Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій
(повна назва)

Кафедра Інформаційно-вимірювальних технологій
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Методика тестування інформаційної системи служби доставки їжі
(тема)

Виконав:
здобувач 2 року навчання,
групи ЗЯм-23-1
Дегтярьов І.Ю.
(прізвище, ініціали)

Спеціальність 175 – Інформаційно-вимірювальні технології
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Забезпечення якості
(повна назва освітньої програми)

Керівник доц. Запорожець О.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Захаров І.П.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій
(повна назва)

Кафедра Інформаційно-вимірювальних технологій

Рівень вищої освіти другий (магістерський)

Спеціальність 175 – Інформаційно-вимірювальні технології
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Забезпечення якості
(повна назва)

ЗАТВЕРДЖУЮ

Зав. кафедри _____
(підпис)

" _____ "_____2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Дегтярьову Івану Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методика тестування інформаційної системи служби доставки їжі
затверджена наказом по університету від "12" листопада 2024 р. № 1202Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 10 січня 2025р.
3. Вихідні дані до роботи (проекту) Розробити компоненти інформаційної системи служби доставки їжі, яка повинна являти собою веб-сайт з інтерфейсом доступу до бази даних, протестувавши систему за допомогою функціонального тестування, тестування інтерфейсу користувача (UI), тестування продуктивності, інтеграційного тестування, тестування бази даних, тестування безпеки та регресійного тестування, використовуючи ОС Microsoft Windows 11, MongoDB, Visual Studio Code, Insomnia на IBM-сумісному ПК з ЦП Intel Core i5 та вище.
4. Перелік питань, що потрібно опрацювати в роботі
4.1 Вступ 4.2 Аналіз предметної галузі: інформаційні системи, їх поняття, види та особливості 4.3 Основи тестування веб-сайтіві 4.4 Типи тестування: ручне та автоматизоване тестування 4.Методи тестування: чорна скринька, біла скринька та тестування за допомогою сценаріїв 4.6 Життєвий цикл розробки та тестування інформаційних систем 4.7 Ітераційний підхід до розробки та тестування: використання Agile, Scrum або Waterfall у процесі розробки та тестування ІС 4.8 Роль тестування на різних етапах життєвого циклу 4.9 Особливості інформаційної системи служби доставки їжі 4.10 Архітектура та функціональні можливості інформаційної системи 4.11 Основні бізнес-процеси служби доставки їжі 4.12 Вимоги до тестування інформаційної системи 4.13 Можливі ризики і проблеми при впровадженні системи 4.14 Тестування інформаційної системи 4.15 Методи тестування 4.16 Ручне тестування 4.17 Автоматизоване тестування 4.18 Тестування роботи БД 4.19 Висновки 4.20 Перелік джерел посилань

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) _____
Слайди презентації кваліфікаційної роботи.

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів атестаційної роботи	Термін виконання етапів роботи	Примітка
1.	Отримання завдання атестаційної роботи	25.11.2024	Виконано
2.	Аналіз завдання, літератури та аналогів з теми	25.11.2024-	Виконано
	атестаційної роботи	30.11.2024	
3.	Вибір засобів для розробки технічних вимог до	01.12.2024-	Виконано
	програми	05.12.2024	
4.	Структурне проектування	06.12.2024-	Виконано
		10.12.2024	
5.	Вибір середовища розробки програми	11.12.2024	Виконано
6.	Розробка програми	12.12.2024-	Виконано
		20.12.2024	
7.	Тестування програми	21.12.2024 -	Виконано
		31.12.24	
8.	Оформлення пояснювальної записки та	01.01.2025-	Виконано
	програмної документації	04.01.2025	
9.	Оформлення графічної частини та презентаційних	за 5 днів	Виконано
	матеріалів комп'ютерного захисту		
10.	Представлення на рецензування	за 3 дні	Виконано
11.	Представлення атестаційної роботи в ЕК	за 2 дні	Виконано

Дата видачі завдання 25 листопада 2024 р.

Здобувач _____ Дегтярьов І.Ю.
(підпис)

Керівник роботи _____ доц. Запорожець О.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи містить: 98 стор., 14 рис., 2 табл., 1 додаток, 27 джерел.

БАЗА ДАНИХ, СИСТЕМА УПРАВЛІННЯ БАЗАМИ ДАНИХ, MONGODB, МАСШТАБУВАННЯ, REACT, JAVASCRIPT, ТЕСТУВАННЯ, JEST, КОНТРОЛЕРИ, ІНДЕКСИ, ІНТЕРФЕЙС ДОСТУПУ, СИСТЕМА ОБЛІКУ, ГЕОКОДУВАННЯ

Об'єктом досліджень кваліфікаційної роботи є інформаційні технології та методи створення та тестування серверної та клієнтської частини інформаційної системи служби доставки їжі.

Предметом досліджень кваліфікаційної роботи є методи тестування інформаційної системи служби доставки їжі.

Метою досліджень є розробка методики тестування інформаційних систем доставки замовлень їжі.

Область застосування – автоматизація тестування інформаційних систем.

Результати роботи – протестована як у клієнтській так і у серверній частинах інформаційна система доставки замовлень їжі, у якій є можливість відстеження шляху кур’єра на мапі.

ABSTRACT

The explanatory note of the certification work contains: 98 p., 14 pic., 2 table, 1 application, 27 source.

DATABASE, DATABASE MANAGEMENT SYSTEM, MONGODB, SCALABILITY, REACT, JAVASCRIPT, TESTING, JEST, CONTROLLERS, INDEXES, ACCESS INTERFACE, ACCOUNTING SYSTEM, GEOCODING

The object of research of the qualification work is information technologies and methods of creating and testing the server and client parts of the food delivery service information system.

The subject of research in the qualification work is methods of testing the information system of a food delivery service.

The purpose of the research is to develop a methodology for testing food order delivery information systems.

The scope of application is automation of information systems testing..

The results of the work are a food order delivery information system tested in both the client and server parts, which has the ability to track the courier's route on the map.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ІС — Інформаційна система

ПЗ — Програмне забезпечення

API — Інтерфейс програмування додатків (Application Programming Interface)

BDD — Розробка через поведінку (Behavior-Driven Development)

CI/CD — Безперервна інтеграція та безперервне постачання (Continuous Integration/Continuous Delivery)

CRUD — Операції створення, читання, оновлення та видалення даних (Create, Read, Update, Delete)

FMEA — Аналіз режимів і наслідків відмов (Failure Modes and Effects Analysis)

JIRA — Платформа для управління проектами і завданнями

QA — Забезпечення якості (Quality Assurance)

SQL — Мова структурованих запитів (Structured Query Language)

SLA — Угода про рівень обслуговування (Service Level Agreement)

TDD — Розробка через тестування (Test-Driven Development)

UI — Користувацький інтерфейс (User Interface)

UX — Досвід користувача (User Experience)

ВСТУП

Сучасний розвиток інформаційних технологій значно змінює спосіб ведення бізнесу в різних сферах, включаючи індустрію доставки їжі. В умовах стрімкого розвитку електронної комерції, високої конкуренції та зміни споживчих звичок, служби доставки їжі повинні пропонувати зручні та ефективні рішення для управління замовленнями. У цьому контексті інформаційні системи, які забезпечують роботу таких служб, займають ключову роль, оскільки вони визначають ефективність і стабільність всієї діяльності компанії.

Забезпечення безперебійної роботи таких систем є критично важливим для успіху компаній, оскільки помилки в обробці замовлень, затримки у доставці або неполадки в інтерфейсі можуть призвести до незадоволення клієнтів. Важливим завданням є забезпечення високої якості функціонування програмного забезпечення служб доставки, що передбачає постійну перевірку і тестування систем.

Тестування інформаційних систем є складним і багатогранним процесом, який включає в себе перевірку не тільки функціональності системи, але й її продуктивності, безпеки, зручності використання та здатності працювати в умовах високого навантаження. У випадку з інформаційними системами для доставки їжі ці критерії стають особливо важливими, адже кожен недолік або збій може безпосередньо вплинути на обслуговування клієнтів і, відповідно, на ефективність бізнесу.

Дана кваліфікаційна робота спрямована на розробку методики тестування інформаційної системи для служби доставки їжі. Метою роботи є створення систематизованого підходу до тестування програмного забезпечення, який дозволить виявити основні недоліки в роботі системи та забезпечити її стабільну і безперебійну роботу.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної галузі: інформаційні системи, їх поняття, види та особливості

Інформаційні системи (ІС) — це комплекси, що складаються з апаратного та програмного забезпечення, даних і процедур, призначених для обробки та передачі інформації в межах організації або між різними організаціями. Вони сприяють автоматизації процесів управління, забезпеченню ефективності прийняття рішень та обробці інформації для різних цілей. ІС є невід’ємною частиною сучасного бізнесу, де вони допомагають оптимізувати роботу з клієнтами, обробляти замовлення, аналізувати дані та підвищувати ефективність організаційних процесів [1].

Інформаційні системи мають кілька основних компонентів:

- апаратне забезпечення: комп'ютери, сервери, мережі та інші технічні пристрої, необхідні для функціонування системи;
- програмне забезпечення: програми та додатки, що обробляють дані і виконують задачі, необхідні для досягнення бізнес-цілей;
- дані: інформація, яка обробляється та зберігається в системі. Це можуть бути клієнтські дані, фінансові звіти, інвентаризація тощо;
- процедури: опис процесів і інструкцій, які визначають, як дані повинні оброблятися і використовуватися;
- люди: користувачі та оператори системи, які взаємодіють з нею і виконують необхідні завдання.

Інформаційні системи можна класифікувати за різними ознаками, зокрема за функціональністю та сферою застосування. Такими категоріями інформаційних систем є [1]:

– операційні системи — системи, що забезпечують автоматизацію основних операцій підприємства, таких як управління замовленнями, обробка

фінансових транзакцій і управління запасами. Вони працюють з великими обсягами даних у реальному часі і є критично важливими для забезпечення безперервності бізнес-процесів;

– управлінські системи — підтримують прийняття рішень на рівні керівництва. Вони аналізують інформацію, надаючи зручні інструменти для планування, прогнозування та розробки стратегій. Такі системи є важливими для формулювання довгострокових цілей організації;

– інформаційно-аналітичні системи — призначені для збору, обробки та аналізу даних з різних джерел. Вони використовуються для виявлення тенденцій і закономірностей, що допомагають у прийнятті стратегічних управлінських рішень. Ці системи зазвичай застосовуються для комплексного аналізу даних про ринок, конкурентів, попит на продукти або послуги;

– системи підтримки прийняття рішень (DSS) — допомагають менеджерам в ухваленні складних рішень, враховуючи багато змінних та альтернатив. Такі системи є важливими для ситуацій, де немає чітких рішень і потрібен глибокий аналіз;

– експертні системи — використовують знання експертів для вирішення задач, які зазвичай потребують людського досвіду і кваліфікації. Ці системи здатні надавати рекомендації на основі логіки та набутого досвіду;

– системи управління взаємодіями з клієнтами (CRM) — допомагають організаціям управлінню взаємодіями з клієнтами та покращенню обслуговування. Ці системи збирають, зберігають та аналізують інформацію про клієнтів, їх переваги, історію замовлень та комунікацій. CRM-системи дозволяють персоналізувати обслуговування клієнтів, сприяючи лояльності та підвищенню рівня задоволеності. Вони також автоматизують процеси продажів і маркетингу, що дозволяє компаніям працювати з потенційними клієнтами більш ефективно;

– системи автоматизованого проектування (CAD) — застосовуються для створення проектів і моделей, зокрема в архітектурі, інженерії та

промисловому дизайні. Ці системи дозволяють створювати точні моделі та схеми, що спрощує процес розробки та реалізації проектів.

Особливості інформаційних систем для обслуговування замовлень включають потребу в обробці великих обсягів даних і забезпеченні їх швидкої обробки для своєчасної доставки замовлень. Ці системи повинні бути інтегровані з платіжними системами, логістичними платформами та системами управління запасами для безперебійного обслуговування клієнтів. Важливим аспектом є персоналізація обслуговування, яка дозволяє враховувати індивідуальні вподобання користувачів, підвищуючи рівень їх задоволеності.

Крім того, безпека даних є надзвичайно важливою. ІС для обслуговування замовлень обробляють персональну інформацію клієнтів і фінансові транзакції, тому питання захисту даних є критичним. Для цього використовуються сучасні методи шифрування та захисту інформації.

Ще однією суттєвою особливістю є масштабованість системи. Вона повинна адаптуватися до зростаючої кількості замовлень та підвищення навантаження, забезпечуючи ефективність навіть при непередбачених піках навантаження або збільшенні клієнтської бази.

Інформаційні системи для обслуговування замовлень повинні бути не тільки надійними, швидкими та безпечними, але й гнучкими, здатними інтегруватися з іншими бізнес-процесами і адаптуватися до змінюваних умов ринку. Тому їх тестування повинно враховувати всі ці аспекти, щоб забезпечити максимальну ефективність і безпеку системи на кожному етапі її функціонування.

1.2 Основи тестування вебсайтів

Тестування вебресурсів — це процес перевірки їх на наявність небажаних наслідків для різних пристроїв і браузерів. Воно включає функціональне тестування, перевірку зручності використання, доступності та

продуктивності. Це дозволяє покращити якість вебзастосунків, що приносить значні переваги для бізнесу [2].

Процес тестування полягає в перевірці відповідності ресурсу встановленим вимогам і виявленні дефектів. Використовуються ручні або автоматизовані інструменти для оцінки таких властивостей, як функціональність, зручність використання, безпека та продуктивність. Основне завдання тестування — виявити помилки, недоліки або невідповідності вимогам, що можуть вплинути на ефективність і користувацький досвід.

Тестування вебсайтів включає перевірку низки аспектів, таких як функціональність, продуктивність, доступність та безпека. Основні завдання тестування включають виявлення помилок у функціонуванні вебсайту, зручність взаємодії користувача з сайтом, а також перевірка швидкості його роботи та захисту даних [2].

Основні типи тестування веб-сайтів [3]:

– функціональне тестування — перевірка правильності виконання функцій на сайті відповідно до вимог. Це включає перевірку форм, кнопок, лінків, інтерактивних елементів, правильності обробки даних.

Типові перевірки: перевірка правильності роботи форм (реєстрації, входу, підписки), тестування кнопок (коректність дій при натисканні), перевірка працездатності лінків (перехід за правильними URL), тестування роботи фільтрів та пошукових функцій, коректність відображення повідомлень про помилки;

– тестування продуктивності — оцінка того, як вебсайт працює під різним навантаженням, а також швидкості завантаження сторінок.

Типові перевірки: вимірювання часу завантаження сторінок, тестування сайтів при великій кількості одночасних користувачів (стрес-тестування), перевірка швидкості виконання запитів до бази даних, тестування на швидкість завантаження з різних пристроїв та мереж;

– тестування на доступність — перевірка можливості доступу до сайту для користувачів з обмеженими можливостями, таких як слабозорі або слухові порушення.

Типові перевірки: перевірка підтримки екранних читалок, доступність навігаційних елементів для людей з обмеженими можливостями, перевірка контрастності тексту, тестування використання клавіатури для навігації (відсутність потреби в миші), забезпечення можливості збільшення шрифтів без втрати змісту;

– тестування безпеки — виявлення потенційних вразливостей вебсайту, які можуть бути використані для несанкціонованого доступу або атак.

Типові перевірки: перевірка на наявність SQL-ін'єкцій, тестування захисту від Cross-Site Scripting (XSS), перевірка на наявність вразливостей, що дозволяють доступ до конфіденційної інформації (наприклад, паролі), тестування на захищеність від атак типу Denial of Service (DoS), перевірка правильності впровадження HTTPS на всіх сторінках сайту;

– тестування платежів — важливий аспект тестування, коли вебсайт має функції електронної комерції. Це включає перевірку коректності обробки платіжних операцій, правильність обчислення вартості товарів або послуг, перевірку інтеграції з платіжними системами (наприклад, PayPal, Stripe) і безпеку транзакцій.

Типові перевірки: перевірка введення правильних і неправильних платіжних даних, перевірка коректності завершення транзакції, тестування різних сценаріїв скасування платежів;

– тестування на придатність (Usability Testing) — перевірка зручності та інтуїтивності користувацького інтерфейсу. Це тестування покликане оцінити, наскільки зручно і ефективно користувачі можуть взаємодіяти з сайтом.

Типові перевірки: оцінка легкості навігації, перевірка часу виконання основних завдань користувачем, перевірка відображення важливих елементів інтерфейсу на різних пристроях і екранах.

1.3 Типи тестування: ручне та автоматизоване тестування

Існують різні типи тестування, які застосовуються до вебсайтів [4]:

- ручне тестування — процес, при якому тестувальники виконують перевірки вручну, перевіряючи функціональність, доступність і зручність використання сайту;
- автоматизоване тестування — використання інструментів і скриптів для автоматизації перевірки певних функцій і процесів на сайті, що дозволяє зменшити час і зусилля на повторювані перевірки.

Обидва методи мають свої переваги та недоліки, і їх вибір залежить від специфіки проекту.

Ручне тестування — це процес перевірки програмного забезпечення, коли тестові випадки виконуються вручну тестувальником без використання автоматизованих інструментів. Головною метою цього виду тестування є виявлення помилок, дефектів і проблем у програмному продукті. Ручне тестування є базовим методом серед усіх типів тестувань і дозволяє виявити критичні недоліки в системі.

