

移动互联网项目文档

- 一、项目介绍
 - 1. 项目功能
 - 2. 交互方式
 - 2.1 前端交互
 - 2.2 后端交互
 - 2.3 网络通信交互
 - 2.4 多线程优化交互
- 二、开发文档
 - 1. 技术选型说明
 - 1.1 前端技术选型
 - 1.2 后端技术选型
 - 1.3 网络通信技术
 - 1.4 数据序列化与反序列化
 - 2. 移动端接口说明
 - 2.1 登录接口
 - 2.2 注册用户接口
 - 2.3 点赞接口
 - 2.4 获取所有新闻接口
 - 2.5 根据类型获取新闻接口
 - 2.6 上传新闻接口
 - 3.多线程优化
 - 4.服务端接口说明
 - 4.1 NewController
 - 4.2 UserController
 - 4.3 OssController
- 三、过程记录文档
 - 1. 协作方式
 - 1.1 版本控制与代码管理
 - 1.2 任务分配与跟踪
 - 1.3 沟通与交流
 - 2. 协作记录

移动互联网项目文档

姓名	学号	分工
孙鲁	221250103	服务端（后端）
翟志阳	221250161	服务端（后端）
曹金	221250163	移动端（前端）
薛嘉伟	221250062	移动端（前端）
张佳辉	211250222	移动端（前端）

一、项目介绍

在当今信息爆炸的时代，人们对于新闻的需求日益增长，同时也面临着海量信息中筛选有效新闻的挑战。为了满足用户随时随地获取新闻的需求，本移动应用应运而生。它适用于多种移动应用设备，无论是智能手机还是平板电脑，都能确保用户在不同场景下都能流畅使用。

本应用的核心功能之一是支持用户发布新闻事件。这不仅为专业媒体人员提供了一个便捷的平台来分享最新的新闻资讯，也为普通用户提供了表达自己观点和分享身边事件的机会。用户可以轻松地创建新闻稿件，通过简单的操作界面输入新闻内容，并附上相关的图片或视频，快速将信息传播给更广泛的受众。

浏览新闻同样是本应用的重要功能。应用采用了高效的新闻聚合技术，能够实时收集来自世界各地的新闻资讯，并将其分类展示给用户。用户打开应用后，即可看到精心编排的新闻列表，涵盖了国内外的重大事件、社会热点、娱乐资讯等多个领域。无论是关注时事政治的用户，还是对娱乐八卦感兴趣的用户，都能在这里找到自己感兴趣的内容。

同时，应用还采用了多模态的交互设计，用户可以通过触摸、滑动、点击等操作，更加便捷地浏览和互动新闻内容。这种沉浸式的阅读体验，不仅满足了用户对信息的需求，也提升了用户对新闻的情感认同和美学体验。

总之，本移动应用以其广泛的设备兼容性、便捷的新闻发布功能、丰富的新闻浏览体验以及个性化的筛选机制，为用户打造了一个全方位、高效能的新闻获取平台。无论用户身处何地，都能通过本应用及时了解全球动态，把握时代脉搏。

1. 项目功能

具体请参考项目演示。

- 用户的创建、登录
- 用户个人信息的查看
- 主页查看新闻
- 发布、创建新闻
- 通过不同类型选择，筛选查看新闻
- 用户可以点赞新闻（不可重复点赞）

2. 交互方式

本项目采用了多种交互方式，以确保用户能够便捷、高效地使用应用，以下是具体的介绍：

2.1 前端交互

- **基于ArkUI框架的声明式布局**：使用ArkUI提供的 `Row`、`Column`、`Image` 等控件，构建出响应式、灵活的界面。例如在新闻列表页面，通过 `Column` 控件垂直排列新闻项，每个新闻项使用 `Row` 控件水平展示新闻标题、发布时间等信息，用户可以通过滑动操作来浏览不同的新闻。这种布局方式使得界面在不同尺寸的设备上都能良好展示，为用户提供了舒适的视觉体验。
- **组件状态管理**：借助ArkTS的 `@State` 装饰器来管理组件的状态。比如在新闻详情页面，新闻内容的加载状态、点赞按钮的选中状态等都通过 `@State` 进行管理。当用户点击点赞按钮

