Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

Рекомендовано до захисту:
протокол засідання кафедри
№ _____ від _____._____.2025 р.
Завідувач кафедри _____

_____ _____
(підпис) (прізвище та ініціали)

Пояснювальна записка

до кваліфікаційної роботи бакалавра

на тему “Розробка засобу створення unit-тестів для веб-застосунків”

Захищено на засіданні ЕК № _____
протокол № _____ від _____._____.2025 р.
Оцінка _____ / _____
Голова ЕК

_____ _____
(підпис) (прізвище та ініціали)

Виконав:
студент 4 курсу, групи ЗКС-41
Спеціальності:
122 Комп’ютерні науки
(шифр і назва спеціальності підготовки)
Брага Катерина Сергіївна
(прізвище, ім’я та по батькові, підпис)
Керівник ст. викл. Мішин О.В.

(ступень, звання, прізвище та ініціали, підпис)
Рецензент _____

(ступень, звання, прізвище та ініціали, підпис)

Харків – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

Кафедра математичного моделювання та аналізу даних

Рівень вищої освіти (освітньо-кваліфікаційний рівень) бакалавр
Галузь знань: 12 Інформаційні технології
Спеціальність: 122 Комп'ютерні науки
Освітня програма: Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.О. завідувача кафедри _____
Володимир СТРУКОВ
«____» _____20___ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ (ПРОЕКТ)

Браги Катерини Сергіївни
(прізвище, ім'я, по батькові студента)

1. Тема роботи "Розробка засобу створення unit-тестів для веб-застосунків"

керівник роботи Мішин Олександр Вікторович,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "16" квітня 2025року №4101-5/962

2.Строк подання студентом роботи 6 червня 2025 року

3. Перелік питань, які потрібно розробити
Провести дослідження найбільш популярних фреймворків для створення веб застосунків та можливості додавання unit-тестів до них. Визначити та дослідити інструменти для створення unit-тестів та визначити найбільш придатний для використання. Дослідити обраний фреймворк для розробки вебзастосунку.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Формулювання мети та повного переліку завдань для виконання кваліфікаційної роботи (КР).	09.01 - 30.01
2	Пошук та аналіз інформаційних джерел за темою КР.	01.02 - 28.02
3	Визначення фреймворку для проекту та для написання тестів.	25.02 - 15.03
4	Налаштування та інтеграція в проект фреймворку для написання автоматизованих тестів	15.03-15.04
5	Розробка набору тест кейсів для автоматизації та їх реалізація	15.03-15.04
6	Налаштування підходу CI/CD для проекту. Та дослідження можливості інтеграції з TMS.	15.04 - 20.05
7	Підсумок отриманих результатів та оформлення пакету документів для процедури захисту КР в екзаменаційній комісії ННІ КН та ШІ.	20.05-01.06

5. Дата видачі завдання “9” січня 2025 року

Студент ___К. С. Брага___ _____
ініціали, прізвище підпис

Керівник роботи ___О. В. Мішин____ _____
ініціали, прізвище підпис

АНОТАЦІЯ

Дипломна робота обсягом 42 сторінок пояснювальної записки, що складається з 3 розділів, містить 18 рисунків, 2 таблиці та 13 джерел згідно з переліком літератури.

У роботі розглянуто підхід до впровадження автоматизованого тестування в сучасні веб-застосунки на базі фреймворку Next.js. Описано основні рівні автоматизації тестування — unit, інтеграційні та UI тести — та роль кожного з них у забезпеченні якості програмного продукту. Основну увагу приділено реалізації unit-тестів із використанням Jest та React Testing Library.

Проведено аналіз інструментів для побудови автоматизованих тестів, налаштовано збір звітів щодо покриття коду тестами та результати виконання. Розроблено та реалізовано набір тестових випадків для ключових компонентів проекту (форми створення магазину, товару, компонент завантаження файлів, форма оновлення магазину). Показано інтеграцію процесу автоматичного тестування у CI/CD пайплайн за допомогою GitHub Actions.

Також розглянуто можливості використання системи управління тестами Qase для організації автоматизованого тестування на рівні процесів команди. Підкреслено важливість впровадження автоматизованого тестування для підтримки стабільності та якості проектів у швидко змінному середовищі веб-розробки.

Ключові слова: Next.js, Jest, unit-тестування, CI/CD, Qase, автоматизація тестування, React, якість веб-застосунків.

ABSTRACT

The diploma thesis consists of a 42-page explanatory note and includes three chapters, 18 figures, 2 tables, and 13 references listed in the bibliography.

This work explores the approach to implementing automated testing in modern web applications based on the Next.js framework. The study describes the main levels of automated testing — unit, integration, and UI tests — and the role each level plays in ensuring software quality. The primary focus is on the implementation of unit tests using Jest and React Testing Library.

An analysis of tools for building automated tests was conducted, along with the configuration of reporting on code coverage and test execution results. A set of test cases was developed and implemented for key project components (store creation form, product form, file upload component, store update form). The integration of the automated testing process into the CI/CD pipeline using GitHub Actions is demonstrated.

Additionally, the study examines the use of the Qase test management system to help organize automated testing within team processes. The importance of introducing automated testing to maintain the stability and quality of projects in the fast-paced environment of web development is emphasized.

Keywords: Next.js, Jest, unit testing, CI/CD, Qase, test automation, React, web application quality.

ЗМІСТ

ВСТУП

Сучасний світ прискорюється з кожним днем і веб додатки мають встигати за ним. Саме тому однією з найбільших вимог до команд розробки

підлаштовувати його під потреби клієнтів та користувачів, а також мати можливість масштабування, бо можливість захопити ринок є критичним для багатьох таких проектів. В цих умовах великим викликом стає підтримка рівня якості застосунку під час його змін та розширення. Саме для вирішенням цієї проблеми застосовують підходи та інструменти для автоматизованого тестування. В сучасних реаліях питання стоїть не в тому чи потрібні автоматизовані тести в проекті, а в тому як саме краще їх організувати та імплементувати в проект.

Автоматизацію тестування прийнято ділити на три основних рівня: unit(component), service, та UI.

На unit (component) рівні тестуються окремі компоненти системи та основна увага приділяється якості коду. Service рівень можна поділити на три групи тестів: інтеграційні – взаємодія між компонентами, контрактні – взаємодія між сервісами та мікросервісами, та API(Інтерфейс Прикладного Програмування) – перевірка взаємодії сервісів з реальними даними через реальні точки входу(API endpoints). UI тести – взаємодіють з застосунком через інтерфейс користувача.

Тестові приклади можуть бути розподілені між цими рівнями в різний спосіб в залежності від проекту та стадії його розробки, але стандартним вважається так звана «піраміда» в основі якої лежать unit-тести в середині тести сервісів і тести інтерфейсу користувача на вершині [1].

Найкращою та найбільш розповсюдженою практикою є розпочинати автоматизацію з основи піраміди, а саме unit-тестів. Саме цьому буде приділено основну увагу у даній роботі.

Сучасні веб-застосунки все частіше будують на базі фреймворків. Одним з найбільш розповсюджених JavaSctipt фреймворків з яких є React. Він включає в себе набір інструментів для створення інтерактивних інтерфейсів, має набір вбудованих компонентів та. Підтримує створення власних інкапсульованих компонентів. За рахунок цього є дуже гнучким та дозволяє будувати застосунки зі складним інтерфейсом. Згідно з [2] майже 40% проектів в 2024 році використовували цей фреймворк. Саме на базі React був створений новий фреймворк Next.js який з'явився у 2016 році та стрімко набирає популярності. Він наслідує більшість переваг React але крім того має ряд вбудованих оптимізацій та унікальні інструменти як, наприклад, просунуту маршрутизацію та Middleware для зручних перенаправлень запитів. Згідно [3] цей фреймворк часто обирають для проектів з великим об'ємом трафіку та його вже використовують такі великі сайти як: Github.com, Adobe.com, Netflix.com, Spotify.com та інші. То ж в даній роботі для імплементації автоматичних тестів був ораний проект реалізований на Next.js.

Далі в цій роботі буде оглянуто ряд інструментів для створення unit-тестів, розроблено і реалізовано набір тестових випадків, застосовані інструменти для створення звітів та аналізу покриття коду автоматичними тестами. Також буде розглянуто можливості та способи налагодження CI/CD(Continuous Integration/ Continuous Delivery) підходу для проекту на Next.js.

1. ПОСТАНОВКА ЗАДАЧІ

1.1 Об’єкт дослідження

Об’єктом дослідження у даній роботі є процес впровадження автоматизації в проект веб-застосунку. Він передбачає використання інструментів для написання unit-тестів, оцінки покриття коду тестами, генерацію звітів та інтеграцію до CI/CD процесу.

1.2 Предмет дослідження

Предметом дослідження засіб для автоматичного тестування що включає в себе:

1. набір реалізованих тест кейсів для тестування компонентів в проекті;

2. звіти, що показують стан покриття колу тестами та інформацію про виконання тестів;

3. налаштування для автоматичного запуску тестового набору в рамках CI/CD процесу.

1.3 Мета роботи

Метою роботи є створення засіб для автоматичного тестування, що дозволяє інтегрувати автоматичні тести в існуючий проект веб-застосунку, відслідковувати покриття тестами, виконання тестів, а також додавати нові тести.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Аналіз предметної області.

a) Провести дослідження найбільш популярних фреймворків для створення веб застосунків та можливості додавання unit-тестів до них.

b) Визначити та дослідити інструменти для створення unit-тестів та визначити найбільш придатний для використання.

c) Дослідити обраний фреймворк для розробки веб-застосунку.

