

Лабораторна робота №7
REST web services

Зміст

1. Лабораторна робота 7	2
1.1 Постановка задачі	2
1.2 Вимоги до роботи	2
1.3 Вимоги до оформлення звіту	2
1.4 Варіанти завдань	2
2. Рекомендації по виконанню роботи	5
2.1 Попередні налаштування.....	5
2.2 Предметна область.....	5
2.3 Розробка додатку	5
2.4 Реєстрація.....	7
2.5 Додавання JWT	10
2.6 Використання JWT для авторизації.....	13
2.7 Малюємо залишок сови.....	15

1. Лабораторна робота 7

1.1 Постановка задачі

Розробіть...

1.2 Вимоги до роботи

Має бути...

1.3 Вимоги до оформлення звіту

Звіт повинен містити:

- титульну сторінку
- постановку задачі
- вихідний код програми (включаючи web.xml, таблиці css, і т.п.)
- опис програми (опис класів, методів, полів)

1.4 Варіанти завдань

Варіант 1

Предметна область Карта Світу

Об'єкти Автори, Книги

Карта світу містить множину *країн*. Для кожної *країни* визначена множина *міст*.

Необхідні операції Отримання повного списку міст із зазначенням назви країни

Варіант 2

Предметна область Бібліотека

Об'єкти Автори, Книги

Книги в бібліотеці згруповані по *авторам*. У кожного *автора* є множина *книг*.

Необхідні операції Отримання повного списку книг із зазначенням ПІБ автора

Варіант 3

Предметна область Відділ кадрів

Об'єкти Підрозділи, Співробітники

Існує множина *підрозділів* підприємства. У кожному *підрозділі* працює множина *співробітників*.

Необхідні операції Отримання списку співробітників із зазначенням назви підрозділу

Варіант 4

Предметна область Навчальний відділ

Об'єкти Групи, Студенти

Існує множина навчальних *груп*. Кожна група включає в себе множину *студентів*.

Необхідні операції Отримання повного списку студентів із зазначенням назви групи

Варіант 5

Предметна область Автосалон

Об'єкти Виробники автомобілів, Марки

Марки автомобілів згруповані по виробникам. У кожного *виробника* є множина *марок*.

Необхідні операції Отримання повного списку марок з назвою виробника

Варіант 6

Предметна область Агентство новин

Об'єкти Категорії новин, Новини

Новини згруповані по *категоріям*. У кожній *категорії* є множина *новин*.

Необхідні операції Отримання повного списку новин із зазначенням категорії

Варіант 7

Предметна область Продуктовий магазин

Об'єкти Категорія продукту, Продукт

Продукти в магазині згруповані за *категоріями*. Для кожної *категорії* визначено множину *продуктів*.

Необхідні операції Отримання списку продуктів із зазначенням категорії

Варіант 8

Предметна область Футбол

Об'єкти Команди, Гравці

Існує множина футбольних *команд*. Для кожної *команди* визначено множину *гравців*.

Необхідні операції Отримання повного списку гравців із зазначенням назви команди

Варіант 9

Предметна область Музичний магазин

Об'єкти Виконавці, Альбоми

У музичному магазині *альбоми* згруповані за виконавцями. Для кожного *виконавця* задано множину *альбомів*.

Необхідні операції Отримання повного списку альбомів із зазначенням виконавця

Варіант 10

Предметна область Аеропорт

Об'єкти Авіакомпанії, Рейси

Існує множина *авіакомпаній*. Для кожної *авіакомпанії* визначені її *рейси*.

Отримання повного списку рейсів із зазначенням назви авіакомпанії

Варіант 11

Предметна область Файлова система

Об'єкти Папки, Файли

Існує множина *папок* (незалежних один від одного). Для кожної *папки* визначено множину *файлів*.

Необхідні операції Отримання списку файлів із зазначенням папки

Варіант 12

Предметна область Розклад занять

Об'єкти Дні тижня, Заняття

Примітка Існує множина *днів*. Для кожного *дня* визначено перелік *занять*.

Необхідні операції Отримання повного списку занять із зазначенням дня

Варіант 13

Предметна область Записна книжка

Об'єкти Календарні дні, Заходи

Примітка Існує множина *днів*. Для кожного *дня* визначено перелік *заходів*.

