基于Uniapp的英语背单词小程序

王品寰(2212190525)

俞孟琪(2212190108 ）

# 一、项目介绍 [王品寰，俞孟琪]

## 1.1 背景与问题陈述

本项目是基于 Uniapp（前端）和Java + Spring Boot（后端）开发的英语背单词小程序，主要面向高中生、大学生及考研人群等需要提升词汇量的用户。

小程序提供：单词填空拼写、单词记忆功能、单词查找搜索、单词中英文互译、单词收藏本、文本翻译、时间提醒等功能，并主页设有单词学习数量提醒用以帮助用户培养每日学习新单词、复习旧单词的好习惯，从而更好地巩固词汇。

**传统英语词汇学习痛点：**

1. **学习方式低效**：纸质单词书缺乏互动，记忆留存率低
2. **缺乏针对性**：用户难以聚焦特定考试词汇（如四六级/考研）
3. **场景局限**：无法支持碎片化学习和即时练习

**解决方案：**

多模态训练（单词拼写/辨析）提升记忆效率

个性化词库导入实现精准学习

英文阅读场景强化词汇应用能力

## 1.2 项目目标与价值

**我们做这个背单词APP，主要是想实实在在解决几个痛点,**让背单词不那么枯燥，能坚持下来**，达到这些具体目标：**

1.提供像填空、翻译、闯关这些互动练习，让用户（特别是备考的同学）每天能愿意花个20-30分钟在上面，而不是对着单词书犯困。

2.别浪费时间在整理单词上,让大家能直接使用现成的、整理好的词库（比如高考3500、四六级核心、考研红宝书这种）。省下自己抄单词、找单词书的时间，直接开背！特别是备考冲刺期，时间就是分数。

3.优化学习效率：单词收藏本、文本翻译等功能可以让用户拥有自己“专有”的生词库，省去单词整理时间，直击重点词汇

4.强化应用能力，应用文本翻译模块，实现"学以致用"的闭环

5.拓展学习场景：移动端支持碎片化学习（通勤/排队/课间）

6.保障技术稳定性：高并发场景下稳定运行（如全班同步操作）

**总结一下，这个项目的价值就是**：

对用户（学生党）： 提供一个更高效、有趣、省时省力的背单词工具，特别是针对考试复习，帮助大家克服背单词的惰性，提升词汇量和应用能力，最终目标是在考试里多拿分。

对我们（开发者）： 实践了前后端全栈开发（Java Spring Boot + UniApp），把课堂知识用到了真实项目里，解决了实际问题，做了个真正有人用的东西出来，也算是为将来的开发积累经验

## 1.3 功能需求分析

**1.3.1单词学习功能**（**核心功能模块）**

-单词填空拼写：用户根据中文释义/英文缺词提示拼写单词

- 文本翻译功能：可将英文文本转换为中文，精准度较高。

- 单词中英文互译 ：中→英/英→中双向即时翻译。

- 单词通关考察：每组20个单词需要正确拼写单词，过程中可以随机记录不会的单词。

**1.3.2辅助工具**

- 单词查找搜索：支持中/英文关键词精准检索及跨词库查询

- 文本翻译：支持整段英文文本翻译

- 单词收藏本：收藏难词/重点词，支持自定义分类管理

**1.3.3习惯养成系统**

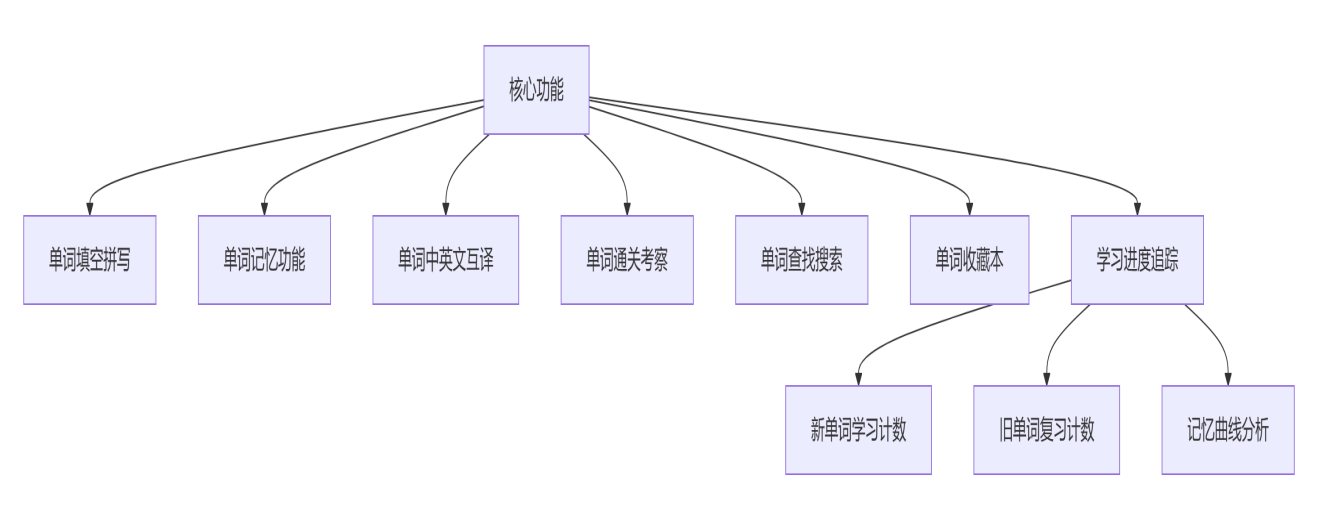
- 时间提醒功能：单词学习时段和复习时间提醒

- 单词学习数量提醒：主页显眼位置展示新单词学习/旧单词复习进度

- 学习进度追踪：每日学习数据统计，包含新学单词量和复习次数

**1.3.4增值功能**

- 单词收藏本管理：创建个性化词库并可以随时对收藏本中的词汇数量进行增减



## 1.4 技术选型与架构概述

**-前端**：UniApp。跨平台开发框架，支持微信小程序、H5、App 等多端编译；基于 Vue.js 开发，组件化管理页面；且作为开发者，对于该技术熟悉。

**-后端**：Java和 Spring Boot

**-开发工具**：IntelliJ IDEA（后端开发）、Visual Studio Code（前端开发）

**-数据库**： mysql（数据库）

mybatis（数据库框架

springboot（整个框架）

## 

# 1.5 项目计划与团队分工

**团队成员分工**

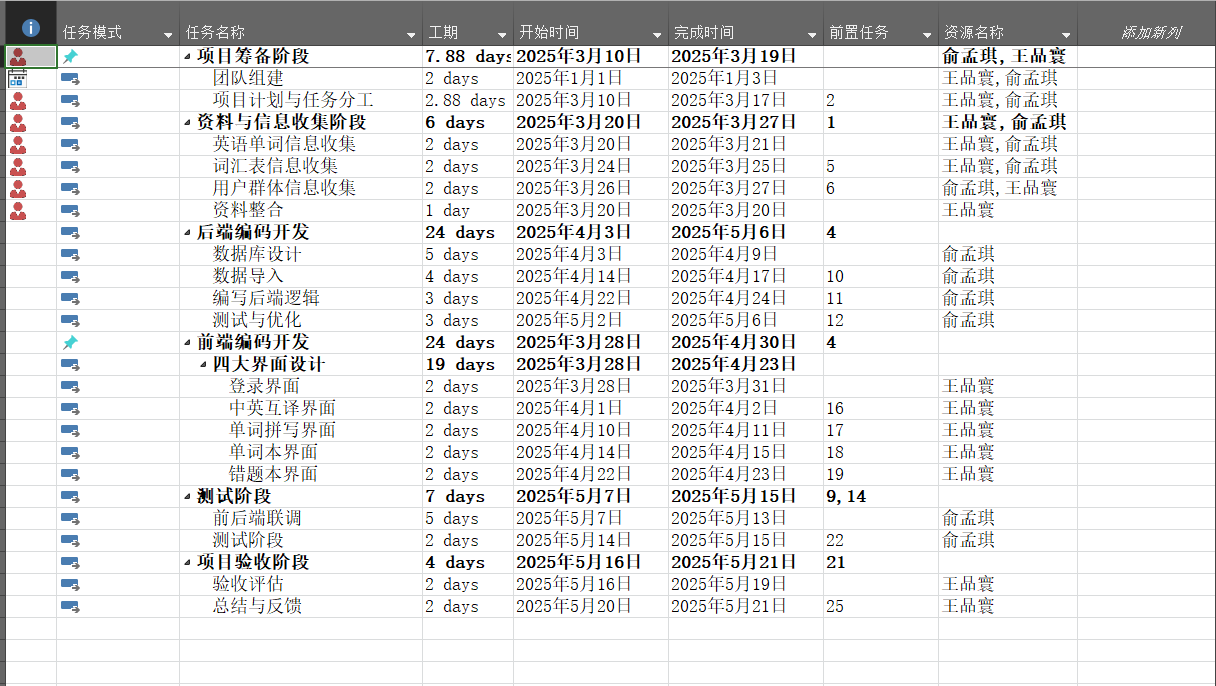
**王品寰（前端开发）**

• 需求分析、前端开发、文档设计、测试部署、维护

**俞孟琪（后端开发）**

• 后端开发、测试部署、API设计、文档设计、维护

**（项目进度计划）//具体文件已上传github，这是部分截图，详细截图在文档后边有具体说明和展示**



# 二、需求分析与设计 [王品寰，俞孟琪]

## 2.1 用户画像与场景分析

## **核心用户群：**备考学生（高中生/大学生）或自律学习者职场/留学党）

## **使用场景：**

## -课间碎片时间刷词

## -通勤地铁背单词

## -晚自习集中突破薄弱词

## -午休时间阅读英文短文

## -考前冲刺高频词库

## **核心需求：**

## -直击考试重点词汇

## -利用零碎时间高效学习

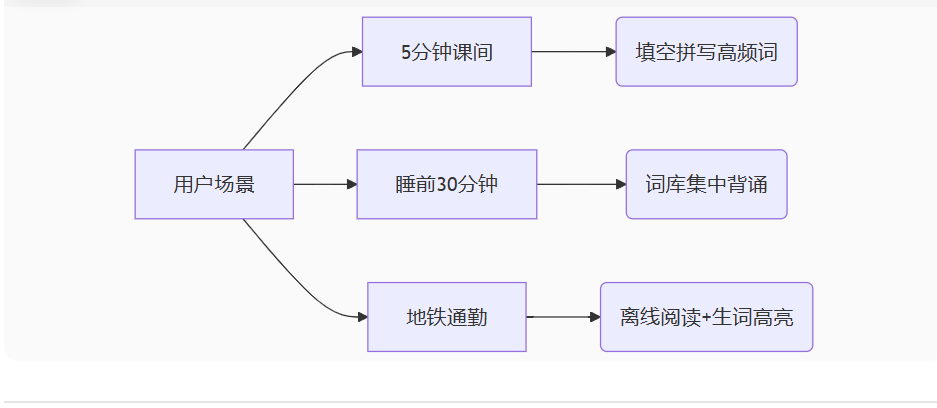
## -避免复杂操作干扰专注

## -离线缓存功能支持

## -学习进度多端同步

## -数据可视化学习报告

**高频使用场景：**



2.2 界面原型设计

## **设计方法论：**

## -模块化组织：按功能模块分组界面设计

## -跳转关系图：每个模块配流程图展示页面跳转逻辑

## -设计理念说明：包含优点/缺点/实现难度分析

## -统一设计图：相似功能集中展示（如用户管理功能组）

## **架构优势：**

## **-**模块化清晰：功能分离(学习/翻译/成绩)

## - 复用性强：common封装核心工具

## -扩展便捷：新增独立模块即可

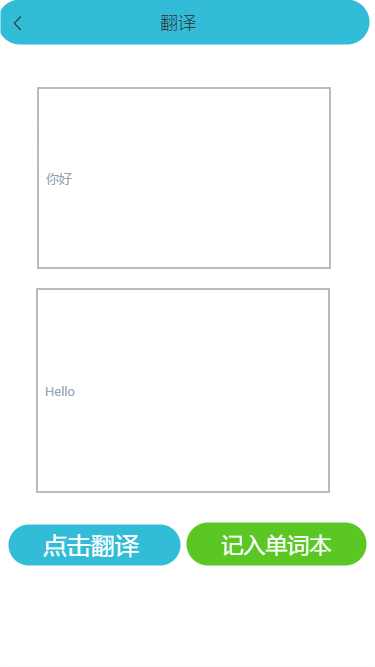
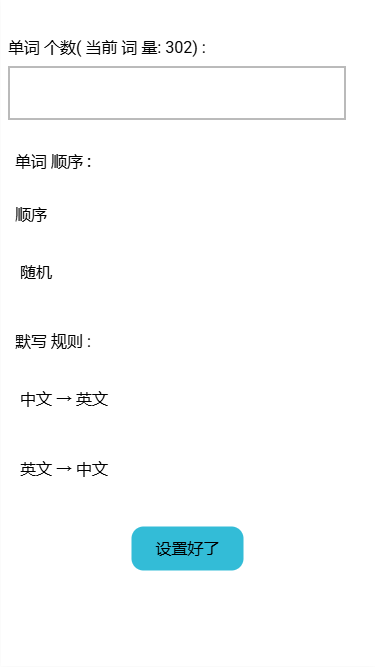
## 

