**武汉大学计算机学院**

**本科生课程实验报告**

**Mido——智能效率工具的设计与实现**

专 业 名 称 ：软件工程

课 程 名 称 ：开源软件生态与开发技术

指 导 教 师 ：桂浩

学 生 姓 名 ：刘子俊

学 号 ：2023302111069

二○二五年十二月

**郑 重 声 明**

本人呈交的设计报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本设计报告不包含他人享有著作权的内容。对本设计报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本设计报告的知识产权归属于培养单位。

目录

[1 项目简介 5](#_Toc216628156)

[1.1项目背景 5](#_Toc216628157)

[1.2功能概述 5](#_Toc216628158)

[1.2.1核心功能 5](#_Toc216628159)

[1.2.2 AI拓展功能 6](#_Toc216628160)

[1.3技术栈综述 6](#_Toc216628161)

[1.4测试计划 7](#_Toc216628162)

[1.4.1单元测试 7](#_Toc216628163)

[1.4.2集成测试 7](#_Toc216628164)

[2 需求分析 7](#_Toc216628165)

[2.1 功能需求 7](#_Toc216628166)

[2.1.1 笔记文件系统 8](#_Toc216628167)

[2.1.2 AI对话系统 9](#_Toc216628168)

[2.2 非功能性需求 9](#_Toc216628169)

[2.2.1 可用性 10](#_Toc216628170)

[2.2.2 性能 10](#_Toc216628171)

[2.2.3 安全性 10](#_Toc216628172)

[2.2.4 可靠性 11](#_Toc216628173)

[2.2.5 可维护性 11](#_Toc216628174)

[3界面设计 12](#_Toc216628175)

[3.1登录界面 12](#_Toc216628176)

[3.2主界面 13](#_Toc216628177)

[4 数据库设计 16](#_Toc216628178)

[4.1数据库设计 16](#_Toc216628179)

[4.2数据库建表 17](#_Toc216628180)

[5 技术实现 21](#_Toc216628181)

[5.1后端架构 21](#_Toc216628182)

[5.1.1主应用入口 21](#_Toc216628183)

[5.1.2模块划分 22](#_Toc216628184)

[5.1.3架构设计模式 28](#_Toc216628185)

[5.1.4请求流程示例 30](#_Toc216628186)

[5.2前端架构 30](#_Toc216628187)

[5.2.1应用入口 32](#_Toc216628188)

[5.2.2路由层 32](#_Toc216628189)

[5.2.3 HTTP请求层 33](#_Toc216628190)

[5.2.4 API层 34](#_Toc216628191)

[5.2.5状态管理 34](#_Toc216628192)

[5.2.6组件管理 35](#_Toc216628193)

[5.2.7视图层 37](#_Toc216628194)

[5.2.8数据流 38](#_Toc216628195)

[6 测试 38](#_Toc216628196)

[6.1后端测试 38](#_Toc216628197)

[6.1.1单元测试 38](#_Toc216628198)

[6.1.2集成测试 41](#_Toc216628199)

[6.1.3安全测试 43](#_Toc216628200)

[6.2前端测试 43](#_Toc216628201)

[6.2.1组件测试 43](#_Toc216628202)

[6.2.2 Store测试 44](#_Toc216628203)

[6.2.3 API请求测试 45](#_Toc216628204)

[6.2.4 端到端测试 45](#_Toc216628205)

[6.3测试工具和框架 45](#_Toc216628206)

[6.4测试结果 46](#_Toc216628207)

[6.4.1 认证服务单元测试 46](#_Toc216628208)

[6.4.2 JWT工具类测试 47](#_Toc216628209)

[6.4.3笔记文件夹服务单元测试 48](#_Toc216628210)

[6.4.4认证控制器集成测试 49](#_Toc216628211)

[7 开发过程与反思 50](#_Toc216628212)

# 1 项目简介

## 1.1项目背景

本项目的目标是开发一款融合 AI 能力的效率型工具。在立项过程中，我参考了自己日常使用较多的 Notion 与 GoodNotes，从两者的产品特点中提炼出核心需求与交互模式。

在个人实际使用各类生产力工具的过程中，我注意到一些产品虽然功能强大，但仍然存在两方面的体验不足：  
　　（1）不够“小而精”——覆盖的使用场景较多，功能模块复杂，导致在记录或整理知识时有一定负担。  
　　（2）AI 融合深度不足——部分AI工具的设计并没有实际提升效率、解决用户使用问题。

基于这些个人使用体验，我希望构建一个更轻量、更聚焦，并且能将 AI 自然融入工作流的 Web 端笔记系统，以改善真实使用过程中的效率痛点。

本项目定位为基于 B/S 架构的企业级应用软件，采用“前端 Web 界面 + 后端服务 + 数据库存储”的模式实现。在功能设计与界面规划上，我重点参考了 Notion 的网页端体验，并结合课程对 Java EE、Spring 等技术体系的要求进行系统性的架构设计。

## 1.2功能概述

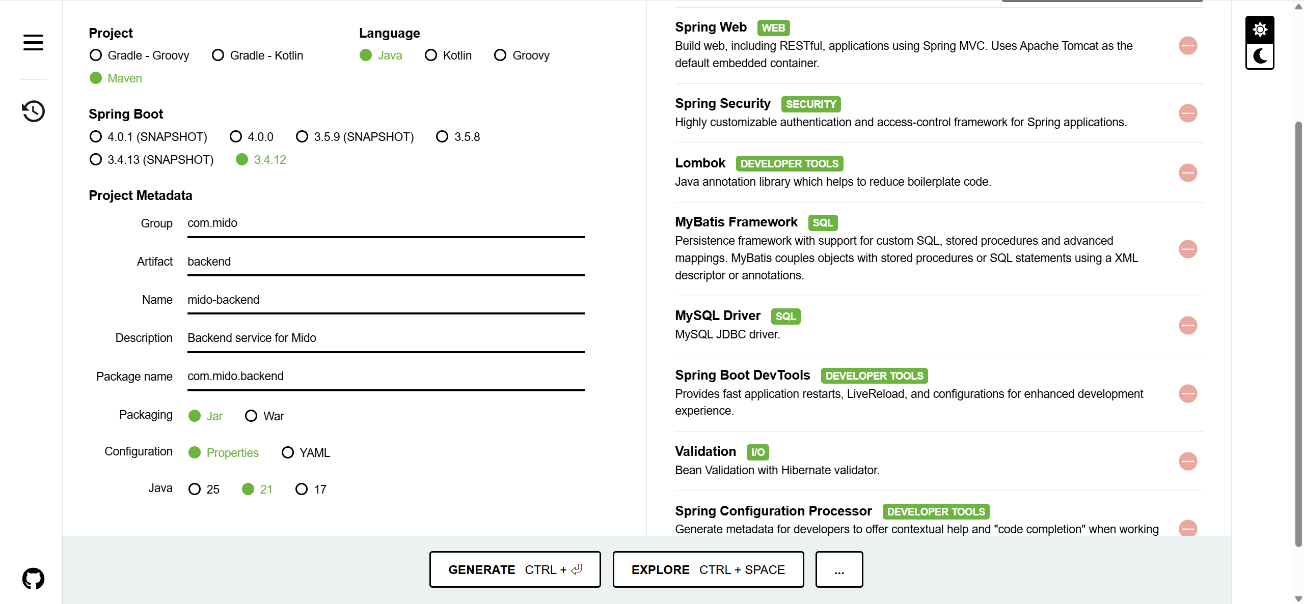
### 1.2.1核心功能

通过问卷形式进行调研，确定开发本系统时所需要的信息，并确定系统开发的功能以及大致实现目标。

### 1.2.2 AI拓展功能

需求分析的任务是深入研究并精确把握用户对软件的需求，包括功能、性能和可靠性等方面。需求分析的目的是提供一套完整、明确且具体的软件需求说明，包括以下三个方面：获取用户需求，深入细致的调研和分析，准确理解用户和项目的具体要求，从而将用户的需求表述转化为完整的系统功能点需求以及确定系统各个模块与其实现的功能点。需求分析具体分为功能性需求、非功能性需求与设计约束三个方面。功能性需求确定软件必须完成的任务和功能，非功能性需求涉及软件性能、可扩展性等，设计约束则需考虑软件设计的约束条件以及软件运行时与其他系统的关系。通过需求分析，能得出一份较为完善、详细的软件功能文档以及工作任务文档。

## 1.3技术栈综述

后端采用 Spring Boot 3.4.12 + MyBatis-Plus，在官网上下载框架：

框架：Spring Boot 3.4.12

数据库：MySQL，JDBC

ORM：MyBatis-Plus 3.5.7

安全：Spring Security + JWT (Auth0)

Java：21

构建：Maven

前端采用：

框架：Vue 3.5.24 (Composition API)

构建工具：Vite 7.2.4

路由：Vue Router 4.6.3

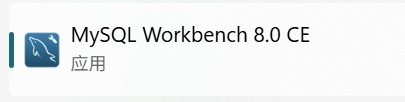
UI 库：Element Plus 2.11.9

HTTP 客户端：Axios 1.13.2

富文本编辑器：Tiptap 2.27.1

Markdown：@kangc/v-md-editor 2.3.18

语言：TypeScript 5.9.3

数据库采用MySQL，本地部署，软件使用MySQL Workbench。

## 1.4测试计划

### 1.4.1单元测试

采用白盒测试法，通过深入分析和测试软件的每个模块，我们能够识别并修复代码中的逻辑错误和功能缺陷，确保每个部分都按照预期精确执行其功能。

### 1.4.2集成测试

结合白盒和黑盒测试策略，该测试专注于分析用例模块间的集成和交互，识别软件设计中的结构性问题、模块调用逻辑以及模块间接口的缺陷。通过输入各种潜在的请求，验证系统是否能够做出恰当的反应。

# 2 需求分析

## 2.1 功能需求

在使用传统 AI 对话工具时，我经常遇到两个问题：

1. **重要对话很难复现**——我记得问过 AI 某个关键内容，但不记得是哪次对话，重新提问也无法得到原来那么好的答案。
2. **对话无法组织管理**——堆积如山的聊天记录无法整理，长期下来几乎没有实际价值。

为解决这些痛点，我设计了 **Mido**——一款融合 AI 能力的笔记与对话管理工具。我的系统包含两个核心部分：

1. 笔记文件系统
2. AI对话系统

两者的重点关系：对话可以像微信中的“聊天记录多选功能”，多选后保存为笔记；笔记、对话间可以相互引用；两个系统都使用树状文件夹分开进行管理。

### 2.1.1 笔记文件系统

目标：提供一个像Notion/Obssidian一样的笔记系统，用来组织用户知识、内容、引用。

具体功能拆解:

