1. **架构设计**
2. 整体架构概览

Frontend

用户界面层（Vue）

JSON/REST API

Backend

应用逻辑层（Flask）

ORM / SQLAlchemy

Datebase

数据持久化层（MySQL）

**技术架构**

* **技术栈:** Python, Vue.js, Flask, SQLAlchemy, MySQL
* **架构模式:** MVC (Model-View-Controller)
* **通信协议:** RESTful API
* **数据存储:** MySQL
* **版本控制:** Git

**数据库设计 (SQLAlchemy & MySQL)**

* 使用 SQLAlchemy 作为 ORM 工具。
* 定义各个模块对应的模型，例如 User, Requirement, Architecture, TestCase, Version 等。
* 使用 MySQL 作为数据库存储数据。

**前端设计 (Vue.js)**

* 使用 Vue.js 框架构建前端应用。
* 使用 Vue Router 管理路由。
* 使用 Vuex 或 Vue’s Composition API 管理状态。
* 按照模块划分组件，例如 UserComponent, RequirementComponent, ArchitectureComponent 等。

**后端设计 (Flask)**

* 使用 Flask 框架构建后端 API。
* 使用 Flask-RESTful 或 Flask 的路由系统创建 RESTful API。
* 使用 Flask-JWT-Extended 或其他 JWT 库处理用户认证和授权。
* 使用 SQLAlchemy 作为 ORM 工具，与 MySQL 交互。

1. 模块与模块关系设计

|  |  |  |
| --- | --- | --- |
| **模块名** | **职责描述** | **调用关系** |
| **User Module** | 用户注册、登录、权限校验 | 所有模块需要用户上下文 |
| **Requirement Module** | 管理需求的提交、修改、版本控制 | 作为入口模块，连接其他模块 |
| **LLM Integration Module** | 调用大模型生成架构、代码、测试 | 与架构/代码/测试模块配合 |
| **Architecture Module** | 管理系统架构图（如UML结构） | 从LLM接收图信息并存储 |
| **Code Generation Module** | 生成各模块的代码（Python/HTML等） | 接收架构信息调用LLM |
| **Database Design Module** | 自动生成ER图与建表SQL | 可与Code模块共享数据结构 |
| **TestCase Module** | 根据需求生成测试用例 | 与LLM模块协作，后续关联部署模块 |
| **Deployment Module** | 执行部署代码、运行环境配置 | 显示运行结果和日志 |
| **Versioning & Config Module** | 保存各版本数据、Git集成 | 支持Requirement、Code等模块 |

* 模块之间的依赖关系示意

User

└── Requirement

├── LLM Integration

│ ├── Architecture

│ ├── Code Generator

│ ├── TestCase

│ └── DB Design

└── Version & Config

└── Deployment

1. 前后端职责分离

|  |  |  |
| --- | --- | --- |
| **层级** | **职责** | **技术栈** |
| **Frontend** | 提供用户交互页面；收集需求输入；展示架构图、代码、测试结果等 | Vue.js, React, Tailwind CSS |
| **Backend** | 提供API服务；处理用户请求逻辑；调用LLM；管理数据 | Flask, FastAPI, |
| **DB** | 存储用户、需求、代码、架构图、测试用例、运行日志等结构化数据 | MySQL |

1. 模块通信方式

模块之间的通信采用RESTful API + 内部方法调用（函数依赖）为主：

前端与后端通信：

* 通过 Axios/Fetch 发起 HTTP 请求

后端模块间通信：

* Controller 层进行模块之间的协调调度
* Service 层实现具体逻辑调用

后端与数据库通信：

* 使用 **ORM 框架**（ SQLAlchemy / Django ORM）简化数据库操作

1. **数据设计**
2. ER图

图示

AI 生成的内容可能不正确。图示, 示意图

AI 生成的内容可能不正确。

1. 数据实体定义

### **1. users**

用户表（平台使用者）

|  |  |  |
| --- | --- | --- |
| **字段名** | **类型** | **描述** |
| id | INT, PK | 用户主键 |
| username | VARCHAR(50) | 用户名 |
| email | VARCHAR(100) | 邮箱 |
| password\_hash | VARCHAR(255) | 加密密码 |
| role | VARCHAR(20) | 用户角色（admin/student） |
| created\_at | TIMESTAMP | 注册时间 |