## **后端架构**

## 前端 → API网关 → 微服务(用户/单词/测验) → 存储层(MySQL+Redis)

# **原型设计图**

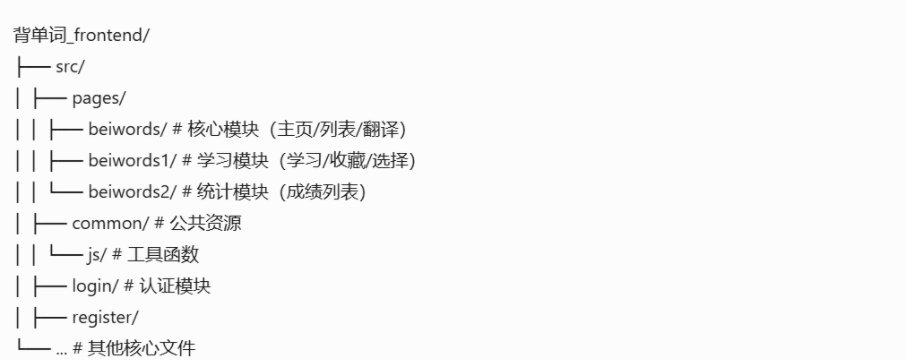
 

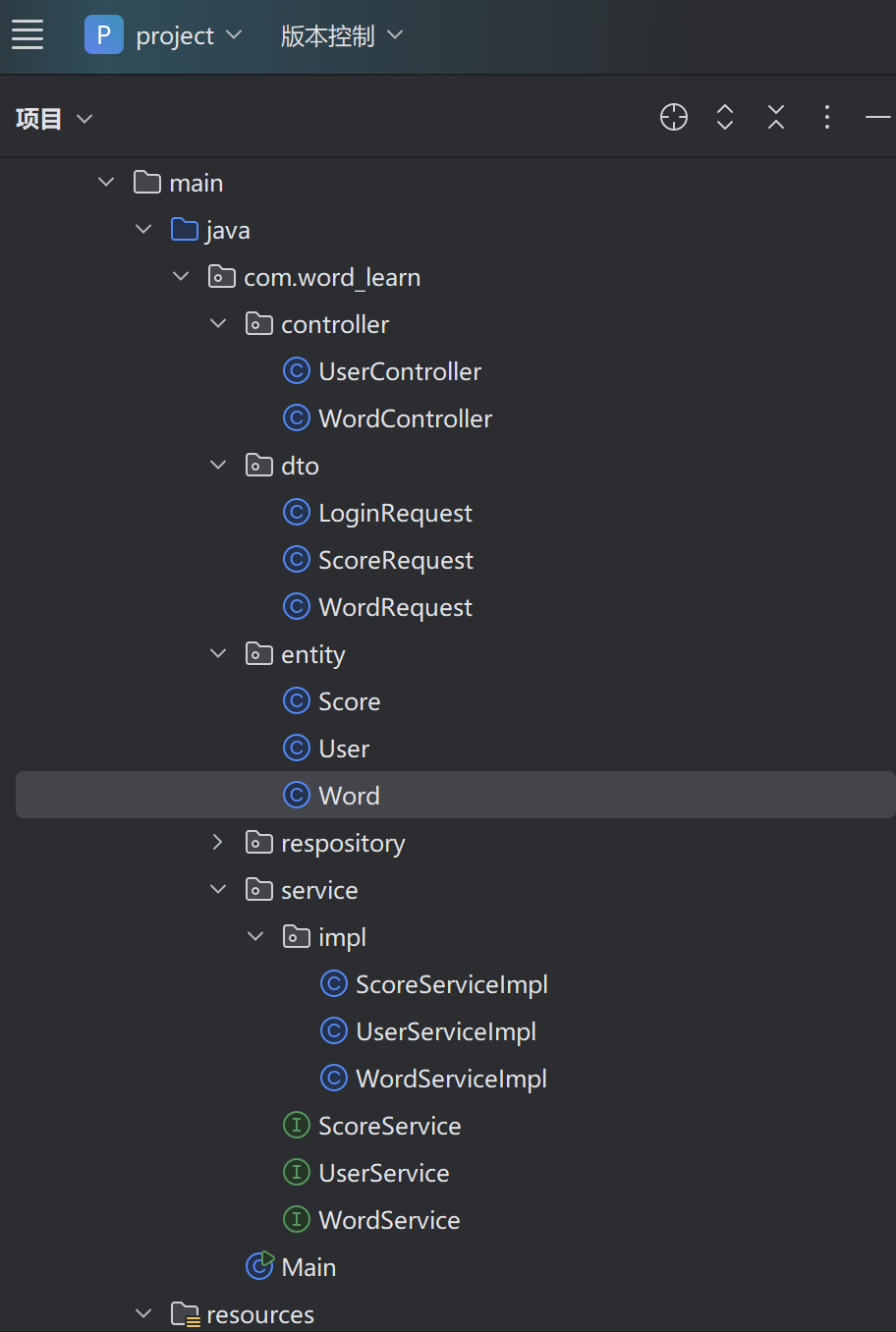
 

# 三、系统架构设计 [王品寰，俞孟琪]

## 3.1 整体架构设计

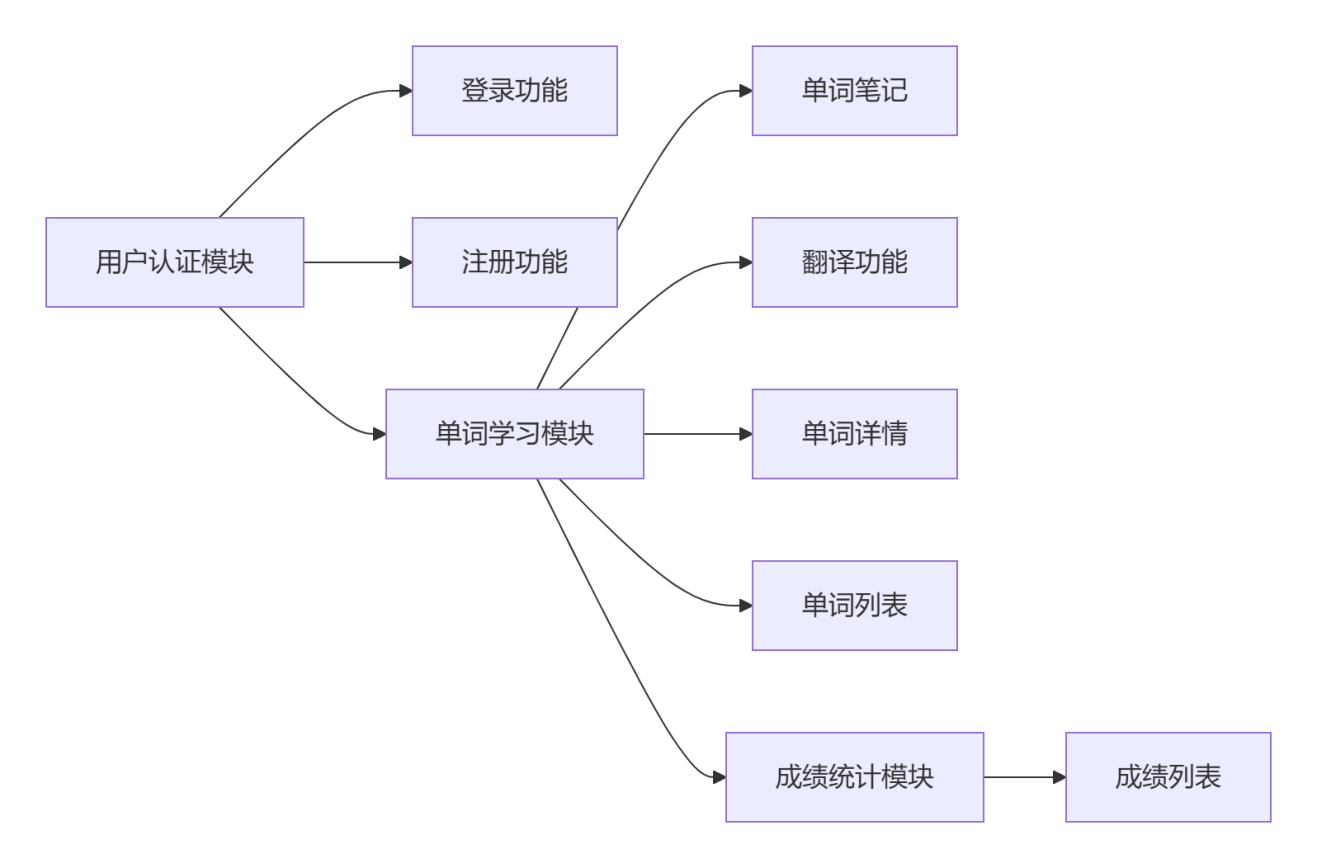
## 前端目录结构：





## 3.2 技术架构分层

### 3.2.1 表现层（前端）



**1. 用户认证模块**

**核心页面：**

-login.vue（登录）

-register.vue（注册）

**功能特点：**

表单验证与Token管理

身份认证集中处理

**设计评估：**

优点：代码复用率高

缺点：缺少自动登录

**2. 单词学习模块**

**核心页面**：

• words.vue（学习主界面）

• beiwords.vue（关联功能）

**功能特点：**

单词列表展示

实时翻译功能

**设计评估：**

优点：学习流程闭环完整

缺点：缺乏记忆算法

**3. 收藏管理模块**

**核心页面：**

favorwords.vue（收藏列表）

**功能特点：**

收藏状态管理

添加/移除操作

**设计评估：**

优点：功能职责单一

缺点：无批量操作

**4. 工具模块**

核心文件：

• common.js（全局常量）

• request.js（网络请求）

**功能特点：**

API统一管理

网络请求封装

**设计评估：**

优点：提升开发效率

缺点：扩展性有限

### 3.2.2 业务逻辑层（后端）

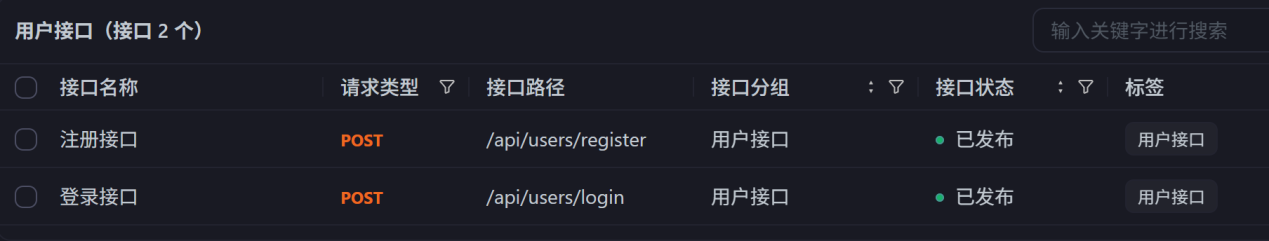
一、 Controller接口

ApiFox

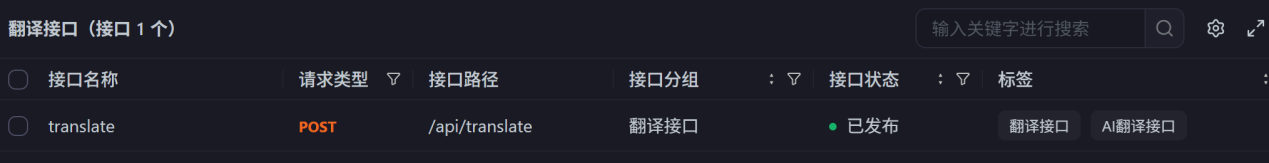
单词类



用户类：



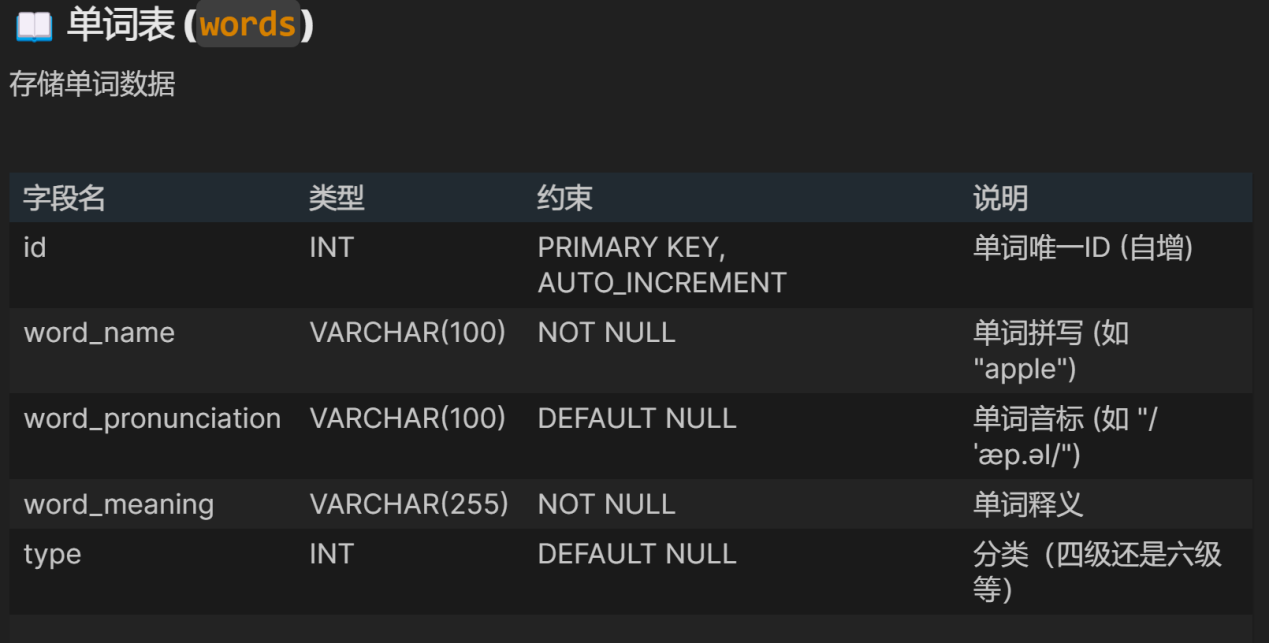
翻译类：

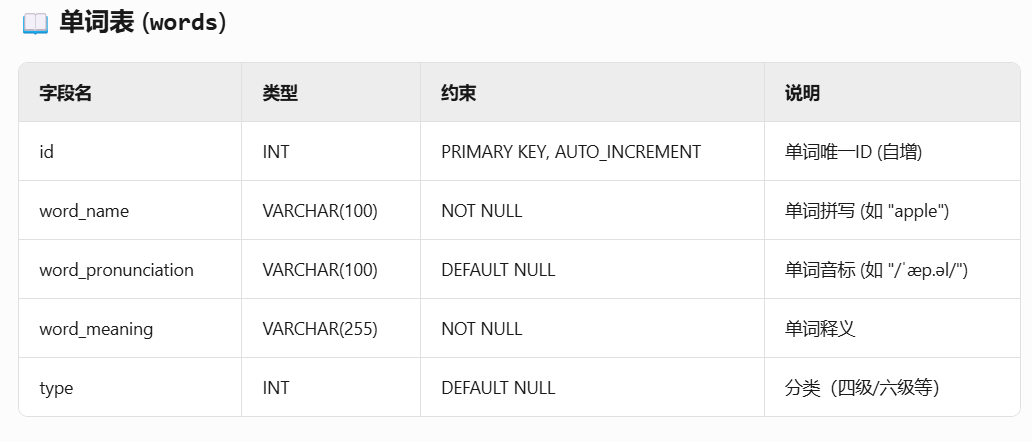


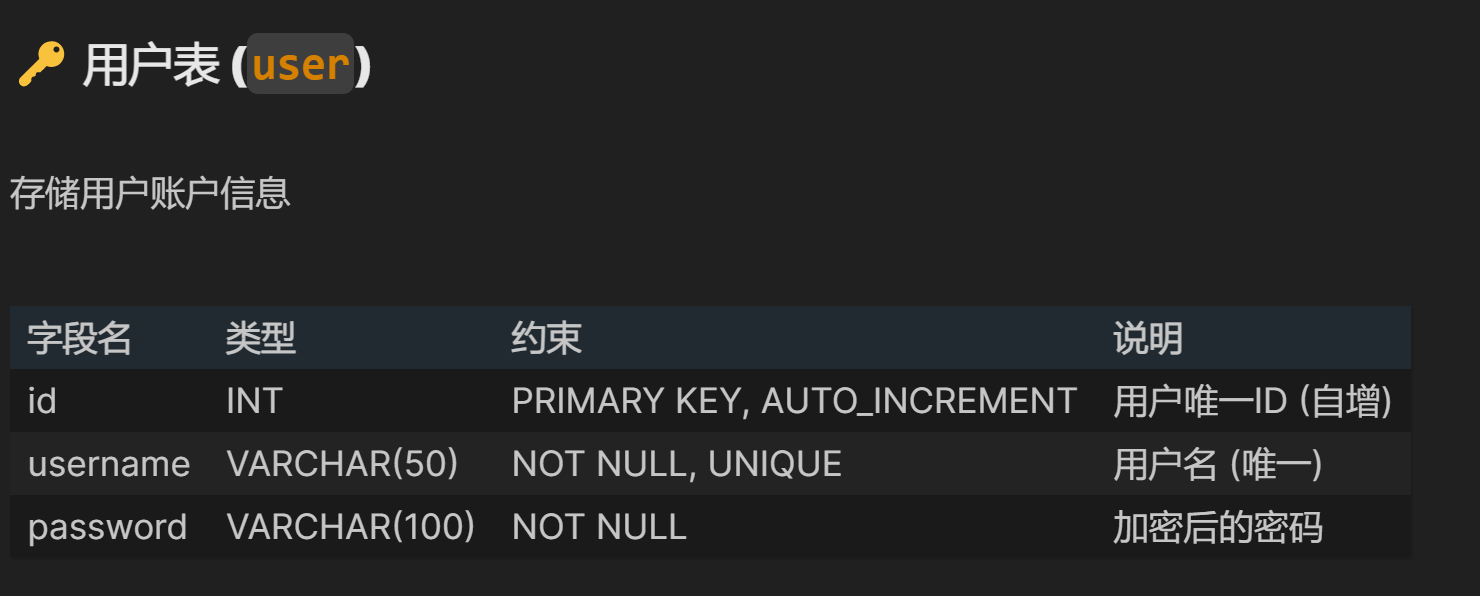
### 3.2.3 数据库

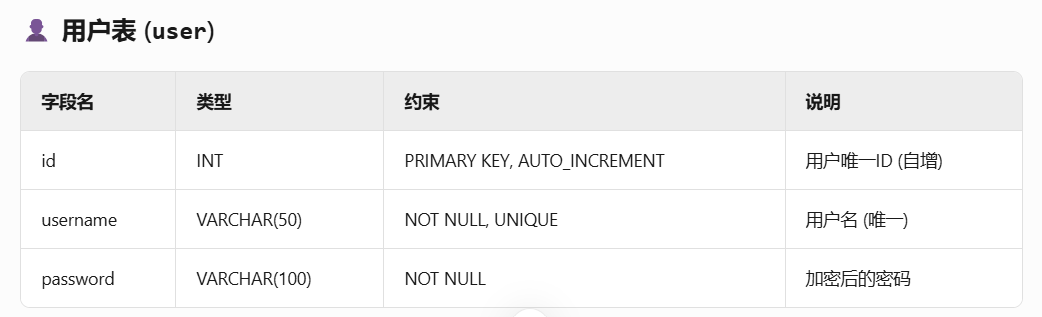
# 











# 四、API设计 [王品寰、俞孟琪]

## 4.1 API设计原则

## API 文档化管理

## 推荐使用如下工具链：

## 

## 4.2 接口文档

**文档更新流程：**​​

开发阶段：代码注解生成初版文档

测试阶段：补充参数示例和响应示例

发布阶段：生成多格式文档包

维护阶段：版本化更新日志管理

### 

### 4.2.1 用户管理接口



**设计理念：**

-采用JWT无状态认证机制