1. 树状文件夹管理

可以创建、删除、重命名文件夹

文件与文件夹的层级结构（无限嵌套）

拖拽移动到不同文件/文件夹

左侧树状结构展示（核心 UI）

约束：文件夹下可以放笔记或其他文件夹，删除一个文件夹会删除内部所有内容（回收站）

1. 笔记管理

创建笔记（标题，创建时间，封面设置）

重命名标题

保存/自动保存

Markdown语法（粗体、标题、引用、列表…）

1. 插入内容

用户可以在写笔记的时候插入：表格、图片、公式、代码块、文件引用、对话引用等等，前端渲染，后端只需要存Markdown。

1. 点评/评论

在正文旁边评论（类似 Google Docs 的右侧批注区）

支持：

* 选中文本 → 添加评论
* 显示右侧评论区
* 评论可编辑/删除

### 2.1.2 AI对话系统

目标：提供一个独立的类似 ChatGPT 的聊天系统，同时允许对话保存成笔记文件，并且可以像文件一样管理。同时对话具有一定的记忆功能。

具体功能拆解：

1. 发起对话

点击新建对话，对话后自动生成标题，左侧显示对话列表。

1. 消息系统

用户消息、AI消息、支持Markdown输出。

1. 多选保存（WeChat风格）

用户可以勾选多条消息，点击保存为笔记，可以选择保存的位置，自动保存为一个Note。

1. 对话文件夹管理

和笔记系统一样，但是分开管理。

## 2.2 非功能性需求

本系统为一款融合笔记管理与 AI 对话能力的效率工具，需满足以下非功能性需求，以保证其稳定性、可用性、性能、安全性和可维护性。

### 2.2.1 可用性

1. 系统界面应简洁直观，功能入口清晰：

左侧双 Sidebar：笔记系统 / 对话系统独立展示

新建笔记、新建对话入口固定，易于发现

2. 用户无需学习成本即可完成以下操作：

创建笔记、创建对话、编辑内容、保存为笔记

文件夹管理（创建/删除/拖拽）

激活内置 AI 助手（空格或 “/” 唤出）

3. 所有功能在三步内可达（Minimal Interaction Principle）。

### 2.2.2 性能

1. 用户侧响应时间要求：

普通操作（切换笔记、打开对话）响应时间 ≤ 200ms

搜索操作响应时间 ≤ 500ms

AI 模型响应由模型能力决定，但系统应在 300ms 内完成 API 转发

2. 系统可同时支持 100 个并发用户访问（课堂项目规模）。

3. 数据库应在 10,000 条笔记及对话记录下仍保持正常查询速度。

### 2.2.3 安全性

 用户登录必须经过鉴权（JWT / Session 皆可）。

 所有 API 请求必须验证用户身份与权限。

 用户数据隔离：用户只能访问自己的笔记和对话。

 删除内容不会立即丢失，进入回收站，以防误删。

 数据传输必须使用 HTTPS（若上线）。

### 2.2.4 可靠性

 系统应保证异常退出不会造成笔记内容丢失：

编辑笔记时开启自动保存（间隔不超过 5 秒）

 删除、移动操作必须可撤销（通过回收站实现）。

 AI 服务不可用时，笔记系统仍可独立使用。

### 2.2.5 可维护性

 代码需遵循模块化结构：

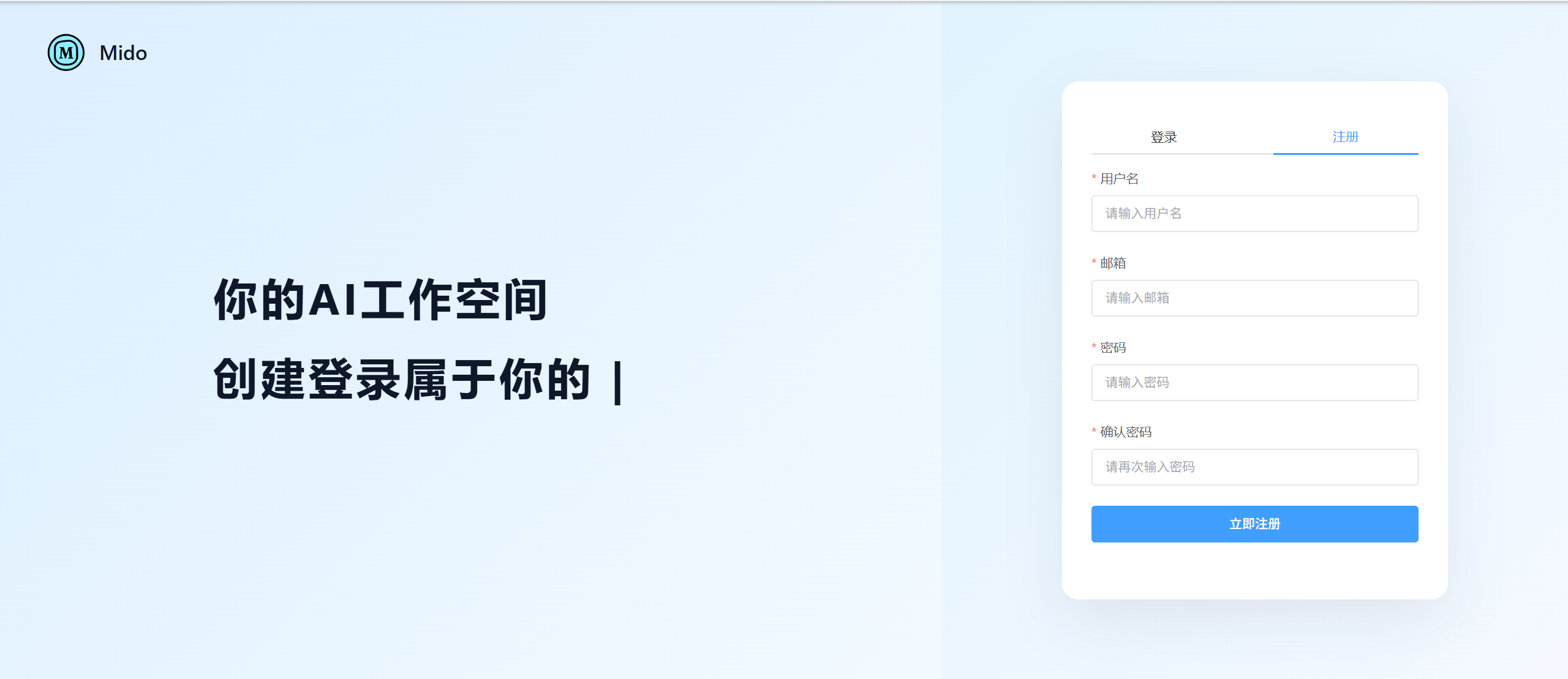
* Notes Module
* Chat Module
* Search Module
* Recycle Bin Module
* AI Model Module

 函数和类命名需遵循统一命名规范（CamelCase / snake\_case）。

 后端所有数据库操作必须经过 Repository 层，禁止直接写 SQL 混进控制器。

# 3界面设计

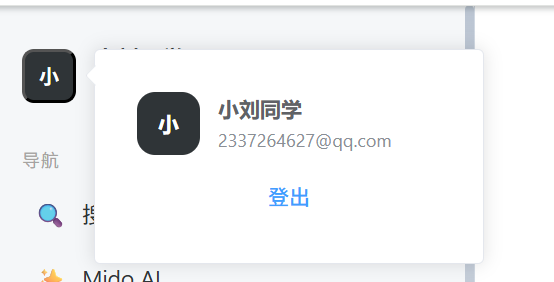
## 3.1登录界面



界面被分为两部分：左侧为动态展示，会像打字机一样展示出如下句子。右侧为登录/注册。同时左上角Mido标志为自己设计。

## 3.2主界面

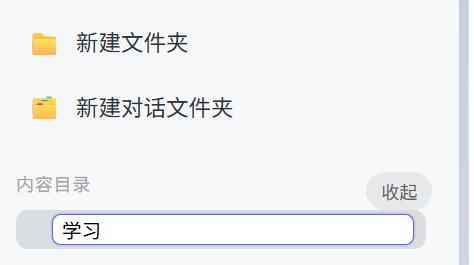
登录后界面如图。这是功能主界面，左上角为用户头像和信息，点击可以出现具体信息，并且支持退出登录。



左侧栏可通过栏内右上角箭头回收，关闭后依旧可通过该按钮展开。并可可以拖拽控制左侧栏的宽度。

导航中有五个功能：

点击新建文件夹可以创建新文件夹，点击即可改名。

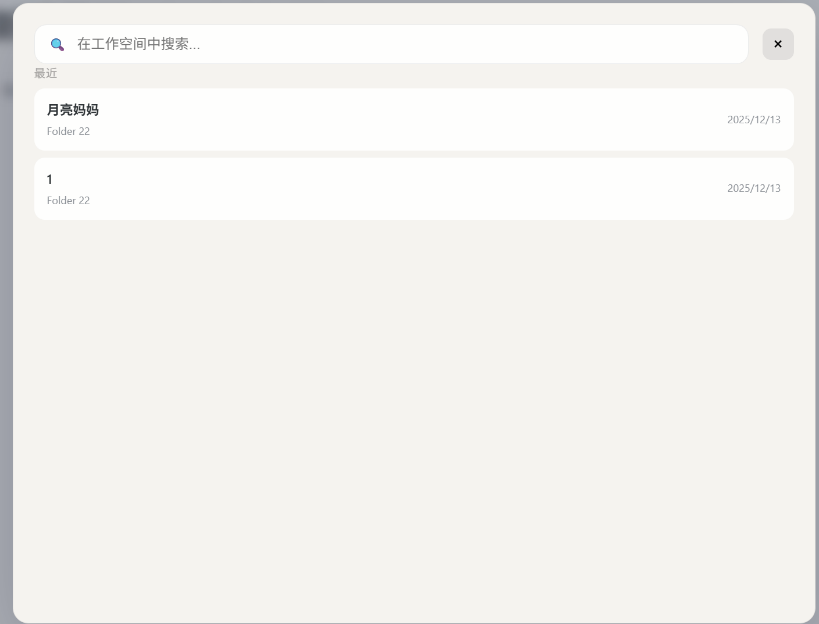


点击新建笔记可以创建，右侧会显示笔记内容，可用两种语法进行编辑，并且可以插入图片。





点击搜索会出现搜索界面，目前仅支持通过标题查找，后续可给出根据时间、内容等查找。

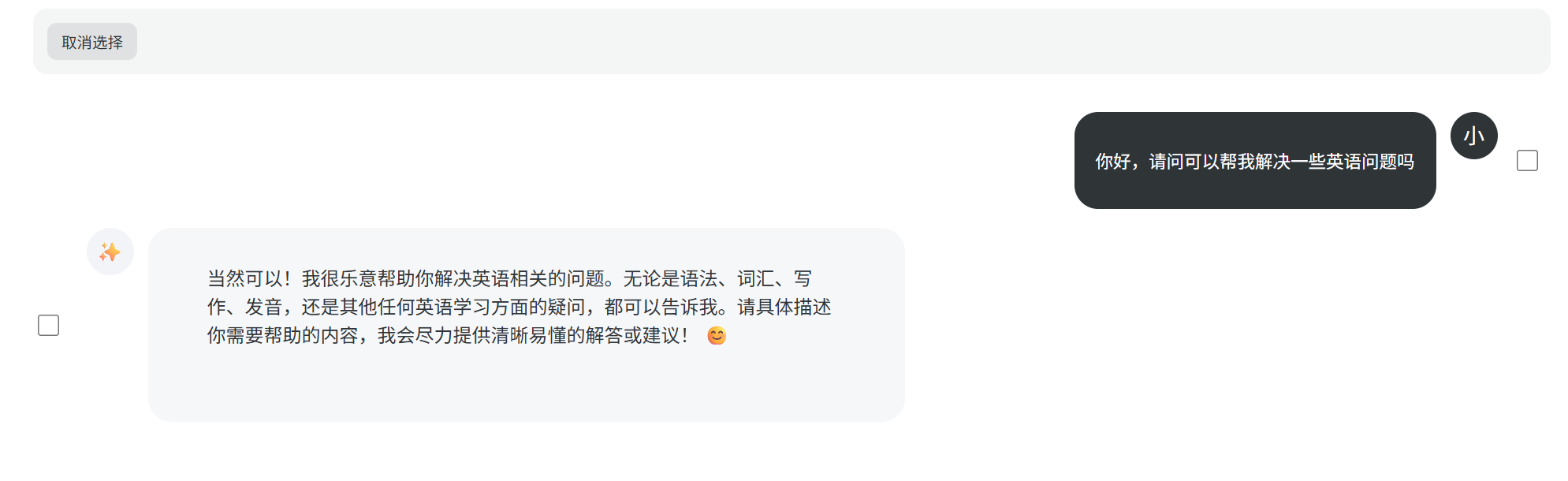
点击Mido AI可以创建新的对话。对话目前API来自deepseek。



点击添加背景信息可以添加文件背景。



选择消息可以导出为文件。





点击新建对话文件夹，更改姓名，把已有的对话拖入。

# 4 数据库设计

## 4.1数据库设计

本系统的数据库设计围绕“以用户为中心的笔记与 AI 对话统一管理”这一核心目标展开。系统需要同时支持结构化数据（如用户信息、文件夹层级关系）与非结构化数据（如笔记正文、对话内容）的高效存储与访问，并保证在多模块协同运行下的数据一致性与可扩展性。因此，在数据库建模阶段，重点关注数据实体的清晰划分、实体间关系的合理约束，以及对后续功能扩展的支持能力。

在整体结构上，数据库以 user 表作为所有业务数据的根实体，所有笔记、文件夹及 AI 对话均通过外键与用户建立关联，从而保证多用户场景下的数据隔离与安全性。围绕用户实体，系统分别设计了笔记子系统与 AI 对话子系统：笔记子系统由笔记文件夹（note\_folder）、笔记（note）及笔记评论（note\_comment）等表构成，用于支持树状目录管理、富文本内容存储以及基于文本锚点的评论功能；AI 对话子系统则通过对话文件夹（chat\_folder）、对话会话（chat）和对话消息（chat\_message）等表实现对多轮对话的持久化存储与层级化管理。

此外，在设计过程中，系统统一采用逻辑删除（is\_deleted）而非物理删除的方式管理数据生命周期，以降低误删风险并为后续的回收站、历史恢复等功能预留扩展空间；同时，通过合理设置外键约束与索引，提高数据一致性与常用查询场景下的访问性能。

## 4.2数据库建表

具体建表语句如下：

-- 1. 用户

CREATE TABLE user (

id BIGINT PRIMARY KEY AUTO\_INCREMENT,

username VARCHAR(50) NOT NULL UNIQUE,

email VARCHAR(100) NOT NULL UNIQUE,

password\_hash VARCHAR(255) NOT NULL,

created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP

);

CREATE INDEX idx\_user\_email ON user(email);

-- 2. 笔记文件夹

CREATE TABLE note\_folder (

id BIGINT PRIMARY KEY AUTO\_INCREMENT,

user\_id BIGINT NOT NULL,

name VARCHAR(100) NOT NULL,

parent\_id BIGINT DEFAULT NULL,

is\_deleted TINYINT(1) DEFAULT 0,

created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES user(id),

FOREIGN KEY (parent\_id) REFERENCES note\_folder(id)

);

CREATE INDEX idx\_note\_folder\_user\_id ON note\_folder(user\_id);

-- 3. 笔记

CREATE TABLE note (

id BIGINT PRIMARY KEY AUTO\_INCREMENT,

user\_id BIGINT NOT NULL,

folder\_id BIGINT NOT NULL,

title VARCHAR(200),

content MEDIUMTEXT,

cover\_url VARCHAR(255),

is\_deleted TINYINT(1) DEFAULT 0,

created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,

updated\_at DATETIME DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES user(id),

FOREIGN KEY (folder\_id) REFERENCES note\_folder(id)

);

CREATE INDEX idx\_note\_user\_id ON note(user\_id);

CREATE INDEX idx\_note\_folder\_id ON note(folder\_id);

-- 4. 笔记评论

CREATE TABLE note\_comment (

id BIGINT PRIMARY KEY AUTO\_INCREMENT,

note\_id BIGINT NOT NULL,

user\_id BIGINT NOT NULL,

anchor\_offset INT,

anchor\_length INT,

content TEXT NOT NULL,

created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (note\_id) REFERENCES note(id),

FOREIGN KEY (user\_id) REFERENCES user(id)