Перед автоматизацією тестування кожен новий продукт має бути протестований вручну, щоб перевірити його основну функціональність. Хоча ручне тестування вимагає значно більше часу і зусиль, воно є необхідним етапом, який дозволяє визначити можливість подальшої автоматизації.

Процес ручного тестування не потребує спеціальних інструментів, однак тестувальник повинен добре розуміти вимоги до програмного забезпечення. Ці вимоги, які можуть бути представлені як користувацькі історії або інші документи, надають необхідну інформацію про функціональність продукту, області для тестування, а також визначають, що є дефектом. Знання цих вимог

є критично важливим на етапі підготовки до тестування, оскільки вони допомагають тестувальнику зосередитися на перевірці ключових аспектів і максимально наблизити продукт до ідеального безпомилкового стану.

Ручне тестування веб-сайтів включає кілька етапів, на кожному з яких тестувальник перевіряє різні аспекти функціонування сайту [4]. Кожен етап важливий для виявлення потенційних проблем і забезпечення якості кінцевого продукту:

– планування тестування — на цьому етапі тестувальник визначає, які функціональні можливості веб-сайту потрібно перевірити, розробляє план тестування та сценарії для різних тестів. Це включає визначення критеріїв успіху та пріоритетів для тестування;

– підготовка середовища для тестування — тестувальник налаштовує необхідне середовище для виконання тестів, перевіряє конфігурацію браузерів, операційних систем та пристроїв, на яких буде проводитись тестування;

– розробка тестових сценаріїв — на цьому етапі створюються конкретні тести, які будуть використовуватись під час тестування. Тестувальник описує конкретні кроки, які необхідно виконати, для перевірки кожної функціональності сайту;

– виконання тестів — тестувальник виконує тестові сценарії вручну, перевіряючи функціональність сайту. Це може включати перевірку навігації, роботи форм, кнопок, інтерактивних елементів і т.д.;

– збір результатів і звітність — після виконання тестування тестувальник фіксує всі знайдені помилки, проблеми з функціональністю або інтерфейсом, а також надає звіт з результатами тестування, вказуючи важливість кожної проблеми;

– виправлення помилок і повторне тестування — після того, як розробники виправили виявлені помилки, тестувальник повторно перевіряє виправлення, щоб упевнитися, що проблема була вирішена, і сайт працює належним чином.

Існують різні типи ручного тестування, які застосовуються для перевірки різних аспектів вебсайтів [5]. Ось деякі з них:

– функціональне тестування — перевірка, чи працюють усі функції веб-сайту відповідно до технічних вимог. Це включає тестування форм, кнопок, лінків, інтерактивних елементів, які повинні виконувати очікувані дії.

Приклад: перевірка правильності роботи кнопки "Подати заявку", перевірка функціонування пошукової системи на сайті;

– тестування інтерфейсу (UI тестування) — перевірка зручності інтерфейсу для користувачів. Оцінюється, чи є інтерфейс інтуїтивно зрозумілим, чи всі елементи доступні для користувачів, чи зручно їх використовувати.

Приклад: перевірка правильності відображення шрифтів, кольорів, логотипів на сторінках сайту;

– тестування сумісності — перевірка роботи сайту на різних браузерах, операційних системах та пристроях. Це забезпечує правильну функціональність сайту для всіх користувачів незалежно від їхніх налаштувань.

Приклад: тестування сайту на браузерах Chrome, Firefox, Safari, а також на мобільних пристроях з різними розмірами екранів;

– тестування на доступність — оцінка того, наскільки сайт доступний для користувачів з обмеженими можливостями. Це включає перевірку підтримки екранних читалок, коректність роботи з клавіатурою тощо.

Приклад: перевірка наявності альтернативних текстів для зображень, тестування навігації за допомогою клавіатури;

– тестування безпеки — перевірка на наявність вразливостей на веб-сайті, які можуть бути використані для несанкціонованого доступу або атак.

Приклад: перевірка захисту від SQL-ін'єкцій, перевірка правильності впровадження HTTPS на всіх сторінках сайту;

– тестування продуктивності — перевірка, як сайт поводиться під високим навантаженням або при великий кількості одночасних користувачів. Тестувальники перевіряють час відгуку та швидкість роботи веб-ресурсу.

Приклад: перевірка часу завантаження головної сторінки сайту, тестування відгуку сайту за умов високої кількості одночасних запитів;

– тестування локалізації — перевірка правильності відображення контенту сайту на різних мовах, перевірка формату дати, часу, валют для різних географічних регіонів.

Приклад: перевірка коректності відображення тексту на різних мовах, відповідність формату дати для різних країн.

Переваги ручного тестування:

- гнучкість і адаптивність. Ручне тестування дозволяє легко адаптуватися до змін у вимогах або дизайні продукту. Тестувальник може оперативно коригувати тести залежно від змін у програмі, що є важливим, коли працюєш із нестабільними або незавершеними функціональностями;

- виявлення проблем із користувацьким інтерфейсом. Ручне тестування є дуже корисним для виявлення проблем із зовнішнім виглядом програми, зручністю інтерфейсу, логікою взаємодії користувача з додатком. Тестувальник може виявити не тільки технічні помилки, але й проблеми, що стосуються досвіду користувача;

- ефективність у малих і середніх проєктах. Для невеликих проєктів або коли потрібно провести тестування специфічних, не часто змінюваних функцій, ручне тестування може бути швидким і менш ресурсозатратним способом перевірки;

- виявлення нетипових або складних помилок. Ручне тестування дозволяє тестувальникам застосовувати інтуїцію та експертні знання для виявлення помилок, які не можуть бути легко знайдені автоматизованими інструментами. Це важливо для тестування специфічних ситуацій, які можуть виникнути під час використання програми в реальному середовищі;

- здатність до тестування складних або нових функцій: При розробці нових функцій або невідомих технологій, де автоматизовані тести можуть бути відсутні або складні в налаштуванні, ручне тестування є єдиним способом перевірки працездатності та коректності роботи нових компонентів.

Ручне тестування є важливим етапом у тестуванні програмного забезпечення, особливо для малих та середніх проектів або при роботі з новими функціями, що вимагають глибшої перевірки з точки зору користувача.

Недоліки ручного тестування:

- висока трудомісткість. Оскільки всі тестові сценарії виконуються вручну, цей процес може бути дуже часозатратним, особливо при великих проектах з великою кількістю функцій;
- обмежена повторюваність. Ручне тестування важко повторити з однаковою точністю, що може призвести до непослідовності результатів. Відсутність автоматизації ускладнює тестування на різних етапах розробки;
- необхідність великих ресурсів. Для ефективного проведення тестування потрібна команда досвідчених тестувальників, що збільшує витрати на тестування. Крім того, велику кількість тестів не можна провести одночасно;
- можливість людських помилок. Оскільки тестування виконується вручну, існує ймовірність допущення помилок через втому, недбальство або пропуск деталей, що може призвести до неякісного тестування;
- обмежена ефективність при великих обсягах даних. Коли необхідно протестувати велику кількість даних або перевірити систему з високими навантаженнями, ручне тестування стає малоефективним і часто не здатне забезпечити належну продуктивність.

Ці недоліки роблять ручне тестування менш ефективним у порівнянні з автоматизованим тестуванням у деяких випадках, але воно все одно є важливим етапом у процесі тестування.

Автоматизоване тестування — це процес перевірки програмного забезпечення з використанням спеціалізованих інструментів та скриптів замість ручного виконання тестів. Цей підхід дозволяє зменшити час тестування та підвищити точність перевірки [6].

Автоматизоване тестування базується на заздалегідь підготовлених тестах, які виконуються автоматично. Після виконання тестів система порівнює отримані результати з очікуваними. Часто термін «тестування» може бути неточно трактований, оскільки автоматизація тестів насправді лише перевіряє, чи коректно працює веб-сайт в певних заздалегідь визначених сценаріях.

На відміну від автоматизованого тестування, ручне тестування проводиться безпосередньо людиною, яка вручну виконує тестові кроки на комп'ютері. Програмне забезпечення для автоматизації тестування може також вводити тестові дані в систему, порівнювати фактичні і очікувані результати, а також генерувати детальні звіти. Хоча автоматизація тестування вимагає значних інвестицій часу та ресурсів, вона дозволяє істотно зменшити навантаження на тестувальників при повторних виконаннях тестів.

Автоматизація тестування вимагає виконання тих самих наборів тестів кілька разів протягом циклів розробки. Використання спеціалізованих інструментів для автоматизації дозволяє записувати ці тести та відтворювати їх у потрібний момент без участі людини. Метою автоматизації є зменшення обсягу ручного тестування, а не повне виключення людської участі.

Автоматизоване тестування є важливим для досягнення високої ефективності, швидкості та охоплення тестуванням, оскільки воно дозволяє:

- знизити витрати часу і грошей на ручне тестування всіх процесів, полів і негативних сценаріїв;
- легше тестувати багатомовні сайти, що є складним завданням для ручного тестування;
- виконувати тестування без необхідності постійного нагляду людини, що значно збільшує автономність процесу;

- прискорити виконання тестів;
- розширити охоплення тестування;
- запобігти помилкам, які можуть виникати при виконанні рутинних і повторюваних тестів вручну.

Процес автоматизації тестування складається з кількох етапів [6]:

– вибір інструменту для тестування. Це важливий крок, який визначає, якими засобами буде здійснюватися автоматизація;

– визначення обсягу автоматизації. Не всі тести підлягають автоматизації, тому потрібно вибрати найважливіші сценарії для автоматичного виконання;

– планування, розробка та дизайн тестів. Тут створюються тестові скрипти та визначаються дані для тестування;

– виконання тестів. На цьому етапі запускаються автоматизовані сценарії, вводяться тестові дані, і виконується перевірка результатів;

– технічне обслуговування. Після виконання тестів необхідно перевірити результати і коригувати автоматизовані тести у разі змін в системі чи виникнення нових вимог.

У ході автоматизованого тестування генеруються детальні звіти, які містять інформацію про успішні та неуспішні перевірки. Всі ці етапи разом забезпечують високу ефективність та якість тестування, скорочуючи час на перевірку багатьох сценаріїв.

Переваги автоматизованого тестування:

- швидкість виконання. Автоматизовані тести виконуються набагато швидше, ніж ручні, що дозволяє значно скоротити час тестування;
- повторюваність. Один раз написані тести можуть виконуватися багаторазово без втрати точності чи змін у якості;
- ефективність для великих проєктів. Автоматизація дозволяє тестувати складні системи з великими обсягами даних;
- масштабованість. Легко додавати нові тести до існуючих сценаріїв автоматизації;

- зниження людського фактору. Мінімізується ймовірність помилок, які можуть виникнути через неуважність людини;
- докладна звітність. Інструменти автоматизації генерують звіти з детальним описом тестів, що спрощує аналіз результатів;
- ефективне тестування регресії. Автоматизація дозволяє легко перевіряти, чи не виникли помилки в результаті внесення нових змін до коду;
- можливість тестування у різних середовищах. Автоматизовані тести можуть одночасно запускатися на різних платформах, браузерах та пристроях;
- економія часу. Скорочується час, потрібний для ручного тестування рутинних та повторюваних сценаріїв.

Недоліки автоматизованого тестування:

- високі початкові витрати. Вимагає значних фінансових вкладень для закупівлі інструментів, навчання та написання тестових сценаріїв;
- складність налаштування. Потребує часу і досвіду для налаштування інструментів автоматизації та створення тестів;
- не підходить для всіх типів тестування. Наприклад, тестування зручності користування або дизайну потребує людського втручання;
- обмежена гнучкість. Автоматизовані тести можуть не враховувати непередбачувані ситуації, які можна виявити лише вручну;
- витрати на обслуговування. Тести необхідно регулярно оновлювати, особливо якщо продукт постійно змінюється;
- непридатність для коротких проєктів. Для невеликих проєктів автоматизація може виявитися дорожчою і зайняти більше часу, ніж ручне тестування;
- високі вимоги до навичок. Тестувальники мають володіти програмуванням і розуміти роботу інструментів автоматизації;
- не гарантує відсутність дефектів. Автоматизовані тести перевіряють лише заздалегідь задані сценарії, але не виявляють помилки поза цими межами.

Ці переваги та недоліки показують, що автоматизоване тестування ефективне для великих і довгострокових проєктів, але для повного охоплення тестування його варто комбінувати з ручними методами.

Для порівняння переваг та недоліків обох підходів можна використовувати таблицю 1.1.

Таблиця 1.1 – Порівняння між тестуваннями

Критерій	Ручне тестування	Автоматизоване тестування
Час виконання тестів	Займає більше часу, особливо для великих наборів тестів.	Швидке виконання тестів, особливо при повторних перевірках.
Вартість	Початково дешевше, оскільки не потребує розробки скриптів.	Вимагає інвестицій у створення та підтримку тестових скриптів.
Точність	Можливі людські помилки.	Висока точність при виконанні повторюваних тестів.
Гнучкість	Легко застосовувати для складних або незвичайних сценаріїв.	Менше гнучкості, обмежене автоматизацією тестування складних випадків.
Застосування	Підходить для перевірки інтерфейсів користувача та нових функцій.	Ідеальне для перевірки регресії та стабільності продукту.
Застосування в ІС доставки їжі	Тестування взаємодії користувача з інтерфейсом для врахування інтуїтивності.	Тестування базових функцій системи замовлень і швидкості обробки замовлень.

З нашого порівняння можемо вивести основні висновки щодо тестування:

– комбінований підхід є найефективнішим.

Поєднання ручного та автоматизованого тестування дозволяє охопити всі аспекти перевірки, забезпечуючи баланс між гнучкістю та швидкістю виконання тестів;

– ручне тестування ідеально підходить для нестандартних перевірок.

Людське втручання необхідне для оцінки зручності використання, дизайну та реакції системи на нестандартні сценарії, що не завжди можна передбачити під час автоматизації;

– автоматизація ефективна для регресійного тестування та повторюваних завдань.

Автоматизовані тести зменшують витрати часу на виконання одноманітних завдань і забезпечують стабільність у перевірці змін у програмному коді;

– вартість та ресурси мають вирішальне значення для вибору підходу;

– автоматизація потребує більше початкових інвестицій та технічних знань, тому може бути доцільною для великих та довготривалих проєктів. Ручне тестування залишається доступним для коротких або менш складних проєктів;

– гнучкість і адаптація важливі для сучасного веб-тестування. У динамічних проєктах із частими змінами ручне тестування є більш гнучким, тоді як автоматизація вимагає постійного оновлення сценаріїв;

– кінцева мета тестування — якість продукту. Незалежно від методу тестування, основна задача полягає у забезпеченні безпомилкової роботи веб-сайту та його відповідності вимогам користувачів.

1.4 Методи тестування: чорна скринька, біла скринька та тестування за допомогою сценаріїв

Для досягнення максимальної ефективності тестування використовуються різні методи, основними методами тестування є методи «чорної скриньки», «білої скриньки» та тестування за допомогою сценаріїв, кожен з яких має свої переваги і недоліки.

Метод «чорної скриньки» передбачає тестування програмного забезпечення без доступу до його внутрішньої структури або коду. Тестувальники працюють лише з інтерфейсом програми, перевіряючи її функціональність з точки зору користувача. Основною перевагою цього методу є те, що він дозволяє зосередитися на тому, як система працює з точки зору кінцевого користувача. Однак цей метод має й недоліки: відсутність доступу до коду може призвести до пропуску важливих дефектів, що виникають на рівні внутрішньої логіки програми [7].

Переваги методу «чорної скриньки»:

- не потребує знання коду;
- доступний для тестувальників без технічного досвіду;
- зосереджений на функціональності та вимогах користувачів.

Недоліки методу «чорної скриньки»:

- обмежене тестування внутрішньої логіки системи;
- висока ймовірність пропуску помилок, пов'язаних з кодом;
- залежність від наданої документації, яка може бути неповною або неточною.

Метод «білої скриньки» використовує знання коду для детального тестування внутрішньої структури програми. Тестувальники перевіряють, як працює програма на рівні алгоритмів, структур даних та логіки. Цей метод дозволяє знайти дефекти в коді, оптимізувати програму та забезпечити її стабільну роботу [7].

Переваги методу «білої скриньки»:

- глибоке тестування внутрішніх компонентів програми;
- виявлення дефектів на рівні коду і структури;
- можливість оптимізації та покращення ефективності коду.

Недоліки методу «білої скриньки»:

- вимагає глибоких знань у програмуванні;
- трудомісткість і висока вартість для великих систем;
- не дає повного уявлення про те, як програма сприймається користувачем.

Метод тестування за допомогою сценаріїв полягає в перевірці програмного забезпечення за допомогою тестових сценаріїв, що моделюють реальні умови використання системи. Цей метод дозволяє перевірити, як система поводиться при взаємодії з користувачем і яким чином вона реагує на різноманітні ситуації. Сценарії можуть бути написані як на основі функціональних вимог, так і в контексті реальних бізнес-процесів [7].

Переваги методу тестування за допомогою сценаріїв:

- імітує реальні умови використання системи;
- дозволяє перевірити систему з точки зору користувача, що може бути корисно для виявлення проблем в інтерфейсі;
- може бути використаний для тестування функціональних та нефункціональних вимог.

Недоліки методу тестування за допомогою сценаріїв:

- вимагає детальної підготовки тестових сценаріїв, що може займати багато часу;
- може не виявити всі внутрішні дефекти системи;
- обмежене покриття всіх можливих ситуацій, якщо сценарії недостатньо різноманітні.