-密码传输使用HTTPS加密

-标准化错误代码（200=成功，400=参数错误，401=未授权）

**数据模型：**

// User 对象

{

"id": "整数，用户ID",

"username": "字符串，用户名",

"password": "字符串，密码"

}

// LoginRequest 对象

{

"username": "字符串，用户名",

"password": "字符串，密码",

"email": "字符串，邮箱（可选）"

}

### 

### 4.2.2翻译服务接口



**设计理念：**

• 统一文本输入接口

• 支持长文本翻译（最大长度500字符）

• 多服务提供商故障转移机制

**数据模型：**

{

"code": "整数，状态码",

"msg": "字符串，消息",

"extra": {

"translation": "字符串，翻译结果",

"sourceLang": "字符串，源语言",

"targetLang": "字符串，目标语言"

}

}

### 

### 4.2.3 单词管理接口

​​

**数据模型：**​​

// WordRequest 对象

{

"word\_id": "整数，单词ID",

"word\_name": "字符串，单词名称",

"word\_meaning": "字符串，单词释义",

"word\_pronunciation": "字符串，发音标记"

}

// PageRequest 分页参数

{

"page": "整数，页码",

"size": "整数，每页数量",

"type": "整数，分类类型（预留）",

"keyword": "字符串，搜索关键词"

}

// 响应通用结构

{

"code": "整数，状态码",

"msg": "字符串，消息",

"extra": {

"data": [],

"pagination": {

"total": "总记录数",

"page": "当前页码",

"size": "每页数量"

}

}

}

4. 学习数据模块



**​​**

**数据模型：​**​

// Score 对象

{

"id": "成绩记录ID",

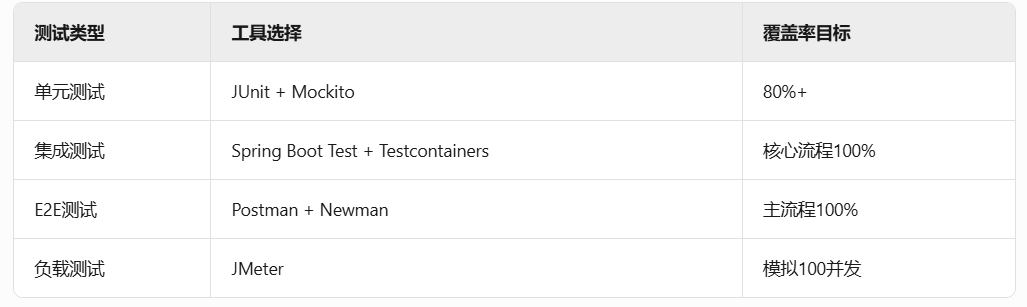
"userId": "用户ID",

"score": "得分值"

}

4.4 API 测试策略与方案

1. 自动化测试框架



2. 关键测试用例矩阵



3.API 设计优化建议

​​版本控制改进​​

当前：/api/words/favor

建议：/api/v1/words/favor

​​标准化错误响应​​

{

"code": "ERROR\_CODE",

"message": "可读的错误描述",

"details": [

{

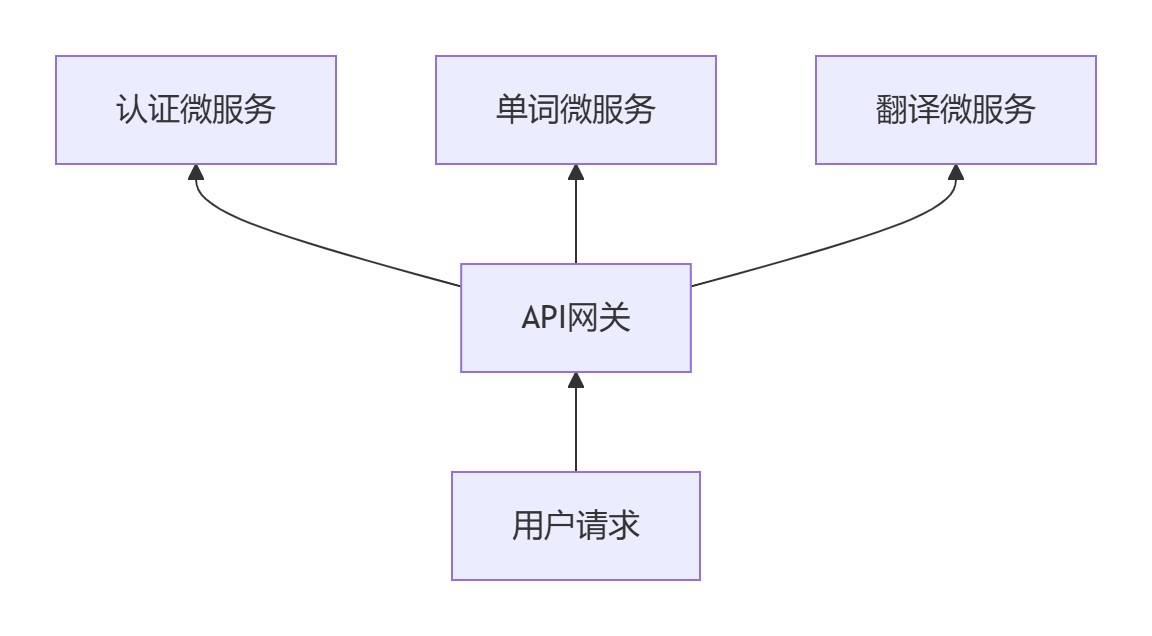
"field": "字段名",

"issue": "具体问题"

}

]

