



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

ОТЧЕТ

по лабораторной работе № 11

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

Дисциплина: Языки интернет-программирования

Студент

ИУ6-33Б

(Группа)

06.09.2024

(Подпись, дата)

Н.Н. Товарас

(И.О. Фамилия)

14.09.2024

Преподаватель

(Подпись, дата)

В.Д. Шульман

(И.О. Фамилия)

Москва, 2024

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

В рамках данной лабораторной работы предлагается ознакомиться с наиболее популярным способом аутентификации и авторизации пользователей в веб-приложениях — jwt-токеном

Ход работы:

- 1) Написание код для сервера auth.
- 2) Настройка jwt Middleware для серверов из 10 задания.

1)api.go

```
package api
```

```
import (  
    "fmt"
```

```
    "web-11/internal/auth/provider"
```

```
    "github.com/golang-jwt/jwt/v5"
```

```
    echojwt "github.com/labstack/echo-jwt/v4"
```

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

```
    "github.com/labstack/echo/v4/middleware"
```

```
)
```

```
type Server struct {
```

```
    minPassword int
```

```
    maxPassword int
```

```
    minUsername int
```

```
    maxUsername int
```

```
    server *echo.Echo
```

```
    r      *echo.Group
```

```
    address string
```

```
    uc Ucase
```

```
}
```

```
func NewServer(ip string, port int, minPassword, maxPassword, minUsername,  
maxUsername int, secret string, uc Ucase) *Server {
```

```
    api := Server{
```

```
        minPassword: minPassword,
```

```
        maxPassword: maxPassword,
```

```
        minUsername: minUsername,
```

```
        maxUsername: maxUsername,
```

```
        uc: uc,
```

```
    }
```

```
    api.server = echo.New()
```

```
    api.server.Use(middleware.Logger())
```

```
    api.server.Use(middleware.Recover())
```

```

api.server.POST("/register", api.Register)
api.server.POST("/login", api.Login)
api.r = api.server.Group("/restricted")

config := echojwt.Config{
    NewClaimsFunc: func(c echo.Context) jwt.Claims {
        return new(provider.JWTClaims)
    },
    SigningKey: []byte(secret),
}

api.r.Use(echojwt.WithConfig(config))
// api.r.Use(middleware.JWTWithConfig(middleware.JWTConfig{
//     SigningKey: []byte("your-secret-key"),
//     TokenLookup: "header:Authorization",
//     AuthScheme: "Bearer",
//     }))
api.r.GET("", api.Restricted)

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

return &api
}

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

handler.go
package api

import (
    "log"
    "net/http"
    "strconv"

    "web-11/internal/auth/provider"

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

func (srv *Server) Register(c echo.Context) error {
    username := c.FormValue("username")
    password := c.FormValue("password")

    if username == "" || password == "" {
        return echo.NewHTTPError(http.StatusUnauthorized, "Invalid credentials")
    }
}

```

```

    if len(username) < srv.minUsername || len(username) > srv.maxUsername {
        return echo.NewHTTPError(http.StatusUnauthorized, "Username should be
"+strconv.Itoa(srv.minUsername)+"-"+strconv.Itoa(srv.maxUsername)+" length")
    }

    if len(password) < srv.minPassword || len(password) > srv.maxPassword {
        return echo.NewHTTPError(http.StatusUnauthorized, "Password should be
"+strconv.Itoa(srv.minPassword)+"-"+strconv.Itoa(srv.maxPassword)+" length")
    }

    err := srv.uc.Register(username, password)
    if err != nil {
        log.Printf("Error creating account: %v", err) // добавлено
        return echo.NewHTTPError(http.StatusInternalServerError, "Couldn't create
account")
    }

    return c.JSON(http.StatusOK, "OK!")
}

func (srv *Server) Login(c echo.Context) error {
    username := c.FormValue("username")
    password := c.FormValue("password")

    if username == "" || password == "" {
        return echo.NewHTTPError(http.StatusUnauthorized, "Invalid credentials")
    }

    token, err := srv.uc.Authenticate(username, password)
    if err != nil {
        return echo.NewHTTPError(http.StatusInternalServerError, "Error generating
token")
    }

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

func (srv *Server) Restricted(c echo.Context) error {
    user := c.Get("user").(*jwt.Token)
    if user == nil {
        return c.JSON(http.StatusUnauthorized, map[string]string{"message": "Token is
missing or invalid"})
    }
    claims := user.Claims.(*provider.JWTClaims)
    log.Printf("Claims: %v", claims)
    username := claims.Username
    log.Printf("Username: %v", username)
    return c.JSON(http.StatusOK, map[string]string{"message": "Welcome " + username})
}

