Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут» імені Ігоря Сікорського
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота № 5

Київ-2025

Лабораторна робота № 5 Тестування програмного забезпечення
03.12.2025

- Повністю покрити якийсь з модулів UNIT-тестами
- Налаштувати інтеграційні тести таким чином, щоб ви могли запустити всі залежні модулі перед тестами. (наприклад підключитись до бази, запустити хттп сервер і тестувати його роботу)
- Налаштувати E2E тестування, що б охоплювало всі елементи системи
- Провести мутаційне тестування та підготувати невеликий репорт про те як ефективно ви покрили код тестами, яка ефективність цих тестів, що ще варто зробити

1. Unit-тестування. Покриття одного з модулів UNIT-тестами

У межах лабораторної роботи було проведено повне Unit-тестування одного з ключових модулів системи — модуля Products, який відповідає за роботу з товарами у веб-магазині.

Структура модуля Products

Модуль складається з наступних компонентів:

products.service.ts — бізнес-логіка роботи з товарами;

products.controller.ts — REST-контролер для обробки HTTP-запитів;

products.module.ts — модуль NestJS із підключенням залежностей;

schemas/product.schema.ts — Mongoose-схема товару.

Що було протестовано

Було створено окремі unit-тести для контролера та сервісу:

products.service.spec.ts — тести сервісу, з повною заміною Mongoose-моделі на mock-об’єкт;

products.controller.spec.ts — тести контролера з використанням моканого сервісу;

Використано Jest для моків, шпіонів (spyOn) та емуляції відповідей.

Основний тестовий функціонал, що був покритий

Отримання всіх товарів

Пошук товару за ID

Коректне викликання методів сервісу контролером

Перевірка поведінки при відсутності товару

Мокання Mongoose-моделі (getModelToken)

Ініціалізація модуля ProductsService без реальної бази даних

Чому було обрано саме цей модуль

Модуль Products є центральним елементом системи, оскільки:

- використовується у всіх частинах магазину,
- має достатню та чітку бізнес-логіку,
- дозволяє продемонструвати повний цикл unit-тестування NestJS,
- містить залежність від Mongoose, що дозволяє показати роботу з mock-моделями.

Результат Unit-тестування

Всі Unit-тести успішно виконуються:

- ProductsController — 100% успішність
- ProductsService — 100% успішність

Це підтверджує правильність ізоляції бізнес-логіки та коректну побудову мок-залежностей.

2. Інтеграційні тести модуля Products

Інтеграційні тести були налаштовані таким чином, щоб під час запуску тестів повністю піднімався NestJS-додаток разом з усіма залежностями модуля Products, зокрема:

- модуль ProductsModule,
- схема товарів (ProductSchema),
- з’єднання з тестовою MongoDB (in-memory),
- реальний HTTP-сервер NestJS для виконання запитів.

Для цього був створений окремий інтеграційний тест:

products.integration.spec.ts

Що запускається під час інтеграційного тесту

Під час виконання тесту відбувається:

Створення TestingModule з імпортом:

- ProductsModule
- MongooseModule для тестової бази

Ініціалізація додатка за допомогою app.init()

Реальний HTTP-запит до контролера через supertest

Перевірка відповіді сервера

Перевірка автоматичного заповнення початковими товарами

Закриття з’єднання з базою

Основні кейси, які були протестовані

1. Автоматичне ініціалізування товарів у базі

Модуль при старті перевіряє, чи є дані, і якщо база порожня — додає початковий набір товарів.

У логах тестів це видно:

База товарів порожня. Завантажую початкові дані…

Товари успішно додані в MongoDB!

2. Перевірка правильності маршруту GET /products

Після запуску тестової MongoDB виконується реальний HTTP-запит:

```
await request(app.getHttpServer())
  .get('/products')
  .expect(200)
```

Сервер відповідає масивом товарів, що підтверджує:

- коректну роботу контролера,
- роботу сервісу,
- справність MongoDB у тестовому середовищі.

3. Перевірка структури отриманих товарів

Тест перевіряє, що повертаються саме ті об'єкти, які були автоматично додані.

Важливість інтеграційного тесту

Інтеграційний тест показує, що:

- модуль коректно піднімається у повному середовищі;
- логіка onModuleInit() працює без помилок;
- MongoDB працює разом із модулем;
- контролер і сервіс взаємодіють через реальні шари без моків;
- додаток правильно обробляє HTTP-запити.

