****

Web应用开发 实验报告

|  |  |
| --- | --- |
| **学 院** | 网络空间安全学院 |
| **专 业** | 网络工程 |
| **班 级** | 23272412 |
| **组长姓名** | 颜璟颢 |
| **组员姓名** | 颜璟颢 |
| **教师姓名** | 胡伟通 |
| **完成日期** | 6.14 |
| **成 绩** |  |

**实验三 综合实验**

1. **实验目的和要求**
2. 实验目的
   1. 掌握RESTful设计准则，能按照前后端分离的开发形式来实现简单的用户注册和登录；
   2. 掌握通过Email来实现4位数字形式的验证码的发送，为以后通过短信发送验证码打下基础；
   3. 掌握业界主流的JWT认证功能。
3. **实验要求**：
4. 在实验二的基础上，增加Email形式的注册功能和相应的登录功能。注册部分，仿照手机端 APP 的短信验证码，通过邮件发这个验证码（比如4位的随机数字），最后是将携带验证码的注册信息发到服务器上（使用第三方组件<如：https://github.com/jordan-wright/email>来进行邮件发送功能）；
   * + - 1. 这个短信可以存在数据库，也可以存在Redis。
5. 重置密码，也是仿照手机端 APP 密码重置过程，通过邮件发送重置码，然后重置信息中包含重置码，最后你将重置信息发到服务器上；
6. 包含对某个物品（自己选择）的操作，以RESTful API风格进行；
7. 选择使用JWT认证（JSON Web Tokens）的小组，有加分（可以使用gtoken组件）；

（5）**不要求提供前端方面的实现，即不需要做网站或APP（如果有的话，更好）。**此外，每组还需要对所写的API进行测试（比如：在Apipost上进行），并进行截图，把截图贴到实验报告。

可以参考：https://www.wolai.com/vXZppyj54kvxQmGCJXKvn4

1. **实验内容**

根据**实验要求中的每条规定**，进行实验。

本实验是在实验二基础上，进一步实现一个基于 RESTful 架构风格的 Web 后端系统，采用 **Go 语言 + GoFrame 框架** 完成综合性开发，完成用户注册、登录、重置密码、资源管理等功能，重点实现邮箱验证码注册流程与 JWT 登录认证机制。

**1.技术栈：**

|  |  |  |
| --- | --- | --- |
| **模块** | **技术/组件** | **说明** |
| **Web 框架** | [GoFrame](https://goframe.org/) | Go 语言高性能企业级开发框架，项目主框架 |
| **数据库 ORM** | GoFrame 内置 ORM + gorm | 数据表映射，简化数据库操作 |
| **验证码邮件发送** | github.com/jordan-wright/email | SMTP 邮件发送功能库，用于发送验证码邮件 |
| **验证码存储** | Redis | 存储验证码，设置过期时间防止重复利用 |
| **JWT认证** | gtoken / github.com/golang-jwt/jwt/v5 | 实现用户登录状态管理和身份验证（可加分） |
| **接口设计** | RESTful API | 遵循 RESTful 规范，接口资源清晰分层 |
| **接口测试工具** | APIPost | 用于接口调试、请求测试、抓包验证 |

**2.主要实验功能：**

（1）邮箱验证码发送

用户注册或重置密码时，系统通过 email.NewEmail() 创建邮件内容，并使用 SMTP 协议将验证码发送到用户邮箱，实现邮箱验证码的自动发送功能。

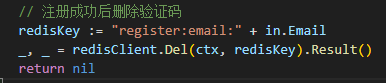


（2）验证码存储，校验，删除

验证码发送后，系统将验证码以 register:email:xxx 或 reset:email:xxx 的形式存储在 Redis 中，并设置 5 分钟有效期。用户注册或重置密码时，系统会从 Redis 获取并校验验证码的正确性和有效性。为安全考虑，注册用户和重置密码校验验证码成功后从Redis库删除验证码。