2. Проектування та реалізація системи.

a) Розробити набір тестових випадків для компонентів проекту.

b) Налаштувати інструменти для розробки unit-тестів для інтеграції в проект у проект.

c) Реалізувати unit-тести для проекту

d) Налаштувати генерацію звітів для аналізу покриття та результатів виконання тестів.

e) Налаштувати CI/CD для автоматичного запуску тестів.

3. Оформлення результатів роботи

a) Складання пояснювальної записки та презентації.

2. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

2.1.1 Важливість автоматизованого тестування

Часто проекти, особливо стартапи, починають невеликі команди що на початкових етапах покладаються виключно на мануальне тестування. Це стає проблемою коли проект починає швидко зростати і потребує додавання нових функцій та змін. На початку проект невеликий, тож прогрес йде швидко, код який в ньому є зазвичай добре продуманий на етапі планування тож все працює за специфікаціями. Але з часом команді доводиться вкладати все більше і більше зусиль для того щоб досягти прогресу. Процес розробки значно сповільнюється, а часто може взагалі зупинитись і команда застрягне у виправленні помилок що виникли після додавання нового функціоналу. Цей ефект сповільнення швидкості розробки називають ентропією програмного забезпечення. Кожен раз коли потрібно внести правки в код проекту ентропія зростає і з часом застосунок втрачає стабільність. Виправлення однієї помилки тягне декілька інших, тож вносити зміни на вимогу клієнтів стає все важче й важче.

Саме щоб не допустити такого розвитку подій і варто впроваджувати автоматизовані тести. Правильний набір тестів може допомогти перевіряти роботу основного функціоналу після рефакторінга коду або додавання нових функцій. Впровадження автоматизованих тестів дає користь на дистанції, але також потребує додаткових зусиль від команди для їх написання та підтримки[4].

Часто виділяють 3 основних групи тестів:

- unit(component) тести - це автоматизовані тести, які перевіряють окремі функції, методи, класи або компоненти в ізоляції від решти програми. Основною метою є перевірка правильності логіки найменших одиниць коду.
- Інтеграційні тести - перевіряють взаємодію кількох компонентів системи, які вже інтегровані (наприклад, компонент + база даних або сервіс +

API). Їх основною метою є перевірка того, що компоненти працюють коректно між собою, а також їх взаємодію з сервісами.

- UI тести(тести інтерфейсу користувача) - тести, які перевіряють, як працює графічний інтерфейс програми з точки зору користувача. Вони імітують взаємодію користувача з додатком: кліки, введення тексту, навігацію, надсилання форм тощо.

Ці групи тестів розподіляють на три рівні що формують так звану «піраміду тестування»: unit тести - нижній рівень, найбільша частка функціоналу покривається саме за рахунок цього типу тестів; інтеграційні тести – середній рівень; UI тести верхній рівень – найменша частка, як правило покривають тільки основні сценарії поведінки користувача.[5]

Може бути розподілення тестів і в інший спосіб, але саме піраміда вважається еталонним варіантом через те що саме unit-тести є найпростішими, а тому найшвидшими і найстабільнішими, тож гарною практикою є починати впроваджувати автоматизацію саме з цього виду тестів і намагатись покривати якомога більше функціоналу за рахунок unit-тестів.[12] UI тести ж зазвичай довго виконуються та є менш стабільними через взаємодію з інтерфейсом користувача. Середню ланку інтеграційних тестів варто впроваджувати, коли більшість компонентів вже покриті unit-тестами[1]. В даній роботі покажемо виконання першого рівня впровадження автоматизованих тестів.

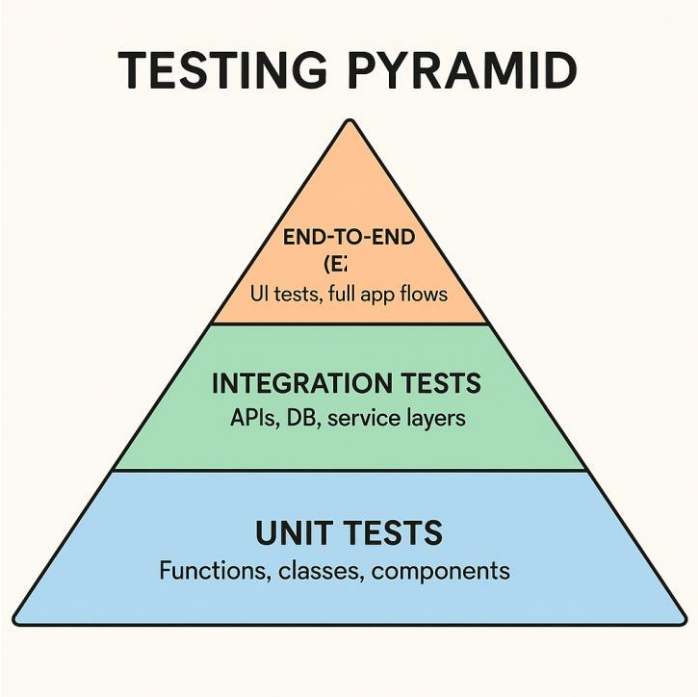


Рисунок 2.1. Піраміда тестування.

2.1.2 Фреймворк Next.js для веб-застосунку

Next.js – сучасний фреймворк для розробки веб застосунків що побудований на основі React та розширює його можливості. Цей фреймворк надає досить зручні інструменти для рендерінгу на стороні сервера та і цілому, для команд що складаються з універсальних(full-stack) розробників. Його часто обирають для проектів що приділяють багато уваги SEO і вимагають високої швидкості роботи, через гнучкість та можливості масштабування.

Одною з основних переваг цього фреймворка є вбудований роутінг, що і розкриває можливості для серверного рендерінгу і пришвидшує роботу інтерфейсу веб-застоссунку та позитивно впливає на SEO показники.

Розробка проекту на Next.js починається з трьох кроків:

1. Встановлення NodeJs
2. З допомогою менеджера пакетів відбувається ініціалізація проекта через виконання команди

npx create-next-app@latest

3. Налаштування конфігурації проекта з можливістю додавання різних додаткових пакетів та технологій. Наприклад, можна обрати використовувати чи ні TypeScript, ESLinkt, Tailwind Css тощо.

Після успішного створення проекту розробник отримає структуру папок та файлів наведену на рисунку нижче(рис. 2.1.2).

Розглянемо структурну огранізацію проекта в Next.js . Проект зазвичай має декілька каталогів високого рівня, це

- app – роутер застосунку
- pages – роутер окремих сторінок
- public – каталог для збереження статичних даних
- src – опціійна вихідна папка застосунку

Також є набір файлів на кореневому рівні:

- next.config.js – конфігураційний файл Next.js
- package.json – залежності проекту та скріпти
- instrumentation.ts – OpenTelemetry та службовий файл
- middleware.ts – файли технології запитів middleware
- .env – змінні середовища
- .env.local - локальні змінні середовища
- .env.production - змінні середовища для продакшен середовища
- .env.development - змінні середовища розробки
- .eslintrc.json – файл конфігурації ESLint
- Та інші файли що відповідають за конфігурацію git, TypeScript, JavaScript

Однією з ключових особливостей Next.js є роутінг. То ж варто приділити увагу цьому інструменту фреймворка.

В Next.js сторінка це React компонент що збирається з файлів що лежать в каталозі цієї сторінки. Кожна сторінка пов’язана с маршрутом, що спирається на ім’я файла. Тож якщо користувач створює каталог

pages/about.js то це експортує компонент та він стане доступним по шляху /about.

Є декілька типів маршрутизації в Next.js:

- Індексні маршрути – в цьому випадку роутер автоматично маршрутизує файл з ім’ям index в кореневу дерикторію, тож шлях pages/blog/index.js – буде аналогічним /blog
- Вкладені маршрути – якщо утворюється вкладена архітектура каталогів файли будуть автоматично маршрутизовані в той самий спосіб
- Сторінки із динамічними маршрутами – факично підтримують індекси, наприклад pages/posts/[id].js буде доступний за шляхом posts/1, posts/2 і так далі.

Розглянемо структурну огранізацію проекта в Next.js . Проект зазвичай має декілька каталогів високого рівня, це

- app – роутер застосунку
- pages – роутер окремих сторінок
- public – каталог для збереження статичних даних
- src – опціійна вихідна папка застосунку

Також є набір файлів на кореневому рівні:

- next.config.js – конфігураційний файл Next.js
- package.json – залежності проекту та скріпти
- instrumentation.ts – OpenTelemetry та службовий файл
- middleware.ts – файли технології запитів middleware
- .env – змінні середовища
- .env.local - локальні змінні середовища
- .env.production - змінні середовища для продакшен середовища
- .env.development - змінні середовища розробки
- .eslintrc.json – файл конфігурації ESLint

- Та інші файли що відповідають за конфігурацію git, TypeScript, JavaScript

Однією з ключових особливостей Next.js є роутінг. То ж варто приділити увагу цьому інструменту фреймворка.

В Next.js сторінка це React компонент що збирається з файлів що лежать в каталозі цієї сторінки. Кожна сторінка пов'язана с маршрутом, що спирається на ім'я файла. Тож якщо користувач створює каталог pages/about.js то це експортує компонент та він стане доступним по шляху /about.