时，组件状态会相应更新，同时UI也会同步变化，如点赞图标变色、点赞数增加等，确保用户操作的即时反馈。

- **多模态交互设计**：融合了触摸、滑动、点击等多种交互方式。用户可以通过触摸屏幕输入文字来发布新闻，通过滑动操作快速切换新闻类别或浏览新闻列表，点击新闻标题进入详情页查看完整内容，还可以点击图片或视频进行预览等。这种多模态的交互设计，让用户能够更加自然、便捷地与应用进行互动，提升了操作的流畅性和趣味性。

2.2 后端交互

- **Spring Boot框架支持的RESTful API**：后端采用Spring Boot框架，开发了一系列RESTful API接口，用于与前端进行数据交互。例如，登录接口 `/api/users/login`，前端通过POST请求发送用户名和密码，后端接收到请求后进行验证，返回登录结果及用户信息；获取新闻列表接口 `/api/news/all`，前端发起GET请求，后端查询数据库后返回所有新闻的数据。这些接口遵循RESTful设计原则，具有清晰的请求方法和路径，使得前后端交互更加规范、高效。
- **Spring Security保障的安全交互**：利用Spring Security框架，对用户认证与授权进行管理。在用户登录时，通过表单登录的方式对用户名和密码进行验证，确保只有合法用户才能访问应用的相应功能。同时，对于一些敏感操作，如发布新闻、点赞新闻等，也会进行权限校验，保障数据的安全性和用户操作的合法性。

2.3 网络通信交互

- **基于HTTP协议的数据交换**：前后端之间通过HTTP协议进行数据交互，符合常见的Web应用开发模式。前端使用 `@kit.NetworkKit` 提供的HTTP请求库，发送GET、POST等请求到后端服务器，并接收服务器返回的响应数据。例如，在获取新闻详情时，前端通过GET请求发送新闻ID，后端查询对应的新闻数据后，以JSON格式通过HTTP响应返回给前端，前端再对数据进行解析和展示。
- **JSON格式的数据序列化与反序列化**：在数据传输过程中，采用JSON格式对数据进行序列化与反序列化。前后端通过 `JSON.parse()` 和 `JSON.stringify()` 等方法，将对象数据转换为JSON字符串进行传输，或将JSON字符串解析为对象数据进行处理。这种轻量级的数据格式，易于阅读和解析，提高了数据交互的效率和兼容性。

2.4 多线程优化交互

- **任务池执行用户注册操作**：在用户注册功能中，引入了多线程优化。通过调用 `taskpool.execute()` 方法，将注册任务提交给任务池，实现异步执行。当用户在前端发起注册请求时，系统会立即响应，不会阻塞主线程，用户可以继续进行其他操作，而注册任务在后台并行处理。注册成功后，会通过回调函数通知用户，并进行相应的页面跳转或提示，提升了应用的响应速度和用户体验。

二、开发文档

1. 技术选型说明

1.1 前端技术选型

- 框架：

HarmonyOS (ArkUI)

- 选择原因：由于项目是面向 HarmonyOS 平台开发，选择了官方的 ArkUI 框架。ArkUI 提供了强大的布局和 UI 控件支持，能够方便地构建跨设备的应用界面，适合开发移动设备及其他设备的应用。
- UI 构建：

使用 ArkUI 提供的声明式布局（如 `Row`，`Column`，`Image` 等控件）来构建响应式、灵活的界面。

- 选择原因：ArkUI 的声明式 UI 编程模型使得构建复杂界面变得更加直观、简洁，同时也具备较高的性能。
- 状态管理：

使用 ArkTS 的 `@State` 装饰器来管理组件的状态。

- 选择原因：`@State` 可以便捷地管理组件状态，保证数据和 UI 的同步。

1.2 后端技术选型

- 框架：

Spring Boot

- 选择原因：Spring Boot 是一种流行的后端开发框架，具有快速开发和自动配置功能，能够支持 RESTful API 的开发，并且有很强的生态系统支持。

- 数据库：

MySQL

- 选择原因：MySQL 是一个成熟的关系型数据库管理系统，适用于存储用户数据、新闻数据等业务数据，具有高性能和易扩展性。

- 安全：

Spring Security

- 选择原因：Spring Security 是一种强大的安全框架，用于管理用户认证与授权，支持各种身份验证机制，如表单登录、JWT 等。

1.3 网络通信技术

- 协议：

HTTP

- 选择原因：前后端之间通过 HTTP 协议进行数据交互，符合常见的 Web 应用开发模式，适合 RESTful API 设计。

- 库：

使用 `@kit.NetworkKit` 提供的 HTTP 请求库来与服务器进行数据交换。

- 选择原因：`@kit.NetworkKit` 提供了简单易用的 API 进行 GET、POST 请求的发送与接收，支持设置超时等请求参数。

1.4 数据序列化与反序列化

- 使用 `JSON` 格式来序列化与反序列化数据，通过 HTTP 接口传输数据。常见的库如 `JSON.parse()` 和 `JSON.stringify()` 被用于处理前后端数据的序列化。

