ginEssential

V0.1

- gorm初始化数据库
  - 连接数据库
  - db.AutoMigrate()自动迁移数据模型
- 在main中实现register API
  - c.PostForm获取数据
  - 数据格式校验
  - 检查用户是否存在
  - 创建用户并存入数据库db.Create()
  - 返回注册成功结果
- Postman调试接口
- navicat创建gorm连接所需数据库
- git提交版本
  - git init
  - git add
  - git commit
  - git log

V0.2

- 代码重构
  - User struct放入model包
  - register API中gin.handlerFunc放入controller包
  - database放入到common包
  - APIl放入到routes.go中
- Postman调试接口
- git提交版本

V0.3

- 实现login API
  - c.PostForm获取数据
  - 数据格式检验
  - 检查用户是否存在
  - 检查密码是否正确
    - bcrypt.CompareHashAndPassword([]byte(user.Password), []byte(password))
  - 下发token
    - 简单实现固定token值，后续优化
  - 返回登录成功结构
- 修改register API
  - 创建user时加密密码，在数据库中保存加密后密码
    - hashedPassword, err := bcrypt.GenerateFromPassword([]byte(password), bcrypt.DefaultCost)
- Postman调试接口
- git提交版本

V0.4

- 配合jwt实现用户权限认证中间件，保护用户信息接口
- 使用jwt模块完善token生成过程
  - 定义claims结构体并初始化
    - type Claims struct { UserId uint jwt.StandardClaims }
  - token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
  - tokenString, err := token.SignedString(jwtKey)
- 用户权限鉴定中间件
  - c.GetHeader("Authorization")获取authorization header
  - header格式校验
  - 获取header中token
  - 分析token是否合法
    - token, err := jwt.ParseWithClaims(tokenString, claims, func(token *jwt.Token) (i interface{}, err error) {}
  - 从claims中获取userID
  - 通过userID在数据库中匹配user
  - 如果用户存在则将用户写入上下文中
- 创建user info API
  - 经过鉴权后从上下文中获取用户信息，并返回前端

V0.5

- 添加请求响应过滤方法，只返回非敏感信息到前端
  - func ToUserDto(user model.User) UserDto {}
- 统一封装请求响应格式
  - 定义请求响应格式
  - 重构register login使用封装过的请求响应

V0.6

- viper统一管理配置项
  - 添加全局统一配置管理application.yml
  - main包中初始化viper

V0.7

- 添加中间件解决跨域问题
- 定义预请求响应
  - ctx.Writer.Header().Set("Access-Control-Allow-Origin", "http://localhost:8989")
  - ctx.Writer.Header().Set("Access-Control-Allow-Methods", "*")
  - ……

V0.8

- 添加文章分类
- 创建路由组
  - categoryRoutes := r.Group("/categories")
- 属性
  - ID主键
  - Name
  - createAt
  - updateAt
- 请求属性
  - type CreateCategoryRequest struct { Name string `json:"name" form:"name" binding:"required"` // name字段是required }
    - 方便使用shouldBind方法从请求中获取数据并填充到请求属性对象中，方便后续方法使用
- 数据库属性
  - 创建嵌入数据库的属性
    - categoryController := controller.NewCategoryController()
- 数据库操作方法
  - 初始化数据库
    - 获取main中连接的数据库对象
    - 迁移定义的文章分类属性
  - 增
    - categoryRoutes.POST("", categoryController.Create)
  - 删
    - categoryRoutes.DELETE("/:id", categoryController.Delete)
  - 改
    - categoryRoutes.PUT("/:id", categoryController.Update)
      - 通过上下文获取ID
      - 通过ID在数据库中找到数据返回填充到结构体中
      - 向前端发送结构体
  - 查
    - categoryRoutes.GET("/:id", categoryController.Show)
- 使用Postman Navicat调试