Є декілька типів маршрутизації в Next.js:

- Індексні маршрути – в цьому випадку роутер автоматично маршрутизує файл з ім'ям index в кореневу дерикторію, тож шлях pages/blog/index.js – буде аналогічним /blog
- Вкладені маршрути – якщо утворюється вкладена архітектура каталогів файли будуть автоматично маршрутизовані в той самий спосіб
- Сторінки із динамічними маршрутами – факично підтримують індекси, наприклад pages/posts/[id].js буде доступний за шляхом posts/1, posts/2 і так далі.

```
1  import Link from 'next/link'
2
3  function Home() {
4    return (
5      <ul>
6        <li>
7          <Link href="/">Home</Link>
8        </li>
9        <li>
10         <Link href="/about">About Us</Link>
11       </li>
12       <li>
13         <Link href="/blog/hello-world">Blog Post</Link>
14       </li>
15     </ul>
16   )
17 }
18
19 export default Home
```

Рис. 2.2. Роутінг на сороні клієнта

Роутінг в Next.js також дозволяє робити маршрутизацію на стороні клієнта переходячи між сторінками подібно односторінковому застосунку. Це стає можливим завдяки React компоненту Link, який використовується у спосіб показаний на Рисунку 2.2. Цей приклад показує використання декількох посилань, кожне з яких вказує на шлях до відомої сторінки.

Кожне з цих посилань буде актуалізовано за замовчанням для сторінок зі статичною генерацією. Відповідні дані для маршрутів сгенерованих сервером оновлюються тільки коли по посиланню хтось клікає.

Зв’язувати таким чином можна й динамічні шляхи. Подібний приклад наведено на Рисунку 2.3. та 2.4.

```
import Link from 'next/link'

function Posts({ posts }) {
  return (
    <ul>
      {posts.map((post) => (
        <li key={post.id}>
          <Link href={`/blog/${encodeURIComponent(post.slug)}`}>
            {post.title}
          </Link>
        </li>
      ))}
    </ul>
  )
}

export default Posts
```

Рис. 2.3. Зв’язування динамічних маршрутів.

При чому у другому випадку використовується об’єкт URL замість інтерполяції.

```
import Link from 'next/link'

function Posts({ posts }) {
  return (
    <ul>
      {posts.map((post) => (
        <li key={post.id}>
          <Link
            href={{
              pathname: '/blog/[slug]',
              query: { slug: post.slug },
            }}
          >
            {post.title}
          </Link>
        </li>
      ))}
    </ul>
  )
}

export default Posts
```

Рис. 2.4. Зв’язування динамічних маршрутів.

Останній тип роутінгу на який звернемо увагу це дрібний роутінг. Цей тип маршрутизації дозволяю змінити URL без застосування методів що підтягують дані, а саме:

- getServerSideProps
- getStaticProps
- getInitialProps

Для того щоб активувати дрібний роутінг треба обрати значення true для опції shallow, як показано на Рисунку 2.5. URL оновиться до /counter=10 та сторінка замінюватись не буде, зміниться тільки стан маршрута.

У дрібного роутінга є свої недоліки, наприклад, він працює тільки для змін URL на поточній сторінці. Так якщо мати іншу сторінку що запросить *pages/about.js* і в той же час буде виконано такий роутер:

router.push('/?counter=10', '/about?counter=10', { shallow: true }),
то оскільки ця сторінка новіша, стара сторінка буде вигружена та загружена нова і далі роутер буде чекати на оновлення даних не зважаючи на запит дрібного роутінгу.

```
import { useEffect } from 'react'
import { useRouter } from 'next/router'

// Current URL is '/'
function Page() {
  const router = useRouter()

  useEffect(() => {
    // Always do navigations after the first render
    router.push('/?counter=10', undefined, { shallow: true })
  }, [])

  useEffect(() => {
    // The counter changed!
  }, [router.query.counter])
}

export default Page
```

Рис. 2.5. Активування дрібного роутінгу.

Популярність Next.js зростає останнім часом, часто новістартапи шукають інструмент який буде зручно підлаштовувати під потреби SEO та споживачів. Само тому є сенс розглянути можливості та інструменти для впровадження автоматизації в проект написаний на Next.Js. тож для досягнення цілей даної роботи було використано застосунок, що реалізовує електронний маркетплейс на якому у користувачів є змога продавати та купляти різні товари.[6]

Загальний функціонал застосунку можна полити на 2 осноні частини. Перша це кабінет користувача, де він може створити собі магазин за якоюсь тематикою, а в магазині може створити продукти різних категорій. А друга це перегляд продуктів доступних для купівлі, які користувач може фільтрувати,

дивитись деталі на сторінці продукту та додавати його собі в кошик. Також доступна авторизація через сервіси Google.

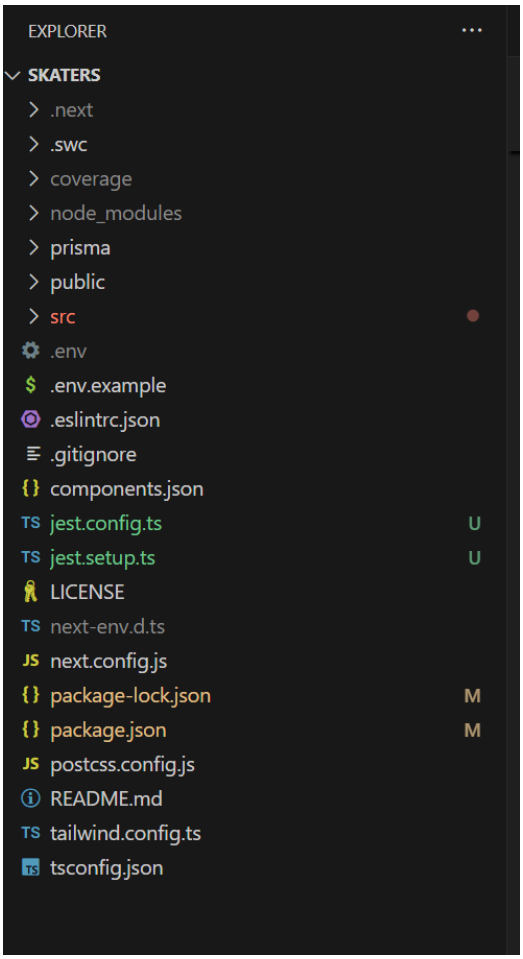
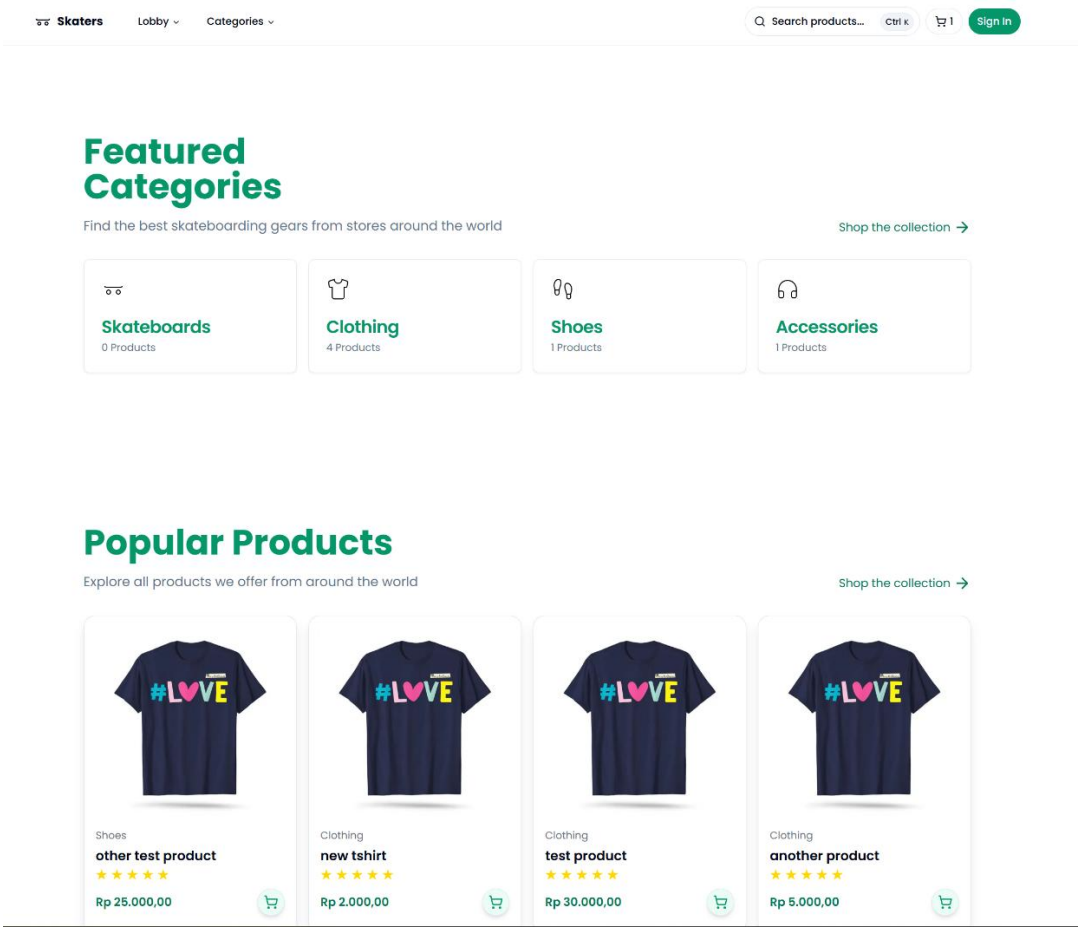


Рисунок 2.6. Структура папок в проекті Next.js

Архітектурно проект має набір компонентів таких як:

- Аутентифікація
- Картки продуктів
- Кошик
- Форми
- Галерея
- Модальні вікна
- Тощо.


