软件工程实训(中级)课程报告

|  |  |
| --- | --- |
| **学院/系：** | 软件工程学院 |
| **专业：** | 软件工程 |
| **完成时间：** | 2023/7/5 |
| **姓名：** | 黄炜尧  李佶  陈达佳  梁珏鸣  刘松凌  陈冠洲 |
| **学号(顺序一致)：** | 21311124  21311135  21311263  21311471  21311509  21311122 |
| **项目名称：** | iTab新标签页 |

1. **项目背景介绍**

**①项目背景**

**(1)立题背景**

随着界面个性化的用户需求越来越广泛，支持自定义的产品也变得越来越受欢迎。那些不支持用户自定义化的产品总是会被用户吐槽其在布局上的“毒点”；与此同时，我们也注意到，浏览器自带的旧标签页在用户真正需求的内容方面仅具有最基础的功能，无法完全满足广大用户的日常使用需求；同时又额外添加了许多用户不需要的功能，这也无法提供给用户更好的使用体验。由此，我们决定开发一个支持用户自定义的浏览器新标签页插件，代替浏览器自带的旧标签页。

**(2)技术背景**

在当今互联网上，虽然已有很多款由网页开发大佬做出的iTab、Atom、OmniTab等新标签页插件，但并没有一款真正受到广大用户好评推荐的新标签页插件。究其原因，是这些新标签页插件在支持一定程度上自定义的同时，还强制添加了许多冗余功能，如未经用户允许，内置了许多广告链接、视频平台、小游戏、短文小说网站——新标签页插件还未完全成熟，就已经迈开了商业化的脚步。

**(3)立题目标**

我们小组的目标为：开发一款更符合用户需求的新标签页插件，使其既切中要害，准确满足用户的需求，又不携带冗余的功能，尽可能保持新标签页的简洁。

**(4)关键词**

我们小组所开发的项目特点的关键词为：**功能齐全，页面简洁，高度自定义。**

**②需求分析**

本项目需要实现的功能主要有：

**(1)用户登录注册：**用户通过指定昵称以及密码创建新账号，该账号具有唯一ID；注册成功后用户可以进行登录，并根据已有设置使个人主页得到相应调整；

**(2)用户个性化设置：**用户登录后通过点击设置按钮可对个人主页进行个性化设置，包括但不限于：按钮颜色及透明度、预设主题颜色及透明度、使用壁纸；

**(3)集成搜索引擎：**搜索引擎的功能作为基本功能可以在任何情况下(包括未登录)使用，用户在搜索栏选择所提供的搜索引擎并键入搜索内容即可跳转至相应搜索结果页面;

**(4)保存搜索记录、显示搜索推荐：**用户最近8条搜索记录可以被自动保存，并且用户可以设置是否启用搜索推荐功能——键入关键词时进行自动补全；

**(5)调用各大资源网站api：**页面提供各大网站的热搜等信息，用户可直接点击进行跳转访问链接；

**(6)用户收藏网址、编辑备忘信息：**用户可以保存希望收藏的网页URL，以便进行下次访问；同时，用户能够编辑个人备忘录，保存更多个人信息。

**③基本功能**

**(1)新用户注册：**前端获取用户自定义的用户名及密码，后端确认密码二次输入正确后生成唯一10位以下ID，并将默认设置内容填充至数据库的user表内；

**(2)现存用户登录：**前端获取用户登录所提供的ID及密码并发送至后端，后端根据信息查找数据库是否有此用户，若有则将用户登录所需的各种信息发送至前端，若无则用状态码400进行标识，前端根据状态码进行相应的处理；

**(3)用户保存设置：**用户每次点击设置按钮并退出后会将当前的设置信息发送至后端，后端将按照用户ID在数据库进行更新信息操作；

**(4)利用搜索引擎进行检索：**用户在页面输入关键字并查找，可以跳转至所使用的搜索引擎的搜索结果界面，同时用户可以选择所使用的搜索引擎，例如baidu、bing、google；

**(5)搜索记录保存：**根据用户的设置，用户最多能够保存最近的8条检索记录，可以通过后端保存至数据库中；

**(6)智能搜索建议：**用户键入检索关键字的同时，后端调用对应的api获取相关搜索推荐并推送给前端，前端将搜索建议显示在搜索栏下方，用户可以直接点击搜索建议，跳转至相应检索页面；

**(7)用户上传图片作为壁纸：**用户将个人拥有的图片上传至前端，前端将编码结果上传至后端，后端将编码结果转为图片，并将对应路径保存在数据库中，用户下次登录将继续使用个人上传的图片；

**(8)用户收藏网页URL：**用户可以在主页添加所收藏的网页URL，方便下次访问，后端根据添加的URL信息在数据库中保存，作为用户所需参数进行传输；

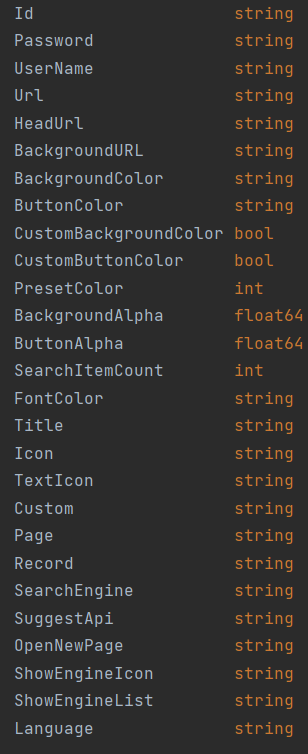
**(9)显示各大平台热搜等信息：**后端调用其他平台的api，并将对应的信息以及url发送至前端，经过处理后用户可直接点击对应的信息跳转至相应页面。

**④数据库设计**

**(1)数据库名：**demo

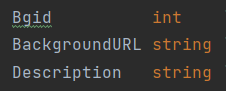
**(2)数据库表名：**user，presetbackground

**(3)user表设计：**表中字段名及其类型如下所示：



该表存放了用户的所有相关信息，包括ID、密码、设置信息、壁纸所在的服务器路径、头像所在的服务器路径、搜索记录、收藏的URL等等。字段id、password、user\_name、url、head\_url、background\_url、background\_color、button\_color、custom\_background\_color、custom\_button\_color、preset\_color、background\_alpha、button\_alpha、search\_item\_count、font\_color、title、icon、text\_icon、custom、page、record，分别表示用户的ID、密码、昵称、收藏url、头像对应的url、背景对应的url、背景颜色、按钮颜色、默认背景颜色、是否默认按钮颜色、预设颜色、字体颜色、搜索历史等。

**(4)presetbackground设计：**表中字段名及其类型如下所示：



该表存放了服务器提供的网络壁纸，供用户选择，包含字段bgid、description、background\_url，分别表示预设背景的id、该预设背景的描述、该预设背景的url。

**⑤工作任务分解、人员分工**

**前端：**黄炜尧、陈达佳

**后端：**梁珏鸣、李佶

**测试：**刘松凌、陈冠洲

**⑥知识点应用**

**(1)前端**

**①Vue.js：**Vue.js是一个流行的JavaScript框架，用于构建用户界面。Vue.js采用了组件化的设计思想，可以让开发者将页面分解为多个独立的组件，从而提高代码的可重用性和可维护性。在这个项目中，Vue.js被用来构建前端用户界面，包括各种页面和组件。

**②Vuex：**Vuex是Vue.js官方的状态管理库，用于管理应用程序中的共享状态。在这个项目中，Vuex被用来存储前端应用程序的状态数据，例如用户信息、页面数据等。Vuex提供了一种集中式的方式来管理这些数据，使得不同的组件可以方便地共享和更新这些数据。

**③HTML/CSS/JavaScript：**HTML、CSS和JavaScript是Web开发的基础技术，用于创建网站和应用程序的用户界面和交互。在这个项目中，HTML用于定义网页的结构和内容，CSS用于定义网页的样式和布局，JavaScript用于实现网页的交互和动态效果。

**(2)后端**

**①Golang：**一种编译型、并发型、静态类型编程语言，用于编写高性能的Web应用程序。在这个项目中，Golang被用来编写后端Web服务的代码，包括路由处理、数据处理、身份验证等。Golang的并发机制和高效的内存管理使得它在处理高并发和大规模数据时表现出色。

**②Gin：**一种Golang的Web框架，用于简化Web应用程序的开发。Gin提供了路由、中间件、参数绑定等常用功能的封装，使得开发者可以更加专注于业务逻辑的实现。在这个项目中，Gin被用来处理服务器端的路由和中间件，使得请求可以被正确地路由到对应的处理函数中。

**③GORM：**一种Golang的ORM（对象关系映射）库，用于简化与数据库的交互。GORM提供了一种面向对象的方式来操作数据库，使得开发者可以更加方便地进行数据的增删改查等操作。在这个项目中，GORM被用来操作数据库，包括定义数据模型、创建和管理数据库表、执行查询等。

**(3)其他的知识点**

**①RESTful API：**一种设计Web API的架构风格，用于创建可伸缩、可维护和可重用的Web服务。RESTful API的设计原则包括使用HTTP协议定义资源、使用标准的HTTP方法定义操作、使用URI表示资源、使用JSON等格式传输数据等。在这个项目中，RESTful API被用来定义前后端之间的数据交互接口，包括请求和响应的格式、HTTP方法的使用等。

**②JSON：**一种轻量级的数据交换格式，用于前后端数据的传输和解析。JSON具有易读易写、易解析的特点，因此被广泛应用于Web应用程序的数据交换。在这个项目中，JSON被用来作为前后端数据交换的格式，使得前后端可以方便地传输和解析数据。

**③HTTP协议：**Web通信的基础协议，用于在客户端和服务器之间传输数据。HTTP协议定义了请求和响应的格式、状态码、请求方法等，是Web应用程序的基础。在这个项目中，HTTP协议被用来传输前后端之间的数据和请求。

**④数据库设计和管理：**用于设计数据库模式、创建和管理数据库表、索引和查询等。在这个项目中，数据库设计和管理被用来存储和管理应用程序中的数据，包括用户信息、订单信息、商品信息等。这个项目使用了GORM作为ORM库，使得开发者可以通过面向对象的方式来操作数据库。

**⑤身份验证和授权：**用于保护Web应用程序和API免受未经授权的访问和操作。在这个项目中，身份验证和授权被用来保护Web应用程序和API免受未经授权的访问和操作。

1. **核心代码介绍**

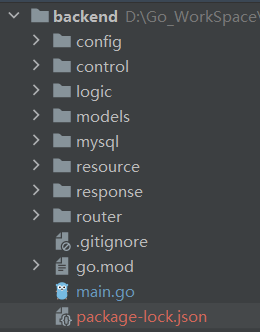
**①前端**

**(1)项目结构**

**②后端**

**(1)项目结构**

Go项目结构如下图所示：



其中，**/config**保存了配置信息，读取yaml文件中的数据库等配置信息以便完成与数据库的连接；**/control**用于编写后端负责接收的http请求并作出相关的处理；**/logic**编写路由组中的各种路由所需的方法；**/models**保存与json格式的数据相对应的结构体，用于上传/接收这种格式的数据；**/mysql**完成具体的与mySQL数据库的连接；**/resource**保存服务器能够提供的各种图片资源；**/response**封装服务器的相应方法；**/router**声明了各种对应的路由，相应前端的相关请求。