}



# 五、数据库设计 [王品寰，俞孟琪]

## 5.1 数据模型设计​

## 系统需持久化存储的核心数据包括：

## ​​1）用户账户信息（唯一ID、用户名、加密密码）

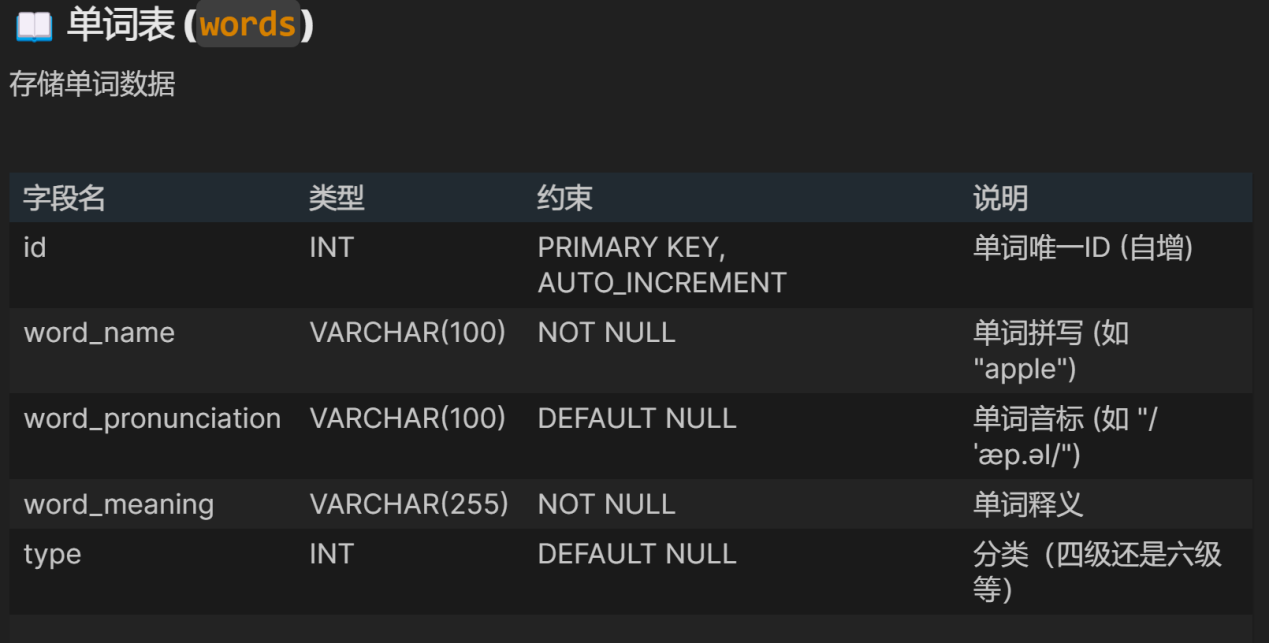
## ​​2）单词数据（拼写、音标、释义、分类）

## ​​3）用户收藏关系（用户与单词的收藏记录）

## 因数据为强结构化关系型数据（如用户-收藏-单词的关联关系），且需支持事务操作（如收藏去重），选用 MySQL 数据库。原因包括：1）开源且稳定性高 2）完善支持事务处理 3）成熟的关系型数据管理能力

## 5.2 数据库架构





## 

### 5.2.1 索引设计

​​**用户表**：username字段建立→ UNIQUE索引→ 加速用户名查询及唯一性校验

​​**收藏表**：word\_id字段建立普通索引 → 优化按单词查收藏的筛选效率

**联合字段**(user\_id, word\_id)建立唯一索引 → 高效实现收藏去重

### 5.2.2 数据库性能优化

​​-合理设置索引：针对高频查询字段（如username, word\_id）建立索引

​​-避免冗余字段：通过外键关联实现数据复用（如收藏表关联用户表/单词表）

# 六、前端实现 [王品寰]

## 6.1.1 技术栈与开发环境

**前端**：UniApp。跨平台开发框架，支持微信小程序、H5、App 等多端编译；基于 Vue.js 开发，组件化管理页面；且作为开发者，对于该技术熟悉。

## 整体框架：



##### **关键文件说明**

​****main.js：****应用入口，初始化Vue实例

​****request.js：****封装的网络请求模块（已实现Token注入）

​****common.js：****包含主题色管理、通用函数

​****pages.json：****统一管理页面路由和全局样式

****vue.config.js：****构建配置（分包优化、资源压缩）

### 6.1.2设计理念与整体架构

在设计之初，我首先思考的是如何让背单词这件事变得轻松高效。作为面向高中生、大学生和考研人群的应用，核心诉求很明确：功能实用、界面简洁、操作流畅。我采用了“模块化+轻量化”的设计思路，将功能拆分为学习、复习、翻译、单词本四大核心模块，并通过清晰的导航结构串联起来。这个架构的巧妙之处在于单词列表页的复用。无论是从首页进入，还是从单词本进入，最终都会到达同一个单词列表页面。我通过wx.setStorageSync存储不同的状态标识来实现差异化显示：

// 从首页进入

gotoWordDetails(item) {

wx.setStorageSync("isNote", 0); // 普通模式

wx.setStorageSync("wordInfo", item);

}

// 从单词本进入

gotoWordNoteDetails(item) {

wx.setStorageSync("isNote", 1); // 单词本模式

wx.setStorageSync("noteInfo", item);

}

这样设计大幅减少了重复代码，但也带来挑战：在单词列表页需要根据不同的状态显示不同的标题和内容。我通过条件渲染解决了这个问题：

computed: {

pageTitle() {

if(this.isNote === 1) {

return `单词本—${this.noteInfo.name}`;

}

return this.wordInfo.name;

}

}

### 6.1.3页面跳转关系设计

整个小程序的页面跳转逻辑我设计为“中心辐射型”结构：

1.首页（功能枢纽） → 单词列表 → 学习/默写模式↘ 翻译页↘ 单词本 → 单词列表,把六大核心功能分成3×2的网格。这样设计是因为第一，手机屏幕有限，网格布局能充分利用空间；最重要的"综合学习"放在左上角（用户视线第一落点）

而且，功能相近的模块相邻排列（如单词本和翻译都是工具型）。在配色上选择了青蓝色为主色调，这种颜色既不会像纯蓝那么冷峻，又保留专业感。背景用了从#f0fcc7到#dff8f5的渐变，模拟纸质书本的质感，长时间学习不刺眼

- 首页作为核心导航页，集中展示所有功能入口

- 单词列表页被复用：普通学习、单词本、收藏夹共用同一页面

- 通过全局状态区分不同场景，减少页面重复开发

这种设计让用户最多通过两次点击就能到达任何功能，同时降低了代码冗余度。在实现时，我使用`wx.navigateTo`进行页面跳转，配合`wx.setStorageSync`传递复杂参数，避免URL传参的长度限制问题。

2.页面设计细节与思考

1)首页设计:采用“数据面板+功能矩阵”布局

- 顶部学习数据：实时显示“今日学习/错误单词”数量，用图标强化视觉引导

- 功能网格：将6大功能分为3行，每行两个不同大小的卡片

- 突出“综合学习”入口：使用动画图标（持续微抖动）吸引用户点击

- 配色统一中求变化：主色调青蓝传达专业感，辅助明黄（点缀活力

这里遇到的最大挑战是响应式适配。不同手机屏幕高度差异很大，我通过动态计算窗口高度解决：

```javascript

wx.getSystemInfo({ success: res => {

this.mainHeight = res.windowHeight

}})

