НТУУ КПІ ім. Ігоря Сікорського
Інститут прикладного системного аналізу
Кафедра Системного Проектування

Лабораторна робота №5

“Проектування інформаційних систем”

Виконав:
студенти групи ДА-72
Самвелян Артур

Київ - 2020

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

Хід роботи:

1. Створити тестові сценарії:

№	Сценарій
1	- створити рахунок - переглянути рахунок
2	- створити рахунок - переглянути рахунок - видалити рахунок - переглянути всі рахунки, щоб упевнитися що створений рахунок був видалений
3	- створити рахунок - переглянути рахунок - змінити дані рахунку - переконатися в правильності змін
4	- створити рахунок - спробувати переглянути рахунок з неправильним ім’ям - отримати помилку

2. Виконати юніт-тестування для складових частин

Приклад тесту:

```python
class TestUserModel(unittest.TestCase):
    def test_create_user_and_get_user_by_id_email(self):
        user_json = {
            "id": 1,
            "name": "Mike",
            "email": "myemail@google.com",
            "password": "kdsjflkjflsk",
            "created_at": user_model.datetime.utcnow()
        }
        user = user_model.UserCreate(**user_json)
        user_id = user_model.create_user(user)
        user_2 = user_model.get_user_by_id(user_id)
        user_3 = user_model.get_user_by_email(user_json['email'])
        self.assertEqual(user_2, user_3)
        self.assertEqual(user_2.id, user_json['id'])
        self.assertEqual(user_2.name, user_json['name'])
        self.assertEqual(user_2.email, user_json['email'])


    def test_create_and_decode_token(self):
        id = 372
        auth_token = user_model.create_token(id)
        self.assertTrue(isinstance(auth_token, bytes))
        self.assertTrue(user_model.decode_token(auth_token) == id)
```

Маємо приклад юніт-тесту для UserModel - бізнес-логіці юзера в даній інформаційній системі.

Перший тест перевіряє створення нового юзеру, потім отримання його через пошту і через id і валідації полів.

Другий тест перевіряє правильність логіки аутентифікації.

```
Ran 2 tests in 0.003s

FAILED (errors=1)
(web) ➜  app git:(dev) ✗ python tests/test_user_model.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.011s

OK
(web) ➜  app git:(dev) ✗
```

Запускаємо тести, усі тести пройдені.

3. Виконати рефакторінг:

```python
from fastapi import APIRouter, Depends

from models import budget as budget_model
from models.dependencies import get_current_user

router = APIRouter()


@router.get('/')
def get_all_budgets(user_id: int = Depends(get_current_user)):
    budgets = budget_model.get_all_budgets(user_id)
    return budgets


@router.get('/{id}', response_model=budget_model.BudgetGet)
def get_budget_by_id(id: int, user_id: int = Depends(get_current_user)):
    budget = budget_model.get_budget_by_id(id, user_id)
    return budget


@router.post('/')
def create_budget(budget: budget_model.BudgetCreate,
                  user_id: int = Depends(get_current_user)):
    id = budget_model.create_budget(budget, user_id)
    return id
```

Проведемо рефакторінг на прикладі захисту ендпоінтів. Кожен ендпоінт захищений JWT токеном, даний захист до рефакторінга прописувався до кожного ендпоінту, тим самим ми наблюдали дублювання коду.

Дублювання коду є одним з сигналів того що треба робити рефакторінг, тому видалимо продубльований код з ендпоінтів та впишемо його у головний файл. Ми можемо це зробити так як фреймворк дає змогу робити

```python
import uvicorn
from fastapi import FastAPI

from routers.users import router as users_router
from routers.budgets import router as budgets_router
from routers.incomes import router as incomes_router
from routers.expenses import router as expenses_router


app = FastAPI()


app.include_router(users_router,
                   prefix='/users')
app.include_router(budgets_router,
                   prefix='/budgets',
                   dependencies=[Depends(get_current_user)])
app.include_router(incomes_router,
                   prefix='/incomes')
app.include_router(expenses_router,
                   prefix='/expenses')


if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

Dependency Injection (на основі якого лежить захист наших ендпоінтів) до всіх ендпоінтів окремого відображення.

Висновки: в даній лабораторній роботі було створено юніт-тести для тестування юзер моделі (за технікою TDD) даної інформаційної систем, а також на прикладі захисту ендпоінтів було показано рефакторіг у випадку дублювання коду. TDD техніка граю дуже велику роль у створенні великих систем; з її допомогою можна уникнути багатьох помилок на стадії розробки програмного продукту. Єдиним мінусом даної техніки вважають кількість коду яку треба написати (та час затрачений на цей код), але в більшій кількості випадків час витрачений на створення тестів в даній методології набагато менший часу витраченого на пошук помилок без даної техніки.

Рефакторінг також є одним з головних етапів розробки. Через деякий час розробки система наростає кодом “із запашком”, який в подальшому можна відкоригувати.