Необхідні операції Отримання повного списку заходів із зазначенням дня

Варіант 14

Предметна область	Відео-магазин
Об'єкти	Жанри, Фільми

Існує множина *жарів*. Для кожного *жанру* визначений перелік *фільмів*.

Необхідні операції	Отримання списку фільмів із зазначенням жанру
--------------------	---

Варіант 15

Предметна область	Залізна дорога
Об'єкти	Дороги, Станції

Існує множина *залізниць*. У відомстві кожної дороги знаходиться множина *станцій*.

Необхідні операції	Отримання повного списку станцій із зазначенням назви дороги
--------------------	--

Варіант 16

Предметна область	Склад
Об'єкти	Секції, Товари

Товари на складі згруповані по *секціях*. Для кожної *секції* задано множину *товарів*.

Необхідні операції	Отримання списку товарів із зазначенням секції
--------------------	--

Варіант 17

Предметна область	Кафедра університету
Об'єкти	Викладачі, Дисципліни

На кафедрі існує множина *викладачів*. Для кожного *викладача* задано множину *дисциплін*.

Необхідні операції	Отримання списку дисциплін із зазначенням ПІБ викладача
--------------------	---

Варіант 18

Предметна область	Програмне забезпечення
Об'єкти	Виробники, Програмні продукти

Програмні *продукти* згруповані по *виробникам*. Для кожного *виробника* задано множину *продуктів*.

Необхідні операції	Отримання списку продуктів із зазначенням виробника
--------------------	---

Варіант 19

Предметна область	Геометрія
Об'єкти	Багатокутники, Вершини

Існує множина *багатокутників*. Кожен *багатокутник* складається з довільної кількості вершин.

Отримання повного списку багатокутників з зазначенням всіх вершин

Варіант 20


Предметна область	Схема метро
Об'єкти	Лінії, Станції

Існує множина *ліній* метрополітену. Кожна *лінія* складається з послідовності *станцій*.

Необхідні операції	Отримання списку станцій із зазначенням лінії
--------------------	---

2. Рекомендації по виконанню роботи

2.1 Попередні налаштування

Для розробки додатку можна використовувати будь-який текстовий редактор (хоч ) , потрібно мати встановлений go, а також нам знадобиться бібліотека gorilla/mux, яку можна встановити наступною командою:

```
go get -u github.com/gorilla/mux
```

2.2 Предметна область

Реалізуємо додаток, який за запитом користувача буде видавати його улюблений тортик :3. Також, звичайно, нам потрібно буде реалізувати реєстрацію користувача, оновлення його даних та їх видалення (от і CRUD виходить).

2.3 Розробка додатку

Почнемо з простого додатку, який буде видавати звичайний тортик на будь-які запити до нього.

Файл `main.go`:

```
package main

import (
    "context"
    "github.com/gorilla/mux"
    "log"
    "net/http"
    "os"
    "os/signal"
    "time"
)

func getCakeHandler(w http.ResponseWriter, r *http.Request) {
    w.WriteHeader(http.StatusOK)
    _, err := w.Write([]byte("cake"))
    if err != nil {
        return
    }
}

func main() {
    r := mux.NewRouter()
    r.HandleFunc("/cake", getCakeHandler).Methods(http.MethodGet)
    srv := http.Server{
        Addr:    ":8080",
        Handler: r,
    }

    interrupt := make(chan os.Signal, 1)
    signal.Notify(interrupt, os.Interrupt)
    go func() {
```

```

    <-interrupt
    ctx, cancel := context.WithTimeout(context.Background(),
        5*time.Second)
    defer cancel()
    err := srv.Shutdown(ctx)
    if err != nil {
        return
    }
}()

log.Println("Server started, hit Ctrl+C to stop")
err := srv.ListenAndServe()
if err != nil {
    log.Println("Server exited with error:", err)
}
log.Println("Good bye :)")
}

```

Спочатку ми створили функцію, яка модифікує об'єкт `http.ResponseWriter`, який і записує відповідь на `http` запит.

```

func getCakeHandler(w http.ResponseWriter, r *http.Request) {
    w.WriteHeader(http.StatusOK)
    _, err := w.Write([]byte("cake"))
    if err != nil {
        return
    }
}

```

Далі створюємо маршрутизатор, який буде відповідальний за прийняття `http` запитів та «перенаправлення» даних у них по відповідних «маршрутах» (функціях).

```

r := mux.NewRouter()
r.HandleFunc("/cake", getCakeHandler).Methods(http.MethodGet)
srv := http.Server{
    Addr:      ":8080",
    Handler: r,
}

```

Аби перевірити, чи дійсно при зверненні до сервера нам повертається тортик, скористаємося `cli`-засобом `curl` (<https://curl.se/>). Якщо ж у вас виникають складнощі з його встановленням (привіт, Windows!) або ви просто не любите працювати з консолі, раджу використати `Postman` (<https://www.postman.com/>).

