МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ
Факультет механіки, енергетики та інформаційних технологій
Кафедра інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

першого (бакалаврського) рівня вищої освіти

на тему:

«РОЗРОБКА СТРУКТУРИ ІНТЕРНЕТ-МАГАЗИНУ ДЛЯ ОПТИМІЗАЦІЇ ВЗАЄМОДІЇ З БАЗОЮ ДАНИХ»

Виконав: здобувач групи ІТ-41
спеціальності 126 «Комп'ютерні науки»
Волч І.В.
(прізвище та ініціали)

Керівник: Смолінський В.Б.
(прізвище та ініціали)

Дубляни-2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Перший (бакалаврський) рівень вищої освіти
Спеціальність 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____
(підпис)
д.т.н., професор, Тригуба А. М.
(вч. звання, прізвище, ініціали)
"____" _____202_ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Волч Ігор Володимирович
(прізвище, ім'я, по батькові)

1. Тема роботи *«Розробка структури інтернет-магазину для оптимізації взаємодії з базою даних»*

керівник роботи *Смолінський В. Б., к.е.н., доцент*

затверджені наказом Львівського НУП *від 27.11.2023 року № 641/к-с*

2. Строк подання студентом роботи *10 червня 2024 року*

3. Вихідні дані до роботи: *характеристика сучасних інформаційних систем; документація до програмного забезпечення, науково-технічна і довідкова література.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Вступ
Теоретичні основи інтернет-магазинів і баз даних
Аналіз існуючих підходів і рішень
Проектування структури інтернет-магазину
Реалізація і тестування
Охорона праці та безпека в надзвичайних ситуаціях
Висновки
Список використаних джерел

5. Перелік графічного матеріалу
Графічний матеріал подається у вигляді презентації

6. Консультанти розділів

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата		Відмітка про виконання
		завдання видав	завдання прийняв	
1, 2, 3,4	*Смолінський В. Б., к.е.н., доцент*			
5	*Городецький І. М., к.т.н., доцент*			

7. Дата видачі завдання *28 листопада 2023 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Відмітка про виконання
1	*Теоретичні основи інтернет-магазинів і баз даних*	*28.11.2023 – 31.12.2023*	
2	*Аналіз існуючих підходів і рішень*	*01.01.2024 – 02.02.2024*	
3	*Проектування структури інтернет-магазину*	*03.02.2024 – 27.02.2024*	
4	*Реалізація і тестування*	*28.02.2024 – 30.04.2024*	
5	*Розгляд питань з охорони праці та безпеки у надзвичайних ситуаціях*	*01.05.2024 – 14.05.2024*	
6	*Завершення оформлення розрахунково-пояснювальної записки та презентаційного матеріалу*	*15.05.2024 – 31.05.2024*	
7	*Завершення роботи в цілому. Підготовка до захисту кваліфікаційної роботи*	*01.06.2024 – 10.06.2024*	

Здобувач _____ *Волч І.В.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Смолінський В. Б.*
(підпис) (прізвище та ініціали)

УДК 004.78

Розробка структури інтернет-магазину для оптимізації взаємодії з базою даних. Волч І. В. Кваліфікаційна робота: Кафедра інформаційних технологій. – Дубляни, Львівський НУП, 2024 р.

75 с. текст. част., 11 табл., 7 рис., 16 літ. джерел

У кваліфікаційній роботі представлено структуру інтернет-магазину, розробленого з використанням Django, React та СУБД MySQL. Показано можливості використання фреймворків для оптимізації взаємодії з базою даних.

Інтернет-магазини користуються попитом, відповідно їх розробка є досить актуальною проблемою, тому робота має важливе практичне значення.

Ключові слова: інтернет-магазин, бази даних, СУБД MySQL, фреймворки, Django, React, кешування

Keywords: web store, databases, MySQL DBMS, frameworks, Django, React, caching

ЗМІСТ

ВСТУП

1. Обґрунтування важливості теми

У сучасному світі інтернет-магазини відіграють ключову роль в електронній комерції, забезпечуючи зручний і швидкий доступ до товарів та послуг для мільйонів споживачів. Розвиток технологій дозволяє споживачам здійснювати покупки з будь-якої точки світу в будь-який час, що суттєво розширює ринки збуту для бізнесів. В умовах зростаючої конкуренції на ринку інтернет-магазинів важливо забезпечити ефективну і безперебійну роботу таких платформ. Центральне місце в цій задачі займає оптимізація взаємодії з базами даних, які є основою для зберігання та обробки великої кількості інформації про товари, користувачів та транзакції.