Skaters
Lobby
Categories
Search products... Ctrl K
1
Sign In
Featured Categories
Find the best skateboarding gears from stores around the world
Shop the collection
Skateboards
0 Products
Clothing
4 Products
Shoes
1 Products
Accessories
1 Products
Popular Products
Explore all products we offer from around the world
Shop the collection
Shoes
other test product
Rp 25.000,00
Clothing
new tshirt
Rp 2.000,00
Clothing
test product
Rp 30.000,00
Clothing
another product
Rp 5.000,00

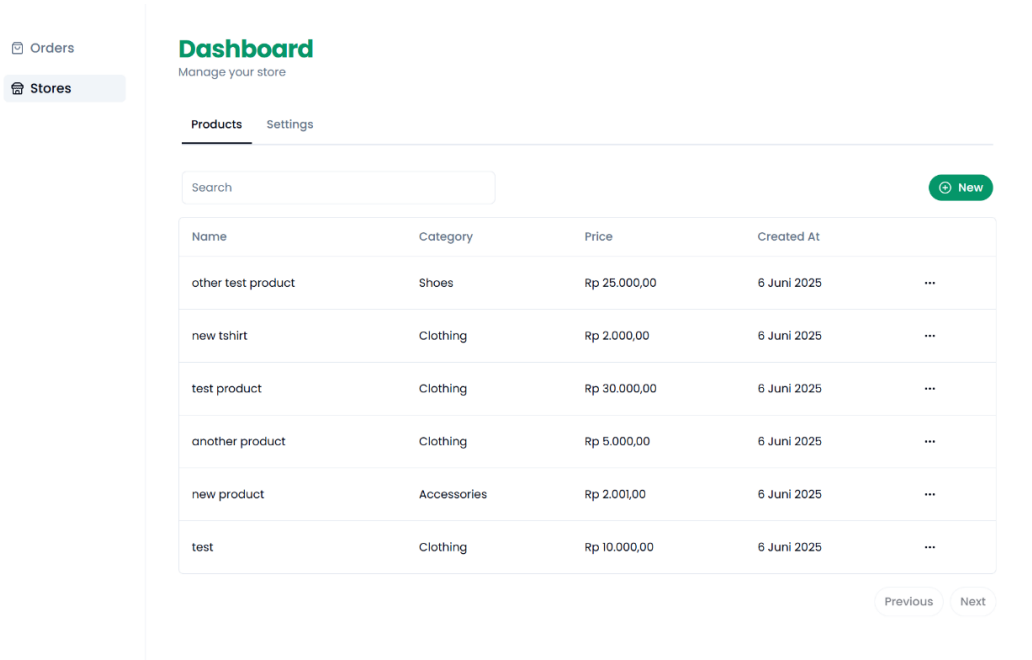

Orders
Stores
Dashboard
Manage your store
Products
Settings
Search
New
Name
Category
Price
Created At
other test product
Shoes
Rp 25.000,00
6 Juni 2025
new tshirt
Clothing
Rp 2.000,00
6 Juni 2025
test product
Clothing
Rp 30.000,00
6 Juni 2025
another product
Clothing
Rp 5.000,00
6 Juni 2025
new product
Accessories
Rp 2.001,00
6 Juni 2025
test
Clothing
Rp 10.000,00
6 Juni 2025
Previous
Next

Рисунок 2.7.(а)

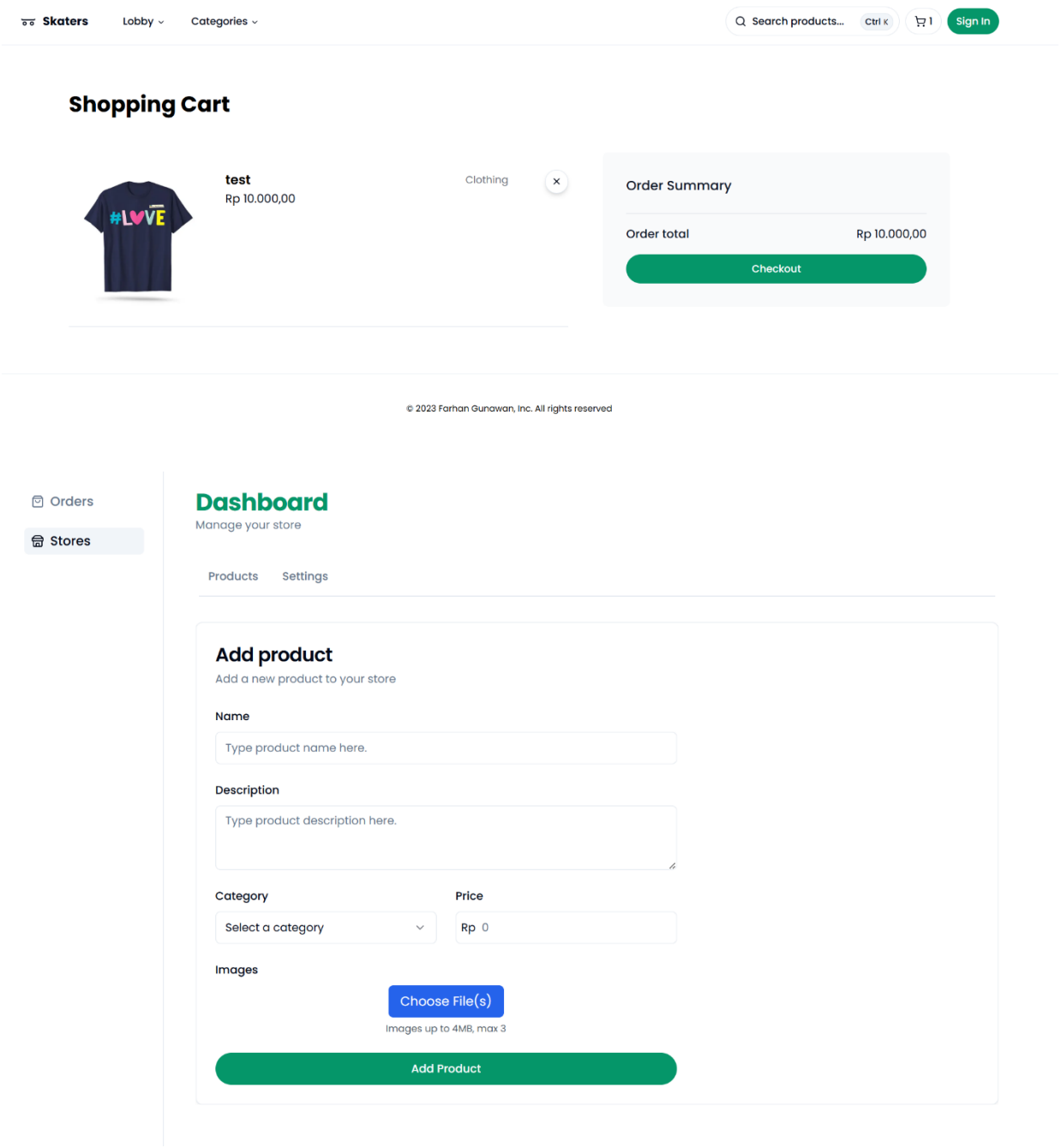


Рисунок 2.7.(б)

Така структура добре підходить для компонентного тестування, або unit-тестування, через те що можна поетапно покривати автоматизованими тестами різні компоненти незалежно один від одного, поступово збільшуючи покриття. Інтерфейс використаного веб застосунку можна побачити на Рисунку 2.7.(а, б).

2.1.3 Інструменти для створення unit-тестів

Одним з найпопулярніших фреймворків для написання unit-тестів є Jest. Свою популярність цей фреймворк здобув завдяки своїй зручності. Jest має

велику кількість інструментів необхідних для створення автоматизованих тестів, це і зручна організація тестів, і вбудовані інструменти для оцінки покриття коду, і простий мокінг (створення «заглушок» для інкапсуляції компонента під час виконання тесту).

Також Jest дуже просто інтегрується з React, а відповідно і з Next.js. Взагалі-то його можна встановити одразу під час створення проекту з допомогою цієї команди:

npx create-next-app@latest --example with-jest with-jest-app

Після встановлення Jest необхідно виконати команду npm init jest@latest після чого буде створено файл конфігурації. В цьому файлі можна налаштувати параметри для інструменту оцінки покриття коду тестами, вибрати тип тестового середовища.

Jest тако надає багато інструментів для організації тестів, за допомогю функції describre() можна створювати тестові набори, а в функції test() описуються кроки які необхідно виконати під час одного тесту. Використовуючи beforeEach()/afterEach() можна створювати необхідні умови для виконання тесту і чистити наслідки його роборти.

Також Jest пропонує набір імітаційних (або mock) функцій за допомогою яких можна заміняти реальну взаємодію за іншими компонентами, сервісами, API чи базою, що дозволяє зробити автоматичні тести значно надійніше і швидше.

2.1.4 Бібліотека React testing library

Ще одним дуже цікавим інструменотм для розробки та підтримки тестів для React компонентів є бібліотека React testing library. Ця бібліотека має за мету зробити тести біль шезалежними від реалізації компонентів, і дати можливість розробникам зосередитися на тестах, які дають впевненість у тому, що застосунок працює правильно. Крім того,

ця бібліотека має всі інструменти для того щоб зробити репозиторій тестів легко підтримуваним в довгостроковій перспективі. Це в свою чергу

може бути корисним для рефакторінга коду, бо він стане менш резикованим та не буде гальмувати розробку чере тести що ввесь час падають.

React Testing Library (RTL) - це легка бібліотека для тестування React компонентів. Вона надає прості утиліти поверх react-dom та react-dom/test-utils.

Основними її перевагами є наступні:

- замість того, щоб працювати з інстансами React-компонентів тести працюють з реальними DOM-вузлами.
- RTL надає утиліти для пошуку елементів так, як це робив би користувач:
    - Дозволяє знаходити елементи форми за їхнім текстом мітки (label text);
    - Дозволяє знаходити кнопки та посилання за текстом;

Далі подивимось як налаштувати цю бібліотеку. Взагалі то вона не потребує налаштування для її використання, але для деяких задач воно все ж таки може знадобитись.

