



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

---

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01 Информатика и вычислительная техника

**О Т Ч Е Т**

**по лабораторной работе № 11**

**Название:** Аутентификация пользователей с помощью jwt-токена

**Дисциплина:** Языки интернет программирования

Студент

ИУ6-33Б

(Группа)

20.12.24

(Подпись, дата)

Пономаренко  
В.М.

(И.О. Фамилия)

Преподаватель

20.12.24

(Подпись, дата)

Шульман В.Д.

(И.О. Фамилия)

Москва, 2024

Цель работы: получение первичных знаний в области авторизации и аутентификации в контексте веб-приложений

Задание: Модернизировать код сервисов hello, count и query, добавив авторизацию с помощью jwt-токенов. Создать отдельный сервис auth для регистрации, авторизации и аутентификации

Ход работы:

/cmd – точка входа в приложение

/configs – статические конфигурации в формате YAML

/internal – модули, не предназначенные для использования вне проекта (работа с БД, API, )

/pkg – библиотеки и пакеты

## Auth

Листинг программы по файлам:

### cmd/auth/main.go

```
package main

import (
    "flag"
    "log"

    _ "github.com/lib/pq"
    "github.com/vera2005/lr11/internal/auth/api"
    "github.com/vera2005/lr11/internal/auth/config"
    "github.com/vera2005/lr11/internal/auth/provider"
    "github.com/vera2005/lr11/internal/auth/usecase"
)

func main() {
    // Считываем аргументы командной строки
    configPath := flag.String("config-path", "D:\\Go\\FINL\\lr11\\configs\\auth.yaml", "путь к файлу конфигурации")
    flag.Parse()
    cfg, err := config.LoadConfig(*configPath)
    if err != nil {
        log.Fatal(err)
    }

    //Инициализация провайдера
    prv := provider.NewProvider(cfg.DB.Host, cfg.DB.Port, cfg.DB.User, cfg.DB.Password, cfg.DB.DBname)
    //Инициализация бизнес-логики
    use := usecase.NewUsecase(prv)
    //Инициализация сервера
    srv := api.NewServer(cfg.IP, cfg.Port, use)
    srv.Run()
}
```

### configs/ auth.yaml

```
ip: "127.0.0.1"
port: 8082
api:
usecase:
db:
  host: "localhost"
  port: 5432
  user: "postgres"
  password: "catjkm8800"
  dbname: "users"
```

### internal/ auth /api/api.go

```
package api

import (
    "fmt"

    "github.com/labstack/echo/v4"
)

type Server struct {
    Server *echo.Echo
    address string

    uc Usecase
}
```

```

func NewServer(ip string, port int, uc Usecase) *Server {
    api := Server{
        uc: uc,
    }
    api.Server = echo.New()
    api.Server.POST("/reg", api.signUp)
    api.Server.POST("/au", api.signIn)
    //api.Server.Use(JWTMiddleware)
    api.address = fmt.Sprintf("%s:%d", ip, port)

    return &api
}

func (api *Server) Run() {
    api.Server.Logger.Fatal(api.Server.Start(api.address))
}

```

## internal/ auth /api/handler.go

```

package api

import (
    "errors"
    "fmt"
    "net/http"

    "github.com/labstack/echo/v4"
)

var ErrEmailAlreadyTaken = errors.New("email уже занят")

//middleware - функция, которая при обработке запросов может выполнять доп действия до или после вызова обработчика
// проверяет наличие и валидность JWT в заголовке Authorization каждого запроса
// next - следующий обработчик
// возвращает новую функцию, соответствующую типу echo.HandlerFunc, которая будет использоваться как middleware

// signUp - обработчик для регистрации пользователя
func (srv *Server) signUp(c echo.Context) error {
    fmt.Println("user signUp from api")
    var user User
    if err := c.Bind(&user); err != nil {
        fmt.Println("ppp")
        return c.JSON(http.StatusBadRequest, Response{
            Message: "некорректное считывание данных",
        })
    }
    fmt.Println("user signUp from api")
    // Вызываем бизнес-логику для регистрации пользователя
    token, err := srv.uc.SignUp(user)
    if err != nil {
        fmt.Println(err.Error(), errors.Is(err, ErrEmailAlreadyTaken))
        if errors.Is(err, ErrEmailAlreadyTaken) { // Проверяем конкретную ошибку
            fmt.Println("rrr")
            return c.JSON(http.StatusConflict, Response{
                Message: err.Error(),
            })
        }
        return c.JSON(http.StatusInternalServerError, Response{
            Message: "ошибка сервера",
        })
    }
    fmt.Println("signUp succs")

    return c.JSON(http.StatusCreated, Response{
        Message: token,
    })
}

// Обработчик для аутентификации пользователя
func (srv *Server) signIn(c echo.Context) error {
    var credentials Credentials
    if err := c.Bind(&credentials); err != nil {
        fmt.Println("Invalid input")
        return c.JSON(http.StatusBadRequest, Response{
            Message: " Ошибка передачи параметров",
        })
    }
    token, err := srv.uc.SignIn(credentials)
    if err != nil {
        fmt.Println("Authentication failed")
        return c.JSON(http.StatusUnauthorized, Response{
            Message: " Ошибка сервера",
        })
    }

    return c.JSON(http.StatusOK, Response{
        Message: token,
    })
}

func (srv *Server) hello(c echo.Context) error {

```

```

    return c.JSON(http.StatusOK, Response{
        Message: "Hello!",
    })
}

```

## internal/ auth /api/interface.go

```

package api

type Usecase interface {
    SignUp(User) (string, error)
    SignIn(Credentials) (string, error) //Возвращает JWT токен,
}

