Enterprise-Grade Go Microservice with RESTful API, JWT Auth, PostgreSQL, Redis, and Goroutines

---

Project Structure

```
go-microservice/
├── cmd/
│   └── main.go
├── config/
│   └── config.go
├── controllers/
│   └── user_controller.go
├── models/
│   └── user.go
├── services/
│   └── user_service.go
├── repository/
│   └── user_repo.go
├── middleware/
│   └── auth_middleware.go
├── utils/
│   └── jwt.go
├── go.mod
```

└── Dockerfile

---

go.mod

```
module go-microservice

go 1.20

require (
	github.com/dgrijalva/jwt-go v3.2.0
	github.com/go-redis/redis/v8 v8.11.5
	github.com/gorilla/mux v1.8.0
	github.com/joho/godotenv v1.4.0
	github.com/lib/pq v1.10.4
)
```

---

config/config.go

```
package config

import (
	"log"
	"os"

	"github.com/joho/godotenv"
)
```

```
func LoadEnv() {
	if err := godotenv.Load(); err != nil {
		log.Fatalf("Error loading .env file")
	}
}
```

---

models/user.go

```
package models

type User struct {
	ID       int    `json:"id"`
	Username string `json:"username"`
	Email    string `json:"email"`
	Password string `json:"-"`
}
```

---

repository/user_repo.go

```
package repository

import (
	"database/sql"
	"go-microservice/models"
)

type UserRepository interface {
```

```
    FindByEmail(email string) (*models.User, error)
    CreateUser(user *models.User) error
}

type userRepo struct {
    db *sql.DB
}

func NewUserRepository(db *sql.DB) UserRepository {
    return &userRepo{db}
}

func (r *userRepo) FindByEmail(email string) (*models.User, error) {
    query := `SELECT id, username, email, password FROM users WHERE email=$1`
    user := models.User{}
    err := r.db.QueryRow(query, email).Scan(&user.ID, &user.Username, &user.Email,
&user.Password)
    return &user, err
}

func (r *userRepo) CreateUser(user *models.User) error {
    query := `INSERT INTO users (username, email, password) VALUES ($1, $2, $3)`
    _, err := r.db.Exec(query, user.Username, user.Email, user.Password)
    return err
}
```

---

services/user_service.go

```
package services

import (
	"errors"
	"go-microservice/models"
	"go-microservice/repository"
	"golang.org/x/crypto/bcrypt"
)

type UserService interface {
	Register(user *models.User) error
	Login(email, password string) (*models.User, error)
}

type userService struct {
	repo repository.UserRepository
}

func NewUserService(repo repository.UserRepository) UserService {
	return &userService{repo}
}

func (s *userService) Register(user *models.User) error {
```

```
	hashed, err := bcrypt.GenerateFromPassword([]byte(user.Password), bcrypt.DefaultCost)
	if err != nil {
		return err
	}
	user.Password = string(hashed)
	return s.repo.CreateUser(user)
}

func (s *userService) Login(email, password string) (*models.User, error) {
	user, err := s.repo.FindByEmail(email)
	if err != nil {
		return nil, errors.New("user not found")
	}
	err = bcrypt.CompareHashAndPassword([]byte(user.Password), []byte(password))
	if err != nil {
		return nil, errors.New("invalid credentials")
	}
	return user, nil
}
```

---

utils/jwt.go

```
package utils

import (
	"time"
```

```
	"github.com/dgrijalva/jwt-go"
)

var jwtKey = []byte("very_secret_key")

type JWTClaim struct {
	Email string `json:"email"`
	jwt.StandardClaims
}

func GenerateJWT(email string) (string, error) {
	expirationTime := time.Now().Add(5 * time.Hour)
	claims := &JWTClaim{
		Email: email,
		StandardClaims: jwt.StandardClaims{
			ExpiresAt: expirationTime.Unix(),
		},
	}
	token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
	return token.SignedString(jwtKey)
}

func ValidateJWT(tokenStr string) (*JWTClaim, error) {
	claims := &JWTClaim{}
```

```
	tkn, err := jwt.ParseWithClaims(tokenStr, claims, func(t *jwt.Token) (interface{}, error)
{
		return jwtKey, nil
	})
	if err != nil || !tkn.Valid {
		return nil, err
	}
	return claims, nil
}
```