Я ж буду використовувати мінімалістичний `curl`:

```
curl localhost:8080/cake
```

У відповідь ви маєте отримати свій тортик ;3.

2.4 Реєстрація

Наступним кроком додамо реєстрацію (для простоти будемо зберігати дані в оперативній пам'яті).

Створіть файл `users.go`, де будуть розміщені типи даних та обробники, що стосуються користувачів:

```
package main

import (
    "crypto/md5"
    "encoding/json"
    "errors"
    "net/http"
    "net/mail"
)

type User struct {
    Email            string
    PasswordDigest   string
    FavoriteCake     string
}

type UserRepository interface {
    Add(string, User) error
    Get(string) (User, error)
    Update(string, User) error
    Delete(string) (User, error)
}

type UserService struct {
    repository UserRepository
}

type UserRegisterParams struct {
    Email            string `json:"email"`
    Password         string `json:"password"`
    FavoriteCake     string `json:"favorite_cake"`
}

func validateRegisterParams(p *UserRegisterParams) error {
    // 1. Email is valid
    if _, err := mail.ParseAddress(p.Email); err != nil {
        return errors.New("must provide an email")
    }

    // 2. Password at least 8 symbols
    if len(p.Password) < 8 {
        return errors.New("password must be at least 8 symbols")
    }

    // 3. Favorite cake not empty
    if len(p.FavoriteCake) < 1 {
        return errors.New("favourite cake can't be empty")
    }

    // 4. Favorite cake only alphabetic
    for _, charVariable := range p.FavoriteCake {
        if (charVariable < 'a' || charVariable > 'z') && (charVariable < 'A' ||
charVariable > 'Z') {
```

```

        return errors.New("favourite cake must contain only alphabetic
characters")
    }
}

return nil
}

func (u *UserService) Register(w http.ResponseWriter, r *http.Request) {
    params := &UserRegisterParams{}
    err := json.NewDecoder(r.Body).Decode(params)
    if err != nil {
        handleError(errors.New("could not read params"), w)
        return
    }
    if err := validateRegisterParams(params); err != nil {
        handleError(err, w)
        return
    }
    passwordDigest := md5.New().Sum([]byte(params.Password))
    newUser := User{
        Email:           params.Email,
        PasswordDigest:  string(passwordDigest),
        FavoriteCake:    params.FavoriteCake,
    }
    err = u.repository.Add(params.Email, newUser)
    if err != nil {
        handleError(err, w)
        return
    }
    w.WriteHeader(http.StatusCreated)
    _, wrErr := w.Write([]byte("registered"))
    if wrErr != nil {
        return
    }
}

func handleError(err error, w http.ResponseWriter) {
    w.WriteHeader(http.StatusUnprocessableEntity)
    _, wrErr := w.Write([]byte(err.Error()))
    if wrErr != nil {
        return
    }
}
}

```

У файлі `user_repository.go` розмістимо сховище користувачів, а також його CRUD операції:

```

package main

import (
    "errors"
    "sync"
)

type InMemoryUserStorage struct {
    lock      sync.RWMutex
    storage map[string]User
}