**(2)核心代码**

**①负责连接数据库的main.go:**

|  |
| --- |
| var (      cfgFile string      rootCmd = &cobra.Command{}  )  func initConfig() {      config.MustInit(os.Stdout, cfgFile) // 配置初始化  }  func init() {      cobra.OnInitialize(initConfig)      rootCmd.PersistentFlags().StringVar(&cfgFile, "config", "config/dev.yaml", "config file (default is $HOME/.cobra.yaml)")      rootCmd.PersistentFlags().Bool("debug", true, "开启debug")      viper.SetDefault("gin.mode", rootCmd.PersistentFlags().Lookup("debug"))  }  func main() {      if err := Execute(); err != nil {          log.Println("start fail:", err.Error())          os.Exit(-1)      }  }  func Execute() error {      rootCmd.RunE = func(cmd \*cobra.Command, args []string) error {          \_, err := mysql.Init(              viper.GetString("db.hostname"),              viper.GetInt("db.port"),              viper.GetString("db.username"),              viper.GetString("db.password"),              viper.GetString("db.dbname"),          ) // 用viper将对应的参数取出来          if err != nil {              return err          }          mysql.DB.AutoMigrate(&models.User{}, &models.PresetBackground{}) // 将数据库的表自动映射为User          defer mysql.DB.Close() // 最后别忘了把连接关了          r := router.SetupRouter() // 初始化路由          err = r.Run(":2020")          if err != nil {              return err          }          port := viper.GetString("port")          log.Println("port = \*\*\* =", port)          return http.ListenAndServe(port, nil) // listen and serve      }      return rootCmd.Execute()  } |

其中在Execute()中将/models中的User以及PresetBackground结构体将数据库中的两个表进行绑定

**②response.go提供三种对前端的响应：**

|  |
| --- |
| func Response(ctx \*gin.Context, httpStatus int, code int, data gin.H, msg string) {      ctx.JSON(httpStatus, gin.H{"code": code, "data": data, "msg": msg})  }  func Success(ctx \*gin.Context, data gin.H, msg string) {      Response(ctx, http.StatusOK, 200, data, msg)  }  func Fail(ctx \*gin.Context, data gin.H, msg string) {      Response(ctx, http.StatusOK, 400, data, msg)  } |

**③封装了各种需要用于对前端和数据库进行传输的数据的models：**

|  |
| --- |
| // 用户基本属性  type User struct {      gorm.Model      Id                    string  `json:"id,string" db:"id"`      Password              string  `json:"password" db:"password"`      UserName              string  `json:"username" db:"user\_name"`      Url                   string  `json:"url" db:"url"`      HeadUrl               string  `json:"avatar" db:"head\_url"`      BackgroundURL         string  `json:"backgroundURL" db:"background\_url"`      BackgroundColor       string  `json:"backgroundColor" db:"background\_color"`      ButtonColor           string  `json:"buttonColor" db:"button\_color"`      CustomBackgroundColor bool    `json:"customBackgroundColor" db:"custom\_background\_color"`      CustomButtonColor     bool    `json:"customButtonColor" db:"custom\_button\_color"`      PresetColor           int     `json:"presetColor" db:"preset\_color"`      BackgroundAlpha       float64 `json:"backgroundAlpha" db:"background\_alpha"`      ButtonAlpha           float64 `json:"buttonAlpha" db:"button\_alpha"`      SearchItemCount       int     `json:"searchItemCount" db:"search\_item\_count""`      FontColor             string  `json:"fontColor" db:"font\_color"`      Title                 string  `json:"title" db:"title"`      Icon                  string  `json:"icon?" db:"icon"`      TextIcon              string  `json:"textIcon" db:"text\_icon"`      Custom                string  `json:"custom" db:"custom"`      Page                  string  `json:"page" db:"page"`      Record                string  `json:"record" db:"record"`      SearchEngine          string  `json:"searchEngine" db:"search\_engine"`      SuggestApi            string  `json:"suggestApi" db:"suggest\_api"`      OpenNewPage           string  `json:"openNewPage" db:"open\_new\_page"`      ShowEngineIcon        string  `json:"ShowEngineIcon" db:"show\_engine\_icon"`      ShowEngineList        string  `json:"showEngineList" db:"show\_engine\_list"`      Language              string  `json:"language" db:"language"`  }  type PresetBackground struct {      Bgid          int    `json:"id" db:"bgid"`      BackgroundURL string `json:"backgroundURL" db:"background\_url"`      Description   string `json:"description" db:"description"`  }  // RegisterForm 注册请求参数  type RegisterForm struct {      UserName        string `json:"username"`      Password        string `json:"password"`      ConfirmPassword string `json:"confirmPassword"`  }  // LoginForm 登录请求参数  type LoginForm struct {      Id       string `json:"id"`      Password string `json:"password"`  }  // 设置请求参数  type SettingForm struct {      Id                    string  `json:"Id" db:"id"`      BackgroundColor       string  `json:"backgroundColor" db:"background\_color"`      BackgroundAlpha       float64 `json:"backgroundAlpha" db:"background\_alpha"`      ButtonColor           string  `json:"buttonColor" db:"button\_color"`      ButtonAlpha           float64 `json:"buttonAlpha" db:"button\_alpha"`      CustomBackgroundColor bool    `json:"customBackgroundColor" db:"custom\_background\_color"`      CustomButtonColor     bool    `json:"customButtonColor" db:"custom\_button\_color"`      PresetColor           int     `json:"presetColor" db:"preset\_color"`      FontColor             string  `json:"fontColor" db:"font\_color"`      SearchItemCount       int     `json:"searchItemCount" db:"search\_item\_count""`  }  // 销号请求参数  type DeleteForm struct {      Id string `json:"Id" db:"id"`  }  // 上传图片请求参数  type UploadImageForm struct {      Id          string `json:"id" db:"id"`      ImageData   string `json:"imageData"`      Description string `json:"description" db:"description"`  }  // 修改密码请求参数  type ChangePasswdForm struct {      Id     string `json:"Id" db:"id"`      NewPwd string `json:"newPassword" db:"password"`  }  // 修改昵称请求参数  type ChangeUserNameForm struct {      Id      string `json:"Id" db:"id"`      NewName string `json:"userName" db:"user\_name"`  }  // 上传URL请求参数  type NewURLForm struct {      Title    string `json:"title" db:"title"`      Url      string `json:"url" db:"url"`      Icon     string `json:"icon" db:"icon"`      TextIcon string `json:"textIcon" db:"text\_icon"`      Custom   string `json:"custom" db:"custom"`      Page     string `json:"page" db:"page"`  }  // 上传搜索记录请求参数  type NewRecordForm struct {      Record string `json:"record" db:"record"`  }  // 百度搜索建议参数  type BaiduSuggestionResponse struct {      Query string   `json:"q"`      P     bool     `json:"p"`      Data  []string `json:"s"`  }  // 必应搜索建议参数  type BingSuggestionResponse struct {      AS struct {          Query       string `json:"Query"`          FullResults int    `json:"FullResults"`          Results     []struct {              Type     string `json:"Type"`              Suggests []struct {                  Txt  string `json:"Txt"`                  Type string `json:"Type"`                  Sk   string `json:"Sk"`                  HCS  int    `json:"HCS"`              } `json:"Suggests"`          } `json:"Results"`      } `json:"AS"`  }  type SortedBingSuggestionResponse struct {      Txt []string `json:"Txt"`  }  // 谷歌搜索建议参数  type Suggestion struct {      Data string `xml:"suggestion>data"`  }  type GoogleSuggestionResponse struct {      Suggestions []Suggestion `xml:"CompleteSuggestion"`  }  type GetAPI struct {      Code int    `json:"code"`      Msg  string `json:"msg"`      Data []struct {          Rank    int    `json:"rank"`          Name    string `json:"name"`          Viewnum string `json:"viewnum"`          Url     string `json:"url"`      } `json:"data"`      Debug    string  `json:"debug"`      ExecTime float64 `json:"exec\_time"`      UserIp   string  `json:"user\_ip"`      LogId    string  `json:"log\_id"`  }  // 数据表名字  func (User) TableName() string {      return "user"  }  func (PresetBackground) TableName() string {      return "presetBackground"  } |

**④定义路由组以及允许前端域名访问的router.go:**

|  |
| --- |
| // 用户路由组  func UserRouterInit(r \*gin.RouterGroup) {      userManager := r.Group("/user")      {          userManager.POST("/register", control.RegisterHandler)          userManager.POST("/login", control.LoginHandler)          userManager.POST("/settings", control.Settings)          userManager.POST("/deleteAccount", control.DeleteAccount)          userManager.POST("/uploadBackground", control.UploadBackground)          userManager.POST("/uploadAvatar", control.UploadAvatar)          userManager.POST("/changePassword", control.ChangePasswd)          userManager.POST("/changeUserName", control.ChangeUserName)          userManager.POST("/newURL/:userId", control.PostNewURL)          userManager.POST("/record/:userId", control.PostRecord)          userManager.GET("/myURL/:userId", control.GetMyURL)          userManager.GET("/image/:imageIndex", control.OnImagesGet)          userManager.GET("/web-images", control.WebImages)          userManager.GET("/baiduSuggestion", control.GetBaiduSuggestion)          userManager.GET("/bingSuggestion", control.GetBingSuggestion)          userManager.GET("/googleSuggestion", control.GetGoogleSuggestion)          userManager.GET("/getMoreInfo", control.GetMoreInfo)          userManager.Use(logic.AuthMiddleware())      }  }  func SetupRouter() \*gin.Engine {      router := gin.Default()      // 添加CORS中间件      config := cors.DefaultConfig()      config.AllowOrigins = []string{"http://localhost:8080", "http://localhost:8081", "http://localhost:8082", "http://localhost:8083",          "http://localhost:8084", "http://localhost:8085"} // 允许访问的域名      config.AllowMethods = []string{"GET", "POST", "PUT", "DELETE", "OPTIONS"} // 允许的HTTP方法      router.Use(cors.New(config))      api := router.Group("")      UserRouterInit(api)      //NewsRouterInit(api)      //CommentRouterInit(api)      return router  } |