);

CREATE INDEX idx\_note\_comment\_note\_id ON note\_comment(note\_id);

CREATE INDEX idx\_note\_comment\_user\_id ON note\_comment(user\_id);

-- 5. AI 对话会话（一个 chat 文件）

CREATE TABLE chat (

id BIGINT PRIMARY KEY AUTO\_INCREMENT,

user\_id BIGINT NOT NULL,

title VARCHAR(200),

is\_deleted TINYINT(1) DEFAULT 0,

created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,

updated\_at DATETIME DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES user(id)

);

CREATE INDEX idx\_chat\_user\_id ON chat(user\_id);

-- 6. 对话文件夹

CREATE TABLE chat\_folder (

id BIGINT PRIMARY KEY AUTO\_INCREMENT,

user\_id BIGINT NOT NULL,

name VARCHAR(100),

parent\_id BIGINT DEFAULT NULL,

is\_deleted TINYINT(1) DEFAULT 0,

created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES user(id),

FOREIGN KEY (parent\_id) REFERENCES chat\_folder(id)

);

CREATE INDEX idx\_chat\_folder\_user\_id ON chat\_folder(user\_id);

-- 7. 对话消息

CREATE TABLE chat\_message (

id BIGINT PRIMARY KEY AUTO\_INCREMENT,

chat\_id BIGINT NOT NULL,

role ENUM('user', 'assistant') NOT NULL,

content MEDIUMTEXT NOT NULL,

created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (chat\_id) REFERENCES chat(id)

);

CREATE INDEX idx\_chat\_message\_chat\_id ON chat\_message(chat\_id);

# 5 技术实现

## 5.1后端架构

整体采用 Spring Boot 3.4.12 + MyBatis-Plus，分层架构如下：

前端 (Vue.js)

↓ HTTP/REST API

Controller层 (REST接口)

↓

Service层 (业务逻辑)

↓

Mapper层 (数据访问)

↓

Database (MySQL)

接下来本人将逐步介绍核心模块结构。

### 5.1.1主应用入口



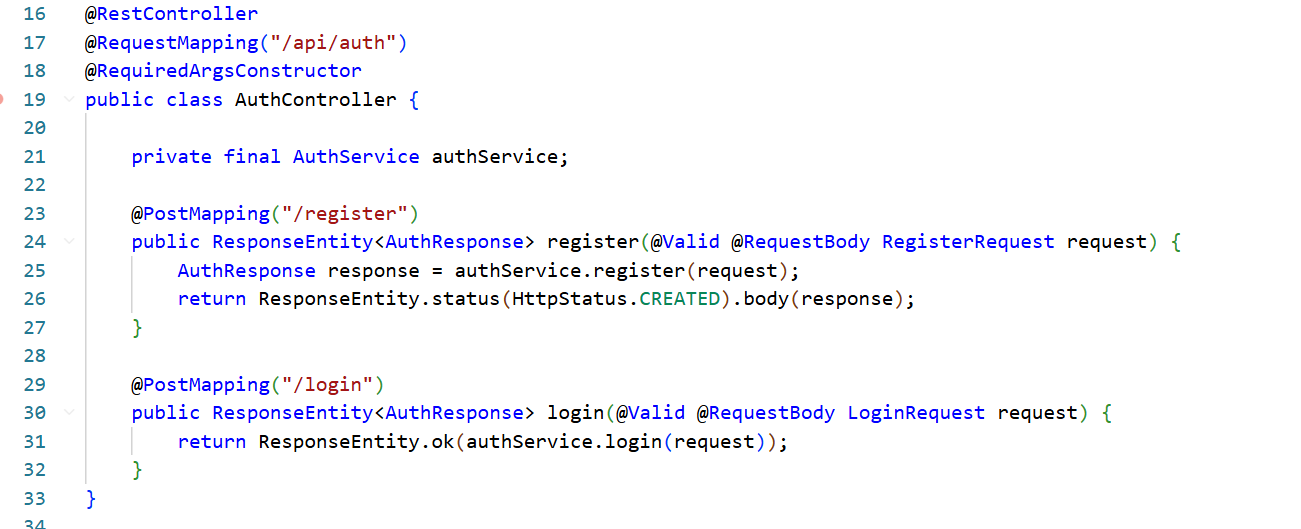
@SpringBootApplication表示这是整个项目的总入口，是三个注解的集合：@SpringBootApplication = @SpringBootConfiguration + @EnableAutoConfiguration + @ComponentScan

@SpringBootConfiguration表示这个类是Spring的配置类，里面可以 @Bean，能被 Spring 当成“总配置入口”。 @EnableAutoConfiguration根据 classpath 里有什么依赖，自动配置Bean。@ComponentScan扫描组件，自动注入Bean。

@MapperScan扫描 MyBatis Mapper，通过@EnableConfigurationProperties启用配置类，这是必须的，因为这些类无@Component，而被当作依赖注入使用。

### 5.1.2模块划分

本项目按照业务域来进行划分：

**auth：**使用Spring Security + JWT，AuthController控制登录和注册，如下图。

AuthService完成认证逻辑。部分代码如下：



**user：**用户模块

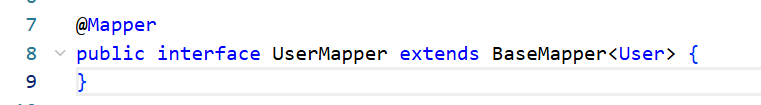
用户实体如下：



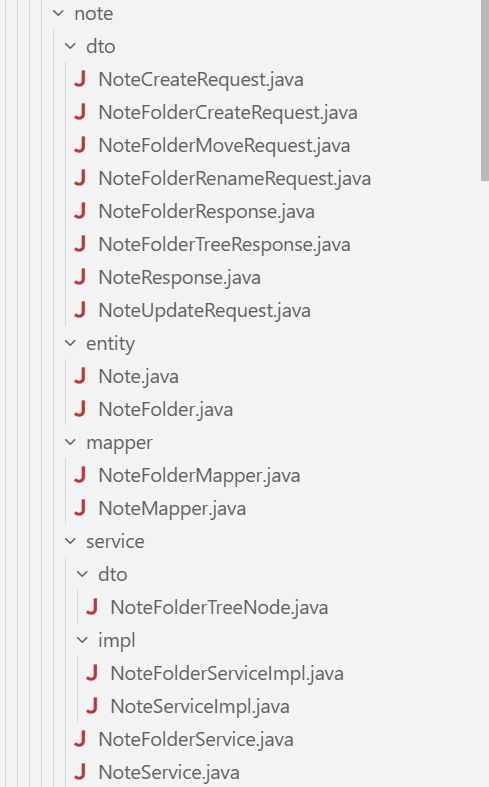
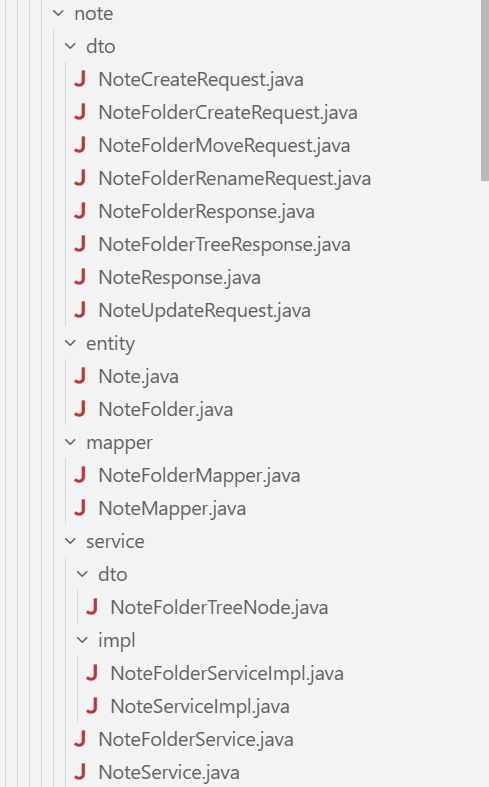
用户业务逻辑如下：