Усі ці методи тестування є необхідними для забезпечення належної якості та безпеки інформаційних систем, зокрема тих, що використовуються для автоматизації обробки замовлень. Використання комбінованого підходу

дозволяє виявляти як помилки, що пов'язані з кодом, так і дефекти функціональності, що можуть вплинути на користувацький досвід.

1.5 Життєвий цикл розробки та тестування інформаційних систем

Життєвий цикл розробки інформаційної системи (ІС) — це процес, який охоплює всі етапи створення програмного продукту, починаючи з визначення цілей і закінчуючи його підтримкою та оновленням після впровадження. Основною метою життєвого циклу є створення якісного продукту, який відповідає вимогам користувачів і бізнесу. Важливо, щоб на кожному етапі враховувалися аспекти тестування для забезпечення стабільності, продуктивності та безпеки системи [8].

Основні етапи життєвого циклу розробки ІС

Етап 1. Планування та аналіз вимог.

Зміст етапу: визначаються цілі проєкту, аналізуються потреби бізнесу та користувачів, формується перелік функціональних та нефункціональних вимог. Зазвичай це відбувається за участі аналітиків, замовників і команд розробників.

Для вебсайтів, особливо у сфері доставки їжі, визначаються такі аспекти:

- ключова функціональність (наприклад, кошик, система оплати, меню);
- цільова аудиторія та їхні потреби;
- інтеграція із зовнішніми сервісами (платіжні системи, API кур'єрських служб).

Особливості тестування:

- верифікація та валідність вимог: чи є вони зрозумілими, повними та такими, що піддаються перевірці;
- виявлення можливих проблем або протиріч у вимогах;
- аналіз ризиків, які можуть вплинути на тестування та якість кінцевого продукту.

Етап 2. Проєктування та архітектура.

Зміст етапу: після визначення вимог створюється технічний проєкт, що описує архітектуру системи, логічну структуру, взаємодію компонентів і технологічні рішення. У контексті вебсайтів це включає:

- розробку макетів і прототипів (Wireframes, UI-дизайн);
- вибір технологій (фреймворки, платформи, мови програмування);
- розробку бази даних (структура, зв’язки);
- планування інтеграції з іншими системами.

Особливості тестування:

- аналіз архітектури на предмет її відповідності вимогам;
- перевірка логіки та цілісності проєкту;
- виявлення потенційних вузьких місць у продуктивності (наприклад, можливість роботи з великими обсягами даних).

Етап 3. Розробка та програмування.

Зміст етапу: це процес створення коду відповідно до проєктної документації. Для вебсайтів це може включати:

- розробку клієнтської частини (фронтенд);
- розробку серверної частини (бекенд);
- інтеграцію бази даних;
- налаштування API для взаємодії з іншими сервісами.

Особливості тестування:

- модульне тестування: перевірка роботи окремих функцій та компонентів;
- аналіз коду на помилки, відповідність стандартам якості та безпеки;
- перевірка інтеграції між компонентами (наприклад, взаємодія фронтенду з бекендом).

Етап 4.Тестування.

Зміст етапу: основною метою тестування є виявлення помилок і недоліків у роботі системи до її впровадження. Для вебсайтів тестування

охоплює всі аспекти роботи: функціональність, продуктивність, сумісність, безпеку, зручність використання тощо.

Особливості тестування:

- використання автоматизованих тестів для рутинних перевірок;
- проведення ручного тестування для оцінки зручності інтерфейсу та поведінки системи у різних сценаріях;
- стрес-тестування для визначення, як сайт працює під високим навантаженням (наприклад, під час пікових годин замовлень).

Етап 5. Впровадження.

Зміст етапу: система переноситься з середовища розробки в робоче середовище. Для вебсайтів це означає запуск на сервері, налаштування домену та забезпечення доступності для користувачів.

Особливості тестування:

- перевірка працездатності в реальних умовах;
- тестування на різних пристроях та браузерах;
- перевірка безпеки (наприклад, захист від SQL-ін’єкцій чи інших атак).

Етап 6. Підтримка та обслуговування.

Зміст етапу: після впровадження система продовжує оновлюватися, доповнюватися новими функціями та підтримуватися для забезпечення стабільної роботи.

Особливості тестування:

- тестування після внесення змін (регресійне тестування);
- постійний моніторинг роботи системи (виявлення проблем до їх виникнення);
- оцінка зворотного зв’язку від користувачів та внесення відповідних коректив.

На кожному етапі життєвого циклу розробки ІС інтеграція тестування дозволяє мінімізувати ризики, забезпечити відповідність продукту вимогам та підвищити його якість. Для вебсайтів це особливо важливо, адже вони часто є

основним інструментом взаємодії з клієнтами та повинні працювати бездоганно.

1.6 Ітераційний підхід до розробки та тестування: використання Agile, Scrum або Waterfall у процесі розробки та тестування ІС

Ітераційні підходи до розробки інформаційних систем забезпечують гнучкість, швидкість реагування на зміни та ефективну інтеграцію тестування в процес створення продукту. У контексті розробки вебсайтів для доставки їжі це особливо актуально, адже такі системи повинні відповідати високим вимогам користувачів щодо функціональності, продуктивності та зручності використання. Розглянемо ключові підходи: Agile, Scrum та Waterfall, їх особливості та роль тестування [8].

Agile (англ. Agile software development, agile-методи) — це філософія розробки програмного забезпечення, що ґрунтується на ітераціях, гнучкості та тісній співпраці між усіма учасниками процесу [9] (рис. 1.1).

Основні принципи:

- проєкт поділяється на невеликі цикли (ітерації);
- після кожної ітерації команда надає працюючу частину продукту;
- постійний зворотний зв'язок від клієнта;
- акцент на швидку адаптацію до змін у вимогах.

Особливості тестування:

- тестування проводиться на кожній ітерації, що дозволяє швидко виявляти та виправляти дефекти;
- автоматизація тестів допомагає прискорити процес перевірки;
- використання методик Test-Driven Development (TDD) або Behavior-Driven Development (BDD) [9].

Приклад у сфері доставки їжі: під час кожної ітерації можуть тестуватися окремі функції, наприклад: система пошуку ресторанів, вибір способу доставки, оплата замовлень.

Завдяки коротким ітераціям можна швидко впровадити нові функції, наприклад інтеграцію з новими платіжними системами або оновлення інтерфейсу [9].

Рисунок 1.1 – Методологія Agile

Scrum — це конкретна реалізація Agile, яка передбачає структурований підхід до організації роботи команд [9] (рис 1.2).

Рисунок 1.2 – Методологія Scrum

Основні принципи:

- робота розбивається на короткі проміжки часу — спринти (зазвичай 2-4 тижні);

- у кожному спринті команда зосереджується на створенні певної частини функціоналу;
- на початку кожного спринту проводиться планування, а в кінці — демонстрація результатів і ретроспектива.

Ролі у Scrum:

– Product Owner — відповідає за формулювання вимог і визначення пріоритетів;

– Scrum Master — координує роботу команди та забезпечує дотримання принципів Scrum;

– Розробники та тестувальники — створюють і тестують продукт.

Особливості тестування:

- тестування є невід'ємною частиною кожного спринту;
- інтеграція тестувальників у команду розробників сприяє тісній співпраці та швидкому виявленню проблем;
- використання тестових стендів для перевірки нового функціоналу перед інтеграцією в основну систему.

Приклад у сфері доставки їжі: під час одного спринту команда працює над розробкою функції попереднього замовлення; тестувальники перевіряють працездатність цієї функції, а також її взаємодію з іншими компонентами системи (наприклад, часова розмітка доставки).

Waterfall або каскадна модель — це традиційний підхід до розробки, де кожен етап проєкту завершується перед переходом до наступного [9] (рис.1.3).

Основні принципи:

- чітка послідовність етапів: планування → проєктування → розробка → тестування → впровадження;
- кожен етап має бути завершений повністю перед початком наступного;
- зміни у вимогах після початку проєкту зазвичай ускладнюють процес.

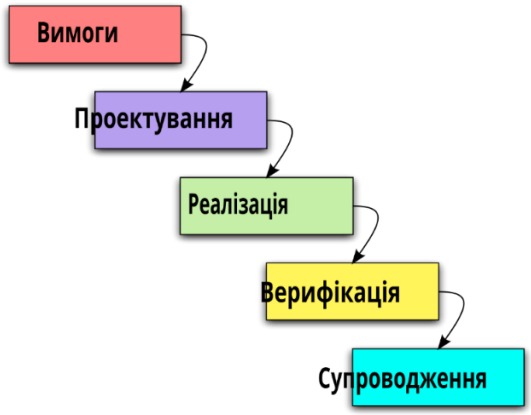

Рисунок 1.3 – Методологія Waterfall

Особливості тестування:

- основне тестування проводиться лише після завершення етапу розробки;
- використовуються ретельно підготовлені тест-кейси для перевірки функціональності, зазначеної у вимогах;
- модульне тестування може проводитися на етапі розробки, але комплексне — тільки після завершення всього коду.

Приклад у сфері доставки їжі: під час розробки системи замовлення тестування починається після завершення всієї розробки; усі функції, такі як меню, оплата, підтвердження замовлення та доставка, перевіряються одночасно.

Хоча Waterfall підходить для проєктів із чітко визначеними вимогами, його недоліком є складність внесення змін у процесі [9].

Тож, можемо порівняти ці методології у таблиці 1.2.

Ітераційні підходи, такі як Agile і Scrum, дозволяють швидко адаптувати систему до змін і проводити тестування на кожному етапі, забезпечуючи високу якість продукту. Для розробки веб-сайтів доставки їжі такі підходи є ідеальними через їхню гнучкість та здатність забезпечувати швидкий вихід функціонального продукту на ринок. У той же час Waterfall підходить для проєктів із чітко визначеними вимогами, але може виявитися неефективним для динамічного середовища.

Таблиця 1.2 – Порівняння методології

Підхід	Гнучкість до змін	Тестування	Тривалість циклу	Приклад застосування
Agile	Висока	На кожній ітерації	Короткі ітерації	Інтеграція нових функцій у вебсайт доставки їжі
Scrum	Висока	На кожному спринті	2-4 тижні	Поетапне створення функціоналу, як-от інтеграція із зовнішнім API
Waterfall	Низька	На етапі тестування	Тривалий цикл	Проєкти з чіткими вимогами, наприклад, розробка системи без змін

1.7 Роль тестування на різних етапах життєвого циклу

Тестування ІС, зокрема вебсайтів для доставки їжі, є одним із найважливіших компонентів життєвого циклу розробки. Воно дозволяє не лише виявляти помилки, але й забезпечувати відповідність продукту вимогам користувачів і бізнесу. Розглянемо, яку роль відіграє тестування на кожному етапі життєвого циклу розробки інформаційної системи.

Етап планування та аналізу вимог.

На цьому етапі визначаються ключові цілі проєкту, функціональні та нефункціональні вимоги до системи, а також архітектура майбутнього продукту [10].

Роль тестування:

- аналіз вимог (Requirements Review). Тестувальники разом із бізнес-аналітиками перевіряють, чи вимоги повні, чіткі та логічні. Це дозволяє виявити прогалини та суперечності ще до початку розробки;

- створення тестових сценаріїв на основі вимог. Наприклад, для вебсайту доставки їжі можна сформувати сценарії перевірки, чи враховані різні способи оплати, можливість вибору адреси доставки, функції перегляду історії замовлень тощо;

- ризик-аналіз. Визначення потенційних ризиків, пов’язаних із функціональністю або технічними обмеженнями.

Значення: ретельний аналіз вимог дозволяє зменшити кількість помилок, які могли б виникнути на наступних етапах через невраховані або нечіткі вимоги.

Етап проєктування та архітектури.

На цьому етапі формується архітектурний дизайн системи, розробляються деталі її структури та логіки роботи [10].

Роль тестування:

- огляд архітектури (Architecture Review). Тестувальники беруть участь у перевірці технічного дизайну, оцінюючи його на відповідність вимогам;

- виявлення потенційних вузьких місць. Наприклад, для веб-сайтів доставки їжі важливо заздалегідь врахувати, як система оброблятиме одночасні замовлення або раптове збільшення трафіку під час свят;

- моделювання тестового середовища. На основі архітектурного дизайну визначаються вимоги до тестового середовища, яке максимально наближається до реальних умов роботи.

Значення: виявлення проблем на етапі проєктування допомагає уникнути значних затрат часу й ресурсів на їх виправлення після реалізації.

Етап розробки (кодинг).

Цей етап включає написання коду системи, реалізацію функціональності, модулів і інтеграцій між компонентами [10].

Роль тестування:

- модульне тестування (Unit Testing). Тестування окремих частин коду, щоб переконатися, що вони працюють правильно. Наприклад, перевірка модуля обробки замовлення;
- Peer Review. Тестувальники можуть брати участь у перевірці коду на предмет логічних помилок або вразливостей;
- інтеграційне тестування. На цьому етапі перевіряється взаємодія між різними модулями. Наприклад, чи правильно система опрацює інформацію з форми замовлення та передасть її у базу даних;
- автоматизовані тести. Використання інструментів для постійного тестування функціональності після внесення змін до коду.

Значення: тестування під час розробки дозволяє виявити помилки на ранніх етапах і уникнути їх поширення на інші частини системи.

Етап тестування.

Цей етап повністю присвячений перевірці роботи системи.

Роль тестування:

- функціональне тестування. Перевірка відповідності функціоналу вимогам. Наприклад, тестування процесу оформлення замовлення або роботи фільтрів у меню;
- нефункціональне тестування. Оцінка продуктивності, безпеки, швидкості завантаження сайту, сумісності з різними браузерами та пристроями;
- ручне та автоматизоване тестування. Ручне тестування використовується для перевірки унікальних сценаріїв, а автоматизація – для рутинних завдань, як-от перевірка коректності форм введення даних;
- тестування користувацького досвіду. Перевірка, наскільки система є інтуїтивно зрозумілою та зручною для користувачів.

Значення: комплексне тестування забезпечує відповідність веб-сайту очікуванням клієнтів та бізнесу.

Етап впровадження (деплоймент).

На цьому етапі система розгортається в робочому середовищі й стає доступною для користувачів [10].

Роль тестування:

- Smoke Testing. Перевірка основної функціональності після розгортання;
- моніторинг у реальному часі. Виявлення проблем, які можуть виникати при взаємодії користувачів із системою;
- перевірка продуктивності. Оцінка, як система працює під навантаженням у пікові години.

Значення: тестування на цьому етапі мінімізує ризики виникнення серйозних проблем після запуску системи.

Етап підтримки та обслуговування.

Після впровадження система повинна регулярно оновлюватися, адаптуватися до нових вимог і підтримувати стабільну роботу.

Роль тестування:

- регресійне тестування. Перевірка, що нові зміни не вплинули на існуючу функціональність;
- моніторинг і звітування. Виявлення дефектів у роботі системи за відгуками користувачів або логами роботи;
- тестування оновлень. Перевірка нових функцій перед їх впровадженням.

Значення: підтримка системи забезпечує довготривалу якість роботи й відповідність вимогам бізнесу.

Тестування відіграє ключову роль на кожному етапі життєвого циклу розробки інформаційної системи. Воно допомагає уникнути помилок, забезпечити стабільну роботу та відповідність системи потребам користувачів і бізнесу. Для вебсайтів доставки їжі це особливо важливо, адже будь-яка помилка може вплинути на задоволеність клієнтів та репутацію компанії.

2 ОСОБЛИВОСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ СЛУЖБИ ДОСТАВКИ ЇЖІ

2.1 Архітектура та функціональні можливості інформаційної системи

Інформаційна система служби доставки їжі (ІС СДЇ) є багатокомпонентною платформою, створеною для ефективного забезпечення взаємодії між користувачами, ресторанами, кур'єрами та адміністраторами. Її архітектура побудована з урахуванням високої продуктивності, масштабованості та адаптивності до сучасних вимог ринку [10,11].

Інформаційна система складається з кількох взаємопов'язаних компонентів, кожен з яких виконує унікальну роль для забезпечення загальної функціональності. Її архітектура базується на клієнт-серверній моделі, що дозволяє забезпечити ефективну обробку даних, зручність використання та стабільність у роботі.

Клієнтська частина представлена мобільними додатками для iOS та Android, а також вебдодатком, що дозволяє охопити широку аудиторію користувачів. Основні функції включають реєстрацію та авторизацію користувачів, пошук ресторанів, перегляд меню, оформлення замовлень, вибір способу оплати, відстеження статусу замовлення, отримання сповіщень та комунікацію з кур'єрами. Завдяки використанню сучасних технологій, таких як React, Angular та Flutter (рис. 2.1), вдалося забезпечити адаптивний, швидкий та інтуїтивно зрозумілий інтерфейс для різних пристроїв [11].

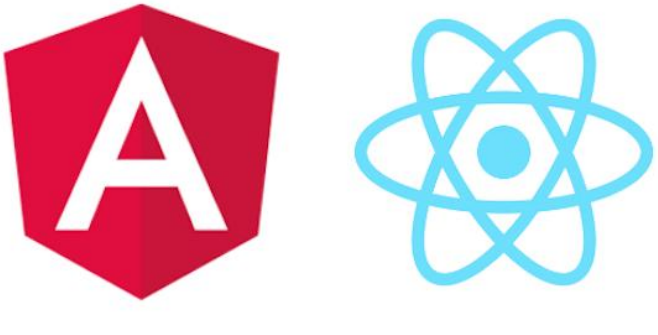

Рисунок 2.1 – Технології для створення вебдодатку

React є потужним інструментом для розробки динамічних вебдодатків із гнучкою компонентою структурою, що дозволяє значно скоротити час розробки [11]. Angular забезпечує структурований підхід до створення додатків з використанням архітектури MVC, що робить його зручним для складних проектів [12]. Flutter дозволяє розробляти мобільні додатки одночасно для двох платформ, використовуючи спільну кодову базу, що є ефективним рішенням для оптимізації ресурсів [13].

