**拼图大师游戏项目 设计文档（初稿）**

**作者**：第一组（邓清月、朱写、白钦宇、丁予晴、曹凯淇）  
**日期**：2025-09-05

1. 引言
   1. 编写目的

本设计文档依据《软件需求规格说明书》编写，目的是在明确需求的基础上，进行系统总体设计与详细设计，为开发与测试提供实现参考和技术路线。

* 1. 适用对象
* 前端、后端开发人员
* UI 设计人员
* 测试人员
* 项目负责人及指导教师

1. 术语表

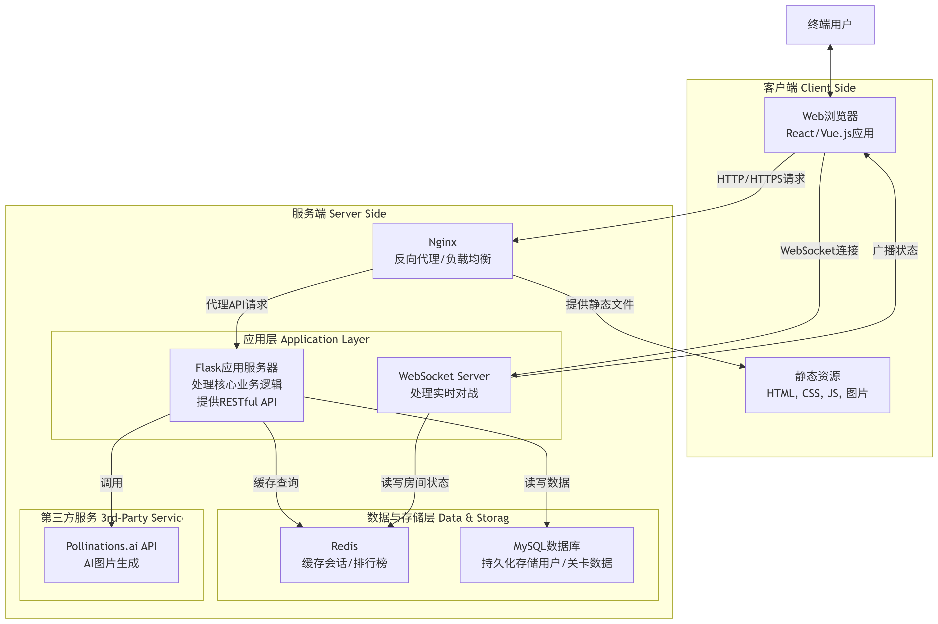
| 术语 / 缩写 | 全称 | 说明 |
| --- | --- | --- |
| API | Application Programming Interface | 应用程序接口，后端对外提供的功能调用方式，本项目采用 RESTful 风格。 |
| RESTful | Representational State Transfer | 一种设计 API 的风格，使用 HTTP 动词（GET、POST、PUT、DELETE 等）对资源进行操作。 |
| JWT | JSON Web Token | 身份验证令牌。登录成功后由后端签发，后续请求通过附带 JWT 来验证身份（通常放在 Authorization: Bearer 头）。 |
| RBAC | Role-Based Access Control | 基于角色的访问控制，用户的操作权限由其角色决定。 |
| bcrypt | —— | 一种加盐哈希算法，用于安全存储用户密码，比 MD5、SHA1 更安全。 |
| Nginx | —— | 高性能的 HTTP 和反向代理服务器，用于静态文件服务、反向代理与负载均衡。 |
| Flask | —— | 轻量级 Python Web 框架，本项目后端主要框架。 |
| Gunicorn | Green Unicorn | Python WSGI HTTP 服务器，用于运行 Flask 应用并处理并发请求。 |
| Redis | —— | 内存数据库/缓存，用于排行榜、热门分享、对战状态等高频访问数据。 |
| MySQL | —— | 关系型数据库系统，存储用户、拼图关卡、存档、分享等数据。 |
| ER 图 | Entity-Relationship Diagram | 实体-关系图，表示数据库中实体及其关系（本项目支持 Chen/IE 表示法）。 |
| 数据字典 | Data Dictionary | 描述数据库表字段的详细信息，包括字段含义、类型、约束条件等。 |
| WebSocket | —— | 客户端与服务器之间的全双工实时通信协议，本项目用于对战进度同步。 |
| AI 生成 | AI-based Image Generation | 借助第三方 AI 服务（如 Pollinations）生成像素风拼图图片。 |
| CI/CD | Continuous Integration / Continuous Deployment | 持续集成与持续部署，自动化测试与上线流程（本项目可使用 GitHub Actions 或 Jenkins）。 |
| 单元测试 | Unit Test | 针对最小功能单元（函数/类）的自动化测试。 |
| 集成测试 | Integration Test | 测试不同模块之间接口交互的正确性。 |
| 系统测试 | System Test | 从用户视角出发，对整个系统进行端到端测试。 |
| 性能测试 | Performance Test | 测试系统在高并发/压力环境下的表现，如响应时间、吞吐量。 |
| XSS | Cross-Site Scripting | 跨站脚本攻击，恶意脚本注入网页。 |
| CSRF | Cross-Site Request Forgery | 跨站请求伪造攻击，冒充用户发送请求。 |
| SQL 注入 | SQL Injection | 通过拼接恶意 SQL 语句来获取或修改数据库数据的攻击方式。 |