```

interface.go

```
package api

import "web-11/internal/auth/provider"

type Usecase interface {
    Authenticate(string, string) (string, error)
    ValidateJWT(string) (*provider.JWTClaims, error)
    Register(string, string) error
}
```

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"`
    JWT    jwt    `yaml:"jwt"`
}
```

```
type api struct {
    MinPasswordSize int `yaml:"min_password_size"`
    MaxPasswordSize int `yaml:"max_password_size"`
    MinUsernameSize int `yaml:"min_username_size"`
    MaxUsernameSize int `yaml:"max_username_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"`
}
```

```
type jwt struct {
    Secret string `yaml:"secret"`
}
```

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
}

```

```

jwt_provider.go
package provider

import (
    "github.com/golang-jwt/jwt/v5"
)

```

```

type JWTProvider struct {
    secretKey string
}

```

```

type JWTClaims struct {
    Username string `json:"username"`
    jwt.RegisteredClaims
}

```

```

func NewJWTProvider(secretKey string) *JWTProvider {
    return &JWTProvider{secretKey: secretKey}
}

```

```

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}  
}
```

```
sql.go  
package provider
```

```
import (  
    "database/sql"  
    "errors"  
    "log"  
)
```

```
func (p *Provider) CreateUser(username, password string) error {  
    _, err := p.conn.Exec("INSERT INTO users_11 (username, password) VALUES ($1,  
$2)", username, password)  
    if err != nil {  
        log.Printf("Error creating user: %v", err)  
    }  
    return err  
}
```

```
// func (p *Provider) SelectRandomHello() (string, error) {  
//     var msg string
```

```
//     // Получаем одно сообщение из таблицы hello, отсортированной в случайном  
//     порядке  
//     err := p.conn.QueryRow("SELECT message FROM hello ORDER BY RANDOM() LIMIT  
1").Scan(&msg)  
//     if err != nil {  
//         if errors.Is(err, sql.ErrNoRows) {  
//             return "", nil  
//         }  
//     }
```

```
//      return "", err
// }
```

```
// return msg, nil
// }
```

```
func (p *Provider) CheckUserByUsername(username string) (bool, error) {
    err := p.conn.QueryRow("SELECT (username) FROM users_11 WHERE username = $1",
username).Scan(&username)
    if err != nil {
        if errors.Is(err, sql.ErrNoRows) {
            return false, nil
        }
        log.Printf("Error checking user by username: %v", err)
        return false, err
    }

    return true, nil
}
```

```
func (p *Provider) CheckPassword(username, password string) (bool, error) {
    var password_db string
    err := p.conn.QueryRow("SELECT password FROM users_11 WHERE username = $1",
username).Scan(&password_db)
    if err != nil {
        log.Printf("Error checking password: %v", err)
        return false, err
    }
    if password == password_db {
        return true, nil
    }
    return false, nil
}
```

```
// func (p *Provider) CheckHelloExitByMsg(msg string) (bool, error) {
// // Получаем одно сообщение из таблицы hello
// err := p.conn.QueryRow("SELECT message FROM hello WHERE message = $1 LIMIT 1",
msg).Scan(&msg)
// if err != nil {
//     if errors.Is(err, sql.ErrNoRows) {
//         return false, nil
//     }
//     return false, err
// }
```

```
// return true, nil
// }
```

```
// func (p *Provider) InsertHello(msg string) error {
// _, err := p.conn.Exec("INSERT INTO hello (message) VALUES ($1)", msg)
// if err != nil {
```



```
//      return err
// }
```

```
// return nil
// }
```

```
token.go
package provider
```

```
import (
    "fmt"
    "time"
```

```
    "github.com/golang-jwt/jwt/v5"
)
```

```
func (j *JWTProvider) GenerateToken(username string) (string, error) {
    claims := JWTClaims{
        Username: username,
        RegisteredClaims: jwt.RegisteredClaims{
            ExpiresAt: jwt.NewNumericDate(time.Now().Add(1 * time.Hour)),
        },
    }
    token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
    return token.SignedString([]byte(j.secretKey))
}
```

```
func (j *JWTProvider) ValidateToken(tokenString string) (*JWTClaims, error) {
    token, err := jwt.ParseWithClaims(tokenString, &JWTClaims{}, func(token
    *jwt.Token) (interface{}, error) {
        return []byte(j.secretKey), nil
    })
    if err != nil || !token.Valid {
        return nil, err
    }
    claims, ok := token.Claims.(*JWTClaims)
    if !ok {
        return nil, fmt.Errorf("invalid claims")
    }
    return claims, nil
}
```

```
auth.go
package usecase
```

```
import (
    "fmt"
```

```
    "web-11/internal/auth/provider"
)
```

```

func (u *Usecase) Authenticate(username, password string) (string, error) {
    exist, err := u.p.CheckUserByUsername(username)
    if !exist {
        return "", fmt.Errorf("user not found")
    }
    if err != nil {
        return "", err
    }

    if correct, _ := u.p.CheckPassword(username, password); !correct {
        return "", fmt.Errorf("invalid password")
    }

    return u.jp.GenerateToken(username)
}

func (u *Usecase) ValidateJWT(token string) (*provider.JWTClaims, error) {
    return u.jp.ValidateToken(token)
}

func (u *Usecase) Register(login, password string) error {
    exist, err := u.p.CheckUserByUsername(login)
    if err != nil {
        return err
    }
    if exist {
        return fmt.Errorf("user already exists")
    }

    return u.p.CreateUser(login, password)
}

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

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

//     return msg, nil
// }

// func (u *Usecase) SetHelloMessage(msg string) error {
//     isExist, err := u.p.CheckHelloExitByMsg(msg)
//     if err != nil {
//         return err
//     }