Серверна частина виконує ключову роль в обробці запитів, управлінні бізнес-логікою та збереженні даних. Використання Node.js з фреймворком Express дозволяє створювати масштабовані додатки з асинхронною обробкою даних. Python у поєднанні з Django чи Flask забезпечує потужний інструментарій для побудови надійної серверної частини із широкими можливостями налаштування. Spring Boot на основі Java дозволяє швидко розгортати серверні рішення з гнучкими параметрами конфігурації. Кожен із цих інструментів використовується залежно від специфічних вимог проекту.

База даних є критично важливим компонентом системи, що відповідає за збереження та управління інформацією. PostgreSQL забезпечує надійне зберігання структурованих даних і високу продуктивність для складних запитів. MySQL використовується для середніх проектів, забезпечуючи баланс між продуктивністю та простотою. MongoDB надає гнучкі можливості для

роботи з великими обсягами неструктурованих даних, що робить її ідеальною для зберігання інформації про замовлення чи логи системи [14].

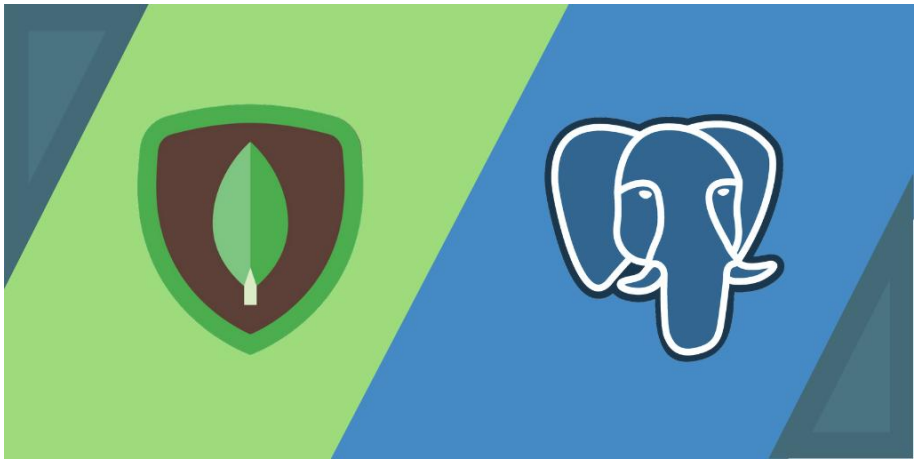

Рисунок 2.3 – СУБД для роботи з БД у системі

Інтеграційні модулі, такі як Stripe, PayPal і LiqPay, забезпечують швидкі й безпечні платіжні транзакції, що є важливим аспектом для користувачів. Google Maps API дозволяє створювати точні маршрути доставки та відстежувати місцеположення кур'єрів у реальному часі. Інтеграція з Google Analytics забезпечує зручний інструмент для аналізу поведінки користувачів та оптимізації функціоналу додатку.

Інфраструктура системи базується на хмарних рішеннях AWS, Google Cloud або Azure, які забезпечують гнучкість у масштабуванні, високу доступність та рівень безпеки. Використання Docker та Kubernetes спрощує процес розгортання системи, полегшує її адміністрування та знижує ризик технічних збоїв.

Функціональні можливості системи охоплюють всі аспекти взаємодії між користувачами, ресторанами, кур'єрами та адміністраторами. Для клієнтів створено інтуїтивно зрозумілий інтерфейс, що дозволяє легко знайти необхідний ресторан, ознайомитися з меню та оформити замовлення. Ресторани мають доступ до зручних інструментів управління своїми послугами та аналітики. Кур'єри отримують навігаційні рішення для

оптимізації маршрутів, а адміністратори можуть контролювати роботу системи та аналізувати її ефективність.

Загалом, архітектура та функціональні можливості інформаційної системи служби доставки їжі створюють надійний інструмент для ефективного виконання завдань бізнесу та забезпечення якісного обслуговування клієнтів.

2.2 Основні бізнес-процеси служби доставки їжі

Бізнес-процеси служби доставки їжі створюють основу для забезпечення надійної роботи системи та охоплюють усі етапи взаємодії між ключовими учасниками: клієнтами, ресторанами, кур’єрами та адміністраторами. Ці процеси забезпечують злагоджену роботу платформи, оптимізацію ресурсів і задоволення потреб усіх сторін.

Основні етапи бізнес-процесу:

1) замовлення їжі клієнтом Цей етап починається з вибору ресторану та страв, що відображаються в інтерфейсі користувача. Система забезпечує клієнтам можливість переглядати меню, сортувати та фільтрувати позиції за популярністю, вартістю або дієтичними уподобаннями. Після цього клієнт оформлює замовлення, обираючи спосіб оплати — онлайн через інтегровані платіжні системи або готівкою при доставці.

Під час оформлення замовлення користувач може додати коментарі або уточнення до страв, що передаються на наступні етапи обробки. Система надсилає повідомлення про підтвердження замовлення через мобільний додаток, SMS або електронну пошту. Завдяки інтуїтивно зрозумілому дизайну, клієнти можуть завершити процес замовлення за лічені хвилини;

2) обробка замовлення у ресторані Після отримання замовлення ресторан через спеціалізований інтерфейс або мобільний додаток підтверджує його прийняття. Автоматизація цього етапу дозволяє уникнути людських помилок і забезпечує оперативність. Замовлення відображається на кухонному

дисплеї, де відзначається його статус: «В обробці», «Готується», «Готове до пакування».

Приготування страв відбувається відповідно до стандартів якості ресторану. Після завершення готування замовлення пакується згідно з екологічними або брендовими стандартами пакування, що забезпечує збереження їжі в належному стані під час доставки. Інтеграція між системою ресторанів і платформою доставки дозволяє ресторанам інформувати кур’єрів про готовність замовлення в режимі реального часу;

3) доставка замовлення кур'єром На цьому етапі кур'єр отримує повідомлення через додаток із деталями замовлення, адресою доставки та оптимальним маршрутом. Завдяки використанню API Google Maps або інших навігаційних сервісів, система пропонує найбільш швидкий та ефективний маршрут, враховуючи поточну ситуацію на дорогах.

Кур'єри можуть відзначати свій статус: «Прийнято», «У дорозі», «Доставлено». Система відстежує їхнє місцеположення, що дозволяє клієнтам в реальному часі бачити, де перебуває їхнє замовлення. Додатково, кур'єри можуть використовувати функцію зв’язку для уточнення деталей доставки, якщо це необхідно;

4) отримання та оцінка замовлення клієнтом Після доставки клієнт перевіряє замовлення та має можливість залишити відгук про якість обслуговування, їжі та пунктуальність кур'єра. Зворотний зв’язок надсилається до бази даних системи, де його аналізують для покращення сервісу. Якщо виникають проблеми, такі як недостача страв або інші інциденти, клієнт може звернутися до служби підтримки, яка через спеціалізований інтерфейс швидко реагує на скаргу.

Додаткові бізнес-процеси, які виконує служба доставки їжі для забезпечення ефективності роботи:

– аналітика та звітність. Платформа автоматично генерує звіти для ресторанів, адміністраторів та кур’єрів, що містять інформацію про кількість замовлень, виручку та зворотний зв’язок від клієнтів;

– маркетингові кампанії: Інформаційна система дозволяє запускати акції, програми лояльності та персоналізовані пропозиції для клієнтів, використовуючи дані про їхні уподобання;

– управління ресурсами. Адміністратори можуть моніторити завантаження ресторанів і кур’єрів, перерозподіляти замовлення та оптимізувати процеси доставки в реальному часі.

Таким чином, бізнес-процеси служби доставки їжі є комплексною системою, яка забезпечує комфорт для клієнтів, ефективність для ресторанів і кур’єрів, а також можливість для масштабування бізнесу за рахунок гнучкої інтеграції нових функцій.

2.3 Вимоги до тестування інформаційної системи

Тестування інформаційної системи служби доставки їжі є критично важливим етапом для забезпечення її надійності, продуктивності та зручності використання. Враховуючи складну архітектуру системи та її багатокомпонентний характер, тестування має охоплювати всі ключові аспекти, від функціональності окремих модулів до інтеграції між ними. Цей процес допомагає виявити й усунути можливі помилки, забезпечити відповідність системи бізнес-вимогам і досягти високого рівня задоволеності користувачів.

2.3.1 Функціональні вимоги до тестування

Основною метою функціонального тестування є перевірка коректності роботи всіх функцій системи відповідно до бізнес-вимог. Зокрема, необхідно забезпечити наступне:

- коректність роботи процесу оформлення замовлення. Верифікація того, що користувач може легко вибрати ресторан, додати страви в кошик, вказати адресу доставки та підтвердити замовлення. Особлива увага

приділяється перевірці відображення інформації про страви, доступність функцій редагування кошика та спрощенню процесу підтвердження замовлення [15];

- інтеграція платіжних систем. Перевірка коректності обробки платежів через різні платіжні сервіси, включаючи повернення коштів у разі скасування замовлення. Важливо врахувати різні методи оплати, а також сценарії, що включають часткові відшкодування та обробку платежів із затримками;
- взаємодія між модулями. Тестування передачі даних між модулями, наприклад, від клієнтської сторони до ресторану, а потім до кур'єра. Важливо переконатися, що всі дані передаються без втрат, а також коректно обробляються помилки в передачі інформації;
- робота з інтерфейсом адміністратора. Перевірка можливості адміністрування замовлень, моніторингу стану доставок і управління кур'єрами. Оцінка зручності інтерфейсу для операторів і адміністраторів системи також є важливою складовою цього етапу.

2.3.2 Нефункціональні вимоги до тестування

Нефункціональні аспекти визначають загальну якість роботи системи, включаючи її продуктивність, масштабованість, безпеку та зручність для користувачів. Основні завдання цього етапу включають [16]:

- продуктивність. Тестування системи під великим навантаженням для оцінки її здатності обробляти велику кількість одночасних замовлень. Наприклад, імітація пікових годин замовлень у п'ятницю ввечері. Крім цього, тестування повинно враховувати можливість раптового збільшення навантаження та аналіз поведінки системи в умовах таких змін;
- сумісність. Перевірка коректної роботи системи на різних пристроях (мобільних телефонах, планшетах, комп'ютерах) та у різних браузерах.

Зокрема, тестування повинно охоплювати різні операційні системи та можливі комбінації браузерів і їхніх версій;

- безпека. Тестування на предмет захисту конфіденційних даних користувачів, включаючи паролі, дані платіжних карток і персональні дані. Особлива увага приділяється захисту від SQL-ін'єкцій, міжсайтових атак (XSS) та інших типів загроз. Крім того, система повинна забезпечувати високий рівень безпеки під час взаємодії між її модулями;
- масштабованість. Аналіз можливості розширення системи при збільшенні кількості користувачів, ресторанів та замовлень. Система має залишатися стабільною та ефективною, навіть якщо кількість одночасних замовлень значно зросте.

2.3.3 Типи тестування

Для досягнення високої якості системи слід застосовувати комплексний підхід до тестування, який охоплює різні види тестування:

1) модульне тестування. Окреме тестування кожного компонента системи (наприклад, модуля оформлення замовлення чи обробки платежів) ізольовано від інших. Цей вид тестування допомагає швидко виявляти проблеми на ранніх етапах розробки;

2) інтеграційне тестування. Перевірка взаємодії між модулями, такими як обмін даними між клієнтським інтерфейсом, рестораном і кур'єром. Важливо також перевірити відповідність даних між базами даних і відображенням у клієнтському інтерфейсі;

3) системне тестування. Тестування системи як єдиного цілого, з урахуванням її поведінки в реальних умовах експлуатації. Оцінка стабільності, швидкодії та загальної якості роботи в реальних сценаріях є ключовим завданням цього етапу;

4) регресійне тестування. Перевірка того, що нові зміни або оновлення системи не спричинили помилок у вже реалізованих функціях. Цей процес

допомагає мінімізувати ризики виникнення несправностей після релізу оновлень;

5) тестування користувацького досвіду (UX). Оцінка зручності використання системи з точки зору клієнтів, ресторанів та кур'єрів. Аналіз результатів дозволяє оптимізувати інтерфейси та процеси для покращення взаємодії користувачів із системою.

2.3.4 Інструменти для тестування

Для ефективного тестування інформаційної системи служби доставки їжі можуть бути використані такі інструменти:

- Selenium для автоматизації тестування веб-інтерфейсу, що дозволяє знизити витрати часу на ручне тестування;
- JMeter для тестування продуктивності та навантаження. Цей інструмент допомагає визначити межі можливостей системи та потенційні вузькі місця;
- Postman для тестування API, що дозволяє забезпечити коректність передачі даних між різними компонентами системи;
- Burp Suite для виявлення вразливостей у безпеці. Використання цього інструмента забезпечує виявлення загроз на ранніх етапах;
- BrowserStack для перевірки сумісності на різних пристроях і браузерах, що дозволяє оцінити доступність системи для широкого кола користувачів.

2.3.5 Умови тестування

Перед початком тестування необхідно створити тестове середовище, яке точно відображає реальні умови роботи системи.

Це включає:

- налаштування серверів і баз даних, ідентичних продуктивному середовищу. Також необхідно врахувати можливість швидкого оновлення цих налаштувань у разі зміни вимог;
- імітацію реальних сценаріїв використання системи за допомогою тестових даних. Тестові дані повинні охоплювати широкий спектр можливих ситуацій, включаючи крайні випадки;
- забезпечення доступу до тестового середовища для всіх учасників процесу тестування. Це дозволить оперативно реагувати на виявлені проблеми та забезпечить узгодженість процесу тестування.

Тестування інформаційної системи служби доставки їжі забезпечує її відповідність технічним і бізнес-вимогам, що є запорукою високої якості обслуговування клієнтів і ефективної роботи всієї екосистеми. Успішне виконання тестування сприяє підвищенню довіри користувачів до системи та її конкурентоспроможності на ринку.

2.4 Можливі ризики і проблеми при впровадженні системи

Впровадження інформаційної системи служби доставки їжі (ІС СДЇ) може супроводжуватися численними ризиками та проблемами, які потрібно врахувати на етапах проектування, розробки й реалізації. Технічні ризики включають можливі перебої в роботі серверів через перевантаження, несумісність із деякими пристроями або платформами, а також помилки в інтеграції зі сторонніми сервісами [20,21].

Фінансові ризики пов'язані з перевищенням бюджету на розробку чи тестування, а також витратами на виправлення критичних помилок після запуску системи. Організаційні проблеми можуть виникнути через недостатню підготовку персоналу до роботи з новою системою або опір змін серед співробітників компанії, які звикли до старих методів роботи.

Безпекові ризики передбачають можливий витік персональних даних користувачів через кібератаки або несанкціонований доступ до бази даних

через недостатній рівень захисту. Проблеми користувацького досвіду можуть виникати через недостатньо зрозумілий інтерфейс, що ускладнює використання системи, або тривалий час завантаження сторінок чи мобільного додатку, який може викликати незадоволення користувачів [20,21].

Для мінімізації цих ризиків важливо ретельно планувати процес тестування, передбачати резерви бюджету та часу для вирішення потенційних проблем, а також забезпечувати навчання персоналу й залучати кваліфікованих спеціалістів на всіх етапах розробки. Крім того, покращення інтерфейсу системи та врахування зручності користувачів ще на етапі проектування допоможе уникнути значної частини проблем. Успішна реалізація цих заходів сприятиме ефективному запуску та подальшій експлуатації ІС СДЇ.

3 ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Методи тестування

Тестування є однією з ключових стадій розробки та підтримки інформаційних систем, що забезпечує їх стабільність, надійність і відповідність вимогам. Процес тестування дозволяє не лише знайти та виправити помилки, але й перевірити функціональність, безпеку та продуктивність системи за різних умов експлуатації. Методи тестування є різноманітними, і їх вибір залежить від багатьох факторів, включаючи тип системи, її складність, цільове середовище та наявні ресурси.

Під методами тестування розуміється сукупність підходів, технік та інструментів, які дозволяють не лише знайти помилки, але й гарантувати, що всі вимоги до системи виконані, а кінцевий продукт відповідає очікуванням користувача. Правильний вибір методів тестування забезпечує оптимальний баланс між витратами часу, ресурсів і якістю.

Методи тестування класифікують за різними критеріями, такими як спосіб виконання, об’єкт дослідження та рівень автоматизації. Загалом, всі методи тестування можна умовно поділити на дві основні групи: ручне тестування та автоматизоване тестування. Кожен з цих підходів має свої переваги, недоліки та сфери застосування.

Ручне тестування передбачає активну участь тестувальника, який вручну перевіряє функціональність системи, оцінює зручність використання та відповідність специфікаціям. Цей метод особливо доречний для перевірки складних сценаріїв, які потребують креативності або суб’єктивного оцінювання, наприклад, тестування користувацького інтерфейсу (UI) або оцінки досвіду користувача (UX).

