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 ИУ6-31Б _____ М.В, Грачева
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ В. Д. Шульман
(Подпись, дата) (И.О. Фамилия)

Москва, 2024

Цель работы

Получение первичных знаний в области авторизации и аутентификации в контексте веб-приложений

Ход работы

Делаем fork репозитория (Рисунок 1).

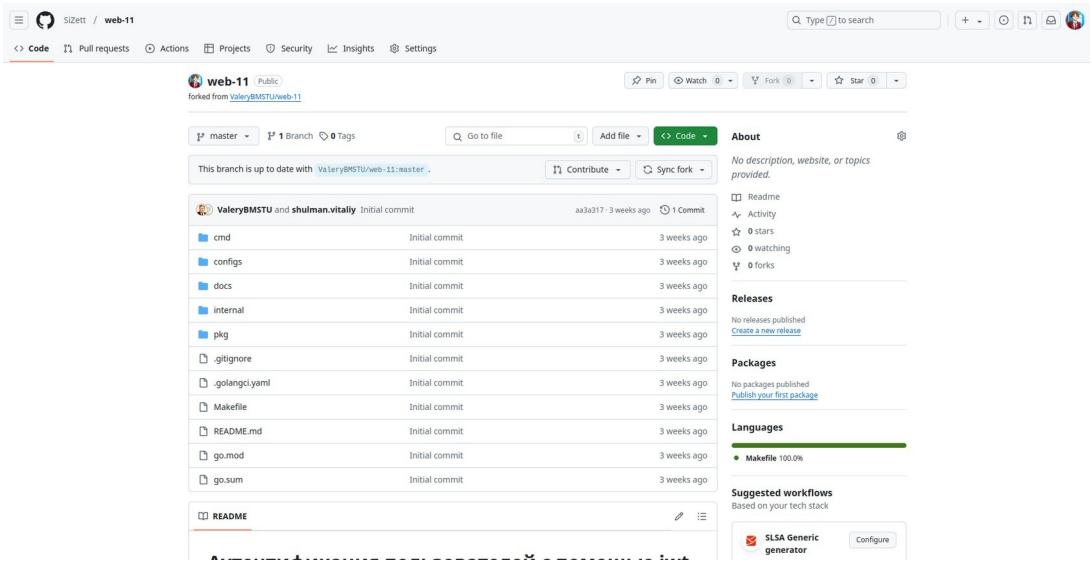

Рисунок 1

Код задания auth:

```
package main

import (
"net/http"
"time"

"github.com/golang-jwt/jwt/v5"
echojwt "github.com/labstack/echo-jwt/v4"
"github.com/labstack/echo/v4"
"github.com/labstack/echo/v4/middleware"
)

// jwtCustomClaims are custom claims extending default ones.
// See https://github.com/golang-jwt/jwt for more examples
type jwtCustomClaims struct {
Name string `json:"name"`
Admin bool `json:"admin"`
jwt.RegisteredClaims
}
```

```
func login(c echo.Context) error {
username := c.FormValue("username")
password := c.FormValue("password")

// Throws unauthorized error
if username != "admin" || password != "admin" {
return echo.ErrUnauthorized
}

// Set custom claims
claims := &jwtCustomClaims{
"admin",
true,
jwt.RegisteredClaims{
ExpiresAt: jwt.NewNumericDate(time.Now().Add(time.Hour * 72)),
},
}

// Create token with claims
token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)

// Generate encoded token and send it as response.
t, err := token.SignedString([]byte("secret"))
if err != nil {
return err
}

return c.JSON(http.StatusOK, echo.Map{
"token": t,
})
}

func accessible(c echo.Context) error {
return c.String(http.StatusOK, "Accessible")
}

func restricted(c echo.Context) error {
user := c.Get("user").(*jwt.Token)
claims := user.Claims.(*jwtCustomClaims)
name := claims.Name
return c.String(http.StatusOK, "Welcome "+name+"!")
}

func main() {
e := echo.New()

// Middleware
e.Use(middleware.Logger())
e.Use(middleware.Recover())

// Login route
```

```
e.POST("/login", login)

// Unauthenticated route
e.GET("/", accessible)

// Restricted group
r := e.Group("/restricted")

// Configure middleware with the custom claims type
config := echojwt.Config{
NewClaimsFunc: func(c echo.Context) jwt.Claims {
return new(jwtCustomClaims)
},
SigningKey: []byte("secret"),
}
r.Use(echojwt.WithConfig(config))
r.GET("", restricted)

e.Logger.Fatal(e.Start(":1323"))
}
```

Код задания hello:

```
package main

import (
"flag"
"log"

"github.com/ValeryBMSTU/web-11/internal/hello/api"
"github.com/ValeryBMSTU/web-11/internal/hello/config"
"github.com/ValeryBMSTU/web-11/internal/hello/provider"
"github.com/ValeryBMSTU/web-11/internal/hello/usecase"
_ "github.com/lib/pq"
)

func main() {
// Считываем аргументы командной строки
configPath := flag.String("config-path", "/home/sizet/web-11/configs/hello_example.yaml", "путь к
файлу конфигурации")
flag.Parse()

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

prv := provider.NewProvider(cfg.DB.Host, cfg.DB.Port, cfg.DB.User, cfg.DB.Password,
cfg.DB.DBname)
use := usecase.NewUsecase(cfg.Usecase.DefaultMessage, prv)
srv := api.NewServer(cfg.IP, cfg.Port, cfg.API.MaxMessageSize, use)

srv.Run()
```

```
}
```

Код задания query:

```
package main

import (
"flag"
"log"

"github.com/ValeryBMSTU/web-11/internal/query/api"
"github.com/ValeryBMSTU/web-11/internal/query/config"
"github.com/ValeryBMSTU/web-11/internal/query/provider"
"github.com/ValeryBMSTU/web-11/internal/query/usecase"
_ "github.com/lib/pq"
)

func main() {
// Считываем аргументы командной строки
configPath := flag.String("config-path", "/home/sizet/web-11/configs/hello_example.yaml", "путь к
файлу конфигурации")
flag.Parse()

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

prv := provider.NewProvider(cfg.DB.Host, cfg.DB.Port, cfg.DB.User, cfg.DB.Password,
cfg.DB.DBname)
use := usecase.NewUsecase(cfg.Usecase.DefaultMessageQuery, prv)
srv := api.NewServer(cfg.IP, cfg.Port, cfg.API.MaxMessageSize, use)

srv.Run()
}
```

Код задания count:

```
package main

import (
"flag"
"log"

"github.com/ValeryBMSTU/web-11/internal/count/api"
"github.com/ValeryBMSTU/web-11/internal/count/config"
"github.com/ValeryBMSTU/web-11/internal/count/provider"
"github.com/ValeryBMSTU/web-11/internal/count/usecase"

_ "github.com/lib/pq"
)

func main() {
// Считываем аргументы командной строки
```

```
configPath := flag.String("config-path", "/home/sizet/web-11/configs/hello_example.yaml", "путь к файлу конфигурации")
flag.Parse()

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

prv := provider.NewProvider(cfg.DB.Host, cfg.DB.Port, cfg.DB.User, cfg.DB.Password, cfg.DB.DBname)
use := usecase.NewUsecase(cfg.Usecase.DefaultMessageCount, prv)
srv := api.NewServer(cfg.IP, cfg.Port, cfg.API.MaxMessageSize, use)

srv.Run()
}
```

Заключение

При выполнении заданий лабораторной работы №11 мы получили первичные знания в области авторизации и аутентификации в контексте веб-приложений и выполнили задание основанное на результатах лабораторной работы №6.