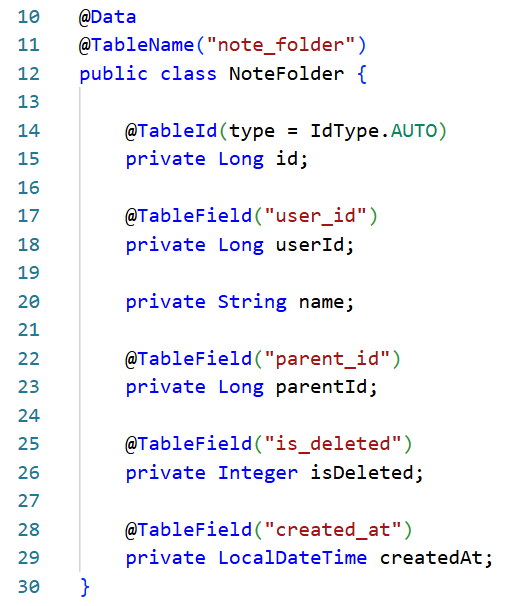
数据访问业务如下：

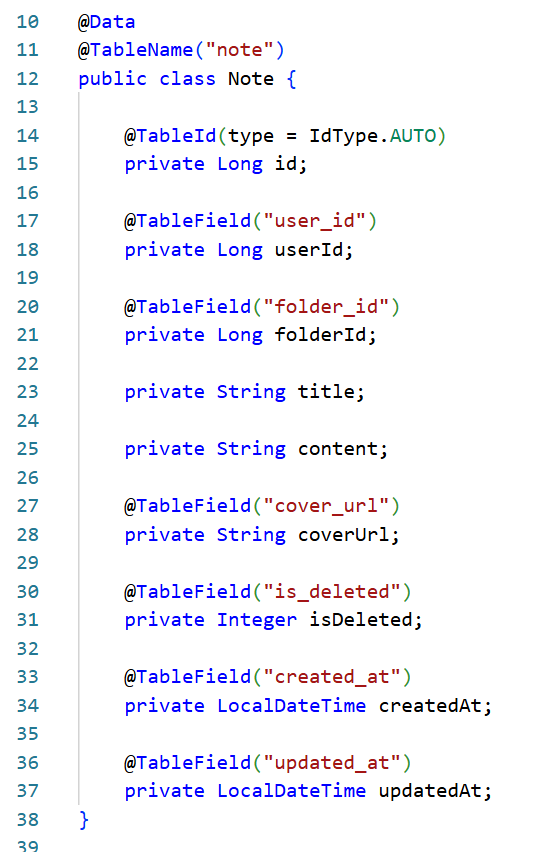


由于mybatis-plus自带的crud，本部分不用多写代码。

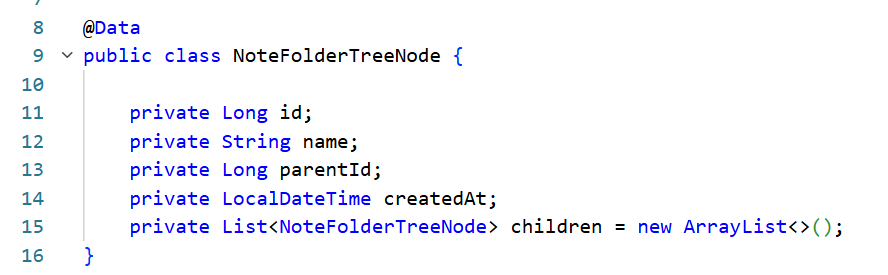
**note:** 笔记系统，较为复杂，先将所有代码文件展示如下

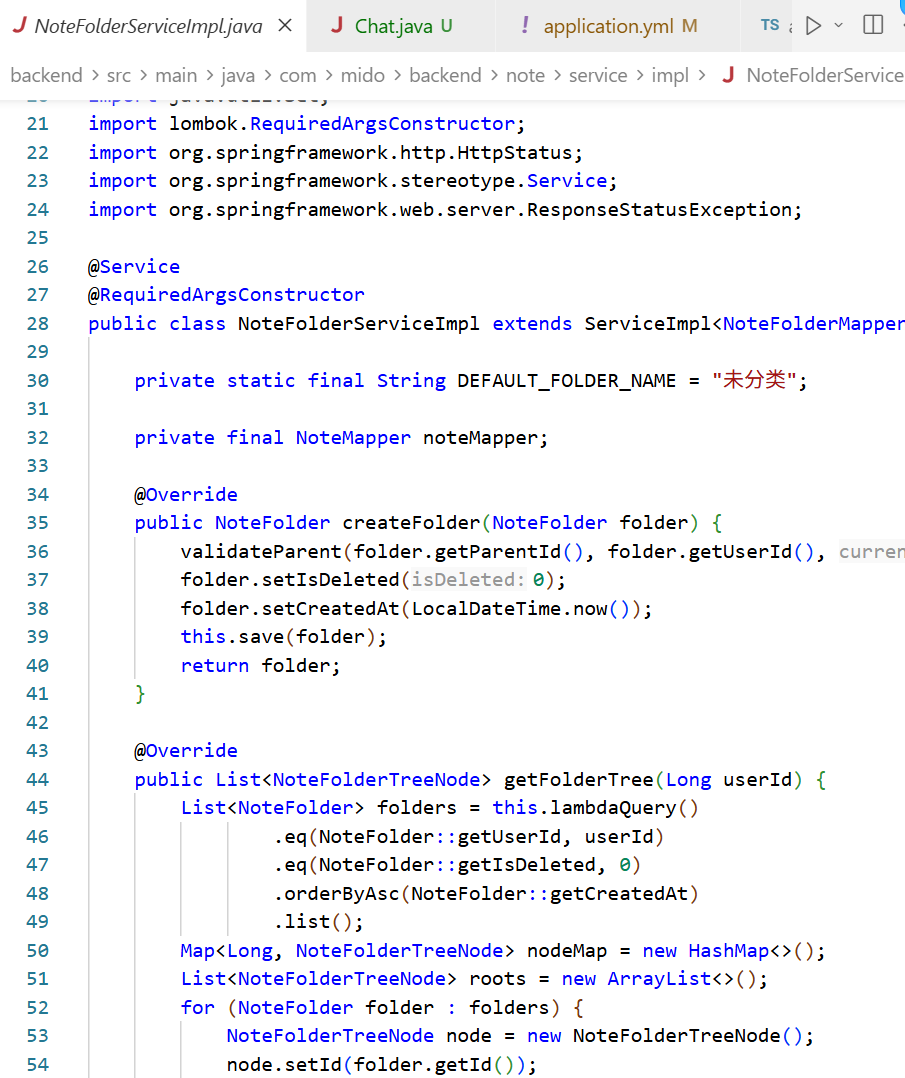


首先是两个实体文件：entity下的Note.java NoteFolder.java，分别如下。

接着是mapper数据访问层，这层由于不涉及到复杂增删改查，也不做赘述。

然后是service业务逻辑层，其中有dto、impl实现类。（展示部分代码）





最后是控制器层的dto（请求/响应），主要是文件名不能为空、文件夹名不能为空等逻辑，都是一些非常简单的逻辑，在这里不做赘述。

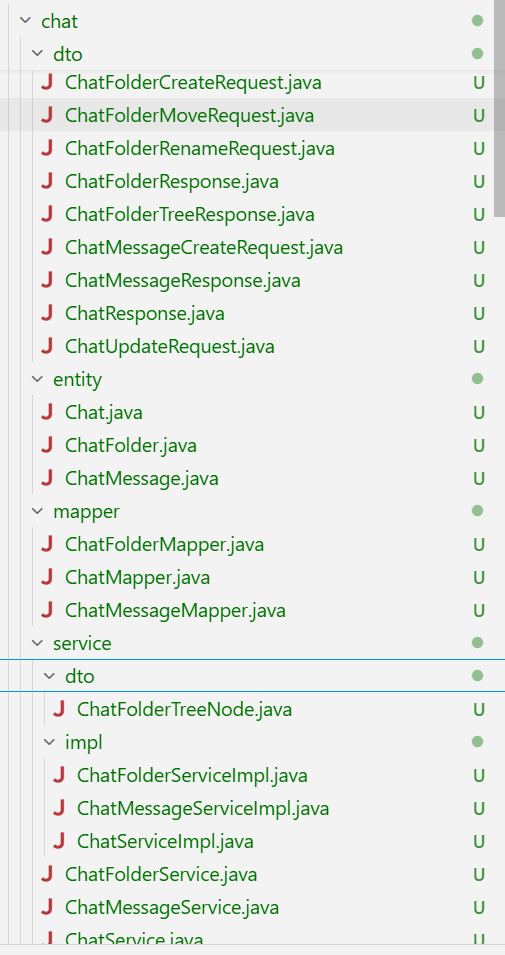
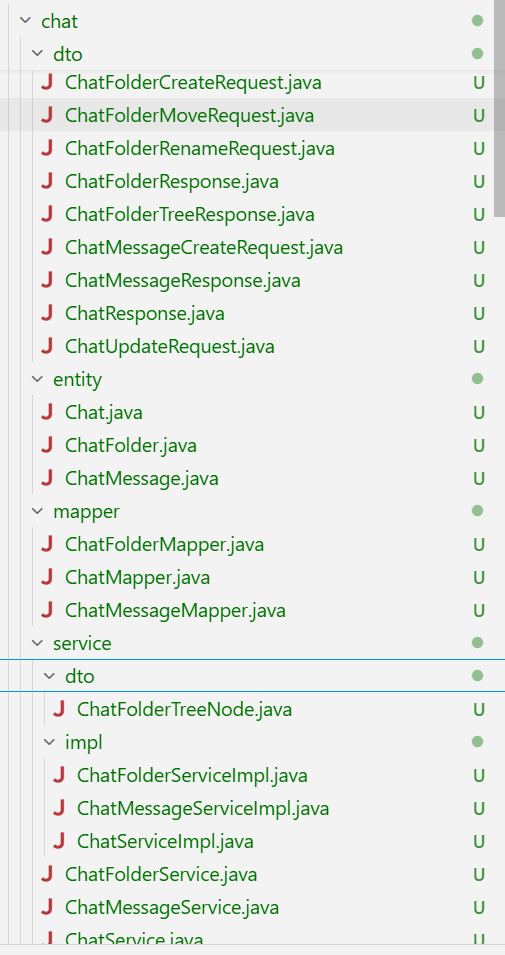
**chat：**对话模块

ChatController、ChatFolderController：对话管理

ChatService、ChatMessageService、ChatFolderService：业务逻辑

Chat、ChatMessage、ChatFolder：实体





本部分实现了类chatgpt的对话系统，完成了：

1. 对话管理：创建、更新、删除、查询对话
2. 消息管理：添加消息、查询消息列表、删除消息
3. 文件夹管理：创建、重命名、移动、删除文件夹（树形结构）
4. 分页查询：按文件夹分页查询对话
5. 权限控制：用户只能操作自己的对话和文件夹
6. 级联操作：删除文件夹时级联删除对话

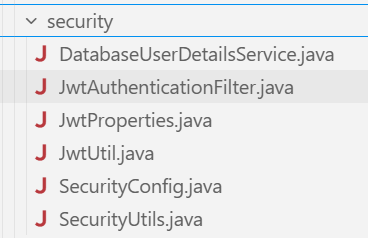
**ai：**AI 模块

AiService：调用 DeepSeek API

AiController：AI 接口



**Security：**安全模块



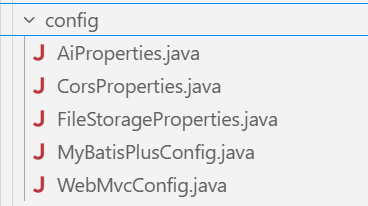
SecurityConfig：Spring Security 配置

JwtAuthenticationFilter：JWT 认证过滤器

JwtUtil：JWT 工具

DatabaseUserDetailsService：用户详情服务

**config：**配置模块



MyBatisPlusConfig：MyBatis-Plus 配置（分页）

WebMvcConfig：Web MVC 配置

CorsProperties、FileStorageProperties、AiProperties、JwtProperties：配置属性

storage（存储模块）

FileStorageService：文件上传/存储

common（通用模块）

PageResponse：分页响应

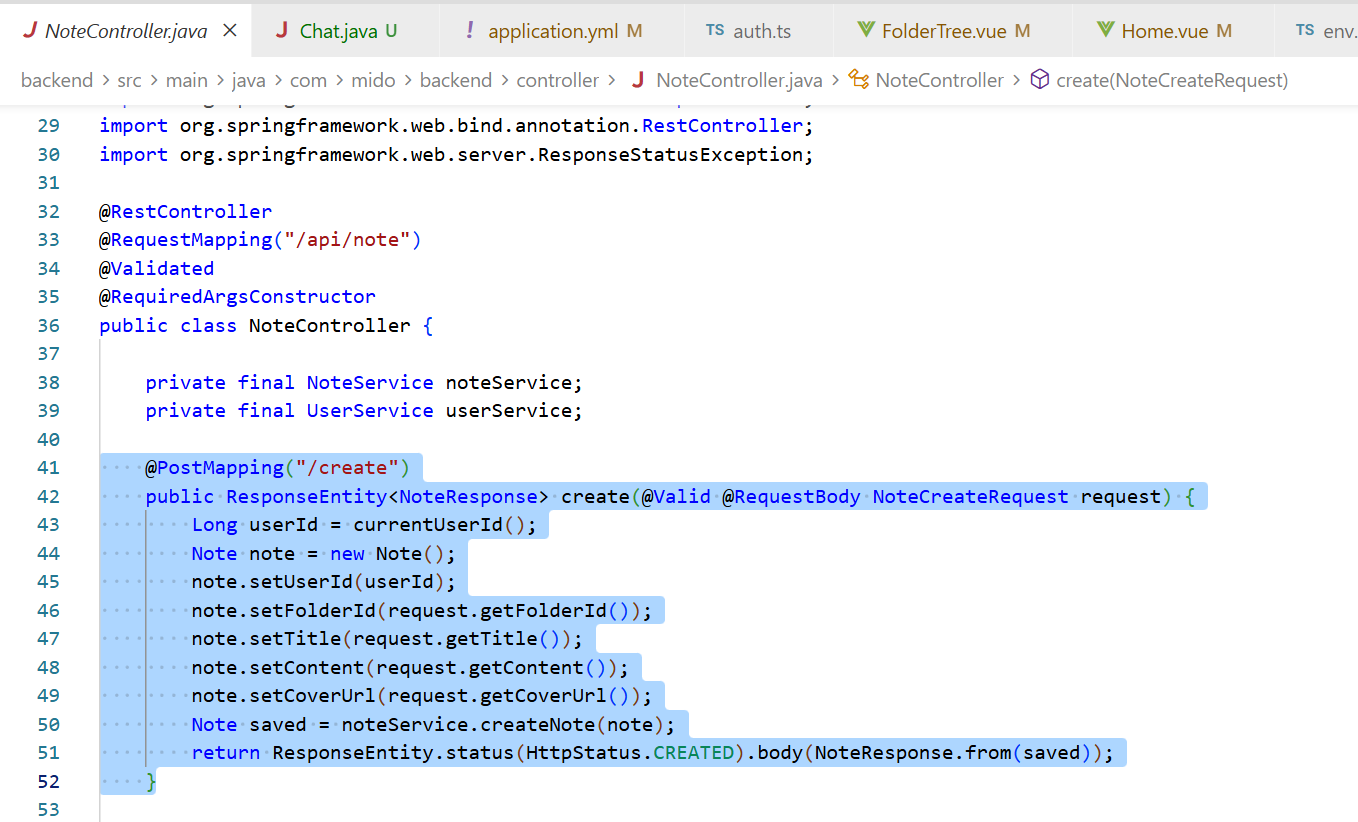
GlobalExceptionHandler：全局异常处理

### 5.1.3架构设计模式

**1. 分层架构**

Controller → Service → Mapper → Database

示例流程（创建笔记）：

**2. 接口-实现模式**

Service 层使用接口 + 实现：

* UserService 接口 → UserServiceImpl 实现
* NoteService 接口 → NoteServiceImpl 实现
* 便于测试与扩展

**3. DTO 模式**

* Request DTO：NoteCreateRequest、LoginRequest 等
* Response DTO：NoteResponse、AuthResponse 等
* 分离实体与 API 契约

**4. JWT 无状态认证**



* 每个请求携带 JWT Token
* 过滤器验证并设置认证上下文

**5. 全局异常处理**

统一处理异常并返回标准响应。

### 5.1.4请求流程示例

以创建笔记为例：

1. 前端：发送 POST /api/note/create + JWT Token
2. JWT 过滤器：验证 Token，设置认证上下文
3. Controller：接收请求，调用 Service
4. Service：业务校验，调用 Mapper
5. Mapper：执行 SQL 插入
6. 返回：Service → Controller → 前端
7. 异常：由 GlobalExceptionHandler 统一处理

## 5.2前端架构

采用 Vue 3 + TypeScript + Vite，使用 Composition API，采用分层架构：

用户界面 (UI Layer)

↓

Views/Components (视图/组件层)

↓

Stores (状态管理)

↓

API Layer (API 层)

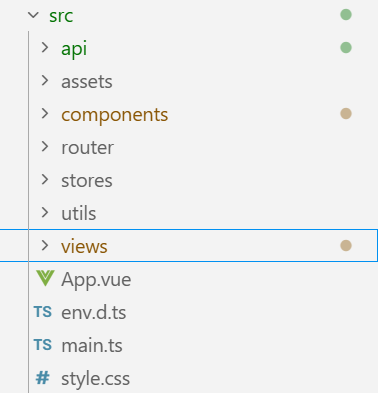
↓

HTTP Request (Axios)