Автоматизоване тестування базується на використанні спеціальних інструментів і скриптів, які дозволяють виконувати повторювані та

стандартизовані тести без прямої участі людини. Цей підхід забезпечує високу швидкість і точність, що робить його незамінним для перевірки великих обсягів даних, регресійного тестування або тестування продуктивності [23].

Для досягнення повного покриття функціональності системи застосовують модельні підходи до тестування, які дозволяють оцінювати її поведінку з різних точок зору. Модельні методи тестування забезпечують комплексний підхід до перевірки системи та дозволяють знайти як поверхневі, так і глибинні помилки.

Чорна скринька (Black-Box Testing). Цей метод тестування фокусується на перевірці відповідності системи очікуваним результатам без аналізу її внутрішньої структури. Тестувальники працюють виключно з вхідними та вихідними даними, що дозволяє ефективно перевіряти функціональність незалежно від деталей реалізації. Метод «чорної скриньки» особливо ефективний на початкових етапах тестування.

Біла скринька (White-Box Testing). На відміну від «чорної скриньки», цей підхід передбачає глибоке розуміння внутрішньої архітектури системи. Тестувальники аналізують код, логіку роботи та структуру системи, що дозволяє знайти помилки, які не виявляються під час зовнішнього тестування. Такий підхід корисний для перевірки складних алгоритмів, оптимізації коду та забезпечення його безпеки.

Сіра скринька (Gray-Box Testing). Цей гібридний метод поєднує елементи чорної та білої скриньок, дозволяючи оцінювати систему як з точки зору кінцевого користувача, так і з точки зору розробника. Він дозволяє отримати більш збалансовану оцінку функціональності та якості системи.

Одним з ефективних підходів є тестування на основі сценаріїв (Scenario-Based Testing), коли тестувальники моделюють реальні ситуації використання системи. Цей метод дозволяє перевіряти, як система реагує на типові, екстремальні або нетипові сценарії використання, а також виявляти потенційні проблеми, які можуть виникнути в реальних умовах.

Для досягнення максимального покриття функціональності часто використовуються комбіновані методи тестування. Це можуть бути сценарії, які включають як ручне, так і автоматизоване тестування, а також поєднання різних модельних підходів. Такий підхід забезпечує збалансований аналіз та дозволяє врахувати всі аспекти роботи системи, включаючи її функціональність, продуктивність, безпеку та зручність використання.

3.2 Ручне тестування

Ручне тестування є основним етапом у процесі забезпечення якості інформаційної системи. Його метою є виявлення дефектів через взаємодію з системою без використання автоматизованих засобів. Це дозволяє тестувальнику імітувати дії користувача та оцінювати, як система виконує свої функції в реальних умовах [25]. У цьому підрозділі будуть розглянуті вибір функціоналу для ручного тестування, переваги цього підходу, а також засоби, які використовуються при його виконанні.

Вибір функціоналу для ручного тестування є важливим завданням, що вимагає стратегічного підходу. Тестувальники повинні виділити ті аспекти системи, які найбільше впливають на її функціональність, зручність використання та безпеку. Зазвичай ручне тестування виконується для:

- користувацького інтерфейсу (UI). Перевірка зручності та логіки розташування елементів, доступності функцій та коректності відображення інформації на екрані;
- сценаріїв взаємодії. Моделювання реальних дій користувачів, таких як реєстрація, оформлення замовлення або зміна налаштувань;
- критичних функцій. Тестування основних процесів, які є ключовими для роботи системи, наприклад, обробка платежів або обробка замовлень;
- перевірка безпеки. Аналіз можливостей несанкціонованого доступу, перевірка шифрування даних та поведінки системи при спробах злому.

Ручне тестування має ряд важливих переваг, які роблять його незамінним інструментом для забезпечення якості. Однією з основних переваг є можливість тестувальника врахувати людський фактор, тобто виявити помилки, які можуть виникати в реальних умовах використання системи. Інші переваги включають:

- гнучкість. Тестувальник може швидко змінювати стратегію тестування залежно від виявлених проблем або нових вимог;
- оцінка досвіду користувача (UX). Завдяки взаємодії з системою тестувальник може визначити, наскільки вона інтуїтивна та зручна для кінцевого користувача;
- мінімальні технічні вимоги. Для виконання ручного тестування зазвичай не потрібні складні інструменти або програми, що знижує поріг входу для новачків;
- перевірка складних сценаріїв. Людина може оцінити, як система поводиться в нестандартних або унікальних ситуаціях, які важко автоматизувати.

Попри те, що ручне тестування не вимагає автоматизованих інструментів, існує безліч засобів, які спрощують цей процес. Наприклад:

- трекери помилок (Bug Trackers). Системи на кшталт JIRA, Bugzilla або Trello, які допомагають документувати знайдені помилки та відслідковувати процес їх виправлення;
- чек-листи та тестові сценарії. Документи, які містять перелік функцій, що підлягають перевірці, та покрокові інструкції для їх тестування;
- інструменти для запису сеансів. Такі програми, як Loom або ScreenRec, дозволяють записувати процес тестування, щоб спростити аналіз знайдених помилок;
- віртуальні машини. Засоби на кшталт VirtualBox або VMware, які дозволяють тестувати систему на різних конфігураціях без використання фізичних пристроїв.

Таким чином, ручне тестування є важливим етапом у забезпеченні якості інформаційних систем. Завдяки своєму унікальному підходу, який враховує людський фактор, та використанню спеціалізованих засобів, цей метод забезпечує високий рівень точності й адаптивності в процесі тестування.

Тестування функціоналу авторизації та реєстрації є важливим етапом забезпечення якості інформаційної системи. Цей процес дозволяє переконатися, що користувачі можуть коректно реєструватися, авторизуватися та безпечно взаємодіяти із системою.

Тестування починається з аналізу вимог. На цьому етапі вивчаються технічне завдання, дизайн-макети та інша документація, щоб зрозуміти, які сценарії використання необхідно перевірити. Основна увага приділяється як позитивним, так і негативним сценаріям, наприклад, реєстрації з коректними даними, авторизації з помилковими паролями або перевірці захисту від SQL-ін'єкцій. На основі аналізу створюються тест-кейси, які охоплюють усі можливі ситуації взаємодії користувачів із системою.

Підготовка тестового середовища включає створення тестових акаунтів, налаштування бази даних та серверів. Усі компоненти системи повинні бути готовими для перевірки, включаючи API, інтерфейс користувача та лог-файли. Це гарантує можливість відслідковувати будь-які помилки або збої під час тестування.

Під час тестування реєстрації та авторизації перевіряються базові сценарії, такі як реєстрація з валідними даними, авторизація зареєстрованого користувача та вихід із системи. Одночасно тестуються негативні сценарії, наприклад, реєстрація з порожніми полями або введення некоректного формату email. Для граничних значень перевіряється довжина пароля, максимальна довжина email та інші обмеження. Особлива увага приділяється обробці валідаційних повідомлень, щоб вони були інформативними, зручними для користувача та не розкривали зайвих даних.

Безпека є критично важливим аспектом тестування. Перевіряється хешування паролів, блокування акаунтів після кількох невдалих спроб входу

та робота функціоналу відновлення пароля. Усі конфіденційні дані повинні передаватися через захищений протокол HTTPS, щоб уникнути їхнього перехоплення.

Крім цього, проводиться тестування інтерфейсу для перевірки зручності використання. Форми авторизації та реєстрації мають бути доступними на різних пристроях, таких як комп'ютери та мобільні телефони. Усі елементи інтерфейсу повинні бути розташовані логічно, а повідомлення про помилки мають з'являтися в потрібний момент [27].

Для перевірки інтеграції тестується робота API, взаємодія з базою даних та надсилання email-повідомлень, наприклад, листів підтвердження або посилань для відновлення пароля. Усі ці елементи мають працювати стабільно та синхронізовано.

Після внесення змін до коду проводиться регресійне тестування, щоб переконатися, що нові зміни не вплинули на вже реалізовані функції. Усі тест-кейси виконуються повторно, щоб уникнути виникнення нових помилок.

Завершальним етапом є документування результатів. Усі знайдені баги фіксуються у спеціальних системах, таких як Jira або TestRail, з докладним описом умов їхнього відтворення. Також формуються звіти про успішно виконані тести [27].

Після завершення тестування очікується, що функціонал авторизації та реєстрації забезпечує безпеку, інтуїтивно зрозумілий інтерфейс та стійкість до некоректних дій користувачів.

3.2.1 Тестування реєстрації та авторизації користувача

Опис тесту

Тестування реєстрації та авторизації є важливим етапом перевірки правильності роботи механізмів доступу до системи. Перевіряються процеси реєстрації нового користувача, авторизації після реєстрації, а також обробка помилок, що виникають при неправильному введенні даних.

1. Тестування реєстрації нового користувача.

Передумови: користувач не зареєстрований у системі, а система доступна для тестування.

Кроки для виконання: користувач має заповнити форму реєстрації з необхідними полями (ім'я, електронна пошта, пароль) і натиснути кнопку «Зареєструватися». Після цього перевіряється, чи з'являється повідомлення про успішну реєстрацію, а також чи користувач додається до бази даних.

Очікувані результати: користувач отримує підтвердження про успішну реєстрацію, а також з'являється в системі.

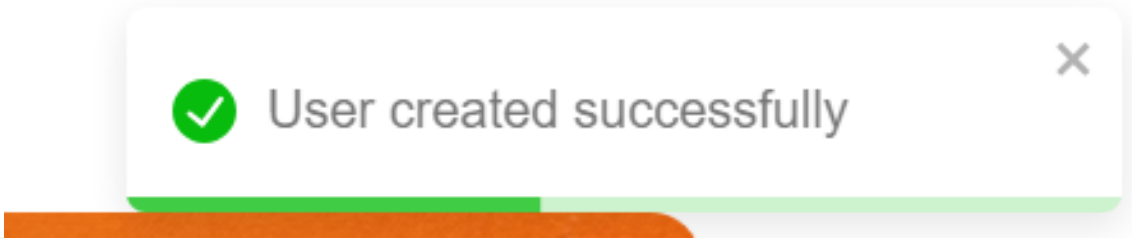

Рисунок 3.1 – Повідомлення про успішну реєстрацію нового користувача у системі

2. Тестування авторизації користувача.

Передумови: користувач зареєстрований у системі.

Кроки для виконання: користувач вводить свою електронну пошту та пароль у форму авторизації та натискає кнопку «Увійти». Після цього перевіряється, чи він перенаправлений на головну сторінку або іншу відповідну сторінку.

Очікувані результати: користувач успішно авторизується і отримує доступ до системи.

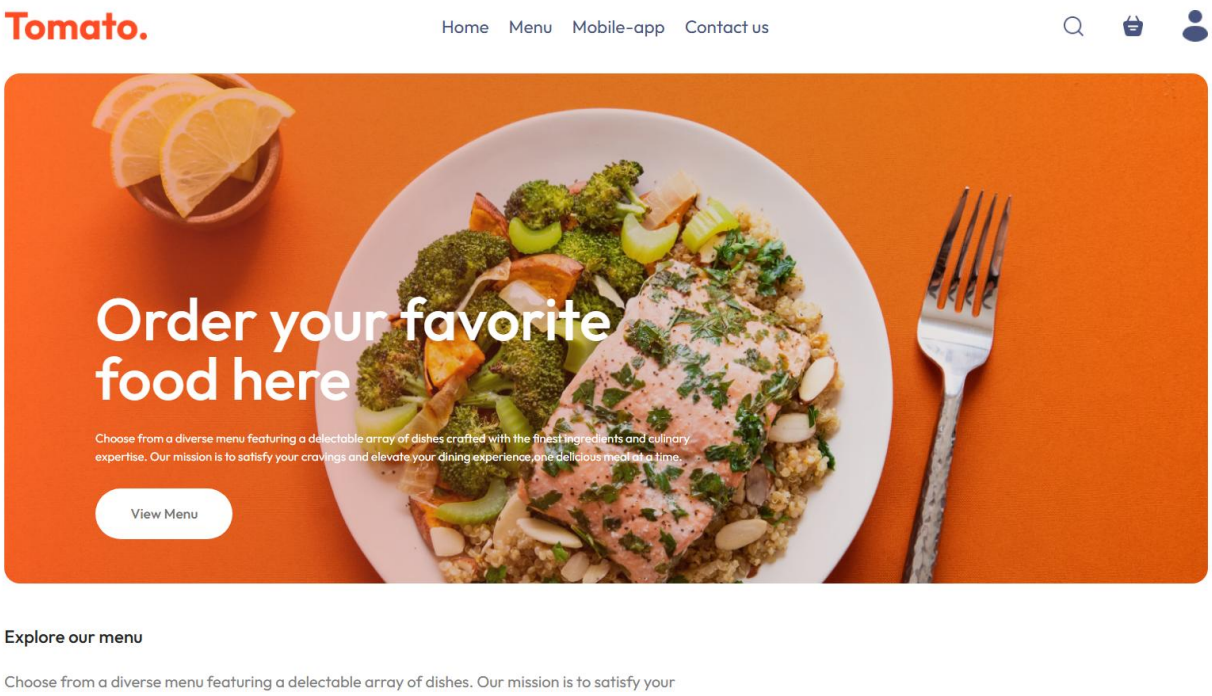

Рисунок 3.2 – Вхід у систему вже авторизованого користувача

3. Тестування помилок при реєстрації та авторизації.

Передумови: користувач вводить неправильні дані під час реєстрації або авторизації.

Кроки для виконання: користувач вводить некоректні дані (неправильний формат електронної пошти або короткий пароль) і намагається зареєструватися або увійти в систему.

Очікувані результати: система має вивести повідомлення про помилку та не дозволити пройти процес реєстрації чи авторизації.

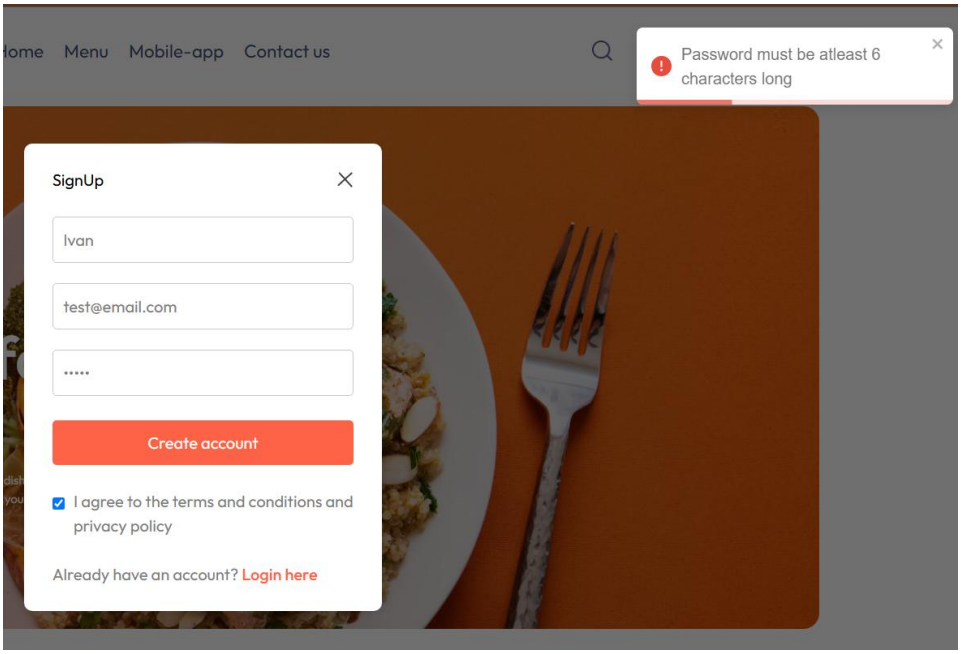

Рисунок 3.3 – Валідація форми вводу паролю для реєстрації

4. Тестування відновлення пароля.

Передумови: користувач забув пароль.

Кроки для виконання: користувач натискає на посилання «Забули пароль?» на сторінці авторизації, вводить свою електронну пошту, отримує лист з посиланням для відновлення пароля і встановлює новий пароль.

Очікувані результати: користувач отримує лист із посиланням для відновлення пароля, після чого успішно змінює пароль і може увійти в систему.

Наступним функціоналом який було вибрано саме для ручного тестування є процес оформлення замовлення. Функціонал оформлення замовлення є критичним етапом у процесі взаємодії користувача з інтернет-магазином або онлайн-системою. Він забезпечує правильність обробки даних клієнта, перевірку доступності товарів, коректність розрахунку вартості, а також обробку платіжних даних. Для успішного тестування цієї функціональності потрібно враховувати кілька важливих аспектів:

1) наявність товарів на складі;
2) оформлення замовлення неможливе, якщо товару немає в наявності

або він недоступний. Необхідно перевірити, чи система правильно обробляє ситуації, коли товар відсутній, і чи повідомляється про це користувач;

3) коректність роботи кошика. Кошик має містити всі товари, які користувач обрав для покупки. Тестування повинно включати перевірку можливості додавання товарів у кошик, видалення товарів та зміну кількості одиниць товару, а також оновлення загальної вартості;

4) правильність підрахунку вартості замовлення. Вартість замовлення повинна бути правильно обчислена, з урахуванням знижок, акцій, вартості доставки, податків та інших націнок. Потрібно перевірити, чи всі зміни, наприклад, застосування купонів на знижку, коректно відображаються в кошику;

5) підтвердження адреси доставки. Потрібно перевірити, чи користувач може правильно вказати свою адресу доставки, обрати зручний метод доставки та оплатити замовлення;

6) тестування процесу оплати. Система має коректно обробляти вибір способу оплати, а також перевіряти правильність введених платіжних реквізитів, якщо такий спосіб передбачений. Окремо необхідно перевірити, чи обробляється відмова від оплати або помилка в процесі транзакції;

7) перевірка підтвердження замовлення. Після оформлення замовлення користувач має отримати підтвердження на екран та на електронну пошту (якщо передбачено). Це підтвердження повинно містити інформацію про зроблену покупку, її вартість, статус замовлення та орієнтовну дату доставки.