```

## internal/ auth /api/middlleware.go

```

package api

import (
    "fmt"
    "net/http"
    "strings"

    "github.com/golang-jwt/jwt/v4"
    "github.com/labstack/echo/v4"
)

func JWTMiddleware(next echo.HandlerFunc) echo.HandlerFunc {
    //контекст содержит информацию о текущем HTTP-запросе и позволяет взаимодействовать с ним
    fmt.Println("ppp")
    return func(c echo.Context) error {
        token := c.Request().Header.Get("Authorization")
        if token == "" {
            return c.JSON(http.StatusUnauthorized, map[string]string{"error": "missing token"})
        }

        token = strings.TrimPrefix(token, "Bearer ") //удаление префикса
        claims := &Claims{} // Claims - это структура, которая содержит данные о пользователе
        tkn, err := jwt.ParseWithClaims(token, claims, func(token *jwt.Token) (interface{}, error) {
            return []byte("your_secret_key"), nil
        })

        if err != nil || !tkn.Valid {
            return c.JSON(http.StatusUnauthorized, map[string]string{"error": "invalid token"})
        }

        c.Set("userId", claims.UserId) // Сохраняем ID пользователя в контексте
        return next(c)
    }
}

func TokenValidationHandler(c echo.Context) error {
    token := c.QueryParam("token")
    if token == "" {
        return c.JSON(http.StatusBadRequest, map[string]string{"error": "missing token"})
    }

    _, err := jwt.Parse(token, func(token *jwt.Token) (interface{}, error) {
        return "your_secret_key", nil
    })

    if err != nil {
        return c.JSON(http.StatusUnauthorized, map[string]string{"error": "invalid token"})
    }

    return c.JSON(http.StatusOK, map[string]string{"message": "valid token"})
}

```

## internal/ auth /api/structs.go

```

package api

import "github.com/golang-jwt/jwt/v4"

type User struct {
    Id          int    `json:"-`
    Name        string `json:"username"`
    Email       string `json:"email"`
    HashedPassword string `json:"password"`
}

//LoginRequest
type Credentials struct {
    Email    string `json:"email"`
    Password string `json:"password"`
}

type Claims struct {
    UserId int `json:"userId"`
}

```

```

    jwt.RegisteredClaims
}

type RegisterRequest struct {
    Email    string `json:"email"`
    Name     string `json:"name"`
    Password string `json:"password"`
}

type Response struct {
    Message string `json:"message"`
}

```

## internal/ auth /config/ config.go

```

package config

type Config struct {
    IP    string `yaml:"ip"`
    Port int    `yaml:"port"`
    DB    db      `yaml:"db"`
}

type db struct {
    Host     string `yaml:"host"`
    Port     int    `yaml:"port"`
    User     string `yaml:"user"`
    Password string `yaml:"password"`
    DBName   string `yaml:"dbname"`
}

```

## internal/ auth /config/load.go

```

package config

import (
    "io/ioutil"
    "path/filepath"

    "gopkg.in/yaml.v3"
)

func LoadConfig(pathToFile string) (*Config, error) {
    filename, err := filepath.Abs(pathToFile)
    if err != nil {
        return nil, err
    }

    yamlFile, err := ioutil.ReadFile(filename)
    if err != nil {
        return nil, err
    }

    var cfg Config

    err = yaml.Unmarshal(yamlFile, &cfg)
    if err != nil {
        return nil, err
    }

    return &cfg, nil
}

```

## internal/ auth /provider/ provider.go

```

package provider

import (
    "database/sql"
    "fmt"
    "log"
)

type Provider struct {
    conn *sql.DB
}

func NewProvider(host string, port int, user, password, dbName string) *Provider {
    psqlInfo := fmt.Sprintf("host=%s port=%d user=%s "+
        "password=%s dbname=%s sslmode=disable",
        host, port, user, password, dbName)

```

```

    // Создание соединения с сервером postgres
    conn, err := sql.Open("postgres", psqlInfo)
    if err != nil {
        log.Fatal(err)
    }

    return &Provider{conn: conn}
}

}

```

## internal/ auth /provider/sql.go

```

package provider

import (
    "database/sql"
    "errors"
    "fmt"

    "github.com/vera2005/lr11/internal/auth/api"
)

func (p *Provider) CheckUser(u api.User) (api.User, error) {
    var user api.User
    err := p.conn.QueryRow(`SELECT id, email, name FROM users WHERE email = $1`, u.Email).Scan(&user.Id, &user.Email, &user.Name)

    fmt.Println("check user")

    if err != nil {
        if err == sql.ErrNoRows {
            // Если пользователь не найден, возвращаем пустого пользователя и nil как ошибку
            return api.User{}, nil
        }
        // Возвращаем ошибку при проблеме с запросом
        return api.User{}, fmt.Errorf("ошибка при проверке пользователя: %w", err)
    }
    fmt.Println("checked from check")
    // Если пользователь найден, возвращаем его и ошибку о том, что почта занята
    return user, nil // Возвращаем найденного пользователя без ошибки
}

// Определение ошибки для занятости email
var ErrEmailAlreadyTaken = errors.New("email уже занят")

func (p *Provider) CreateUser(u api.User) error {
    _, err := p.conn.Exec("INSERT INTO users (name, email, hashedPassword) VALUES ($1, $2, $3)", u.Name, u.Email, u.HashedPassword)
    fmt.Println("create user")
    if err != nil {
        return err
    }
    return nil
}

func (p *Provider) SelectUser(email string) (api.User, error) {
    var user api.User
    err := p.conn.QueryRow("SELECT id, name, email, hashedPassword FROM users WHERE email = $1", email).Scan(&user.Id, &user.Name, &user.Email, &user.HashedPassword)
    if err != nil {
        if err == sql.ErrNoRows {
            return api.User{}, errors.New("invalid credentials") // Пользователь не найден
        }
        return api.User{}, err // Возвращаем ошибку при проблеме с запросом
    }
    return user, nil
}

//func (p *Provider) CheckIsAuthor() (bool, error)

```

## internal/ auth /usecase/interface.go

```

package usecase

import "github.com/vera2005/lr11/internal/auth/api"

type Provider interface {
    CheckUser(api.User) (api.User, error)
    CreateUser(api.User) error
    SelectUser(string) (api.User, error)
}

