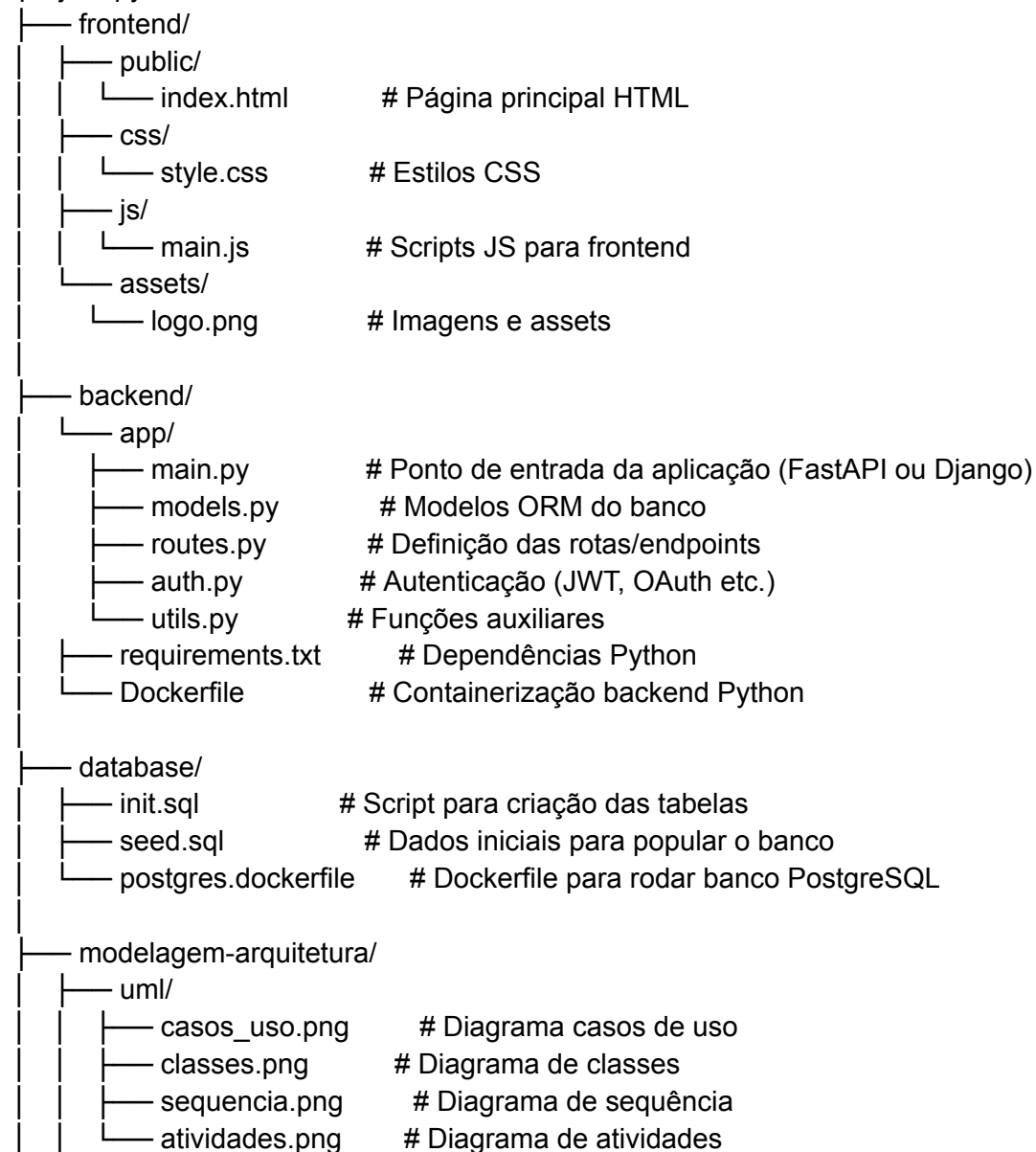


Estrutura e arquivos principais em cada projeto

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

/projeto-python-fullstack



```

├── banco_dados/
│   ├── MER.png          # Modelo entidade relacionamento
│   ├── DER.png          # Diagrama entidade relacionamento
│   ├── arquitetura.md    # Documento de arquitetura (MVC, camadas etc)
│   ├── prototipos/
│   │   └── wireframes_figma.png # Protótipos de tela / wireframes
├── materiais-suporte/
│   ├── checklist_planejamento.pdf # Lista de verificação do planejamento
│   ├── apostila_git.pdf          # Apostila sobre Git e GitHub
│   ├── exemplos_codigo/
│   │   └── utilitarios.py        # Exemplos de funções úteis
├── projeto-final/
│   ├── imagens/
│   │   ├── tela_login.png      # Imagem da tela de login
│   │   ├── tela_dashboard.png  # Imagem da dashboard
│   │   └── fluxo_usuarios.png   # Diagrama de fluxo do usuário
│   └── explicacao.md           # Documentação explicativa do projeto final
├── docker-compose.yml          # Orquestração de containers: frontend, backend, banco
└── README.md                   # Documentação geral do projeto

```

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

/projeto-node-fullstack

```

├── frontend/          # Igual estrutura frontend do Python
│   ├── public/
│   ├── css/
│   ├── js/
│   └── assets/
├── backend/
│   └── src/
│       ├── controllers/ # Lógica de negócio e controle
│       ├── models/      # Modelos do banco (ORM/ODM)
│       ├── routes/      # Definição das rotas API
│       └── app.js        # Arquivo principal Express
├── package.json        # Dependências e scripts Node.js