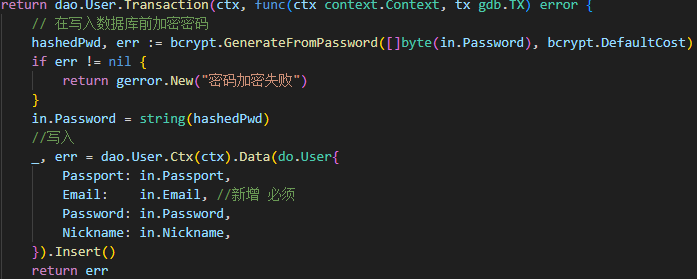




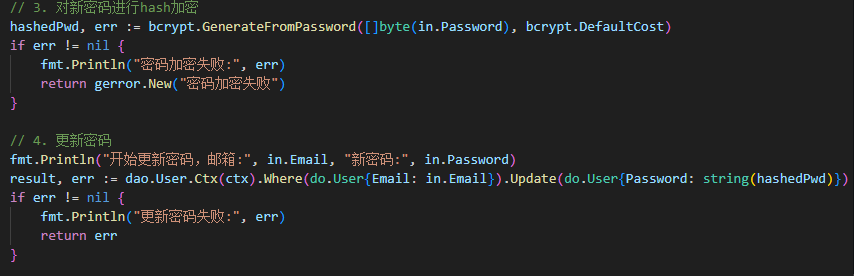


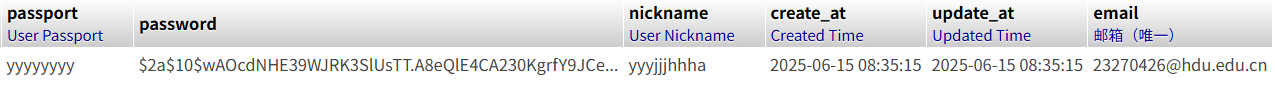
1. 密码hash加密功能：

为保证用户信息安全，在用户注册、修改密码、重置密码时，写入数据库之前对密码进行 hash 加密。



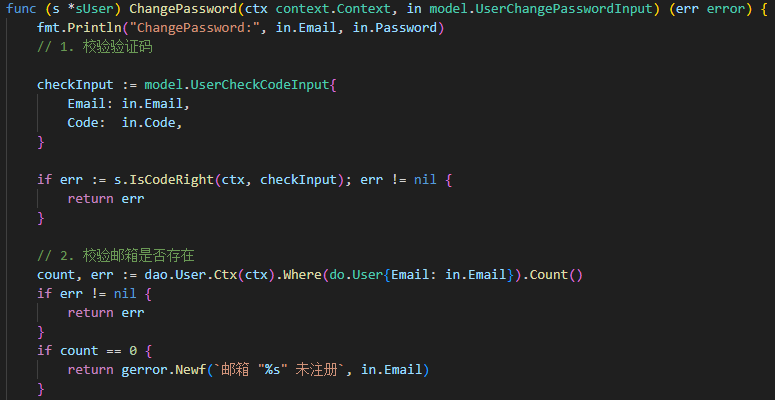






（4）密码重置功能

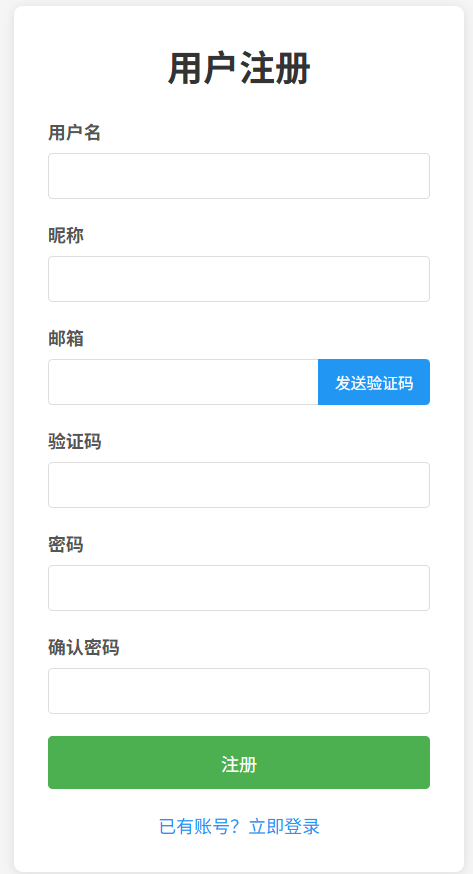
用户忘记密码时，可通过邮箱验证码验证身份，提交新密码，系统校验通过后更新数据库中的用户密码，实现安全的密码重置流程。

