软件工程课设

1. **Rust学习空间**
   a. 技术：rust 语言 rocket 后端框架，前端，(使用docker运行练习代码)
   b. 功能：
      i. 视频学习（B站链接 ）【根据搜索框匹配（搜索框：模糊匹配）】
      ii. 编程练习，题目练习【题目列表，进行比较，测试用例通过百分比，错误报错、错误处理】，题目要打标签。
      iii. blog发布与提问交流【用户登录，每道题下面有】
      iv. AI提问(可将代码提交给AI进行分析纠正)、推荐知识点复习（调用LLM，异步等待结果[流输出]）
   c. 效果：

2. 分工：
   a. hk：Git分支、开发文档、数据库设计(**MySQL**)
   b. lgr：LLM大模型（历史记录？？？，根据错误输出编程建议，推荐知识点复习，保持会话状态维护上下文）
   c. wrk：前端接口实现
   d. czg：blog发布与提问交流、视频学习，软件说明书
   e. ws：编程练习、题目练习（返回错误信息）【编译运行，结果正确？等待结果？等待重定向？】
   f. Postman，Rust单元测试工具

> 📜 参考网址：AI掘金编程

1. 架构分层

1. 开发架构
   - **前端**：Web 应用（React/Vue 或 Rust WASM 框架如 Yew）

- **后端**：Rust 高性能框架（Actix-web 或 Rocket）
- **数据库**：MySQL（支持复杂查询和标签系统）
- **AI 服务**：通过 HTTP/gRPC 调用外部 LLM（如 OpenAI 或本地部署的模型）
- **代码沙箱**：Docker 容器隔离用户提交的代码

2. 后期运维测试
   - **容器化**：Docker + Kubernetes（高并发场景）
   - **监控**：Prometheus + Grafana 监控 API 性能
   - **日志**：`tracing` 库 + ELK 栈
3. 设计要点
   a. **安全设计**:
      - 用户密码使用 `Hash` 加密存储
      - JWT Token 有效期控制在 2 小时
      - 代码沙箱使用 Docker 隔离（限制 CPU/内存/网络）
   b. **性能优化**:
      - 为高频查询字段（如 `problems.tags`）添加 GIN 索引

      代码块

      ```
      CREATE INDEX idx_problems_tags ON problems USING GIN(tags);
      ```

      - 使用 Redis 缓存题目列表和热门博客
   c. **扩展性**:
      - 通过 `test_cases` 的 JSONB 设计支持多种测试类型
      - `ai_history.full_data` 保留原始数据供后续分析
   d. **数据一致性**:
      - 使用外键约束确保关联数据有效性
      - 重要操作（如提交代码）添加数据库事务保护

2. Github代码托管

📜 项目地址：https://github.com/Limerc/SoftwareEngineeringDesign

打开上面链接看一下，我帮你们每个人都建立了一个分支，**add origin 自己名字的分支就行了，不要pull 和 merge别人的分支！！！**

2.1 配置SSH Key，安装git

你们没有配置过的自己上网找教程，配置过的直接下一步就行

2.2 新建本地github仓库并推送代码

1. 打开终端初始化git仓库：`git init`
2. 添加全部代码：`git add .` **（这一步是把代码添加到工作区）**
3. 提交代码：`git commit -m "批注"` **【这一步是把代码推送到本地仓库】**
4. 链接远程仓库：`git remote add origin + 你的ssh地址` **（项目地址：**`git@github.com:Limerc/SoftwareEngineeringDesign.git`**）**
5. 转换到特定分支下：`git checkout -b <你的分支>` **（例：git checkout -b hk）**
6. 删除默认master分支：`git branch -D master` **（本地初始后默认创建master分支，这个在github上没有，可以删掉）**
7. 覆盖之前提交记录强制推送代码：`git push origin <你的分支> --force` **(git push origin hk --force)【这一步是将本地仓库代码推送到远端】**

2.3 推送修改后的代码

1. 查看当前git仓库状态：`git status`
2. 更新全部：`git add .`
3. 输入`git commit -m "更新说明"`提交更新
4. 拉取main分支与当前代码合并：`git pull origin <你的分支>`
5. 将项目Push到远程main分支上：`git push origin <你的分支>`

3. 用户相关接口

3.1 注册

3.1.1 基本信息

请求路径：`/api/user/register`

请求方式：POST

接口描述：该接口用于注册新用户

3.1.2 请求参数

请求参数格式：x-www-form-urlencoded

请求参数说明：

参数名称	说明	类型	是否必须	备注
username	用户名	string	是	3~16位非空字符
password	密码	string	是	3~16位非空字符
phone	联系方式	string	是	11位手机号(唯一)

