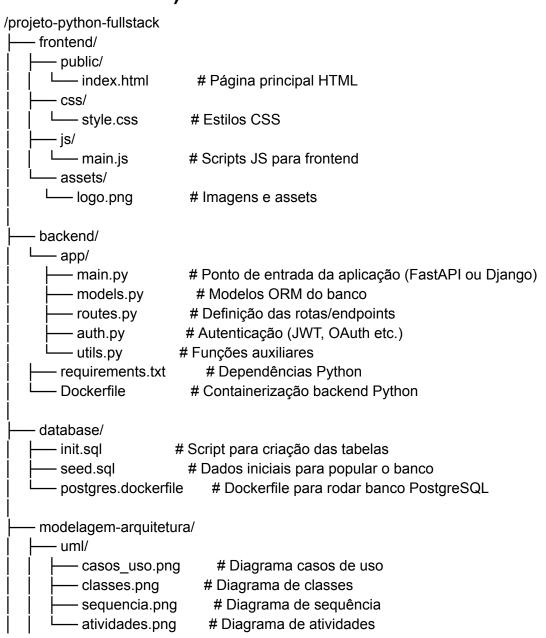
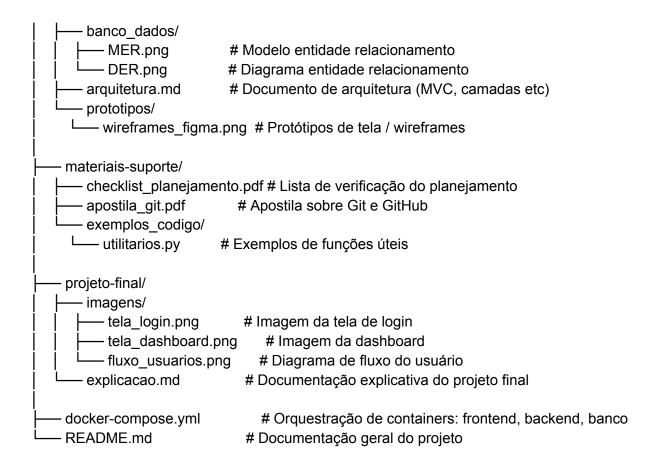
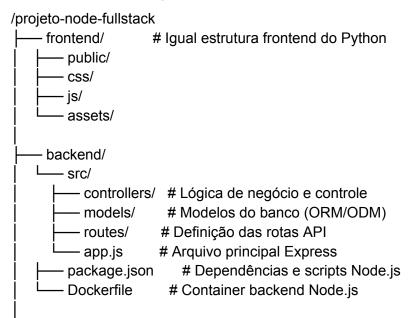
Estrutura e arquivos principais em cada projeto

1. Python Fullstack (FastAPI ou Django + PostgreSQL + HTML/CSS/JS)

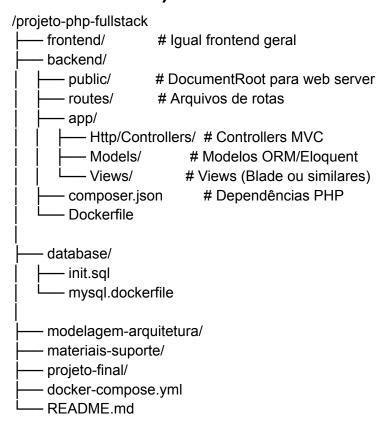




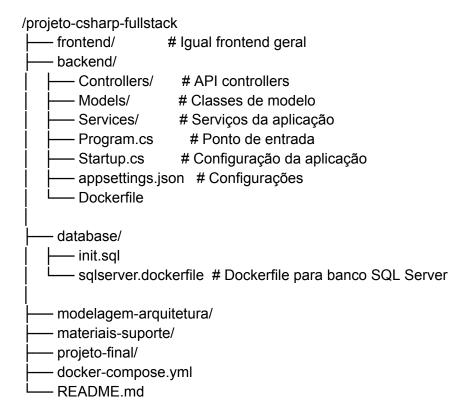
2. Node.js Fullstack (Express + MySQL + HTML/CSS/JS)



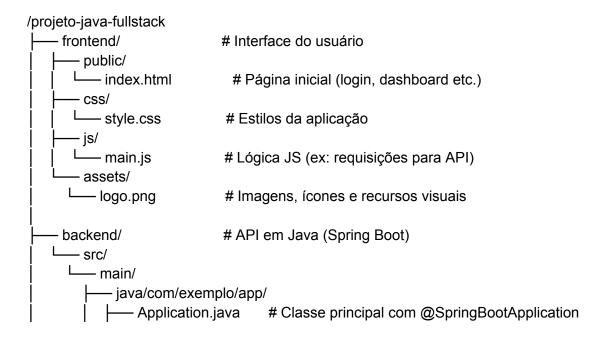
3. PHP Fullstack (Laravel ou estruturado + MySQL + HTML/CSS/JS)