```

## internal/ auth /usecase/ auth.go

```

package usecase

```

```
// реализуем бизнес-логику, ей вызывают обработчики запросов, а она передает задачу бд
import (
    "fmt"

    "github.com/vera2005/lr11/internal/auth/api"
    "github.com/vera2005/lr11/utlils"
)

func (u *Usecase) SignUp(user api.User) (string, error) {
    // Проверяем, существует ли уже пользователь с таким email
    existingUser, err := u.p.CheckUser(user)
    if err != nil {
        fmt.Println(err)
        return "", err // Обработка ошибки при проверке пользователя
    }
    fmt.Println("checked user")
    if existingUser.Email != "" {
        fmt.Println("email")
        return "", api.ErrEmailAlreadyTaken // Возвращаем конкретное сообщение об ошибке
    }
    // Хешируем пароль перед сохранением
    hashedPassword, err := utlils.HashPassword(user.HashedPassword) // Измените на user.Password
    if err != nil {
        fmt.Println("pas error")
        return "", err
    }
    newUser := user
    newUser.HashedPassword = hashedPassword

    // Сохраняем нового пользователя в базу данных
    err = u.p.CreateUser(newUser)
    if err != nil {
        fmt.Println("error of create")
        return "", err
    }
    token, err := utlils.GenerateToken(user.Id)
    if err != nil {
        return "", err // Возвращаем ошибку при генерации токена
    }
    fmt.Println("SignUp ssuccsesf")
    return token, nil
}

// SignIn аутентифицирует пользователя и возвращает JWT токен
func (u *Usecase) SignIn(credentials api.Credentials) (string, error) {
    var user api.User
    // Получаем пользователя по email
    user, err := u.p.SelectUser(credentials.Email)
    if err != nil {
        fmt.Println("user not found")
        return "", err
    }
    // Сравниваем хешированный пароль с введенным паролем
    if err := utlils.ComparePasswords(user.HashedPassword, credentials.Password); err != nil {
        fmt.Println("password not compared", utlils.ComparePasswords(user.HashedPassword, credentials.Password))
        return "", err
    }
    fmt.Println("password is correct")
    // Генерация JWT токена после успешной аутентификации
    token, err := utlils.GenerateToken(user.Id)
    if err != nil {
        return "", err // Возвращаем ошибку при генерации токена
    }
    return token, nil // Возвращаем сгенерированный токен
}

```

## internal/ auth /usecase/usecase.go

```
package usecase

type Usecase struct {
    p Provider
}

func NewUsecase(p Provider) *Usecase {
    return &Usecase{
        p: p,
    }
}

```

## pkg/consts/default.go

```
package consts

```

```
const (  
    IP = "127.0.0.1"  
)
```

## pkg/vars/err.go

```
package vars  
  
import "errors"  
  
var (  
    ErrAlreadyExist = errors.New("already exist")  
)
```


## utils/utils.go

```
package utils  
  
import (  
    "fmt"  
    "time"  
  
    "github.com/vera2005/lr11/internal/auth/api"  
  
    "github.com/golang-jwt/jwt/v4"  
    "golang.org/x/crypto/bcrypt"  
)  
  
var jwtSecret = []byte("your_secret_key") // ваш секретный ключ  
  
// HashPassword хеширует пароль с помощью bcrypt  
func HashPassword(password string) (string, error) {  
    fmt.Println(password, "dddd")  
    bytes, err := bcrypt.GenerateFromPassword([]byte(password), bcrypt.DefaultCost)  
    fmt.Println("password for hash", password, string(bytes))  
    return string(bytes), err  
}  
  
// GenerateToken создает новый JWT-токен  
func GenerateToken(userID int) (string, error) {  
    claims := &api.Claims{  
        UserID: userID,  
        RegisteredClaims: jwt.RegisteredClaims{  
            ExpiresAt: jwt.NewNumericDate(time.Now().Add(24 * time.Hour)), // Токен истекает через 24 часа  
            Issuer:    "your_app_name",  
        },  
    }  
    token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)  
    return token.SignedString(jwtSecret)  
}  
  
// ComparePasswords сравнивает хешированный пароль с введенным паролем  
func ComparePasswords(hashPassword string, password string) error {  
    fmt.Println(password)  
    return bcrypt.CompareHashAndPassword([]byte(hashPassword), []byte(password))  
}
```

## Демонстрация результатов тестирования

Отметим, что пароли в БД хранятся в хешированном виде. Также есть проверка на то, что почта еще не является привязанной к аккаунту пользователя.



 **http://127.0.0.1:8082/reg**

Save

Share

POST

http://127.0.0.1:8082/reg

Send

ParamsAuthorizationHeaders (9)BodyScriptsSettings

none

form-data

x-www-form-urlencoded

raw

binary



GraphQL

JSON

Beautify

```
1 {
2   "username": "vera",
3   "email": "vera@mail",
4   "password": "1"
5 }
```

BodyCookiesHeaders (3)Test Results

201 Created • 225 ms • 278 B •  




Pretty

Raw


Preview

Visualize

JSON



```
1 {
2   "message": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJAsImIzcyI6InlvdXJfYXBwX25hbWUiLCJleHAiOiJlE3MzQ3ODAxNDV9.Lpp6RN0nDsX1g1Jkc1JTQDYmIQZxTe6u15kwYm-4"
3 }
```

 **http://127.0.0.1:8082/au**

Save

Share

POST

http://127.0.0.1:8082/au

Send

ParamsAuthorizationHeaders (9)BodyScriptsSettings

none

form-data

x-www-form-urlencoded

raw

binary



GraphQL

JSON

Beautify

```
1 {
2   "email": "vera@mail",
3   "password": "1"
4 }
```

BodyCookiesHeaders (3)Test Results

200 OK • 145 ms • 275 B •  




Pretty

Raw

Preview

Visualize

JSON



```
1 {
2   "message": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJlE3MzQ3ODAxNDV9.Lpp6RN0nDsX1g1Jkc1JTQDYmIQZxTe6u15kwYm-4"
3 }
```

POST

http://127.0.0.1:8082/au

Send

ParamsAuthorizationHeaders (9)BodyScriptsSettings

noneform-datax-www-form-urlencodedrawbinaryGraphQLJSON

```
1 {
2   "email": "vera@m",
3   "password": "1"
4 }
```

BodyCookiesHeaders (3)Test Results401 Unauthorized3 ms161 B

PrettyRawPreviewVisualizeJSON

```
1 {
2   "message": "Ошибка сервера"
3 }
```

POST

http://127.0.0.1:8082/reg

Send

ParamsAuthorizationHeaders (9)BodyScriptsSettings

noneform-datax-www-form-urlencodedrawbinaryGraphQLJSON

```
1 {
2   "username": "qqq",
3   "email": "vera@mail",
4   "password": "1"
5 }
```

BodyCookiesHeaders (3)Test Results409 Conflict4 ms152 B

PrettyRawPreviewVisualizeJSON

```
1 {
2   "message": "email уже занят"
3 }
```

## Задание 1. Hello

1. Функционал аналогичен функционалу в предыдущих лабораторных работах  
Ниже приведены листинги