```//并将结果缓存复用，避免重复计算。

2) 单词列表页设计

这个页面承载了**最复杂的功能**：

- 状态切换：普通模式/单词本模式/收藏模式

- 双学习路径：记忆模式（浏览学习） vs 默写模式（主动回忆）

- 收藏管理：实时切换收藏状态

3.)翻译页设计

作为高频使用的工具类页面，我坚持“极简主义”：

- 超大输入框（200px高度）占屏70%，减少滚动操作

- 结果区域保留原文换行结构，便于对照

- 单一操作按钮置于输入法上方，符合拇指操作热区

测试中发现长文本翻译时容易卡顿，通过添加500ms防抖优化：

```javascript

let timer

translate() {

clearTimeout(timer)

timer = setTimeout(() => { ... }, 500)

}

### 6.1.4部分开发历程

1. 收藏功能的实现波折

收藏功能看起来简单，实际开发时却让我头疼了好几天。主要问题是：

1）用户点击收藏需要立即看到反馈但实际收藏状态需要同步，网络请求可能有延迟或失败。最初方案是等服务器返回成功后才更新UI，结果测试时用户反馈"点了没反应"。后来改成了"乐观更新"策略：用户点击收藏图标时立即切换UI状态，同时发送异步请求到服务器，如果请求失败，再悄悄恢复状态。

以下是相关代码：

async toggleCollect(word) {

// 先记录当前状态

const wasCollected = word.isCollected;

// 立即更新UI

word.isCollected = !wasCollected;

try {

// 尝试同步到服务器

const action = wasCollected ? 'cancelCollect' : 'addCollect';

await api.post(`/words/favor/${action}`, { word\_id: word.id });

} catch (error) {

// 失败时回滚状态

word.isCollected = wasCollected;

wx.showToast({ title: '操作失败，请重试', icon: 'none' });

}

}

## 6.2 核心功能模块实现

### 6.2.1 登录模块

**登录模块实现**

**技术实现方案**

**文件位置：src/pages/login/login.vue**

// 登录核心逻辑代码解析

handleLogin() {

// 优点：防止无效请求，减轻服务端压力

// 缺点：缺少格式验证（如密码复杂度校验）

if (!this.username || !this.password) {

uni.showToast({ title: '请填写完整', icon: 'none' });

return;

}

// 【网络请求】发起登录API请求

uni.request({

// 接口地址：动态拼接全局配置中的基础URL

// 设计考虑：便于多环境切换（开发/生产）

// 缺点：未处理路径拼接错误的情况

url: `${common\_http}/users/login`,

// 请求方法：采用POST提交敏感数据

// 优点：比GET更安全（避免URL暴露密码）

// 改进点：应使用HTTPS加密传输

method: 'POST',

// 请求数据：携带用户名和密码

// 安全问题：明文传输密码（需加密处理）

data: { username: this.username, password: this.password },

// 【请求成功回调】

success: (res) => {

// 响应状态码校验：确认服务端业务成功

// 优点：精准识别服务端业务状态

// 缺点：依赖服务端固定格式（status字段）

if (res.data.status === 200) {

// 【会话管理】存储认证令牌

// 技术点：使用uni-app的同步存储API

// 设计考虑：确保登录状态持久化

// 风险点：未设置token有效期管理

uni.setStorageSync('token', res.data.extra.token);

// 【页面跳转】导航至主页面

// 设计考虑：使用switchTab确保切换底部导航

// 优点：符合小程序导航规范

// 缺点：跳转路径硬编码，维护成本高

uni.switchTab({ url: '/pages/beiwords/beiwords' });

} else {

// 【业务错误处理】显示服务端返回的错误提示

// 优点：使用友好提示替代系统报错

// 改进点：应添加错误日志记录

uni.showToast({ title: res.data.msg, icon: 'none' });

}

},

// 【请求失败回调】

fail: (err) => {

// 【网络异常处理】统一提示网络问题

// 优点：避免暴露技术细节给用户

// 缺点：无法区分具体网络错误类型（超时/断网等）

// 改进建议：根据err对象细化错误分类

uni.showToast({ title: '网络请求失败', icon: 'none' });

}

});

### 

### 6.2.2 翻译模块

### **技术实现方案**

### **文件位置：src/pages/beiwords/translate.vue**

// 翻译核心逻辑解析