1. 总体设计
   1. 系统架构

系统采用 **前后端分离 + Web 部署** 的架构模式：

* **前端**：基于 React / Vue.js，采用像素风 UI，适配桌面端与移动端。
* **后端**：基于 Flask (Python) 框架，提供 RESTful API 接口，负责业务逻辑与数据存取。
* **数据库**：MySQL 8.0，存储用户信息、关卡数据、存档与分享内容；Redis 用于缓存排行榜与热门分享。
* **AI 接口**：调用第三方服务（如 pollinations.ai）生成像素风拼图图片。

架构示意：  
用户浏览器 ←→ Nginx ←→ Flask 后端 ←→ MySQL / Redis ←→ AI 接口



* 1. 模块划分

1. 用户认证模块
2. 单人游戏模块
3. 拼图编辑模块
4. 对战模块
5. 分享与排行榜模块
6. 存档管理模块
7. 游戏体验模块（音乐、操作指南）
8. 模块设计
   1. 用户认证模块

* **功能**：注册、登录、记住我、账号锁定。
* **实现**：
  + 使用 JWT（JSON Web Token）进行认证。
  + 用户密码采用 MD5 + 盐值存储。
  + 登录失败超过 3 次 → 锁定 10 分钟。
  1. 单人游戏模块
* **功能**：关卡选择、拼图操作、通关判定。
* **实现**：
  + 前端 Canvas 渲染拼图块，支持拖拽与旋转。
  + 误差检测（≤10px 吸附）。
  + 后端存储用户最佳成绩。
  1. 编辑模块
* **功能**：上传图片 / AI 生成 → 拼图配置 → 预览 → 保存 / 分享。
* **实现**：
  + 前端图像裁剪库（Cropper.js）。
  + 调用 AI 接口生成图片。
  + 拆分算法：根据块数和形状切割图片 → 生成拼图数据结构。
  1. 对战模块
* **功能**：房间创建、加入、进度同步、胜负判定。
* **实现**：
  + WebSocket 实时通信（进度同步 ≤500ms）。
  + 后端房间管理（Redis 记录对战状态）。
  1. 分享与排行榜模块
* **功能**：排行榜展示、拼图分享。
* **实现**：
  + 排行榜数据存入 Redis，前端缓存 30 分钟。
  + 分享关卡存入数据库，并展示缩略图、难度、作者。
  1. 存档管理模块
* **功能**：保存、读取、删除存档。
* **实现**：
  + 存档数据（拼图状态、步数、时间）以 JSON 格式存储在 MySQL。
  1. 游戏体验模块
* **功能**：背景音乐、音效、操作指南。
* **实现**：
  + 音频采用 HTML5 <audio> API。
  + 不同风格音乐文件存储在服务器，可动态切换。

1. 数据库设计
   1. 数据字典
      1. users（用户表）

| 字段名 | 类型 | 约束 | 说明 |
| --- | --- | --- | --- |
| user\_id | INT | PK，自增 | 用户唯一 ID |
| username | VARCHAR(16) | 唯一，非空 | 用户名（4–16 位） |
| email | VARCHAR(50) | 唯一，非空 | 用户邮箱 |
| password\_hash | CHAR(64) | 非空 | 加密存储的密码 |
| avatar\_url | VARCHAR(255) | 可空 | 用户头像 URL |
| created\_at | DATETIME | 默认当前时间 | 注册时间 |