### cmd/hello/main.go

```
package main

import (
    "flag"
    "log"

    _ "github.com/lib/pq"
    "github.com/vera2005/lr10/internal/query/api"
    "github.com/vera2005/lr10/internal/query/config"
    "github.com/vera2005/lr10/internal/query/provider"
    "github.com/vera2005/lr10/internal/query/usecase"
)

func main() {
    // Считываем аргументы командной строки
    configPath := flag.String("config-path", "D:\\Go\\lr10\\configs\\query.yaml", "путь к файлу конфигурации")
    flag.Parse()

    cfg, err := config.LoadConfig(*configPath)
    if err != nil {
        log.Fatal(err)
    }

    //Инициализация провайдера
    prv := provider.NewProvider(cfg.DB.Host, cfg.DB.Port, cfg.DB.User, cfg.DB.Password, cfg.DB.DBname)
    //Инициализация бизнес-логики
    use := usecase.NewUsecase(cfg.Usecase.DefaultMessage, prv)
    //Инициализация сервера
    srv := api.NewServer(cfg.IP, cfg.Port, cfg.API.MaxMessageSize, use)

    srv.Run()
}
```

### configs/ hello.yaml

```
ip: "127.0.0.1"
port: 8081
api:
  max_message_size: 32
usecase:
  default_message: "User"
db:
  host: "localhost"
  port: 5432
  user: "postgres"
  password: "catjkm8800"
  dbname: "hello"
```

### internal/ hello /api/api.go

```
package api

import (
    "fmt"

    "github.com/labstack/echo/v4"
)

type Server struct {
    maxSize int

    server *echo.Echo
    address string

    uc Usecase
}

func NewServer(ip string, port int, maxSize int, uc Usecase) *Server {
    api := Server{
        maxSize: maxSize,
    }
```

```

        uc:      uc,
    }

    api.server = echo.New()
    api.server.Use(JWTMiddleware)

    api.server.GET("/query", api.GetQuery)
    api.server.POST("/query", api.PostQuery)
    api.server.PUT("/query", api.PutQuery)

    api.address = fmt.Sprintf("%s:%d", ip, port)

    return &api
}

func (api *Server) Run() {
    api.server.Logger.Fatal(api.server.Start(api.address))
}

```

## internal/ hello /api/handler.go

```

package api

import (
    "errors"
    "net/http"
    "regexp"

    "github.com/labstack/echo/v4"
    "github.com/vera2005/lr10/pkg/vars"
)

// GetQuery возвращает приветствие случайному пользователю
func (srv *Server) GetQuery(e echo.Context) error {
    msg, err := srv.uc.FetchQuery()
    if err != nil {
        return e.String(http.StatusInternalServerError, err.Error())
    }

    return e.JSON(http.StatusOK, msg)
}

// PostQuery Помещает нового пользователя в БД
func (srv *Server) PostQuery(c echo.Context) error {
    nameInput := c.QueryParam("name") // Получаем Query-параметр
    if nameInput == "" {
        return c.String(http.StatusBadRequest, "Missing 'name' query parameter")
    }
    //проверка на корректность имени
    re := regexp.MustCompile(`[a-zA-Za-zA-Я]`)
    if !re.MatchString(nameInput) {
        return c.String(http.StatusBadRequest, "empty string")
    }
    if len([]rune(nameInput)) > srv.maxSize {
        return c.String(http.StatusBadRequest, "name too large")
    }
    err := srv.uc.SetQuery(nameInput)
    if err != nil {
        if errors.Is(err, vars.ErrAlreadyExist) {
            return c.String(http.StatusConflict, err.Error())
        }
        return c.String(http.StatusInternalServerError, err.Error())
    }
    return c.String(http.StatusCreated, "OK")
}

func (srv *Server) PutQuery(c echo.Context) error {
    nameInput := c.QueryParam("name") // Получаем Query-параметр
    if nameInput == "" {
        return c.String(http.StatusBadRequest, "Missing 'name' query parameter")
    }
    //проверка на корректность имени
    re := regexp.MustCompile(`[a-zA-Za-zA-Я]`)
    if !re.MatchString(nameInput) {
        return c.String(http.StatusBadRequest, "empty string")
    }
    if len([]rune(nameInput)) > srv.maxSize {
        return c.String(http.StatusBadRequest, "name too large")
    }

```

```

    }
    err := srv.uc.ChangeQuery(nameInput)
    if err != nil {
        if errors.Is(err, vars.ErrAlreadyExist) {
            return c.String(http.StatusConflict, err.Error())
        }
        return c.String(http.StatusInternalServerError, err.Error())
    }
    return c.String(http.StatusCreated, "OK")
}

```

## internal/ hello /api/interface.go

```

package api

type Usecase interface {
    FetchQuery() (string, error)
    SetQuery(name string) error
    ChangeQuery(name string) error
}

```

## internal/ hello /api/midl.go

```

package api

import (
    "encoding/json"
    "fmt"
    "net/http"
    "strings"

    "github.com/golang-jwt/jwt/v4"
    "github.com/labstack/echo/v4"
)

// Claims структура для хранения данных о пользователе
type Claims struct {
    UserId json.Number `json:"userId"` // ID пользователя
    jwt.RegisteredClaims
}

var secretKey = []byte("your_secret_key") // Тот же секретный ключ, что и на сервере аутентификации

// JWTMiddleware проверяет наличие и валидность JWT-токена
func JWTMiddleware(next echo.HandlerFunc) echo.HandlerFunc {
    return func(c echo.Context) error {
        token := c.Request().Header.Get("Authorization")
        if token == "" {
            return c.JSON(http.StatusUnauthorized, map[string]string{"error": "missing token"})
        }

        token = strings.TrimPrefix(token, "Bearer ")
        claims := &Claims{}

        tkn, err := jwt.ParseWithClaims(token, claims, func(token *jwt.Token) (interface{}, error) {
            return secretKey, nil
        })

        if err != nil || !tkn.Valid {
            fmt.Println(tkn.Valid, err)
            return c.JSON(http.StatusUnauthorized, map[string]string{"error": "invalid token"})
        }

        c.Set("userId", claims.UserId) // Сохраняем информацию о пользователе в контексте
        return next(c)
    }
}

```

## internal/ hello /config/ config.go

```

package config

type Config struct {
    IP    string `yaml:"ip"`
    Port int    `yaml:"port"`

    API    api    `yaml:"api"`
    Usecase usecase `yaml:"usecase"`
    DB     db     `yaml:"db"`
}