└── Dockerfile          # Container backend Node.js

```

```
├── database/
│   ├── init.sql      # Script criação banco (MySQL ou PostgreSQL)
│   ├── seed.sql      # Script para popular banco
│   └── mysql.dockerfile # Dockerfile para banco MySQL
├── modelagem-arquitetura/ # Mesma organização com diagramas e protótipos
├── materiais-suporte/
├── projeto-final/
├── docker-compose.yml
└── README.md
```

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

```
/projeto-php-fullstack
├── frontend/      # Igual frontend geral
├── backend/
│   ├── public/    # DocumentRoot para web server
│   ├── routes/    # Arquivos de rotas
│   ├── app/
│   │   ├── Http/Controllers/ # Controllers MVC
│   │   ├── Models/          # Modelos ORM/Eloquent
│   │   └── Views/           # Views (Blade ou similares)
│   ├── composer.json      # Dependências PHP
│   └── Dockerfile
├── database/
│   ├── init.sql
│   └── mysql.dockerfile
├── modelagem-arquitetura/
├── materiais-suporte/
├── projeto-final/
├── docker-compose.yml
└── README.md
```

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

```

/projeto-csharp-fullstack
├── frontend/           # Igual frontend geral
├── backend/
│   ├── Controllers/    # API controllers
│   ├── Models/         # Classes de modelo
│   ├── Services/       # Serviços da aplicação
│   ├── Program.cs      # Ponto de entrada
│   ├── Startup.cs      # Configuração da aplicação
│   ├── appsettings.json # Configurações
│   └── Dockerfile
├── database/
│   ├── init.sql
│   └── sqlserver.dockerfile # Dockerfile para banco SQL Server
├── modelagem-arquitetura/
├── materiais-suporte/
├── projeto-final/
├── docker-compose.yml
└── README.md