Наприклад, для того щоб використати користувацькийфайл з тестами разом з Jest можна налаштування показані на Рисунку 2.8.

Це дозволить імпортувати всі файли .js з каталога test-utils без додавання «../»[13]

В цілому основними методами які надає дана бібліотека це

- render const {/* */} = render(Component)
- величезна кількість різноманітних запитів(queries)
- асинхронні функції типу waitFor, та інші вейтери
- а також івенти(events), наприклад fireEvent, який є викликає DOM event: fireEvent(node, event)

my-component.test.js

```
- import { render, fireEvent } from '../test-utils';
+ import { render, fireEvent } from 'test-utils';
```

jest.config.js

```
module.exports = {
  moduleDirectories: [
    'node_modules',
+   // add the directory with the test-utils.js file, for example:
+   'utils', // a utility folder
+    __dirname, // the root directory
  ],
  // ... other options ...
}
```

Рис. 2.8. Спрощений імпорт бібліотек.

3. РЕАЛІЗАЦІЯ ЗАСОБУ СТВОРЕННЯ UNIT-ТЕСТІВ

3.1.1 Налаштування фреймворка JEST та розробка тест-кейсів.

Для реалізації завдання спочатку було необхідно налаштувати інструменти для створення unit-тестів. Для цього в основний конфігураційний файл проекта package.json було внесено зміни, а саме додано відповідні залежності та скріпти для запуску тестів is Jest(рис. 3.1.).

Рис 3.1. Додавання Jest залежностей

Після встановлення та налаштування Jest проаналізуємо проект. Оскільки проект складється з окремих компонентів спочатку було треба виділити які з компонентів є найбільш важливими і покрити тестами їх.

Для цього спочатку визначемо, що саме в нашему випадку будемо вважати юнітом для тестування. Часто одиницею для тестування вважається окемий клас, або метод, але найкращою практикою є покривати тестами саме логічні юніти, тобточастинки функціоналу, які можна вважати окремими.

Для того, щоб краще це зрозуміти побудуємо use case діаграму для користувачів застосунку.

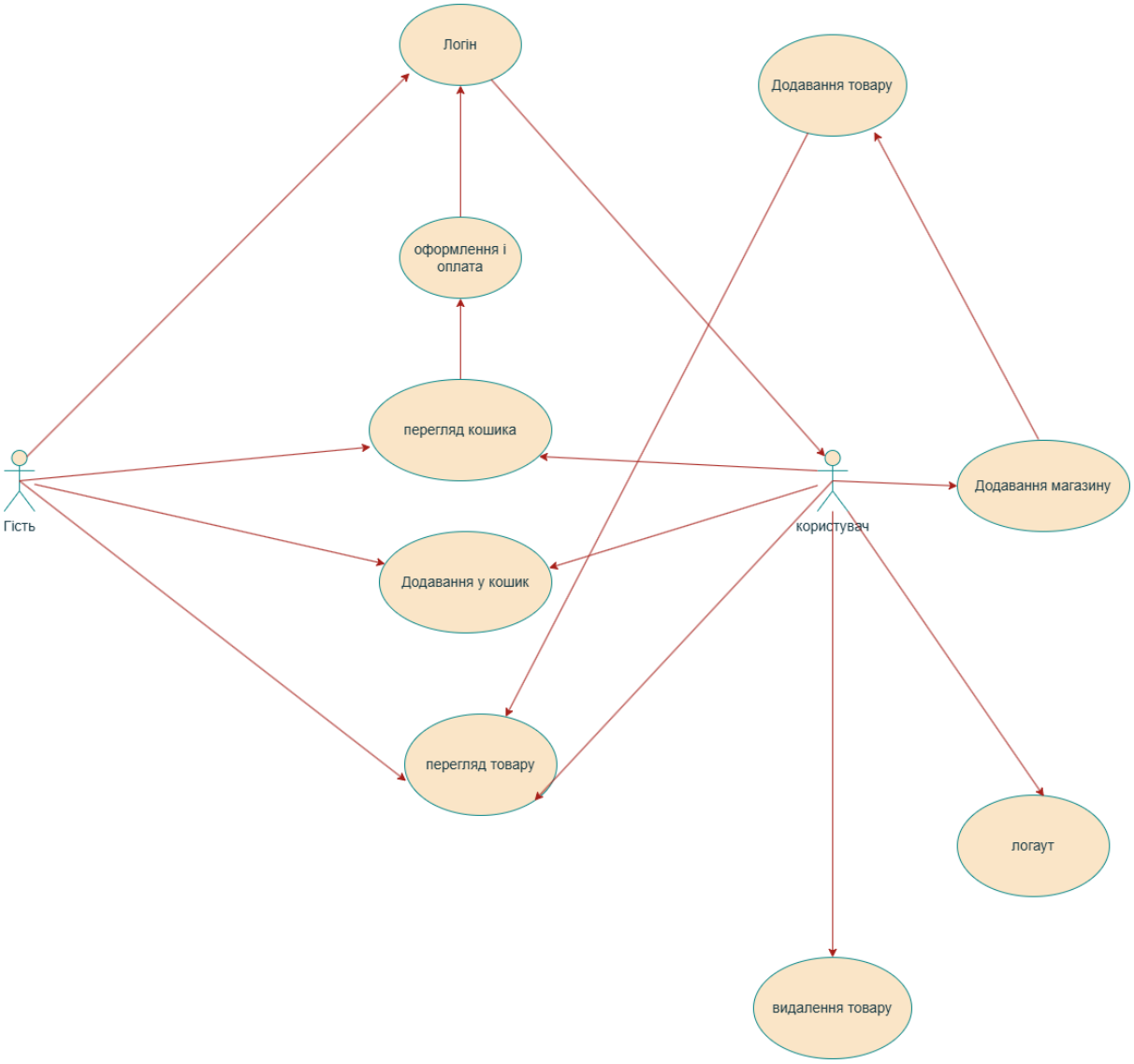


Рисунок 3.2. Діаграма варіантів використання застосунку.

Можемо побачити, що найбільш затребуваною частиною є додавання магазину та товару, саме цей функціонал додає контент.

Логін та логаут також важливі, але в цій версії застосунку реалізована можливість авторизуватись тільки через аккаунт Google, тож там не так багато кейсів для тестування і оскільки це взаємодія з зовнішнім сервісом, то ці тести краще віднести до середнього рівня піраміди.

Далі розробимо набір тестових випадків для форм створення магазину та товару.

Таблиця. 3.1. Набір тест кейсів для ProductForm

Компонент	**ID**	**Назва тест-кейсу**	**Очікувана поведінка**
UpdateProductForm	TC-001	Рендер усіх полів	Форма відображає всі обов’язкові поля
	TC-002	Відображення кнопки сабміту	Кнопка 'Add Product' є у DOM
	TC-003	Сабміт з порожніми полями	Показуються повідомлення про помилки
	TC-004	Сабміт без категорії	Показується помилка 'Category is required'
	TC-005	Сабміт без зображень	Показується помилка 'Images is required'
	TC-006	Некоректне значення Price	Значення не проходить валідацію (тільки числа)
	TC-007	Успішний сабміт форми	Дані відправляються, показується toast, відбувається redirect
AddProductForm	TC-008	Toast успішного створення	toast.success викликається після сабміту
	TC-009	Редирект після створення продукту	router.push викликається з правильним URL
	TC-010	Помилка з axios	toast.error викликається з повідомленням помилки
	TC-011	Некоректна відповідь сервера	Форма не зависає, помилка обробляється
	TC-012	Завантаження зображення	Зображення додається до масиву
	TC-013	Видалення зображення	Зображення видаляється з масиву
	TC-014	Кнопка дизейблиться при сабміті	Кнопка недоступна під час сабміту
	TC-015	Поля disabled при isLoading	Інпути неактивні під час сабміту

Таблиця 3.2. Набір тест кейсів для StoreForm

Компонент	ID	Назва тест-кейсу	Очікувана поведінка
AddStoreForm	TC-AS001	Рендеринг форми	Форма та всі поля відображаються у DOM
	TC-AS002	Сабміт без значень	Показується помилка 'required'
	TC-AS003	Успішний сабміт	toast.success викликається, редирект після створення
	TC-AS004	Сабміт з невалідним полем	Zod валідація показує помилку
	TC-AS005	Дизейбл елементів при сабміті	Інпути та кнопка недоступні при isLoading
FileUpload	TC-FU001	Рендер без зображень	Кнопка Upload відображається
	TC-FU002	Видалення зображення	onRemove викликається з URL
	TC-FU003	Рендер з кількома зображеннями	Всі зображення відображаються як прев'ю
	TC-FU004	Помилка при завантаженні	toast.error викликається
UpdateStoreForm	TC-US001	Рендер з даними	Початкові значення store відображаються у формі
	TC-US002	Оновлення успішне	toast.success та axios.patch викликаються
	TC-US003	Сабміт без імені	Відображається повідомлення про помилку
	TC-US004	Модальне вікно видалення	Клік по delete відкриває підтвердження
	TC-US005	Помилка видалення	toast.error викликається при axios.delete
AddStoreForm	TC-AS001	Рендеринг форми	Форма та всі поля відображаються у DOM

3.1.2 Додавання unit-тестів до проекту.

Для того щоб почати реалізовувати описані в таблиці 3.1.2 тести в папку Components в проекті для кожного з компонентів створюємо відповідний файл

*.test.tsx в якому і будуть зберігатись unit-тести для цього компоненту, як показано на Рис. 3.3.

Також для збільшення стабільності проходження тестів додаємо спеціальний атрибут data-testid до відповідних DOM-елементів для того щоб використовувати саме цей атрибут з коду теста і уникнути залежності від змін атрибутів або тексту в елементах при внесенні змін в код проекта під час розробки(Рис 3.4.).