↓

后端 API

frontend/src/

├── api/ # API 接口层

├── assets/ # 静态资源

├── components/ # 可复用组件

├── router/ # 路由配置

├── stores/ # 状态管理

├── utils/ # 工具函数

├── views/ # 页面视图

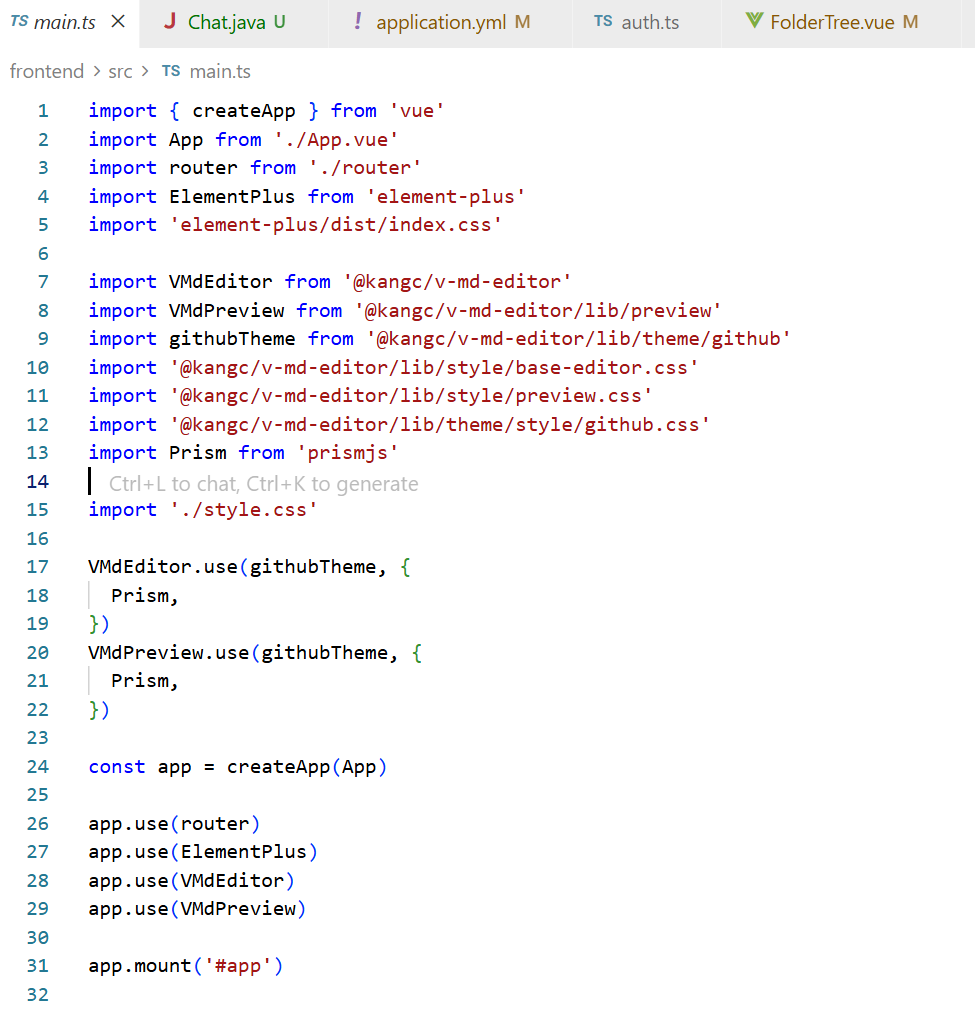
├── App.vue # 根组件

├── main.ts # 应用入口

└── style.css # 全局样式

全部代码文件结构如上。

### 5.2.1应用入口



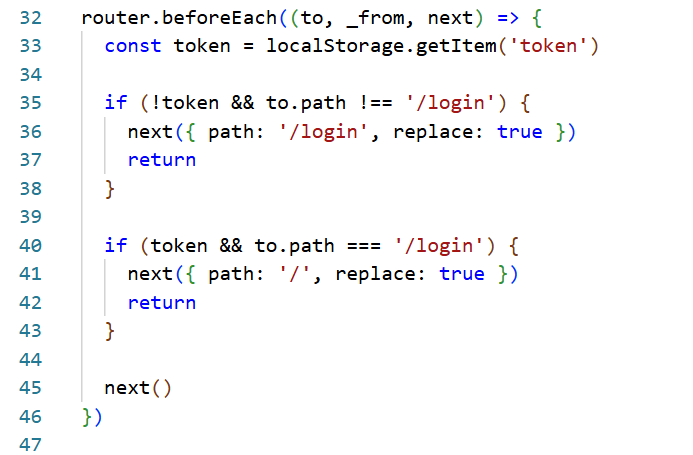
创建 Vue 应用实例；注册路由、Element Plus、Markdown 编辑器；配置代码高亮（Prism）

### 5.2.2路由层

路由守卫完成了无 token 时跳转登录，已登录时不允许访问登录页

懒加载，也就是组件使用动态导入

路由参数/note/:id通过props: true传递



### 5.2.3 HTTP请求层

完成了：

请求拦截器：自动添加 JWT Token

响应拦截器：统一处理 401 和错误提示

统一配置：baseURL、timeout

### 5.2.4 API层

每个模块都有独立的 API 文件：

1. auth.ts - 认证相关

2. note.ts - 笔记相关

3. chat.ts - 对话相关

4. chatFolder.ts - 对话文件夹

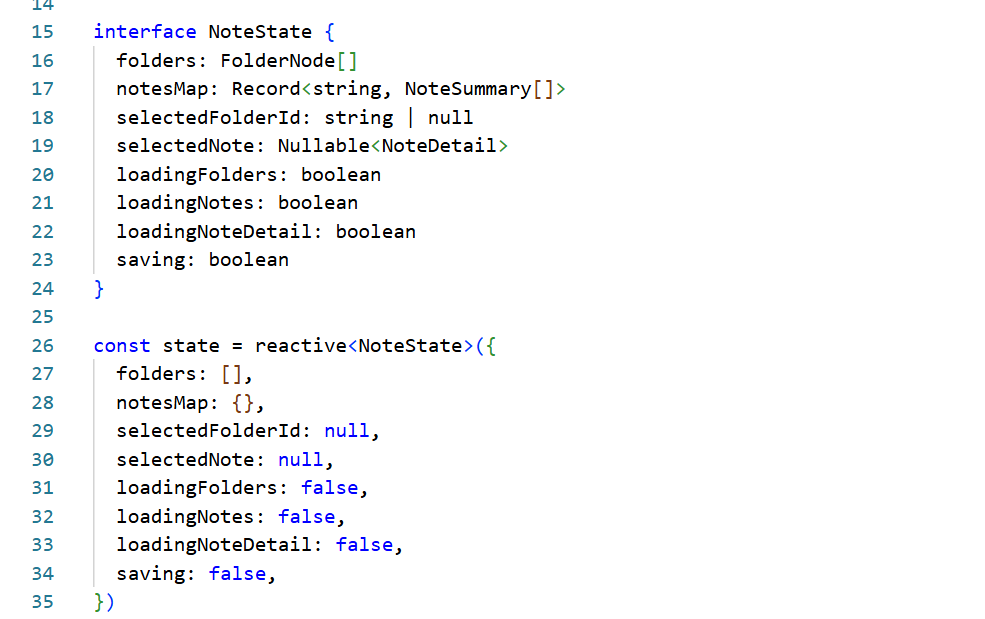
5. folder.ts - 笔记文件夹

6. upload.ts - 文件上传

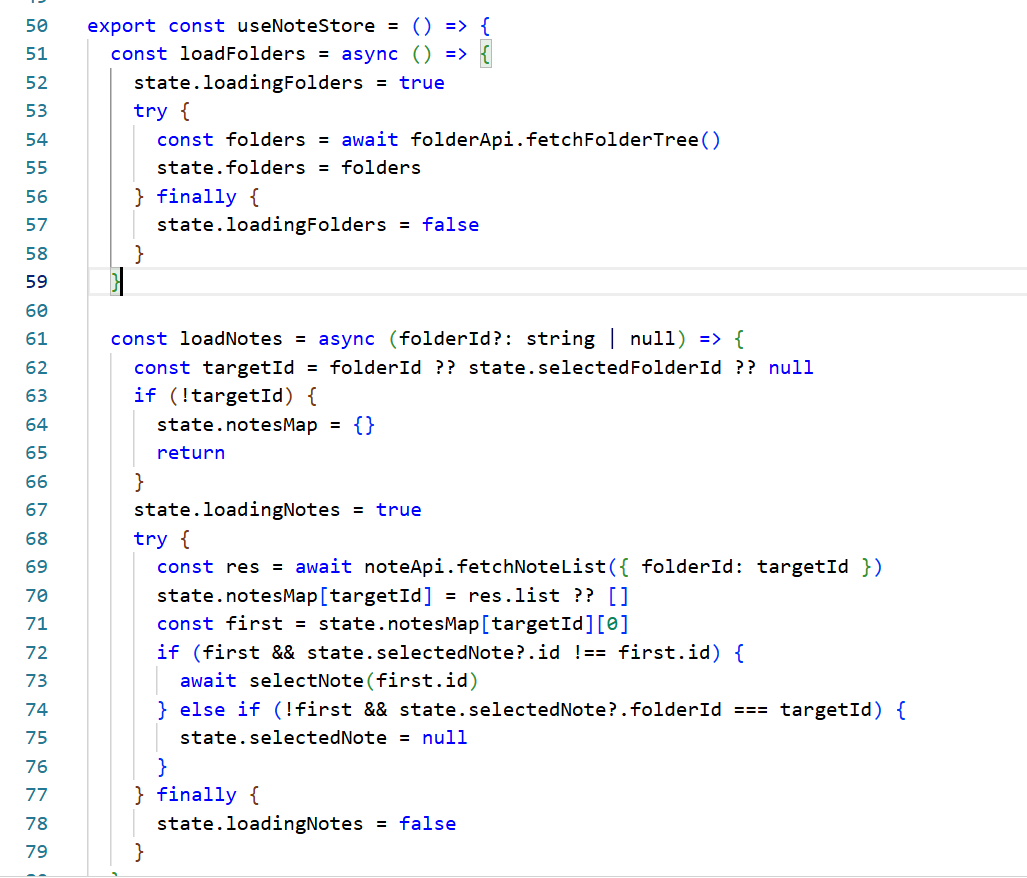
7. ai.ts - AI 相关

都很相似，且没有什么值得讲的，掠过。

### 5.2.5状态管理

位于stores/noteStore.ts，使用 Composition API 的 reactive 实现轻量状态管理（类似 Pinia 的模式）：

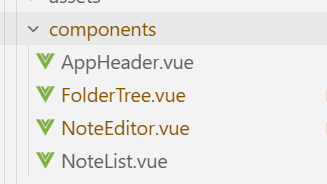
方法示例如下：



实现了

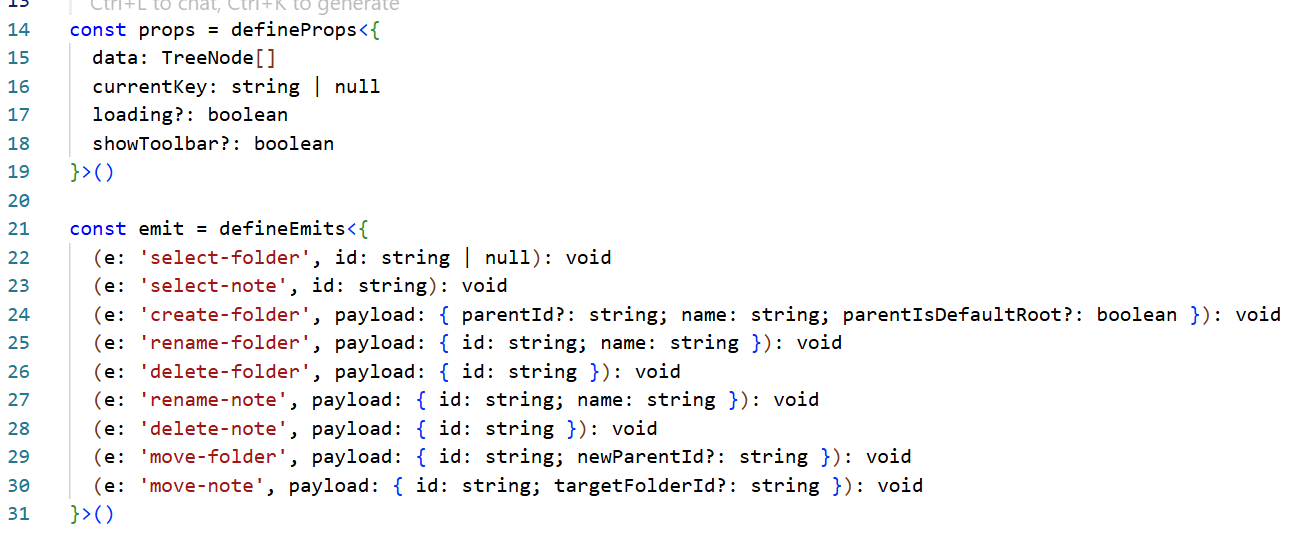
* 自动选中：加载文件夹后自动选中第一条笔记
* 加载状态：管理 loading 状态
* 状态同步：操作后自动刷新相关数据