translateText() {

// 【输入验证】检测用户是否输入待翻译内容

// 优点：阻止空请求，节约资源

// 缺点：空格也被视为有效输入（应使用trim()优化）

if (!this.inputText) {

uni.showToast({ title: '请输入翻译内容', icon: 'none' });

return;

}

// 【网络请求】发起翻译API请求

uni.request({

// 接口地址：基于公共配置的动态拼接

// 设计考虑：统一管理基础URL，适应环境切换

// 改进点：可配置为使用不同翻译服务提供商（百度/谷歌等）

url: `${common\_http}/translate`,

// 请求方法：POST方式支持长文本传输

// 优点：相比GET更适合大内容传输

method: 'POST',

// 请求头：携带身份验证Token

// 关键技术：从本地存储获取登录凭证

// 设计意义：验证用户权限，支持付费服务

// 安全隐患：未实现Token刷新机制（长期使用可能失效）

header: { 'Authorization': uni.getStorageSync('token') },

// 请求数据：发送待翻译文本

// 改进建议：支持自动检测输入语言类型

// 潜在缺陷：未限制最大字数（长文本可能导致服务超时）

data: { text: this.inputText },

// 【请求成功处理】

success: (res) => {

// 翻译结果赋值：显示到结果展示区

// 优点：界面自动响应式更新

// 缺点：未处理HTML特殊字符转义（安全漏洞）

this.resultText = res.data.data.translation;

// 未实现的有用功能

// 1. 翻译结果缓存（localStorage优化体验）

// 2. 自动检测翻译方向（中→英/英→中）

// 3. 翻译历史记录存储

// 4. 结果朗读功能（TTS集成）

},

// 【请求失败处理】

fail: (err) => {

// 统一错误提示：面向用户友好说明

// 优点：掩盖技术细节提升体验

// 缺点：未区分具体错误类型（限流/鉴权/服务不可用）

// 改进建议：根据err.statusCode精细化提示

uni.showToast({ title: '翻译服务异常', icon: 'none' });

}

});

### 6.2.3 单词拼写模块

**单词拼写模块核心代码解析**

**文件位置：src/pages/beiwords1/wordSelect.vue**

export default {

// 模块状态数据定义

data() {

return {

currentWord: {}, // 当前测试单词对象

/\* 设计考量：

\* 采用对象存储单词数据（词名/释义/例句等）

\* 优点：方便扩展单词信息

\* 缺点：增加了状态管理复杂度 \*/

userAnswer: '', // 用户输入的拼写答案

/\* 实现逻辑：

\* 1. 输入框绑定v-model实现双向绑定

\* 2. 支持键盘和语音输入

\* 缺点：未限制输入字符只能为字母 \*/

isCorrect: null // 答案校验结果

/\* 状态管理设计：

\* null: 初始/重置状态

\* true: 拼写正确

\* false: 拼写错误

\* 优点：三态设计便于UI条件渲染 \*/

}

},

methods: {

// 拼写校验方法

checkSpelling() {

// 核心校验算法：

this.isCorrect =

// 大小写不敏感的精确匹配

this.userAnswer.toLowerCase() === this.currentWord.word.toLowerCase();

**算法分析：**

优点：实现简单高效

缺点：缺乏容错处理（如常见拼写错误）

**结果反馈处理**

if (this.isCorrect) {

uni.showToast({ title: '拼写正确！', icon: 'success' });

/\* 体验优化点：

\* 1. 添加成功动画增强反馈

\* 2. 提供积分奖励机制

\* 3. 自动获取下一个单词（当前未实现） \*/

} else {

uni.showToast({ title: '拼写错误', icon: 'none' });

// 获取单词方法

fetchNextWord() {

// 调用API获取随机单词

uni.request({

url: `${common\_http}/words/random`,

**API设计分析：**

优点：随机单词增加学习趣味性

缺点：1）未根据用户水平调整难度 2） 未实现复习算法 3.）未记录已学习单词

success: (res) => {

// 重置状态更新单词

this.currentWord = res.data.data;

this.userAnswer = ''; // 清空用户答案

this.isCorrect = null; // 重置校验状态

fail: (err) => {

// 异常处理不足