2. 移动端接口说明

2.1 登录接口

- 接口名称: `/api/users/login`
- 请求方法: `POST`
- 请求参数:
 - `username` (string): 用户名
 - `password` (string): 密码
- 返回参数:
 - `status` (string): 请求状态 (`success` 或 `failure`)
 - `message` (string): 错误信息或成功提示
- 功能说明: 该接口用于用户登录, 客户端提交用户名和密码, 服务器返回登录结果。如果登录成功, 返回用户的基本信息; 如果失败, 返回错误提示。

```
1  export async function loginService(username: string, password:
   string): Promise<void> {
2      // 构建请求体
3      const userDto: UserDto = {
4          username: username,
5          password: password,
6      };
7      const url = 'http://192.168.1.22:8080/api/users/login'; // TODO
8
9      const httpRequest = http.createHttp();
10     return await httpRequest.request(url, {
11         method: http.RequestMethod.POST,
12         header: {
13             'Content-Type': 'application/json',
14         },
15         extraData: JSON.stringify(userDto),
16         connectTimeout: 5000,
17         readTimeout: 5000,
18     }).then((response) => {
19         if (response.responseCode !== 200) {
20             throw new Error('Login failed');
21         }
22
23         const result : httpResult = JSON.parse(response.result as string);
24         if (result.code !== "000") {
25
26             throw new Error(result.msg || 'Login failed');
27         }
28
29         // 假设需要保存用户信息到全局上下文//这里有问题
30         console.error(JSON.stringify(result.result));
31         GlobalContext.getContext().setObject('id',
            result.result.id.toString());
```

```
32     GlobalContext.getContext().setObject('user',  
    JSON.stringify(result.result));  
33   });  
34 }
```

2.2 注册用户接口

- 接口名称: `/api/users/register`
- 请求方法: `GET`
- 请求参数:
 - `userId` (string): 用户ID
- 返回参数:
 - `id` (number): 用户ID
 - `username` (string): 用户名
 - `email` (string): 邮箱
 - `role` (string): 用户角色
 - `profileImage` (string): 用户头像URL
- 功能说明: 该接口用于获取当前登录用户的个人信息。同时, 本接口实现了多线程并发。

```
1  @Concurrent  
2  export async function registerService(username: string, password:  
    string): Promise<void> {  
3      // 构建请求体  
4      const userDto: UserDto = {  
5          username: username,  
6          password: password,  
7      };  
8      const url = 'http://192.168.1.22:8080/api/users/register'; // TODO  
9  
10     const httpRequest = http.createHttp();  
11     return await httpRequest.request(url, {  
12         method: http.RequestMethod.POST,  
13         header: {  
14             'Content-Type': 'application/json',  
15         },  
16         extraData: JSON.stringify(userDto),  
17         connectTimeout: 5000,  
18         readTimeout: 5000,  
19     }).then((response) => {  
20         if (response.responseCode !== 200) {  
21             throw new Error('Registration failed');  
22         }  
23  
24         const result : registerResult = JSON.parse(response.result as  
            string);  
25         if (result.code !== "000") {  
26             throw new Error(result.msg || 'Registration failed');  
27         }  
28     });
```

2.3 点赞接口

- 接口名称: `/api/users/like/{newsId}/{userId}`
- 请求方法: `POST`
- 请求参数:
 - 无
- 返回参数:
 - `status` (string): 请求状态 (`success` 或 `failure`)
- 功能说明: 用户对新闻进行点赞, 服务器返回更新后的点赞数。

```
1  async likeNews(newsId: string): Promise<void> {
2      const userId =
GlobalContext.getContext().getObject("id")?.toString();
3      if (!userId){
4          throw new Error("未登录!");
5      }
6      const url = `http://192.168.1.22:8080/api/users/like/` + newsId +
"/" + userId; // TODO
7      const httpRequest = http.createHttp();
8      try {
9          const response = await httpRequest.request(url, {
10             method: http.RequestMethod.POST,
11             header: {
12                 'Content-Type': 'application/json',
13             },
14             connectTimeout: 5000,
15             readTimeout: 5000,
16         });
17         console.error(url)
18         if (response.responseCode !== 200) {
19             throw new Error('点赞失败');
20         }
21     } catch (error) {
22         throw new Error(`点赞请求异常: ${error instanceof Error ?
error.message : '未知错误'}`);
23     }
24 }
25 }
```