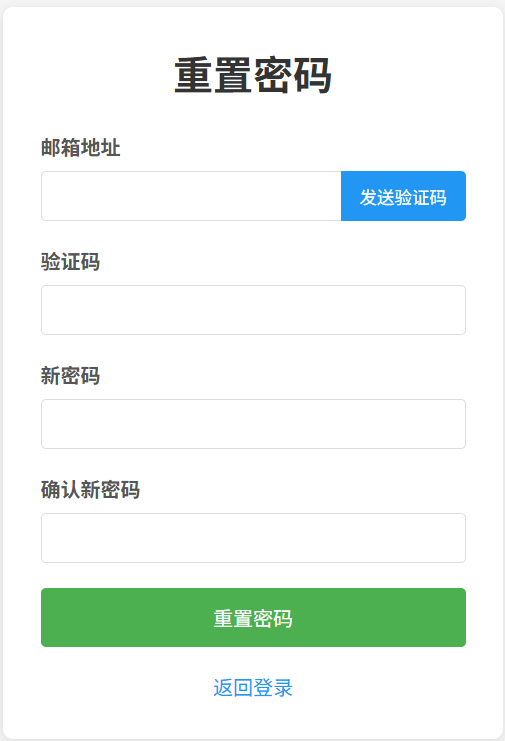


（5）简单前端演示

对 user 表的用户注册、登录、重置密码功能，设计了极简的前端页面（见 frontend 目录下的 register.html、index.html、reset-password.html），并用原生 JavaScript 实现了基本的表单交互和接口调用，方便进行功能演示和手动测试。

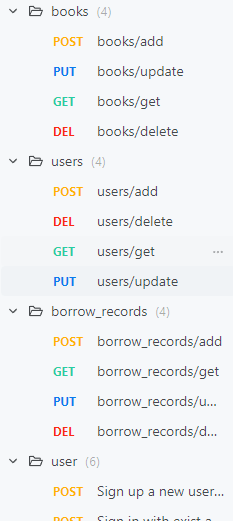






（6）用户操作接口，接口测试与验证

系统实现了用户注册、登录、重置密码、用户信息查询等接口，所有接口均采用 RESTful API 设计风格，支持其他表等数据的增删改查操作。通过 APIPost 工具对各接口进行功能测试，并结合数据库查询验证接口操作的正确性和数据一致性。



（7）JWT 用户认证

用户登录成功后，系统生成并颁发 JWT（JSON Web Token），前端保存该 Token。后续所有需要身份认证的接口，前端需在请求头中携带 Token，后端通过中间件校验 Token 的有效性，实现无状态的用户认证机制。

通过本实验，进一步掌握了前后端分离的 Web 后端 API 设计、邮件服务集成、缓存服务使用（Redis）、JWT身份认证机制，以及 RESTful 设计规范与接口测试过程。

1. **程序设计思路**

本实验在实验二的基础上，主要进一步完善了用户注册与密码管理功能，主要包括邮箱验证码的发送、存储、校验与删除机制，以及密码重置功能与密码加密存储机制。系统采用分层架构设计，代码结构清晰，逻辑合理，便于维护与扩展。

**一、邮箱验证码模块设计**

1. 功能目标

本模块用于完成用户注册、密码重置过程中的邮箱验证码生成与校验功能，确保邮箱所有权验证的安全性，防止恶意注册与篡改。

2. 设计方案

（1）验证码生成

采用 rand.Intn(9000) + 1000 生成 4 位随机数字，避免出现非数字字符，提高用户识别度。

随机数种子初始化 rand.Seed() 在程序启动时执行，确保每次生成不同验证码。

（2）验证码发送

使用 github.com/jordan-wright/email 邮件发送库，通过 SMTP 协议连接 QQ邮箱服务器（smtp.qq.com:587） 实现邮件发送。

