# 1. Directory

```
modular_rag_system_full/
├── app/                          # 系统核心逻辑
│   ├── core/                     # 抽象接口 & LLM 路由
│   │   ├── dto.py                # 跨模块数据传输对象 (GraphBuildIn/Out, RetrievalIn/Out...)
│   │   ├── interfaces.py         # 四大模块抽象接口 (GraphConstruction, RetrievalAgent...)
│   │   ├── llm_router.py         # 统一 LLM 路由器 (支持 OpenAI/Ollama 等 provider)
│   │
│   ├── di/                       # 依赖注入
│   │   ├── factory.py            # 工厂: load_settings, build_providers, build_router, build_modules
│   │
│   ├── modules/                  # 模块实现
│   │   ├── graph_construction/
│   │   │   ├── flow.py           # GraphConstructionFlow (orchestration: LoadExternal→Normalize→Build→Summarize)
│   │   │   ├── impl_networkx.py # GraphConstruction 的一个实现 (用 networkx 存储图)
│   │   │   └── impl_neo4j.py     # 可选: Neo4j 存储实现 (预留)
│   │   │
│   │   ├── retrieval/
│   │   │   ├── flow.py           # RetrievalAgentFlow (QueryExpand→BM25/FAISS→GraphExpand→RankSelect)
│   │   │   └── impl_hybrid.py # 另一种实现: Hybrid 检索 (BM25+Dense)
│   │   │
│   │   ├── reasoning/
│   │   │   └── flow.py           # ReasoningAgentFlow (Planner→Synthesizer)
│   │   │
│   │   ├── verification/
│   │   │   └── flow.py           # VerifierAgentFlow (RuleCheck→LLMCheck→Aggregate)
│   │
│   ├── orchestrator/
│   │   ├── nodes.py              # workflow 各节点 (node_build_graph, node_retrieval...)
│   │   ├── state.py              # workflow 的状态定义
│   │   └── workflow.py           # build_workflow(), 系统顶层 orchestrator (LangGraph 实现)
│   │
│   ├── system.py                 # 系统统一入口 (init_system, answer_question)
│
├── config/
│   └── settings.yaml             # 系统配置 (providers, llm_policy, modules, retrieval, verification...)
```

```
│
├── data/                          # 数据存储
│   ├── hotpotqa/
│   │   └── hotpot_dev_distractor_v1.json    # HotpotQA 开发集
│   ├── graph/                     # 生成的图存储目录
│   └── logs/                      # 系统运行日志
│
├── my_code/
│   └── run_system.py              # 用户/开发者入口：单问题/批量测试
│
├── requirements.txt               # Python 依赖
```

---

# 2. 📖 各部分介绍

## 3. 🔷 app/core/

- **dto.py**
  定义所有模块之间交互的数据模型，基于 Pydantic，例如：
  - `GraphBuildIn` / `GraphBuildOut`
  - `RetrievalIn` / `RetrievalOut`
  - `ReasoningIn` / `ReasoningOut`
  - `VerifyIn` / `VerifyOut`
- **interfaces.py**
  定义四大模块的抽象接口（Protocol/ABC）：
- `class GraphConstruction(Protocol): def build(self, req: GraphBuildIn) -> GraphBuildOut: ...`
- `class RetrievalAgent(Protocol): def retrieve(self, req: RetrievalIn) -> RetrievalOut: ...`
- `class ReasoningAgent(Protocol): def reason(self, req: ReasoningIn) -> ReasoningOut: ...`
- `class VerifierAgent(Protocol): def verify(self, req: VerifyIn) -> VerifyOut: ...`
- **llm_router.py**
  实现一个统一的 LLMRouter，支持：
  - 模型选择策略（cost/latency/ctx）
  - provider 适配器（OpenAI, Ollama 等）
  - 回退策略 / A/B 测试

---

## 4. 🔷 app/di/factory.py

依赖注入工厂，负责：

- `load_settings()`：读取 `config/settings.yaml`
- `build_providers()`：构造 LLM provider 实例
- `build_router()`：装配 LLMRouter

- `build_modules()`：根据 `modules` 配置加载四大模块（优先调用 `from_settings`）

---

### 5. 🔷 `app/modules/`

每个大模块都有 **Flow（内部 Orchestration）** 和 **实现类（impl_xxx）**。

- **graph_construction/flow.py**
  内部子编排：`LoadExternal → Normalize → Build → Summarize`
- **retrieval/flow.py**
  内部子编排：`QueryExpand → RetrieveText → GraphExpand → RankSelect`
- **reasoning/flow.py**
  内部子编排：`Planner → Synthesizer`
- **verification/flow.py**
  内部子编排：`RuleCheck → LLMCheck → Aggregate`

底层实现（impl_networkx/impl_hybrid）可替换。

---

### 6. 🔷 `app/orchestrator/`

- **workflow.py**：构建顶层 Orchestration（LangGraph StateGraph），按顺序调用四大模块。
- **nodes.py**：定义各节点函数，例如 `node_build_graph`, `node_retrieval`, `node_reasoning`, `node_verification`。
- **state.py**：定义 workflow 的状态模型（WFState）。

---

### 7. 🔷 `app/system.py`

系统统一入口：

- `init_system()`：加载 settings → 构建 providers/router/modules/workflow
- `answer_question(question, mode="full")`：系统封装的 API，外部调用时只传问题，返回答案和日志。
- 内部自动生成 trace_id。

---

### 8. 🔷 `config/settings.yaml`

系统配置，包含：

- **providers**（OpenAI, Ollama 等）
- **llm_policy**（路由策略）
- **modules**（四大模块的实现类）
- **graph_construction**（数据集路径、底层图存储）
- **retrieval**（索引配置、top_k）
- **reasoning**（推理策略）
- **verification**（验证规则）

---

### 9. 🔷 `my_code/run_system.py`

开发者/用户入口：

- 参数：
  - `--dataset` 数据集路径
  - `--index` 起始样本位置
  - `--limit` 取多少条（1=单问题，N=多问题，-1=全集）
  - `--mode` graph_only/full
  - `--output` 输出路径（批量模式）
- 逻辑：
  - 加载数据集
  - 调用 `answer_question`
  - 打印单问题结果 / 批量保存结果

---

## 10. ✅ 总结

- **外部用户入口**：`my_code/run_system.py`（只管提问，不管配置）
- **系统统一入口**：`app/system.py`（内部自己加载 settings，生成 trace_id，封装 workflow 调用）
- **核心模块**：四大 Flow（GraphConstruction, Retrieval, Reasoning, Verification），内部用 LangGraph orchestration
- **配置驱动**：`config/settings.yaml`，一切模块/LLM 路由/数据源通过配置文件控制