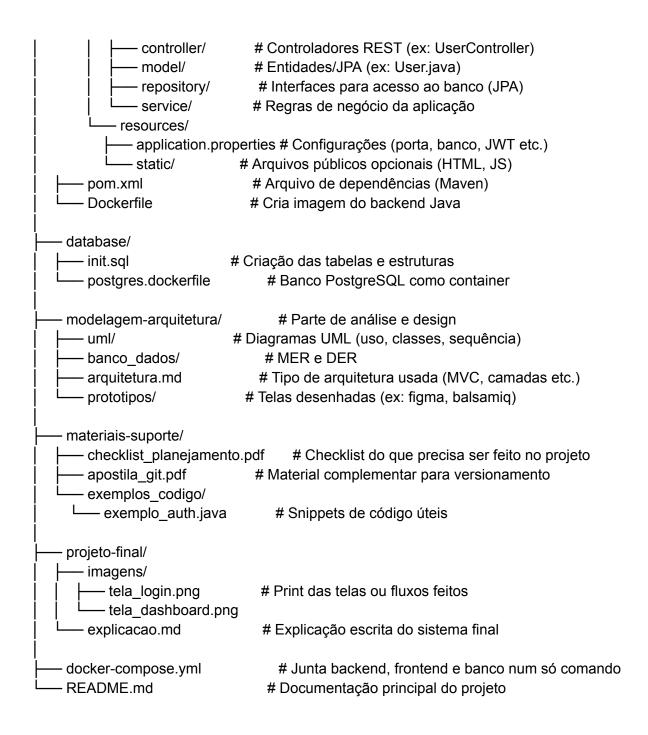


4. C# Fullstack (.NET Core API + SQL Server + HTML/CSS/JS)



5. Projeto Java Fullstack (Spring Boot + PostgreSQL + HTML/CSS/JS)





6. Projeto Kotlin Fullstack (Spring Boot ou Ktor + PostgreSQL + HTML/CSS/JS)

/projeto-kotlin-fullstack	
frontend/	# Interface do usuário
public/	

```
— index.html
                           # Estrutura básica da interface
    css/
   └── style.css
                          # Arquivo de estilos personalizados
   - js/
   └── main.js
                          # Scripts de frontend
   - assets/
  └── logo.png
                          # Imagens e arquivos estáticos
 backend/
                          # API em Kotlin (Spring Boot ou Ktor)
L— src/
  └── main/
        - kotlin/com/exemplo/app/

    Application.kt

                               # Classe principal da aplicação
            controller/
                              # Controllers HTTP
           - model/
                              # Classes de domínio
           repository/
                              # Interfaces de repositório com JPA
           – service/
                             # Camada de lógica e serviços
        - resources/
           - application.yml
                                # Configurações gerais do app
                           # Arquivos públicos opcionais
          – static/
    - build.gradle.kts
                              # Gerenciador de dependências Gradle Kotlin DSL
    Dockerfile
                             # Criação de container backend Kotlin
- database/
   — init.sql
                          # SQL inicial com tabelas e dados base

    postgres.dockerfile

                                # Configura container do PostgreSQL
- modelagem-arquitetura/
                                 # Planejamento visual e técnico
   - uml/
   - banco_dados/
   arquitetura.md
   - prototipos/
  wireframes_figma.png
 materiais-suporte/
   – checklist_planejamento.pdf
   apostila_git.pdf
   - exemplos codigo/
  ____ exemplo_servico.kt
 projeto-final/
   - imagens/
   - explicacao.md
```

Observações

- Todos os projetos têm o docker-compose.yml para facilitar o desenvolvimento integrado e deploy.
- Os arquivos .sql em database/ permitem criar e popular o banco, bem como executar migrações manuais.
- README.md é fundamental para documentar o funcionamento e como executar o projeto.
- As pastas de modelagem, protótipos, materiais e projeto final são para dar suporte didático e organizacional, mostrando o "porquê" de cada escolha.

🔽 1. README.md – Apresentação, execução e visão geral

Deve conter tudo o que alguém precisa saber para entender e rodar o sistema.

Exemplo real de conteúdo:

🟥 Sistema de Agendamento - Clínica Médica

Sistema Fullstack para gestão de agendamentos médicos, cadastro de pacientes, médicos, horários e autenticação de usuários com papel de administrador.