邮件主题和正文格式设计简洁明确，例如：

您的验证码是：1234，有效期5分钟，请勿泄露。

（3）验证码存储

验证码采用 Redis 缓存存储，键名格式为：

register:email:{用户邮箱}

有效期设置为 5 分钟，超过时间后自动失效，防止验证码滥用。

（4）验证码校验与删除

在用户提交验证码进行注册或重置密码时，系统从 Redis 获取对应邮箱的验证码进行匹配。

匹配成功后，Redis 中该验证码立即删除，防止验证码被重复利用。

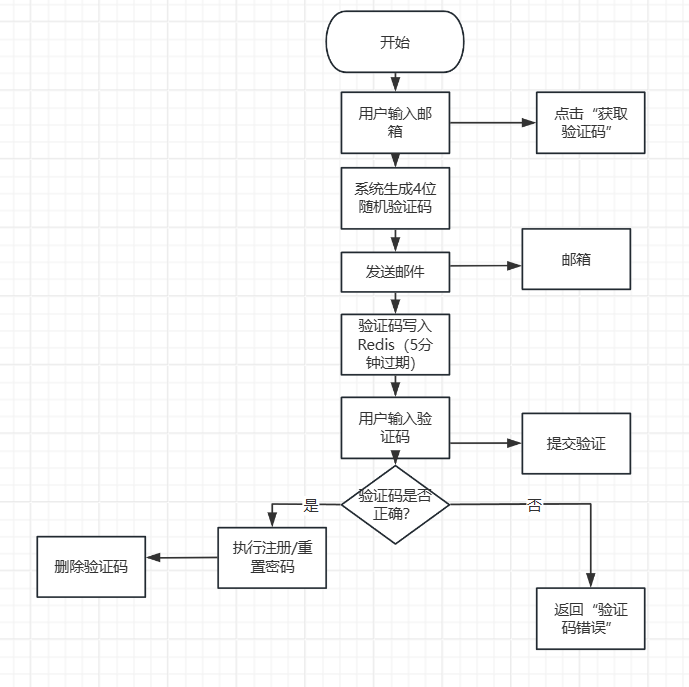
匹配失败 或验证码不存在时，提示“验证码错误或已过期”。

（5）安全性设计

限制验证码有效时间，防止长时间暴露风险。

防止暴力请求，可进一步添加 IP 限制或冷却机制作为扩展。

建议后续加入图形验证码防止机器人滥用接口。



**二、密码重置与密码加密模块设计**

1. 功能目标

实现通过邮箱验证码验证身份后，允许用户重置登录密码。为保障安全，所有密码均采用不可逆加密算法存储，防止明文泄露。

2. 设计方案

（1）密码重置流程

用户输入邮箱 → 请求验证码 → 邮箱接收验证码 → 验证成功后 → 输入新密码完成重置。

（2）密码加密

采用 bcrypt 加密算法，将用户输入密码转化为哈希值存储。

采用 bcrypt.DefaultCost 加密强度，保障安全性的同时保持性能可接受。

bcrypt 特点：

单向不可逆 → 数据库泄露也无法还原密码

自动生成盐值 → 同一密码多次加密结果不同 → 防止彩虹表攻击

（3）重置逻辑实现

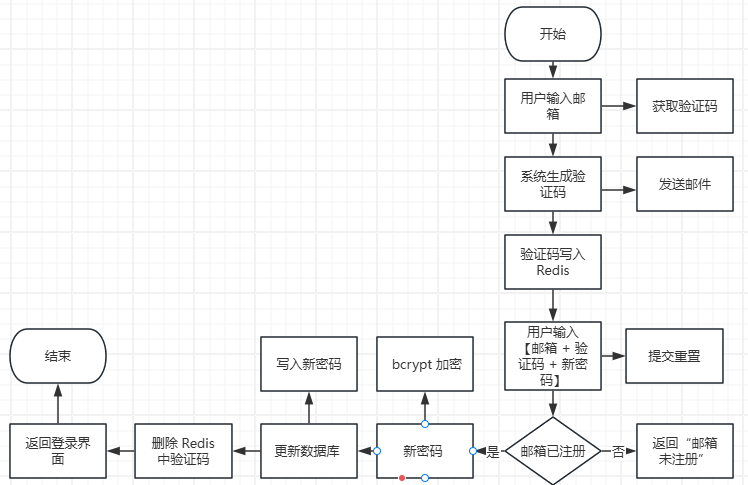
校验验证码： 使用统一的验证码校验逻辑 IsCodeRight()。

校验邮箱是否存在： 在数据库中验证邮箱对应账号是否存在，避免无效重置。

加密新密码： 使用 bcrypt.GenerateFromPassword([]byte(新密码), bcrypt.DefaultCost) 生成哈希密码。

更新数据库： 将哈希密码存入数据库 users 表对应记录中。

删除验证码： 重置成功后，主动删除 Redis 中该验证码。



**三、jwt认证流程设计**

整个认证流程分为以下几个环节：

用户登录： 用户提供账号（或邮箱）和密码登录。

生成 Token： 服务器验证用户身份后，生成包含用户 ID、过期时间等信息的 JWT Token。

返回 Token： 将生成的 Token 返回给客户端，由客户端本地保存。

请求携带 Token： 客户端在后续请求中通过 Authorization 请求头携带 Token。

Token 验证： 后端通过中间件验证 Token 的有效性，并解析其中的用户信息。

授权访问： 验证通过后，允许访问受保护资源；验证失败则返回未授权错误提示。

2. 系统架构设计

本项目专门设计了 JWT 服务模块 负责 Token 的生成、解析和验证，接口定义如下：

type IJwt interface {

GenerateToken(ctx context.Context, userId int) (string, error) // 生成Token

ParseToken(ctx context.Context, token string) (\*model.JwtClaims, error) // 验证解析Token