### **2. requirements**

### 需求表（用户输入的项目需求）

|  |  |  |
| --- | --- | --- |
| **字段名** | **类型** | **描述** |
| id | INT, PK | 主键 |
| user\_id | INT, FK | 外键，关联用户 |
| content | TEXT | 需求原文（自然语言） |
| version | INT | 当前版本号 |
| name | VARCHAR(20) | 需求简述 |
| created\_at | TIMESTAMP | 创建时间 |

### 3. architectures

### 架构设计表（由LLM生成或手动上传）

|  |  |  |
| --- | --- | --- |
| **字段名** | **类型** | **描述** |
| id | INT, PK | 主键 |
| requirement\_id | INT, FK | 所属需求 |
| architecture\_json | TEXT | 架构内容（JSON格式/UML） |
| generated\_by | VARCHAR(20) | LLM / 手动 |
| created\_at | TIMESTAMP | 时间戳 |

### 4. module\_codes

### 模块代码表（代码生成结果）

|  |  |  |
| --- | --- | --- |
| **字段名** | **类型** | **描述** |
| id | INT, PK | 主键 |
| architecture\_id | INT, FK | 所属架构 |
| module\_name | VARCHAR(50) | 模块名称 |
| language | VARCHAR(20) | 编程语言 |
| code | TEXT | 源代码内容 |
| created\_at | TIMESTAMP | 时间戳 |

### 5. database\_designs

### 数据库设计表（ER图和SQL脚本）

|  |  |  |
| --- | --- | --- |
| **字段名** | **类型** | **描述** |
| id | INT, PK | 主键 |
| requirement\_id | INT, FK | 所属需求 |
| erd\_diagram | TEXT | ER图JSON或图片链接 |
| sql\_script | TEXT | 建表SQL脚本 |
| created\_at | TIMESTAMP | 时间戳 |

### 6. test\_cases

### 测试用例表（自动生成或手动添加）

|  |  |  |
| --- | --- | --- |
| **字段名** | **类型** | **描述** |
| id | INT, PK | 主键 |
| requirement\_id | INT, FK | 所属需求 |
| input\_data | TEXT | 测试输入 |
| expected\_output | TEXT | 预期输出 |
| type | VARCHAR(30) | 功能测试 / 边界 / 异常等 |
| created\_at | TIMESTAMP | 时间戳 |

### 7. deployment\_logs

### 部署记录表（系统部署结果追踪）

|  |  |  |
| --- | --- | --- |
| **字段名** | **类型** | **描述** |
| id | INT, PK | 主键 |
| architecture\_id | INT, FK | 所属架构 |
| status | VARCHAR(20) | 成功 / 失败 |
| log\_output | TEXT | 输出日志 |
| executed\_at | TIMESTAMP | 执行时间 |

### 8. version\_history

### 通用版本历史表（可用于需求、架构、代码等）

|  |  |  |
| --- | --- | --- |
| **字段名** | **类型** | **描述** |
| id | INT, PK | 主键 |
| entity\_type | VARCHAR(30) | 版本对象（如requirement） |
| entity\_id | INT | 版本主体的ID |
| version | INT | 版本号 |
| data\_snapshot | TEXT | 数据快照 |
| created\_at | TIMESTAMP | 创建时间 |