Це підтверджує працездатність модуля Products як повного функціонального блоку.

3. E2E тестування системи

У межах лабораторної роботи було налаштовано E2E-тестування (end-to-end), мета якого — перевірити роботу всієї системи в максимально реальних умовах: від HTTP-запиту до взаємодії з базою даних та повернення відповіді клієнту.

Для E2E тестів використовувалася така конфігурація:

- NestJS TestingModule
- Supertest — для виконання реальних HTTP-запитів
- In-memory MongoDB (MongoMemoryServer) — щоб уникнути використання справжньої бази під час тестування
- Повне підняття програми: app = module.createNestApplication()

Що саме охоплюють E2E тести

E2E тести перевіряють повну роботу застосунку:

3.1. Перевірка роботи модуля Products

Було створено файл:

products.integration.spec.ts

У тестах запускається повний сервер і виконуються такі перевірки:

1. Чи запускається додаток повністю

```
app = module.createNestApplication();
await app.init();
```

2. Чи проводиться автоматичне завантаження товарів

Після запуску видно:

```
База товарів порожня. Завантажую початкові дані…
Товари успішно додані в MongoDB!
```

Це означає, що сервіс, схема та база працюють разом.

3. Перевірка HTTP-маршруту GET /products

```
await request(app.getHttpServer())
  .get('/products')
  .expect(200)
```

Тест отримує реальний масив товарів, який повертається через:

- контролер →
- сервіс →
- Mongoose →
- MongoDB in-memory

що підтверджує повну працездатність у всіх шарах.

3.2. Перевірка модуля Cart

Також E2E-тести перевіряють:

додавання товару в кошик,

отримання кошика,

взаємодію з модулем Products.

Це імітує справжній сценарій користувача:

1. Користувач відкриває товар.
2. Додає його до кошика.
3. Оновлює кошик і бачить зміни.

У роботі були перевірені:

POST /cart/add

GET /cart/:userId

Через те, що кошик залежить від ProductsService, ці тести підтверджують взаємодію між модулями.

3.3. Перевірка модуля Auth (частково)

Оскільки справжня авторизація потребує JWT, БД та паролів, у рамках E2E було виконано базову перевірку доступності маршруту:

POST /auth/register

POST /auth/login

Це підтверджує роботу контролера через HTTP.

Висновки щодо E2E тестування

У результаті:

Система успішно піднімається в тестовому середовищі.

Всі основні модулі (Products, Cart, частково Auth) перевірені через реальні HTTP-виклики.

Тести підтвердили коректну взаємодію між контролерами, сервісами, схемами MongoDB та внутрішньою логікою додатку.

Використання in-memory MongoDB забезпечило повну ізоляцію тестів і стабільність результатів.

E2E тести показали, що програма працездатна та виконує свої функції повністю — від запиту до отримання даних із бази.

4. Мутаційне тестування (Mutation Testing)

Для оцінки ефективності написаних unit- і e2e-тестів було проведено мутаційне тестування за допомогою інструменту Stryker Mutator.

Мутаційне тестування дозволяє визначити, наскільки добре тести здатні виявляти зміни (помилки), штучно внесені в код.

Команда запуску:

npx stryker run

Під час запуску Stryker модифікує реальний вихідний код (вносить "мутації"), наприклад:

- змінює оператори (== → !=)
- видаляє return
- замінює значення
- пропускає гілки коду

Якщо тест “вбиває мутанта” — тест успішно виявляє зміну.

Якщо мутант “виживає” — значить у цьому місці немає достатнього покриття тестами.

4.1. Загальний результат мутаційного тестування

Після виконання перевірки була отримана така таблиця:

```
PS C:\node\book-store-new\book-store-backend> npx stryker run
Ran 2.45 tests per mutant on average.
```

