**一、Flask（轻量微内核 + 插件扩展）**

* **架构风格**
  + 微内核（Microkernel）：核心极小，负责路由与请求分发。
  + 插件化扩展：通过 Blueprint 挂载模块（路由、过滤器、静态资源、模板）。
  + 适配器：WSGI 适配 Web 服务器；Jinja2 模板作为视图层。
* **关键组件**
  + Flask（核心应用对象）、Blueprint（模块/插件）、Request/Response、Jinja2Environment、Werkzeug 路由/WSGI。
* **核心设计模式**
  + Microkernel、Decorator（@app.route）、Factory（create\_app）、Adapter（WSGI/模板）、Registry（路由表）。
* **典型流程**
  + Request → 路由匹配（核心）→ 调用蓝图注册的视图函数 → 渲染模板/返回 Response。
* **适合 Agent 测试点**
  + 路由冲突/覆盖、模板变量未定义、扩展版本不兼容、WSGI 中间件顺序问题、安全配置（Debug/Secret/CSRF）。

**二、Django（全栈 MVT + 中间件责任链）**

* **架构风格**
  + MVT：Model（ORM）- View - Template；与 MVC 同宗。
  + Pipe & Filter：Middleware 责任链处理请求/响应。
  + 插件化：App/Signals 可扩展。
* **关键组件**
  + HttpRequest/HttpResponse、MiddlewareMixin、URLResolver、View（CBV/FBV）、Model/Manager/QuerySet、Migrations。
* **核心设计模式**
  + MVC/MVT、Chain of Responsibility（中间件）、Active Record + Query Builder（ORM）、Template Method（CBV 钩子）、Inversion of Control（App/Signals）。
* **典型流程**
  + Request → Middleware 链 → URLResolver → View.dispatch → ORM/Template → Response → Middleware 链。
* **适合 Agent 测试点**
  + 路由优先级/冲突、ORM 生成 SQL 异常、模板上下文/变量错误、中间件顺序/安全配置（CSRF、DEBUG）、迁移一致性。

**三、Pandas（分层 + 数据内核与可插拔类型）**

* **架构风格**
  + 分层：API 层（DataFrame/Series）→ 逻辑层 → 存储层（Block/ArrayManager）→ 基础（NumPy/Cython）。
  + 微内核：NDFrame 为共同基类；类型扩展通过 ExtensionArray 插件化。
* **关键组件**
  + NDFrame/Series/DataFrame、Index、BlockManager/Block（或 ArrayManager）、ExtensionArray、I/O 模块。
* **核心设计模式**
  + Facade（DataFrame 封装底层存储）、Strategy（合并/聚合策略）、Template Method（groupby/rolling 骨架）、Adapter（多种 I/O 格式）、Flyweight（Index 共享/不可变）。
* **典型流程**
  + 高层 API（如 df.merge）→ 内部选择策略 → 访问底层块/数组并执行变换 → 返回新 NDFrame。
* **适合 Agent 测试点**
  + dtype/对齐问题、索引操作、时间序列边界、I/O 转换一致性、性能退化。

**对 CodeAgent 的借鉴：**

* 参考 Flask 微内核：把 Detector/Analyzer 做成插件；协调器为核心。
* 参考 Django 责任链：任务的校验/调度/汇总做中间件式管道。
* 参考 Pandas Facade：对外提供统一扫描接口，内部封装不同工具链。

**自有项目“拼图”缺陷清单**

* 配置与调试
* socketio.run(..., debug=True)、cors\_allowed\_origins="\*"（风险：生产暴露调试/宽 CORS）
* 建议：按环境开关、收紧 CORS 白名单
* 数据库一致性
* ORM 与 SQL 不一致（如 username 长度 50 vs 16；rank\_records vs records）
* 建议：以迁移为准统一 Schema，补非空/索引
* 资源下载健壮性
* requests.get() 无 timeout/校验和/重试（download\_fonts.py）
* 建议：超时/sha256 校验/有限重试
* 命名与规范
* ORM 类小写（class users），模板名疑似拼写错误（edditor.html）
* 建议：PEP8 类名、统一格式化、修正模板名
* 参数/安全
* 直接 request.args.get，缺少类型与范围校验；未见 CSRF 保护
* 建议：引入校验（pydantic/flask-wtf），开启 CSRF

**用 CodeAgent 查出缺陷方法**

* 工具集
* 质量/规范：pylint、flake8
* 安全：bandit（debug、subprocess、硬编码等）
* 语义：semgrep（Flask 规则：CSRF、XSS、CORS 任意、debug）
* （可选）类型：mypy
* 自定义检测
* Schema 漂移：比对 extensions.py vs table.sql（字段名/类型/长度/表名）
* 网络健壮性：requests 无 timeout/校验和
* CORS/Debug：扫描不安全配置

**架构与模式分析**

**架构风格总览**

* 分层与模块化：
  + 表示层：templates/（HTML）、static/（静态资源）
  + 接口层：views/\*.py（按功能拆分 Blueprint 路由）
  + 应用层：app.py（应用组装、注册蓝图与 SocketIO）
  + 领域/数据层：extensions.py（SQLAlchemy ORM 模型）
  + 配置层：config.py（环境变量加载）
* 运行时特性：
  + 实时通信：Flask-SocketIO（eventlet 异步）
  + 持久化：MySQL（SQLAlchemy ORM）
  + 配置：dotenv 注入环境变量

**关键组件与职责**

* app.py：应用工厂/组装（create\_app）、注册 Blueprint、启动 SocketIO。
* views/\*.py：路由与控制器逻辑（认证、图片、AI、PVP、排行、关卡、社交、分享）。
* extensions.py：ORM 模型（用户、拼图、成绩、好友、消息、进度）。
* templates/ 与 static/：前端页面与静态资源。
* config.py：配置中心（数据库、密钥、端口）。

**典型交互流程（时序简述）**

1. 浏览器请求 → app.py 路由分发到对应 views/\*.py 处理。
2. 业务逻辑读取/写入数据库（extensions.py ORM 模型）。
3. 返回模板渲染的页面或 JSON 响应；实时交互走 SocketIO 事件。