1. 数据库表关系设计

明确表之间的关系

* users 表与 requirements 表是一对多关系（一个用户可以有多个需求）。
* requirements 表与 architectures 表是一对多关系（一个需求可以有多个架构设计）。
* architectures 表与 module\_codes 表是一对多关系（一个架构设计可以有多个模块代码）。
* requirements 表与 database\_designs 表是一对多关系（一个需求可以有多个数据库设计）。
* requirements 表与 test\_cases 表是一对多关系（一个需求可以有多个测试用例）。
* architectures 表与 deployment\_logs 表是一对多关系（一个架构设计可以有多个部署记录）。
* version\_history 表与所有其他表是一对多关系（一个实体可以有多个版本历史）。

**详细设计**

1. **接口设计**
2. 六个核心模块的接口：
3. 用户模块（User）

**职责:** 用户注册、登录、权限校验

**调用关系:** 所有模块都需要进行用户认证和权限校验，因此都会调用 User Module 的 API。

* **POST /api/users/register：用户注册  
  Request Body:**

{

"username": "\*\*\*",

"email": "\*\*\*@example.com",

"password": "\*\*\*"

}

**Response:**

{ "message": "User created", "user\_id": 1 }

* **POST /api/users/login ：用户登录  
  Request Body:**

{

"email": "\*\*\*@example.com",

"password": "\*\*\*"}

**Response:**

{ "token": "jwt\_token\_here" }

1. 需求模块（Requirement）

* **POST /api/requirements/：提交新需求**  
  **Headers: Authorization: Bearer <token>  
  Request Body:**

{

"content": "Build an AI-based book recommendation system"}

**Response:**

{ "requirement\_id": 21, "version": 1 }

* **GET /api/requirements/{id}：获取需求详情**  
  **Response:**

{

"id": 21,

"user\_id": 1,

"content": "Build an AI-based book recommendation system",

"version": 1,

"created\_at": "2025-05-25T10:00:00Z"

}

1. 架构模块（Architecture）

* **POST /api/architectures/generate：基于需求生成架构（调用 LLM）  
  Request Body:**

{

"requirement\_id": 21

}

**Response:**

{

"architecture\_id": 102,

"architecture\_json": { module structure }

}

* **GET /api/architectures/{id}：获取架构详情  
  Response:**

{

"id": 102,

"requirement\_id": 21,

"generated\_by": "LLM",

"architecture\_json": { "modules": [...] }

}

1. 代码生成模块（Code Generation）

* **POST /api/modules/generate：生成模块代码**  
  **Request Body:**

{

"architecture\_id": 102

}

**Response:**

{

"modules": [

{

"module\_name": "UserService",

"language": "Python",

"code": "class UserService: ..."

}

]

}

1. 数据库设计模块（DB Design）

* **POST /api/databases/generate：自动生成数据库设计  
  Request Body:**

{

"requirement\_id": 21

}

**Response:**

{

"erd\_diagram": "base64encoded\_or\_url",

"sql\_script": "CREATE TABLE user (...)"

}

1. 测试用例模块（TestCase）

* **POST /api/testcases/generate：生成测试用例  
  Request Body:**

{

"requirement\_id": 21

}

**Response:**

{

"testcases": [

{

"input\_data": "GET /books?id=1",

"expected\_output": "200 OK",

"type": "functional"

}

]