RefreshToken(ctx context.Context, oldToken string) (string, error) // 刷新Token

InvalidateToken(ctx context.Context, token string) error // 加入黑名单

}

##### **Token数据结构**

type JwtClaims struct {

UserId int `json:"uid"` // 用户ID

ExpiresAt int64 `json:"exp"` // 过期时间戳

IssuedAt int64 `json:"iat"` // 签发时间

Issuer string `json:"iss"` // 签发者

}

此外，通过 JWT 中间件 实现接口保护机制，负责从请求中提取 Token → 验证合法性 → 将用户 ID 注入请求上下文。

3. 用户服务与 JWT 的集成

SignIn： 用户登录成功后生成 Token 并返回客户端。

IsSignedIn： 验证当前请求是否包含有效 Token。

4. 安全性设计

为保障系统安全性，JWT 模块采取以下措施：

密钥保护： Token 使用安全密钥进行签名，防止伪造。

过期时间控制： Token 设置合理过期时间，降低被盗用风险。

HTTPS 传输： 确保所有认证信息通过 HTTPS 安全传输，防止中间人攻击。

最小权限原则： Token 中仅包含必要的用户标识信息，避免敏感数据泄露。

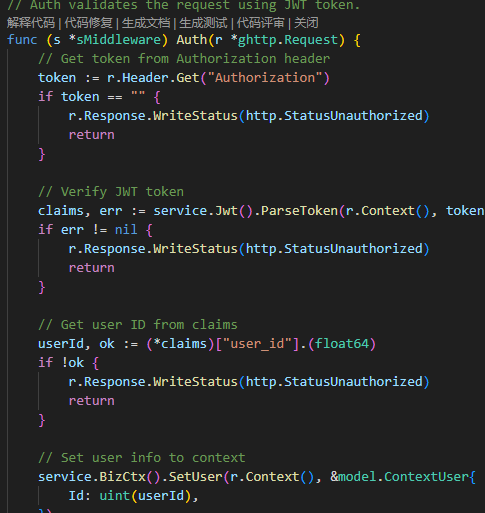
5. 技术实现

本项目使用 github.com/golang-jwt/jwt/v5 库实现 JWT 的生成与解析。Token 结构包含：

Header： 指定算法（如 HS256）

Payload： 包含用户 ID、过期时间等

Signature： 使用密钥对前两部分签名，防止篡改



**四、设计合理性与安全性分析**

1.验证码采用 Redis 缓存，具有高效读写性能和自动过期特性，适合临时数据。

2.验证码通过 邮件发送 符合现代系统邮箱验证机制，安全性高。

3.验证码删除机制 防止被二次利用，保障注册和重置密码流程安全可靠。

4.密码存储采用 bcrypt 加密，单向不可逆，防止数据库泄露带来的安全风险。