```

```

}

type api struct {
    MaxMessageSize int `yaml:"max_message_size"`
}

type usecase struct {
    DefaultMessage string `yaml:"default_message"`
}

type db struct {
    Host     string `yaml:"host"`
    Port     int    `yaml:"port"`
    User     string `yaml:"user"`
    Password string `yaml:"password"`
    DBname   string `yaml:"dbname"`
}

```

internal/ hello /config/load.go

```

package config

import (
    "gopkg.in/yaml.v3"
    "io/ioutil"
    "path/filepath"
)

func LoadConfig(pathToFile string) (*Config, error) {
    filename, err := filepath.Abs(pathToFile)
    if err != nil {
        return nil, err
    }

    yamlFile, err := ioutil.ReadFile(filename)
    if err != nil {
        return nil, err
    }

    var cfg Config

    err = yaml.Unmarshal(yamlFile, &cfg)
    if err != nil {
        return nil, err
    }

    return &cfg, nil
}

```

internal/ hello /provider/ provider.go

```

package provider

import (
    "database/sql"
    "fmt"
    "log"
)

type Provider struct {
    conn *sql.DB
}

func NewProvider(host string, port int, user, password, dbName string) *Provider {
    psqlInfo := fmt.Sprintf("host=%s port=%d user=%s "+
        "password=%s dbname=%s sslmode=disable",
        host, port, user, password, dbName)

    // Создание соединения с сервером postgres
    conn, err := sql.Open("postgres", psqlInfo)
    if err != nil {
        log.Fatal(err)
    }

    return &Provider{conn: conn}
}

```

```
}
```

## internal/ hello /provider/sql.go

```
package provider

import (
    "database/sql"
    "errors"
)

func (p *Provider) SelectName() (string, error) {
    var msg string
    err := p.conn.QueryRow("SELECT name FROM query ORDER BY id DESC LIMIT 1").Scan(&msg)
    if err != nil {
        if errors.Is(err, sql.ErrNoRows) {
            return "", nil
        }
        return "", err
    }

    return msg, nil
}

func (p *Provider) InsertQuery(msg string) error {
    _, err := p.conn.Exec("INSERT INTO query (name) VALUES ($1)", msg)
    if err != nil {
        return err
    }

    return nil
}

func (p *Provider) UpdateQuery(n string) error {
    _, err := p.conn.Exec("UPDATE query SET name = $1 WHERE id = (SELECT MAX(id) FROM query)", n)
    if err != nil {
        return err
    }

    return nil
}
```

## internal/ hello /usecase/interface.go

```
package usecase

type Provider interface {
    SelectName() (string, error)
    InsertQuery(string) error
    UpdateQuery(string) error
}
```

## internal/ hello /usecase/ hello.go

```
package usecase

func (u *Usecase) FetchQuery() (string, error) {
    nam, err := u.p.SelectName()
    msg := "Hello," + nam + "!"
    if err != nil {
        return "", err
    }

    if msg == "" {
        return u.defaultMsg, nil
    }

    return msg, nil
}

func (u *Usecase) SetQuery(name string) error {
    err := u.p.InsertQuery(name)
    if err != nil {
        return err
    }
}
```

```

    return nil
}

func (u *Usecase) ChangeQuery(name string) error {
    err := u.p.UpdateQuery(name)
    if err != nil {
        return err
    }

    return nil
}

```

internal/ hello /usecase/usecase.go

```

package usecase

type Usecase struct {
    defaultMsg string

    p Provider
}

func NewUsecase(defaultMsg string, p Provider) *Usecase {
    return &Usecase{
        defaultMsg: defaultMsg,
        p:          p,
    }
}

```

pkg/consts/default.go

```

package consts

const (
    IP = "127.0.0.1"
)

```

pkg/vars/err.go

```

package vars

import "errors"

var (
    ErrAlreadyExist = errors.New("already exist")
)

```

## Задание 2. Query

2. Функционал аналогичен функционалу в предыдущих лабораторных работах  
Ниже приведены листинги

cmd/query/main.go

```

package main

import (
    "flag"
    "log"

    _ "github.com/lib/pq"
    "github.com/vera2005/lr10/internal/query/api"
    "github.com/vera2005/lr10/internal/query/config"
    "github.com/vera2005/lr10/internal/query/provider"
    "github.com/vera2005/lr10/internal/query/usecase"

```



```

)

func main() {
    // Считываем аргументы командной строки
    configPath := flag.String("config-path", "D:\\Go\\lr10\\\\configs\\query.yaml", "путь к файлу конфигурации")
    flag.Parse()

    cfg, err := config.LoadConfig(*configPath)
    if err != nil {
        log.Fatal(err)
    }

    //Инициализация провайдера
    prv := provider.NewProvider(cfg.DB.Host, cfg.DB.Port, cfg.DB.User, cfg.DB.Password, cfg.DB.DBname)
    //Инициализация бизнес-логики
    use := usecase.NewUsecase(cfg.Usecase.DefaultMessage, prv)
    //Инициализация сервера
    srv := api.NewServer(cfg.IP, cfg.Port, cfg.API.MaxMessageSize, use)

    srv.Run()
}

```

## configs/query.yaml

```

ip: "127.0.0.1"
port: 8081
api:
    max_message_size: 32
usecase:
    default_message: "User"
db:
    host: "localhost"
    port: 5432
    user: "postgres"
    password: "catjkm8800"
    dbname: "query"

```

## internal/query/api/api.go

```

package api

import (
    "fmt"

    "github.com/labstack/echo/v4"
)

type Server struct {
    maxSize int

    server *echo.Echo
    address string

    uc Usecase
}

func NewServer(ip string, port int, maxSize int, uc Usecase) *Server {
    api := Server{
        maxSize: maxSize,
        uc:      uc,
    }

    api.server = echo.New()
    api.server.GET("/query", api.GetQuery)
    api.server.POST("/query", api.PostQuery)
    api.server.PUT("/query", api.PutQuery)

    api.address = fmt.Sprintf("%s:%d", ip, port)

    return &api
}

func (api *Server) Run() {
    api.server.Logger.Fatal(api.server.Start(api.address))
}

