

Міністерство освіти і науки України  
Національний технічний університет України „КПІ  
імені Ігоря Сікорського ”  
Факультет інформатики та обчислювальної техніки  
Кафедра інформатики і програмної інженерії

**ЗВІТ**  
лабораторної роботи №2  
з курсу «Основи WEB - технологій»

Тема: «Створення системи авторизації сайту за допомогою JWT  
токенів»

Перевірив:

Викл. Альбрехт Й.О.

Виконала:

ст. Мельникова  
Катерина гр. ІС-12

Київ 2024

## 1. Завдання 1.

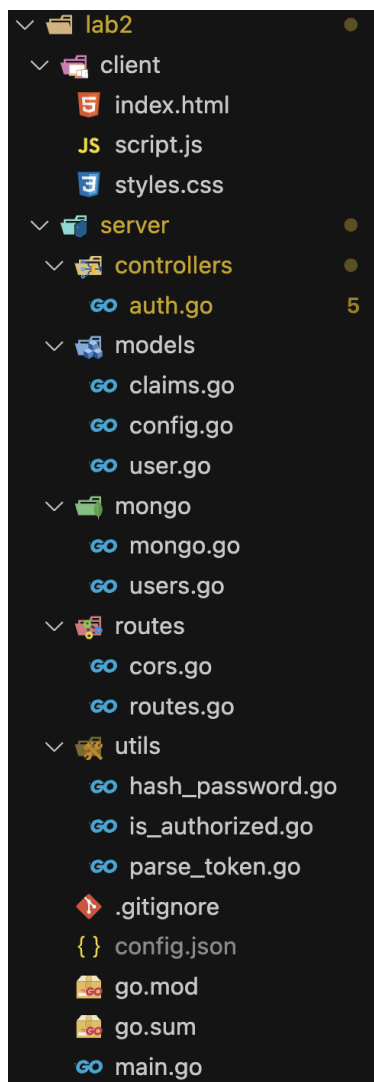
### Завдання.

1. При реєстрації та при оновленні користувача пароль зберігати в зашифрованому вигляді;
2. Створити запит `/users/login` на вхід
3. Застосувати авторизацію: в запиті відправити `authToken` в заголовок `Authorization`. Сервер отримує токен, верифікує його і авторизує користувача (або ні в іншому випадку).

Забезпечити використання ролей (`user` та `admin`) з відповідними діями.

### Хід роботи

Файлова структура проекту наведена на малюнку:



Повний код проєкту можна переглянути за посиланням:  
<https://github.com/katerynamelnykova/web-technologies-labs>

Файл auth.go:

```
package controllers

import (
    "encoding/json"
    "fmt"
    "net/http"
    "os"
    "time"

    "github.com/dgrijalva/jwt-go"
    "github.com/katerynamelnykova/web-technologies-labs/lab2/server/models"
    "github.com/katerynamelnykova/web-technologies-labs/lab2/server/mongo"
    "github.com/katerynamelnykova/web-technologies-labs/lab2/server/utils"
    "go.mongodb.org/mongo-driver/bson"
)

var jwtKey = []byte(os.Getenv("KEY"))

var mh *mongo.MongoHandler

func Connect() error {
    config, configErr := models.LoadConfiguration()
    if configErr != nil {
        return configErr
    }
    mongoDbConnection := config.Database

    var err error
    mh, err = mongo.NewHandler(mongoDbConnection)

    return err
}

func getAdminPassword() string {
    config, configErr := models.LoadConfiguration()
    if configErr != nil {
        return ""
    }
    AdminPass := config.AdminPassword

    return AdminPass
}
```

```

}

func Signup() http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        if err := Connect(); err != nil {
            http.Error(w, fmt.Sprintf(err), 500)
            return
        }

        var user models.User

        json.NewDecoder(r.Body).Decode(&user)
        if len(user.Password) < 8 {
            http.Error(w, fmt.Sprintf("Password must have at least 8 characters"), 403)
            return
        }

        existingUser := &models.User{}
        err := mh.GetOneUser(existingUser, bson.M{"email": user.Email})
        if err == nil {
            http.Error(w, fmt.Sprintf("Something went wrong with the signup process"), 500)
            return
        }

        if user.Password == getAdminPassword() {
            user.IsAdmin = true
        } else {
            user.IsAdmin = false
        }

        var errHash error
        user.Password, errHash = utils.GenerateHashPassword(user.Password)
        if errHash != nil {
            http.Error(w, fmt.Sprintf(errHash), 500)
            return
        }

        _, err = mh.AddOneUser(&user)
        if err != nil {
            http.Error(w, fmt.Sprintf(err), 500)
            return
        }

        w.WriteHeader(201)
    }
}