2.4 获取所有新闻接口

- 接口名称: `/api/news/all`
- 请求方法: `GET`
- 请求参数:
 - 无
- 返回参数:

- `newsList` (array): 新闻列表，每个新闻对象包含以下字段：
- `id` (number): 新闻ID
- `title` (string): 新闻标题
- `content` (string): 新闻内容
- **功能说明：** 该接口用于获取所有新闻的信息列表。

```

1  async getNewsList(): Promise<NewsData[]> {
2      const url = 'http://192.168.1.22:8080/api/news/all'; // TODO
3
4      const httpRequest = http.createHttp();
5      try {
6          const response = await httpRequest.request(url, {
7              method: http.RequestMethod.GET,
8              header: {
9                  'Content-Type': 'application/json',
10             },
11             connectTimeout: 5000,
12             readTimeout: 5000,
13         });
14
15         if (response.responseCode !== 200) {
16             throw new Error('请求失败');
17         }
18
19         return JSON.parse(response.result as string).newsList;
20     } catch (error) {
21         throw new Error(`请求异常: ${error instanceof Error ? error.message : '未知错误'}`);
22     }
23 }
24

```

2.5 根据类型获取新闻接口

- **接口名称：** `/api/news/type/{type}`
- **请求方法：** `GET`
- **请求参数：**
 - `type` (string): 新闻的类型
- **返回参数：**
 - `newsList` (array): 指定类型的新闻列表
- **功能说明：** 该接口用于根据特定类型获取新闻，便于用户筛选所感兴趣的新闻内容。

```

1  async getNewsListType(type: string): Promise<NewsData[]> {
2      const url = 'http://192.168.1.22:8080/api/news/type/' +
3          encodeURIComponent(type); // TODO
4
5      const httpRequest = http.createHttp();
6
7      try {
8          const response = await httpRequest.request(url, {
9              method: http.RequestMethod.GET,
10             header: {
11                 'Content-Type': 'application/json',
12             },
13             connectTimeout: 5000,
14             readTimeout: 5000,
15         });
16
17         if (response.responseCode !== 200) {
18             throw new Error('请求失败');
19         }
20
21         return JSON.parse(response.result as string).newsList;
22     } catch (error) {
23         throw new Error(`请求异常: ${error instanceof Error ? error.message : '未知错误'}`);
24     }
25 }
26

```



```

5   try {
6       const response = await httpRequest.request(url, {
7           method: http.RequestMethod.GET,
8           header: {
9               'Content-Type': 'application/json',
10            },
11            connectTimeout: 5000,
12            readTimeout: 5000,
13        });
14
15        if (response.responseCode !== 200) {
16            throw new Error('请求失败');
17        }
18
19        return JSON.parse(response.result as string).newsList;
20    } catch (error) {
21        throw new Error(`请求异常: ${error instanceof Error ? error.message : '未知错误'}`);
22    }
23 }
24 }

```

2.6 上传新闻接口

- **接口名称:** `/api/news/create`
- **请求方法:** `POST`
- **请求参数:**
 - `newsData` (object): 新闻信息对象, 包含以下字段:
 - `title` (string): 新闻标题
 - `content` (string): 新闻内容
- **返回参数:**
 - `status` (string): 请求状态 (success 或 failure)
- **功能说明:** 该接口用于上传新闻信息, 客户端将新闻数据提交到服务器, 服务器返回上传结果。

```

1   async uploadNews(newsData: NewsData): Promise<void> {
2       const url = 'http://192.168.1.22:8080/api/news/create'; // TODO
3
4       const httpRequest = http.createHttp();
5       return httpRequest.request(url, {
6           method: http.RequestMethod.POST,
7           header: {
8               'Content-Type': 'application/json',
9           },
10          extraData: JSON.stringify(newsData),
11          connectTimeout: 5000,
12          readTimeout: 5000,

```

```

13     }).then((response) => {
14         if (response.responseCode !== 200) {
15             throw new Error('Upload failed');
16         }
17
18         const result = JSON.parse(response.result as string);
19         if (result.status !== "success") {
20             throw new Error(result.message || 'Upload failed');
21         }
22     });
23 });
24 }