```



```

}

func NewInMemoryUserStorage() *InMemoryUserStorage {
    return &InMemoryUserStorage{
        lock:    sync.RWMutex{},
        storage:  make(map[string]User),
    }
}

func (userStorage *InMemoryUserStorage) Add(email string, user User) error {
    userStorage.lock.Lock()
    defer userStorage.lock.Unlock()

    _, err := userStorage.storage[email]

    if err {
        return errors.New("user already exists")
    } else {
        userStorage.storage[email] = user
    }

    return nil
}

func (userStorage *InMemoryUserStorage) Get(email string) (User, error) {
    userStorage.lock.Lock()
    defer userStorage.lock.Unlock()

    user, err := userStorage.storage[email]

    if err {
        return User{}, errors.New("user doesn't exist")
    }

    return user, nil
}

func (userStorage *InMemoryUserStorage) Update(email string, newUser User) error {
    userStorage.lock.Lock()
    defer userStorage.lock.Unlock()

    _, err := userStorage.storage[email]

    if err {
        return errors.New("user doesn't exist")
    }

    userStorage.storage[email] = newUser

    return nil
}

func (userStorage *InMemoryUserStorage) Delete(email string) (User, error) {
    userStorage.lock.Lock()
    defer userStorage.lock.Unlock()

    user, err := userStorage.storage[email]

    if err {
        return User{}, errors.New("user doesn't exist")
    }

```

```
}

delete(userStorage.storage, email)

return user, nil
}
```

Додамо до нашого маршрутизатора шлях для реєстрації користувача:

```
func main() {
    r := mux.NewRouter()

    userService := UserService{
        repository: NewInMemoryUserStorage(),
    }

    r.HandleFunc("/cake", getCakeHandler).Methods(http.MethodGet)
    r.HandleFunc("/user/register", userService.Register).Methods(http.MethodPost)

    srv := http.Server{
        Addr:    ":8080",
        Handler: r,
    }
    ...
}
```

Спробуємо зареєструвати користувача:

```
> curl -X POST localhost:8080/user/register --data
'{"email":"test@gmail.com", "favorite_cake":"cheesecake",
"password":"hello1234"}'
```

Як підтвердження коректної роботи додатку, маємо в консолі побачити

```
: registered
```

2.5 Додавання JWT

Фактично, JWT – це закодований рядок, що містить певний JSON об’єкт та «підпис». Таким чином, ми можемо довіряти даним в JSON об’єкті, якщо «підпис» коректний.

Отже, для роботи з JWT нам знадобиться:

1. Створити JWT та підписати його певним ключем.
2. Прочитати JWT та підтвердити підпис наданим ключем.

Для роботи з JWT будемо використовувати пакет Rango:

```
go get -u github.com/openware/rango
```

Створимо `jwt.go` файл, у який додамо од для створення та читання JWT:

```
package main

import (
    "crypto/md5"
    "encoding/json"
    "errors"
    "github.com/openware/rango/pkg/auth"
    "net/http"
)

type JWTService struct {
    keys *auth.KeyStore
}

func NewJWTService(privKeyPath, pubKeyPath string) (*JWTService, error) {
    keys, err := auth.LoadOrGenerateKeys(privKeyPath, pubKeyPath)
    if err != nil {
        return nil, err
    }
    return &JWTService{keys: keys}, nil
}

func (j *JWTService) GenerateJWT(u User) (string, error) {
    return auth.ForgeToken("empty", u.Email, "empty", 0, j.keys.PrivateKey, nil)
}

func (j *JWTService) ParseJWT(jwt string) (auth.Auth, error) {
    return auth.ParseAndValidate(jwt, j.keys.PublicKey)
}
```

Для створення JWT, який ми будемо повертати користувачу, нам потрібні його email та пароль. Зазвичай JWT додають до заголовка http запиту: «Authorization: Bearer <jwt>», тому будемо зчитувати його звідти.

Той же `jwt.go` файл:

```
...

type JWTParams struct {
    Email    string `json:"email"`
    Password string `json:"password"`
}

func (u *UserService) JWT(w http.ResponseWriter, r *http.Request, jwtService
*JWTService) {
    params := &JWTParams{}
    decErr := json.NewDecoder(r.Body).Decode(params)

    if decErr != nil {
        handleError(errors.New("could not read params"), w)
        return
    }

    passwordDigest := md5.New().Sum([]byte(params.Password))
```

```

user, err := u.repository.Get(params.Email)

if err != nil {
    handleError(err, w)
    return
}

if string(passwordDigest) != user.PasswordDigest {
    handleError(errors.New("invalid login params"), w)
    return
}

token, jwtErr := jwtService.GenerateJWT(user)

if jwtErr != nil {
    handleError(jwtErr, w)
    return
}

w.WriteHeader(http.StatusOK)
_, wrErr := w.Write([]byte(token))
if wrErr != nil {
    return
}
}