```

```
// if isExist {
//     return nil
// }
```

```
// err = u.p.InsertHello(msg)
// if err != nil {
//     return err
// }
```

```
// return nil
// }
```

interface.go

```
package usecase
```

```
import "web-11/internal/auth/provider"
```

```
type Provider interface {
    CreateUser(string, string) error
    CheckUserByUsername(string) (bool, error)
    CheckPassword(string, string) (bool, error)
}
```

```
type JWTProvider interface {
    GenerateToken(string) (string, error)
    ValidateToken(string) (*provider.JWTClaims, error)
}
```

usecase.go

```
package usecase
```

```
type Usecase struct {
    defaultMsg string
}
```

```
    p Provider
    jp JWTProvider
}
```

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

2) Добавим в код handler серверов функции для jwt валидации:

```
func validateToken(token string) (bool, error) {
    client := &http.Client{}
    req, _ := http.NewRequest("GET", "http://localhost:8885/restricted", nil)
    req.Header.Set("Authorization", "Bearer "+token)
```

```

    resp, err := client.Do(req)
    if err != nil || resp.StatusCode != http.StatusOK {
        return false, err
    }

    return true, nil
}

func (srv *Server) jwtAuthMiddleware(next echo.HandlerFunc) echo.HandlerFunc {
    return func(c echo.Context) error {
        token := c.Request().Header.Get("Authorization")[7:]
        valid, err := validateToken(token)
        if err != nil || !valid {
            return c.JSON(http.StatusUnauthorized, map[string]string{"message":
"Unauthorized"})
        }
        return next(c)
    }
}

```

В файле api добавим в пути структуры server middleware функции:

```

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

    api.server = echo.New()
    api.server.GET("/count", api.GetCounter, api.jwtAuthMiddleware)
    api.server.POST("/count", api.PostCounter, api.jwtAuthMiddleware)

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

    return &api
}

```

это пример для count, другие аналогично.

Вывод: Я научился реализовывать jwt авторизацию на серверах.