```

```

}

func Login() http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        if err := Connect(); err != nil {
            http.Error(w, fmt.Sprintf(err), 500)
            return
        }

        user := &models.User{}

        json.NewDecoder(r.Body).Decode(&user)

        existingUser := &models.User{}
        err := mh.GetOneUser(existingUser, bson.M{"email": user.Email})
        if err != nil {
            http.Error(w, fmt.Sprintf("Not found"), 404)
            return
        }

        passwordErr := utils.CompareHashPassword(user.Password, existingUser.Password)
        if !passwordErr {
            http.Error(w, fmt.Sprintf("Invalid password"), 400)
            return
        }

        expirationTime := time.Now().Add(48 * time.Hour)

        claims := &models.Claims{
            StandardClaims: jwt.StandardClaims{
                Subject:    existingUser.Email,
                ExpiresAt: expirationTime.Unix(),
            },
        }

        token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)

        tokenString, err := token.SignedString(jwtKey)

        if err != nil {
            http.Error(w, fmt.Sprintf(err), 500)
            return
        }

        http.SetCookie(w, &http.Cookie{

```

```

        Name:      "token",
        Value:      tokenString,
        Expires:    expirationTime,
        Path:       "/",
        Domain:     "localhost:5500",
        HttpOnly:   false,
        SameSite:   http.SameSiteNoneMode,
        Secure:     true,
    })

    response := map[string]any{
        "token": tokenString,
        "admin": existingUser.IsAdmin,
    }

    w.WriteHeader(200)
    json.NewEncoder(w).Encode(response)
}

func Logout() http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        http.SetCookie(w, &http.Cookie{
            Name:      "token",
            Value:      "",
            Expires:    time.Now().Add(-50 * time.Hour),
            Path:       "/",
            Domain:     "localhost:5500",
            HttpOnly:   false,
            SameSite:   http.SameSiteNoneMode,
            Secure:     true,
        })

        w.WriteHeader(200)
    }
}

func validateJWT(tokenString string) (*models.Claims, error) {
    claims := &models.Claims{}

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

```

```

    if err != nil {
        if err == jwt.ErrSignatureInvalid {
            return nil, fmt.Errorf("invalid token signature")
        }
        return nil, fmt.Errorf("error parsing token: %v", err)
    }

    if !token.Valid {
        return nil, fmt.Errorf("token is invalid")
    }

    return claims, nil
}

func AuthHandler() http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        tokenString := r.Header.Get("Authorization")
        if tokenString == "" {
            http.Error(w, "Token not found", http.StatusUnauthorized)
            return
        }

        claims, err := validateJWT(tokenString)
        if err != nil {
            http.Error(w, err.Error(), http.StatusUnauthorized)
            return
        }

        w.WriteHeader(200)
        json.NewEncoder(w).Encode(claims)
    }
}

```

Отримані результати:

### Auth System

Sign Up

Email:

Password:

Sign Up

Login

Email:

john.doe2@gmail.com

Password:

\*\*\*\*\*

Login

Logged in as: john.doe2@gmail.com

Logout

### Auth System

Sign Up

Email:

Password:

Sign Up

Login

Email:

Password:

Login

Application

Filter

http://127.0.0.1:5500

Origin http://127.0.0.1:5500

Storage	Key	Value
Local storage	admin	false
http://127.0.0.1:5500	email	john.doe2@gmail.com
Session storage	token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV...
IndexedDB		
Cookies		
Private storage		
Interest groups		
Shared storage		
Cache storage		
Storage buckets		

1 false

Background sync

Backforward cache

Background sync		
Background sync		
Bounce timing		
Notification		
Payment		
Periodic background sync		
Speculation rules		

Logged in as: kate.meln2803@gmail.com

Logout

Admin Button

### Auth System

Sign Up

Email:

Password:

Sign Up


Login

Email:

Password:

Login



[Documents](#) [Aggregations](#) [Schema](#) [Explain Plan](#) [Indexes](#) [Validation](#)[Filter](#)  Type a query: { field: 'value' }

Reset

Find

[More Options](#)[ADD DATA](#)[EXPORT COLLECTION](#)

1 - 3 of 3



```
_id: ObjectId('66e1a6f939e467241afcc90e')
email: "kate.meln2803@gmail.com"
password: "$2a$14$mRYpAkQ2x8GKR4RGe2JXVeMCqCRLWDzT3CaJ7z0UJcsg0hennx9rK"
admin: true
```

```
_id: ObjectId('66e1a71939e467241afcc912')
email: "john.doe2@gmail.com"
password: "$2a$14$4uHH24doSuFyP0omzpyca.6Jv8Xv.Dpo5Lmx.L0drim.f4VvdsXjK"
admin: false
```

```
_id: ObjectId('66e1ffaa39e467241afcc916')
email: "john.doe7@gmail.com"
password: "$2a$14$n6hZ8p9oTmjzH04cw6V.707WkrNbqxQTWmPuRREoLMtDtVVY4YBn0"
admin: false
```

## Висновок

Під час виконання даної лабораторної роботи я навчилася працювати з JWT-токенами, розробила бекенд та фронтенд а також застосувала авторизацію для отримання даних користувача (електронної пошти).