```

Тепер у файлі `main.go` створимо новий шлях для маршрутизатора:

```

...

func wrapJwt(jwt *JWTService, f func(http.ResponseWriter, *http.Request,
*JWTService)) http.HandlerFunc {
    return func(rw http.ResponseWriter, r *http.Request) {
        f(rw, r, jwt)
    }
}

...

userService := UserService{
    repository: NewInMemoryUserStorage(),
}

jwtService, jwtErr := NewJWTService("pubkey.rsa", "privkey.rsa")
if jwtErr != nil {
    panic(jwtErr)
}

r.HandleFunc("/cake", getCakeHandler).Methods(http.MethodGet)
r.HandleFunc("/user/register", userService.Register).Methods(http.MethodPost)
r.HandleFunc("/user/jwt", wrapJwt(jwtService,
userService.JWT)).Methods(http.MethodPost)

...

```

Запустимо наш додаток та зареєструємо користувача:

```
: registered
```

```
> curl -X POST localhost:8080/user/jwt --data '{"email":"test@gmail.com","password":"wrongpass"}'
```

Отримаємо коректний JWT:

```
> curl -X POST localhost:8080/user/jwt --data '{"email":"test@gmail.com","password":"hello1234"}'
```

```
:eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOi0lsicGVhdGlviIwiYmFyb25nIl0sImVtYWlsIjoiaGFja2FkZW15IiwiaXhwIjoxNjMwNDA0MDY1LCJpYXQiOi0jE2MzA0MDA0NjUsImIzcyI6ImJhcm9uZyIsImp0aSI6IjE2MzA0MDA0NjUiLCJzZXZlbCI6MCwicmVmZXJyYWxfawQiOm51bGwsInJvbGUoiOiJlbXB0eSIsInN0YXRLIjoiiYWNoaXZlIiwic3ViIjoici2Vzc2lubiIsInVpZCI6ImVtcHR5In0.1a9jBmHANEetSuWl8NhDI5js62r2roiZRlR5-I2mPPjCLjS9eobziBvguCKq9wggWfBGpGLk-5ZwpdSPjG4EEWTJYIBgcqI1BXdZkXNJWPa5dLIo1-Xmw55hVmHtZPnWU05GIZye8SurU0aGghNiYN2GHkBV2hMNY-KTHVaBg7w--A0zpcgIhH8PQprIsk1zL1ekstCXo-dUDMhjBGcYTLbhekJBQBcF2ahGWxYuw5vF18QSshW51dfS8htL8IgTrkK3bjLvqGbRCuPcyL5Uiun3IVHcTQVNYH3vra0R37y70r-DS1hiVoYWVOoL66SBsxF0BFUVAC5XNLk5MF1pw
```

Для використання JWT і авторизації користувачів, створимо проміжну функцію і приєднаємо її до всіх «захищених» (ті, що потребують авторизації) шляхів маршрутизатора. Всі захищені шляхи прийматимуть користувача як аргумент.

jwt.go файл:

```

...

type ProtectedHandler func(rw http.ResponseWriter, r *http.Request, u User)

func (j *JWTService) jwtAuth(users UserRepository, h ProtectedHandler)
http.HandlerFunc {
    return func(rw http.ResponseWriter, r *http.Request) {
        authHeader := r.Header.Get("Authorization")
        token := strings.TrimPrefix(authHeader, "Bearer ")
        jwtAuth, err := j.ParseJWT(token)
        if err != nil {
            rw.WriteHeader(401)
            _, err := rw.Write([]byte("unauthorized"))
            if err != nil {
                return
            }
            return
        }
        user, err := users.Get(jwtAuth.Email)
        if err != nil {
            rw.WriteHeader(401)
            _, err := rw.Write([]byte("unauthorized"))
            if err != nil {
                return
            }
            return
        }
        h(rw, r, user)
    }
}

```

Тепер час повертати улюблений тортик саме авторизованого користувача та додати авторизацію до інших шляхів маршрутизатора:

main.go:

```

...

func getCakeHandler(w http.ResponseWriter, r *http.Request, u User) {
    w.WriteHeader(http.StatusOK)
    _, err := w.Write([]byte(u.FavoriteCake))
    if err != nil {
        return
    }
}

func main() {
    r := mux.NewRouter()

    users := NewInMemoryUserStorage()
    userService := UserService{repository: users}

    jwtService, jwtErr := NewJWTService("pubkey.rsa", "privkey.rsa")
    if jwtErr != nil {
        panic(jwtErr)
    }
}