```

3.多线程优化

使用多线程（通过任务池）来执行用户注册操作，具体来说，是调用 registerService 函数进行用户注册。以下是对这段代码的详细解析：

多线程任务执行

taskpool.execute(...):

这里使用了一个鸿蒙中的 taskpool 的对象，其 execute 方法用于将任务（即一个函数和相应的参数）提交给任务池。任务池通常用于管理异步任务的执行，允许多个操作并行进行，提高应用程序的响应速度与性能，而不会阻塞主线程。

代码逻辑

开始注册:

用户在前端发起注册请求，调用 taskpoolExecuteregister 方法。

方法内部首先调用 taskpool.execute(registerService, username, password)，将 registerService 函数及其所需的 username 和 password 作为参数传递。

```

1  taskpoolExecuteregister(username: string, password: string) {
2      taskpool.execute(registerService, username, password).then(() => {
3          this.isRegistering = false;
4          showToast('注册成功');
5          router.back();
6      })
7      .catch((err: string) => {
8          console.error("taskPoolTest test occur error: " + err);
9          this.isRegistering = false;
10         showToast('注册失败，请重试');
11     });
12 }

```

4.服务端接口说明

按照控制器层面进行分类说明：

4.1 NewController

- 创建新闻

```
1  @PostMapping("/create")
2  public ResultVO<Boolean> addNew(@RequestBody NewVO newVO ){
3      return ResultVO.buildSuccess(newService.createNew(newVO));
4  }
```

- 删除新闻:

```
1  @PostMapping("/delete/{newId}")
2  public ResultVO<Boolean> deleteNew(@PathVariable("newId") Integer
id){
3      return ResultVO.buildSuccess(newService.deleteNew(id));
4  }
```

- 获取新闻各方法

```
1  @GetMapping("/all")
2  public ResultVO<List<NewVO>> getAllNews(){
3      return ResultVO.buildSuccess(newService.getAllNews());
4  }
5
6  //获取某个新闻
7  @GetMapping("/{newId}")
8  public ResultVO<NewVO> getNewById(@PathVariable("newId") Integer
id){
9      return ResultVO.buildSuccess(newService.getNewById(id));
10 }
11
12 //获取不同类型的新闻
13 @GetMapping("/type/{type}")
14 public ResultVO<List<NewVO>> getNewsByType(@PathVariable("type")
NewType type){
15     return ResultVO.buildSuccess(newService.getNewsByType(type));
16 }
17
```

4.2 UserController

- 创建用户

```
1  @PostMapping("/register")
2  public ResultVO<Boolean> register(@RequestBody UserVO userVO )
throws Exception {
3      return ResultVO.buildSuccess(usersService.register(userVO));
4  }
```

- 登录

```

1      @PostMapping("/login")
2      public ResultVO<UserVO> login(@RequestBody UserVO userVO) throws
Exception {
3          return ResultVO.buildSuccess(userService.login(userVO));
4      }

```

- 获取及编辑用户信息

```

1      @PostMapping("/info")
2      public ResultVO<UserVO> getUserInfoById(@RequestBody UserVO userVO)
{
3          return
ResultVO.buildSuccess(userService.getUserInfoById(userVO.getId()));
4      }
5      @PostMapping
6      public ResultVO<Boolean> updateInformation(@RequestBody UserVO
userVO){
7          return
ResultVO.buildSuccess(userService.updateInformation(userVO));
8      }

```

- 点赞新闻相关

```

1      //用户点赞新闻
2      @PostMapping("/like/{newId}/{userId}")
3      public ResultVO<Boolean> likeNew(@PathVariable("newId") Integer
newId, @PathVariable("userId") Integer userId){
4          return ResultVO.buildSuccess(userService.likeNew(newId,
userId));
5      }
6
7      //用户取消点赞新闻
8      @PostMapping("/unlike/{newId}/{userId}")
9      public ResultVO<Boolean> unlikeNew(@PathVariable("newId") Integer
newId, @PathVariable("userId") Integer userId){
10         return ResultVO.buildSuccess(userService.cancelLikeNew(newId,
userId));
11     }
12
13     //获取用户创建的新闻
14     @GetMapping("/user/{userId}")
15     public ResultVO<List<NewVO>> getNewsByUser(@PathVariable("userId")
Integer userId){
16         return
ResultVO.buildSuccess(userService.getNewsByUserCreated(userId));
17     }
18
19     //获取用户点赞的新闻
20     @GetMapping("/like/{userId}")
21     public ResultVO<List<NewVO>> getLikeNews(@PathVariable("userId")
Integer userId){