}

1. **界面设计**
2. **详细模块拆解​**

**​**1. 左侧导航栏 (Primary Navigation)​​

组成​：

品牌标识​：顶部Logo（无文字）

功能菜单​（垂直排列）：

1. 需求管理（当前选中状态）

2. 架构设计

3. 模块代码

4. 数据库设计

5. 测试用例生成

6. 版本历史

2. 顶部标题栏 (Workspace Header)​​

组成​：

标题​："架构设计"（与左侧导航选中项同步）

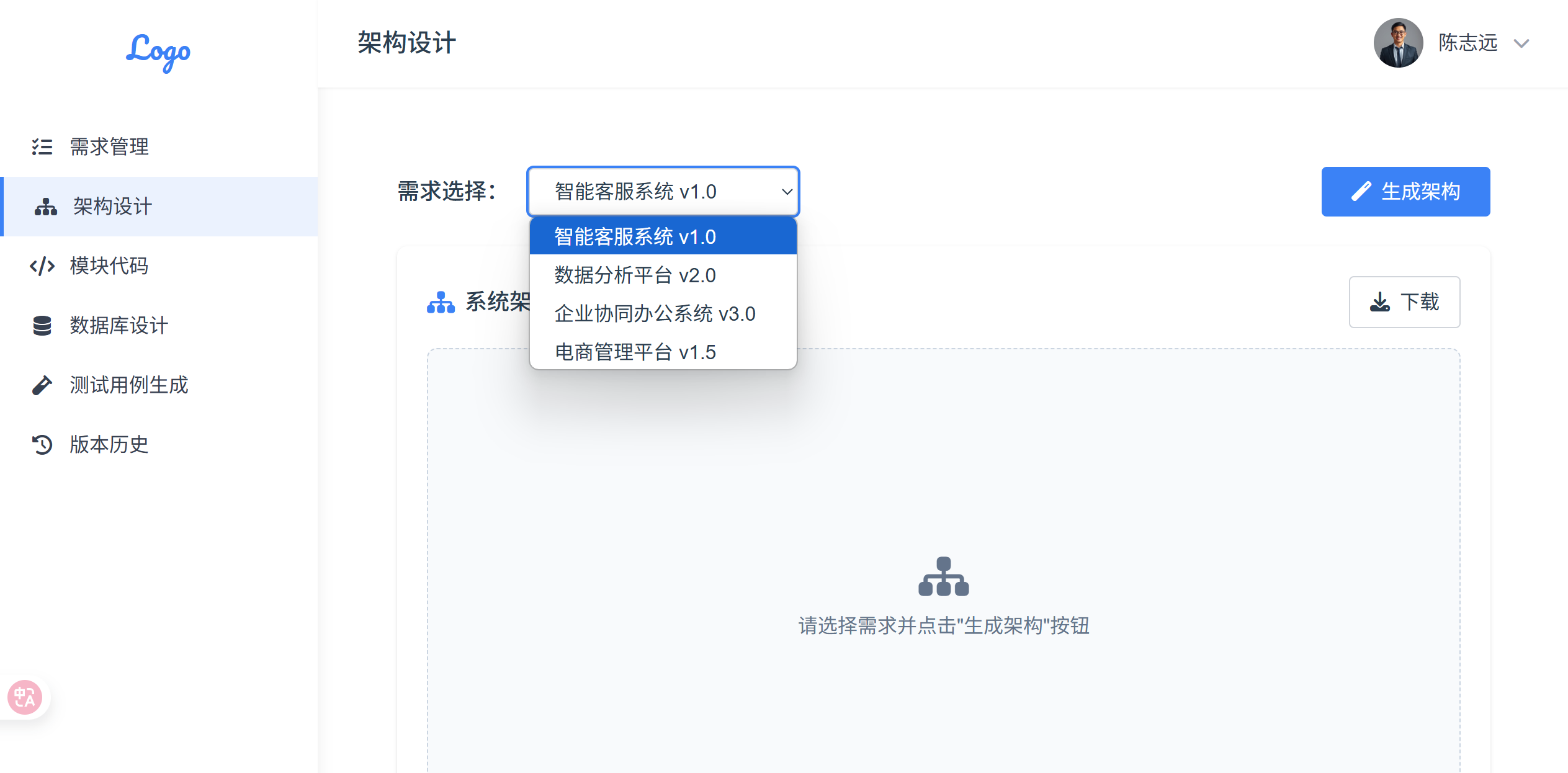
用户信息​：右对齐

操作按钮​：蓝色"+新建需求"按钮（Floating Action Button样式）

3. 功能模块



图表 1需求管理



图表 2架构设计



图表 3模块代码



图表 4数据库设计



图表 5测试用例生成



图表 6版本历史