5.系统整体具备良好的安全性和可扩展性，后续可进一步增加验证码请求频率限制、防刷机制等高级防护功能。

1. **程序源代码**

**注意源代码要有详细的注释。同学们提交的每个程序都应该遵循Honor Code（诚实代码保证）的要求。**

**请大家特别注意一定要在每个程序首部的注释中加上以下保证：**

**# 我真诚地保证：**

**# 我自己独立地完成了整个程序从分析、设计到编码的所有工作。**

**# 如果在上述过程中，我遇到了什么困难而求教于人，那么，我将在程序实习报告中**

**# 详细地列举我所遇到的问题，以及别人给我的提示。**

**# 在此，我感谢 XXX, …, XXX对我的启发和帮助。下面的报告中，我还会具体地提到**

**# 他们在各个方法对我的帮助。**

**# 我的程序里中凡是引用到其他程序或文档之处，**

**# 例如教材、课堂笔记、网上的源代码以及其他参考书上的代码段,**

**# 我都已经在程序的注释里很清楚地注明了引用的出处。**

**# 我从未没抄袭过别人的程序，也没有盗用别人的程序，**

**# 不管是修改式的抄袭还是原封不动的抄袭。**

**# 我编写这个程序，从来没有想过要去破坏或妨碍其他计算机系统的正常运转。**

**# 颜璟颢**

1. **源代码（包含数据库表设计代码）**

**详细代码见附件，**放在压缩包中跟本文档一起上传。

1. **程序测试方法及测试结果记录（不能光截图，要有相应的文字说明）**
2. **测试方法**

本实验采用黑盒测试与接口功能测试相结合的方法，对新增的邮箱验证码功能、用户注册登录、密码重置等功能模块进行全面测试。

主要测试工具及方法如下：

1.接口测试工具：

使用 APIPOST 软件构建请求、发送数据、查看响应结果，对所有后端接口进行功能性测试。

通过 APIPOST 自动记录测试用例，便于后续维护和验证。

2. 邮箱接收验证：

使用 QQ邮箱 作为接收端，手动查收邮件，验证验证码邮件是否准确送达。

3. Redis 数据验证：

使用 Redis-cli 工具，通过命令：

get register:email:{邮箱地址}

查看验证码是否成功写入 Redis，并验证过期时间及删除逻辑。

4. 数据库验证（MySQL）：

使用 phpMyAdmin 工具，手动查询 users 表，验证注册信息是否正确写入数据库。