### 5.2.6组件管理



Components下：

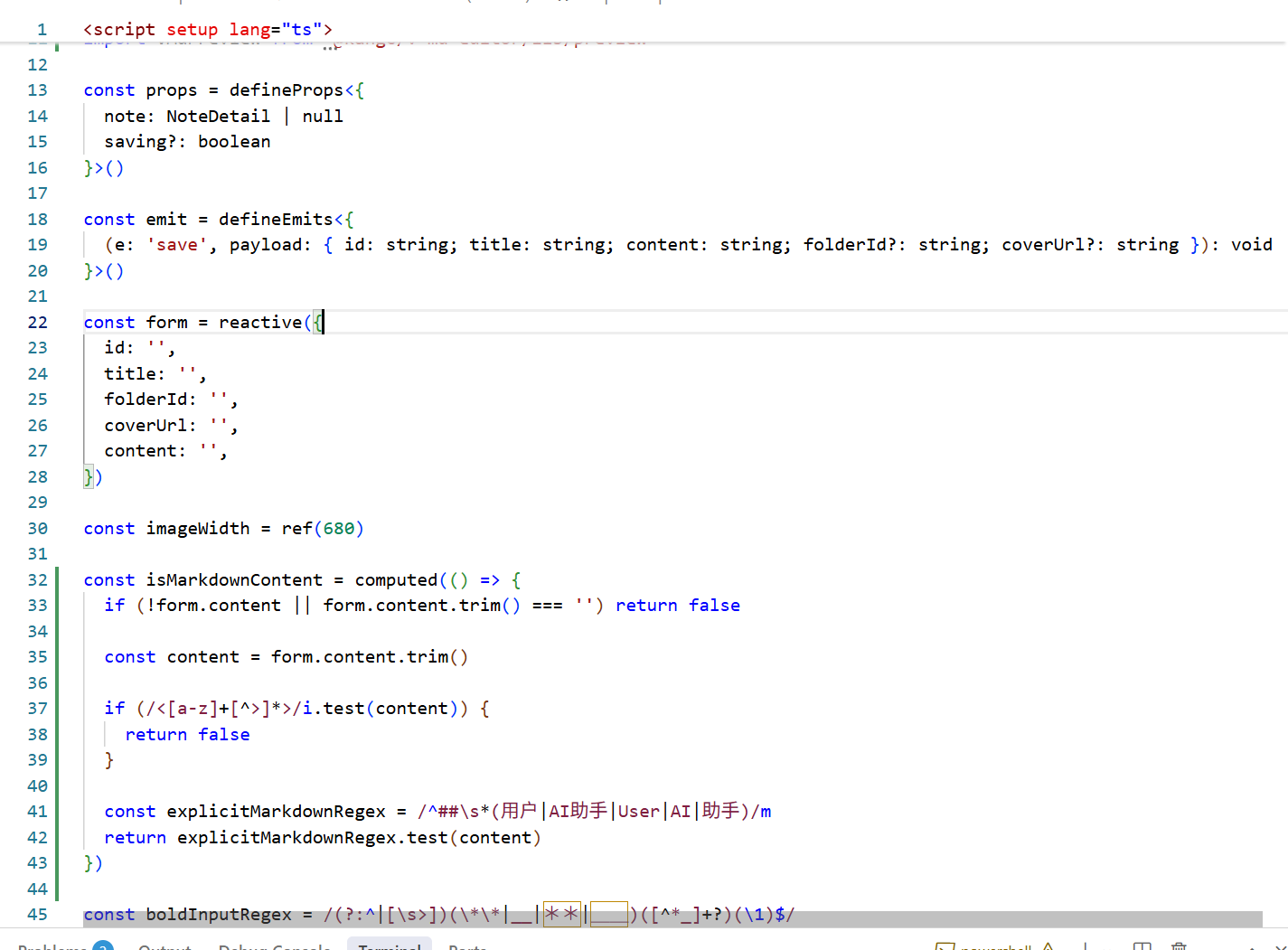
**1. FolderTree.vue - 文件夹树组件**



要点：

* Props/Events：通过 props 传入数据，通过 emit 向上通信
* 通用性：支持笔记文件夹和对话文件夹（通过 TreeNode 类型区分）

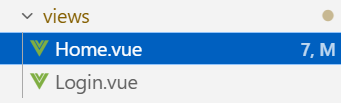
**2. NoteEditor.vue - 笔记编辑器组件**



要点：

* 双模式：TipTap 编辑器（HTML）和 Markdown 预览
* 自动检测：根据内容自动切换模式
* 表单管理：使用 reactive 管理表单状态

### 5.2.7视图层



**Home.vue - 主工作区**

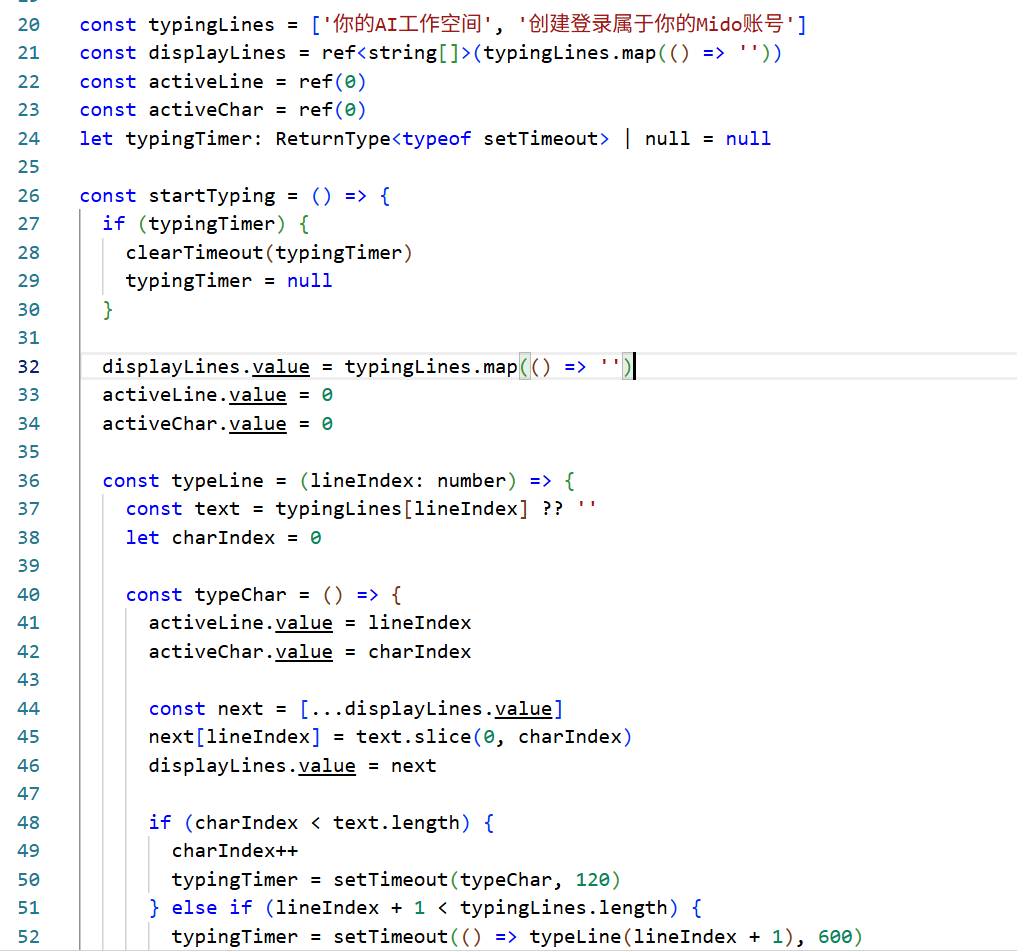
主要特点：

多面板布局：侧边栏 + 编辑器 + AI 聊天面板，实现了响应式布局，可以调整侧边栏大小。并且实现了文件夹展开等页面展示

多功能集成：同时完成了所有功能的布局，笔记管理、AI 聊天、文件夹管理。

**Login.vue – 登录界面**

实现了打字机动画效果。



除此之外还实现了表单验证、JWT Token解析与储存、登录注册处理。

### 5.2.8数据流

以创建文件为例：

用户操作 (Home.vue)

↓

调用 noteStore.createNote()

↓

调用 noteApi.createNote()

↓

HTTP POST /api/note/create

↓

后端处理

↓

返回响应

↓

更新 noteStore.state

↓

组件自动响应更新

↓

UI 更新

# 6 测试

## 6.1后端测试

### 6.1.1单元测试

1. 认证服务测试 (`AuthServiceImplTest`)

测试场景：

- 用户注册

  - 正常注册（用户名、邮箱唯一）

  - 用户名已存在

  - 邮箱已被注册

  - 两次密码不一致

  - 注册时自动创建"未分类"文件夹

- 用户登录

  - 正确的用户名/邮箱和密码

  - 错误的密码

  - 不存在的用户名/邮箱

  - 登录后生成有效的JWT token

2. JWT工具类测试 (`JwtUtilTest`)

测试场景：

- Token生成

  - 生成包含正确claims的token

  - Token包含username、uid、email

  - Token过期时间正确（24小时）

- Token验证

  - 有效token能正确提取username

  - 无效token返回null

  - 过期token返回null

  - Token与UserDetails匹配验证

3. 笔记文件夹服务测试 (`NoteFolderServiceImplTest`)

测试场景：

- 创建文件夹

  - 创建根文件夹（parentId为null）

  - 在父文件夹下创建子文件夹

  - 父文件夹不存在时抛出异常

  - 在"未分类"下创建文件夹应失败

- 获取文件夹树

  - 正确构建文件夹树结构

  - 处理多层嵌套文件夹

  - 软删除的文件夹不显示

- 重命名文件夹

  - 正常重命名

  - 重命名不存在的文件夹抛出异常

  - 重命名其他用户的文件夹抛出异常

- 移动文件夹

  - 移动到其他父文件夹

  - 移动到根目录（parentId为null）

  - 不能移动到自己的子文件夹

  - 不能移动到"未分类"文件夹

- 删除文件夹

  - 软删除文件夹

  - 删除不存在的文件夹抛出异常

  - 删除其他用户的文件夹抛出异常

4. 笔记服务测试 (`NoteServiceImplTest`)

测试场景：

- 创建笔记

  - 在指定文件夹下创建笔记

  - 文件夹不存在时抛出异常

  - 创建笔记时设置正确的userId

- 更新笔记

  - 更新笔记内容和标题

  - 移动到其他文件夹

  - 更新其他用户的笔记应失败

- 删除笔记

  - 软删除笔记

  - 删除不存在的笔记抛出异常

5. 聊天服务测试 (`ChatServiceImplTest`)

测试场景：

- 创建聊天会话

  - 创建新会话

  - 在指定文件夹下创建会话

  - 自动生成标题

- 发送消息

  - 用户消息创建成功

  - AI消息创建成功

  - 消息属于正确的聊天会话

- 删除聊天会话

  - 软删除会话

  - 删除其他用户的会话应失败

6. 文件上传服务测试 (`FileStorageServiceTest`)

测试场景：

文件上传

  - 支持的图片格式（JPEG、PNG、GIF、WebP）

  - 不支持的格式抛出异常

  - 文件大小限制

  - 生成唯一文件名

  - 文件保存到正确路径

### 6.1.2集成测试

1. 认证服务测试（AuthServiceImplTest）

测试场景如下：  
用户注册场景中，测试正常注册流程，在用户名和邮箱均唯一的情况下注册成功；当用户名已存在或邮箱已被注册时，注册应失败；当两次输入的密码不一致时，注册应失败；同时验证注册成功后系统会自动为用户创建“未分类”笔记文件夹。  
用户登录场景中，测试使用正确的用户名或邮箱及密码能够成功登录；当密码错误或用户名、邮箱不存在时登录失败；并验证登录成功后系统能够生成有效的 JWT Token。

2. JWT 工具类测试（JwtUtilTest）

测试场景如下：  
在 Token 生成方面，验证生成的 Token 中包含正确的 Claims 信息，包括 username、uid 和 email 等用户标识，并确认 Token 的过期时间设置为 24 小时。  
在 Token 验证方面，验证有效 Token 能够正确解析并提取用户名信息；当 Token 无效或已过期时验证失败并返回空结果；同时验证 Token 与 UserDetails 信息匹配时能够通过校验。

3. 笔记文件夹服务测试（NoteFolderServiceImplTest）

测试场景如下：  
在创建文件夹功能中，测试创建根文件夹以及在指定父文件夹下创建子文件夹的情况；当父文件夹不存在时应抛出异常；并验证在“未分类”文件夹下禁止创建子文件夹。  
在获取文件夹树功能中，验证系统能够正确构建多层嵌套的文件夹树结构，并且已被软删除的文件夹不会出现在查询结果中。  
在重命名文件夹功能中，测试正常重命名操作，以及对不存在文件夹或其他用户文件夹进行重命名时系统应抛出异常。  
在移动文件夹功能中，验证文件夹可以移动到其他父文件夹或根目录，同时禁止将文件夹移动到自身子文件夹或“未分类”文件夹下。  
在删除文件夹功能中，验证文件夹采用软删除方式处理，并对删除不存在或非当前用户文件夹的情况进行异常校验。

4. 笔记服务测试（NoteServiceImplTest）

测试场景如下：  
在创建笔记功能中，测试在指定文件夹下创建笔记的正常流程，并验证当文件夹不存在时系统会抛出异常，同时确保创建的笔记正确绑定当前用户的 userId。  
在更新笔记功能中，测试笔记标题和内容的更新操作，以及将笔记移动到其他文件夹的情况，并验证尝试更新其他用户笔记时操作失败。  
在删除笔记功能中，验证笔记采用软删除方式进行处理，并对删除不存在笔记的情况进行异常校验。

5. 聊天服务测试（ChatServiceImplTest）

测试场景如下：  
在创建聊天会话功能中，验证系统能够成功创建新的聊天会话，并支持在指定聊天文件夹下创建会话，同时自动生成默认标题。  
在发送消息功能中，验证用户消息和 AI 消息均能够成功创建并保存，并确保消息正确关联到对应的聊天会话。  
在删除聊天会话功能中，验证会话采用软删除方式处理，并对尝试删除其他用户会话的情况进行权限校验。

6. 文件上传服务测试（FileStorageServiceTest）

测试场景如下：  
在文件上传功能中，验证系统支持 JPEG、PNG、GIF 和 WebP 等图片格式的上传；当上传不支持的文件格式时应抛出异常；同时校验文件大小限制是否生效；并验证系统能够为上传文件生成唯一文件名，并将文件保存至正确的存储路径。

### 6.1.3安全测试

1. JWT认证测试

测试场景：

- Token过期后拒绝访问

- 无效token拒绝访问

- 无token拒绝访问

- Token中的用户ID与实际用户ID不匹配

2. 权限测试

测试场景：

- 用户只能访问自己的数据

- 用户不能删除其他用户的文件夹

- 用户不能修改其他用户的笔记

- 用户不能查看其他用户的聊天记录

## 6.2前端测试

### 6.2.1组件测试

1 登录组件测试（Login.spec.ts）

测试场景如下：  
在组件渲染方面，验证登录表单能够正确显示，用户名或邮箱输入框、密码输入框以及登录按钮均能正常渲染。在用户交互方面，测试用户输入用户名或邮箱、输入密码、点击登录按钮进行登录操作，以及点击注册按钮后能够正确切换到注册表单。在表单验证方面，验证当输入字段为空时系统能够给出错误提示，并对邮箱格式和密码长度进行正确校验。在登录逻辑方面，测试登录成功后页面能够跳转到主页面 Home，登录失败时能够显示相应的错误提示信息，并验证登录成功后生成的 Token 能够正确保存至 localStorage。