```

## internal/query/api/handler.go

```
package api

import (
    "errors"
    "net/http"
    "regexp"

    "github.com/labstack/echo/v4"
    "github.com/vera2005/lr10/pkg/vars"
)

// GetQuery возвращает приветствие случайному пользователю
func (srv *Server) GetQuery(e echo.Context) error {
    msg, err := srv.uc.FetchQuery()
    if err != nil {
        return e.String(http.StatusInternalServerError, err.Error())
    }

    return e.JSON(http.StatusOK, msg)
}

// PostQuery Помещает нового пользователя в БД
func (srv *Server) PostQuery(c echo.Context) error {
    nameInput := c.QueryParam("name") // Получаем Query-параметр
    if nameInput == "" {
        return c.String(http.StatusBadRequest, "Missing 'name' query parameter")
    }
    //проверка на корректность имени
    re := regexp.MustCompile(`[a-zA-Za-яА-Я]`)
    if !re.MatchString(nameInput) {
        return c.String(http.StatusBadRequest, "empty string")
    }
    if len([]rune(nameInput)) > srv.maxSize {
        return c.String(http.StatusBadRequest, "name too large")
    }
    err := srv.uc.SetQuery(nameInput)
    if err != nil {
        if errors.Is(err, vars.ErrAlreadyExist) {
            return c.String(http.StatusConflict, err.Error())
        }
        return c.String(http.StatusInternalServerError, err.Error())
    }
    return c.String(http.StatusCreated, "OK")
}

func (srv *Server) PutQuery(c echo.Context) error {
    nameInput := c.QueryParam("name") // Получаем Query-параметр
    if nameInput == "" {
        return c.String(http.StatusBadRequest, "Missing 'name' query parameter")
    }
    //проверка на корректность имени
    re := regexp.MustCompile(`[a-zA-Za-яА-Я]`)
    if !re.MatchString(nameInput) {
        return c.String(http.StatusBadRequest, "empty string")
    }
    if len([]rune(nameInput)) > srv.maxSize {
        return c.String(http.StatusBadRequest, "name too large")
    }
    err := srv.uc.ChangeQuery(nameInput)
    if err != nil {
        if errors.Is(err, vars.ErrAlreadyExist) {
            return c.String(http.StatusConflict, err.Error())
        }
        return c.String(http.StatusInternalServerError, err.Error())
    }
    return c.String(http.StatusCreated, "OK")
}
```

## internal/query/api/interface.go

```
package api

type Usecase interface {
    FetchQuery() (string, error)
    SetQuery(name string) error
    ChangeQuery(name string) error
}
```

```
}
```

## internal/query/config/ config.go

```
package config

type Config struct {
    IP    string `yaml:"ip"`
    Port  int    `yaml:"port"`

    API    api    `yaml:"api"`
    Usecase usecase `yaml:"usecase"`
    DB     db     `yaml:"db"`
}

type api struct {
    MaxMessageSize int `yaml:"max_message_size"`
}

type usecase struct {
    DefaultMessage string `yaml:"default_message"`
}

type db struct {
    Host    string `yaml:"host"`
    Port    int    `yaml:"port"`
    User    string `yaml:"user"`
    Password string `yaml:"password"`
    DBname  string `yaml:"dbname"`
}
```

## internal/query/config/load.go

```
package config

import (
    "gopkg.in/yaml.v3"
    "io/ioutil"
    "path/filepath"
)

func LoadConfig(pathToFile string) (*Config, error) {
    filename, err := filepath.Abs(pathToFile)
    if err != nil {
        return nil, err
    }

    yamlFile, err := ioutil.ReadFile(filename)
    if err != nil {
        return nil, err
    }

    var cfg Config

    err = yaml.Unmarshal(yamlFile, &cfg)
    if err != nil {
        return nil, err
    }

    return &cfg, nil
}
```

## internal/query/provider/ provider.go

```
package provider

import (
    "database/sql"
    "fmt"
    "log"
)

type Provider struct {
```

```

    conn *sql.DB
}

func NewProvider(host string, port int, user, password, dbName string) *Provider {
    psqlInfo := fmt.Sprintf("host=%s port=%d user=%s "+
        "password=%s dbname=%s sslmode=disable",
        host, port, user, password, dbName)

    // Создание соединения с сервером postgres
    conn, err := sql.Open("postgres", psqlInfo)
    if err != nil {
        log.Fatal(err)
    }

    return &Provider{conn: conn}
}

```

## internal/query/provider/sql.go

```

package provider

import (
    "database/sql"
    "errors"
)

func (p *Provider) SelectName() (string, error) {
    var msg string
    err := p.conn.QueryRow("SELECT name FROM query ORDER BY id DESC LIMIT 1").Scan(&msg)
    if err != nil {
        if errors.Is(err, sql.ErrNoRows) {
            return "", nil
        }
        return "", err
    }

    return msg, nil
}

func (p *Provider) InsertQuery(msg string) error {
    _, err := p.conn.Exec("INSERT INTO query (name) VALUES ($1)", msg)
    if err != nil {
        return err
    }

    return nil
}

func (p *Provider) UpdateQuery(n string) error {
    _, err := p.conn.Exec("UPDATE query SET name = $1 WHERE id = (SELECT MAX(id) FROM query)", n)
    if err != nil {
        return err
    }

    return nil
}

```

## internal/query/usecase/interface.go

```

package usecase

type Provider interface {
    SelectName() (string, error)
    InsertQuery(string) error
    UpdateQuery(string) error
}

```

## internal/query/usecase/query.go

```

package usecase

func (u *Usecase) FetchQuery() (string, error) {
    nam, err := u.p.SelectName()
    msg := "Hello," + nam + "!"
    if err != nil {

```

```

        return "", err
    }

    if msg == "" {
        return u.defaultMsg, nil
    }

    return msg, nil
}

func (u *Usecase) SetQuery(name string) error {
    err := u.p.InsertQuery(name)
    if err != nil {
        return err
    }

    return nil
}

func (u *Usecase) ChangeQuery(name string) error {
    err := u.p.UpdateQuery(name)
    if err != nil {
        return err
    }

    return nil
}

```

#### internal/query/usecase/usecase.go

```

package usecase

type Usecase struct {
    defaultMsg string

    p Provider
}

func NewUsecase(defaultMsg string, p Provider) *Usecase {
    return &Usecase{
        defaultMsg: defaultMsg,
        p:          p,
    }
}

```

#### pkg/consts/default.go

```

package consts

const (
    IP = "127.0.0.1"
)

```

#### pkg/vars/err.go

```

package vars

import "errors"

var (
    ErrAlreadyExist = errors.New("already exist")
)

```

### Задаие 3. Count

1. Функционал аналогичен функционалу в предыдущих лабораторных работах  
Ниже приведены листинги