请求参数样例：

```
代码块
1  {
2      "username":"wangba",
3      "password":"123456",
4      "phone":"15632334667"
5  }
```

3.1.3 响应数据

响应数据类型：application/json

响应参数说明：

参数名称	类型	是否必须	默认值	备注	其他信息
code	number	是		响应码,0成功,1失败	
message	string	否		提示信息	
data	object	否		返回数据	

响应参数样例：

```
代码块
1  {
2      "code":0,
3      "message":"操作成功",
4      "data":"注册成功"
```

```
5   }
```

- 注：响应信息返回的数据data暂时默认为Null，后续有需要可以进行相应的添加。

3.2 用户登录

3.2.1 基本信息

请求路径：`/api/user/loginIn`

请求方式：POST

接口描述：该接口用于登录

3.2.2 请求参数

请求参数格式：x-www-form-urlencoded

请求参数说明：

参数名称	说明	类型	是否必须	备注
phone/usrname	联系方式/用户名	string	是	11位手机号
password	密码	string	是	3~16位非空字符

请求参数样例：

代码块

```
1   {
2       "phone": 13684930299,
3       "password": "12345678",
4   }
```

3.2.3 响应数据

响应数据类型：application/json

响应参数说明：

参数名称	类型	是否必须	默认值	备注	其他信息
code	number	是		响应码,0成功,1失败	
message	string	否		提示信息	

data	object	否		返回数据,JWT令牌	

响应参数样例：

代码块

```
{
    "code":0,
    "message":"操作成功",
   "data":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbGFpbXMiOnsiaWQiOjUsInVzZXJuYW1lIjoid2FuZ2JhI
n0sImV4cCI6MTY5MzcxNTk3OH0.pE_RATcoF7Nm9KEp9eC3CzcBbKWAFOL0IsuMNjnZ95M"
}
```

- **注：用户登录成功后，系统会自动下发JWT令牌，然后在后续的每次请求中，浏览器都需要在请求头header中携带到服务端，请求头的名称为 Authorization，值为登录时下发的JWT令牌。如果检测到用户未登录，则http响应状态码为401**

3.3 更新用户信息

3.3.1 基本信息

请求路径：`/api/user/update`

请求方式：PUT

接口描述：该接口用于更新已登录用户的基本信息（除密码、ID、Is_admin外）

3.3.2 请求参数

请求参数格式：application/json

请求参数说明：

参数名称	说明	类型	是否必须	备注
u_id	用户id	int	是	
username	用户名	string	否	3~16位非空字符
phone	联系方式	string	是	11位手机号

请求参数样例：

```
{
    "u_id":1,
    "username":"翟天博",
    "phone":"18218739303",
    "IDNumber":"440402933849583992"
}
```

- 前端传递的字段名是`IDNumber`，而后端的`UserList`类中属性名是`IDNumber`（按照驼峰命名法）。但是，JSON默认会遵循小写+驼峰规则，因此需要确保反序列化时匹配字段名。
- 解决方法：将前端传参改为idNumber或者使用@JsonProperty注解：

代码块

```
@NotEmpty
@Pattern(regexp = "^\\d{17}[0-9Xx]$")
@JsonProperty("IDNumber")
private String IDNumber;
```

3.3.3 响应数据

响应数据类型：json

响应参数说明：

参数名称	类型	是否必须	默认值	备注	其他信息
code	number	是		响应码,0成功,1失败	
message	string	否		提示信息	
data	object	否		返回数据	

