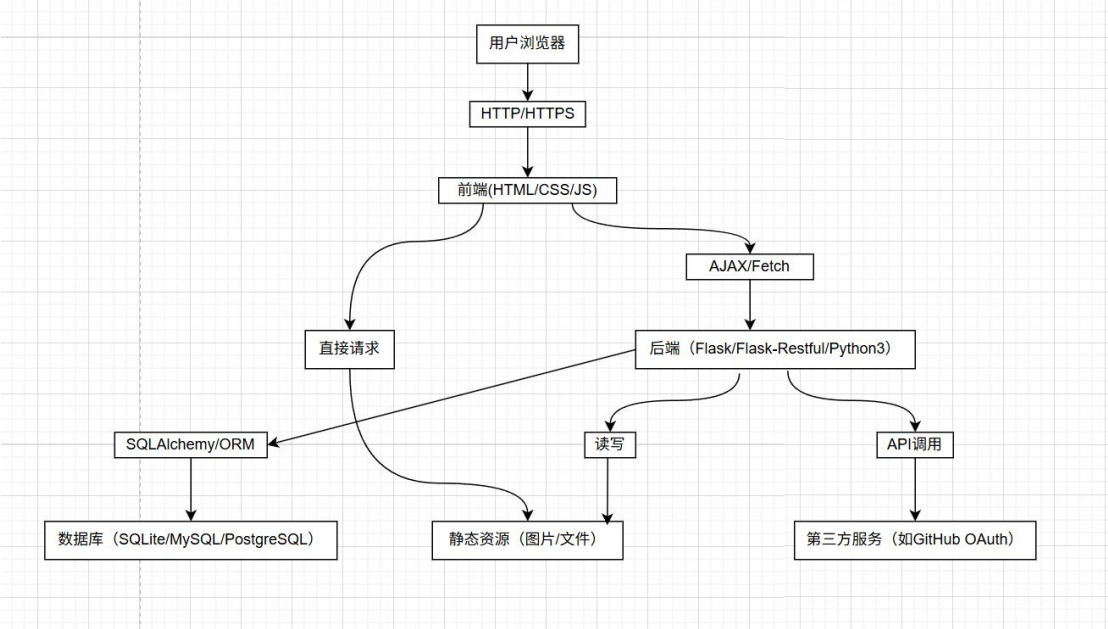
DuckStudy 系统架构设计文档

1. 项目概述

DuckStudy 是一个面向学习与交流的综合性平台，支持课程管理、帖子发布、二手市场、用户互动等功能。系统采用前后端分离架构，前端以静态资源为主，后端基于 Python Flask 提供 RESTful API 服务。

2. 整体架构

2.1 架构图



2.2 技术选型

前端：HTML5, CSS3, JavaScript (原生/部分框架), Bootstrap, Quill

后端：Python 3.x, Flask, Flask-Restful

数据库：SQLite（开发环境），可扩展为 MySQL/PostgreSQL(曾实施扩展 MySQL 计划,但由于工期问题中止）

依赖管理：pip, requirements.txt

测试：pytest, Jest（前端）

版本控制：Git

前端

2.2.1 HTML5

位置：/frontend/pages/*.html

说明：所有页面结构均采用 HTML5 语法编写，如 index.html、courses.html、login.html 等。

2.2.2 CSS3

位置：/frontend/css/*.css

说明：所有页面样式均采用 CSS3，分为全局样式（如 common.css、main.css）和页面/模

块样式（如 course-detail.css、market.css）。

2.2.3 JavaScript（原生/部分框架）

位置：/frontend/js/*.js

说明：页面交互、数据请求、DOM 操作等均由原生 JavaScript 实现。例如：

main.js：主页面逻辑

login.js、register.js：登录注册逻辑

api.js：封装与后端 API 的交互

market.js、posts.js 等：各业务模块逻辑

2.2.4 Bootstrap

位置：/frontend/lib/bootstrap/

说明：页面布局与部分组件样式采用 Bootstrap 框架，CSS 和 JS 文件分别在 css/ 和 js/ 目录下，通过 HTML <link> 和 <script> 标签引入。