```

```
r.HandleFunc("/cake", jwtService.jwtAuth(users,
getCakeHandler)).Methods(http.MethodGet)
r.HandleFunc("/user/register", userService.Register).Methods(http.MethodPost)
r.HandleFunc("/user/jwt", wrapJwt(jwtService,
userService.JWT)).Methods(http.MethodPost)
...
```

Тепер випробуємо роботу додатку:

```
> curl -X GET localhost:8080/cake
```

: unauthorized

Реєстрація користувача:

```
> curl -X POST localhost:8080/user/register --data
'{"email":"test@gmail.com", "password":"hello1234", "favorite_cake":"
cheesecake"}'
```

: registered

Експортування JWT цього користувача в системну змінну:

```
> export JWT=$(curl -X POST localhost:8080/user/jwt --data
'{"email":"test@gmail.com","password":"hello1234"}')
```

Використання JWT, аби отримати тортик:

```
> curl -X GET localhost:8080/cake -H "Authorization: Bearer ${JWT}"
```

: cheesecake

2.7 [Малюємо залишок сови](#)

Весь реалізований CRUD виглядає наступним чином (зміни додані до 2-х файлів).

users.go:

```
package main

import (
    "crypto/md5"
    "encoding/json"
    "errors"
    "net/http"
    "net/mail"
```

```

)

type User struct {
    Email            string
    PasswordDigest   string
    FavoriteCake     string
}

type UserRepository interface {
    Add(string, User) error
    Get(string) (User, error)
    Update(string, User) error
    Delete(string) (User, error)
}

type UserService struct {
    repository UserRepository
}

type UserRegisterParams struct {
    Email            string `json:"email"`
    Password         string `json:"password"`
    FavoriteCake     string `json:"favorite_cake"`
}

func validateEmail(email string) error {
    if _, err := mail.ParseAddress(email); err != nil {
        return errors.New("must provide an email")
    }
    return nil
}

func validatePassword(password string) error {
    // 2. Password at least 8 symbols
    if len(password) < 8 {
        return errors.New("password must be at least 8 symbols")
    }
    return nil
}

func validateFavoriteCake(cake string) error {
    // 3. Favorite cake not empty
    if len(cake) < 1 {
        return errors.New("favourite cake can't be empty")
    }
    // 4. Favorite cake only alphabetic
    for _, charVariable := range cake {
        if (charVariable < 'a' || charVariable > 'z') && (charVariable < 'A' ||
charVariable > 'Z') {
            return errors.New("favourite cake must contain only alphabetic
characters")
        }
    }
    return nil
}

func validateRegisterParams(p *UserRegisterParams) error {
    err := validateFavoriteCake(p.FavoriteCake)
    if err != nil {
        return err
    }

    err = validatePassword(p.Password)

```



```

    if err != nil {
        return err
    }

    err = validateEmail(p.Email)
    return err
}

func (u *UserService) Register(w http.ResponseWriter, r *http.Request) {
    params := &UserRegisterParams{}
    err := json.NewDecoder(r.Body).Decode(params)
    if err != nil {
        handleUnprocError(errors.New("could not read params"), w)
        return
    }

    if err := validateRegisterParams(params); err != nil {
        handleUnprocError(err, w)
        return
    }

    passwordDigest := md5.New().Sum([]byte(params.Password))
    newUser := User{
        Email:           params.Email,
        PasswordDigest:  string(passwordDigest),
        FavoriteCake:    params.FavoriteCake,
    }

    err = u.repository.Add(params.Email, newUser)
    if err != nil {
        handleUnprocError(err, w)
        return
    }
    w.WriteHeader(http.StatusCreated)
    _, _ = w.Write([]byte("registered"))
}

func getCakeHandler(w http.ResponseWriter, _ *http.Request, u User, _
UserRepository) {
    w.WriteHeader(http.StatusOK)
    _, _ = w.Write([]byte "[" + u.Email + "], your favourite cake is " +
u.FavoriteCake))
}

func updateCakeHandler(w http.ResponseWriter, r *http.Request, u User, users
UserRepository) {
    params := &UserRegisterParams{}
    err := json.NewDecoder(r.Body).Decode(params)
    if err != nil {
        handleUnprocError(errors.New("could not read params"), w)
        return
    }

    if err := validateFavoriteCake(params.FavoriteCake); err != nil {
        handleUnprocError(err, w)
        return
    }

    passwordDigest := string(md5.New().Sum([]byte(params.Password)))

    if params.Email != u.Email || passwordDigest != u.PasswordDigest {