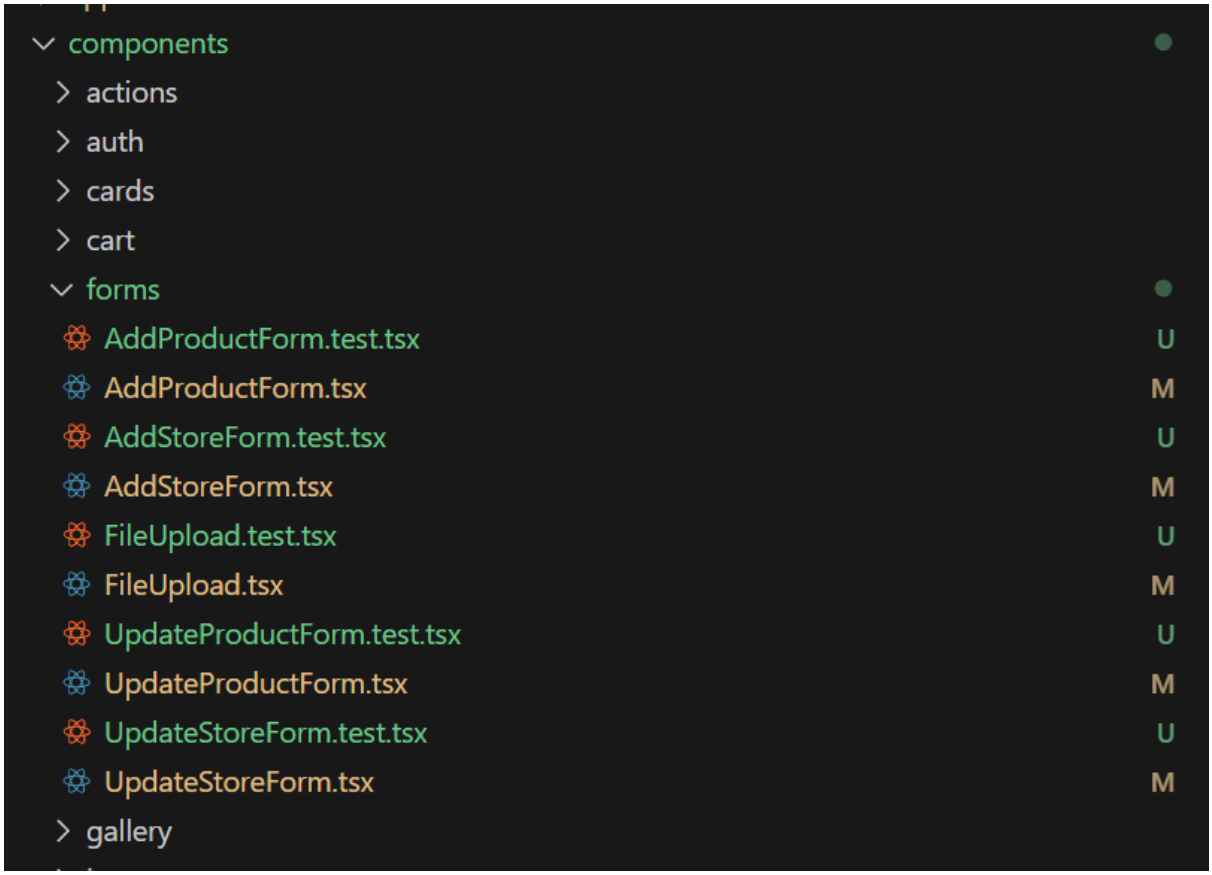


Рисунок 3.3. Архітектура каталогів Next.js проекта

При створенні юніт тестів варто розбивати їх на окремі набори, для того щоб мати гнучкість запускати тільки ті набори тестів які потрібно, наприклад якщо необхідно перевірити я тільки один конкретний компонент.

Рисунок 3.4. Додавання спецыального атрибуту

Для того щоб це реалізувати використовуємо вбудовану функцію describe:

describe('AddProductForm Test Suite', () => {

...

}

В середині describe пишемо окремі тести використовуючи ще одну вбудовану функцію it:

it('TC-001: renders all fields and submit button', () => {

…

})

В цілому тестовий набір буде мати вигляд я на Рис 3.5. Також звернемо увагу на те що слід використовувати конструкцію beforeEach/afterEach для того щоб винести з самих тестів код що повторюється. Також в якості локаторів DOM-елементів використовуємо атрибут data-testid із допомогою вбудованого метода getByTestId з модуля screen бібліотеки React Testing Library.

3.1.3 Створення звітів та аналіз покриття.

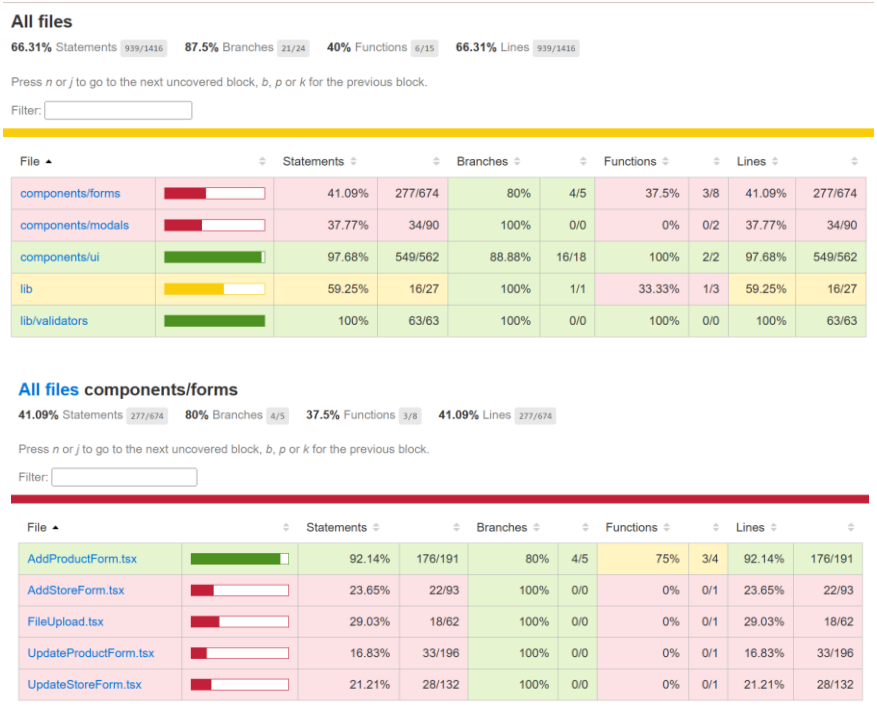
Фреймворк Jest має інформативний вбудований інструмент для генерації звітів з допомогою якого можна проаналізувати покриття кода тестами, як в цілому так і по окремим компонентам. Для того щоб Jest сформував такий звіт треба запустити тести з параметро «--coverage». В такому випадку під час виконання тестів Jest проаналізує код проекта і сформує звіт який можна

відкрити як локальну html сторінку. Знаходиться ця сторінка разом з усіма даними в папці coverage в корінному каталозі проекту. Як виглядає подібний звіт можна побачити на рисунку 3.6.(а, б). Також можна імпортувати дані звіту і відкрити його будь яким іншим інструментом для формування звітів.

```
describe('AddProductForm Test Suite', () => {
  const pushMock = jest.fn()
  const toastError = toast.error as jest.Mock
  const toastSuccess = toast.success as jest.Mock

  beforeEach(() => {
    jest.clearAllMocks()
    ;(useRouter as jest.Mock).mockReturnValue({ push: pushMock })
    ;(useParams as jest.Mock).mockReturnValue({ storeId: 'test-store' })
  })

  it('TC-001: renders all fields and submit button', () => {
    render(<AddProductForm />)
    expect(screen.getByTestId('name-input')).toBeInTheDocument()
    expect(screen.getByTestId('description-input')).toBeInTheDocument()
    expect(screen.getByTestId('category-select')).toBeInTheDocument()
    expect(screen.getByTestId('price-input')).toBeInTheDocument()
    expect(screen.getByTestId('submit-button')).toBeInTheDocument()
  })

  it('TC-003: validation error on empty submit', async () => {
    render(<AddProductForm />)
    fireEvent.click(screen.getByTestId('submit-button'))
    await waitFor(() => {
      expect(screen.getAllByText(/required/i).length).toBeGreaterThan(0)
    })
  })

  it('TC-012: simulates image upload', () => {
    render(<AddProductForm />)
    fireEvent.click(screen.getByTestId('upload-btn'))
    expect(screen.getByTestId('mock-file-upload')).toBeInTheDocument()
  })

  it('TC-013: simulates image removal', () => {
    render(<AddProductForm />)
    fireEvent.click(screen.getByTestId('upload-btn'))
    fireEvent.click(screen.getByTestId('remove-btn'))
  })

})
```

Рисунок 3.5. Приклад організації набора unit-тестів

Також після кожного прогону тестів Jest видає результати в консоль у вигляді таблички. Про тести що не пройшли також можна знайти інформацію в консолі. Зокрема на рисунку 3.7. справа можна побачити, що тест впав через те що елемент не отримав атрибут «disabled» в момент сабміта форми, що є багом в коді.