查看密码字段是否为 hash 加密 数据，确认密码加密逻辑是否生效。

5.功能验证范围：

功能正常场景：验证码发送、注册、登录、密码重置

异常场景：验证码错误、验证码过期、邮箱未注册、重复注册、密码不一致等，确保系统在错误情况下也能正常响应并给出提示。

6.密码安全性验证：

验证数据库中密码为加密后的哈希字符串，防止明文存储风险。

1. **测试流程**

1.先利用APIPOST软件对后端主要新功能(验证码发送)进行测试



查看邮箱，看看是否能收到邮件



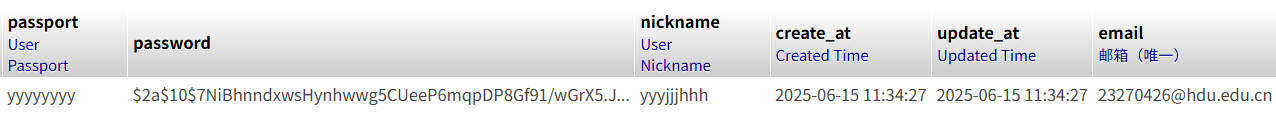
打开redis服务，并get register:email:23270426@hdu.edu.cn查看发送的邮箱查看redis是否写入数据，成功，则说明成功发送并写入，



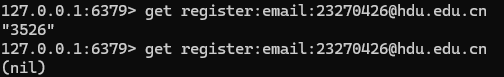
2.再设计前端，调用后端的接口，测试能否具体实现功能（注册，登录，重置密码）



填入接收的验证码，自动跳到登录页面，同时查看数据库，比照数据，核对是否符合预期，



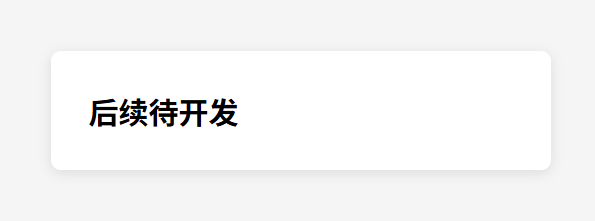
此时在看看Redis，测试是否成功删除验证码数据，



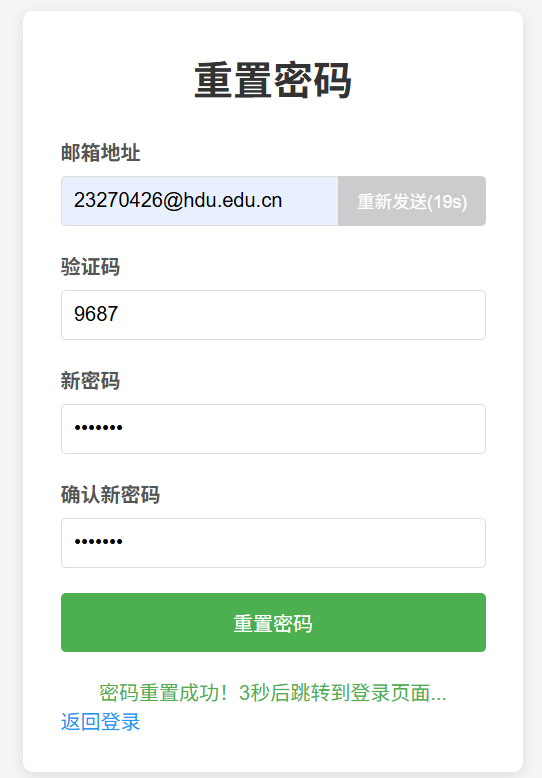
随后直接测试登录，测试邮箱和账号是否都能登录

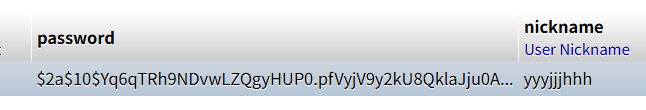


成功登录，页面跳转：

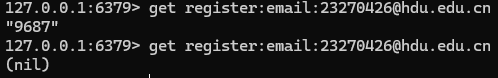


最后点击找回密码提示，测试重置密码功能





成功后查看Redis数据库内数据是否已删除



再比对密码是否成功修改，随后用新密码再尝试登录。最后确定功能都能实现。

**注意：本文档只展示成功部分，报错功能（输错密码、验证码，输未注册邮箱，重复注册邮箱等）也同时测试，确认了功能的正确性和完整性**

1. **实验分析总结及心得（该部分也是评分的一个重点）**

本次综合实验基于前两个实验的成果，进一步实现了邮箱验证码注册登录机制、密码重置功能以及 JWT 用户认证模块。通过本次实验，我不仅加深了对 Web 后端开发流程的理解，更通过遇到的问题与反复调试，锻炼了独立思考与动手能力。