* + 1. puzzles（拼图关卡表）

| 字段名 | 类型 | 约束 | 说明 |
| --- | --- | --- | --- |
| puzzle\_id | INT | PK，自增 | 关卡 ID |
| creator\_id | INT | FK → users.user\_id | 创建者 |
| title | VARCHAR(20) | 非空 | 关卡标题 |
| image\_url | VARCHAR(255) | 非空 | 拼图图片路径 |
| difficulty | ENUM('easy','medium','hard') | 非空 | 难度等级 |
| piece\_count | INT | 非空 | 拼块数量 |
| piece\_shape | ENUM('rect','irregular') | 非空 | 拼块形状 |
| is\_rotatable | BOOLEAN | 默认 false | 是否可旋转 |
| is\_system\_level | BOOLEAN | 默认 false | 是否系统关卡 |
| created\_at | DATETIME | 默认当前时间 | 创建时间 |

* + 1. records（游戏记录表）

| 字段名 | 类型 | 约束 | 说明 |
| --- | --- | --- | --- |
| record\_id | INT | PK，自增 | 记录 ID |
| user\_id | INT | FK → users.user\_id | 玩家 ID |
| puzzle\_id | INT | FK → puzzles.puzzle\_id | 关卡 ID |
| time\_used | INT | 非空 | 耗时（秒） |
| step\_count | INT | 非空 | 操作步数 |
| completed\_at | DATETIME | 非空 | 完成时间 |

* + 1. saves（存档表）

| 字段名 | 类型 | 约束 | 说明 |
| --- | --- | --- | --- |
| save\_id | INT | PK，自增 | 存档 ID |
| user\_id | INT | FK → users.user\_id | 玩家 ID |
| puzzle\_id | INT | FK → puzzles.puzzle\_id | 关卡 ID |
| save\_name | VARCHAR(20) | 非空 | 存档名称 |
| state\_data | JSON | 非空 | 存档状态 JSON |
| time\_used | INT | 非空 | 已耗时 |
| step\_count | INT | 非空 | 已用步数 |
| updated\_at | DATETIME | 默认当前时间 | 更新时间 |

* + 1. shares（分享表）

| 字段名 | 类型 | 约束 | 说明 |
| --- | --- | --- | --- |
| share\_id | INT | PK，自增 | 分享 ID |
| user\_id | INT | FK → users.user\_id | 分享者 |
| puzzle\_id | INT | FK → puzzles.puzzle\_id | 被分享关卡 |
| view\_count | INT | 默认 0 | 浏览量 |
| created\_at | DATETIME | 默认当前时间 | 分享时间 |

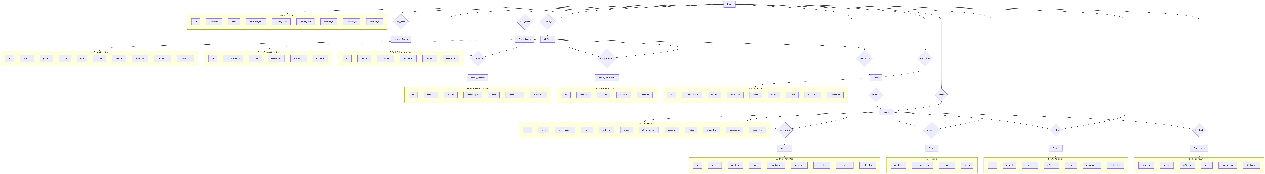
* + 1. battle\_rooms（对战房间表）

| 字段名 | 类型 | 约束 | 说明 |
| --- | --- | --- | --- |
| room\_id | INT | PK，自增 | 房间 ID |
| room\_code | VARCHAR(6) | 唯一 | 房间码 |
| creator\_id | INT | FK → users.user\_id | 房主 ID |
| puzzle\_id | INT | FK → puzzles.puzzle\_id | 使用的关卡 |
| status | ENUM('waiting','running','finished','closed') | 非空 | 房间状态 |
| winner\_id | INT | FK → users.user\_id，可空 | 胜者 ID |
| created\_at | DATETIME | 默认当前时间 | 创建时间 |

* + 1. battle\_users（对战参与者表）

| 字段名 | 类型 | 约束 | 说明 |
| --- | --- | --- | --- |
| id | INT | PK，自增 | 参与记录 ID |
| battle\_id | INT | FK → battle\_rooms.room\_id | 对应房间 |
| user\_id | INT | FK → users.user\_id | 玩家 ID |
| is\_ready | BOOLEAN | 默认 false | 是否准备 |
| progress | FLOAT | 默认 0.0 | 当前进度（0–1） |
| joined\_at | DATETIME | 默认当前时间 | 加入时间 |