```

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

```

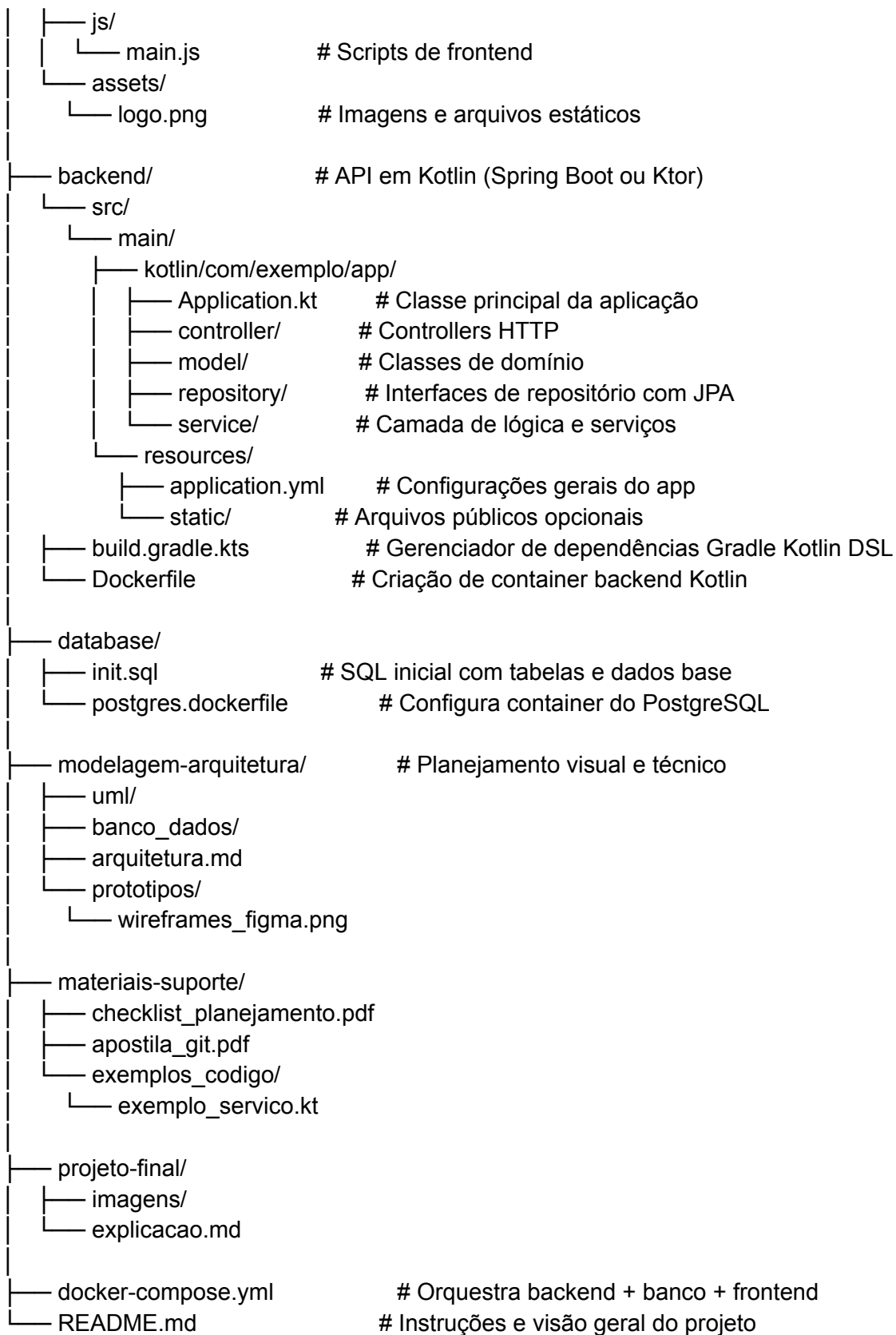
/projeto-java-fullstack
├── frontend/           # Interface do usuário
│   ├── public/
│   │   └── index.html    # Página inicial (login, dashboard etc.)
│   ├── css/
│   │   └── style.css     # Estilos da aplicação
│   ├── js/
│   │   └── main.js       # Lógica JS (ex: requisições para API)
│   └── assets/
│       └── logo.png      # Imagens, ícones e recursos visuais
├── backend/           # API em Java (Spring Boot)
│   └── src/
│       └── main/
│           ├── java/com/exemplo/app/
│           │   ├── Application.java # Classe principal com @SpringBootApplication
│           │   └── controller/      # Controladores REST (ex: UserController)