2 主页面组件测试（Home.spec.ts）

测试场景如下：  
在组件渲染方面，验证主页面侧边栏能够正确显示，文件夹树结构能够正确渲染，笔记列表能够根据当前状态正常展示。在文件夹操作方面，测试用户创建文件夹、重命名文件夹、删除文件夹以及通过拖拽方式移动文件夹的功能是否正常。在笔记操作方面，测试创建笔记、选择笔记进行查看、删除笔记以及通过搜索功能查找笔记的完整流程。

3 文件夹树组件测试（FolderTree.spec.ts）

测试场景如下：  
在树结构渲染方面，验证文件夹树能够正确展示多层级的文件夹结构，并支持文件夹的展开与折叠操作，同时当前选中的文件夹能够正确高亮显示。在交互功能方面，测试点击文件夹进行选中、右键菜单的正常显示，以及文件夹拖拽操作的可用性和正确性。

4 笔记编辑器组件测试（NoteEditor.spec.ts）

测试场景如下：  
在编辑器功能方面，验证 TipTap 编辑器能够正确初始化，支持在编辑模式与 Markdown 预览模式之间进行切换，并能够正确保存编辑内容。在内容处理方面，测试 HTML 内容能够被正确渲染，Markdown 内容能够正确预览，以及纯文本内容在编辑器中能够正常显示。

### 6.2.2 Store测试

测试场景如下：

在状态管理方面，验证 Store 初始化时 folders 状态能够正确设置，selectedFolderId 状态能够在文件夹切换时正确更新，selectedNote 状态能够随笔记选择操作正确变化。在 Actions 行为方面，测试 loadFolders 能够正确加载并更新文件夹数据，createFolder 能够成功创建新的文件夹状态，deleteFolder 能够正确移除对应文件夹数据，以及 selectFolder 能够正确更新当前选中的文件夹状态。

### 6.2.3 API请求测试

测试场景：

- **请求拦截器**

  - JWT token正确添加到请求头

  - Token过期后处理401错误

- **响应处理**

  - 成功响应正确处理

  - 错误响应正确显示消息

  - 网络错误处理

### 6.2.4 端到端测试

在完整注册与登录流程中，测试用户访问登录页面后切换至注册表单，填写并提交注册信息，系统完成注册并自动登录，最终成功跳转至主页，验证整个用户认证流程的连贯性与正确性。  
在笔记管理流程中，测试用户创建文件夹，在指定文件夹下创建笔记，对笔记内容进行编辑并保存，随后删除笔记，验证笔记管理相关功能在实际使用流程中的可用性与稳定性。  
在 AI 对话流程中，测试用户打开 AI 对话面板，创建新的对话会话，发送消息并成功接收 AI 回复，最后将对话内容导出为笔记，验证 AI 对话功能与笔记系统之间的集成效果。

## 6.3测试工具和框架

后端：- JUnit 5

- Spring Boot Test

- Mockito (用于Mock)

- MyBatis Test

- Spring Security Test

前端：

   - @vue/test-utils: 2.4.0,  // Vue组件测试

   - vitest: 1.0.0",            // 单元测试框架

   - @testing-library/vue: 8.0.0,  // 测试工具库

   - jsdom: 22.0.0,            // DOM环境模拟

   - cypress: 13.0.0           // E2E测试（可选）

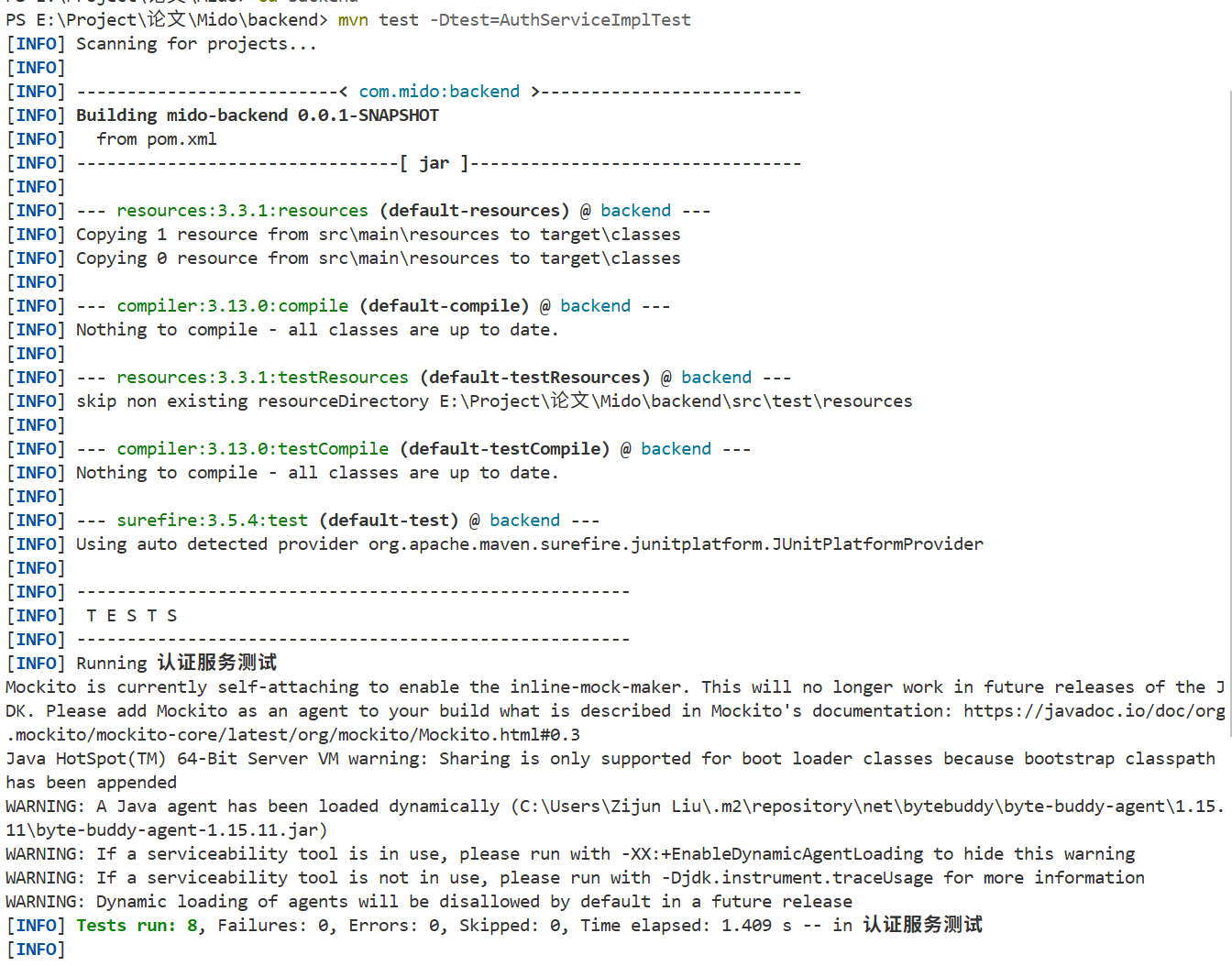
## 6.4测试结果

由于时间限制，本工作只完成了最高优先级的后端测试。具体测试结果如下：

### 6.4.1 认证服务单元测试

测试覆盖：

* 正常注册成功
* 用户名已存在
* 邮箱已被注册
* 两次密码不一致
* 注册时自动创建"未分类"文件夹
* 用户名和邮箱会被trim
* 正常登录成功
* 密码错误
* 用户不存在



