**图书管理系统架构报告**

**一、引言**

**（一）项目背景**

本项目旨在开发一款基于 Spring Boot 与 Vue 的图书管理系统，实现图书馆的数字化管理，涵盖图书信息管理、借阅管理、用户管理等核心功能。通过前后端并行开发模式，面向接口与服务设计，确保系统的可扩展性、可维护性及高可用性。

**（二）架构设计目标**

前后端分离：前端基于 Vue.js 构建用户界面，后端采用 Spring Boot 提供服务接口，实现开发解耦与并行协作。

面向接口与服务：定义标准化 RESTful API，支持服务复用与第三方集成；基于微服务思想拆分核心功能模块，降低耦合度。

分层架构：遵循经典三层架构（表现层、业务层、数据层），结合 Web 应用架构的层次特性，实现 “分而治之” 的设计原则。

数据驱动：针对图书、用户、借阅等结构化数据，采用关系型数据库（MySQL）进行持久化管理，确保数据一致性与完整性。

**二、架构设计原则**

**（一）分层设计原则**

层次隔离：各层仅通过接口交互，如前端通过 HTTP 调用后端 API，后端业务层通过 DAO 访问数据层，避免跨层依赖。

耦合：模块间通过抽象接口通信，例如用户服务与借阅服务通过公共 API 交互，局部变化不影响全局逻辑。

可扩展性：支持动态扩展各层资源，如通过负载均衡扩展前端服务器，通过数据库分库分表扩展数据层。

**（二）接口设计原则**

RESTful 规范：采用标准 HTTP 方法（GET/POST/PUT/DELETE）操作资源，如GET /books/{id}获取单本图书信息，POST /borrows创建借阅记录。

版本化管理：通过 URL 路径（如/api/v1/users）或请求头（Accept: application/vnd.app.v1+json）区分接口版本，支持迭代升级。