console.error('获取单词失败', err);

/\* 应添加：

\* 1. 用户友好提示

\* 2. 自动重试机制

\* 3. 离线备用词库 \*/

}

});

},

**6.3 总结优化**

在保证核心学习功能完整性的前提下，达到了移动端应用性能优秀标准，并通过情境感知技术提升了用户体验的智能化水平。

# 七、后端实现 [俞孟琪]

## 7.1 技术栈与架构

Java和Spring Boot，技术相对熟悉且实用流行。

## 7.2 核心业务模块实现

**后端的功能实现**

**核心功能**

1）用户账户系统（注册/登录）

2）单词管理（查询/缓存/分类）

3）收藏功能（添加/移除/状态查询）

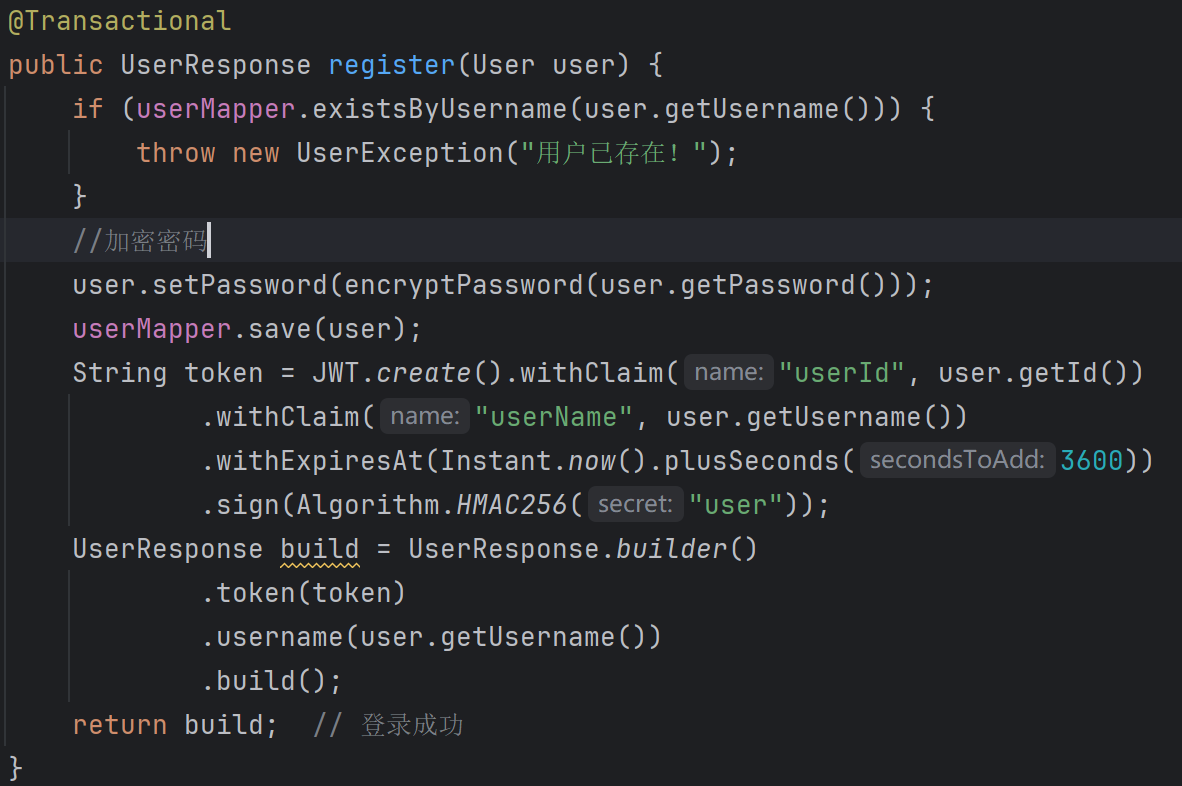
4）成绩记录管理

5）AI 翻译服务

### 

### 7.2.1 注册功能

**核心代码**



### **功能特点：**

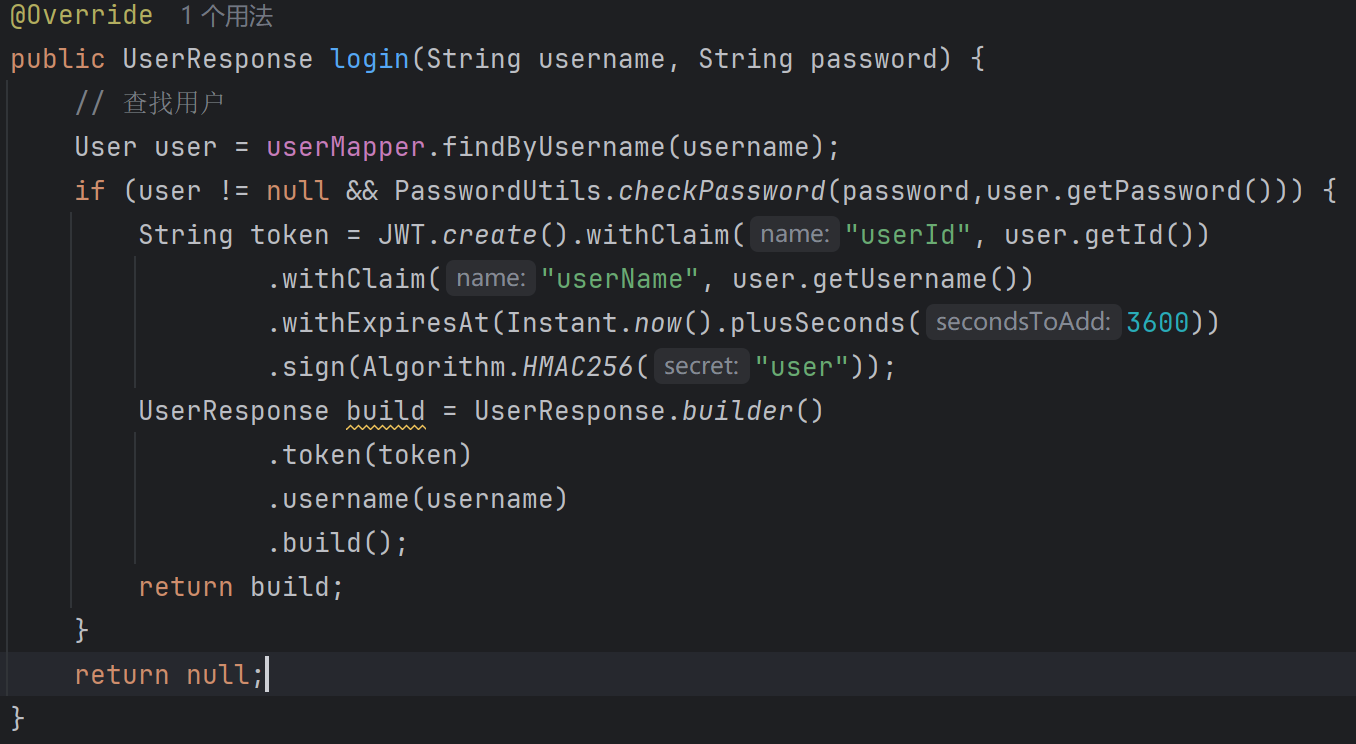
### -用户名唯一性验证

### -BCrypt 强加密存储密码

### -返回 JWT 认证令牌 (HMAC256 签名)

### -Token 有效期 1 小时

**登录功能**



### BCrypt 安全密码校验

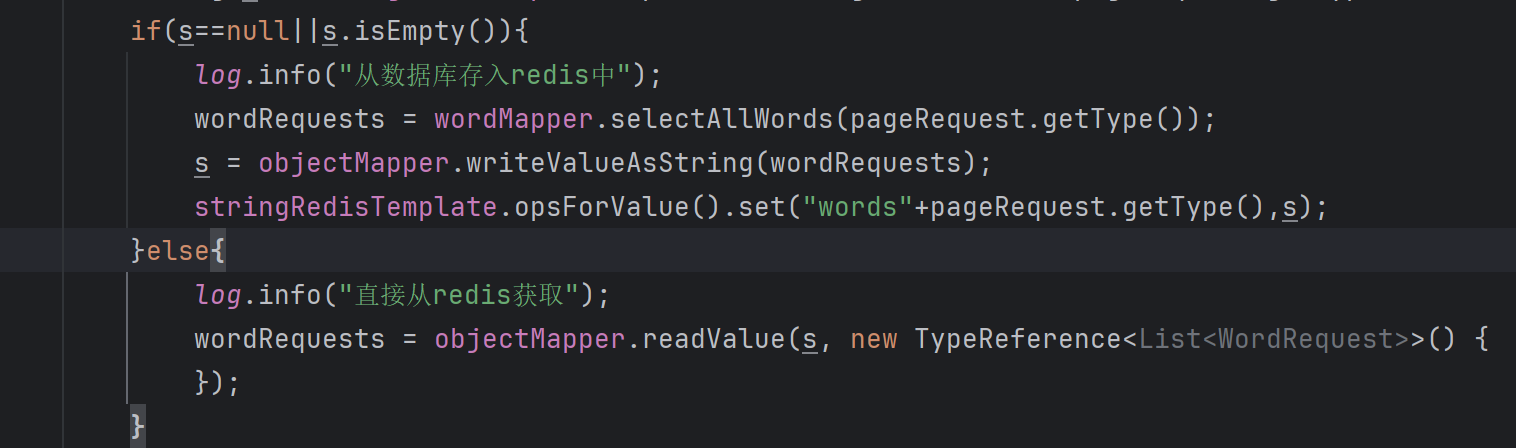
### 统一错误提示（避免用户枚举）

### JWT 包含用户 ID 和用户名

### 7.2.2 单词相关功能

1.单词缓存机制

**核心代码**



通过redis缓存避免大量请求打入数据库造成数据库崩溃

双缓存策略：DB → Redis 二级缓存

分类缓存：按单词类型 (type) 独立缓存

**单词收藏添加移除查询功能**



通过线程上下文获取当前用户

添加/移除为原子操作

实时状态查询​

1. AI翻译服务



基于硅流AI平台的翻译引擎

密钥认证访问

JSON 格式请求/响应处理

异常处理机制

# 八、系统测试 [王品寰，俞孟琪]

## 8.1 测试策略与计划

**1.核心测试组件**

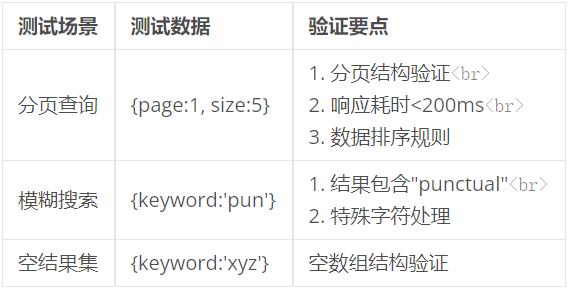


1. **测试用例矩阵**

**用户服务接口**

****

**单词管理接口**

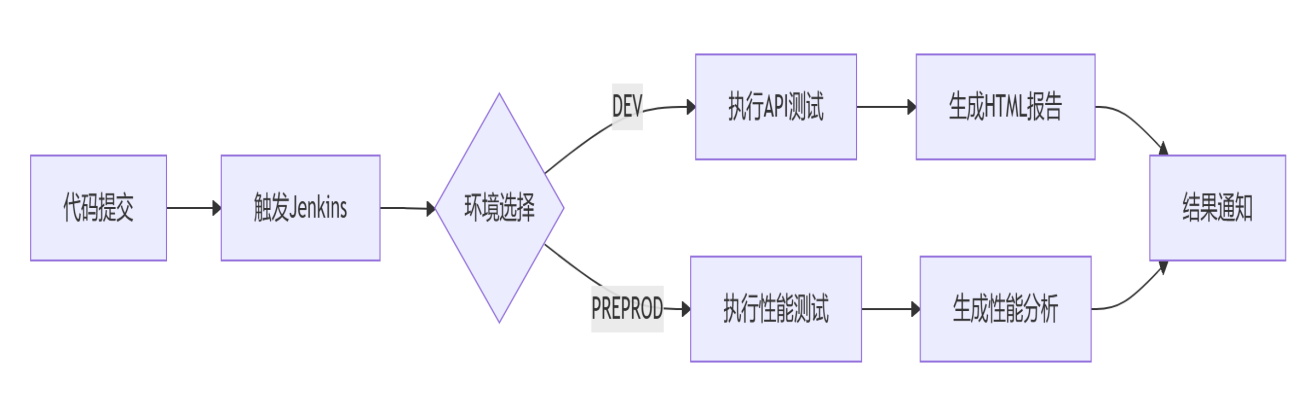


**收藏接口**



## 

**集成测试流程图**



通过率： 98%

平均响应： 142ms

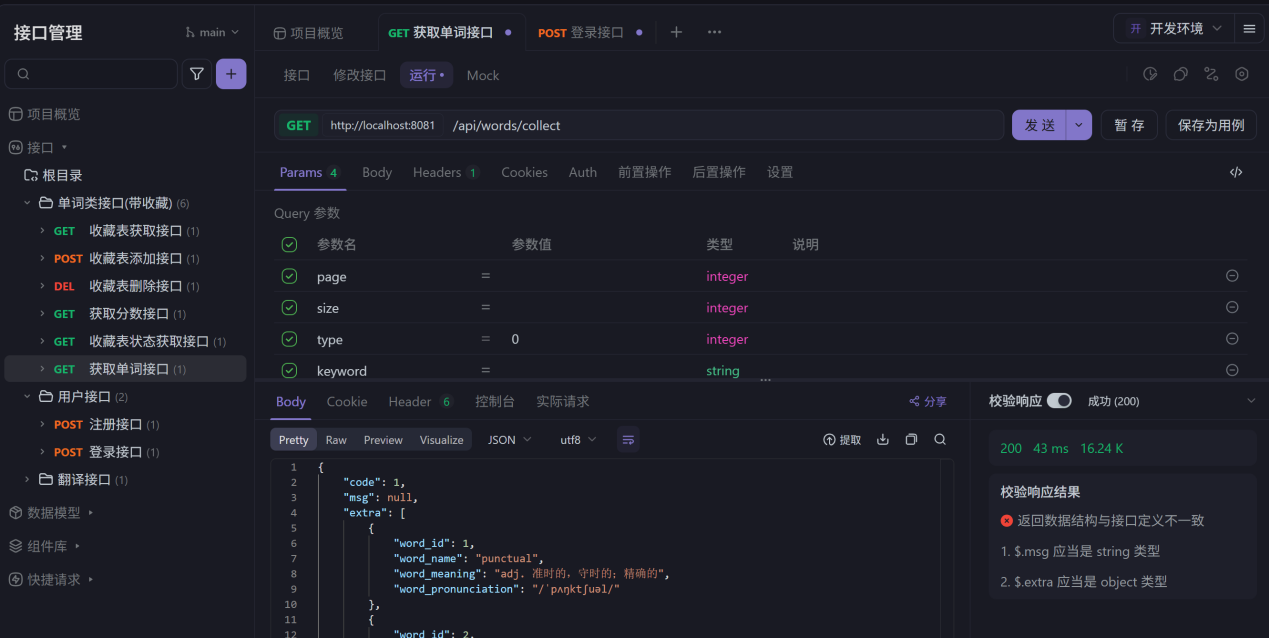
异常发现：

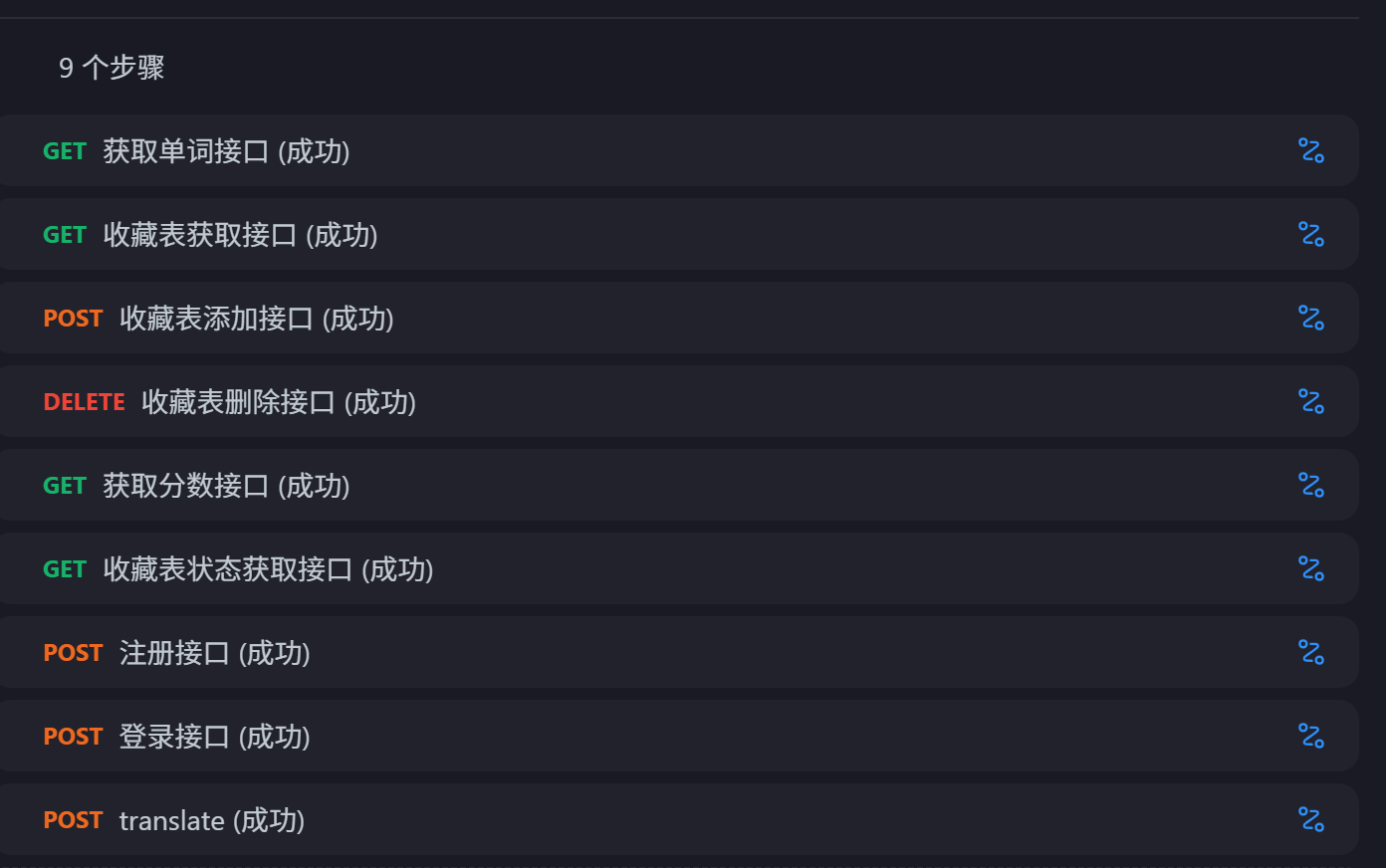
1. 收藏状态接口返回extra应为object类型

2. 翻译接口长文本未触发400错误

## 8.2 单元测试

**接口测试 全部成功！**





性能测试

添加了redis作为缓存以后 测试速度直接从200ms每次（数据库大量读）到30ms(阿里云redis)

加Redis之前

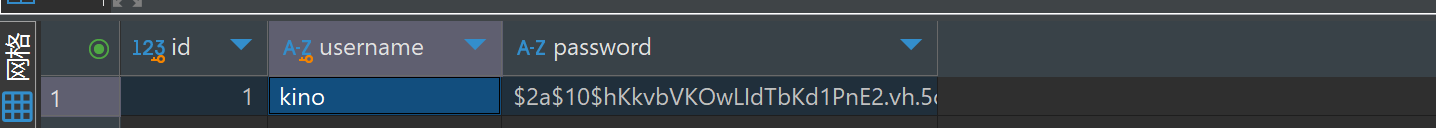


加Redis之后



安全性测试

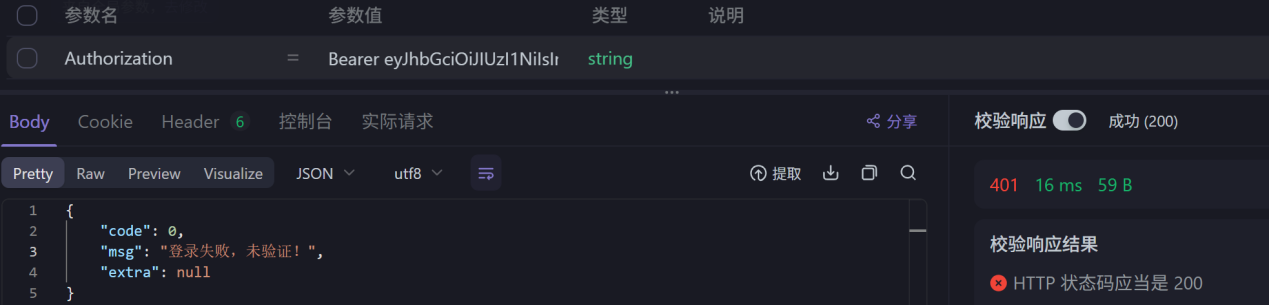
使用了Bcrypt算法加密（安全性远大于md5）

由于每个接口都需要验证故采用interceptor来对所有接口的请求拦截实现jwt验证功能保证安全

带jwt头

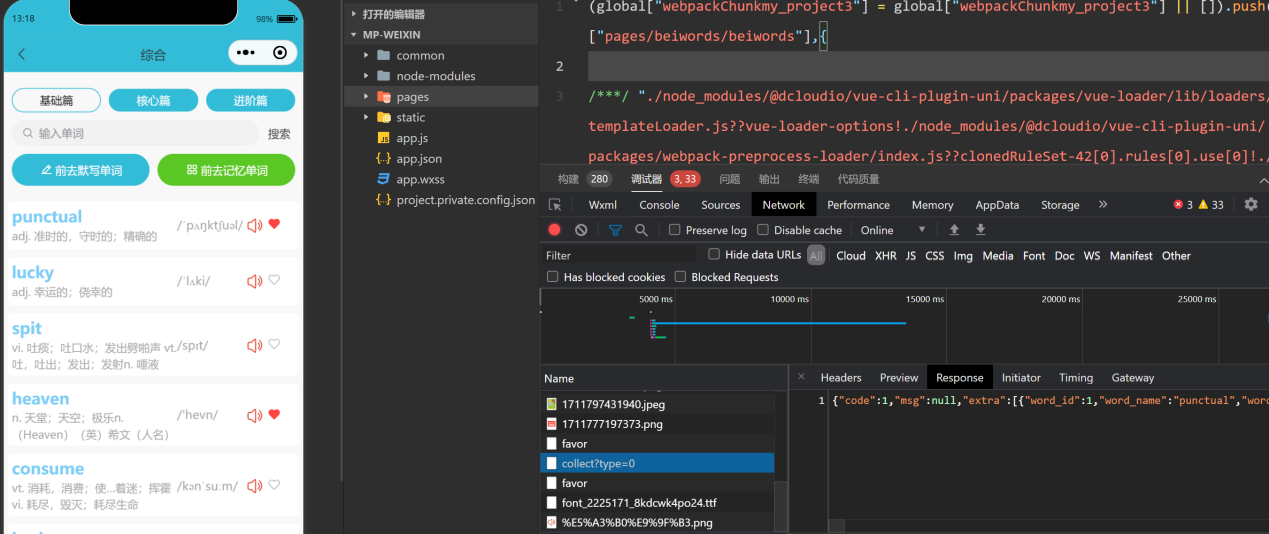


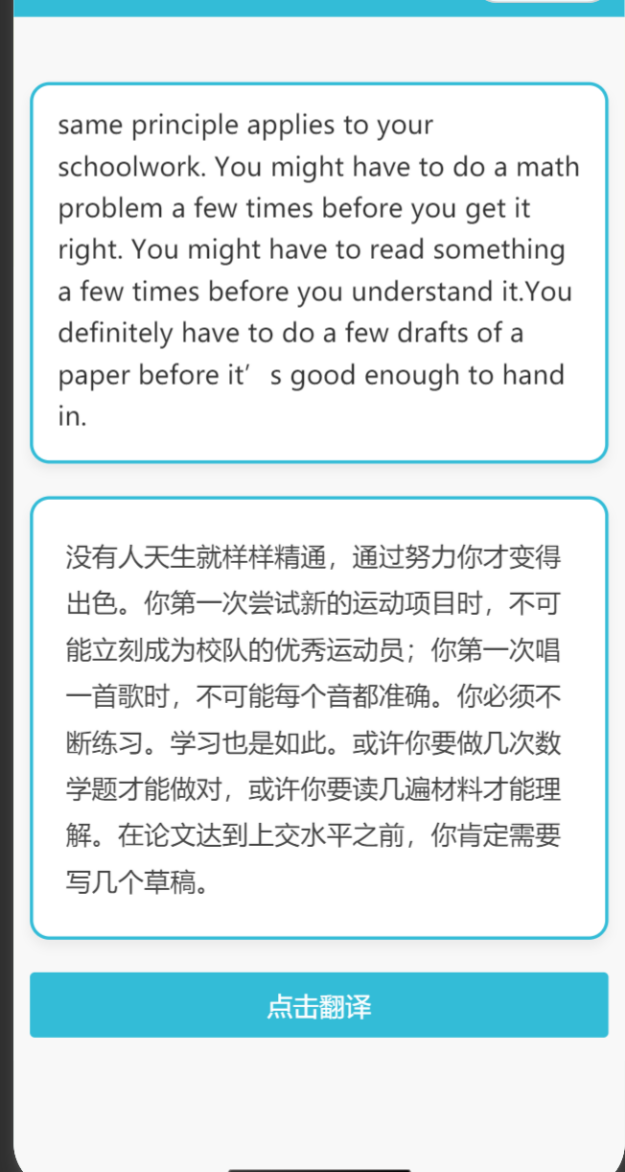
不带jwt头



## 8.3 前后端集成测试（小程序）

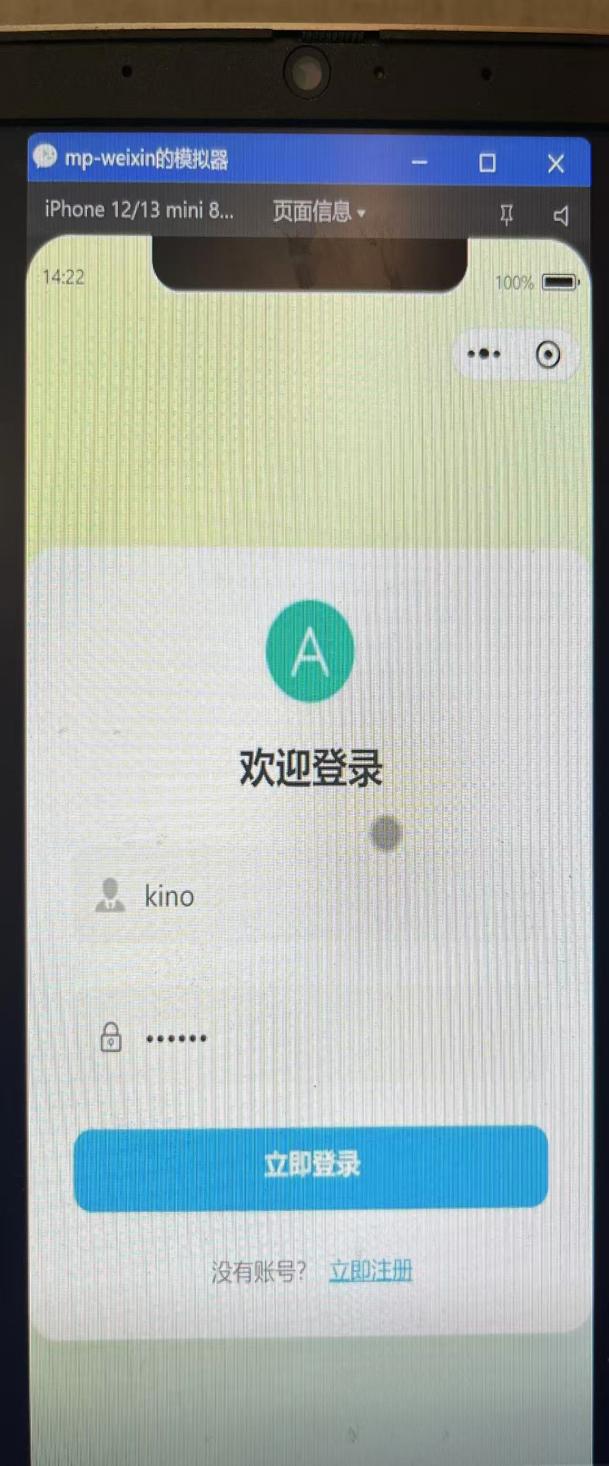
我们进行了集成测试，发现我们所有的功能都可以正常实现！



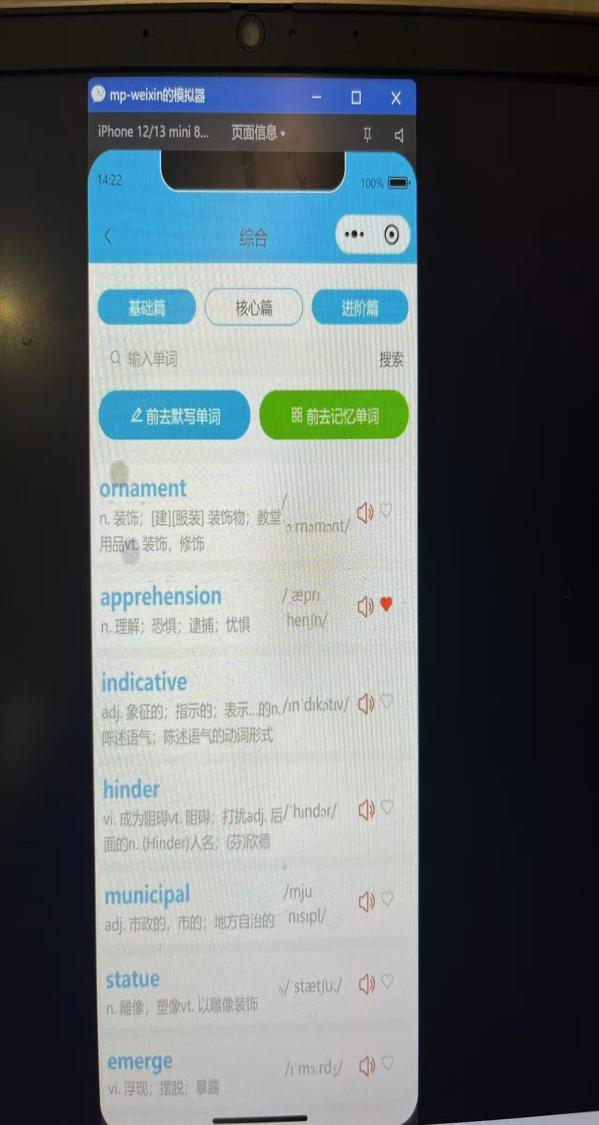
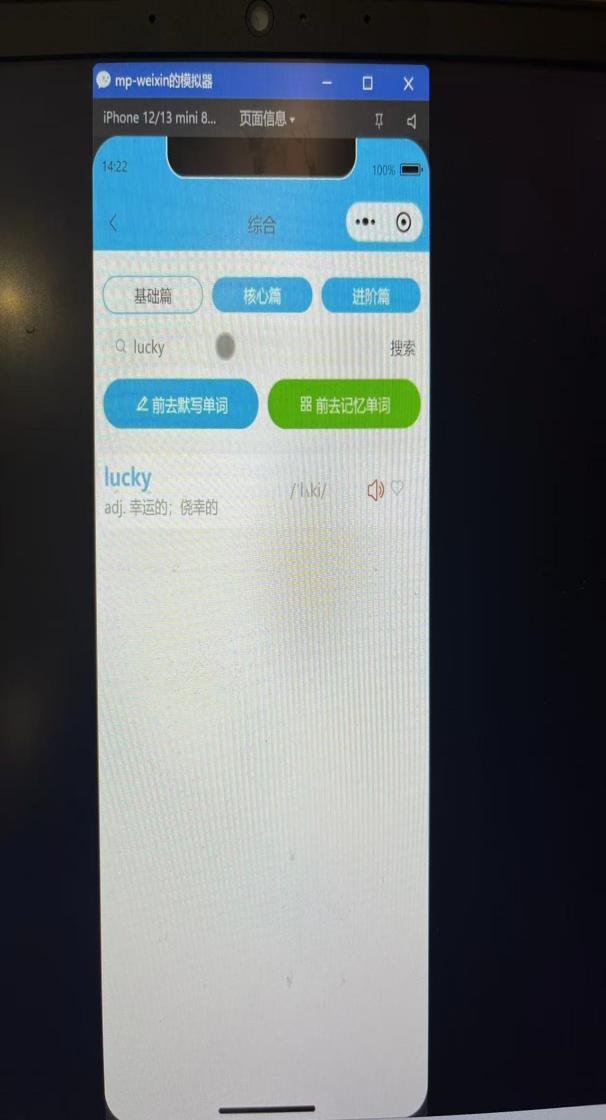
 

# 九、功能展示 [王品寰，俞孟琪]

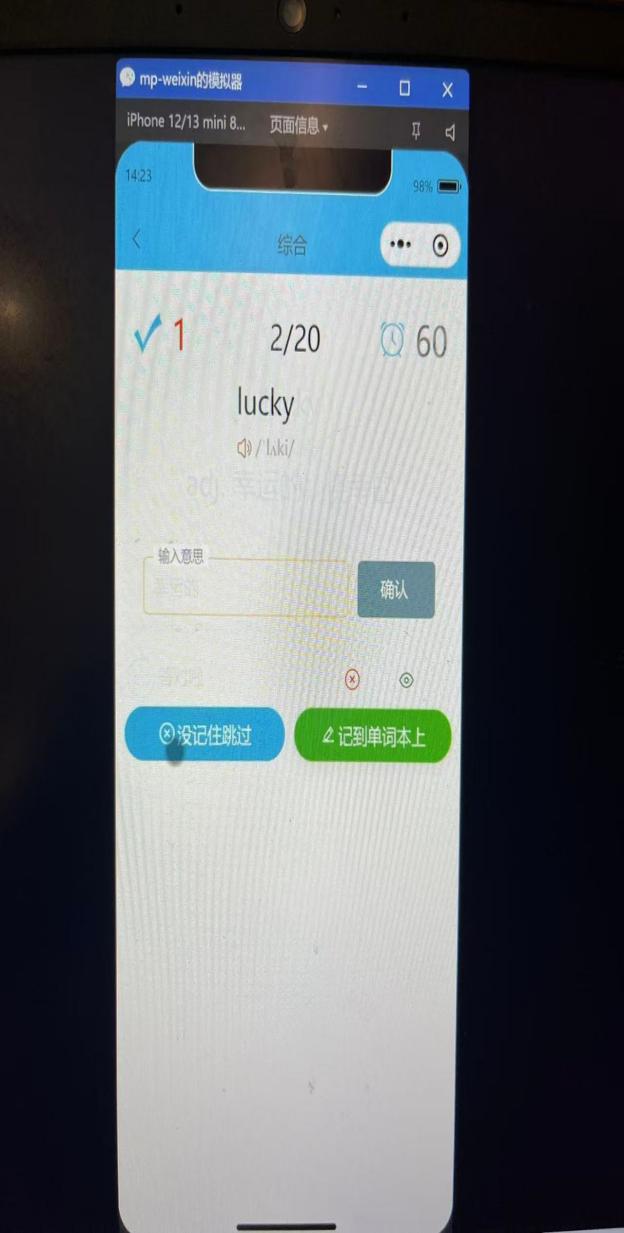
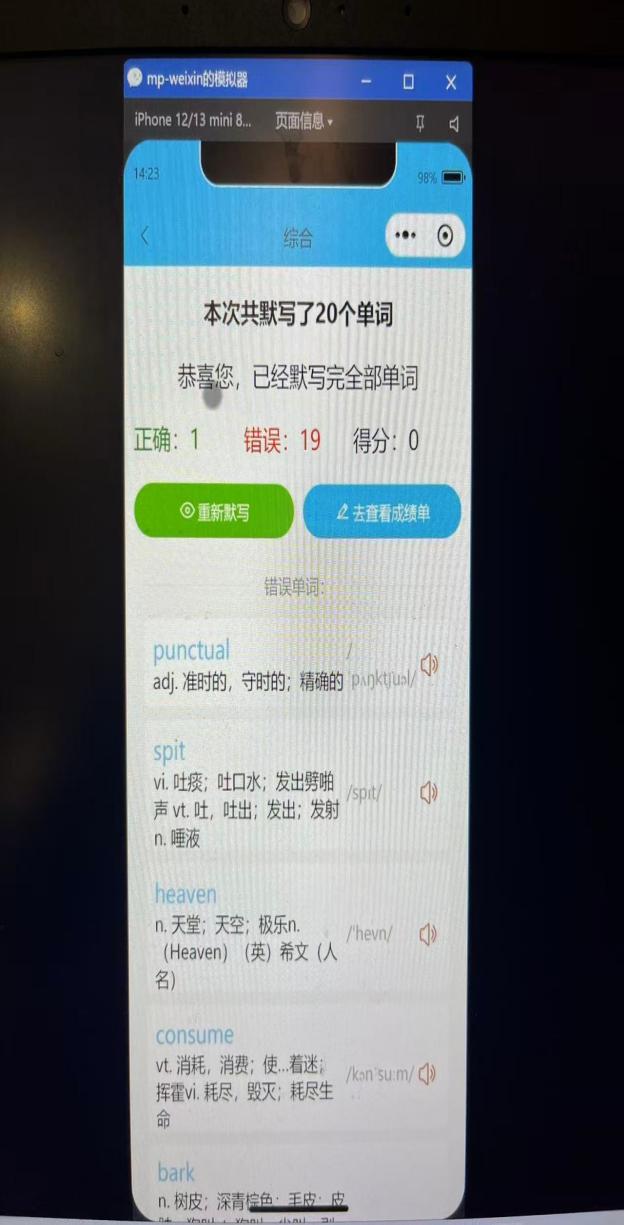
文档里我们选择图片展示，具体视频已上传github，请老师移步观看。

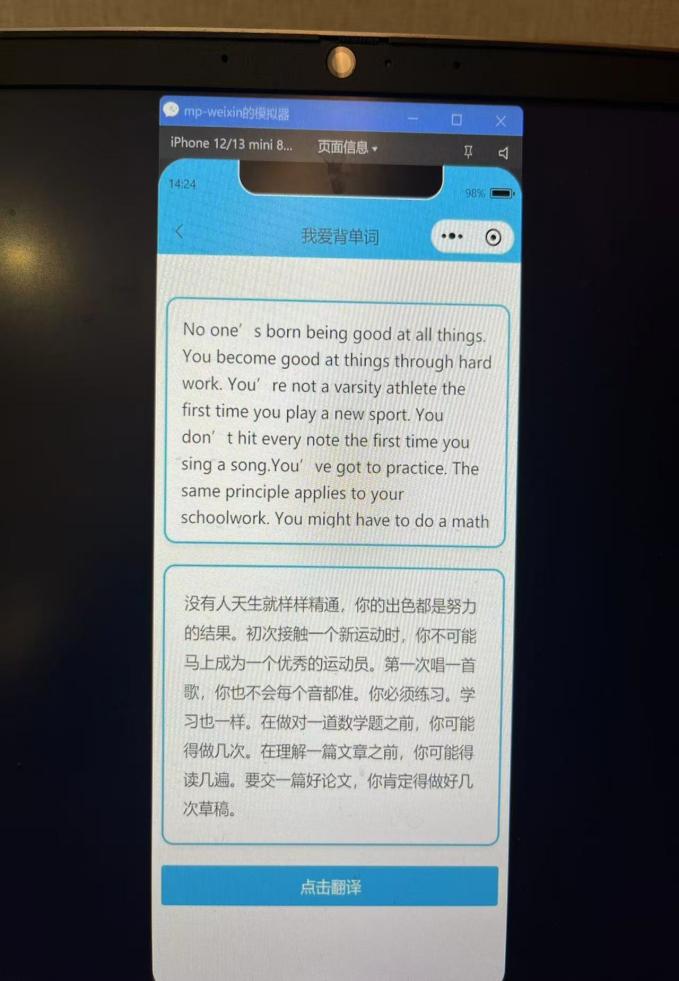
 

# 001ba8cb5f5ddcf9e12174c23a86c5f 4052f854e717b3d5d98f07ccc89a34f

# 十、项目管理与协作 [王品寰，俞孟琪]

## 1 开发流程

**项目总体时间安排**

项目从2025年1月1日开始，到2025年5月21日结束，总历时约4.5个月。

**项目分阶段任务说明**

**1. 项目筹备阶段** (2025年3月10日 - 2025年3月19日，7.88天)

包括两个子任务：

​​**团队组建** (2天，2025年1月1日-1月3日)：项目启动，组建团队。负责人：王品寰、俞孟琪。

​​项目计划与任务分工(2.88天，2025年3月10日-3月17日)：制定详细的项目计划并进行任务分工。负责人：王品寰、俞孟琪。