₱ Funcionalidades do Sistema

- Autenticação de usuários com JWT
- Cadastro de médicos e pacientes
- Agendamento de consultas
- Listagem por filtros de especialidade, data e disponibilidade
- Interface web responsiva
- Integração completa com API REST

Tecnologias

```bash

Clone o repositório git clone https://github.com/usuario/sistema-agendamento-clinica.git cd sistema-agendamento-clinica

Rode com Docker Compose docker-compose up --build

Frontend: http://localhost:3000 # Backend: http://localhost:8000/docs

Banco de dados: localhost:5432 (usuário: postgres / senha: postgres)

📂 Estrutura do Projeto

frontend/	# Interface
backend/	# API e lógica
— database/	# Scripts SQL e dockerfile
modelagem-arq	uitetura/ # Diagrama de fluxo, casos de uso, banco
materiais-supor	e/ # PDFs de apoio e exemplos
projeto-final/	# Prints e explicações
L— docker-compos	e.yml



- Admin: Cadastra médicos e usuários
- Recepção: Marca consultas
- Paciente: Acompanha seus agendamentos

Documentações Importantes

- modelagem-arquitetura/arquitetura.md
- <u>materiais-suporte/</u>

Desenvolvedor

Alan Souza – Desenvolvedor Fullstack

LinkedIn: <u>linkedin.com/in/alan</u>
GitHub: <u>github.com/alan</u>

2. `modelagem-arquitetura/` – Toda a estrutura lógica, visual e de dados

> Serve para quem precisa **entender como o sistema funciona por dentro**.

Subpastas e conteúdo ideal:

puml/

- `casos_de_uso.png` mostra quem interage com o sistema e o que pode fazer
- `sequencia_login.png` ilustra o passo a passo do login
- `diagrama_classes.png` mostra entidades e suas relações

banco_dados/

- `MER.png` Modelo Entidade Relacionamento (desenho do banco)
- `DER.png` Diagrama lógico com tipos dos campos e relacionamentos
- `explicacao.md` texto explicando:
- ```markdown

- Cada tabela
- Relacionamentos (1:1, 1:N, N:N)
- Chaves primárias e estrangeiras
- Regras de integridade

prototipos/

- tela_login.png Wireframe da tela de login (imagem ou link do Figma)
- tela_agendamento.png Protótipo da tela de agendamento
- fluxo_usuario.png Fluxo de uso do sistema

arquitetura.md

Arquitetura do Sistema

Estilo adotado: MVC + API REST

- **Model**: Representa as entidades (Usuário, Médico, Consulta)
- **View**: HTML/CSS/JS exibido no navegador
- **Controller**: Roteamento e chamadas de regras de negócio

Padrões utilizados:

- Separação de responsabilidades
- Validação de entrada no backend
- Token JWT para sessões
- Comunicação entre frontend e backend via fetch + JSON

Justificativa da Stack:

- Python FastAPI: leve, rápido e ótimo para ensino
- PostgreSQL: banco confiável e com bom suporte a relacionamentos

3. materiais-suporte/ - Auxiliares para alunos e devs



Checklist de Planejamento do Projeto

Objetivo definido

- Personas identificadas
- Requisitos funcionais levantados
- Protótipo desenhado
- Diagrama de casos de uso criado
- Arquitetura definida
- Manco de dados modelado
- Ambiente configurado (Docker)
- Git configurado
- README inicial feito

apostila_git.pdf ou .md

Comandos Git Essenciais

- `git init`
- `git add .`
- `git commit -m "mensagem"`
- 'git push origin main'
- `git pull`
- 'git checkout -b nome-da-branch'

Fluxo de Branches (Git Flow)

- main → produção
- dev → desenvolvimento geral
- feat/login → nova funcionalidade

exemplos_codigo/

- exemplo_auth.py código com JWT + FastAPI
- exemplo_fetch.js JS com fetch + tratamento de resposta
- exemplo_sql.sql exemplo de query com JOIN entre tabelas

Dica Final: Sempre responda no início do projeto:

"Se eu estivesse chegando hoje nesse projeto, o que eu precisaria ler primeiro para entender e começar a trabalhar?"

Se o seu README.md, a pasta de modelagem-arquitetura e os materiais-suporte responderem isso, seu projeto está bem documentado!

O que esse projeto contém:

Pasta/Arquivo	Conteúdo incluído
README.md	Visão geral do projeto, funcionalidades, tecnologias e instruções de execução
modelagem-arquit etura/	Diagramas UML, MER, DER, protótipos de telas e explicações técnicas
materiais-suport e/	Checklist completo, apostila de Git, exemplos de código (auth, fetch, SQL)
frontend/	Estrutura base com HTML/CSS/JS
backend/	API base com arquivos organizados (FastAPI como exemplo)
database/	Scripts SQL e Dockerfile para PostgreSQL
projeto-final/	Prints de tela reais e explicação do sistema para documentação final
<pre>docker-compose.y ml</pre>	Pronto para subir backend, frontend e banco

Agora você pode:

- Usar esse modelo como ponto de partida para qualquer sistema
- Garantir que todos seus projetos futuros sejam claros, completos e replicáveis