**⑤具体定义路由组所需要的各种方法的/control/user.go：**

|  |
| --- |
| // 注册  func RegisterHandler(c \*gin.Context) {      var userRegister \*models.RegisterForm      node, err := snowflake.NewNode(1)      if err != nil {          panic(err)      }      if err := c.ShouldBind(&userRegister); err != nil {          response.Fail(c, nil, "注册失败!")      } else {          //判断用户名是否存在          var findUser models.User          result := mysql.DB.Where("user\_name=?", userRegister.UserName).First(&findUser)          if result.Error == nil {              response.Fail(c, nil, "用户名已被使用!")          } else if result.Error != nil {              var newUser = models.User{                  Id:                    strconv.FormatUint(uint64(node.Generate().Int64()%10000000000), 10),                  Password:              userRegister.Password,                  UserName:              userRegister.UserName,                  Url:                   "https://baidu.com/",                  HeadUrl:               "http://localhost:2020/user/1",                  BackgroundURL:         "http://localhost:2020/user/1",                  ButtonColor:           "#ffffff",                  BackgroundColor:       "#ffffff",                  CustomButtonColor:     true,                  CustomBackgroundColor: true,                  FontColor:             "black",                  PresetColor:           100,                  BackgroundAlpha:       0.8,                  ButtonAlpha:           0.8,                  SearchEngine:          "Baidu",                  SuggestApi:            "Baidu",                  OpenNewPage:           "true",                  ShowEngineIcon:        "ture",                  ShowEngineList:        "true",                  Language:              "Auto",              }              mysql.DB.Create(&newUser)              response.Success(c, gin.H{"id": newUser.Id}, "注册成功!")          }      }  }  // 登录  func LoginHandler(c \*gin.Context) {      var userLogin \*models.LoginForm      if err := c.ShouldBind(&userLogin); err != nil {          response.Fail(c, nil, "登录失败")      } else {          var findUser models.User          result := mysql.DB.Where("id=?", userLogin.Id).First(&findUser)          if result.Error != nil {              response.Fail(c, nil, "用户不存在!")          } else if result.Error == nil {              // 查找密码              if findUser.Password != userLogin.Password {                  response.Fail(c, nil, "密码错误!")              } else {                  findUser.Url = strings.Replace(findUser.Url, "{\"url\": \"", "", -1)                  findUser.Url = strings.Replace(findUser.Url, "\"}", "", -1)                  response.Success(c, gin.H{                      "userName":              findUser.UserName,                      "backgroundColor":       findUser.BackgroundColor,                      "backgroundAlpha":       findUser.BackgroundAlpha,                      "buttonColor":           findUser.ButtonColor,                      "buttonAlpha":           findUser.ButtonAlpha,                      "customBackgroundColor": findUser.CustomBackgroundColor,                      "customButtonColor":     findUser.CustomButtonColor,                      "presetColor":           findUser.PresetColor,                      "fontColor":             findUser.FontColor,                      "avatar":                findUser.HeadUrl,                      "backgroundURL":         findUser.BackgroundURL,                      "url":                   findUser.Url,                      "title":                 findUser.Title,                      "icon":                  findUser.Icon,                      "textIcon":              findUser.TextIcon,                      "custom":                findUser.Custom,                      "page":                  findUser.Page,                      "record":                findUser.Record,                      "searchEngine":          findUser.SearchEngine,                      "suggestAPI":            findUser.SuggestApi,                      "openNewPage":           findUser.OpenNewPage,                      "showEngineIcon":        findUser.ShowEngineIcon,                      "showEngineList":        findUser.ShowEngineList,                      "language":              findUser.Language,                  }, "登录成功!")              }          }      }  }  // 获取图片  func WebImages(c \*gin.Context) {      var backGroundURLList []models.PresetBackground      var imageList string      var descriptionList string      var bg models.PresetBackground      mysql.DB.Where("bgid=?", 1).First(&bg)      mysql.DB.Select("background\_url").Find(&backGroundURLList)      for \_, presetBackGroundURL := range backGroundURLList {          imageList += presetBackGroundURL.BackgroundURL + ","      }      mysql.DB.Select("description").Find(&backGroundURLList)      for \_, presetBackGroundURL := range backGroundURLList {          descriptionList += presetBackGroundURL.Description + ","      }      response.Success(c, gin.H{"url": imageList, "description": descriptionList}, "")  }  // 获取图片  func OnImagesGet(c \*gin.Context) {      imageIndex := c.Param("imageIndex")      URL := "resource/" + imageIndex + ".jpg"      data, err := ioutil.ReadFile(URL)      if err != nil {          c.String(http.StatusInternalServerError, fmt.Sprintf("读取文件出错：%s", err.Error()))          return      }      // 将图片数据编码为 base64 字符串      imageBase64 := base64.StdEncoding.EncodeToString(data)      response.Success(c, gin.H{"image": imageBase64}, "200")  }  // 保存设置  func Settings(c \*gin.Context) {      var userSettings \*models.SettingForm      if err := c.ShouldBind(&userSettings); err != nil {          response.Fail(c, nil, "设置失败")      } else {          var findUser models.User          result := mysql.DB.Where("id=?", userSettings.Id).First(&findUser)          if result.Error != nil {              response.Fail(c, nil, "用户不存在!")          } else if result.Error == nil {              if err := mysql.DB.Model(&findUser).UpdateColumns(                  models.User{                      ButtonAlpha:           userSettings.ButtonAlpha,                      ButtonColor:           userSettings.ButtonColor,                      BackgroundAlpha:       userSettings.BackgroundAlpha,                      BackgroundColor:       userSettings.BackgroundColor,                      CustomBackgroundColor: userSettings.CustomBackgroundColor,                      CustomButtonColor:     userSettings.CustomButtonColor,                      PresetColor:           userSettings.PresetColor,                      FontColor:             userSettings.FontColor,                      SearchItemCount:       userSettings.SearchItemCount,                  }).Error; err != nil {                  response.Fail(c, gin.H{}, "设置失败")              }              response.Success(c, gin.H{}, "设置成功!")          }      }  }  // 销号  func DeleteAccount(c \*gin.Context) {      var userDelete \*models.DeleteForm      if err := c.ShouldBind(&userDelete); err != nil {          response.Fail(c, nil, "删除失败")      } else {          var findUser models.User          result := mysql.DB.Where("id=?", userDelete.Id).First(&findUser)          if result.Error != nil {              response.Fail(c, nil, "用户不存在!")          } else if result.Error == nil {              if err := mysql.DB.Delete(findUser).Error; err != nil {                  response.Fail(c, gin.H{}, "删除失败")              }              response.Success(c, gin.H{}, "删除成功!")          }      }  }  // 用户上传壁纸  func UploadBackground(c \*gin.Context) {      var uploadBackgroundForm \*models.UploadImageForm      node, err := snowflake.NewNode(1)      if err != nil {          panic(err)      }      if err := c.ShouldBind(&uploadBackgroundForm); err != nil {          response.Fail(c, nil, "上传失败")      } else {          var findUser models.User          ran := uint64(node.Generate().Int64() % 10000000000)          newurl := "resource/" + strconv.FormatUint(ran, 10) + ".jpg"          newURLOfUser := "http://localhost:2020/user/image/" + strconv.FormatUint(ran, 10)          // 生成图片          imageData := strings.ReplaceAll(uploadBackgroundForm.ImageData, "data:image/png;base64,", "")          imageBytes, err := base64.StdEncoding.DecodeString(imageData)          if err != nil {              response.Fail(c, gin.H{"error": "Invalid base64 encoding"}, "上传失败")          }          err = os.MkdirAll("image", os.ModePerm)          if err != nil {              response.Fail(c, gin.H{"error": "Failed to create image directory"}, "上传失败")          }          file, err := os.Create(newurl)          if err != nil {              response.Fail(c, gin.H{"error": "Failed to create image file"}, "上传失败")          }          defer file.Close()          \_, err = io.Copy(file, strings.NewReader(string(imageBytes))) // 将字节数组写入文件          if err != nil {              response.Fail(c, gin.H{"error": "Failed to write image file"}, "上传失败")          }          // 更新数据库          result := mysql.DB.Where("id=?", uploadBackgroundForm.Id).First(&findUser)          if result.Error != nil {              response.Fail(c, nil, "用户不存在!")          } else if result.Error == nil {              err = mysql.DB.Model(&findUser).UpdateColumns(                  models.User{                      BackgroundURL: newURLOfUser,                  }).Error              if err != nil {                  response.Fail(c, gin.H{}, "设置失败")              } else {                  response.Success(c, gin.H{}, "上传成功")              }          }      }  }  // 用户上传头像  func UploadAvatar(c \*gin.Context) {      var uploadAvatarForm \*models.UploadImageForm      node, err := snowflake.NewNode(1)      if err != nil {          panic(err)      }      if err := c.ShouldBind(&uploadAvatarForm); err != nil {          response.Fail(c, nil, "上传失败")      } else {          var findUser models.User          ran := uint64(node.Generate().Int64() % 10000000000)          newurl := "resource/" + strconv.FormatUint(ran, 10) + ".jpg"          newURLOfUser := "http://localhost:2020/user/image/" + strconv.FormatUint(ran, 10)          // 生成图片          imageData := strings.ReplaceAll(uploadAvatarForm.ImageData, "data:image/png;base64,", "")          imageBytes, err := base64.StdEncoding.DecodeString(imageData)          if err != nil {              response.Fail(c, gin.H{"error": "Invalid base64 encoding"}, "上传失败")          }          err = os.MkdirAll("image", os.ModePerm)          if err != nil {              response.Fail(c, gin.H{"error": "Failed to create image directory"}, "上传失败")          }          file, err := os.Create(newurl)          if err != nil {              response.Fail(c, gin.H{"error": "Failed to create image file"}, "上传失败")          }          defer file.Close()          \_, err = io.Copy(file, strings.NewReader(string(imageBytes))) // 将字节数组写入文件          if err != nil {              response.Fail(c, gin.H{"error": "Failed to write image file"}, "上传失败")          }          // 更新数据库          result := mysql.DB.Where("id=?", uploadAvatarForm.Id).First(&findUser)          if result.Error != nil {              response.Fail(c, nil, "用户不存在!")          } else if result.Error == nil {              err = mysql.DB.Model(&findUser).UpdateColumns(                  models.User{                      HeadUrl: newURLOfUser,                  }).Error              if err != nil {                  response.Fail(c, gin.H{}, "设置失败")              } else {                  response.Success(c, gin.H{}, "上传成功")              }          }      }  }  // 修改密码  func ChangePasswd(c \*gin.Context) {      var changePwd \*models.ChangePasswdForm      if err := c.ShouldBind(&changePwd); err != nil {          response.Fail(c, nil, "修改失败")      } else {          var findUser models.User          result := mysql.DB.Where("id=?", changePwd.Id).First(&findUser)          if result.Error != nil {              response.Fail(c, nil, "用户不存在!")          } else if result.Error == nil {              if err := mysql.DB.Model(&findUser).UpdateColumns(                  models.User{                      Password: changePwd.NewPwd,                  }).Error; err != nil {                  response.Fail(c, gin.H{}, "修改失败")              }              response.Success(c, gin.H{}, "修改成功!")          }      }  }  // 修改昵称  func ChangeUserName(c \*gin.Context) {      var changeUserName \*models.ChangeUserNameForm      if err := c.ShouldBind(&changeUserName); err != nil {          response.Fail(c, nil, "修改失败")      } else {          var findUser models.User          result := mysql.DB.Where("id=?", changeUserName.Id).First(&findUser)          if result.Error != nil {              response.Fail(c, nil, "修改失败")          } else if result.Error == nil {              if err := mysql.DB.Model(&findUser).UpdateColumns(                  models.User{                      UserName: changeUserName.NewName,                  }).Error; err != nil {                  response.Fail(c, gin.H{}, "修改失败")              }              response.Success(c, gin.H{}, "修改成功!")          }      }  }  // 得到当前URL  func GetMyURL(c \*gin.Context) {      userId := c.Param("userId")      var findUser models.User      result := mysql.DB.Where("id=?", userId).First(&findUser)      if result.Error != nil {          response.Fail(c, nil, "失败")      } else if result.Error == nil {          response.Success(c,              gin.H{                  "url":      findUser.Url,                  "title":    findUser.Title,                  "icon":     findUser.Icon,                  "textIcon": findUser.TextIcon,                  "custom":   findUser.Custom,              }, "成功!")      }  }  // 上传新URL  func PostNewURL(c \*gin.Context) {      userId := c.Param("userId")      var newURLForm []\*models.NewURLForm      if err := c.ShouldBind(&newURLForm); err != nil {          response.Fail(c, nil, "上传失败")      } else {          var findUser models.User          result := mysql.DB.Where("id=?", userId).First(&findUser)          if result.Error != nil {              response.Fail(c, nil, "修改失败")          } else if result.Error == nil {              newURL := ""              newTitle := ""              newIcon := ""              newTextIcon := ""              newCustom := ""              newPage := ""              for \_, obj := range newURLForm {                  newURL += obj.Url + " "                  newTitle += obj.Title + " "                  newIcon += obj.Icon + " "                  newTextIcon += obj.TextIcon + " "                  newCustom += obj.Custom + " "                  newPage += obj.Page + " "              }              if err := mysql.DB.Model(&findUser).UpdateColumns(                  models.User{                      Url:      newURL,                      Title:    newTitle,                      Icon:     newIcon,                      TextIcon: newTextIcon,                      Custom:   newCustom,                      Page:     newPage,                  }).Error; err != nil {                  response.Fail(c, gin.H{}, "修改失败")              }              response.Success(c, gin.H{}, "修改成功!")          }      }  }  // 上传搜索记录  func PostRecord(c \*gin.Context) {      userId := c.Param("userId")      var newRecordForm []\*models.NewRecordForm      if err := c.ShouldBind(&newRecordForm); err != nil {          response.Fail(c, nil, "上传失败")      } else {          var findUser models.User          result := mysql.DB.Where("id=?", userId).First(&findUser)          if result.Error != nil {              response.Fail(c, nil, "修改失败")          } else if result.Error == nil {              newRecord := ""              for \_, obj := range newRecordForm {                  newRecord += obj.Record + " "              }              if err := mysql.DB.Model(&findUser).UpdateColumns(                  models.User{                      Record: newRecord,                  }).Error; err != nil {                  response.Fail(c, gin.H{}, "修改失败")              }              response.Success(c, gin.H{}, "修改成功!")          }      }  }  // 获取百度搜索建议  func GetBaiduSuggestion(c \*gin.Context) {      keyword := c.Query("keyword")      r, err := http.Get(fmt.Sprintf("http://suggestion.baidu.com/su?wd=%s", keyword))      if err != nil {          response.Fail(c, gin.H{}, "失败")      } else {          defer r.Body.Close()          body, err := ioutil.ReadAll(r.Body)          if err != nil {              response.Fail(c, gin.H{}, "失败")          } else {              decoder := simplifiedchinese.GBK.NewDecoder()              jsonStr, \_, \_ := transform.String(decoder, string(body))              jsonStr = strings.TrimPrefix(jsonStr, "window.baidu.sug(")              jsonStr = strings.TrimSuffix(jsonStr, ");")              jsonStr = strings.ReplaceAll(jsonStr, "{", "{\"")              jsonStr = strings.ReplaceAll(jsonStr, ":", "\":")              jsonStr = strings.ReplaceAll(jsonStr, ",p", ",\"p")              jsonStr = strings.ReplaceAll(jsonStr, ",s", ",\"s")              var suggestionResponse models.BaiduSuggestionResponse              if err := json.Unmarshal([]byte(jsonStr), &suggestionResponse); err != nil {                  response.Fail(c, gin.H{}, "失败")              } else {                  response.Success(c, gin.H{"suggestion": suggestionResponse}, "成功")              }          }      }  }  // 获取必应搜索建议  func GetBingSuggestion(c \*gin.Context) {      keyword := c.Query("keyword")      r, err := http.Get(fmt.Sprintf("https://api.bing.com/qsonhs.aspx?type=cb&q=%s&cb=window.bing.sug", keyword))      if err != nil {          response.Fail(c, gin.H{}, "失败")      } else {          defer r.Body.Close()          body, err := ioutil.ReadAll(r.Body)          if err != nil {              response.Fail(c, gin.H{}, "失败")          } else {              jsonStr := strings.TrimPrefix(string(body), "if(typeof window.bing.sug == 'function') window.bing.sug(")              jsonStr = strings.TrimSuffix(jsonStr, "/\* pageview\_candidate \*/);")              var suggestionResponse models.BingSuggestionResponse              var sortedSuggestionResponse models.SortedBingSuggestionResponse              if err := json.Unmarshal([]byte(jsonStr), &suggestionResponse); err != nil {                  response.Fail(c, gin.H{}, "失败")              } else {                  for \_, obj := range suggestionResponse.AS.Results {                      for \_, objNext := range obj.Suggests {                          sortedSuggestionResponse.Txt = append(sortedSuggestionResponse.Txt, objNext.Txt)                      }                  }                  response.Success(c, gin.H{"suggestion": sortedSuggestionResponse}, "成功")              }          }      }  }  // 获取谷歌搜索建议  func GetGoogleSuggestion(c \*gin.Context) {      keyword := c.Query("keyword")      r, err := http.Get(fmt.Sprintf("https://api.bing.com/qsonhs.aspx?type=cb&q=%s&cb=window.bing.sug", keyword))      if err != nil {          response.Fail(c, gin.H{}, "失败")      } else {          defer r.Body.Close()          body, err := ioutil.ReadAll(r.Body)          if err != nil {              response.Fail(c, gin.H{}, "失败")          } else {              jsonStr := strings.TrimPrefix(string(body), "if(typeof window.bing.sug == 'function') window.bing.sug(")              jsonStr = strings.TrimSuffix(jsonStr, "/\* pageview\_candidate \*/);")              var suggestionResponse models.BingSuggestionResponse              var sortedSuggestionResponse models.SortedBingSuggestionResponse              if err := json.Unmarshal([]byte(jsonStr), &suggestionResponse); err != nil {                  response.Fail(c, gin.H{}, "失败")              } else {                  for \_, obj := range suggestionResponse.AS.Results {                      for \_, objNext := range obj.Suggests {                          sortedSuggestionResponse.Txt = append(sortedSuggestionResponse.Txt, objNext.Txt)                      }                  }                  response.Success(c, gin.H{"suggestion": sortedSuggestionResponse}, "成功")              }          }      }  }  // 调用各大网站api  func GetMoreInfo(c \*gin.Context) {      typ := c.Query("type")      r, err := http.Get(fmt.Sprintf("https://api.itapi.cn/api/hotnews/all?key=NMmQ5oB0oncJ4HKlQTL0ohAfbB&type=%s", typ))      if err != nil {          response.Fail(c, gin.H{}, "失败")      } else {          defer r.Body.Close()          body, err := ioutil.ReadAll(r.Body)          if err != nil {              response.Fail(c, gin.H{}, "失败")          } else {              var getAPI models.GetAPI              if err := json.Unmarshal(body, &getAPI); err != nil {                  response.Fail(c, gin.H{}, "失败")              } else {                  response.Success(c, gin.H{"data": getAPI.Data}, "成功")              }          }      }  } |