1. 实验分析总结

（1）项目结构认知进一步深化

在实验过程中，我逐步掌握了 GoFrame 框架的整体项目结构和职责划分，尤其是 api → controller → service → logic → dao 各层之间的关系。从初期不熟悉项目目录，到可以主动规划接口层级，逐渐建立了系统性思维。

（2）接口设计与 RESTful 规范的掌握

实验初期接口命名不规范，常使用 /user/send-email-code 这类带动词路径，通过查阅 RESTful 设计原则和调整路由结构，改为 /email-codes 等资源型接口，使接口更符合行业标准，也提升了接口测试与维护效率。

（3）遇到的典型问题与解决过程

本次实验过程中，遇到的最大挑战是邮箱验证码无法发送与 Redis 无法写入问题。

问题 1：邮箱验证码发送失败

现象： 邮箱验证码接口调用后未收到邮件。

分析过程：

初步排查代码逻辑 → SMTP 地址、端口配置正常

检查日志 → 发现 535 authorization failed

查阅资料 → 原因是 QQ 邮箱需要单独获取授权码 → 普通邮箱密码无效

解决方案：

登录 QQ 邮箱 → 开启 IMAP/SMTP → 获取授权码 → 替换代码中的密码 → 成功发送邮件。

问题 2：Redis 无法写入验证码

现象： 邮箱验证码接口执行后 Redis 内部无对应键值

分析过程：

使用 redis-cli 检查连接 → 报错 connectex: actively refused

确认防火墙状态 → Redis 端口未放行

解决方案：

临时关闭 Windows 防火墙 → 重新启动 Redis → 验证写入成功

后续配置 Redis 绑定本地接口 → 避免网络冲突

问题 3：密码加密导致登录失败

现象： 注册成功，但登录时提示密码错误

分析过程：

查看数据库 → 密码字段为 bcrypt hash

登录逻辑仍采用明文比对 → 逻辑冲突

解决方案：

引入 bcrypt.CompareHashAndPassword() 方法 → 改为使用 hash 比对 → 登录恢复正常

问题 4：JWT Token 无法识别

现象： 登录成功后携带 Token 请求 → 返回未授权

分析过程：

检查请求头 → Authorization 格式正确

查看中间件逻辑 → Token 解析失败

解决方案：

查阅 JWT 结构与签名逻辑 → 发现是密钥不一致 → 统一配置密钥 → 测试通过

2. 个人心得

独立完成本次实验，让我深刻体会到“开发不是写代码那么简单，更是一个持续排查和解决问题的过程”。 每一个错误背后，都是知识体系中尚未完善的部分。

在这个过程中，我从最初对 GoFrame 框架的陌生，到能够独立设计接口、规范参数结构、优化项目目录、实现复杂逻辑，整个过程是逐步“填坑”的过程。遇到错误时，我学会了冷静思考，从日志入手分析问题，从原理上找到原因，再查阅官方文档或社区资料，寻找解决方案。

特别是通过这次实验，我进一步认识到：

细节决定成败，接口路径拼写、参数类型、Redis 键名都可能成为 bug 根源

安全意识不可忽视，密码加密、JWT Token 等不仅是功能实现，更是系统安全的保障

系统设计比代码本身更重要，一个结构合理的系统能大幅降低后期维护难度

这次实验不仅是技术能力的提升，更是心态和思维方式的进化。

3. 总结

通过本次实验，我全面掌握了 RESTful API 设计规范、邮件验证码机制、Redis 缓存、密码加密、JWT 无状态认证机制 等主流后端开发技术。同时，也积累了大量调试经验，进一步提升了独立开发、独立排查问题的能力。

未来的学习和项目开发中，我会继续保持严谨细致的工作态度，加强原理学习，不断完善知识体系，为后续更复杂的项目开发打下坚实基础。

每一次错误，都是一次宝贵的成长。