Міністерство освіти і науки України
Національний Технічний Університет України
«Київський Політехнічний Інститут»
Навчально-науковий комплекс «Інститут прикладного системного аналізу»
Кафедра системного проектування

Лабораторна робота No6

з дисципліни

“Проектування інформаційних систем”

Модульне тестування (Unit-тести) та рефакторинг

Виконав:
студент групи ДА-72
Черномаз В.С.
В-28

Київ – 2020

Мета роботи: оволодіти навичками створення програмного забезпечення за метолологією TDD та ознайомитися з процедурами рефакторинга.

Задача:

1. Використовувати методологію Test Driven Development для створення класів архітектурної програмної моделі.

2. Скласти тестові сценарії, які продемонструють функціювання всіх методів проектованої моделі.

3. Виконати юніт-тестування складових частин (внутрішніх класів), що реалізують об'єкт моделювання.

4. Виконати "зовнішнє" юніт-тестування для API.

5. Провести рефакторинг коду програми, для поліпшення реалізації.

Хід роботи

1. Скласти тестові сценарії

No	Сценарій
1	● Створити нотатку ● Переглянути нотатку
2	● Створити нотатку ● Переглянути нотатку ● Редагувати нотатку ● Переглянути нотатку
3	● Створити нотатку ● Переглянути нотатку ● Видалити нотатку ● Переконатись, що нотатки більше немає
4	● Створити нотатку ● Спробувати дістати нотатку з неправильним ім’ям ● Отримати помилку
5	● Створити нотатку ● Спробувати видалити нотатку з неправильним ім’ям ● Отримати помилку

2. Виконати юніт-тестування для складових частин

Мова програмування Go надає вбудовану бібліотеку для тестування testing. Вона працює так, що необхідно назвати файл **<ім'я>_test.go** і після команди **go test** відбувається виконання цього файлу. Для даної роботи був створений файл **main_test.go**, який містить в собі 7 тестів.

Приклад тесту:

```
func TestGetNoteSuccess(t * testing.T) {
    resp, err := http.Get(fmt.Sprintf("%s/get_note/note_name", ts.URL))

    if err != nil {
        log.Printf("Expected not errors, gor %v", err)
    }

    if resp.StatusCode != 200 {
        log.Printf("Expected status code 200, got %v", resp.StatusCode)
    }

    var result map[string]interface{}

    json.NewDecoder(resp.Body).Decode(&result)

    if result["name"] != "note_name" {
        log.Printf("Expected note name \"note_name\", got %s", result["name"])
    }
}
```

В даному тесті перевіряється отримання нотатки після її успішного створення. Можна побачити перевірки на statusCode та правильне ім'я нотатки.

Результат після виконання файлу:

```
(base) Users-MacBook-Pro-2:backend vadym.chernomaz$ go test
[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery middleware already attached.

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)

[GIN-debug] POST   /add_note                --> _/Users/user/project/notes/backend.addNote (3 handlers)
[GIN-debug] GET    /get_note/:name          --> _/Users/user/project/notes/backend.getNote (3 handlers)
[GIN-debug] PUT    /update_note/:name       --> _/Users/user/project/notes/backend.updateNote (3 handlers)
[GIN-debug] DELETE /delete_note/:name       --> _/Users/user/project/notes/backend.deleteNote (3 handlers)
2020/11/27 23:28:30 test2
[GIN] 2020/11/27 - 23:28:30 | 400 |     1.22751ms |       127.0.0.1 | POST   "/add_note"
2020/11/27 23:28:30 Expected status code 400, got 400
2020/11/27 23:28:30 test2
[GIN] 2020/11/27 - 23:28:30 | 200 |   29.069951ms |       127.0.0.1 | POST   "/add_note"
[GIN] 2020/11/27 - 23:28:30 | 200 |   27.754026ms |       127.0.0.1 | GET    "/get_note/note_name"
[GIN] 2020/11/27 - 23:28:30 | 200 |   34.640344ms |       127.0.0.1 | PUT    "/update_note/note_name"
[GIN] 2020/11/27 - 23:28:30 | 200 |   34.386085ms |       127.0.0.1 | DELETE "/delete_note/updated_name"
[GIN] 2020/11/27 - 23:28:30 | 200 |   32.561756ms |       127.0.0.1 | DELETE "/delete_note/updated_name"
2020/11/27 23:28:30 Error while retrieving documents from MongoDB
[GIN] 2020/11/27 - 23:28:30 | 500 |   26.649283ms |       127.0.0.1 | GET    "/get_note/fail_name"
2020/11/27 23:28:30 Expected status code 500, got 500
PASS
ok      _/Users/user/project/notes/backend      0.229s
```

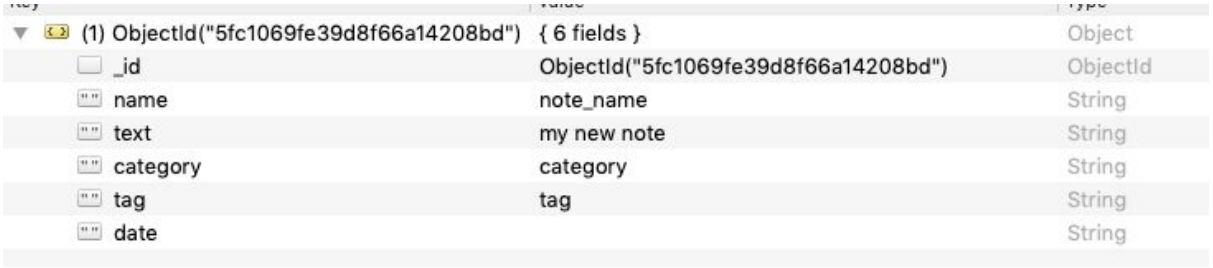
Внизу можна побачити відмітку PASS, що сигналізує про успішне проходження всіх тестів, та показник часу проходження тестів.