All files

66.31% Statements 939/1416 **87.5%** Branches 21/24 **40%** Functions 6/15 **66.31%** Lines 939/1416

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
components/forms	41.09%	277/674	80%	4/5	37.5%	3/8	41.09%	277/674
components/modals	37.77%	34/90	100%	0/0	0%	0/2	37.77%	34/90
components/ui	97.68%	549/562	88.88%	16/18	100%	2/2	97.68%	549/562
lib	59.25%	16/27	100%	1/1	33.33%	1/3	59.25%	16/27
lib/validators	100%	63/63	100%	0/0	100%	0/0	100%	63/63

All files components/forms

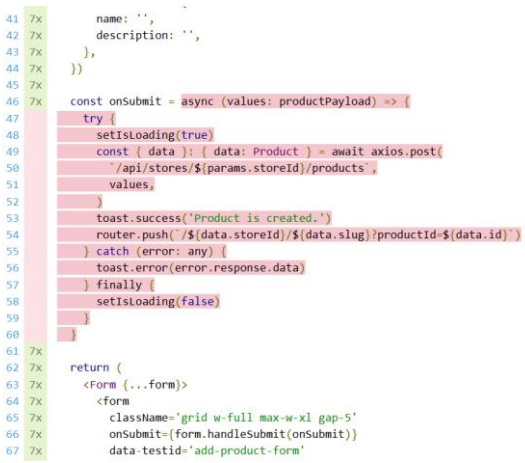
41.09% Statements 277/674 **80%** Branches 4/5 **37.5%** Functions 3/8 **41.09%** Lines 277/674

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

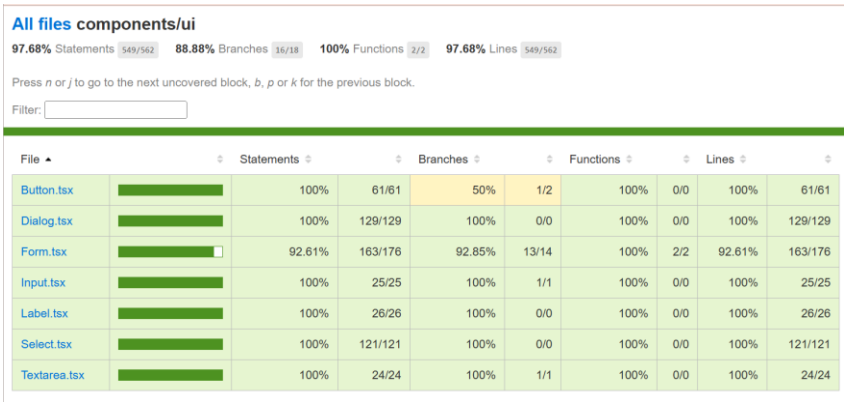
Filter:

File	Statements		Branches		Functions		Lines	
AddProductForm.tsx	92.14%	176/191	80%	4/5	75%	3/4	92.14%	176/191
AddStoreForm.tsx	23.65%	22/93	100%	0/0	0%	0/1	23.65%	22/93
FileUpload.tsx	29.03%	18/62	100%	0/0	0%	0/1	29.03%	18/62
UpdateProductForm.tsx	16.83%	33/196	100%	0/0	0%	0/1	16.83%	33/196
UpdateStoreForm.tsx	21.21%	28/132	100%	0/0	0%	0/1	21.21%	28/132

Рисунок 3.6.(а). Покриття коду



```
41 7x       name: '',
42 7x       description: '',
43 7x     },
44 7x   })
45 7x
46 7x   const onSubmit = async (values: productPayload) => {
47        try {
48          setIsLoading(true)
49          const { data }: { data: Product } = await axios.post(
50            `/api/stores/${params.storeId}/products`,
51            values,
52          )
53          toast.success('Product is created.')
54          router.push(`/${data.storeId}/${data.slug}?productId=${data.id}`)
55        } catch (error: any) {
56          toast.error(error.response.data)
57        } finally {
58          setIsLoading(false)
59        }
60      }
61 7x
62 7x   return (
63 7x     <Form {...form}>
64 7x       <form
65 7x         className='grid w-full max-w-xl gap-5'
66 7x         onSubmit={form.handleSubmit(onSubmit)}
67 7x         data-testid='add-product-form'
```



All files components/ui

97.68% Statements 549/562 **88.88%** Branches 16/18 **100%** Functions 2/2 **97.68%** Lines 549/562

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
Button.tsx	100%	61/61	50%	1/2	100%	0/0	100%	61/61
Dialog.tsx	100%	129/129	100%	0/0	100%	0/0	100%	129/129
Form.tsx	92.61%	163/176	92.85%	13/14	100%	2/2	92.61%	163/176
Input.tsx	100%	25/25	100%	1/1	100%	0/0	100%	25/25
Label.tsx	100%	26/26	100%	0/0	100%	0/0	100%	26/26
Select.tsx	100%	121/121	100%	0/0	100%	0/0	100%	121/121
Textarea.tsx	100%	24/24	100%	1/1	100%	0/0	100%	24/24

Рисунок 3.6.(б). Покриття коду.

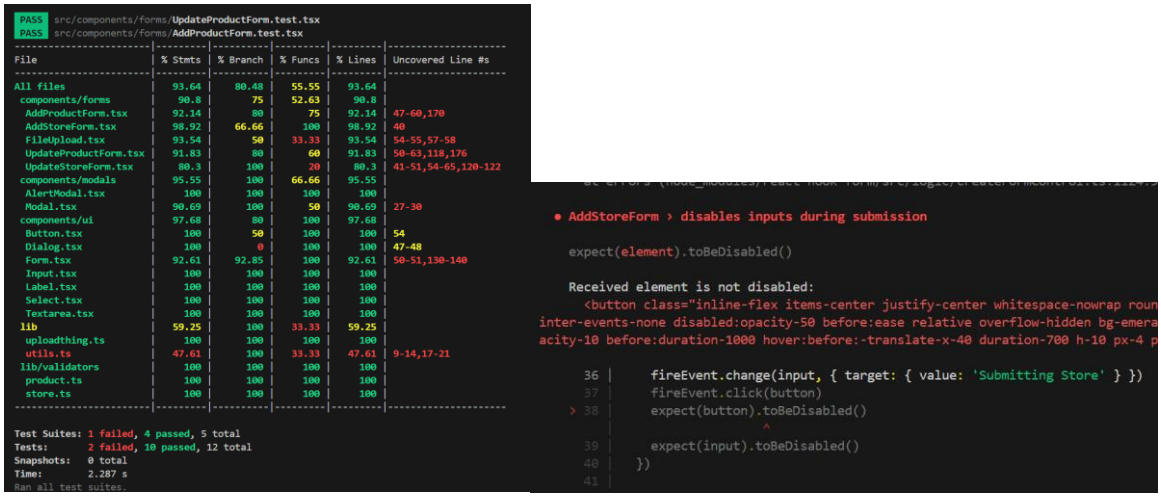


Рисунок 3.7. Результат роботи Unit тестів

3.1.4 Застосування методики CI/CD.

Continuous Integration (CI) та Continuous Deployment (CD) є дуже важливою частиною процесу розробки в наш час. Цей підхід дозволяє зняти величезну кількість напруги з команди та значно скоротити час на виконання релізу в продакшен. До того ж правильно використаний цей підхід дозволяє значно зменшити ризики під час релізу. Таким чином CI/CD дозволяє значно прискорити процес розробки, що є дуже важливим для сучасного ринку веб-застосунків. Фреймворк Jest підтримує інтеграцію CI/CD, що є безперечною перевагою.

Для того щоб налаштувати Continuous Integration треба виконати декілька кроків.

По-перше треба мати встановлений та ініціалізований Jest, щоб в проекті був конфігураційний файл jest.config.js Далі необхідно створити наступну архітектуру папок: .gihub/workflows/ та покласти туди файл main.yml. І в цей файл додати набір дій для github (Рис. 3.8). До речі те саме можна зробити і з допомогою інші інструментів, наприклад Travis CI, Jenkins, Gitlab, Bitbucket тощо.

Далі треба додати пункт «repository» до package.json:

"repository": {

"type": "git",

"url": "https://github.com/yourusername/<your-repo>.git"

},

У файлі main.yml відбувають основні налаштування. Так «on» є основною подією яка викликає workflow. А jobs це ті дії які будуть виконуватись під час workflow. Також можна задати операційну систему в якій будуть відбуватись дії. На пикладі з Рисунку 3.1.4.1 будуть виконані такі кроки: Github клонує гілку проекту на віртуальну машину в хмарі та запустить команди npm install та npm test, що відповідно запустить тести на виконання, а результат можна буде побачити в консолі на вкладці actions на github.