---

middleware/auth_middleware.go

```
package middleware

import (
	"context"
	"go-microservice/utils"
	"net/http"
	"strings"
)

func JWTMiddleware(next http.Handler) http.Handler {
	return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
		auth := r.Header.Get("Authorization")
		if auth == "" || !strings.HasPrefix(auth, "Bearer ") {
			http.Error(w, "Unauthorized", http.StatusUnauthorized)
```

```
            return
        }
        tokenStr := strings.TrimPrefix(auth, "Bearer ")
        claims, err := utils.ValidateJWT(tokenStr)
        if err != nil {
            http.Error(w, "Forbidden", http.StatusForbidden)
            return
        }
        ctx := context.WithValue(r.Context(), "email", claims.Email)
        next.ServeHTTP(w, r.WithContext(ctx))
    })
}
```

---

controllers/user_controller.go

```
package controllers

import (
    "encoding/json"
    "go-microservice/models"
    "go-microservice/services"
    "go-microservice/utils"
    "net/http"
)

type UserController struct {
```

```
	service services.UserService
}

func NewUserController(service services.UserService) *UserController {
	return &UserController{service}
}

func (uc *UserController) Register(w http.ResponseWriter, r *http.Request) {
	var user models.User
	json.NewDecoder(r.Body).Decode(&user)
	err := uc.service.Register(&user)
	if err != nil {
		http.Error(w, err.Error(), http.StatusBadRequest)
		return
	}
	w.WriteHeader(http.StatusCreated)
}

func (uc *UserController) Login(w http.ResponseWriter, r *http.Request) {
	var user models.User
	json.NewDecoder(r.Body).Decode(&user)
	authUser, err := uc.service.Login(user.Email, user.Password)
	if err != nil {
		http.Error(w, "Invalid login", http.StatusUnauthorized)
		return
```

```
	}
	token, err := utils.GenerateJWT(authUser.Email)
	if err != nil {
		http.Error(w, "Token generation failed", http.StatusInternalServerError)
		return
	}
	json.NewEncoder(w).Encode(map[string]string{"token": token})
}
```

---

cmd/main.go

```
package main

import (
	"database/sql"
	"fmt"
	"go-microservice/config"
	"go-microservice/controllers"
	"go-microservice/middleware"
	"go-microservice/repository"
	"go-microservice/services"
	"log"
	"net/http"
	"os"
	"os/signal"
	"syscall"
```

```
	"time"


	"github.com/gorilla/mux"
	_ "github.com/lib/pq"
)

func main() {
	config.LoadEnv()

	db, err := sql.Open("postgres", os.Getenv("DB_URL"))
	if err != nil {
		log.Fatal("PostgreSQL connection failed:", err)
	}
	defer db.Close()

	userRepo := repository.NewUserRepository(db)
	userService := services.NewUserService(userRepo)
	userController := controllers.NewUserController(userService)

	router := mux.NewRouter()

	router.HandleFunc("/register", userController.Register).Methods("POST")
	router.HandleFunc("/login", userController.Login).Methods("POST")
	router.Handle("/secure", middleware.JWTMiddleware(http.HandlerFunc(func(w
http.ResponseWriter, r *http.Request) {
```

```
        email := r.Context().Value("email").(string)
        w.Write([]byte("Welcome " + email))
})))).Methods("GET")

server := &http.Server{
        Addr:    ":8080",
        Handler: router,
}

go func() {
        log.Println("Server started on :8080")
        if err := server.ListenAndServe(); err != nil {
                log.Println("Shutting down server...")
        }
}()

quit := make(chan os.Signal, 1)
signal.Notify(quit, syscall.SIGINT, syscall.SIGTERM)
<-quit

ctx, cancel := context.WithTimeout(context.Background(), 5*time.Second)
defer cancel()

if err := server.Shutdown(ctx); err != nil {
        log.Fatal("Server shutdown failed:", err)
```

```
    }
    log.Println("Server gracefully stopped")
}
```

Dockerfile

```
FROM golang:1.20

WORKDIR /app

COPY go.mod ./
COPY go.sum ./
RUN go mod download

COPY . .

RUN go build -o main ./cmd/main.go

EXPOSE 8080

CMD ["./main"]
```