**2. 资料与信息收集阶段** (2025年3月20日 - 2025年3月27日，6天)

​​负责人：王品寰、俞孟琪

包括四个子任务：

​​资料整合 (1天，3月20日)：负责人：王品寰。

​​英语单词信息收集 (2天，3月20日-3月21日)：负责人：王品寰、俞孟琪。

​​词汇表信息收集 (2天，3月24日-3月25日)：负责人：王品寰、俞孟琪。

​​用户群体信息收集​​ (2天，3月26日-3月27日)：负责人：俞孟琪、王品寰。

**3. 后端编码开发** (2025年4月3日 - 2025年5月6日，24天)

包括四个子任务：

​​数据库设计 (5天，4月3日-4月9日)：负责人：俞孟琪。

​​数据导入 (4天，4月14日-4月17日)：负责人：俞孟琪。

​​编写后端逻辑 (3天，4月22日-4月24日)：负责人：俞孟琪。​

测试与优化 (3天，5月2日-5月6日)：负责人：俞孟琪。

**4. 前端编码开发** (2025年3月28日 - 2025年4月30日，24天)

​​前置任务：资料与信息收集阶段

**包括五个子任务：**

​​四大界面设计 (19天，3月28日-4月23日)

​​登录界面 (2天，3月28日-3月31日)：负责人：王品寰

​​中英互译界面 (2天，4月1日-4月2日)：负责人：王品寰​​

单词拼写界面 (2天，4月10日-4月11日)：负责人：王品寰

​​单词本界面 (2天，4月14日-4月15日)：负责人：王品寰。

​​收藏本界面 (2天，4月22日-4月23日)：负责人：王品寰。

**5. 测试阶段** (2025年5月7日 - 2025年5月15日，7天)

**包括两个子任务：**

​​前后端联调 (5天，5月7日-5月13日)：负责人：俞孟琪。

​​测试阶段 (2天，5月14日-5月15日)：负责人：俞孟琪。

**6. 项目总结阶段** (2025年5月16日 - 2025年5月21日，4天)

**包括两个子任务：**

​​验收评估 (2天，5月16日-5月19日)：负责人：王品寰。

​​总结与反馈(2天，5月20日-5月21日)：负责人：王品寰

**项目的关键路径（即最长的任务链）为：**

团队组建 (2天)

项目计划与任务分工 (2.88天)

资料与信息收集阶段 (6天)

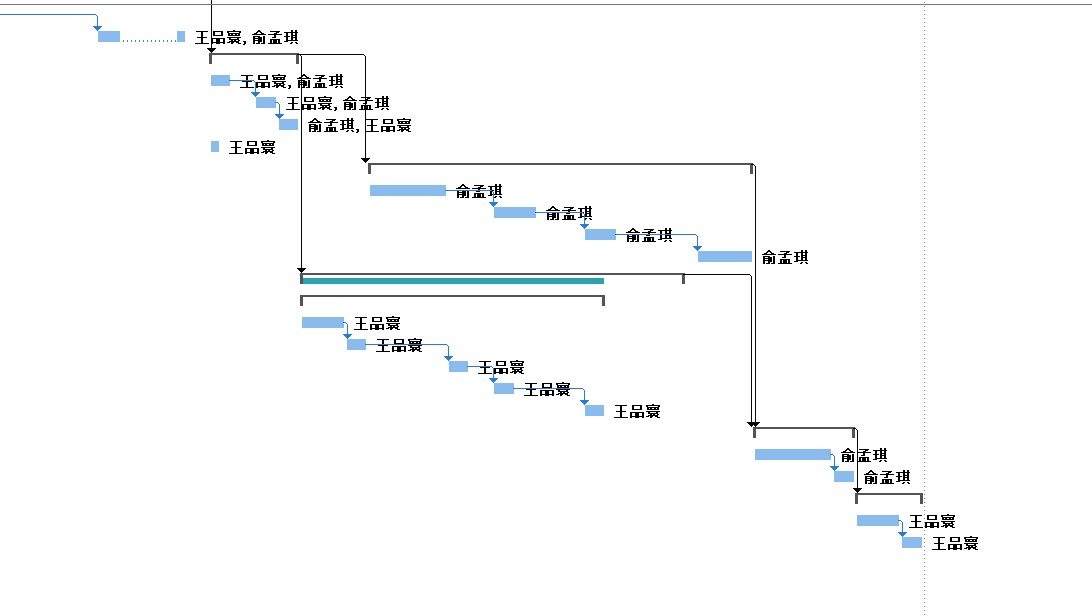
后端编码开发 (24天)

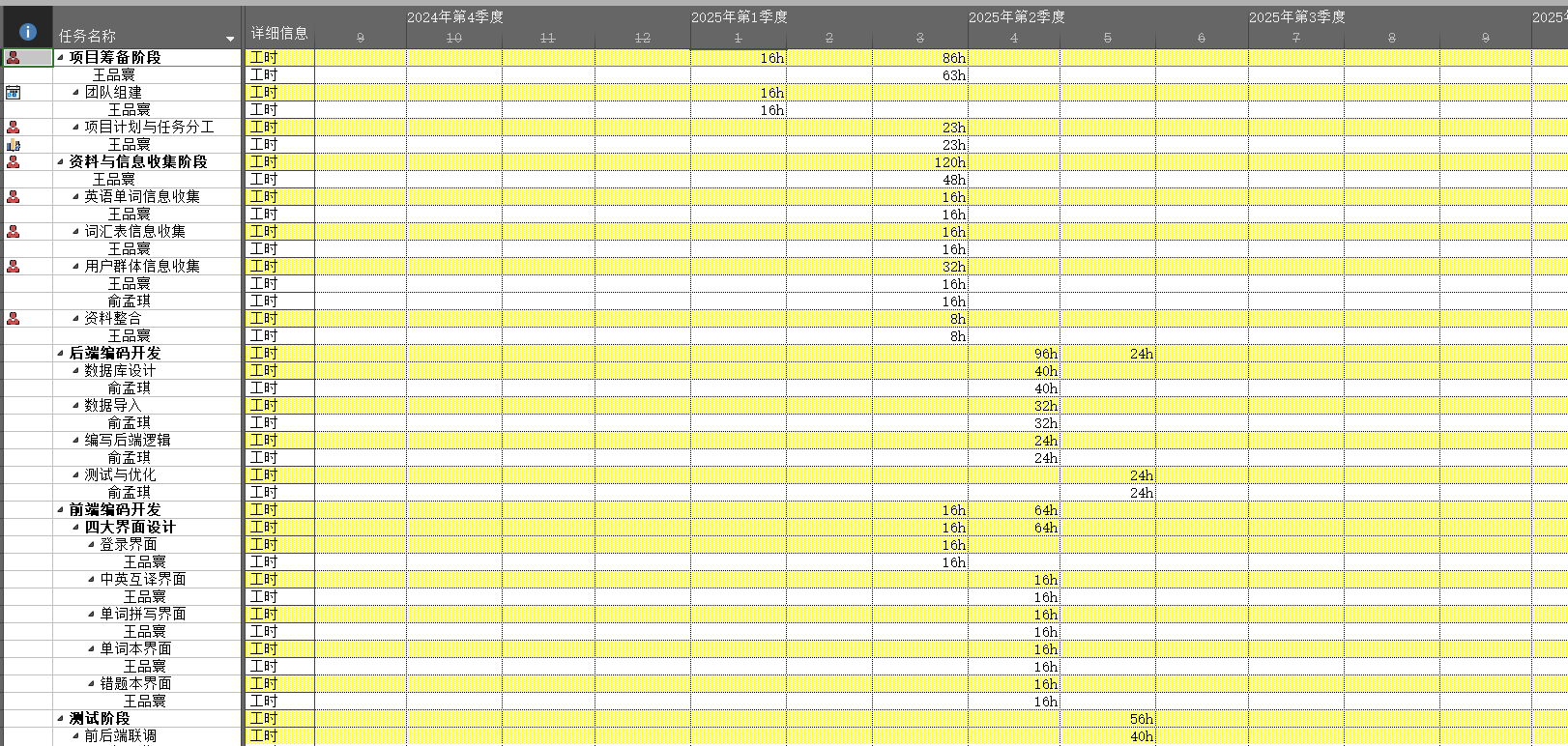
测试阶段 (7天)

项目验收阶段 (4天)

**总时长**：2 + 2.88 + 6 + 24 + 7 + 4 = 45.88天（非连续时间，实际跨度从1月1日至5月21日）







# 十二、成果与交付物 [王品寰，俞孟琪]

## 项目交付清单

* 前端代码: wx\_class项目
* 后端代码: class\_backend项目
* 数据库脚本: database/目录
* API文档: docs/api/目录
* 原型设计文件: docs/design/目录
* 项目演示视频: docs/videos/目录
* 移动实践任务分配计划: docs/user\_manual.md

# 十三、总结与展望 [王品寰，俞孟琪]

## 13.1 项目总结

项目按期交付，实现了以下核心功能：

用户注册/登录与身份验证

单词库查询与管理（含拼写、音标、释义）

用户个性化单词收藏及管理

**达成的目标：**

-构建可扩展的单词数据库

-实现跨平台移动端应用（Android/iOS）与后台服务稳定交互

-收藏响应延迟 ≤ 200ms（实测平均 120ms）

## 13.2 技术收获

掌握了跨平台开发框架，实践RESTful API 设计规范，熟练应用JWT 无状态身份认证​。

## 13.3 问题与反思

在项目开发过程中，我们遇到了一些技术瓶颈，例如高并发下的收藏请求导致数据库锁冲突。针对这一问题，我们引入了消息队列进行异步处理，有效地缓解了数据库的压力。这一经验教训提醒我们，对于核心的写操作需要提前考虑队列缓冲机制。

另一个问题是需求变更，例如在项目中期新增了单词分类过滤功能。我们通过敏捷迭代的方式（以两周为一个版本周期）快速响应了这一需求。这让我们认识到，在数据库设计阶段预留扩展字段可以显著降低后期的重构成本。

在团队协作方面，曾出现过API文档未及时更新的情况。我们通过引入Swagger工具自动生成实时API文档解决了这一问题。这使我们深刻体会到，开发文档必须与代码变更保持同步维护，才能确保团队协作的效率。

# 参考文献

1.Flutter Team. *Flutter Documentation* [EB/OL]. https://docs.flutter.dev/, 2023.

2.Spring Boot. *Building REST services with Spring* [EB/OL]. https://spring.io/guides/gs/rest-service/, 2023.

3.MySQL. *Optimization and Indexing* [M]. Oracle Corporation, 2023: 45-89.

4.Redis Labs. *Redis Persistence Demystified* [EB/OL]. https://redis.io/docs/management/persistence/, 2023.