响应参数样例：

代码块

```
{
    "code":0,
    "message":"操作成功",
    "data":"更新成功"
}
```

3.4 更新用户密码

3.4.1 基本信息

请求路径：`/api/user/updatePwd`

请求方式：PATCH

接口描述：该接口用于更新已登录用户的密码

3.4.2 请求参数

请求参数格式：application/json

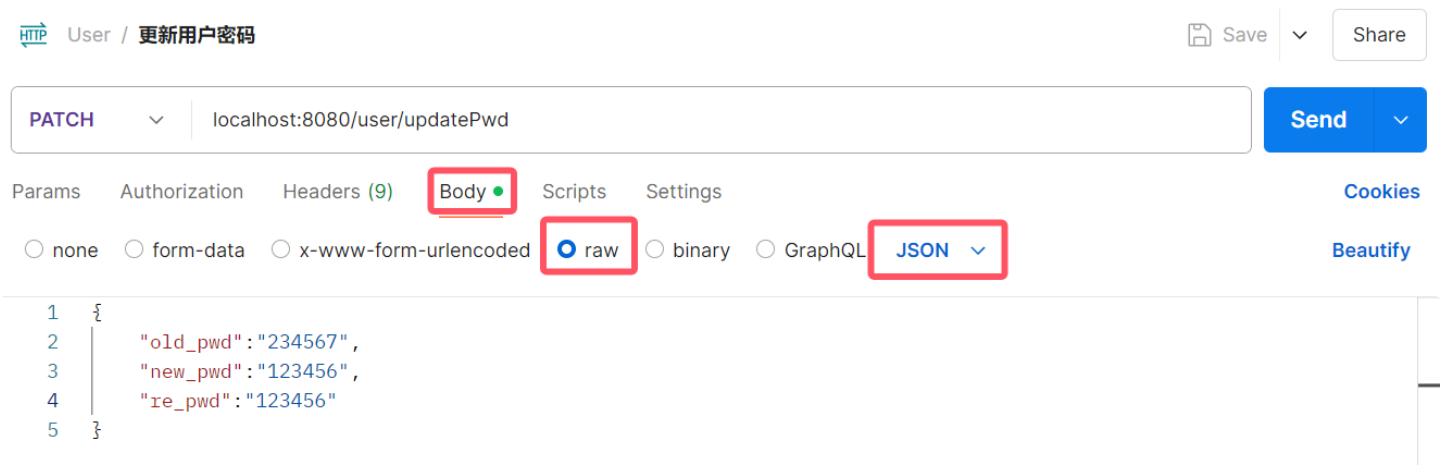
请求参数说明：

参数名称	说明	类型	是否必须	备注
old_pwd	原密码	string	是	
new_pwd	新密码	string	是	
re_pwd	确认新密码	string	是	

请求参数样例：

代码块

```
{
    "old_pwd":"123456",
    "new_pwd":"234567",
    "re_pwd":"234567"
}
```



3.4.3 响应数据

响应数据类型：json

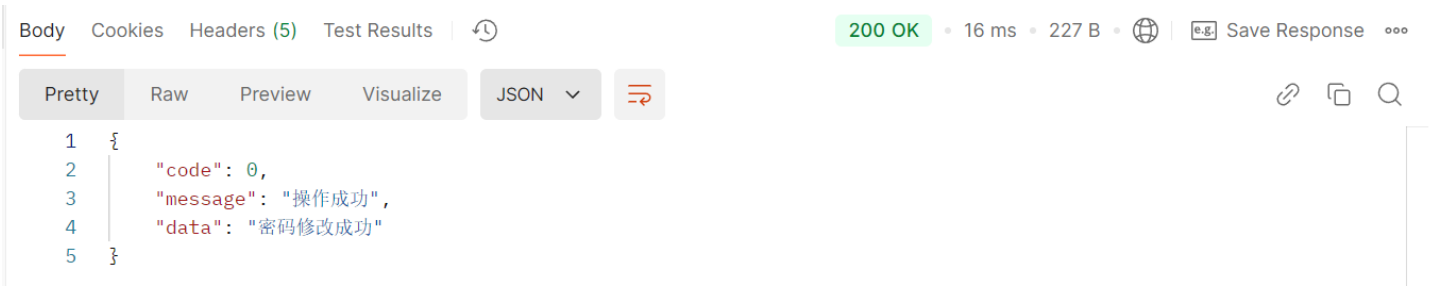
响应参数说明：

参数名称	类型	是否必须	默认值	备注	其他信息
code	number	是		响应码,0成功,1失败	
message	string	否		提示信息	
data	object	否		返回数据	

响应参数样例：

代码块

```
{
    "code":0,
    "message":"操作成功",
    "data":"密码修改成功"
}
```



3.5 查看用户信息

3.5.1 基本信息

请求路径：`/api/user/userInfo`

请求方式：GET

接口描述：该接口用于获取当前已登录用户的详细信息

3.5.2 请求参数

无

3.5.3 响应数据

响应数据类型：application/json

响应参数说明：

参数名称	类型	是否必须	默认值	备注	其他信息

code	number	是		响应码,0成功,1失败	
message	string	否		提示信息	
data	object	是		返回数据	
|-u_id	int	否		用户ID	
|-username	string	否		3~16位非空字符	
|-phone	string	否		11位手机号	