**（三）服务设计原则**

模块化拆分：将系统拆分为用户服务、图书服务、借阅服务三大核心模块，各模块独立部署、独立维护。

无状态性：后端服务不保存客户端会话状态，通过 JWT（JSON Web Token）实现用户认证与授权，支持分布式部署。

高内聚：每个服务聚焦单一业务领域，如用户服务负责用户注册、登录、权限管理；图书服务负责图书分类、信息维护。

1. **技术选型与架构分层**

**（一）架构分层设计**

**1. 前端层（表现层**）

职责：负责用户交互界面展示、数据输入输出及前端逻辑处理。

架构特性：

采用 SPA 架构，通过路由（Vue Router）实现页面无刷新切换，提升用户体验。

组件化开发：将界面拆分为登录组件、图书列表组件、借阅表单组件等，支持复用与热更新。

状态管理：通过 Vuex 集中管理用户登录状态、全局配置等，确保数据一致性。

交互方式：通过 Axios 发送 HTTP 请求调用后端 API，支持请求拦截（添加 JWT 令牌）与响应处理（错误提示）。

**2. 后端层（业务逻辑层**）

职责：处理核心业务逻辑，协调数据层与前端层交互。

分层结构：

控制器（Controller）：接收前端请求，校验参数，调用服务层接口，返回响应数据。

服务层（Service）：封装业务逻辑，如图书借阅时检查库存、更新借阅状态，支持事务管理（@Transactional）。

数据访问层（DAO）：通过 MyBatis 操作数据库，实现 CRUD 操作与复杂查询，提供统一数据访问接口。

模块划分：

用户服务：处理用户注册、登录、权限校验（如管理员与普通读者权限区分）。

图书服务：管理图书信息增删改查、分类维护（关联book\_type表）。

借阅服务：处理借阅申请（更新isBorrowed状态）、归还操作（记录归还时间）、逾期提醒逻辑。

**3. 数据层（持久化层）**

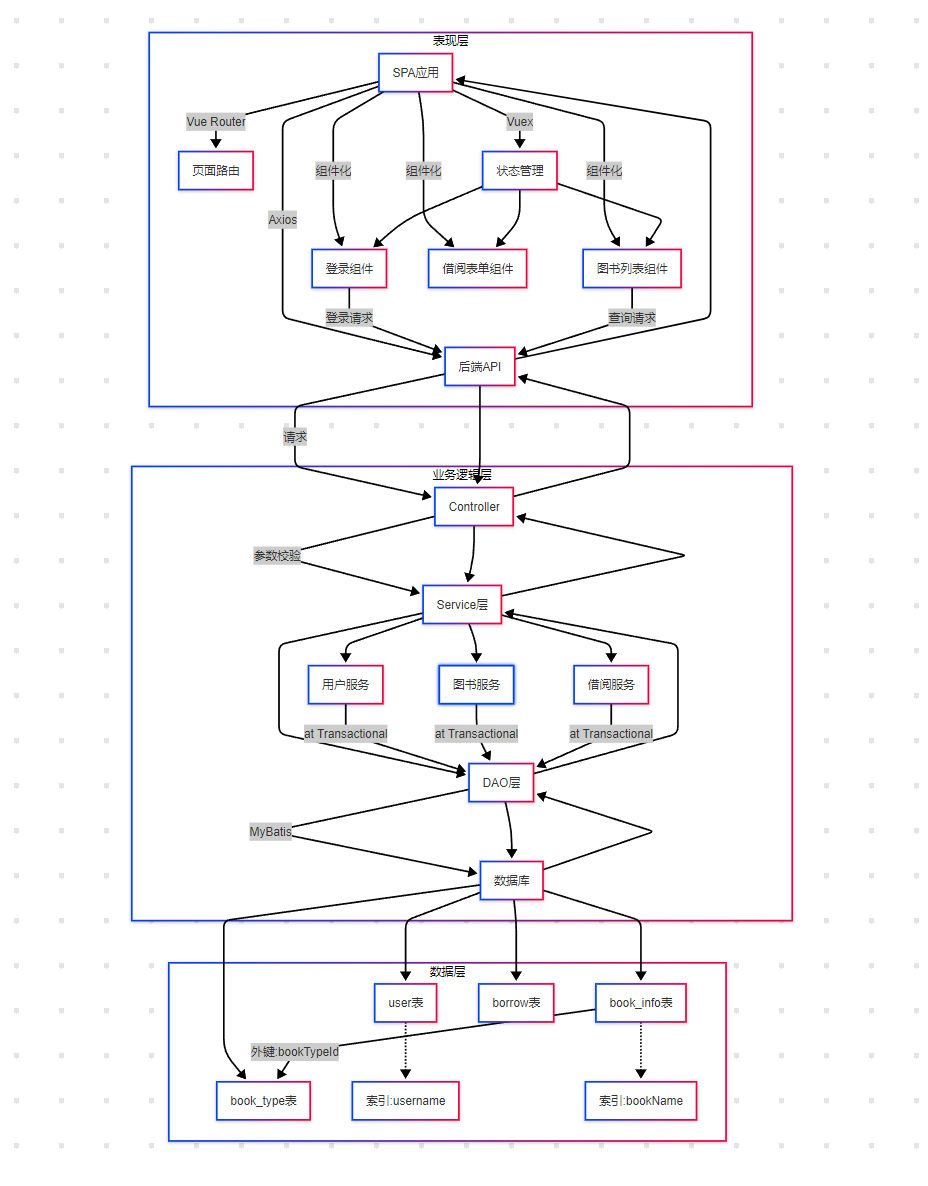
职责：存储与管理系统核心数据，提供数据持久化接口。

设计要点：

数据库设计：遵循第三范式，设计user（用户表）、book\_info（图书表）、borrow（借阅表）、book\_type（图书分类表），通过外键关联（如book\_info.bookTypeId关联book\_type.bookTypeId）。

索引优化：对高频查询字段（如user.username、book\_info.bookName）添加索引，提升查询效率。

事务管理：通过 Spring 声明式事务确保数据一致性，如借阅操作同时更新图书表与借阅表。



**四、服务设计**

**用户登录流程**：

前端发送用户名、密码至/login接口。

后端校验通过后生成 JWT 令牌，返回给前端存储（如 LocalStorage）。

后续请求携带 JWT 令牌，后端通过拦截器（Interceptor）验证令牌有效性，解析用户权限。

**图书借阅流程**：

读者在前端选择图书，调用/borrows/{bookId}接口。

后端服务层检查图书状态（isBorrowed），若可用则更新图书表为 “已借出”，并在借阅表新增记录。

事务管理确保两步操作原子性，失败时回滚。

**四、开发策略与接口设计**

**前后端并行开发**

接口优先：在开发初期，定义清晰的接口规范，确保前后端团队对接口的理解一致。

Mock Server：前端团队可以使用Mock Server模拟后端接口，提前进行前端开发和测试。

**接口设计**

接口规范：定义清晰的接口规范，确保前后端团队对接口的理解一致。

服务化设计：将系统功能模块化，每个模块提供独立的服务，便于扩展和维护。

**五、总结与展望**

**（一）当前架构优势**

前后端分离：提升开发效率，支持团队并行协作，降低模块间依赖。

面向接口设计：增强系统灵活性与可复用性，为后续微服务化改造奠定基础。

分层架构：符合 Web 应用架构的最佳实践，确保系统可维护性与可扩展性。

**（二）未来优化方向**

微服务化：将用户、图书、借阅服务拆分为独立微服务，通过 API 网关（如 Spring Cloud Gateway）统一路由，支持动态扩容与熔断机制。

缓存优化：引入 Redis 缓存高频访问数据（如热门图书列表、用户权限信息），降低数据库压力。

容器化部署：使用 Docker 容器封装服务，结合 Kubernetes 实现自动化部署与运维，提升系统可靠性。

通过以上架构设计，图书管理系统将实现前后端高效协作、接口标准化与服务模块化，满足当前功能需求的同时，为未来扩展预留充足空间。