1. **测试报告**

**测试结论：我们小组的网页开发已经达到可用的水准，基本功能齐全，与预期目标一致。**

**测试时间：2023/07/05**

**测试人员：刘松凌，陈冠洲**

**测试环境：Windows 11**

**测试大纲：**

**（1）安全性测试：**

账号唯一性测试：试图注册已存在的账号是不可行的。

权限管理测试：授权和用户角色数据存放在服务器端。

访问控制安全测试：复制用户登录后的页面url地址，关闭该页面以后，不能直接进入登录后的页面url。

关键数据加密测试：在页面输入密码，页面显示为加密字符\*\*\*\*，查看源文件也无法看到刚刚输入的密码。

SQL注入测试：无法在输入框用SQL语句获取用户的密码

**（2）兼容性测试**

经测试，网页在各个浏览器、操作系统和分辨率下都能正常运行。

**（3）网页功能性测试**

完美实现的功能有：添加网站导航、导航分页切换、用户登录与注册、自定义新标签页背景、自定义组件颜色样式等。

有待改进的功能有：登录后对用户信息的管理界面、保存用户设置、Tab切换。

无法使用的功能有：搜索时报错。

1. **UI测试**

栏目名称、文章内容等处的文字正确，无错别字和乱码；

同一级别的字体、大小、颜色都统一；

窗体的中心位置在对角线交点附近；

按钮的样式风格统一，大小相似；

所有的图片都能被正确装载；

鼠标移动到Flash焦点特效上实现，移出焦点特效消失

1. **表单提交测试**

注册、登陆功能成功实现

提交按钮功能成功实现

修改表单与注册页面数据项相同，修改表单有对重名做验证

提交数据能正常保存到后台数据库中

注册/登录时表单提交后没有提示内容

点击浏览器刷新按钮，会造成登录失效，需要再次登录

注册/登录时表单不支持回车键和Tab键

1. **输入域测试**

输入字符不会显示在文本框以外；

对各种字符都能接收，不会报错；

密码输入域内容不可见。

1. **利用go test测试后端代码**

我们主要用go test测试user表中对于HTTP报文能否正常响应，其他功能只需要配合前端进行简单测试。

测试代码如下所示:

package control

import (

    "backend/models"

    "backend/mysql"

    "backend/response"

    "backend/router"

    "encoding/base64"

    "encoding/json"

    "fmt"

    "io"

    "io/ioutil"

    "net/http"

    "net/http/httptest"

    "os"

    "strconv"

    "strings"

    "testing"

"github.com/bwmarrin/snowflake"

    "github.com/gin-gonic/gin"

    "golang.org/x/text/encoding/simplifiedchinese"

    "golang.org/x/text/transform"

)

// 测试注册

func TestRegisterHandler(t \*testing.T)

{

    r:=router.SetupRouter()

    r.POST("/user/register", RegisterHandler)

    w:=httptest.NewRecorder()

    TestregisterForm :=models.RegisterForm{

        UserName: "testuser",

        Password: "password",

        ConfirmPassword: "password"

    }

    payload, \_ := json.Marshal(TestregisterForm)

    req, \_ := http.NewRequest("POST", "/user/register", bytes.NewBuffer(payload))

    req.Header.Set("Content-Type", "application/json")

    c, \_ := gin.CreateTestContext(w)

    c.Request = req。

    r.ServeHTTP(w, req)

    assert.Equal(t, http.StatusOK, w.Code)

    var response map[string]interface{}

    err := json.Unmarshal(w.Body.Bytes(), &response)

    if err != nil

    {

        fmt.Println(err)

    }

    assert.Equal(t, "注册成功!", response["message"])

    assert.Contains(t, response["data"], "id")

}

// 测试登录

func TestLoginHandler(t \*testing.T)

{

    r:=router.SetupRouter()

    r.POST("/user/login", LoginHandler)

    w:=httptest.NewRecorder()

    TestLoginForm:=models.RegisterForm{

        Id: "testuser",

        Password: "password",

    }

    payload, \_ := json.Marshal(TestLoginForm)

    req, \_ := http.NewRequest("POST", "/user/login", bytes.NewBuffer(payload))

    req.Header.Set("Content-Type", "application/json")

    c, \_ := gin.CreateTestContext(w)

    c.Request = req

    r.ServeHTTP(w, req)

    assert.Equal(t, http.StatusOK, w.Code)

    var response map[string]interface{}

    err := json.Unmarshal(w.Body.Bytes(), &response)

    if err != nil

    {

        fmt.Println(err)

    }

    assert.Equal(t, "登录成功!", response["message"])

    assert.Contains(t, response["data"])

}

/ 测试获取图片 (仅测试能否正常相应HTTP请求，经过前端测试能正确加载图片)

func TestWebImages(t \*testing.T)

{

    r=router.SetupRouter()

    r.GET("/user/web-images", WebImages)

    req, err := http.NewRequest("GET", "/user/web-images", nil)

    req.Header.Set("Content-Type", "application/json")

    if err != nil {

        t.Fatal(err)