3.2.2 Тестування оформлення замовлення

Опис тесту

Тестування оформлення замовлення включає перевірку коректності процесу покупки товарів в інтернет-магазині. Це важливий етап, який забезпечує правильне додавання товарів до кошика, обробку даних доставки, правильність підрахунку вартості та вибір способу оплати.

1. Тестування додавання товарів у кошик.

Передумови: користувач перебуває на сторінці товару та готовий додати його до кошика.

Кроки для виконання: користувач вибирає товар, вказує кількість і натискає кнопку «Додати в кошик».

Очікувані результати: товар додається до кошика, і загальна вартість замовлення оновлюється відповідно до кількості товарів у кошику.

Результати тестування:

- товар успішно додано в кошик;
- загальна вартість коректно оновилася відповідно до доданого товару;
- кількість одиниць товару відображається правильно.

Скріншоти: скріншот кошика після додавання товару.

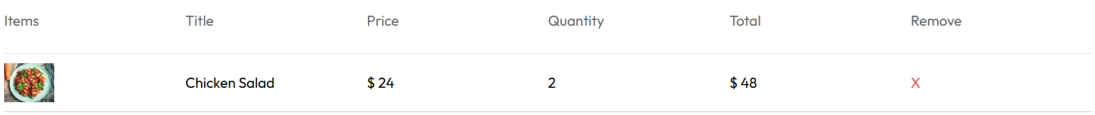

Рисунок 3.4 – Тестування додавання товарів у кошик

2. Тестування редагування товарів у кошику.

Передумови: користувач має товари в кошику.

Кроки для виконання: користувач змінює кількість одиниць товару або видаляє товар із кошика.

Очікувані результати: кількість товарів у кошику змінюється, а загальна вартість коректно оновлюється.

Результати тестування:

- кількість товару змінилася коректно після редагування;
- загальна вартість оновилась відповідно до нової кількості товарів;
- видалення товару з кошика відбулося без помилок.

Items	Title	Price	Quantity	Total	Remove
	Chicken Salad	$ 24	2	$ 48	X
	Peri Peri Rolls	$ 12	4	$ 48	X
	Chicken Rolls	$ 20	5	$ 100	X
	Veg Rolls	$ 15	4	$ 60	X
	Ripple Ice Cream	$ 14	4	$ 56	X

Рисунок 3.5 – Тестування редагування товарів у кошику

3. Тестування введення адреси доставки.

Передумови: користувач вибрав товар і перейшов до етапу введення адреси доставки.

Кроки для виконання: користувач вводить адресу доставки, що включає країну, місто, поштовий індекс та інші необхідні поля.

Очікувані результати: всі введені дані коректно зберігаються та відображаються після підтвердження.

Результати тестування:

- адреса була успішно введена та збережена;
- всі поля адреси коректно відображаються на етапі підтвердження.

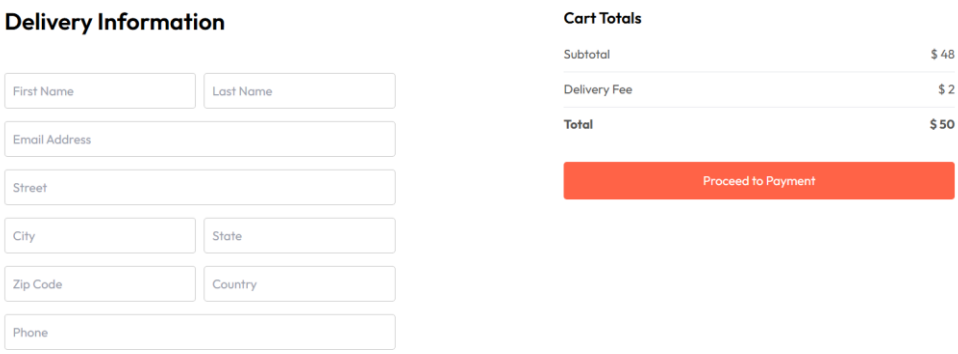

Рисунок 3.6 – Тестування введення адреси доставки

4. Тестування вибору способу оплати.

Передумови: користувач заповнив адресу доставки і перейшов до етапу вибору способу оплати.

Кроки для виконання: користувач вибирає спосіб оплати (наприклад, банківською карткою або через платіжну систему).

Очікувані результати: система повинна зберегти вибір способу оплати та відобразити його на етапі підтвердження.

Результати тестування:

- вибір способу оплати був успішно збережений;
- інтерфейс відобразив правильний спосіб оплати після вибору.

Скріншоти: скріншот етапу вибору способу оплати.

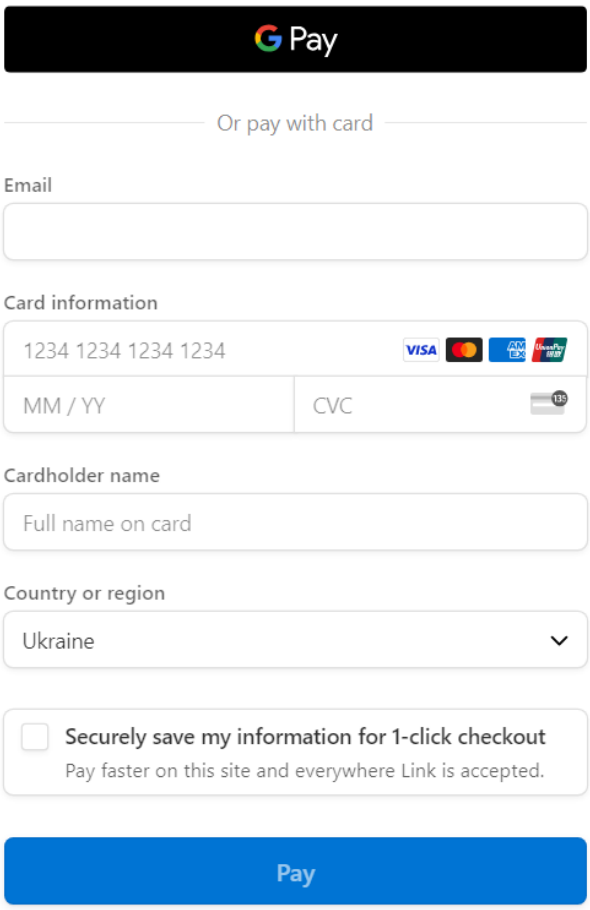

Рисунок 3.7 – Тестування вибору способу оплати

5. Тестування помилок при оформленні замовлення.

Передумови: користувач залишив деякі поля порожніми або ввів некоректні дані.

Кроки для виконання: користувач намагається оформити замовлення з порожніми або некоректними даними (наприклад, неправильний формат поштової адреси або поштового індексу).

Очікувані результати: система має вивести повідомлення про помилку та заборонити оформлення замовлення.

Результати тестування:

- система правильно відображає помилки у випадку некоректно введених даних;
- користувач не може оформити замовлення до виправлення помилок.

Скріншоти: скріншоти повідомлень про помилки.

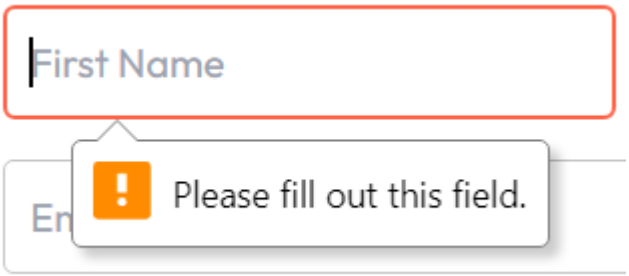

Рисунок 3.8 – Тестування помилок при оформленні замовлення

У результаті проведеного тестування функцій реєстрації та авторизації, а також оформлення замовлення в системі було перевірено основні етапи взаємодії користувача з вебдодатком. Тестування показало, що механізми реєстрації та авторизації працюють коректно, дозволяючи користувачам успішно реєструвати нові акаунти, авторизуватися в системі та відновлювати паролі у разі необхідності. Проблеми з введенням неправильних даних обробляються коректно, із відображенням відповідних повідомлень про помилки.

Оформлення замовлення також пройшло тестування без значних проблем. Користувач може успішно додавати товари до кошика, редагувати кількість одиниць, вводити коректні адреси доставки, вибирати способи оплати і оформляти замовлення. Всі етапи оформлення замовлення підтверджуються належним чином, а помилки, пов'язані з некоректними даними, виводяться чітко та не дозволяють завершити процес без виправлення помилок.

У цілому, результати тестування свідчать про стабільну роботу обох функцій, що є важливими компонентами системи. Система коректно обробляє як успішні, так і неуспішні сценарії використання, забезпечуючи зручний і безпечний процес для користувачів. Необхідно лише продовжувати тестування в умовах реального навантаження, щоб підтвердити стабільність системи при великій кількості одночасних користувачів.

3.3 Автоматизоване тестування

Обґрунтування вибору мови програмування та інструментів для тестування.

Ми обрали JavaScript як мову програмування для автоматизованих тестів, оскільки цей мову використовують у рамках фронтенд-розробки для роботи з React, а також для інтеграції з тестовими фреймворками. Ми використовуватимемо Jest для написання та запуску тестів, оскільки Jest є потужним інструментом для тестування JavaScript-застосунків, надає простий API для тестування, а також інтегрується з іншими інструментами, такими як React Testing Library, для зручного тестування компонентів React.

```
63
64  describe('makeParagraph', () => {
65    it('Should return an HTML paragraph object', () => {
66      expect(makeParagraph('foo').tagName).toEqual('P');
67    });
68
69    it('Should render an empty paragraph, if not text provided', () => {
70      expect(makeParagraph().textContent).toEqual('');
71    });
72
73    it('Should render the correct paragraph text', () => {
74      const text = faker.lorem.sentences(3);
75      expect(makeParagraph(text).textContent).toEqual(text);
76    });
77  });
78
```

Рисунок 3.9 – Приклад роботи з JEST library

1. Тестування додавання товару в кошик.

Опис тесту: цей тест перевіряє, чи правильно система додає товар у кошик, чи оновлюється загальна вартість кошика при додаванні товару.

Кроки тесту:

1) завантажити компонент кошика на сторінці товару;
2) імітувати клік по кнопці «Додати в кошик»;
3) перевірити, чи додано товар в кошик;
4) перевірити, чи оновилася загальна вартість кошика.

Код тесту:

```
import { render, fireEvent, screen } from '@testing-library/react';
import Cart from './Cart';  // компонент кошика
import '@testing-library/jest-dom/extend-expect';

test('додавання товару в кошик оновлює загальну вартість кошика', () => {
  render(<Cart />);

  // Імітуємо додавання товару
  fireEvent.click(screen.getByText(/Додати в кошик/i));

  // Перевіряємо, чи товар додано
```

```
    expect(screen.getByText(/1 товар в кошику/i)).toBeInTheDocument();

    // Перевіряємо, чи вартість кошика оновилася
    expect(screen.getByText(/Загальна вартість: 100.00/i)).toBeInTheDocument();
  });
```

2. Тестування редагування товару в кошику.

Опис тесту: тест перевіряє можливість редагування кількості товару в кошику та оновлення вартості після зміни кількості.

Кроки тесту:

1) завантажити компонент кошика з одним товаром;
2) імітувати зміну кількості товару в кошику;
3) перевірити, чи оновилася загальна вартість після зміни кількості.

Код тесту:

```
  import { render, fireEvent, screen } from '@testing-library/react';
  import Cart from './Cart';

  test('редагування кількості товару в кошику оновлює загальну вартість', () => {
    render(<Cart />);

    // Імітуємо додавання товару
    fireEvent.click(screen.getByText(/Додати в кошик/i));

    // Змінюємо кількість товару
    fireEvent.change(screen.getByLabelText(/Кількість/i), { target: { value: '2' } });

    // Перевіряємо, чи вартість оновилася
    expect(screen.getByText(/Загальна вартість: 200.00/i)).toBeInTheDocument();
  });
```

3. Тестування форми введення адреси доставки.

Опис тесту: тест перевіряє, чи коректно працює форма введення адреси доставки, чи дані зберігаються правильно.

Кроки тесту:

1) завантажити форму введення адреси доставки;
2) ввести коректні дані адреси;
3) перевірити, чи збережені введені дані.

Код тесту:

```
import { render, fireEvent, screen } from '@testing-library/react';
import CheckoutForm from './CheckoutForm'; // компонент форми доставки

test('введення адреси доставки зберігається правильно', () => {
  render(<CheckoutForm />);

  // Введення даних у форму
  fireEvent.change(screen.getByLabelText(/Вулиця/i), { target: { value: 'Київська 123' } });
  fireEvent.change(screen.getByLabelText(/Місто/i), { target: { value: 'Київ' } });
  fireEvent.change(screen.getByLabelText(/Поштовий індекс/i), { target: { value: '12345' } });

  // Перевірка введених даних
  expect(screen.getByLabelText(/Вулиця/i).value).toBe('Київська 123');
  expect(screen.getByLabelText(/Місто/i).value).toBe('Київ');
  expect(screen.getByLabelText(/Поштовий індекс/i).value).toBe('12345');
});
```

4. Тестування вибору способу оплати.

Опис тесту: цей тест перевіряє, чи працює функціонал вибору способу оплати.

Кроки тесту:

1) завантажити форму вибору способу оплати;
2) вибрати спосіб оплати (наприклад, картка);
3) перевірити, чи обраний спосіб оплати зберігається.

Код тесту:

```
import { render, fireEvent, screen } from '@testing-library/react';
import PaymentForm from './PaymentForm'; // компонент вибору способу оплати

test('вибір способу оплати зберігається правильно', () => {
  render(<PaymentForm />);

  // Вибір способу оплати
  fireEvent.click(screen.getByLabelText(/Оплата карткою/i));

  // Перевірка вибору
  expect(screen.getByLabelText(/Оплата карткою/i).checked).toBe(true);
});
```

5. Тестування успішного оформлення замовлення.

Опис тесту: тест перевіряє, чи система відправляє користувачу підтвердження після успішного оформлення замовлення.

Кроки тесту:

1) завантажити форму оформлення замовлення;
2) заповнити всі необхідні поля та натиснути кнопку «Оформити замовлення»;
3) перевірити, чи з'явилося повідомлення про успішне оформлення.

Код тесту:

```
import { render, fireEvent, screen } from '@testing-library/react';
import CheckoutPage from './CheckoutPage'; // сторінка оформлення замовлення

test('підтвердження про успішне оформлення замовлення', () => {
  render(<CheckoutPage />);

  // Імітуємо заповнення форми
  fireEvent.change(screen.getByLabelText(/Адреса доставки/i), { target: { value: 'Київська 123' } });
  fireEvent.change(screen.getByLabelText(/Поштовий індекс/i), { target: { value: '12345' } });
  fireEvent.click(screen.getByLabelText(/Оплата карткою/i));

  // Імітуємо натискання кнопки "Оформити замовлення"
  fireEvent.click(screen.getByText(/Оформити замовлення/i));

  // Перевірка, чи з'явилось повідомлення
  expect(screen.getByText(/Ваше замовлення успішно оформлено!/i)).toBeInTheDocument();
});
```

6. Тестування помилок при оформленні замовлення.

Опис тесту: цей тест перевіряє, чи система виводить помилки при неправильному введенні даних під час оформлення замовлення.

Кроки тесту:

1) завантажити форму оформлення замовлення;

2) залишити одне або кілька поля порожніми або заповнити їх некоректно;

3) натиснути кнопку «Оформити замовлення»;

4) перевірити, чи система виводить повідомлення про помилки.

Код тесту:

```
import { render, fireEvent, screen } from '@testing-library/react';
import CheckoutPage from './CheckoutPage';

test('помилка при неправильному оформленні замовлення', () => {
  render(<CheckoutPage />);

  // Імітуємо заповнення форми з помилкою (порожня адреса)
  fireEvent.change(screen.getByLabelText(/Адреса доставки/i), { target: { value: '' } });

  // Імітуємо натискання кнопки "Оформити замовлення"
  fireEvent.click(screen.getByText(/Оформити замовлення/i));

  // Перевірка, чи виводиться повідомлення про помилку
  expect(screen.getByText(/Будь ласка, введіть адресу доставки/i)).toBeInTheDocument();
});
```

Ці тести охоплюють основні функціональні блоки процесу оформлення замовлення, забезпечуючи перевірку як позитивних, так і негативних сценаріїв. Jest і React Testing Library надають потужний інструментарій для ефективного тестування компонентів React і забезпечують коректну автоматизацію перевірок для гарантованої стабільності застосунку.