```

```

22         return ResultVO.buildSuccess(userService.getLikeNews(userId));
23     }

```

- 收藏相关

```

1         //获取用户收藏的新闻
2         @GetMapping("/collect/{userId}")
3         public ResultVO<List<NewVO>>
4         getCollectNews(@PathVariable("userId") Integer userId){
5             return
6             ResultVO.buildSuccess(userService.getCollectNews(userId));
7         }
8
9         //用户收藏新闻
10        @PostMapping("/collect/{newId}/{userId}")
11        public ResultVO<Boolean> collectNew(@PathVariable("newId") Integer
12        newId, @PathVariable("userId") Integer userId){
13            return ResultVO.buildSuccess(userService.collectNew(newId,
14            userId));
15        }
16
17        //用户取消收藏新闻
18        @PostMapping("/uncollect/{newId}/{userId}")
19        public ResultVO<Boolean> uncollectNew(@PathVariable("newId")
20        Integer newId, @PathVariable("userId") Integer userId){
21            return
22            ResultVO.buildSuccess(userService.cancelCollectNew(newId, userId));
23        }

```

4.3 OssController

- 上传文件到阿里云以获得url

```

1         @PostMapping("/upload")
2         public String uploadFile(@RequestParam("file") MultipartFile file)
3         {
4             try {
5                 // 生成文件名
6                 String fileName = file.getOriginalFilename();
7                 if (fileName == null || fileName.isEmpty()) {
8                     return "文件名不能为空";
9                 }
10
11                // 生成唯一的对象名称，防止文件名冲突
12                String objectName = System.currentTimeMillis() + "_" +
13                fileName;
14
15                // 上传文件并获取URL

```

```
14         String url = ossUtil.upload(objectName,
file.getInputStream());
15         return url;
16     } catch (Exception e) {
17         e.printStackTrace();
18         return "上传失败: " + e.getMessage();
19     }
20 }
```

三、过程记录文档

1. 协作方式

1.1 版本控制与代码管理

1. 使用Git进行版本控制

- 本项目采用Git作为版本控制系统，所有代码变更都通过Git进行跟踪和管理。团队成员在本地克隆项目仓库后，进行代码开发。每次代码修改后，成员需先在本地进行测试，确保代码的正确性和稳定性。
- 在提交代码前，成员需执行 `git pull` 命令，拉取最新的代码变更，解决可能出现的冲突。冲突解决后，再次进行本地测试，确认无误后，通过 `git add` 添加变更文件，使用 `git commit` 提交本地变更，并附上清晰、简洁的提交信息，说明本次变更的内容和目的。
- 最后，执行 `git push` 命令，将本地分支的变更推送到远程仓库，以便其他团队成员能够获取最新的代码版本。

1.2 任务分配与跟踪

1. 任务分解与分配

- 项目开始前，团队成员共同参与需求分析和设计阶段，将整个项目分解为多个具体的功能模块和任务。根据每个成员的专业技能、经验和工作量等因素，将任务分配给相应的成员。
- 任务分配后，成员需对所负责的任务进行进一步的细化，明确任务的具体要求、预期成果和时间节点。

2. 任务跟踪与进度更新

- 团队成员在工作交流群中定期汇报各自任务的进展情况，包括已完成的工作、遇到的问题和下一步计划。同时，通过项目管理工具实时更新任务的状态，如“进行中”、“已完成”、“阻塞”等，方便团队成员随时查看项目的整体进度。
- 对于遇到的问题，团队成员需及时在项目管理工具中记录问题详情，并@相关成员寻求帮助。问题解决后，及时更新问题的状态和解决方案，确保信息的透明和共享。

1.3 沟通与交流

1. 日常沟通

- 团队成员通过即时通讯工具（如QQ、微信等）进行日常的沟通交流。在开发过程中，成员之间可以随时提问、讨论技术问题或分享经验。对于一些紧急的问题，可以通过语音或视频通话的方式进行快速沟通，提高解决问题的效率。