    }

    w := httptest.NewRecorder()

    router.ServeHTTP(w, req)

    assert.Equal(t, http.StatusOK, w.Code)

    var response map[string]interface{}

    err := json.Unmarshal(w.Body.Bytes(), &response)

    assert.NotNil(t, response["BackgroundURL"])

    assert.NotNi2(t, response["Description"])

}

// 测试获取图片 (仅测试能否正常相应HTTP请求，经过前端测试能正确加载图片)

func TestOnImagesGet(t \*testing.T)

{

    r:=router.SetupRouter()

    r.GET("/user/image", UploadImageForm)

    // 创建一个新HTTP报文

    req, err := http.NewRequest("GET", "/user/image", nil)

    req.Header.Set("Content-Type", "application/json")

    if err != nil {

        t.Fatal(err)

    }

    w := httptest.NewRecorder()

    router.ServeHTTP(w, req)

    assert.Equal(t, http.StatusOK, w.Code)

    var response map[string]interface{}

    err := json.Unmarshal(w.Body.Bytes(), &response)

    assert.NotNil(t, response["Id"])

    assert.NotNi2(t, response["ImageData"])

    assert.NotNi2(t, response["Description"])

}

// 测试保存设置

func TestSettings(t \*testing.T)

{

    r:=router.SetupRouter()

    r.POST("/user/settings", Setting)

    w := httptest.NewRecorder()

    jsonPayload := models.SettingForm{

        "Id": "user123",

        "backgroundColor": "#FFFFFF",

        "backgroundAlpha": 0.5,

        "buttonColor": "#000000",

        "buttonAlpha": 0.8,

        "customBackgroundColor": true,

        "customButtonColor": true,

        "presetColor": 1,

        "fontColor": "#333333",

        "searchItemCount": 10

    }

    req, \_:= http.NewRequest("POST", "/user/settings", strings.NewReader(jsonPayload))

    req.Header.Set("Content-Type", "application/json")

    c, \_ := gin.CreateTestContext(w)

    c.Request = req。

    r.ServeHTTP(w, req)

    assert.Equal(t, http.StatusOK, w.Code)

    var response map[string]interface{}

    err := json.Unmarshal(w.Body.Bytes(), &response)

    if err != nil

    {

        fmt.Println(err)

    }

    expectedResponseBody := `{"message": "设置成功!"}`

    assert.Equal(t, expectedResponseBody, res.Body.String())

}

// 测试销号

func TestDeleteAccount(t \*testing.T)

{

    r=router.SetupRouter()

    r.POST("//user/deleteAccount", Setting)

    w := httptest.NewRecorder()

payload :=models.DeleteForm{

        "Id": "123456", // 需要销号的ID

    }

    body, \_ := json.Marshal(payload)

    req, \_ := http.NewRequest("POST", "/user/deleteAccount", bytes.NewBuffer(body))

    req.Header.Set("Content-Type", "application/json")

    router.ServeHTTP(w, req)

    assert.Equal(t, http.StatusOK, w.Code)

    var response map[string]interface{}

    json.NewDecoder(res.Body).Decode(&response)

    assert.Equal(t, "删除成功!", response["message"])

}

// 测试用户上传壁纸

func TestUploadBackground(t \*testing.T)

{

    r:=router.SetupRouter()

    r.POST("/user/uploadBackground", UploadBackground)

    w := httptest.NewRecorder()

    //假设数据中存在Id=123456

    payload :=models.UploadImageForm{

        Id:          "123456",

        ImageData:   "base64-encoded-image-data",

        Description: "",

    }

    body, \_ := json.Marshal(payload)

    req, \_ := http.NewRequest("POST", "/user/uploadBackground", bytes.NewBuffer(body))

    req.Header.Set("Content-Type", "application/json")

    router.ServeHTTP(w, req)

    assert.Equal(t, http.StatusOK, w.Code)

    var response map[string]interface{}

    json.NewDecoder(w.Body).Decode(&response)

    assert.Equal(t, "上传成功!", response["message"])

}

// 测试用户上传头像

func TestUploadAvatar(t \*testing.T)

{

    r:=router.SetupRouter()

    r.POST("/user/uploadAvatar", UploadAvatar)

    w := httptest.NewRecorder()

    //假设数据中存在Id=123456

    payload :=models.UploadImageForm{

        Id:          "123456",

        ImageData:   "base64-encoded-image-data",

        Description: "",

    }

    body, \_ := json.Marshal(payload)

    req, \_ := http.NewRequest("POST", "/user/uploadAvatar", bytes.NewBuffer(body))

    req.Header.Set("Content-Type", "application/json")

    router.ServeHTTP(w, req)

    assert.Equal(t, http.StatusOK, w.Code)

    var response map[string]interface{}

    json.NewDecoder(w.Body).Decode(&response)

    assert.Equal(t, "上传成功!", response["message"])

}

// 测试修改密码

func TestChangePasswd(t \*testing.T)

{

    r:=router.SetupRouter()

    r.POST("/user/changePassword", ChangePasswd)

    w := httptest.NewRecorder()

    //假设数据中存在Id=123456

    type ChangePasswdForm struct {

        Id     "123456"

        NewPwd "123456"

    }

    payload, \_ := json.Marshal(ChangePasswdForm)

    req, \_ := http.NewRequest("POST", "/user/changePassword", bytes.NewBuffer(payload))

    req.Header.Set("Content-Type", "application/json")

    router.ServeHTTP(w, req)

    assert.Equal(t, http.StatusOK, w.Code)

    var response map[string]interface{}

    err := json.Unmarshal(w.Body.Bytes(), &response)

    if err != nil

    {

        fmt.Println(err)

    }

    expectedResponseBody := `{"message": "修改成功!"}`

    assert.Equal(t, expectedResponseBody, res.Body.String())

}

// 测试修改昵称

func TestChangeUserName(t \*testing.T)

{

    r:=router.SetupRouter()

    r.POST("/user/changeUserName", ChangeUserName)

    w := httptest.NewRecorder()

    //假设数据中存在Id=123456

    type ChangeUserNameForm struct {

        Id     "123456"

        NewNmae "cxz"

    }

    payload, \_ := json.Marshal(ChangeUserNameForm)

    req, \_ := http.NewRequest("POST", "/user/changeUserName", bytes.NewBuffer(payload))

    req.Header.Set("Content-Type", "application/json")

    router.ServeHTTP(w, req)

    assert.Equal(t, http.StatusOK, w.Code)

    var response map[string]interface{}

    err := json.Unmarshal(w.Body.Bytes(), &response)

    if err != nil

    {

        fmt.Println(err)

    }

    expectedResponseBody := `{"message": "修改成功!"}`

    assert.Equal(t, expectedResponseBody, res.Body.String())

}

// 测试得到当前URL

func TestGetMyURL(t \*testing.T)

{

    router:= control.SetupRouter()

    req, \_ := http.NewRequest("GET", "/user/myURL/:userId", nil)

    req.Header.Set("Content-Type", "application/json")

    userId := "123"

    w := httptest.NewRecorder()

    router.ServeHTTP(w, req)

    assert.Equal(t, http.StatusOK, recorder.Code)

    expectedResponseBody := "Expected response body"

    assert.Contains(t, recorder.Body.String(), expectedResponseBody)

}

// 测试上传新URL

func TestPostNewURL(t \*testing.T)

{

    r:=router.SetupRouter()

    r.POST("/user/newURL", PostNewURL)

    userId := "123"

    w := httptest.NewRecorder()

    payload :=models.NewURLForm

        {

            "title": "Example Title",

            "url": "https://example.com",

            "icon": "example-icon",

            "textIcon": "example-text-icon",

            "custom": "example-custom",

            "page": "example-page"

        }

    req, \_:= http.NewRequest("POST", "/user/newURL/"+userId, strings.NewReader(payload))

    req.Header.Set("Content-Type", "application/json")

    c, \_ := gin.CreateTestContext(w)

    c.Request = req

    r.ServeHTTP(w, req)

    assert.Equal(t, http.StatusOK, w.Code)

    assert.Contains(t, w.Body.String(), "修改成功!")

    payload :=models.NewRecordForm={

        "record": "search record 1"

    }

}

// 测试上传搜索记录

func TestPostRecord(t \*testing.T)

{

    r=router.SetupRouter()

    r.POST("/user/record/:userId",PostRecord)

    userId := "123"

    w := httptest.NewRecorder()

    req, \_:= http.NewRequest("POST", "/user/record/"+userId, strings.NewReader(payload))

    req.Header.Set("Content-Type", "application/json")

    c, \_ := gin.CreateTestContext(w)

    c.Request = req

    r.ServeHTTP(w, req)

    assert.Equal(t, http.StatusOK, w.Code)

    expectedResponse := `{"message":"修改成功!"}`

    if w.Body.String() != expectedResponse

    {

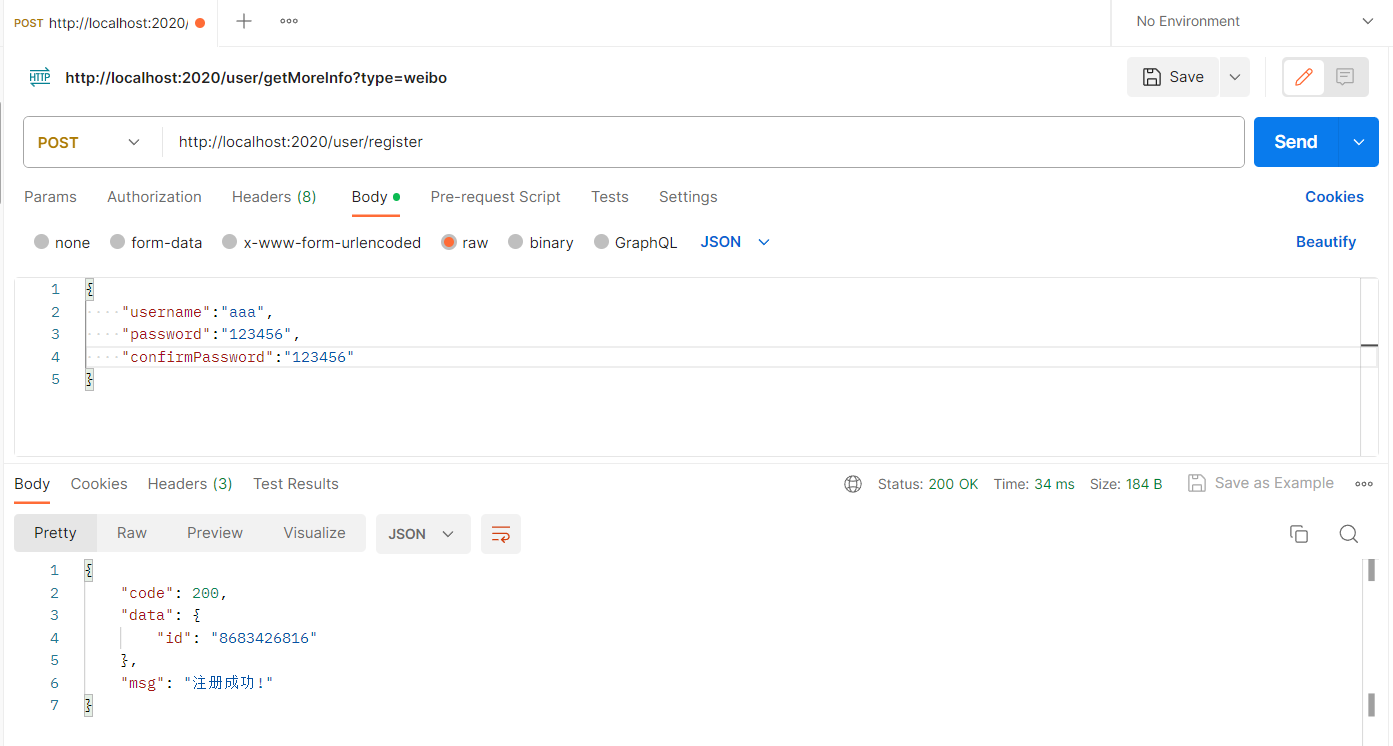
        t.Errorf("Expected response body %s but got %s", expectedResponse, w.Body.String())