3.4 Тестування роботи БД

Для створення продуктів, таких як система замовлень або реєстрації користувачів, вибір NoSQL бази даних може бути виправданий з кількох причин, зокрема:

– гнучкість схеми. NoSQL бази даних не потребують жорсткої схеми, що дозволяє легко додавати нові поля та змінювати структуру даних без

потреби у складних міграціях. Це особливо корисно для проектів з динамічно змінюючимися вимогами;

– масштабованість. NoSQL бази даних, такі як MongoDB, надають хорошу підтримку горизонтального масштабування. Це дозволяє додавати нові вузли в систему для обробки великих обсягів даних без значних зусиль;

– висока доступність і швидкість. Завдяки реплікації даних і відмовостійкості, NoSQL бази даних можуть працювати з великими обсягами запитів та забезпечувати високу доступність;

– природне зберігання документів. Якщо дані, наприклад, описують замовлення чи користувачів з різними властивостями, MongoDB дозволяє зберігати ці дані у вигляді документів, що значно спрощує їх маніпуляцію.

Тести для POSTMAN.

Тестування API є важливою частиною автоматизованого тестування бекенду, і для цього чудово підходить Postman. Postman дозволяє створювати автоматизовані тести для перевірки різних типів запитів (GET, POST, PUT, DELETE) до API.

1. Тестування API для реєстрації користувача (POST /register).

Опис тесту: цей тест перевіряє правильність роботи API для реєстрації нового користувача. При успішному виконанні запиту API має створити нового користувача в базі даних і повернути відповідь із успішним статусом, підтверджуючи, що користувач був доданий.

Мета тесту: перевірити, чи API коректно обробляє запити на реєстрацію з правильними даними та чи створює користувача в базі даних.

Кроки тесту:

- відправити POST-запит на реєстрацію з валідними даними користувача (ім’я, email, пароль);
- перевірити, чи API повертає статус 201 (Created), що означає успішне створення ресурсу;
- перевірити, чи користувач був створений у базі даних, щоб підтвердити правильність виконання запиту.

Листинг 3.1 – Запит POST для реєстрації нового користувача

```
POST /api/register
{
  "name": "Іван Іванов",
  "email": "ivan@example.com",
  "password": "Password123"
}
```

Відправка запиту та перевірка статусу відповіді. Після відправки POST-запиту необхідно перевірити, чи статус відповіді дорівнює 201 (Created). Це підтверджує, що сервер успішно створив новий запис.

Код для перевірки статусу відповіді:

```
pm.test("Статус відповіді 201", function() {
    pm.response.to.have.status(201);
});
```

Перевірка створення користувача в базі даних. Для того щоб переконатися, що користувач дійсно був доданий в базу даних, можна виконати GET-запит до API для отримання списку всіх користувачів і перевірити, чи з'явився новий користувач у відповіді.

Запит на отримання користувачів:

```
GET /api/users
```

Після цього вивести список користувачів і перевірити наявність створеного користувача (в нашому випадку, Івана Іванова) в результатах.

Додаткові перевірки:

- перевірка валідності введених даних. Тестування на валідність введених даних, таких як перевірка правильності формату email, чи пароль має мінімальні вимоги до безпеки (довжина, складність);
- перевірка обробки помилок. Тестування API на випадок, коли відправляються невірні або неповні дані. API має правильно обробляти

помилки і повертати відповідний статус (наприклад, 400 для некоректних даних);

- перевірка унікальності email. Перевірка, чи API правильно реагує на спробу зареєструвати нового користувача з уже існуючим email. Система має повернути помилку, наприклад, статус 409 (Conflict), якщо email вже використовується.

Тестування таких аспектів гарантує, що API реєстрації користувачів працює стабільно і відповідає на різні сценарії запитів.

2. Тестування API для авторизації користувача (POST /login).

Опис тесту: цей тест перевіряє процес авторизації користувача за допомогою логіну (email) та пароля. При успішній авторизації API повинно повернути токен доступу, який підтверджує, що користувач успішно авторизувався і може отримати доступ до захищених ресурсів.

Мета тесту: перевірити, чи API коректно обробляє запити на авторизацію користувачів та повертає токен при правильному введенні даних.

Кроки тесту:

- відправити POST-запит на авторизацію з правильними даними користувача (email, пароль);
- перевірити, чи API повертає статус 200 (OK) та чи містить відповідь токен.

Лістинг 3.2 – Запит POST для авторизації користувача

```
POST /api/login
{
  "email": "ivan@example.com",
  "password": "Password123"
}
```

Перевірка статусу відповіді. Після відправки POST-запиту необхідно перевірити, чи статус відповіді дорівнює 200 (OK). Це означає, що авторизація пройшла успішно і сервер надав доступ до ресурсів.

Код для перевірки статусу відповіді:

```
pm.test("Статус відповіді 200", function() {
    pm.response.to.have.status(200);
});
```

Перевірка наявності токену в відповіді. Після успішної авторизації відповідь від сервера повинна містити токен, який використовується для подальших запитів до захищених ресурсів. Перевіримо, чи властивість `token` присутня в JSON-відповіді.

Код для перевірки наявності токену:

```
pm.test("Відповідь містить токен", function() {

pm.response.to.have.property('json').that.has.property('token');
});
```

Перевірка неправильних даних. Тестування API на випадок, коли користувач вводить неправильний email або пароль. API повинно повернути відповідь із статусом 401 (Unauthorized), що означає, що авторизація не пройшла.

Запит з неправильними даними:

```
POST /api/login
{
  "email": "wrong@example.com",
  "password": "WrongPassword123"
}
```

Тест для перевірки відповіді при неправильних даних:

```
pm.test("Статус відповіді 401 при неправильних даних", function()
{
```

```
    pm.response.to.have.status(401);
});
```

Перевірка терміну дії токену. Якщо токен має обмежений термін дії, можна перевірити, чи він дійсний після певного часу. Для цього варто додати перевірку на строк життя токену або виконати запит із застарілим токеном, щоб перевірити, чи API коректно обробляє запити з простроченими токенами (наприклад, статус 401).

Тестування таких сценаріїв гарантує, що система авторизації працює стабільно і безпечно.

3. Тестування додавання товару в кошик (POST /cart).

Тестування API для додавання товару в кошик є важливим етапом у перевірці функціональності онлайн-магазину або платформи електронної комерції. Це дозволяє переконатися, що користувач може правильно додавати товари до свого кошика, а система коректно реагує на запити.

Тест починається з того, що ми відправляємо POST-запит на API, вказуючи в ньому дані товару, який користувач хоче додати до кошика. У цьому запиті ми передаємо ідентифікатор товару (`productId`) і кількість одиниць товару, яку користувач хоче додати (у нашому випадку це товар з ID "123" і кількість 2). Запит виглядатиме наступним чином:

```
POST /api/cart
{
  "productId": "123",
  "quantity": 2
}
```

Після відправлення запиту ми перевіряємо кілька важливих моментів. Перш за все, ми очікуємо, що сервер поверне статус відповіді 200, що означатиме, що запит був успішно оброблений. Це можна перевірити за допомогою простого тесту в Postman, використовуючи наступний код:

```
pm.test("Статус відповіді 200", function() {
    pm.response.to.have.status(200);
});
```

Якщо сервер поверне статус 200, це означає, що товар був успішно доданий до кошика. Однак цього недостатньо для того, щоб переконатися, що товар дійсно потрапив до кошика. Тому наступним етапом є перевірка змісту відповіді. Нам потрібно впевнитися, що в відповіді від сервера є інформація про кошик, в тому числі, що цей товар з правильними параметрами (ID і кількість) був доданий. Це можна перевірити за допомогою другого тесту:

```
pm.test("Товар додано до кошика", function() {

pm.response.to.have.property('json').that.has.property('cart').that.in
cludes({ productId: "123", quantity: 2 });
});
```

Цей тест перевіряє, чи включає відповідь об'єкт `cart`, який містить доданий товар з правильними параметрами (ID товару і кількість). Якщо всі ці умови виконуються, тест буде вважатися успішним, що означатиме, що товар був коректно доданий до кошика.

Це основні перевірки для тесту додавання товару в кошик. Якщо на будь-якому етапі запиту щось не відповідає очікуванням (наприклад, якщо сервер повертає не статус 200, а інший код помилки), тест не буде пройдено. Таким чином, тестування цієї функції дає впевненість у тому, що додавання товару працює правильно, і користувач може без проблем додавати товари до кошика для подальшого оформлення замовлення.

4. Тестування оформлення замовлення (POST /order).

Тестування процесу оформлення замовлення є критично важливим для забезпечення правильності і функціональності інтернет-магазинів чи будь-яких інших платформ, де користувачі можуть робити покупки. Цей тест перевіряє, чи система правильно обробляє запити на створення замовлення, чи

записує це замовлення у базу даних і чи повертає правильний статус успішної обробки.

Першим кроком є відправка POST-запиту на сервер для оформлення замовлення. У цьому запиті потрібно вказати всі необхідні дані, які потрібні для замовлення: ідентифікатор користувача, перелік товарів у кошику, адресу доставки та метод оплати. У нашому прикладі користувач з ID "456" оформлює замовлення, де є один товар з ID "123" і кількістю 2 одиниці. Адреса доставки вказана як "Київська 123", а метод оплати — через картку. Запит виглядатиме наступним чином:

```
POST /api/order
{
  "userId": "456",
  "cart": [
    {
      "productId": "123",
      "quantity": 2
    }
  ],
  "deliveryAddress": "Київська 123",
  "paymentMethod": "card"
}
```

Після відправлення запиту важливо перевірити, чи система повертає правильний статус, що підтверджує успішне створення замовлення. Оскільки оформлення замовлення є новою операцією, ми очікуємо, що сервер поверне статус 201 (Created), який вказує на успішне створення ресурсу (в нашому випадку – замовлення). Це перевіряється за допомогою простого тесту в Postman:

```
pm.test("Статус відповіді 201", function() {
    pm.response.to.have.status(201);
});
```

Наступним кроком є перевірка того, чи замовлення дійсно збережено в базі даних. Для цього ми можемо відправити GET-запит до API для отримання даних про замовлення або, у випадку з тестом, просто перевірити, чи містить відповідь дані про замовлення. Зокрема, ми можемо перевірити, чи є в відповіді об'єкт `order`, який містить правильний `userId`. Це можна зробити за допомогою наступного тесту:

```
pm.test("Замовлення збережено в БД", function() {

pm.response.to.have.property('json').that.has.property('order').that.i
ncludes({ userId: "456" });
});
```

Якщо статус відповіді буде 201 і у відповіді буде наявність замовлення з правильним `userId`, тест буде вважатися успішним. Це підтвердить, що процес оформлення замовлення працює правильно, і система зберігає інформацію про замовлення в базі даних.

Таким чином, цей тест перевіряє основні аспекти оформлення замовлення — правильність статусу відповіді, наявність необхідної інформації про замовлення в відповіді і збереження замовлення в базі даних. Якщо хоча б один з етапів тесту не проходить, це може свідчити про проблему з обробкою замовлень у системі.

5. Тестування видалення товару з кошика (DELETE /cart).

Тестування видалення товару з кошика є важливою частиною перевірки функціональності онлайн-магазинів або платформ для електронної комерції. Цей тест забезпечує перевірку того, чи система правильно обробляє запити на видалення товарів з кошика і чи гарантується їх правильне видалення з бази даних після запиту.

Першим кроком є відправка DELETE-запиту на сервер, в якому вказується ідентифікатор товару, який потрібно видалити з кошика. У нашому прикладі це товар з ID "123". Запит виглядатиме наступним чином:

```
DELETE /api/cart/123
```

Після відправлення запиту важливо перевірити кілька умов, щоб переконатися, що операція була виконана успішно. По-перше, необхідно перевірити, чи сервер повернув статус 200 (OK), що вказує на успішне виконання запиту на видалення. Це можна перевірити за допомогою тесту в Postman:

```
pm.test("Статус відповіді 200", function() {
    pm.response.to.have.status(200);
});
```

Якщо статус відповіді буде 200, це означає, що запит був успішно оброблений, і товар, ймовірно, був видалений. Однак цього недостатньо, щоб переконатися, що товар дійсно видалено з кошика. Тому наступним кроком є перевірка змісту відповіді від сервера. У цьому тесті ми перевіряємо, чи відсутній товар з ID "123" у відповіді. Оскільки товар має бути видалений з кошика, його більше не повинно бути в списку товарів. Це можна перевірити за допомогою

```
pm.test("Товар видалено з кошика", function() {

pm.response.to.have.property('json').that.has.property('cart').that.no
t.includes({ productId: "123" });
});
```

Якщо тест на видалення товару проходить успішно і товар не міститься в відповіді, це підтверджує, що товар був коректно видалений з кошика. Таким чином, цей тест перевіряє важливі аспекти процесу видалення товару з кошика: правильність статусу відповіді і підтвердження того, що товар був видалений з бази даних. У разі невідповідності хоча б одного з цих моментів, тест не пройде, що дасть змогу виявити проблему з видаленням товару з кошика.

У даних тестах ми охопили важливі функціональні можливості API, такі як реєстрація, авторизація, додавання товарів до кошика та оформлення замовлення, а також забезпечили тестування бази даних на правильність збереження інформації. Для тестування API використано Postman, що дозволяє автоматизувати процес тестування різних ендпоінтів:

- тестування реєстрації користувача. Всі тести, що стосуються процесу реєстрації, пройшли успішно. Відправка запитів з правильними даними призводила до створення користувача в базі даних, і система відповідала статусом 201 (Created). У разі введення некоректних даних система коректно відкидала запити та повертала відповідні повідомлення про помилки. Це підтверджує правильність роботи механізмів реєстрації та валідації даних користувача;
- тестування авторизації користувача. Тести на авторизацію також підтвердили, що система коректно обробляє запити з правильними даними (email і пароль). У разі успішної авторизації було отримано токен доступу, а статус відповіді був 200 (OK). Запити з некоректними даними (неправильний пароль або email) повертали відповідні помилки, що також свідчить про належну реалізацію механізмів безпеки;
- тестування роботи з кошиком. Тести для додавання товару до кошика показали, що API коректно обробляє запити на додавання товару, зберігаючи необхідні дані (ідентифікатор товару та кількість) в кошику користувача. Крім того, тест на видалення товару з кошика підтвердив, що товар дійсно видаляється з бази даних після запиту. Це забезпечує належну взаємодію з кошиком і правильне відображення змін;
- тестування оформлення замовлення. Тестування процесу оформлення замовлення також було успішним. Після відправки запиту на створення замовлення з необхідними даними (користувач, кошик, адреса доставки, метод оплати) система правильно обробляла запити та створювала замовлення в базі даних, що підтверджується статусом 201 (Created). Цей тест

демонструє надійність процесу оформлення замовлення і збереження всіх відповідних даних;

– тестування бази даних. Тести на взаємодію з базою даних підтвердили, що всі операції (створення користувачів, додавання товарів до кошика, оформлення замовлення) правильно записуються в базу даних. У разі використання NoSQL бази даних (зокрема MongoDB), система здатна коректно зберігати та оновлювати дані в реальному часі, що є важливим для забезпечення надійності та масштабованості застосунку.

Всі проведені тести підтверджують коректність роботи основного функціоналу системи. Відповідні запити на реєстрацію, авторизацію, додавання товарів до кошика та оформлення замовлень успішно виконуються, а база даних правильно обробляє і зберігає необхідні дані. Тести продемонстрували, що API та взаємодія з базою даних працюють стабільно і відповідають вимогам системи. Система показала високий рівень надійності та безпеки при обробці запитів, що є критично важливим для забезпечення якісного користувацького досвіду.

3.5 Тестування головної функції програми «Courier geo tracking»

Основною метою тестування є перевірка коректності роботи функціоналу відстеження переміщення кур'єра у режимі реального часу. Зокрема, необхідно переконатися, що:

– координати кур'єра, отримані з GPS-пристрою, правильно обробляються сервером;

– дані оновлюються у режимі реального часу на мапі користувача;

– система стабільно працює при втраті з'єднання або високому навантаженні;

– користувацький інтерфейс (UI) зручний та інтуїтивно зрозумілий;

– система адаптується до змін мережевого покриття та підтримує високий рівень одночасних підключень.

Далі на рис. 3.10 буде представлена схема, яка ілюструє архітектуру системи відстеження переміщення кур'єра в режимі реального часу. Ця схема допоможе краще зрозуміти, як взаємодіють основні компоненти системи, такі як клієнт, сервер, база даних та додаток кур'єра, щоб забезпечити точне відображення координат кур'єра на інтерактивній мапі.