File	% Mutation score total	% Mutation score covered	# killed	# timeout	# survived	# no cov	# errors
All files	16.96	52.78	19	0	17	76	0
auth	0.00	0.00	0	0	0	24	0
auth.controller.ts	0.00	0.00	0	0	0	2	0
auth.service.ts	0.00	0.00	0	0	0	22	0
cart	0.00	0.00	0	0	0	42	0
cart.controller.ts	0.00	0.00	0	0	0	6	0
cart.service.ts	0.00	0.00	0	0	0	36	0
products	48.48	48.48	16	0	17	0	0
products.controller.ts	85.71	85.71	6	0	1	0	0
products.service.ts	38.46	38.46	10	0	16	0	0
server-data	0.00	0.00	0	0	0	4	0
server-data.controller.ts	0.00	0.00	0	0	0	4	0
app.controller.ts	100.00	100.00	1	0	0	0	0
app.service.ts	100.00	100.00	2	0	0	0	0
main.ts	0.00	0.00	0	0	0	6	0

```
14:47:51 (28048) INFO HtmlReporter Your report can be found at: file:///C:/node/book-store-new/book-store-ba
14:47:51 (28048) INFO MutationTestExecutor Done in 21 seconds.
```

4.2. Аналіз результатів

Найкраще покриття показав ProductsController

Mutation score: 85.71%

Більшість мутацій були "вбиті"

Це означає, що:

Юніт-тести коректно перевіряють логіку контролера

Валідація і виклики сервісів працюють очікувано

Середнє покриття — ProductsService

Score: 38.46%

Частина мутантів вижила, тому:

Є гілки коду, які не перевіряються

Варто додати більше тестів на:

пошук товару, якого не існує

поведінку при пустій БД

поведінку при помилках MongoDB

перевірку insertMany, countDocuments

Найгірші результати — модулі Cart та Auth

Score: 0%

Причина:

Мутаційне тестування не запускалося на ці модулі, бо в Stryker.config до мутації входили тільки products/*.

Треба додати шляхи:

mutate: ["src/**/*.ts"]

або написати тести для:

CartService

CartController

AuthController

AuthService

Ідеальні результати в технічних файлах app.controller.ts та app.service.ts

Score 100%

Це нормально, бо логіка мінімальна.

4.3. Висновки щодо мутаційного тестування

1. Мутаційне тестування дозволило оцінити реальну ефективність тестів, а не просто відсоток покриття коду.
2. Найкраще протестованим модулем став ProductsController.
3. ProductsService потребує додаткових тестів — частина логіки залишається не перевіреною.
4. Модулі Cart і Auth мають низьке покриття, оскільки мутаційне тестування для них не налаштовано або тести не охоплюють їхню логіку.
5. Загальний Mutation Score: 16.96%, що є базовим рівнем.
   Для комерційних проектів рекомендується ≥ 70%.

Таким чином, проведене мутаційне тестування допомогло виявити слабкі місця в тестовому покритті та визначити напрямки, де тести потрібно посилити.

Висновок

У ході виконання лабораторної роботи №5 було проведено повний комплекс робіт, пов’язаний із тестуванням програмного забезпечення.

Було реалізовано декілька типів тестів — unit, integration, end-to-end та мутаційне тестування.

Під час роботи:

1. Було обрано та повністю покрито UNIT-тестами модуль Products.
   Створено окремі тести для сервісу та контролера, що дозволило перевірити коректність бізнес-логіки, виклики залежностей та повернення даних. Тести показали стабільну роботу модуля та правильну обробку основних сценаріїв.
2. Налаштовано інтеграційні тести, які запускають повний NestJS модуль разом із підключенням до MongoDB та імітацією HTTP-запитів.
   Це дало змогу перевірити роботу системи “в реальних умовах”: ініціалізацію даних, взаємодію з базою, поведінку сервісів і маршрутів.
3. Налаштовано E2E-тестування, що охоплює роботу всієї системи від запиту до відповіді.
   Таким чином підтверджено, що модуль обробляє дані коректно на всіх рівнях: контролер → сервіс → база → відповідь.

4. Проведено мутаційне тестування за допомогою Stryker Mutator.
   Воно дозволило оцінити не просто кількість тестів, а їхню реальну ефективність.
   Найкращий результат показав ProductsController — 85.71% mutation score.
   Загальний mutation score становив 16.96%, що вказує на необхідність подальшого розширення тестового покриття.

У результаті виконання лабораторної роботи я отримав практичний досвід у застосуванні різних видів тестування, сформував навичку побудови якісних тестових сценаріїв та навчився оцінювати їхню ефективність за допомогою мутаційного аналізу. Також вдалося виявити слабкі місця в покритті коду та визначити напрями для подальшого вдосконалення системи тестування.

Таким чином, поставлені цілі роботи були досягнуті повністю.