- 每天随时进行沟通，分享自己的工作进展、遇到的问题以及需要协助的地方，确保团队成员对项目动态有清晰的了解。

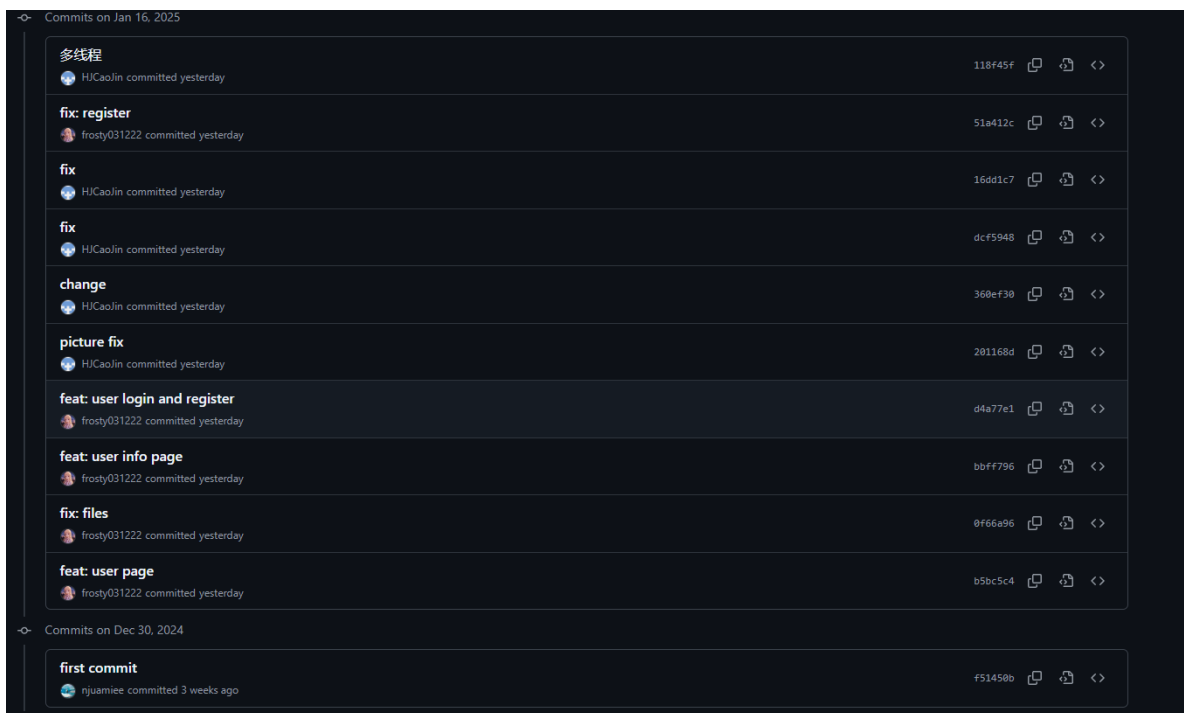
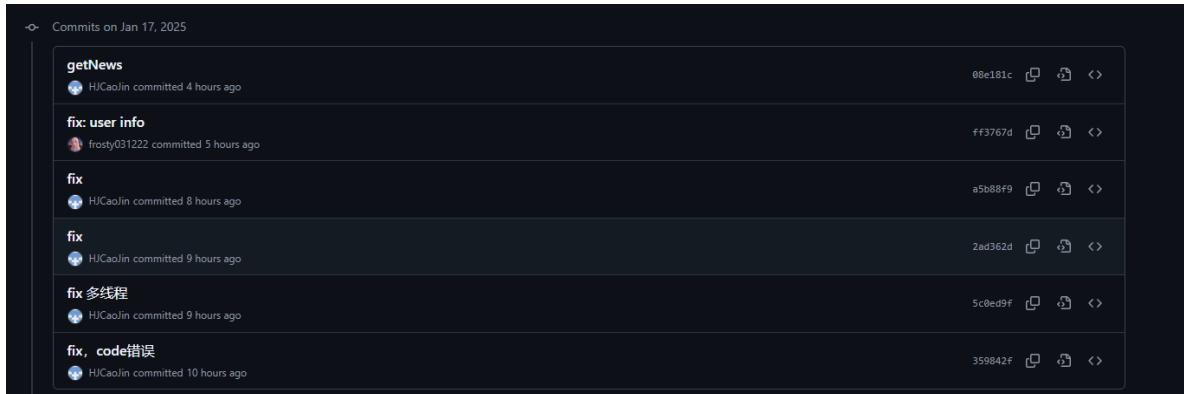
2. 文档共享与知识沉淀