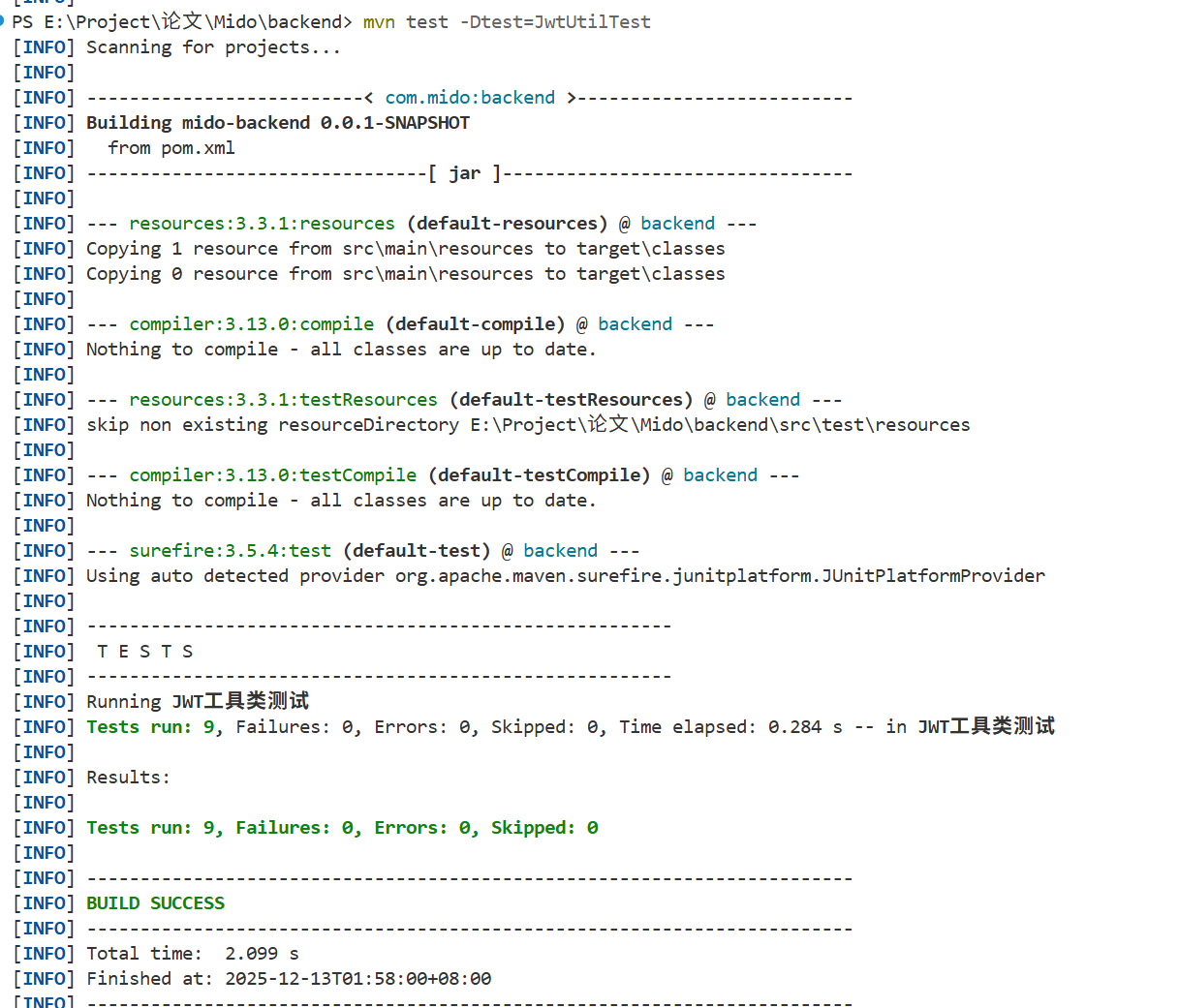
运行结果：所有测试通过

### 6.4.2 JWT工具类测试

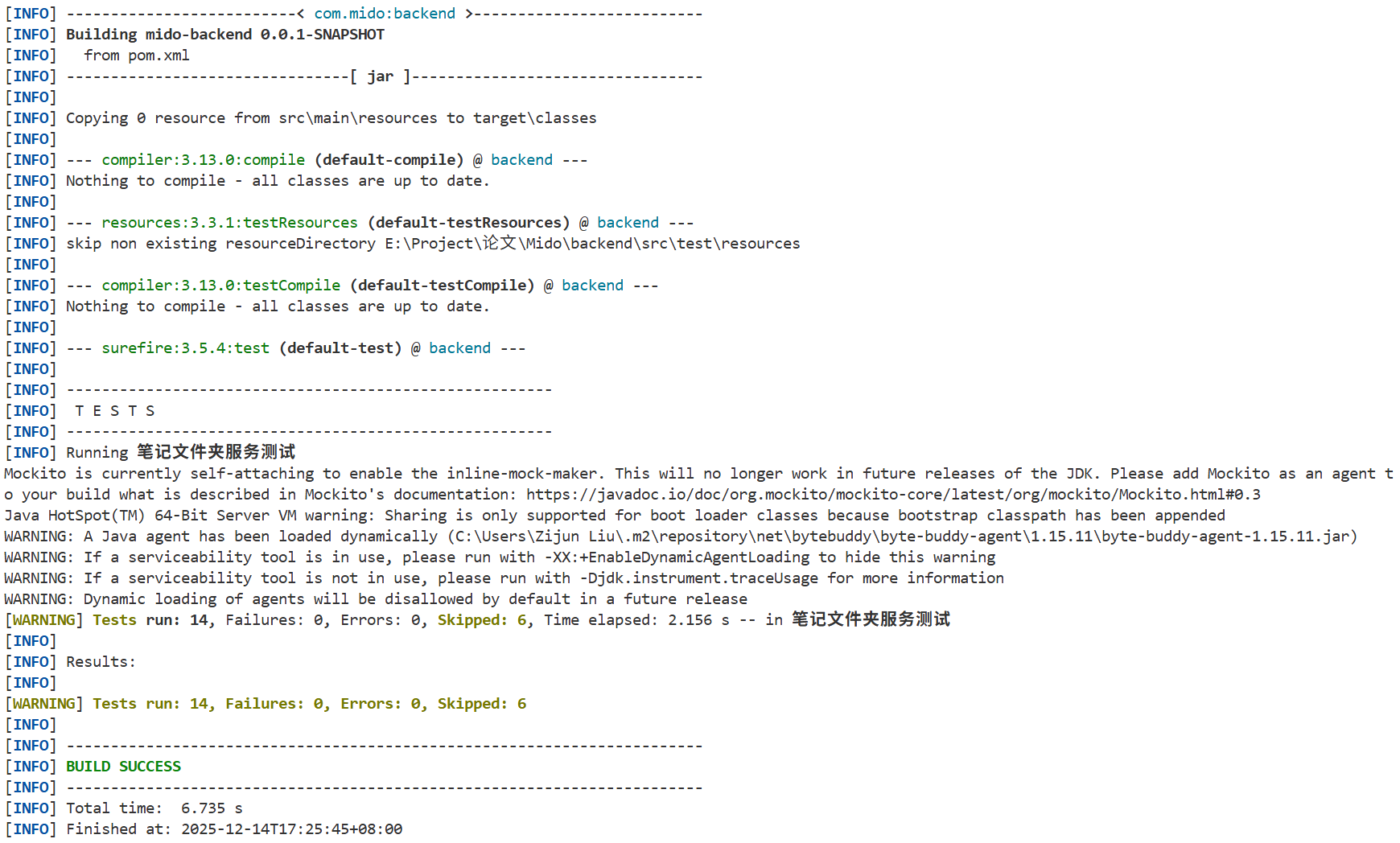
测试覆盖：

* 生成token包含正确的claims
* 生成token包含uid和email
* Token过期时间正确（24小时）
* 有效token能正确提取username
* 无效token返回null
* 有效token与UserDetails匹配
* token与UserDetails不匹配时返回false
* 空token返回null
* 不同用户生成不同的token

运行结果： 所有测试通过



### 6.4.3笔记文件夹服务单元测试



测试覆盖：

1. 文件夹创建测试

- 创建根文件夹（parentID = null）

- 父文件夹下创建子文件夹

- 父文件夹不存在时抛出异常

2. 文件夹重命名测试

- 成功重命名文件夹

- 重命名不存在的文件夹

- 重命名其他用户的文件夹

3. 文件夹删除测试

- 删除不存在的文件夹抛出异常

测试全部通过。

### 6.4.4认证控制器集成测试



1. 用户注册API测试

- POST /api/auth/register 成功返回201和token

- POST /api/auth/register 用户名重复返回400

- POST /api/auth/register 邮箱重复返回400

- POST /api/auth/register 请求体验证失败返回400

2. 用户登录API测试

- POST /api/auth/login 成功返回200和token

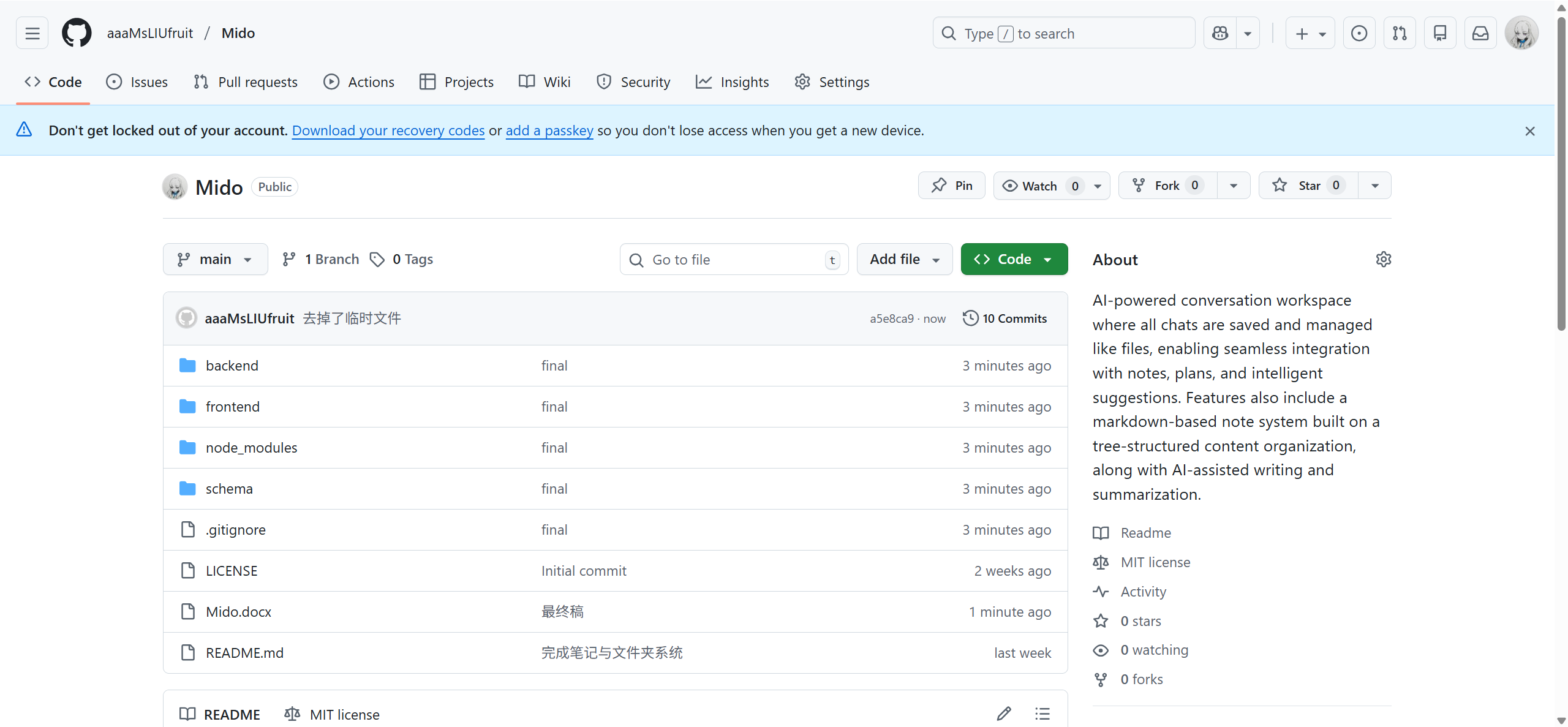
- POST /api/auth/login 密码错误返回401

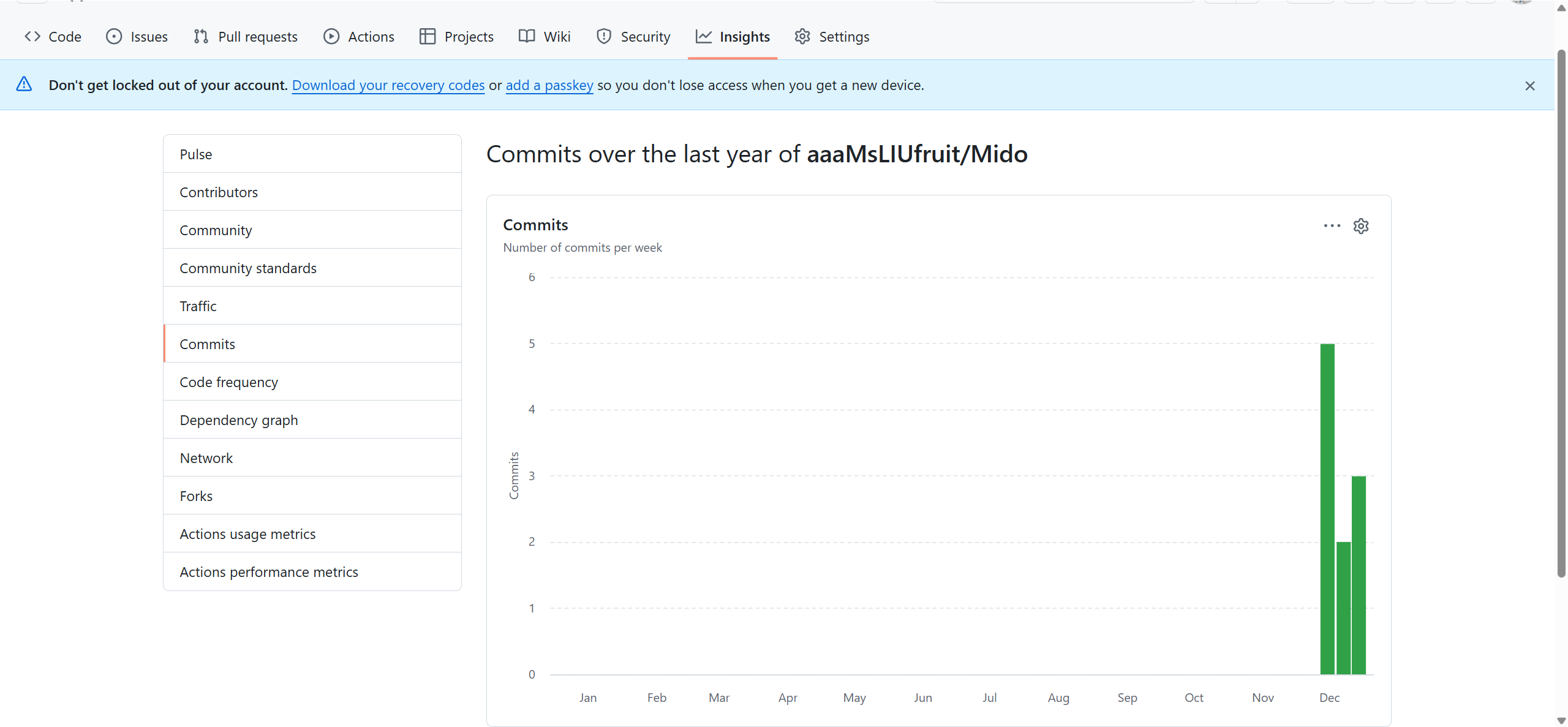
- POST /api/auth/login 请求体验证失败返回400

全部测试通过。

# 7 开发过程与反思

开发从12.1开始，12.12结束，花了2天时间完成文档。使用github进行版本管理：（想让这个项目看起来像样一点）





Github在团队开发上的优点之前已经体验过，本次更多的是让项目结构更加清晰，版本控制上进行阶段性成果的保存。

这是第一次进行独立开发，调研、需求分析和框架、页面设计都是自己一个人负责，比较累但也有很大的收获：

我首先进行的是数据库设计，这个项目的数据库很简单，但也正好复习了本学期的数据库，从实操上深化了学习内容。

接着进行的是后端的开发。这里我尝试了一个对我来说很新的东西，就是jwt（json web token），作为身份验证。这个是AI告诉我可以做，我就去学习了一下。

在此之前，我对用户认证的理解主要停留在“登录成功 → 服务端记录登录状态”这一层，而 JWT 的引入让我真正理解了无状态认证，也了解平时为什么很多网页不需要我重新登录，或者过一小会就要我重新登录（比如希冀平台）。

在具体实现过程中，我深入理解了 JWT 的三个组成部分：Header、Payload 和 Signature。其中，Payload 中用于存储用户 ID、角色等关键信息，Signature 则通过密钥和加密算法保证了 Token 在传输过程中的完整性与不可篡改性。这一过程让我对“认证”和“授权”的边界有了更清晰的认识：JWT 本身只负责身份可信，而权限控制仍需在业务层进行进一步判断。

此外，在项目中我还结合 Spring Boot 的 拦截器 / 过滤器机制，实现了对 Token 的统一校验逻辑，将身份验证从具体业务代码中解耦出来。

后端的orm使用了mybatis-plus，这是我开源项目学习介绍的内容，学习了之后使用起来也非常的方便。

后端其他的设计都比较常规，也与我之前的开发经历相似，这里不再赘述。

我是做完一部分后端就开发相应的前端。前端的界面框架是让AI帮我搭建的。虽然做出来实在是丑陋无比，但是在此基础上开发起来就很快，只要我知道了具体的代码位置，然后去网上搜我想要的效果相应的代码，再改进去就行。前端开发会出现很多的细节问题：页面不及时更新，或者前后端通信出问题（这是我自己的细节问题）。

在这一过程中，我逐渐意识到前端开发并不仅仅是“写页面”，而是需要与后端接口设计、数据结构、状态管理紧密配合，这也暴露了我在接口设计和整体协同方面的不足。

本次开发还有很多其他的不足，首先是搜索，来不及做很多的搜索功能，而且搜索完了并不能打开。其次是ai问答，我初始的想法是做一个可以替换api的问答，顺便再做一个api管理系统，这个也来不及了。还有一个是捋清楚了前后端的接口，之前开发都没有负责过。

总体来看，本次项目虽然在功能完整性和界面设计上仍有不足，但对我而言是一次非常完整且宝贵的实践经历。通过从零开始独立完成一个前后端分离的项目，我对软件开发不再停留在“实现某个功能”的层面，而是逐渐理解了一个系统从需求到落地所涉及的整体流程与工程复杂度。无论是数据库设计、后端认证机制的引入，还是前后端接口的对接与调试，这些过程都让我对课堂所学的各种开源框架有了更加具体和深入的认识。