    }

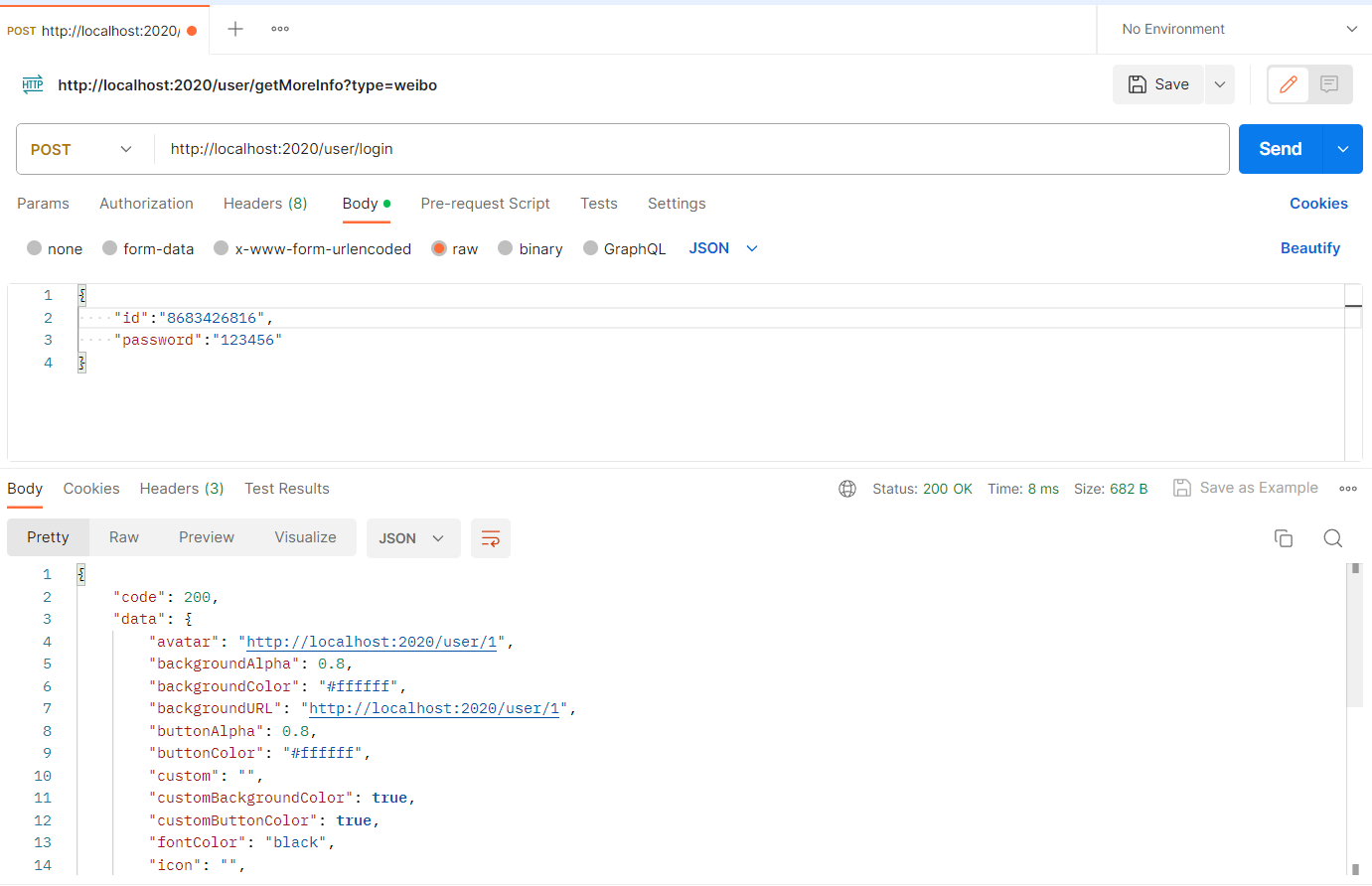
}

**除此以外，我们还对各个接口进行数据测试**

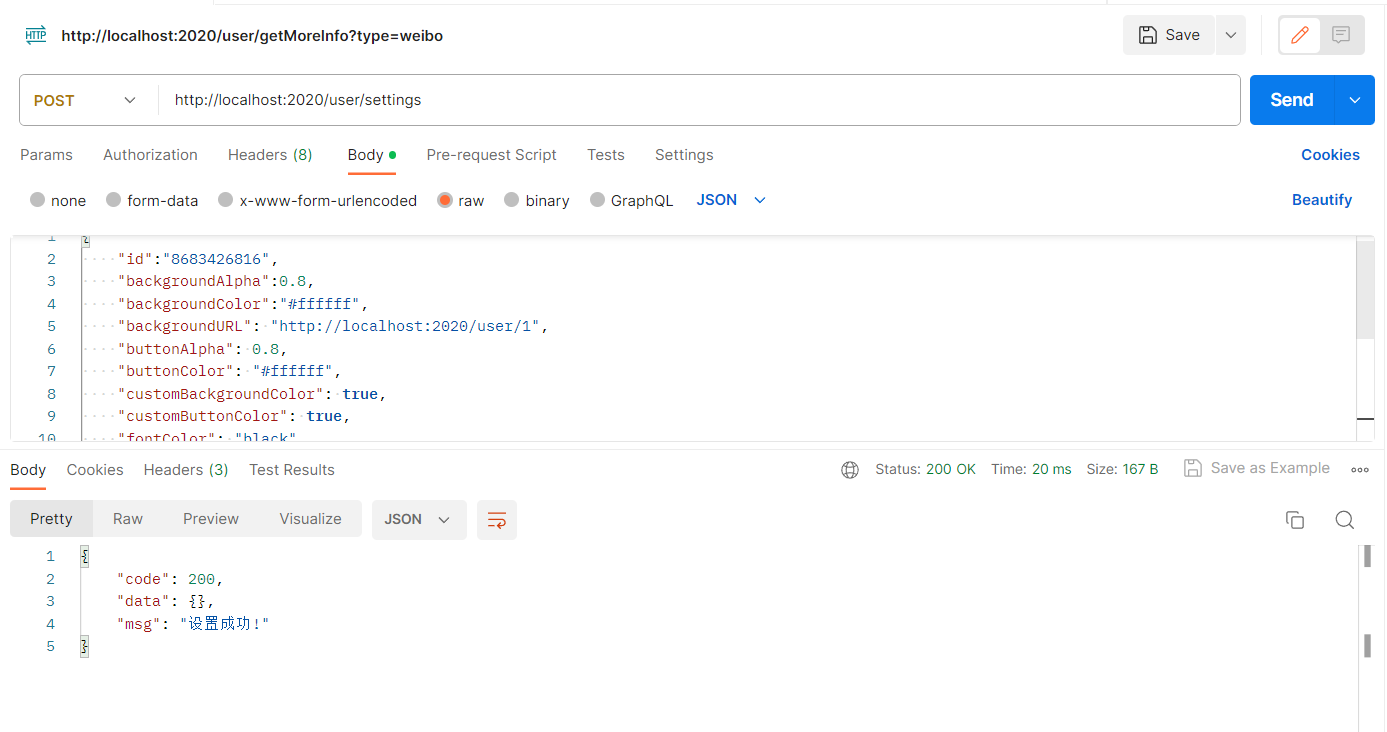
**①注册接口RegisterHandler：**



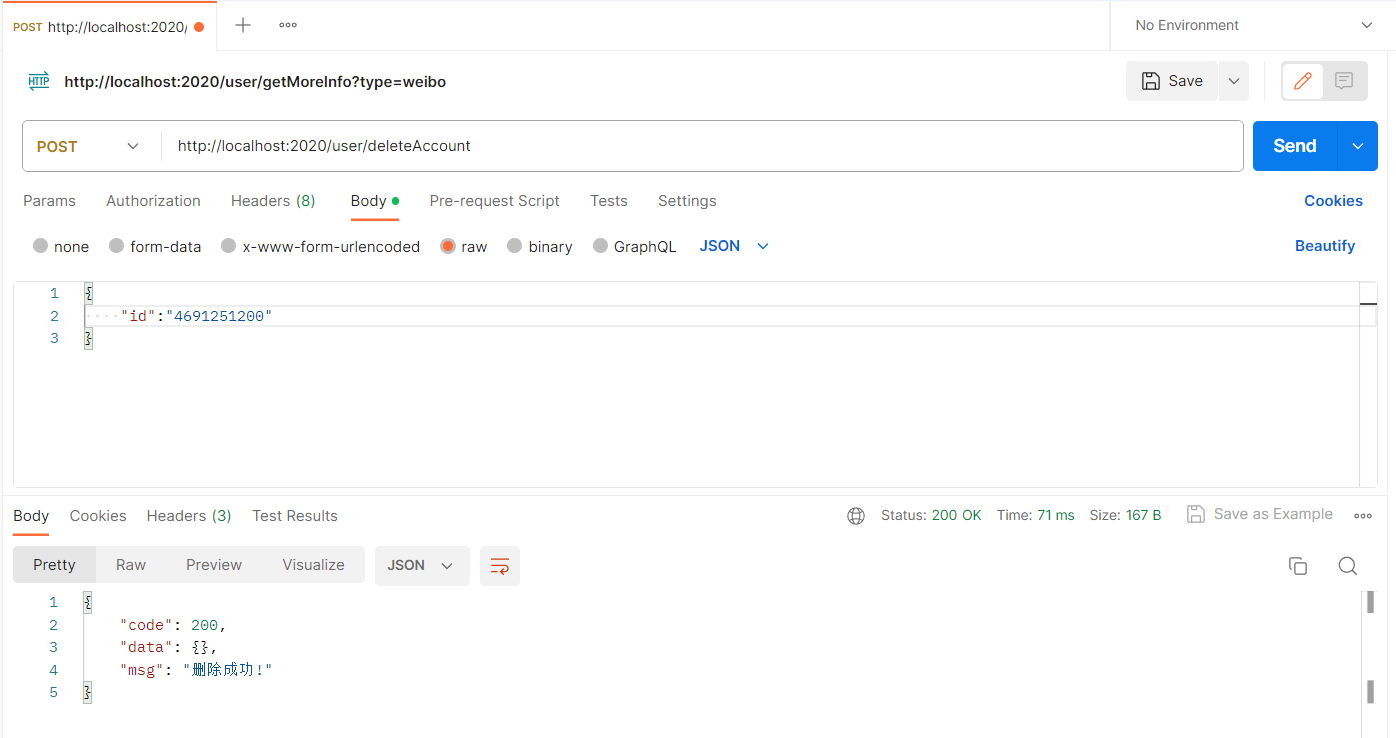
**②登录接口LoginHandler：**



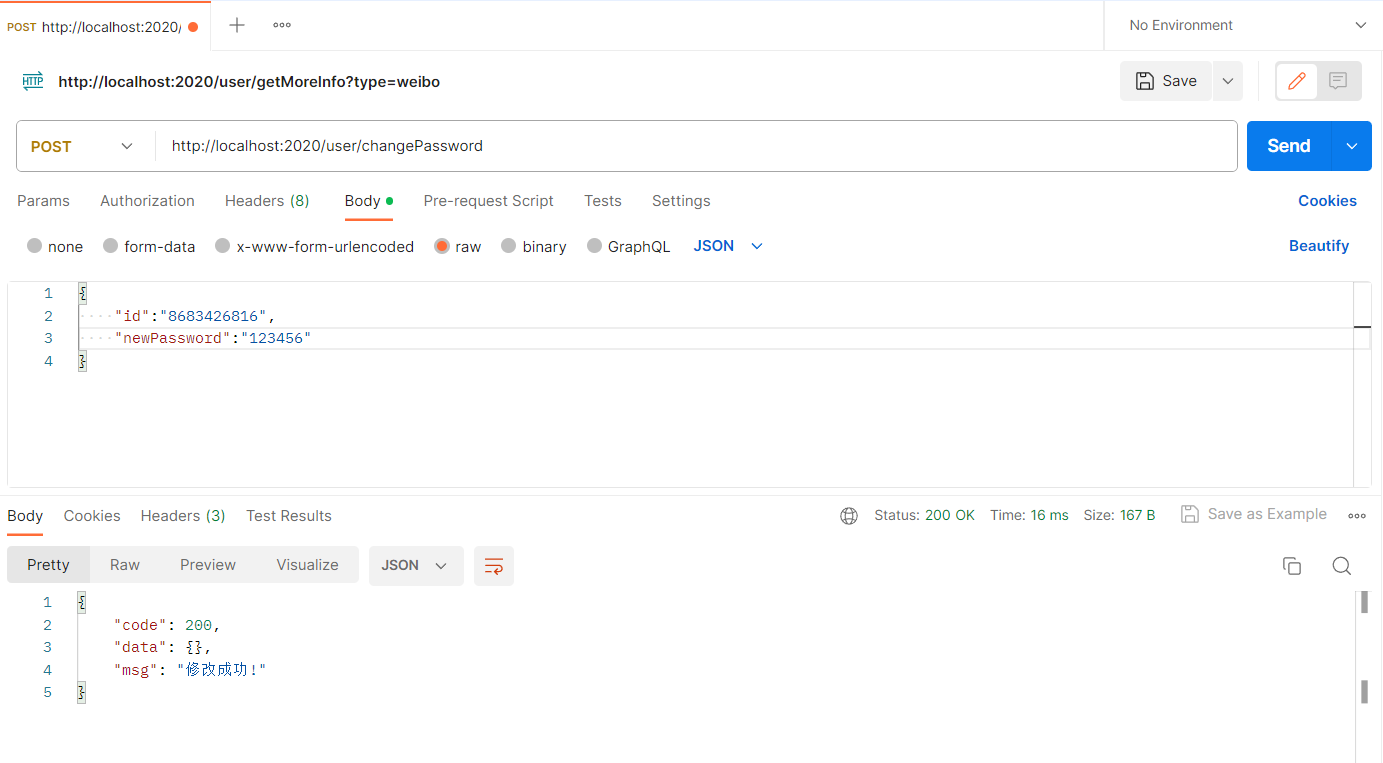
**③设置接口Settings：**



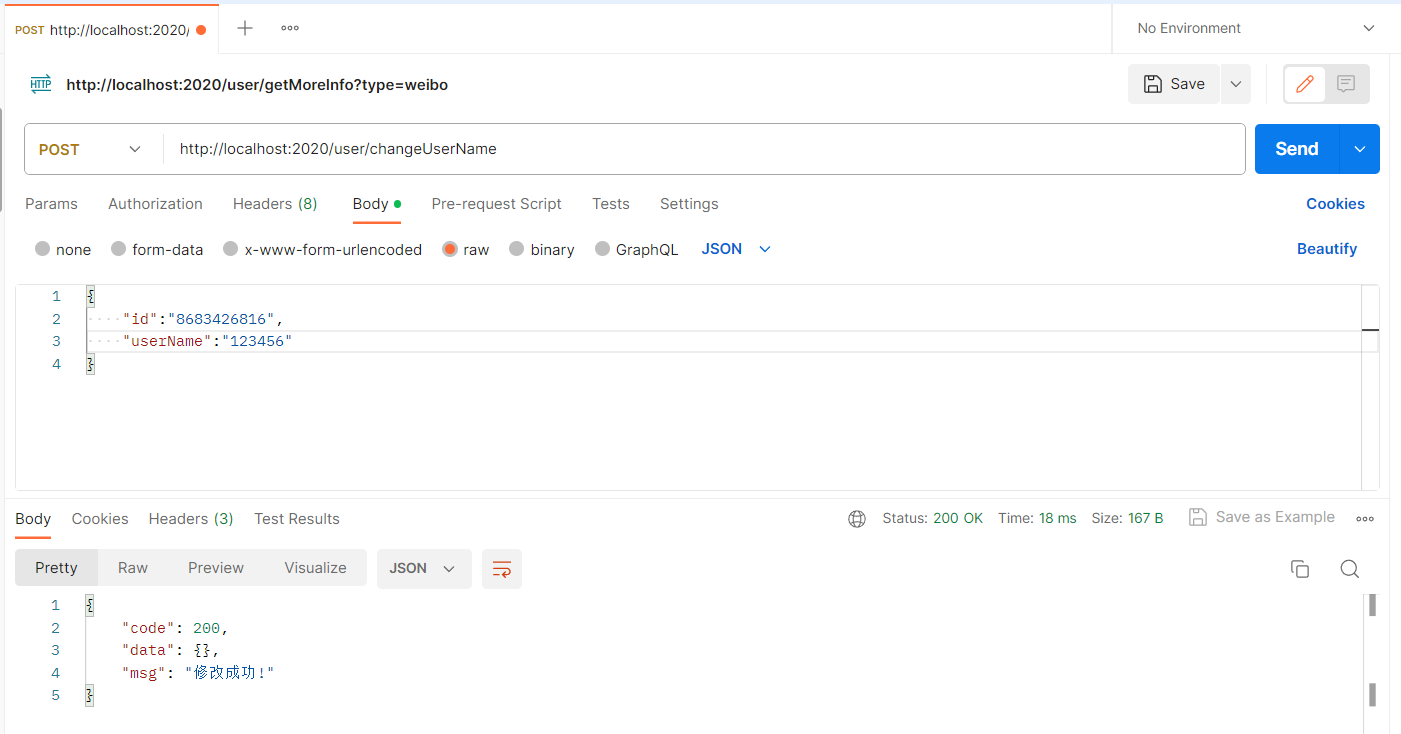
**④销号接口DeleteAccount：**



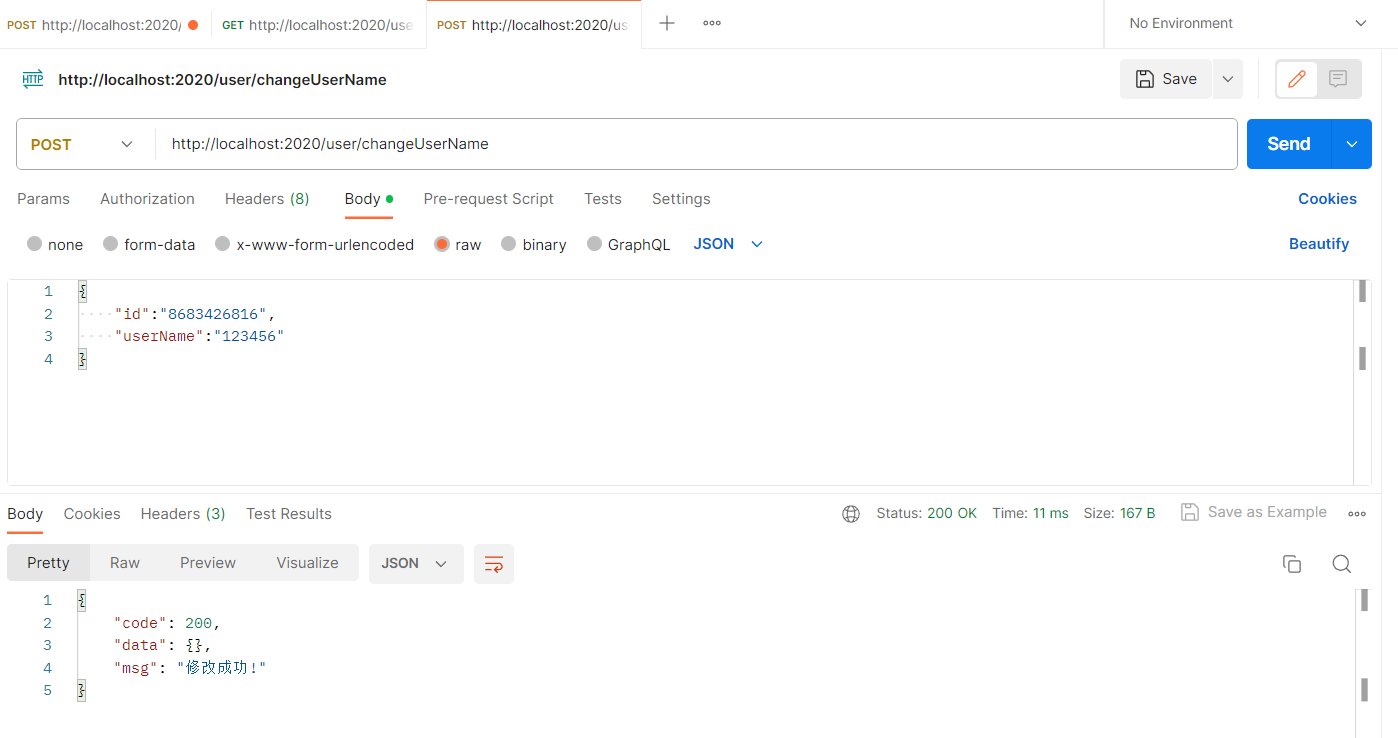
**⑤修改密码ChangePasswd：**



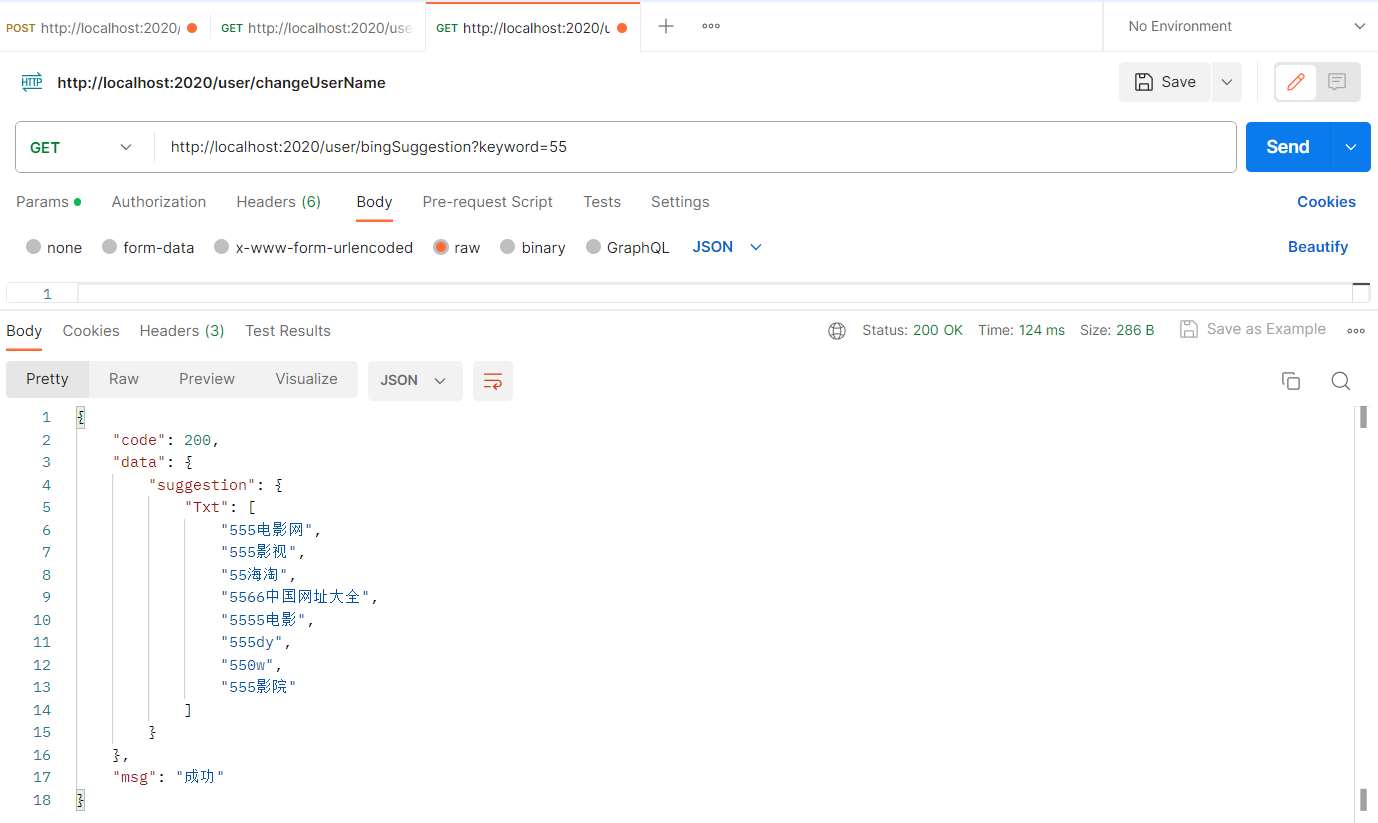
**⑥修改昵称ChangeUserName：**



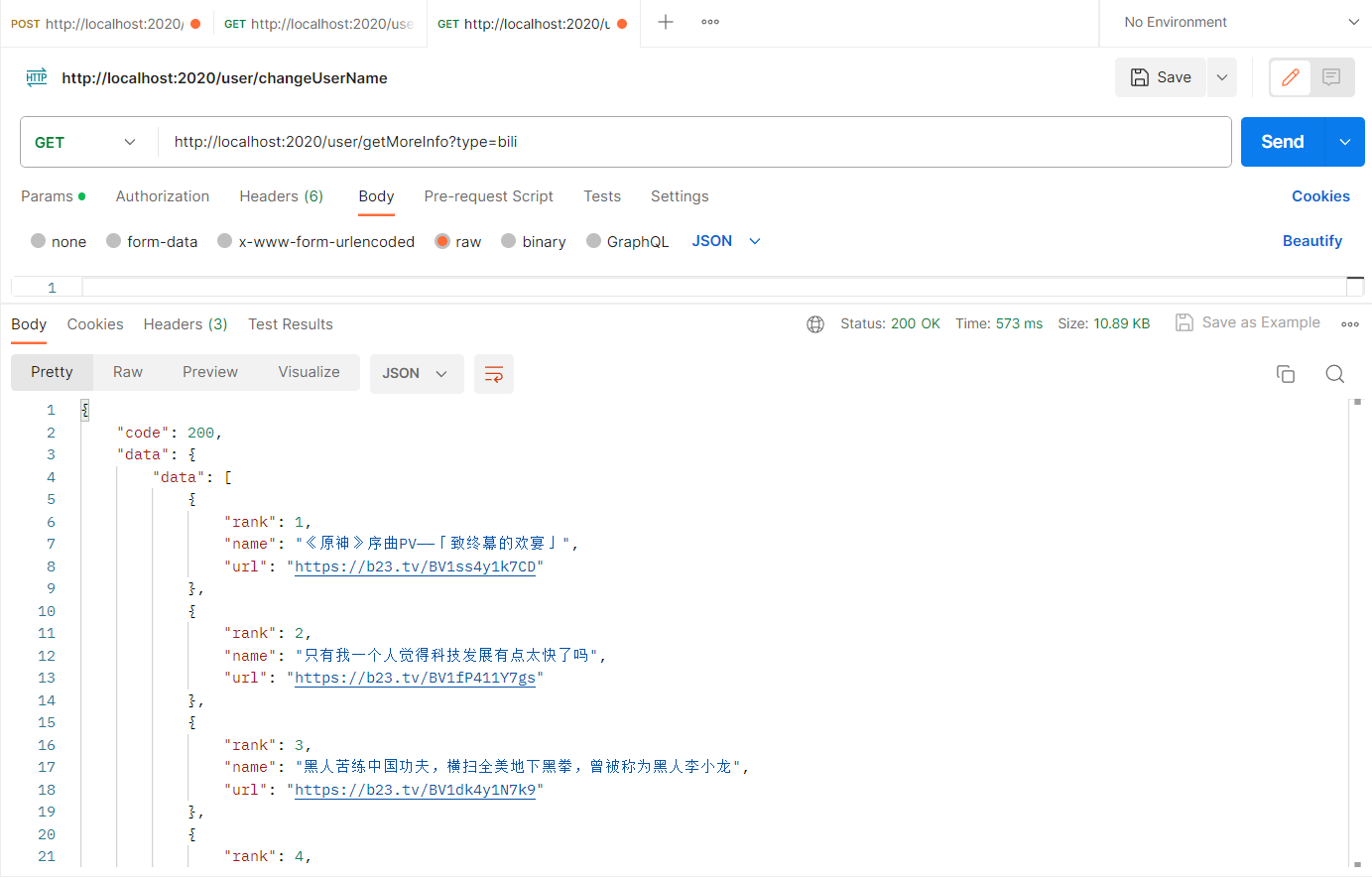
**⑦修改昵称ChangeUserName：**



**⑧获取搜索推荐接口GetBingSuggestion：**



**⑨获取热搜接口GetMoreInfo：**



1. **项目展望**

当我们完成了这个新标签页项目的基本功能后，我们能够展望一些可能的扩展和改进，以进一步提升用户体验和增加功能。以下是一些可能的项目展望：

**①定制化主题和布局：**在目前的项目中，我们可以提供一些预设的主题和布局供用户选择。但是，我们可以进一步扩展，让用户能够自定义和定制自己喜欢的主题和布局，以满足不同用户的个性化需求；

**②多设备同步：**为了提供更好的用户体验，我们可以增加多设备同步功能。