2. Сучасні тенденції розвитку інтернет-магазинів

- Зростання мобільної комерції (m-commerce): Збільшення кількості покупок, здійснюваних через мобільні пристрої, вимагає адаптації інтернет-магазинів для мобільних платформ.

- Персоналізація: Використання даних для надання персоналізованих рекомендацій і пропозицій стає ключовим фактором успіху.

- Інтеграція з соціальними мережами: Підключення до соціальних платформ для збільшення охоплення аудиторії та впливу на рішення про покупку.

- Штучний інтелект і машинне навчання: Застосування цих технологій для прогнозування попиту, управління запасами та покращення обслуговування клієнтів.

- Багатоканальна (омніканальна) стратегія: Забезпечення узгодженого досвіду покупки через різні канали (онлайн, офлайн, мобільні додатки).

3. Проблематика взаємодії з базами даних у контексті інтернет-магазинів

- Швидкодія і масштабованість: Забезпечення швидкого доступу до даних навіть при великій кількості одночасних користувачів. Проблеми можуть виникати через зростання обсягів даних і необхідність обробки складних запитів у реальному часі.

- Безпека даних: Захист особистих даних користувачів і фінансової інформації від несанкціонованого доступу і кібератак. Важливо дотримуватися стандартів безпеки, таких як PCI DSS.

- Надійність і доступність: Забезпечення безперебійної роботи бази даних з мінімальним часом простою. Важливими є резервне копіювання і відновлення даних.

- Підтримка транзакцій: Гарантування цілісності даних при виконанні транзакцій (замовлення, оплати, повернення). Важливо, щоб всі операції були завершеними і коректними.

- Інтеграція з зовнішніми системами: Потрібна ефективна взаємодія з платіжними системами, системами управління запасами, службами доставки та іншими сторонніми сервісами.

- Аналітика і звітність: Збір та аналіз даних для прийняття бізнес-рішень. Це включає звіти про продажі, поведінку користувачів, ефективність маркетингових кампаній тощо.

- Гнучкість і масштабованість структури: Необхідність у можливості швидкого внесення змін до структури бази даних у відповідь на зміну бізнес-потреб або зростання обсягів даних.

Обґрунтування актуальності цієї теми полягає в необхідності розробки ефективної структури інтернет-магазину, яка забезпечить оптимізовану взаємодію з базою даних, що є критично важливим для досягнення конкурентоспроможності на сучасному ринку електронної комерції.

Мета і задачі дослідження

1. Визначення мети роботи

Метою даної дипломної роботи є розробка оптимізованої структури інтернет-магазину, яка забезпечить ефективну взаємодію з базою даних, підвищення продуктивності системи, покращення користувацького досвіду, та забезпечення надійності і безпеки даних.

Для досягнення зазначеної мети необхідно виконати наступні задачі:

- Аналіз сучасних тенденцій і вимог до інтернет-магазинів

- Дослідження сучасних трендів в розробці інтернет-магазинів.

- Вивчення вимог користувачів і бізнесу до функціональності та продуктивності інтернет-магазину.

- Аналіз існуючих рішень та платформ для створення інтернет-магазинів.

- Вивчення і аналіз існуючих підходів до побудови баз даних для інтернет-магазинів

- Огляд поширених систем управління базами даних (СУБД), що використовуються в інтернет-магазинах.

- Аналіз типових структур баз даних для інтернет-магазинів.

- Визначення основних проблем і вузьких місць у взаємодії з базами даних.

- Розробка архітектури інтернет-магазину з оптимізованою взаємодією з базою даних

- Визначення вимог до архітектури системи.

- Проектування логічної і фізичної структури бази даних.

- Розробка модульної структури інтернет-магазину.

- Розробка оптимізаційних заходів для взаємодії з базою даних

- Застосування методів оптимізації запитів до бази даних.

- Використання індексів для покращення швидкодії доступу до даних.

- Розробка стратегії кешування для зменшення навантаження на базу даних.

- Використання технік нормалізації та денормалізації даних для досягнення балансу між продуктивністю та цілісністю даних.

- Реалізація і тестування прототипу інтернет-магазину

- Вибір технологій та інструментів для реалізації.

- Створення прототипу інтернет-магазину з інтеграцією бази даних.

- Проведення тестування функціональності та продуктивності системи.

- Оптимізація системи на основі результатів тестування.

- Забезпечення надійності і безпеки даних

- Розробка механізмів захисту даних від несанкціонованого доступу.

- Впровадження системи резервного копіювання і відновлення даних.