Для зовнішнього тестування використовувалась утиліта curl, за допомогою якої відбувалось звертання до API.

API AddNote
Команда:

```
curl -H "Content-Type: application/json" -X POST -d '{"name":"note_name","text":"my new note", "category": "category", "tag": "tag"}' http://localhost:3000/add_note
```

Результат:



(1) ObjectId("5fc1069fe39d8f66a14208bd")	{ 6 fields }	Object
_id	ObjectId("5fc1069fe39d8f66a14208bd")	ObjectId
name	note_name	String
text	my new note	String
category	category	String
tag	tag	String
date		String

API GetNote

Команда:

```
curl -H "Content-Type: application/json" -X GET http://localhost:3000/get_note/note_name
```

Результат:

{"name":"note_name","text":"my new note","category":"category","tag":"tag"}

API UpdateNote

Команда:

```
curl -H "Content-Type: application/json" -X PUT -d'{"name": "note_name_to_update",
"text": "updated text", "category": "updated category", "tag": "updated tag", "date":
"updated_date"}' http://localhost:3000/update_note/note_name
```

Результат:

▼ (1) ObjectId("5fc1069fe39d8f66a14208bd")	{ 6 fields }	Object
_id	ObjectId("5fc1069fe39d8f66a14208bd")	ObjectId
name	note_name_to_update	String
text	updated text	String
category	updated category	String
tag	updated tag	String
date	updated_date	String

API DeleteNote

Команда:

```
curl -H "Content-Type: application/json" -X DELETE
http://localhost:3000/delete_note/note_name_to_update
```

Результат:

```
Fetched 0 record(s) in 4ms
```

3. Розробка функціоналу за TDD
   Для початку створено тест для створення користувача у MongoDB

```
func TestAddUserSuccess(t *testing.T) {
        requestBody, err := json.Marshal(map[string]string{
                "name":     "note_name",
                "email":    "note_text",
                "password": "note_category",
        })

        if err != nil {
```

```
                log.Printf("Expected no error, got %v", err)
        }

        resp, err := http.Post(fmt.Sprintf("%s/add_user", ts.URL),
"application/json", bytes.NewBuffer(requestBody))

        if err != nil {
                log.Printf("Expected no error, got %v", err)
        }

        if resp.StatusCode != 200 {
                log.Printf("Expected status code 200, got %v", resp.StatusCode)
        }
}
```

Потім створену саму API AddUser

```
func addUser(c *gin.Context) {
        client, ctx := connectToMongo()

        var requestBody User
        if err := c.ShouldBind(&requestBody); err != nil {
                c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
                return
        }

        collection := client.Database("notes").Collection("accounts")
        insertResult, err := collection.InsertOne(ctx, requestBody)
        if err != nil {
                log.Panic(err)
        }

        c.JSON(http.StatusOK, insertResult.InsertedID)
        client.Disconnect(ctx)
}
```

Результат виконання тесту

```
[GIN] 2020/12/09 - 13:44:18 | 200 |   36.859036ms |       127.0.0.1 | POST     "/add_user"
PASS
```

▼ (2) ObjectId("5fd0b8920e3cdcbe7348a6de")	{ 4 fields }	Object
_id	ObjectId("5fd0b8920e3cdcbe7348a6de")	ObjectId
name	user_name	String
email	user_email	String
password	user_password	String

4. Провести рефакторинг

   Передусім слід зазначити, що компілятор Go є дуже суворим до стилю коду. Не можна просто імпортувати бібліотеку, а потім її не використовувати - з'явиться помилка при компілюванні. Саме тому при розробці на мові Go не виникає питань типу "Чи потрібно писати коментар до цієї функції?". Відповідь: якщо вона публічна, то обов'язково, компілятор про це нагадає.

   Після проведення рефакторингу функцію main, яка до цього містила весь функціонал, була розбита на багато маленьких функцій, які відповідають як за API, так і за запуск серверу і підключення до MongoDB.

   Репозиторій з кодом: https://github.com/vaduja/pis

   Висновок: у ході виконання роботи проект був доповнений юніт-тестами та був виконаний рефакторинг коду. Ці два етапа є обов'язковими при розробці будь-якої системи, оскільки піклуються як про зручність користувача, так і про зручність розробника. До того ж, слід розуміти, що важливим є підхід Test-Driven Development, тобто написання тестів ще до того як був написаний код. Це дає уявлення про те, як код повинен працювати та на порядок зменшує можливість появи неочікуваних багів.