用户登录后，他们的设置和收藏的URL可以在不同设备之间同步，这样无论在哪个设备登录，他们的个人喜好和收藏都能保持一致并始终可用；

**③智能推荐和搜索建议：**我们可以利用用户的设置和收藏历史，实现智能推荐功能。根据用户的偏好和使用习惯，我们可以向用户推荐相关的URL或支持搜索建议，帮助用户更快地找到他们需要的内容；

**④个人化的广告推荐：**通过用户的搜索和浏览行为，我们可以为用户提供个性化的广告推荐。这将使广告对于用户更加相关和有趣，并提供更好的广告效果以及更好的商业机会；

**⑤第三方应用集成：**我们可以考虑将第三方应用集成到新标签页中，例如天气预报、待办事项、日历等常用的功能。通过集成这些应用，我们可以让用户在打开新标签页时获取更多的有用信息，提高其生产力和便利性；

**⑥社交分享和评论功能：**为了增加用户互动和社交性，我们可以添加社交分享和评论功能。用户可以分享他们喜欢的网址到社交媒体，并与其他用户进行交流和讨论，增强用户参与度和社群；

**⑦数据分析和用户行为统计：**通过收集和分析用户的行为数据，我们可以深入了解用户的偏好和使用习惯，并据此优化我们的产品。通过数据分析，我们可以提供更好的用户体验和更精确的个性化推荐。

以上是一些我们可以展望的可能的项目扩展和改进。当然，具体的项目展望将根据实际需求和资源可用性来确定。展望未来，我们的目标是不断改进和创新，为用户提供更优秀的新标签页体验，并不断满足和超越用户的期望。

1. **心得体会**

在这个项目中，我们前端团队主要使用了**Vue.js**和**Vuex**，而后端团队则采用了**Golang**以及**Gin**和**GORM**框架。这个前后端分离的架构为我们提供了许多好处，并且我们从中获得了很多宝贵的经验和体会：

**①Vue.js和Vuex的强大：**在前端开发中，我们选择了Vue.js作为我们的主要框架。Vue.js具有简单易学、灵活和高效的特点。通过组件化的开发方式，我们能够将复杂的界面拆分为可重用和独立的组件，这使得代码更加简洁和易于维护。而Vuex则提供了一个集中式的状态管理方案，使我们能够更好地管理和共享状态，提高开发效率和代码可维护性。