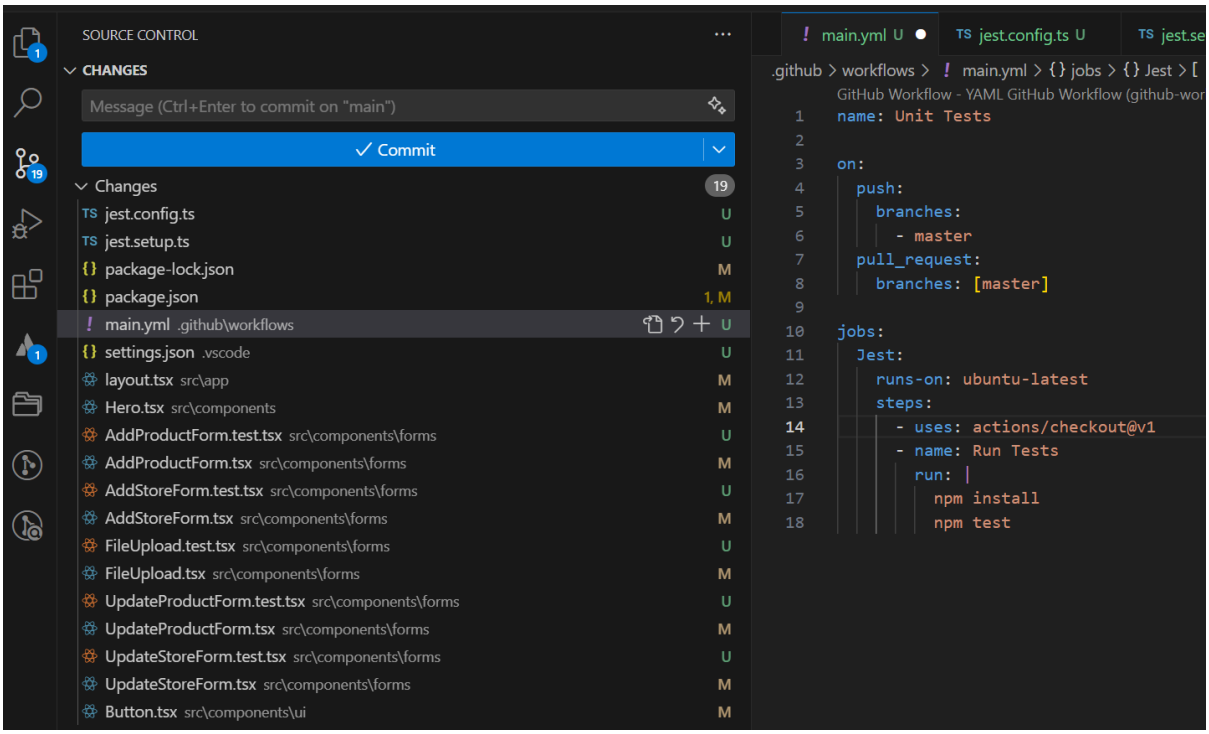


Рис. 3.8. Налаштування CI/CD для Github.

3.1.5 Додавання інтеграції з TMS.

В попередніх розділах ми вже сформували досить потужний засіб для використання автоматизованих тестів, але ще є простір для вдосконалення. Для того щоб фактично перетворити автоматичні тести на частину документації проекта можна інтегрувати в проект ще систему управління

тестами або TMS(Tests Management System). Далі розглянемо як це можна зробити на прикладі інструмента Qase.io.

Qase це велика платформа створена для QA та покликана допомагати організовувати всі тести на проекті як мануальні так і автоматизовані. Вона дозволяє управляти тест кейсами, багрепортами, тест планами, виконанням тестів та багато чого іншого. Також ця система має можливість глибокої інтеграції наприклад з Jira.

Так само можлива інтеграція з Jest. Qase має набір вбудованих застосунків серед яких є і Jest. Тож для інтергації цих інструментів необхідно встановити застосунок Jest, активувати його та отримати токен доступу який дозволить використовувати Qase API. Тож можна буде отримати:

• автоматичне додавання багів до Jira

• через панель управління Qase запускати прогони автоматизованих тестів

• додавати автоматизовані тести що лежать в проекті в загальний репозиторій з тест-кейсами, чеклістами і так далі

• отримувати сповіщення про запуск або закінчення виконання автоматизованих тестів, в тому числі і unit-тестів. [8]

Для того щоб Jest відсилав результативиконання тесті ра реєстрував тестові прогони в Qase треба встановити jest-qase-reporter через node packet manager:

npm install --save-dev jest-qase-reporter

Щоб почати використовувати цей репортер необхідно додати в файл jest.config.ts конфігурацію для цього репортера:

3.1.6 Конфігурація репортера для Jest.

Токе треба підставити той який був згенерований в застосунку Jest, а код проекта можна побачити в URL відкривши відповідний проект в

https://app.qase.io/. Для першого проекту це скоріше всього буде https://app.qase.io/project/DEMO.

Тепер для того щоб відправляти інформацію до Qase необхідно оновити самі тести а саме імпортувати jest-qase-reporter

Const { qase } = require('jest-qase-reporter/jest');

А також переробити тест так як зроблено на рисунку 3.9

```
41
42  describe('AddProductForm Test Suite', () => {
43    const pushMock = jest.fn()
44    const toastError = toast.error as jest.Mock
45    const toastSuccess = toast.success as jest.Mock
46
47    beforeEach(() => {
48      jest.clearAllMocks()
49      ;(useRouter as jest.Mock).mockReturnValue({ push: pushMock })
50      ;(useParams as jest.Mock).mockReturnValue({ storeId: 'test-store' })
51    })
52
53    it(qase(11, 'TC-001: renders all fields and submit button'), () => {
54      render(<AddProductForm />)
55      expect(screen.getByTestId('name-input')).toBeInTheDocument()
56      expect(screen.getByTestId('description-input')).toBeInTheDocument()
57      expect(screen.getByTestId('category-select')).toBeInTheDocument()
58      expect(screen.getByTestId('price-input')).toBeInTheDocument()
59      expect(screen.getByTestId('submit-button')).toBeInTheDocument()
60    })
61
```

Рис. 3.9. оновлення теста для роботи з Qase.

Після цього можна запустити виконання тестового набору із Jest і спостерігати за його виконанням в проекті Qase. Репортер Jest може створювати автоматично там тест кейси та тестові набори проганяючи тести. Для доступу к параметрам записів в Qase можна використати наступні атрибути:

- qase.title - встановити назву тесту
- qase.fields - задати поля тест кейсу
- qase.suite - створити тестовий набір
- qase.comment - створити коменар до тест кейсу
- qase.parameters - задати параметри для тест кейсу

- qase.groupParameters - задати набір параметрів параметри для тест кейсу
- qase.ignore - ігнорувати цей тест кейс в Qase, але він все одно буде виконуватись, хоча результати не підуть в Qase.
- qase.step - додати кток в репозиторій кроків для тест кейсів
- qase.attach – приєднати файл до тест кейсу

ВИСНОВКИ

У даній роботі було розроблено засіб для автоматичного тестування, що дозволяє інтегрувати автоматичні тести в існуючий проект веб-застосунку, відслідковувати покриття тестами, виконання тестів, а також додавати нові тести.

Було оглянуто сучасний стан інструментів розробки веб-застосунків, значна доля яких будується з допомогою фреймворків, що мають вбудовані набори компонентів з яких, як з блоків збирається веб-застосунок. Як раз для тестування таких застосунків дуже зручно використовувати unit-тести, якими можна покрити більшість функціоналу. Тож є дуже актуально працювати в напрямку створення комплексного засобу, який буде включати в себе інструменти що дозволять максимально автоматизувати процес перевірки якості та значно прискорить процес розробки.

В роботі було використано набір технологій таких як фреймворк для розробки автоматизованих тестів, генерації звітів та аналізу покриття Jest, бібліотеку React Testing Library, та фреймворк для розробки веб застосунків Next.js, а також застосовано методологію розробки Continous Integration/ Continous Delivery(CI/CD). Показано інтеграцію процесу автоматичного тестування у CI/CD пайплайн за допомогою GitHub Actions.

Було розглянуто можливості інтеграції з системою управління тестами Qase. І ця інтеграція має високий потенціал через можливість через Jest фактично керувати репозиторієм тестів в Qase. Також це дає можливість дуже зручно розмежувати автоматизовані тести і мануальні, що дозволить біль ефективно використовувати ресурси команди розробки.

Ці технології в комплексі показали себе як потужний засіб для створення unit-тестів та автоматизованих тестів взагалі. Застосування такого засобу дозволить якісно підвищити свій рівень будь якій команді розробки та значно підвищити власну продуктивність.

Розроблені з допомогою цього засобу тести показали спроможність знаходити недоліки в реалізації компонентів проекту, а повноцінна інтеграція з системою управління тестами дозволяє зручно збільшувати об’єм тестів та надає можливість тісно інтегрувати їх з проектною документацією.

В якості шляхів вдосконалення роботи можна визначити все ж таки завершену інтеграцію Jest та Qase. А також розгляд інструментів для автоматизації тестування заснованих на технології Штучного Інтелекту, а саме генерації тест кейсів і unit-тестів, а також тестової документації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Andrew Pollner (Chair), Péter Földházi, Patrick Quilter, Gergely Ágnecz, László Szikszai, ISTQB, Test Automation Strategy v1.0 syllabus - Режим доступу: https://www.istqb.org/wp-content/uploads/2024/11/ISTQB_CT-TAS_Syllabus_v1.0.pdf (дата звернення: 14.05.2025)

2. Режим доступу: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/ (дата звернення: 10.05.2025)

3. Режим доступу: https://w3techs.com/technologies/details/js-nextjs (дата звернення: 11.05.2025)

4. Vladimir Khorikov, Unit Testing: Principles, Practices, and Patterns, Manning, 2020, 304с.

5. Roy Osherove, Vladimir Khorikov, The Art of Unit Testing, Manning, 2024, 259с.

6. Режим доступу: https://nextjs.org/docs (дата звернення: 05.05.2025)

7. Режим доступу: https://jestjs.io/docs/getting-started (дата звернення: 05.04.2025)

8. Режим доступу: https://docs.qase.io/automation/reporters/javascript/jest (дата звернення: 25.05.2025)

9. Sambamurthy M. Test Automation Engineering Handbook: Learn and implement techniques for building robust test automation frameworks. Packt Publishing Ltd, 2023. 276 с. ISBN 9781804615492.

10. Gerard Meszaros, xUnit Test Patterns: Refactoring Test Code Pearson Education, May 21, 2007, 944 с

11. Patel P. Testing Pyramid: A Successful Approach to End to End testing.. Alphabin.co. blog. 16.08.2024. URL: https://www.alphabin.co/blog/testing-pyramid (дата звернення: 30.04.2025).

12. Cohn M. The Forgotten Layer of the Test Automation Pyramid. www.mountaingoatsoftware.com. blog. URL: https://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid (дата звернення: 30.04.2025).

13. Режим доступу: https://testing-library.com/docs/react-testing-library/setup(дата звернення: 27.05.2025)