响应数据样例：

代码块

```
{
    "code": 0,
    "message": "操作成功",
    "data": {
        "username": "陈志刚",
        "phone": "14323948005",
        "u_id": 2,
    }
}
```

4. 视频学习接口

4.1 搜索相关视频

4.1.1 基本信息

请求路径：/api/videos?q={keyword}

请求方式：GET

接口描述：搜索学习视频

4.1.2 请求参数

请求参数格式：application/json

请求参数说明：

参数名称	说明	类型	是否必须	备注
keyword	搜索内容	string	是	

请求参数样例：

```
代码块
1  {
2      "keyword":动态规划,
3  }
```

4.1.3 响应数据

响应数据类型：application/json

响应参数说明：

参数名称	类型	是否必须	默认值	备注	其他信息
code	number	是		响应码,0成功,1失败	
message	string	否		提示信息	
data	object	是		返回数据	
|-title	string	是		链接标题	
|-url	string	是		链接地址	

响应数据样例：

```
代码块
1  {
2      "code": 0,
3      "message": "操作成功",
4      "data": [
5          {
6              "title":"动态规划",
7              "url":"https://billbill/dp_learning"
8          },
9          {
10             "title":"递归",
11             "url":"https://billbill/digui_learning"
12         }
```

```
    ]
}
```

5. 编程练习接口

5.1 获取题目列表

5.1.1 基本信息

请求路径：`/api/problems`

请求方式：GET

接口描述：获取题目列表

5.1.2 请求参数

请求参数格式：x-www-form-urlencoded

请求参数说明：

参数名称	说明	类型	是否必须	备注
page	搜索内容	int	是	
per_page	每页数量	int	是	
difficulty	难度	String	否	

请求参数样例：

代码块

```
{
    "page": 2           // 当前页码（默认1）
    "per_page": 10      // 每页数量（默认10）
    "difficulty": "easy"  // 难度（easy/medium/hard）
}
```

5.1.3 响应数据

响应数据类型：application/json

响应参数说明：

参数名称	类型	是否必须	默认值	备注	其他信息
code	number	是		响应码,0成功,1失败	
message	string	否		提示信息	
total	int	是		练习题总数	
problems	Object	是		返回数据	
|-id	int	是		题目id	
|-title	string	是		题目标题	
|-description	string	是		题目描述	
|-tags	string[]	是		标签	
|-difficulty	string	是		难度	
|-submit_count	int	是		提交次数	
|-accept_rate	double	是		通过率	

代码块

```
{
  "code": 0,
  "message": "操作成功"
  "total": 100,
  "problems": [
    {
      "id": 1,
      "title": "所有权练习",
      "description": "修复以下代码...",
      "tags": ["ownership", "basic"],
      "difficulty": "easy",
      "submit_count": 120,
      "accept_rate": 0.75
    }
  ]
}
```

5.2 搜索题目

5.3 提交题目

5.3.1 基本信息

请求路径：`/api/problems/{id}/submit`

请求方式：POST

接口描述：提交题目

5.3.2 请求参数

请求参数格式：application/json

请求头：`Authorization: Bearer {token}`

请求参数说明：

参数名称	说明	类型	是否必须	备注
code	代码主体	txt	是	
language	语言	string	是	

请求参数样例：

代码块

```
{
  "code": "fn main() { ... }",
  "language": "rust" // 预留多语言支持
}
```

5.3.3 响应数据

响应数据类型：application/json

响应参数说明：

参数名称	类型	是否必须	默认值	备注	其他信息
code	number	是		响应码,0成功,1失败	
message	string	否		提示信息	
passed	int	是		通过样例	
total	int	是		总样例	

details	Object	是		详细信息	
|-case_id	string	是		样例id	
|-status	string	是	passed/failed/outof time	测试情况	

响应数据样例：

代码块

```
{
  "code": 0,
  "message": "提交成功",
  "passed": 3,
  "total": 5,
  "details": [
    { "case_id": 1, "status": "passed"},
    { "case_id": 2, "status": "failed"}
  ]
}
```