**②Golang、Gin和GORM的协同工作：**后端团队采用了Golang作为开发语言，结合Gin和GORM框架，为我们提供了快速、高效和可扩展的后端开发解决方案。Gin是一个轻量级的HTTP Web框架，具有快速路由和中间件支持，使我们能够方便地构建强大的API。同时，GORM作为一个强大的ORM（对象关系映射）库，简化了数据库操作的过程，提高了开发效率。

**③前后端分离架构的优势：**这个项目的前后端分离架构使我们的工作更加灵活和高效。通过使用RESTful API进行数据交互，前后端团队能够并行开发，提高了协作效率。前端开发人员可以专注于用户界面的设计和交互，而后端开发人员则可以专注于构建高性能的API服务。同时，通过明确的界限和接口定义，我们能够更容易地进行模块化开发，实现代码的复用。

**④强大的工具生态系统：**Vue.js和Golang都拥有活跃且强大的工具生态系统，为我们提供了丰富的插件、库和工具。在前端开发中，我们使用了许多Vue.js的插件和库，如Vue Router和Axios，来简化路由和网络请求的处理。而Golang的工具生态系统也提供了许多有用的工具，如gojsonschema和gin-swagger，帮助我们更好地进行数据验证和自动生成API文档。

**⑤学习和成长：**这个项目为我们全体团队成员提供了一个良好的学习和成长机会。我们不仅学到了新的技术和工具，还了解了如何协作和沟通，如何在项目中解决问题和优化性能。通过团队合作和知识共享，我们共同解决了许多挑战，提高了自己的技能和能力。

通过使用Vue.js、Vuex、Golang、Gin和GORM等工具和框架，以及前后端分离的架构，我们成功地完成了这个新标签页网址的项目。我们获得了许多宝贵的经验和技能，包括前端组件化的开发、状态管理、后端API的设计和性能优化等。这个项目也加强了我们的团队协作和问题解决能力。我们期待将这些经验应用到未来的项目中，创造出更优秀的Web应用程序。