## cmd/count/main.go

```
package main

import (
    "flag"
    "log"

    _ "github.com/lib/pq"
    "github.com/vera2005/lr10/internal/count/api"
    "github.com/vera2005/lr10/internal/count/config"
    "github.com/vera2005/lr10/internal/count/provider"
    "github.com/vera2005/lr10/internal/count/usecase"
)

func main() {
    // Считываем аргументы командной строки
    configPath := flag.String("config-path", "D:\\Go\\lr10\\configs\\count.yaml", "путь к файлу конфигурации")
    flag.Parse()

    cfg, err := config.LoadConfig(*configPath)
    if err != nil {
        log.Fatal(err)
    }

    //Инициализация провайдера
    prv := provider.NewProvider(cfg.DB.Host, cfg.DB.Port, cfg.DB.User, cfg.DB.Password, cfg.DB.DBname)
    //Инициализация бизнес-логики
    use := usecase.NewUsecase(cfg.Usecase.DefaultCount, prv)
    //Инициализация сервера
    srv := api.NewServer(cfg.IP, cfg.Port, cfg.API.MaxNum, use)

    srv.Run()
}
```

## configs/ count.yaml

```
ip: "127.0.0.1"
port: 8081
api:
  max_number: 100000
usecase:
  default_count: "0"
db:
  host: "localhost"
  port: 5432
  user: "postgres"
  password: "catjkm8800"
  dbname: "count"
```

## internal/ count /api/api.go

```
package api

import (
    "fmt"

    "github.com/labstack/echo/v4"
)

type Server struct {
    maxNum int

    server *echo.Echo
    address string

    uc Usecase
}

func NewServer(ip string, port int, maxNum int, uc Usecase) *Server {
    api := Server{
        maxNum: maxNum,
        uc:     uc,
    }

    api.server = echo.New()
    api.server.GET("/count", api.GetCount)
    api.server.POST("/count", api.PostCount)
    api.server.PUT("/count", api.PutCount)

    api.address = fmt.Sprintf("%s:%d", ip, port)

    return &api
}

func (api *Server) Run() {
    api.server.Logger.Fatal(api.server.Start(api.address))
}
```

## internal/ count /api/handler.go

```
package api

import (
    "errors"
    "net/http"

    "github.com/labstack/echo/v4"
    "github.com/vera2005/lr10/pkg/vars"
)

type CountInput struct {
    Val float32 `json:"val"` // Используем float32 для автоматической проверки числового значения
}

func (srv *Server) GetCount(c echo.Context) error {
    msg, err := srv.uc.FetchCount()
    if err != nil {
        return c.String(http.StatusInternalServerError, err.Error())
    }

    return c.JSON(http.StatusOK, msg)
}

func (srv *Server) PostCount(c echo.Context) error {
    input := CountInput{}
    // Привязка входных данных и проверка на ошибки
    if err := c.Bind(&input); err != nil {
        return c.String(http.StatusBadRequest, "Неправильный формат JSON")
    }
    err := srv.uc.SetCount(input.Val)
    if err != nil {
        if errors.Is(err, vars.ErrAlreadyExist) {
            return c.String(http.StatusConflict, err.Error())
        }
        return c.String(http.StatusInternalServerError, err.Error())
    }
    return c.String(http.StatusCreated, "OK")
}

func (srv *Server) PutCount(c echo.Context) error {
    input := CountInput{}
    if err := c.Bind(&input); err != nil {
        return c.String(http.StatusBadRequest, "Неправильный формат JSON")
    }
    err := srv.uc.ChangeCount(input.Val)
    if err != nil {
        if errors.Is(err, vars.ErrAlreadyExist) {
            return c.String(http.StatusConflict, err.Error())
        }
        return c.String(http.StatusInternalServerError, err.Error())
    }
    return c.String(http.StatusCreated, "OK")
}
```

## internal/ count /api/interface.go

```
package api

type Usecase interface {
    FetchCount() (string, error)
    SetCount(float32) error
    ChangeCount(float32) error
}
```

## internal/ count /config/ config.go

```
package config

type Config struct {
    IP    string `yaml:"ip"`
    Port int    `yaml:"port"`

    API    api    `yaml:"api"`
    Usecase usecase `yaml:"usecase"`
    DB     db     `yaml:"db"`
}

type api struct {
    MaxNum int `yaml:"max_number"`
}

type usecase struct {
    DefaultCount string `yaml:"default_count"`
}

type db struct {
    Host    string `yaml:"host"`
    Port    int    `yaml:"port"`
    User    string `yaml:"user"`
}
```

```

    Password string `yaml:"password"`
    DBname    string `yaml:"dbname"`
}

```

## internal/ count /config/load.go

```

package config

import (
    "gopkg.in/yaml.v3"
    "io/ioutil"
    "path/filepath"
)

func LoadConfig(pathToFile string) (*Config, error) {
    filename, err := filepath.Abs(pathToFile)
    if err != nil {
        return nil, err
    }

    yamlFile, err := ioutil.ReadFile(filename)
    if err != nil {
        return nil, err
    }

    var cfg Config

    err = yaml.Unmarshal(yamlFile, &cfg)
    if err != nil {
        return nil, err
    }

    return &cfg, nil
}

```

## internal/ count /provider/ provider.go

```

package provider

import (
    "database/sql"
    "fmt"
    "log"
)

type Provider struct {
    conn *sql.DB
}

func NewProvider(host string, port int, user, password, dbName string) *Provider {
    psqlInfo := fmt.Sprintf("host=%s port=%d user=%s "+
        "password=%s dbname=%s sslmode=disable",
        host, port, user, password, dbName)

    // Создание соединения с сервером postgres
    conn, err := sql.Open("postgres", psqlInfo)
    if err != nil {
        log.Fatal(err)
    }

    return &Provider{conn: conn}
}

```

## internal/ count /provider/sql.go

```

package provider

import (
    "database/sql"
    "errors"
)

func (p *Provider) SelectCount() (string, error) {
    var msg string
    err := p.conn.QueryRow("SELECT summa FROM count ORDER BY id DESC LIMIT 1").Scan(&msg)
    if err != nil {
        if errors.Is(err, sql.ErrNoRows) {
            return "", nil
        }
        return "", err
    }

    return msg, nil
}