2.2.5 Quill

位置：/frontend/lib/quill/

说明：富文本编辑器功能（如发帖、评论）使用 Quill，相关样式和 JS 在 quill/ 目录下，通过页面引入。

后端

2.2.6 Python 3.x

位置：/backend/ 下所有 .py 文件

说明：后端所有代码均采用 Python 3.x 编写。

2.2.7 Flask

位置：/backend/app.py（主应用入口）

说明：Flask 作为 Web 框架，负责路由分发、请求处理、服务启动。

2.2.8 Flask-Restful

位置：/backend/routes/*.py

说明：RESTful API 路由定义，采用 Flask-Restful 进行资源管理和接口设计，如 user_routes.py、upload_routes.py。

测试

2.2.9 pytest（后端）

位置：/tests/backend/*.py

说明：后端单元测试与集成测试脚本，采用 pytest 编写，如 test_user_auth.py、test_post.py。

2.2.10 Jest（前端）

位置：/tests/frontend/*.js

说明：前端单元测试脚本，采用 Jest 编写，如 Login.test.js、Main.test.js。

3. 模块划分

3.1 前端模块

页面：课程、帖子、市场、用户中心等（/frontend/pages/）

脚本：各页面对应 JS 逻辑（/frontend/js/）

样式：全局与页面级 CSS（/frontend/css/）

静态资源：图片、图标等（/frontend/images/）

3.2 后端模块

路由层（/backend/routes/）：定义 API 路由，如用户、上传、帖子等
服务层（/backend/services/）：业务逻辑封装，如 GitHub 服务
工具层（/backend/utils/）：缓存、通用工具
配置层（/backend/config/）：环境与参数配置
主应用（/backend/app.py）：Flask 应用入口

3.3 数据层

数据库模型（可扩展）
静态数据（/frontend/data/）：前端本地模拟数据，后续可由后端 API 替换

4. 关键设计决策

4.1 前后端分离

前端与后端通过 RESTful API 通信，解耦开发与部署。
静态资源可托管于 CDN，提升访问速度。

4.2 路由与服务分层

路由层仅负责请求分发与参数校验，业务逻辑下沉至服务层，便于单元测试与复用。

4.3 配置与环境隔离

配置文件独立（/backend/config/config.py），支持多环境切换（开发、测试、生产）。

4.4 安全性

用户认证采用 Token（如 JWT）或 Session 机制，接口需鉴权。
文件上传需校验类型与大小，防止恶意文件。
重要操作需防止 CSRF、XSS、SQL 注入等常见安全风险。

4.5 可扩展性

模块化设计，便于新增功能（如新业务模块、第三方集成）。
数据库 ORM 设计，便于迁移与扩展表结构。
前端组件化，便于复用与维护。

4.6 测试与质量保障

后端采用 pytest，前端采用 Jest，保证核心功能的单元与集成测试覆盖。
代码分层便于 Mock 与隔离测试。

5. 数据流与接口设计

5.1 典型数据流

用户登录：前端表单 → 后端 /api/login → 数据库校验 → 返回 Token
课程浏览：前端请求 /api/courses → 后端查询 → 返回课程列表 JSON
帖子发布：前端表单 → /api/posts（POST）→ 后端校验与存储 → 返回结果

5.2 API 设计原则

遵循 RESTful 规范，资源路径清晰，方法语义明确（GET/POST/PUT/DELETE）。
返回统一 JSON 格式，包含 code、msg、data 字段。
错误处理规范，前端可据此友好提示。

6. 部署与运维

前端可静态部署于 Nginx、GitHub Pages 等。
后端可容器化部署（Docker），支持 Gunicorn + Nginx 生产环境。
日志与监控建议集成（如 Sentry、Prometheus）。

7. 未来优化方向

支持多语言与国际化
前端引入现代框架（如 React/Vue）提升交互体验
后端引入异步框架（如 FastAPI）提升高并发能力
数据库分库分表、缓存优化（如 Redis）