* 1. ER关系图



1. 接口设计
   1. 用户认证

* POST /api/auth/register  
  输入：username, email, password  
  输出：{ code:200, msg:"success", token:"..." }
* POST /api/auth/login  
  输入：username/email, password  
  输出：{ code:200, msg:"success", token:"..." }
* POST /api/auth/logout  
  输入：token  
  输出：{ code:200, msg:"logout success" }
* GET /api/auth/me  
  输出：用户信息 JSON
  1. 用户资料
* GET /api/users/:user\_id → 获取用户公开信息
* PUT /api/users/me → 更新昵称/头像
* PUT /api/users/me/password → 修改密码
  1. 媒体上传与处理
* POST /api/media/upload → 上传图片（jpg/png ≤5MB），返回 url
* POST /api/media/crop → 图片裁剪，返回新 url
  1. AI 生成
* POST /api/ai/generate  
  输入：prompt（像素风提示词）  
  输出：task\_id
* GET /api/ai/generate/:task\_id → 返回生成状态 / 图片 url
  1. 拼图关卡
* POST /api/puzzles → 创建自定义关卡
* PUT /api/puzzles/:puzzle\_id → 更新关卡信息
* DELETE /api/puzzles/:puzzle\_id → 删除关卡
* GET /api/puzzles/:puzzle\_id → 获取关卡详情
* GET /api/puzzles → 获取关卡列表（支持筛选、分页）
* POST /api/puzzles/:puzzle\_id/compile → 生成拼图切片数据
  1. 游戏记录
* POST /api/record/save  
  输入：puzzle\_id, time\_cost, steps  
  输出：{ code:200, msg:"saved" }
* GET /api/record/rank/:puzzle\_id  
  输出：排行榜 JSON
* GET /api/record/me/best?puzzle\_id= → 我的最佳成绩
* GET /api/record/me/summary → 我的成绩统计
  1. 分享与排行榜
* GET /api/rank/global → 总排行榜（完成次数排序）
* GET /api/rank/puzzles/:puzzle\_id → 单关卡排行榜
* GET /api/share/feeds?type=hot|new → 分享中心热门/最新关卡
* GET /api/share/mine → 我的分享列表
* POST /api/puzzles/:puzzle\_id/share → 分享关卡
* DELETE /api/puzzles/:puzzle\_id/share → 取消分享
  1. 存档管理
* POST /api/saves → 保存存档
* GET /api/saves → 获取存档列表
* GET /api/saves/:save\_id → 获取存档详情
* DELETE /api/saves/:save\_id → 删除存档
* POST /api/saves/cleanup → 批量清理存档
  1. 对战
* POST /api/battle/create → 返回 room\_id
* POST /api/battle/join → 输入 room\_id
* POST /api/battle/leave → 退出房间
* POST /api/battle/ready → 准备
* POST /api/battle/unready → 取消准备
* GET /api/battle/rooms/:room\_id → 房间详情
* WebSocket /api/battle/sync
  + 事件：auth, ready, unready, progress, start, result, room\_update, heartbeat
  1. 游戏体验
* GET /api/ux/music → 获取音乐清单
* PUT /api/ux/music/preference → 设置音乐偏好
* GET /api/ux/help → 获取操作指南
  1. 系统与管理
* GET /api/system/health → 健康检查
* GET /api/system/version → 获取版本信息
* GET /api/admin/audit → 审计日志（管理员）
* POST /api/admin/users/:user\_id/ban → 封禁用户
* DELETE /api/admin/users/:user\_id/ban → 解封用户

1. 非功能性需求设计
   1. 性能需求

* 响应时间：普通页面加载时间不超过2秒，核心游戏界面首屏加载时间不超过3秒。用户关键操作（如图块拖拽放置、旋转）的响应延迟应低于100毫秒。实时对战模式下，状态同步延迟应低于500毫秒。
* 吞吐量与并发：系统应支持至少200名用户同时在线，其中允许最多50个对战房间同时进行。核心API接口的吞吐量应达到每分钟处理1000次以上请求。
* 资源利用率：在典型负载下，服务器CPU平均使用率应低于70%，内存使用率应低于80%。
  1. 安全性需求
* 数据安全：所有用户密码必须使用bcrypt算法（工作因子为12）进行加盐哈希处理，严禁明文存储。所有前后端通信必须使用HTTPS协议加密。
* 应用安全：后端接口必须实施有效的SQL注入防护（使用参数化查询）、XSS攻击防护（对用户输入进行转义或过滤）和CSRF攻击防护。
* 访问控制：严格实行基于角色的访问控制（RBAC）。用户只能访问和操作其自身拥有的数据（如存档、自定义关卡），除非该数据已被明确分享。
  1. 可靠性需求
* 可用性：系统核心功能（游戏、编辑、账号管理）的年可用性目标不低于99.5%。
* 容错性：系统应具备一定的容错能力。对于非关键功能的失败（如AI图片生成失败、第三方音乐加载失败），不应导致整个应用崩溃，而应向用户提供友好的错误提示。
* 数据持久性：用户数据（包括账号信息、关卡数据、存档）必须保证持久化存储，避免数据丢失。
  1. 兼容性需求
* 浏览器兼容：应确保在Chrome、Firefox、Safari、Edge等主流浏览器的最新两个稳定版本中功能正常。
* 设备兼容：前端界面应具备响应式设计，能够自适应不同尺寸的桌面端和移动端屏幕。
  1. 可维护性与可扩展性
* 代码应结构清晰、模块化程度高，并配有必要的文档和注释，以降低后续维护和功能扩展的难度。
* 数据库表结构设计应合理，预留必要的扩展字段。后端API接口应版本化，为未来可能的升级提供支持。