```



```
func (p *Provider) InsertCount(v float32) error {
    _, err := p.conn.Exec("INSERT INTO count (val, summa) VALUES ($1, $1 + (SELECT COALESCE(summa, 0) FROM count ORDER BY id DESC LIMIT 1))", v)
    if err != nil {
        return err
    }

    return nil
}

func (p *Provider) UpdateCount(v float32) error {
    _, err := p.conn.Exec("UPDATE count SET val = $1, summa = (val + (SELECT summa FROM count WHERE id = ((SELECT MAX(id) FROM count) - 1))) WHERE id = (SELECT MAX(id) FROM count)", v)
    if err != nil {
        return err
    }

    return nil
}
```

## internal/ count /usecase/interface.go

```
package usecase

type Provider interface {
    SelectCount() (string, error)
    InsertCount(float32) error
    UpdateCount(float32) error
}
```

## internal/ count /usecase/count.go

```
package usecase

func (u *Usecase) FetchCount() (string, error) {
    msg, err := u.p.SelectCount()
    if err != nil {
        return "", err
    }
    return msg, nil
}

func (u *Usecase) SetCount(v float32) error {
    err := u.p.InsertCount(v)
    if err != nil {
        return err
    }

    return nil
}

func (u *Usecase) ChangeCount(v float32) error {
    err := u.p.UpdateCount(v)
    if err != nil {
        return err
    }

    return nil
}
```

## internal/ count /usecase/usecase.go

```
package usecase

type Usecase struct {
    defaultMsg string

    p Provider
}

func NewUsecase(defaultMsg string, p Provider) *Usecase {
    return &Usecase{
        defaultMsg: defaultMsg,
        p:         p,
    }
}
```

## pkg/consts/default.go

```
package consts
```

```
const (
```

```
IP = "127.0.0.1"  
)
```

pkg/vars/err.go

```
package vars  
  
import "errors"  
  
var (  
    ErrAlreadyExist = errors.New("already exist")  
)
```

## Демонстрация результатов тестирования



http://127.0.0.1:8082/reg



Save



Share

</>

POST



http://127.0.0.1:8082/reg

Send



Params Auth Headers (9) **Body** Scripts Settings

Cookies

raw



JSON



Beautify

```
1  {  
2    "email": "1@11",  
3    "password": "1",  
4    "username": "123"  
5  }
```

Body



409 Conflict

• 134 ms • 152 B •



Pretty

Raw

Preview

Visualize

JSON



```
1  {  
2    "message": "email уже занят"  
3  }
```



```
1 {
2     "email": "1@1112",
3     "password": "1",
4     "username": "123"
5 }
```



- 144 ms • 278 B •  | 



🔗 📄 🔍

```
1 {
2   "message": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQ0IjAsIm1ldncyI6InlvdXJfYXBwX25hbWUuIChleHAiOjE3MzQ3NzYyMzR9.Fj6rS1nNmCuF50hZIH0FN1TjtY_stZb00QYxJnbP3fQ"
3 }
```



<

POST http: •

GET http: •

POST http: •

GET http: •

>

+

▼

No environment ▼

⌵

HTTP

http://127.0.0.1:8081/hello

Save ▼

Share

</>

GET ▼

http://127.0.0.1:8081/hello

Send ▼

↺

Params

Auth

Headers (6)

Body

Scripts

Settings

Cookies

Query Params

	Key	Value	Description	⋮	Bulk Edit
	Key	Value	Description		

Body ▼

🕒

401 Unauthorized • 5 ms • 144 B • 🌐 • ⋮

Pretty

Raw

Preview

Visualize

JSON ▼

⌵

🔗

📄

🔍

1 {

2 | "error": "missing token"

3 }

🔗 Postbot

▶ Runner

🔄 Start Proxy

🍪 Cookies

🏠 Vault

🗑️ Trash

⌵

?

GET

http://127.0.0.1:8081/hello

Send

ParamsAuthorizationHeaders (9)Body ●ScriptsSettingsCookies

Headers8 hidden

	Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp...				
	Key	Value	Description			

BodyCookiesHeaders (3)Test Results200 OK75 ms123 B

PrettyRawPreviewVisualizeJSON

```
1 "hello, world"
```

POST

http://127.0.0.1:8081/hello

Send

ParamsAuthorizationHeaders (9)Body ●ScriptsSettingsCookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON


Beautify

```
1 {
2   "msg": "qwertyyyyy"
3 }
```

BodyCookiesHeaders (3)Test Results201 Created19 ms123 B

PrettyRawPreviewVisualizeText

```
1 OK
```

 http://127.0.0.1:8081/query?name=Qwe

Save

Share

POST

http://127.0.0.1:8081/query?name=Qwe

Send

Params • Authorization Headers (8) Body Scripts Settings Cookies

Headers 

7 hidden


	Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp...				
	Key	Value	Description			

Body

Cookies

Headers (3)

Test Results

201 Created • 105 ms • 123 B • 

...

Pretty

Raw


Preview

Visualize

Text

...

1 OK

 http://127.0.0.1:8081/query?name=Qwe

Send

Params • Authorization Headers (7) Body Scripts Settings Cookies

Headers 

6 hidden


	Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp...				
	Key	Value	Description			

Body

Cookies

Headers (3)

Test Results

200 OK • 5 ms • 121 B • 

...

Pretty

Raw


Preview

Visualize

JSON

...

1 "Hello,Qwe!"

 http://127.0.0.1:8081/count

Save

Share

GET

http://127.0.0.1:8081/count

Send

Params

Authorization

Headers (7)

Body

Scripts

Settings

Cookies

Headers

6 hidden

	Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp...				
	Key	Value	Description			

Body

Cookies

Headers (3)

Test Results

200 OK

82 ms

116 B

...

Pretty

Raw

Preview

Visualize

JSON

...

1

"325.00"

POST

http://127.0.0.1:8081/count

Send

Params

Authorization

Headers (9)

Body

Scripts

Settings

Cookies

Beautify

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

{

2

"val":23

3

}

Body

Cookies

Headers (3)

Test Results

201 Created

15 ms

123 B

...

Pretty

Raw

Preview

Visualize

Text

...

1

OK

Выводы: В ходе выполнения данной лабораторной работы был изучен и применен на практике метод реализации регистрации, авторизации и аунтетификации.

Список использованных источников:



- 1) <https://github.com/golang-standards/project-layout?tab=readme-ov-file>
- 2) <https://youtu.be/V6lQG6d5LgU?si=17sjfwTYCWZMSHlw>