- 鼓励团队成员在解决技术难题或完成重要功能后，将相关的经验、技巧和知识点整理成文档，分享到团队的知识库中，方便其他成员学习和参考，促进团队整体技术水平的提升。

2. 协作记录

选取git部分记录如下图：

- 移动端：



- 后端：

Commits on Jan 16, 2025		
修改login接口	jiahuike committed yesterday	b02e8ca
fix	HJCaoJin committed yesterday	1f8e6a3
Commits on Jan 6, 2025		
feat achieve encipher	njuamiee committed 2 weeks ago	155f77d
feat achieve encipher	njuamiee committed 2 weeks ago	876e31d
Commits on Jan 5, 2025		
feat: achieve basic function of the user, new, comment, like and collect	njuamiee committed 2 weeks ago	ac5e3cd
Commits on Jan 4, 2025		
start	njuamiee committed 2 weeks ago	6d9eeaf
Commits on Dec 30, 2024		
first commit	njuamiee committed 3 weeks ago	6a1d8d7

Commits on Jan 17, 2025		
feat:likecount	elloLeon committed 46 minutes ago	67b1938
feat:likecount	elloLeon committed 1 hour ago	f89e508
fix:newCreate类型转换	elloLeon committed 3 hours ago	f7eeddb
fix:newCreate类型转换	elloLeon committed 4 hours ago	91b0aaf
fix	HJCaoJin committed 4 hours ago	d0b2c60
fix	HJCaoJin committed 8 hours ago	1d2f2a4
fix	HJCaoJin committed 8 hours ago	7bae209
fix:改变查询方法传参方式为body	elloLeon committed 9 hours ago	8158d42
feat	elloLeon authored and elloLeon committed 9 hours ago	449203c
fix:改变查询方法传参方式为body	elloLeon committed 9 hours ago	d17caa5
feat:提供上传文件(图片)获得url的接口;添加按照id获取用户信息的接口	elloLeon committed 9 hours ago	29d71ed
fix:修改登陆方法的返回类型	elloLeon committed 10 hours ago	bd4143e

工作交流记录：

移动互联网小组 (5)



曹金 LV8

曹金

不

各位目前的任务大概分为如下 后端，阿里云接口，...

各位这些大家各自说一下自己要做哪些，否则现在太乱了

LV10 群主 Elon



后端都我来吧 我连上github了

LV10 群主 Elon



就是前端需要后端什么接口在群里跟我说



曹金 LV8

阿里云的那个你先弄一下，否则没法传图片

LV10 群主 Elon



ok



张佳辉 LV5

我刚才试了下，后端这么改就不被拦截了，但还是登录失败，应该是前端有点问题



张佳辉 LV5

```
@PostMapping("@v"/"login") 新*
public ResultV0<String> login(@RequestBody UserV0 userV0 ) throws Exception {
    return ResultV0.buildSuccess(userService.login(userV0.getUsername(), userV0.ge
}
```



曹金 LV8

登录的时候后端有报错吗？



曹金 LV8

没有报错的话，就是登录成功了。



张佳辉 LV5

无



曹金 LV8

前端那只是个提示。



冷冰冰

我发现你根本没给这个方法写 url。。现在补上了@Leno

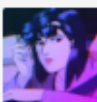


冷冰冰

```

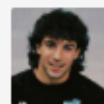

[Info] 4.0.0.0
[UserVo] goldenInfoPyId,StatusInfo,Isset 0
[7] traceVoInfo,Isset 0, goldenInfoPyId

```

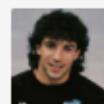


曹金

@Leno 我这边上传新闻类型给你发汉语，你那边解析一下，翟志阳数据库用的是英文。



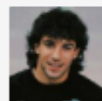
是那样就会映射到默认的这一层
路径就是api/users 你改好能调用
就行



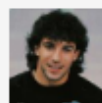
冷冰冰：我发现你根本没给这个方法写url。。现在补上了@Leno

13:55

嗷



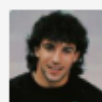
我不知道你那边传的对象到我这是啥



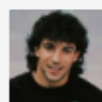
冷冰冰

```
{  
  id: 10  
}
```

这个body需要一个类型，你前端不用指定这个类型吗



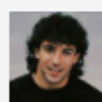
嘶



冷冰冰

这样的东西

我查查怎么搞



如果body里面只有id这一个属性的话 我还要在po vo里面专门加一个只有id属性的类来接受这种结构体