```

<ul style="list-style-type: none"> model/ repository/ service/ resources/ <ul style="list-style-type: none"> application.properties static/ pom.xml Dockerfile 	<ul style="list-style-type: none"> # Entidades/JPA (ex: User.java) # Interfaces para acesso ao banco (JPA) # Regras de negócio da aplicação # Configurações (porta, banco, JWT etc.) # Arquivos públicos opcionais (HTML, JS) # Arquivo de dependências (Maven) # Cria imagem do backend Java
<ul style="list-style-type: none"> database/ <ul style="list-style-type: none"> init.sql postgres.dockerfile 	<ul style="list-style-type: none"> # Criação das tabelas e estruturas # Banco PostgreSQL como container
<ul style="list-style-type: none"> modelagem-arquitetura/ <ul style="list-style-type: none"> uml/ banco_dados/ arquitetura.md prototipos/ 	<ul style="list-style-type: none"> # Parte de análise e design # Diagramas UML (uso, classes, sequência) # MER e DER # Tipo de arquitetura usada (MVC, camadas etc.) # Telas desenhadas (ex: figma, balsamiq)
<ul style="list-style-type: none"> materiais-suporte/ <ul style="list-style-type: none"> checklist_planejamento.pdf apostila_git.pdf exemplos_codigo/ <ul style="list-style-type: none"> exemplo_auth.java 	<ul style="list-style-type: none"> # Checklist do que precisa ser feito no projeto # Material complementar para versionamento # Snippets de código úteis
<ul style="list-style-type: none"> projeto-final/ <ul style="list-style-type: none"> imagens/ <ul style="list-style-type: none"> tela_login.png tela_dashboard.png explicacao.md 	<ul style="list-style-type: none"> # Print das telas ou fluxos feitos # Explicação escrita do sistema final
<ul style="list-style-type: none"> docker-compose.yml README.md 	<ul style="list-style-type: none"> # Junta backend, frontend e banco num só comando # Documentação principal do projeto

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

/projeto-kotlin-fullstack	
<ul style="list-style-type: none"> frontend/ <ul style="list-style-type: none"> public/ <ul style="list-style-type: none"> index.html css/ <ul style="list-style-type: none"> style.css 	<ul style="list-style-type: none"> # Interface do usuário # Estrutura básica da interface # Arquivo de estilos personalizados



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

Camada	Tecnologia	
-----	-----	
Frontend	HTML5, CSS3, JavaScript puro (ou React)	

Backend	Python (FastAPI)	
Banco de Dados	PostgreSQL	
DevOps	Docker + Docker Compose	
Versionamento	Git + GitHub	

Como Executar Localmente

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-arquitetura/ # Diagrama de fluxo, casos de uso, banco
├── materiais-suporte/  # PDFs de apoio e exemplos
├── projeto-final/      # Prints e explicações
└── docker-compose.yml
```

Perfis de Usuário

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

Documentações Importantes

- [modelagem-arquitetura/arquitetura.md](#)
- [materiais-suporte/](#)

Desenvolvedor

Alan Souza – Desenvolvedor Fullstack

LinkedIn: [linkedin.com/in/alan](https://www.linkedin.com/in/alan)

GitHub: github.com/alan

✓ 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:

uml/

- `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_planejamento.pdf` (ou .md para facilitar)

#### Checklist de Planejamento do Projeto

-  Objetivo definido
-  Personas identificadas
-  Requisitos funcionais levantados
-  Protótipo desenhado
-  Diagrama de casos de uso criado
-  Arquitetura definida
-  Banco 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**

|                                     |                                                                               |
|-------------------------------------|-------------------------------------------------------------------------------|
| <code>README.md</code>              | Visão geral do projeto, funcionalidades, tecnologias e instruções de execução |
| <code>modelagem-arquitetura/</code> | Diagramas UML, MER, DER, protótipos de telas e explicações técnicas           |
| <code>materiais-suporte/</code>     | Checklist completo, apostila de Git, exemplos de código (auth, fetch, SQL)    |
| <code>frontend/</code>              | Estrutura base com HTML/CSS/JS                                                |
| <code>backend/</code>               | API base com arquivos organizados (FastAPI como exemplo)                      |
| <code>database/</code>              | Scripts SQL e Dockerfile para PostgreSQL                                      |
| <code>projeto-final/</code>         | Prints de tela reais e explicação do sistema para documentação final          |
| <code>docker-compose.yml</code>     | 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**