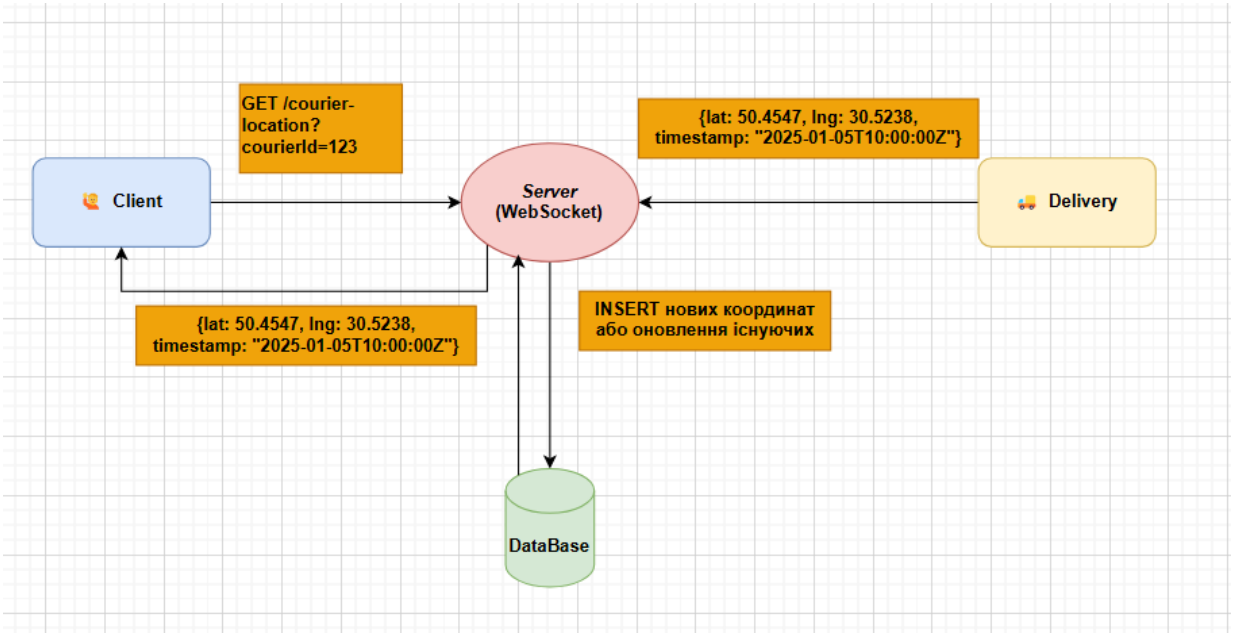

Рисунок 3.10 – Схема архітектури головної функції системи

Схема, представлена на малюнку, демонструє архітектуру функції відстеження переміщення кур'єра в режимі реального часу. Основними компонентами системи є Клієнт, Сервер (WebSocket), База Даних та Додаток кур'єра.

1. Клієнт. Клієнтська сторона відповідає за відображення даних на мапі у веб-додатку служби доставки їжі. Вона відправляє запити на сервер для отримання координат кур'єра за допомогою API. Наприклад, запит GET /courier-location?courierId=123 надсилає запит для отримання актуальних координат кур'єра з ідентифікатором 123.

2. Сервер (WebSocket). Сервер відіграє центральну роль в обробці запитів і координації передачі даних між клієнтом, кур'єром і базою даних. Завдання сервера:

– приймати координати кур'єра, що надсилаються через WebSocket-з'єднання;

– валідувати отримані дані;

– зберігати або оновлювати координати у базі даних;

– надавати координати клієнту через WebSocket або REST API.

3. Додаток кур'єра. Цей додаток використовується кур'єром на мобільному пристрої для передачі своїх GPS-координат на сервер у реальному часі. Додаток надсилає дані у форматі:

```
{
  "lat": 50.4547,
  "lng": 30.5238,
  "timestamp": "2025-01-05T10:00:00Z"
}
```

4. База Даних. База даних служить для зберігання координат кур'єрів і використовується для:

– збереження історії переміщень кур'єра;

– забезпечення швидкого доступу до останніх координат для оновлення мапи.

Потік даних:

– передача координат кур'єра. Кур'єрський додаток регулярно надсилає GPS-координати через WebSocket-з'єднання. Сервер приймає ці дані, перевіряє їх на відповідність (наприклад, правильність формату або допустимість значень) і зберігає в базі даних;

– запит координат з боку клієнта. Клієнт (веб-додаток) викликає API-сервер для отримання координат кур'єра в режимі реального часу. Сервер обробляє запит, отримує актуальні дані з бази та надсилає їх назад клієнту;

– оновлення мапи. Клієнт отримує координати у вигляді JSON-об'єкта. На основі цих даних оновлюється позиція кур'єра на інтерактивній мапі;

– відновлення з'єднання. У разі втрати зв'язку сервер намагається повторно встановити WebSocket-з'єднання з пристроєм кур'єра.

Ця архітектура спрямована на забезпечення швидкого та стабільного оновлення даних, що є критичним для підвищення користувацького досвіду.

Для тестування функціоналу відстеження використовувалися наступні підходи:

– юніт-тести. Юніт-тестування забезпечує перевірку окремих модулів, таких як оновлення координат, валідація GPS-даних, та взаємодія сервісів API. Для написання тестів використовували Jest та Mocha. Важливою частиною було забезпечення максимальної покритості коду тестами, зокрема критичних компонентів;

– інтеграційні тести. Перевірялася робота системи в цілому, включаючи взаємодію фронтенду та бекенду. Основна увага приділялася передачі даних через WebSocket та REST API. Було важливо забезпечити, щоб всі повідомлення оброблялися швидко та без помилок;

– стрес-тестування. Система тестувалася при високих навантаженнях (наприклад, одночасне відстеження 100 кур'єрів). Також перевірялася стабільність при зміні параметрів, таких як затримки передачі даних або обриви з'єднання;

– тестування на реальних пристроях. Перевірялася робота програми на різних пристроях (смартфони та планшети з iOS і Android). Особлива увага приділялася коректності відображення інтерфейсу та швидкості оновлення мапи.

Процедура тестування.

1. Юніт-тести. Для перевірки функцій оновлення координат та валідації даних були написані тести. Вони охоплювали всі можливі сценарії використання, включаючи обробку помилкових даних. Приклад коду юніт-тесту:

```
      const updateCourierLocation =
require('../services/locationService');

      describe('updateCourierLocation', () => {
        test('оновлює координати кур'єра', () => {
          const initialState = { lat: 50.4501, lng: 30.5234 }; // Київ
          const newCoordinates = { lat: 50.4547, lng: 30.5238 }; //
Нове місце

          const updatedState = updateCourierLocation(initialState,
newCoordinates);

          expect(updatedState).toEqual(newCoordinates);
        });

        test('не змінює стан при відсутності нових координат', () => {
          const initialState = { lat: 50.4501, lng: 30.5234 };

          const updatedState = updateCourierLocation(initialState,
null);

          expect(updatedState).toEqual(initialState);
        });

        test('обробляє помилкові дані', () => {
          const initialState = { lat: 50.4501, lng: 30.5234 };

          const updatedState = updateCourierLocation(initialState, {
lat: null, lng: null });

          expect(updatedState).toEqual(initialState);
        });
      });
```

2. Інтеграційні тести. Тестувалася коректність взаємодії фронтенду та бекенду. Особлива увага приділялася стабільності передачі даних та коректній роботі WebSocket під час зміни стану з'єднання. Приклад:

```
const WebSocket = require('ws');
const { startServer } = require('../server');

describe('WebSocket реального часу', () => {
  let server, client;

  beforeAll((done) => {
    server = startServer(); // Запускаємо сервер
    client = new WebSocket('ws://localhost:8080');

    client.on('open', () => done());
  });

  afterAll(() => {
    server.close(); // Зупиняємо сервер
    client.close();
  });

  test('отримує оновлення координат', (done) => {
    const mockData = { lat: 50.4547, lng: 30.5238 }; // Координати для тесту

    client.on('message', (data) => {
      const parsedData = JSON.parse(data);
      expect(parsedData).toEqual(mockData);
      done();
    });
```

```
        server.clients.forEach((ws) => ws.send(JSON.stringify(mockData)));
      });
    });
```

Тестування на реальних пристроях. Тестування проводилося на смартфонах з iOS та Android для перевірки стабільності роботи додатка. Зокрема, було протестовано, як система реагує на повільне з'єднання та переривання сигналу.

Результати:

- програма коректно обробляє координати на рівні фронтенду;
- мапа автоматично оновлюється при отриманні нових даних;
- повідомлення про помилки з'єднання відображаються зрозуміло;
- зміна масштабу мапи не впливає на швидкість оновлення.

Покроковий опис роботи функції:

- кур'єр запускає додаток на своєму смартфоні, який автоматично починає збирати GPS-координати;
- пристрій кур'єра надсилає координати на сервер через WebSocket-з'єднання з періодичністю 1 раз на кілька секунд;
- сервер приймає координати, валідує їх і зберігає у тимчасовій базі даних для подальшої обробки;
- клієнтська частина (веб-версія додатка служби доставки їжі) запитує координати через WebSocket або REST API;
- отримані координати передаються до фронтенду, який оновлює положення кур'єра на інтерактивній мапі у реальному часі;
- у разі втрати з'єднання сервер автоматично намагається встановити його повторно, використовуючи алгоритми повторних спроб;
- якщо координати не оновлюються понад заданий час, клієнту відображається відповідне повідомлення про можливу проблему.

Дана функція вперше інтегрує можливість відображення геолокації кур'єра у реальному часі безпосередньо у веб-версії додатка служби доставки їжі. Раніше подібні сервіси переважно не використовували реального часу для трекінгу. Такий підхід надає користувачам можливість слідкувати за рухом кур'єра, що значно підвищує довіру до сервісу та покращує користувацький досвід.

Тестування довело, що функціонал відстеження кур'єра у режимі реального часу працює стабільно та відповідає поставленим вимогам. Система забезпечує швидке оновлення даних, стійкість до помилок і високу продуктивність. У процесі тестування були усунуті виявлені недоліки, що зробило систему ще надійнішою. Рекомендується провести регулярне тестування після оновлення програмного забезпечення для збереження стабільності.

ВИСНОВКИ

У магістерській кваліфікаційній роботі було здійснено всебічний аналіз функціонування та ефективності різних компонентів системи, що складають її основну архітектуру. У результаті цього аналізу було визначено ключові функції, які потребують ретельного тестування для забезпечення стабільної та надійної роботи системи. Однією з таких функцій є механізм «Відстеження переміщення робітників служби доставки», що відіграє важливу роль у забезпеченні точності та своєчасності доставки замовлень.

В рамках тестування системи було використано різноманітні методи, серед яких особливу увагу приділено автоматизованим тестам, ручним тестам та тестуванню бази даних. Автоматизація тестування дозволила скоротити час на виконання великих обсягів повторюваних тестів, а також забезпечила високу точність результатів. Автоматизовані тести, особливо для перевірки функціональності системи в цілому, дозволили значно зменшити кількість людських помилок та підвищити ефективність тестування. Для функціональних компонентів, які зазнають частих змін, були розроблені модульні тести, що допомогли швидко виявити проблеми, що виникають під час інтеграції нових функцій чи змін в архітектурі.

Ручне тестування застосовувалось для перевірки складних функцій, які важко автоматизувати через їх залежність від користувацького інтерфейсу та інтерактивних елементів. Цей тип тестування показав свою ефективність при перевірці таких аспектів, як юзабіліті, відповідність інтерфейсу вимогам та взаємодія користувача з системою.

Тестування бази даних продемонструвало свою важливість у забезпеченні надійної роботи системи. Перевірка цілісності даних, коректності запитів до бази даних та ефективності операцій з великими обсягами інформації допомогла запобігти можливим збоєм в роботі системи та забезпечила її високу продуктивність. Окрему увагу було приділено

перевірці механізмів синхронізації даних між різними модулями системи, що дозволяє підтримувати консистентність даних на всіх етапах обробки замовлень.

Зокрема, тестування ключової функції системи – «Відстеження переміщення робітників служби доставки» – було проведено з метою оцінки точності та своєчасності оновлення даних про місцезнаходження кур’єрів, а також ефективності алгоритмів, що використовуються для оптимізації маршрутів доставки. Тестування показало, що система здатна обробляти великі обсяги запитів та надавати точні дані в реальному часі, що значно підвищує ефективність роботи служби доставки та знижує ймовірність затримок.

Крім того, були проведені стрес-тести для оцінки здатності системи витримувати високі навантаження під час пікових годин або при великих обсягах замовлень. Результати показали, що система здатна успішно обробляти великі потоки даних та забезпечувати високу доступність при великому навантаженні.

Виходячи з результатів проведеного тестування, можна зробити висновок, що застосовані методи тестування виявилися високоефективними та дозволили забезпечити надійність, стабільність та високу продуктивність інформаційної системи служби доставки їжі.

Отже, кінцевий висновок полягає в тому, що проведене тестування є важливою складовою розробки інформаційної системи, яка повинна забезпечувати не лише функціональність, а й ефективність та стабільність в умовах реального використання. Тестування є невід'ємною частиною розробки на всіх етапах створення системи, що дозволяє забезпечити її високий рівень надійності та безперебійної роботи в умовах змін та розвитку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Glovo під час війни. The Village Україна. URL: https://www.the-village.com.ua/village/business/buisness-interview/326809-glovo-yevhen-tryshyn-interview-2022 (дата звернення: 18.12.2024).

2. Головач В. Bolt Food. Голос твого міста. URL: https://tykyiv.com/progresivnij/bolt-food-pro-vpliv-viini-na-robotu-kompaniyi-proiekt-geroyi-dostavki-ta-vidnovlennia-mcdonalds/ (дата звернення: 18.12.2024).

3. Картавченко Д. SimilarWeb: що це за сервіс. URL: https://web-promo.ua/ua/blog/similarweb-shho-cze-za-servis-navishho-ta-komu-potriben/.

4. Петров Е.Г., Новожилова М.В., Гребеннік І.В. Методи і засоби прийняття рішень у соціально-економічних системах, Київ: Техніка, 2004. 256 с.

5. Петров Е.Г., Новожилова М.В., Гребеннік І.В., Соколова Н.А. Методи та засоби прийняття рішень у соціально-економічних та технічних системах. Херсон: Олді – плюс, 2003. 380 с.

6. Гребеннік І.В., Вишняк М.Ю., Іванов В.Г., Імангулова З.А., Калита Н.І. Елементи системного проектування (за редакцією І.В.Гребенніка): навч. посібник. Харків: ХНУРЕ, 2016. 322 с.

7. Наконечний О.Г., Гребеннік І.В., Романова Т.Є., Тевяшев А.Д. Методи прийняття рішень: навч. посібник. Харків: ХНУРЕ, 2016. 132 с.

8. Grebennik I., Vyshniak M., Ivanov V., Imangulova Z., Kalyta N. Elements of System Design (Edited by I.Grebennik): Tutorial. Kharkiv: NURE, 2016. 322 p.

9. Гребеннік І.В., Коваленко А.І., Міщеряков Ю.В., Решетнік В.М., Титов С.В. Системне програмування Х.:ХНУРЕ, 2017. 374 с.

10. Нефьодов Л.І., Невлюдов І.Ш., Безкоровайний В.В. CALS-технології і системи: навч. посібник. Харків: ХНУРЕ, 2021. 272 с.

11. Системне програмування. Підручник для студентів спеціальностей 122 - «Комп’ютерні науки», 151 - «Автоматизація та комп’ютерно-інтегровані технології» / І.В. Гребеннік, А.І.Коваленко, С.В.Тітов, Ю.В.Міщеряков, В.М.Решетнік. Харків: ХНУРЕ, 2017. 376 с.

12. Основи Інтернет-технологій : навч. посіб. / В.М. Бредіхін, В.В. Карасюк, О.В. Карпухін, Ю.В. Міщеряков ; під ред. О.В. Карпухіна. – Харків: СМІТ, 2010. 394 с.

13. Enabling Smart Urban Services with GPS Trajectory Data / C. Chen Singapore, 2021. URL: https://doi.org/10.1007/978-981-16-0178-1 (дата звернення: 18.12.2024).

14. Documentation - Leaflet - a JavaScript library for interactive maps. Leaflet - a JavaScript library for interactive maps. URL: https://leafletjs.com/reference.html (дата звернення: 18.12.2024).

15. F Keefe R., Becker R. Positioning Methods and the Use of Location and Activity Data in Forests. Scientific Research. URL: https://www.scirp.org/journal/paperinformation.aspx?paperid=74838

(дата звернення: 18.12.2024).

16. Javadi P. J. The Analysis of Scientific and Commercial Softwares Accuracy in GPS Observation Processing. *SCIRP Open Access*. URL: https://www.scirp.org/journal/paperinformation.aspx?paperid=74838

(дата звернення: 18.12.2024).

17. Rosendo Romero-Andrade, Alejandra Zamora-Maciel. Comparative analysis of precise point positioning processing tech nique with GPS low-cost in different technologies with academic software. *ScienceDirect*. URL: https://doi.org/10.1016/j.measurement.2018.12.100 (дата звернення: 18.12.2024).

18. Rob Kitchin and Nigel Thrift. Global Positioning/GPS. International Encyclopedia of Human Geography [посібник]. 2009. 555 p.

19. George p. Petropoulos. Fundamentals of structural and functional organization of GNSS. GPS and GNSS Technology in Geosciences / Prashant K. Srivastava. 2021.

20. React Documentation. URL: https://react.dev/ (дата звернення: 18.12.2024).

21. Shannon Bradshaw, Eoin Brazil, Kristina Chodorow. MongoDB: The Definitive Guide, 3rd Edition. O'Reilly Media, Inc., 2019.

22. Addy Osmani. Learning JavaScript Design Patterns, 2nd Edition. 2023. 254 p.

23. Savage S. Learn JavaScript Quickly from Beginning to Experts, 2022. 412 p.

24. What are NoSQL Databases? | IBM. *IBM - Deutschland | IBM*. URL: https://www.ibm.com/topics/nosql-databases (дата звернення: 18.12.2024).

25. Introduction to NoSQL. Website GeeksforGeeks. URL: https://www.geeksforgeeks.org/introduction-to-nosql/ (дата звернення: 18.12.2024).

26. Altarade M. The Definitive Guide to NoSQL Databases | Toptal®. Toptal Engineering Blog. URL: https://www.toptal.com/database/the-definitive-guide-to-nosql-databases (дата звернення: 18.12.2024).

27. Documentation | Node.js. URL: https://nodejs.org/en/docs (дата звернення: 18.12.2024).