```

```

        handleUnauthError(errors.New("unauthorized"), w)
        return
    }

    updatedUser := User{
        Email:           params.Email,
        PasswordDigest: passwordDigest,
        FavoriteCake:     params.FavoriteCake,
    }

    err = users.Update(params.Email, updatedUser)
    if err != nil {
        handleUnprocError(err, w)
        return
    }
    w.WriteHeader(http.StatusOK)
    _, _ = w.Write([]byte("favorite cake updated"))
}

func updateEmailHandler(w http.ResponseWriter, r *http.Request, u User, users
UserRepository) {
    params := &UserRegisterParams{}
    err := json.NewDecoder(r.Body).Decode(params)
    if err != nil {
        handleUnprocError(errors.New("could not read params"), w)
        return
    }

    if err := validateEmail(params.Email); err != nil {
        handleUnprocError(err, w)
        return
    }

    passwordDigest := string(md5.New().Sum([]byte(params.Password)))

    if params.FavoriteCake != u.FavoriteCake || passwordDigest !=
u.PasswordDigest {
        handleUnauthError(errors.New("unauthorized"), w)
        return
    }

    updatedUser := User{
        Email:           params.Email,
        PasswordDigest: passwordDigest,
        FavoriteCake:     params.FavoriteCake,
    }

    err = users.Add(updatedUser.Email, updatedUser)
    if err != nil {
        handleUnprocError(err, w)
        return
    }

    _, err = users.Delete(u.Email)
    if err != nil {
        handleUnprocError(err, w)
        return
    }
    w.WriteHeader(http.StatusOK)
    _, _ = w.Write([]byte("email updated"))
}

```

```

func updatePasswordHandler(w http.ResponseWriter, r *http.Request, u User, users
UserRepository) {
    params := &UserRegisterParams{}
    err := json.NewDecoder(r.Body).Decode(params)
    if err != nil {
        handleUnprocError(errors.New("could not read params"), w)
        return
    }

    if err := validatePassword(params.Password); err != nil {
        handleUnprocError(err, w)
        return
    }

    passwordDigest := string(md5.New().Sum([]byte(params.Password)))

    if params.Email != u.Email || params.FavoriteCake != u.FavoriteCake {
        handleUnauthError(errors.New("unauthorized"), w)
        return
    }

    updatedUser := User{
        Email:           params.Email,
        PasswordDigest:  passwordDigest,
        FavoriteCake:     params.FavoriteCake,
    }

    err = users.Update(params.Email, updatedUser)
    if err != nil {
        handleUnprocError(err, w)
        return
    }
    w.WriteHeader(http.StatusOK)
    _, _ = w.Write([]byte("password updated"))
}

func handleUnprocError(err error, w http.ResponseWriter) {
    handleError(err, 422, w)
}

func handleUnauthError(err error, w http.ResponseWriter) {
    handleError(err, 401, w)
}

func handleError(err error, status int, w http.ResponseWriter) {
    w.WriteHeader(status)
    _, _ = w.Write([]byte(err.Error()))
}

```

main.go:

```

...

func newRouter(u *UserService, jwtService *JWTService) *mux.Router {
    r := mux.NewRouter()

    r.HandleFunc("/user/register", u.Register).Methods(http.MethodPost)

```

```
    r.HandleFunc("/user/jwt", wrapJwt(jwtService,
u.JWT)).Methods(http.MethodPost)
    r.HandleFunc("/user/me", jwtService.jwtAuth(u.repository,
getCakeHandler)).Methods(http.MethodGet)
    r.HandleFunc("/user/favorite_cake", jwtService.jwtAuth(u.repository,
updateCakeHandler)).Methods(http.MethodPut)
    r.HandleFunc("/user/email", jwtService.jwtAuth(u.repository,
updateEmailHandler)).Methods(http.MethodPut)
    r.HandleFunc("/user/password", jwtService.jwtAuth(u.repository,
updatePasswordHandler)).Methods(http.MethodPut)
    return r
}

...
```