1. 测试策略
   1. 测试层级与范围

* 单元测试：针对后端核心业务逻辑、工具函数、前端工具函数和组件进行测试。目标是保证核心代码块的正确性。后端单元测试覆盖率应不低于80%。
* 集成测试：测试模块间的接口调用和数据交互是否正确。包括：前后端API接口联调测试、数据库读写操作测试、Redis缓存操作测试、WebSocket连接与消息处理测试。
* 系统测试（端到端测试）：从用户视角出发，使用自动化测试工具模拟真实用户操作，对关键业务流程进行完整测试。例如：用户注册->登录->创建拼图->游玩->存档->分享整个流程。
* 专项测试：包括性能测试（压力测试、负载测试）、安全性测试（漏洞扫描、渗透测试）、兼容性测试（跨浏览器、跨设备测试）。
  1. 测试环境与数据
* 建立独立的测试环境，包括测试数据库、测试Redis服务器等，确保测试不会影响生产环境数据。
* 测试数据管理：使用工厂模式或Fixture工具预先制造高质量的测试数据，并在测试开始前初始化数据库状态，测试结束后进行清理，保证测试的独立性和可重复性。
  1. 测试工具选型
* 后端单元/集成测试：采用 Pytest 框架。
* 前端单元测试：采用 Jest 或 Vitest。
* 端到端测试：采用 Cypress 或 Playwright。
* 性能测试：采用 Locust 或 JMeter。
* 自动化测试流程：使用 GitHub Actions 或 Jenkins 配置CI/CD流水线，实现代码提交后自动运行测试套件。

1. 部署与监控方案
   1. 部署架构

* 采用前后端分离的部署方式。
* 前端：将Vue.js/React应用构建生成的静态文件（HTML, CSS, JS）部署至Nginx服务器上，由Nginx提供静态文件服务。
* 后端：使用Gunicorn作为WSGI HTTP服务器来运行Flask应用，并使用Nginx作为反向代理，将动态请求转发给Gunicorn。使用Supervisor进程管理工具来监控和守护Gunicorn进程，确保应用崩溃后能自动重启。
* 数据库：MySQL数据库部署在独立的服务器或云数据库服务上。
* 缓存：Redis部署在独立的服务器或云缓存服务上。
  1. 部署流程
* 实现自动化部署流程。使用Git进行版本控制，在代码合并到主分支后，通过CI/CD工具（如GitHub Actions、Jenkins）自动执行构建、测试和部署脚本。
* 部署过程应做到零停机或短时停机，可采用蓝绿部署或滚动更新策略。
  1. 监控与日志
* 应用性能监控：使用Prometheus监控服务器资源使用情况（CPU、内存、磁盘、网络），使用Grafana进行可视化展示和告警。监控关键业务指标，如QPS、接口响应时长、错误率。
* 错误追踪：集成Sentry等错误追踪系统，实时收集前端和后端的运行时错误、异常信息，并及时通知开发团队。
* 日志记录：应用应输出结构化的日志文件。使用ELK Stack或Loki等日志管理系统对日志进行集中收集、存储、检索和分析，便于故障排查和业务分析。

1. 总结

本设计文档在《软件需求规格说明书》的基础上，进一步明确了“拼图大师”游戏系统的技术实施方案与质量保障要求。文档涵盖了系统架构、模块设计、数据库规划、接口定义等核心设计要素，并特别补充了关于非功能性属性、系统测试以及后期运维部署的详细策略。

后续开发工作将严格遵循本设计文档的指导进行。各开发成员需确保其代码实现与设计目标保持一致。测试团队可依据本文档中的测试策略编写具体的测试用例。随着开发的深入，本设计文档也可能需要根据实际情况进行迭代和微调，届时需及时同步更新，以保证其始终是项目开发最权威的参考依据。