Estrutura e arquivos principais em cada projeto

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

```
/projeto-python-fullstack
├── frontend/
│   ├── public/
│   │   └── index.html          # Página principal HTML
│   ├── css/
│   │   └── style.css           # Estilos CSS
│   ├── js/
│   │   └── main.js             # Scripts JS para frontend
│   └── assets/
│       └── logo.png            # Imagens e assets
│
├── backend/
│   └── app/
│       ├── main.py             # Ponto de entrada da aplicação (FastAPI ou Django)
│       ├── models.py           # Modelos ORM do banco
│       ├── routes.py           # Definição das rotas/endpoints
│       ├── auth.py             # Autenticação (JWT, OAuth etc.)
│       └── utils.py            # Funções auxiliares
│   ├── requirements.txt        # Dependências Python
│   └── Dockerfile              # Containerização backend Python
│
├── database/
│   ├── init.sql                # Script para criação das tabelas
│   ├── seed.sql                # Dados iniciais para popular o banco
│   └── postgres.dockerfile     # Dockerfile para rodar banco PostgreSQL
│
├── modelagem-arquitetura/
│   ├── uml/
│   │   ├── casos_uso.png       # Diagrama casos de uso
│   │   ├── classes.png         # Diagrama de classes
│   │   ├── sequencia.png       # Diagrama de sequência
│   │   └── atividades.png      # Diagrama de atividades
```

```
│   ├── 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)
│       │   ├── model/               # Entidades/JPA (ex: User.java)
│       │   ├── repository/          # Interfaces para acesso ao banco (JPA)
│       │   └── service/             # Regras de negócio da aplicação
│       └── resources/
│           ├── application.properties # Configurações (porta, banco, JWT etc.)
│           └── static/              # Arquivos públicos opcionais (HTML, JS)
│   ├── pom.xml                      # Arquivo de dependências (Maven)
│   └── Dockerfile                   # Cria imagem do backend Java
│
├── database/
│   ├── init.sql                     # Criação das tabelas e estruturas
│   └── postgres.dockerfile          # Banco PostgreSQL como container
│
├── modelagem-arquitetura/           # Parte de análise e design
│   ├── uml/                         # Diagramas UML (uso, classes, sequência)
│   ├── banco_dados/                 # MER e DER
│   ├── arquitetura.md               # Tipo de arquitetura usada (MVC, camadas etc.)
│   └── prototipos/                  # Telas desenhadas (ex: figma, balsamiq)
│
├── materiais-suporte/
│   ├── checklist_planejamento.pdf   # Checklist do que precisa ser feito no projeto
│   ├── apostila_git.pdf             # Material complementar para versionamento
│   └── exemplos_codigo/
│       └── exemplo_auth.java        # Snippets de código úteis
│
├── projeto-final/
│   ├── imagens/
│   │   ├── tela_login.png           # Print das telas ou fluxos feitos
│   │   └── tela_dashboard.png
│   └── explicacao.md                # Explicação escrita do sistema final
│
├── docker-compose.yml               # Junta backend, frontend e banco num só comando
└── README.md                        # Documentação principal do projeto
```

🟪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)
│   └── 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
│               └── static/               # Arquivos públicos opcionais
│   ├── 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
│
```

```
├── docker-compose.yml          # Orquestra backend + banco + frontend
└── README.md                   # Instruções e visão geral do projeto
```

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
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
`README.md`	Visão geral do projeto, funcionalidades, tecnologias e instruções de execução
`modelagem-arquitetura/`	Diagramas UML, MER, DER, protótipos de telas e explicações técnicas
`materiais-suporte/`	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
`docker-compose.yml`	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**