6. 博客相关接口

6.1 获取博客列表

6.1.1 基本信息

请求路径：`/api/problems/{message}/posts`

请求方式：GET

接口描述：根据题目ID获取相关博客

6.1.2 请求参数

请求参数格式：application/json

请求参数说明：

参数名称	说明	类型	是否必须	备注
message	搜索内容	string	否	
page	当前页	int	是	

per_page	每页数据量	int	是	

请求参数样例：

代码块

```
{
    "message":"动态规划",
    "page": 2
    "per_page": 10
}
```

6.1.3 响应数据

响应数据类型：application/json

响应参数说明：

参数名称	类型	是否必须	默认值	备注	其他信息
code	number	是		响应码,0成功,1失败	
message	string	否		提示信息	
posts	object	是		返回数据	
|-id	int	是		博客ID	
|-title	string	是		博客标题	
|-author	string	是		作者名称	
|-content	txt	是		博客内容	
|-created_at	date	是		博客创建时间	
|-comment_count	int	是		博客评论数	
|-related_problem	int	否		关联题目	

代码块

```
{
```

```
    "code": 0,
    "message":"操作成功",
    "posts": [
      {
        "id": 101,
        "title": "所有权问题详解",
        "author": "user123",
        "content": "题目的解析是......"
        "created_at": "2024-03-20T10:00:00Z",
        "comment_count": 5,
        "related_problem":"小红的书包"
      }
    ]
  }
```

6.2 添加新博客

6.2.1 基本信息

请求路径：`/api/posts`

请求方式：POST

接口描述：添加新博客

6.2.2 请求参数

请求参数格式：application/json

请求头：`Authorization: Bearer {token}`

请求参数说明：

参数名称	说明	类型	是否必须	备注
title	搜索内容	string	是	
content	内容	txt	是	
related_problem	关联题目	string	否	

请求参数样例：

代码块

```
  {
```

```
  "title": "Rust生命周期指南",
  "content": "## 生命周期基础...",
  "related_problem":"小红的书包"
}
```

6.2.3 响应数据

响应数据类型：application/json

响应参数说明：

参数名称	类型	是否必须	默认值	备注	其他信息
code	number	是		响应码,0成功,1失败	
message	string	否		提示信息	
post_id	int	是		博客ID	
created_at	date	是		博客创建时间	

响应数据样例：

代码块

```
{
  "code": 0,
  "message": "操作成功",
  "post_id": 101,
  "created_at": "2024-03-20T10:00:00Z"
}
```

6.3 对博客进行评论

6.3.1 基本信息

请求路径：`/api/posts/{id}/comments`

请求方式：POST

接口描述：在博客下添加评论

6.3.2 请求参数

请求参数格式：application/json

请求头：`Authorization: Bearer {token}`

请求参数说明：

参数名称	说明	类型	是否必须	备注
post_id	博客ID	int	是	
content	评论内容	string	是	

请求参数样例：

```
代码块
1  {
2    "content": "感谢分享！"
3  }
```

6.3.3 响应数据

响应数据类型：application/json

响应参数说明：

参数名称	类型	是否必须	默认值	备注	其他信息
code	number	是		响应码,0成功,1失败	
message	string	否		提示信息	
comment_id	int	是		评论ID(博客ID+评论数)	
created_at	date	是		评论时间	

响应数据样例：

```
代码块
1  {
2    "code": 0,
3    "message": "操作成功",
4    "comment_id": "32-1",
5    "created_at": "2024-03-20T10:05:00Z"
6  }
```

7. AI相关接口

7.1 设置 deepseek token

请求路径 `/api/ai/token`

请求方式：PUT

接口描述：设置接入api的token

请求头： `Authorization: Bearer {token}`

7.1.1 请求参数说明：

参数名称	说明	类型	是否必须	备注
token	接入大模型的token	string	是	

7.2 提问大模型

7.2.1 基本信息

请求路径： `/api/ai/ask`

请求方式：POST

接口描述：向AI发送代码或问题，获取实时流式响应

7.2.2 请求参数

请求参数格式：application/json

请求头： `Authorization: Bearer {token}`

请求参数说明：

参数名称	说明	类型	是否必须	备注
code	编程代码	txt	否	
question	说明	string	是	

请求参数样例：

```
代码块
```

```
{
  "code": "fn main() { ... }", // 可选
  "question": "如何修复这个错误? "
}
```

7.2.3 响应数据

响应数据类型： ~~`text/event-stream`~~ TextStream

响应参数说明：

返回字符串片段

参数名称	类型	是否必须	默认值	备注	其他信息
data	Object	是		返回数据	
|-type	string	是		内容类型	
|-content	txt	是		返回内容	

