| **阶段** | **任务内容** | **AI 能力** | **你需要主导的关键点** |
| --- | --- | --- | --- |
| **1. 需求分析** | **确定平台功能、角色、业务流程、端到端用户体验** | **可协助流程图、用例图、竞品分析** | **明确业务目标、做功能取舍、优先级规划** |
| **2. 架构设计** | **模块划分、数据库结构、服务拆分、接口规划** | **能自动生成 ER 图、数据库建模、接口契约** | **拆微服务边界、做技术选型、考虑长期扩展性** |
| **3. 接口设计** | **OpenAPI、GraphQL schema、Mock** | **AI 可自动生成接口文档、测试数据** | **定义接口规范、字段含义、版本控制** |
| **4. 后端开发** | **商品、订单、支付、物流等核心业务逻辑** | **能生成 CRUD、支付/回调示例、REST API** | **处理复杂流程逻辑、幂等、事务、一致性、安全控制** |
| **5. 前端开发** | **Web、App、小程序界面、状态管理** | **可输出组件、请求封装、页面模板** | **做 UI 调优、交互逻辑、性能优化、适配不同平台** |
| **6. 测试 & QA** | **单元测试、集成测试、接口测试、性能测试** | **可生成 Jest/Cypress/E2E 脚本** | **补充边界用例、解 Bug、调试多端一致性** |
| **7. DevOps** | **CI/CD、打包部署、日志监控、版本发布** | **可写 Dockerfile、GitHub Actions 配置** | **配置实际部署环境、安全策略、排错手段** |
| **8. 上线运维** | **上架、运营支持、用户反馈、更新迭代** | **能自动归纳用户反馈、辅助写补丁脚本** | **处理线上故障、收集日志、规划迭代节奏** |

**帮我把每一个微服务以及web前端的所有流程告诉我，以及所需的所有CLI和SDK，以及统一用网关接口？  
每一步都需要一个详细的说明，尤其是API合同，比如最后所有微服务如果做一个网关？所以我想做一个电商的流程大概是这样，后端核心就是API wagger/OpenAPI，商品、订单、支付、物流、用户中心、鉴权、安全、运营分析和网关，和前端（暂时是WEB）对接就是单一 API、统一 OAuth2，其中后端和API完全解耦。**

**/contracts # OpenAPI YAML（唯一耦合点）需要知道后端各微服务的接口规范**

**/src**

**/gateway # 统一入口**

**/catalog # 商品**

**/order # 订单**

**/payment # 支付**

**/shipping # 物流**

**/user # 用户中心**

**/auth # OAuth2/OIDC**

**/security # 风控**

**/analytics # 运营分析**

**/web # Web 前端**

**1.4 小结：对接方“最低需要”哪些文档**

**对接支付微服务，最“核心”的文档就是：**

1. **API 合同（OpenAPI/Swagger）：接口路径、参数格式、返回值、错误码，至少能在线读到或下载一个 YAML/JSON。**
2. **认证鉴权文档：如何拿沙盒或生产环境的 Token/API Key，以及应该怎么在请求里带上。**
3. **测试环境信息：测试网关的 Base URL、测试商户号/测试密钥、环境切换方式（比如把 sandbox = true 放在环境变量里）。**
4. **示例代码（可选却高效）：如果官方做了示例，拿来跑一遍，省去自己手写拼接、验签的时间。**

**其他诸如数据库设计文档、健康检查说明、运维流程、监控告警配置、备份与恢复策略等，对于“单纯的 API 对接”来说并非立即必须。但如果未来你要“二次开发”或“自己也要出一个微服务”时，就需要上面那些更详细的文档了。**

**├── contracts/ # 存放所有 OpenAPI / gRPC IDL（集中管理，仍按服务子目录划分）**

**├── gateway/ # 统一入口（YARP）——独立 Git 仓库**

**├── frontend-web/ # React Web 前端——独立 Git 仓库**

**├── services/ # 仅示意层级，实际每个服务独立仓库**

**│   ├── catalog-service/**

**│   ├── order-service/**

**│   ├── payment-service/**

**│   ├── shipping-service/**

**│   ├── user-service/**

**│   ├── auth-service/**

**│   ├── security-service/**

**│   └── analytics-service/**

**└── infra/ # Helm Chart、Terraform、GitHub Actions Workflow**

**微服务的对接方“最低需要”文档，以便于之后需要的一些关键信息**

**多个微服务由网关聚合成同一段路由，微服务之间互不干扰，网关在转发时识别不同协议。**

**数据库是单实例 PostgreSQL，**所有微服务共享同一实例，按 **Schema 隔离，**

|  |  |
| --- | --- |
| **未来迁移** | **按 Schema → 独立实例或云托管几乎零改动** |

**。**

**业务仍必须遵守“一服务一写库”原则 —— 只是在物理上共用同一 PostgreSQL；逻辑上依旧解耦。**

**每个服务独立契约，确保独立性，不会因为某一个微服务API契约改动，而导致其他的微服务也需要改动！**

Catalog商品，语言 / 框架用：.NET 8 WebAPI。核心依赖 & CLI：dotnet-ef, Swashbuckle, FluentValidation。契约格式（需要和其他的微服务不同就行，独立好维护，最后聚合到网关）：OpenAPI 3.1 YAML

Order订单，语言 / 框架用：.NET 8 WebAPI + Rebus。核心依赖 & CLI：rebus-cli, Quartz.NET。契约格式（需要和其他的微服务不同就行，独立好维护，最后聚合到网关）：OpenAPI 3.1 YAML

Payment支付系统，语言 / 框架用：Go 1.22 + Gin。核心依赖 & CLI：go-swagger, gorm, cobra。契约格式（需要和其他的微服务不同就行，独立好维护，最后聚合到网关）：**gRPC** + OpenAPI (gRPC‑Gateway)

Shipping物流，语言 / 框架用：.NET 8 WebAPI。核心依赖 & CLI：MassTransit, Hangfire。契约格式（需要和其他的微服务不同就行，独立好维护，最后聚合到网关）：OpenAPI 3.1 YAML

User用户中心，语言 / 框架用：.NET 8 WebAPI + Rebus。核心依赖 & CLI：rebus-cli, Quartz.NET。契约格式（需要和其他的微服务不同就行，独立好维护，最后聚合到网关）：OpenAPI 3.1 YAML

Security风控，语言 / 框架用：Spring Boot 3。核心依赖 & CLI：Spring Security, spring-shell。契约格式（需要和其他的微服务不同就行，独立好维护，最后聚合到网关）：REST Doc (OpenAPI)

Auth，语言 / 框架用：.NET 8 + Duende IdentityServer。核心依赖 & CLI：dotnet-idsvr CLI。契约格式（需要和其他的微服务不同就行，独立好维护，最后聚合到网关）：OpenAPI 3.1 YAML

Analytics运营分析，语言 / 框架用：Python 3.12 + FastAPI。核心依赖 & CLI：Poetry, Pandas。契约格式（需要和其他的微服务不同就行，独立好维护，最后聚合到网关）：OpenAPI 3.1 YAML

**Gateway (YARP) 设计**

**认证 & 授权流程：**

**前端 ➜ auth-service 走 OAuth2 PKCE；获取 access\_token + refresh\_token。**

**网关验证 access\_token，把 sub、role 注入 X-User-Id / X-Roles Header。**

**微服务仅做细粒度业务授权（如果有）**