响应数据样例：

代码块

```
**
解答
:
.
理解
**
我们需要
两个
数的
,
 \(
 +
1
\)
.
进行
运算
**
 \[

```

```
 +
1

2
 \
**
答案
:
**
\[
box
{
2


```

代码块

```
data: {"type": "text", "content": "这里存在所有权问题..."}
data: {"type": "code", "content": "建议修改为: let s = ..."}
```

7.3 获取历史记录（存本地）

7.3.1 基本信息

请求路径：`/api/ai/history`

请求方式：GET

接口描述：获取用户与AI的交互历史

7.3.2 请求参数

请求参数格式：application/json

请求头：`Authorization: Bearer {token}`

请求参数说明：

参数名称	说明	类型	是否必须	备注
id	用户ID	int	是	
page	当前页	int	是	
per_page	每页数据量	int	是	

请求参数样例：

```
代码块
1  {
2      "id":5,
3      "page": 2,
4      "per_page": 10
5  }
```

7.3.3 响应数据

响应数据类型：application/json

响应参数说明：

参数名称	类型	是否必须	默认值	备注	其他信息
code	number	是		响应码,0成功,1失败	
message	string	否		提示信息	
data	object	是		返回数据	
|-question	string	是		提问问题	
|-summary	txt	是		回答文本	
|-timestamp	date	是		提问时间	

响应数据样例：

```
代码块
1  {
2    "history": [
3      {
4        "question": "如何理解Box指针？",
5        "summary": "Box用于堆分配...",
6        "timestamp": "2024-03-20T09:00:00Z"
7      }
8    ]
9  }
```

7.4 与当前编程窗口链接并输出意见

7.4.1 基本信息

7.4.2 请求参数

7.4.3 响应数据

这个功能想集成在5.1提问大模型里，后面再说吧。

8. 数据库设计

核心表结构

1. 用户表 (`**users**`)

代码块

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    phone VARCHAR(100) UNIQUE NOT NULL,
    password_hash VARCHAR(128) NOT NULL, -- 加密后密码
    created_at TIMESTAMPTZ DEFAULT NOW()
);
```

2. 题目表 (`**problems**`)

代码块

```
CREATE TABLE problems (
    id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    description TEXT NOT NULL,
    difficulty VARCHAR(10) CHECK (difficulty IN ('easy','medium','hard')),
    tags JSONB NOT NULL DEFAULT '[]', -- 示例: ["ownership", "async"]
    test_cases JSONB NOT NULL,        -- 输入输出样例对
    created_at TIMESTAMPTZ DEFAULT NOW()
);
```

3. 提交记录表 (`**submissions**`)

代码块

```
CREATE TABLE submissions (
    id SERIAL PRIMARY KEY,
```

```
    user_id INT REFERENCES users(id),  -- 提交用户
    problem_id INT REFERENCES problems(id),
    code TEXT NOT NULL,
    passed INT NOT NULL,     -- 通过数
    total INT NOT NULL,      -- 测试样例总数
    created_at TIMESTAMPTZ DEFAULT NOW()
);
```

4. 博客表（`**posts**`）

代码块

```
CREATE TABLE posts (
    id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(id),
    problem_id INT REFERENCES problems(id), -- 允许NULL（非题解类博客）
    title VARCHAR(255) NOT NULL,
    content TEXT NOT NULL,
    created_at TIMESTAMPTZ DEFAULT NOW()
);
```

5. 评论表（`**comments**`）

代码块

```
CREATE TABLE comments (
    id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(id),
    post_id INT REFERENCES posts(id),
    content TEXT NOT NULL,
    created_at TIMESTAMPTZ DEFAULT NOW()
);
```

6. AI历史记录表（`**ai_history**`）

代码块

```
CREATE TABLE ai_history (
    id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(id),
    question TEXT NOT NULL,
    response_summary TEXT NOT NULL, -- 精简后的回答摘要
    full_data JSONB NOT NULL,       -- 完整交互数据
    created_at TIMESTAMPTZ DEFAULT NOW()
```

```
);
```

修改

代码块

```
CREATE TABLE ai_history (
    id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(id),
    question TEXT NOT NULL, -- 问题
    response TEXT NOT NULL, -- 回答
    --response_summary TEXT NOT NULL, -- 精简后的回答摘要
    --full_data JSON NOT NULL,       -- 完整交互数据
    created_at DATETIME DEFAULT NOW()
);
```

关系示意图

代码块

```
users
  │
  ├── submissions (1:N)
  ├── posts (1:N)
  ├── comments (1:N)
  └── ai_history (1:N)

problems
  │
  ├── submissions (1:N)
  └── posts (1:N)
```