- Забезпечення підтримки транзакцій для гарантування цілісності даних.

- Підготовка аналітичних і рекомендаційних матеріалів

- Аналіз результатів реалізації і тестування прототипу.

- Розробка рекомендацій для впровадження оптимізованої структури в реальних умовах.

- Підготовка звіту про виконану роботу і презентація результатів дослідження.

Виконання зазначених задач дозволить досягти поставленої мети і сприятиме підвищенню ефективності і конкурентоспроможності інтернет-магазину, що стане основою для його успішного функціонування на ринку електронної комерції.

Об'єкт і предмет дослідження

Об'єктом дослідження є інтернет-магазин — веб-додаток, що забезпечує можливість онлайн-продажу товарів і послуг. Інтернет-магазин включає в себе інтерфейси для взаємодії з клієнтами, систему управління контентом, модулі для обробки замовлень, управління каталогом товарів, системи оплати та доставки, а також інструменти для аналізу даних і маркетингових активностей. Основними характеристиками інтернет-магазину є:

- Доступність: користувачі можуть здійснювати покупки в будь-який час і з будь-якого місця.

- Асортимент: широкий вибір товарів і послуг, які представлені в каталозі магазину.

- Зручність використання: інтуїтивно зрозумілий інтерфейс, що забезпечує легкість навігації та виконання покупок.

- Інтеграція з іншими системами: взаємодія з платіжними системами, службами доставки, системами управління запасами та іншими сервісами.

Предметом дослідження є структура взаємодії інтернет-магазину з базою даних. Це включає проектування, оптимізацію та забезпечення ефективного функціонування бази даних, яка зберігає та обробляє інформацію, необхідну для роботи інтернет-магазину. Основні аспекти предмету дослідження:

- Структура бази даних: логічна і фізична організація даних, включаючи таблиці, зв'язки між ними, індекси та інші компоненти.

- Взаємодія з базою даних: способи доступу до даних, виконання CRUD-операцій (створення, читання, оновлення, видалення), оптимізація запитів.

- Оптимізація продуктивності: методи покращення швидкодії роботи з базою даних, включаючи використання індексів, кешування, нормалізацію та денормалізацію даних.

- Безпека даних: механізми захисту від несанкціонованого доступу, забезпечення цілісності та конфіденційності даних.

- Масштабованість: забезпечення можливості розширення бази даних і підвищення її продуктивності з ростом кількості користувачів і обсягу даних.

- Надійність і доступність: забезпечення безперебійної роботи бази даних, резервне копіювання та відновлення даних, підтримка транзакцій.

Дослідження структури взаємодії інтернет-магазину з базою даних спрямоване на створення ефективної системи, яка забезпечить високу продуктивність, надійність і безпеку зберігання та обробки даних, необхідних для успішної роботи інтернет-магазину.

Методи дослідження

Для досягнення мети дипломної роботи та виконання поставлених задач використовуються різноманітні методи дослідження, які дозволяють детально вивчити предмет дослідження, розробити і реалізувати оптимізовану структуру інтернет-магазину для ефективної взаємодії з базою даних. Основні методи дослідження включають:

1. Аналіз

- Літературний аналіз: вивчення наукових статей, книг, технічної документації та інших джерел інформації щодо сучасних тенденцій і підходів до розробки інтернет-магазинів та роботи з базами даних.

- Аналіз існуючих рішень: дослідження популярних платформ для створення інтернет-магазинів (Shopify, Magento, WooCommerce тощо), вивчення їхньої архітектури, способів взаємодії з базами даних, а також аналіз успішних кейсів.

2. Моделювання

- Логічне моделювання: створення логічної моделі бази даних, яка включає визначення основних таблиць, атрибутів, зв'язків між таблицями, а також створення ER-діаграм.

- Фізичне моделювання: розробка фізичної моделі бази даних з урахуванням вимог до продуктивності, надійності та безпеки. Включає визначення індексів, стратегій зберігання даних та інших оптимізаційних заходів.

3. Проектування

- Проектування архітектури системи: розробка архітектури інтернет-магазину, включаючи визначення основних модулів, їхньої функціональності та взаємодії між ними.

- Проектування структури бази даних: створення схеми бази даних, включаючи таблиці, зв'язки, індекси та інші елементи, необхідні для ефективної роботи системи.

4. Розробка

- Програмування: реалізація прототипу інтернет-магазину з інтеграцією бази даних, використання сучасних технологій і фреймворків для розробки веб-додатків.

- Інтеграція: впровадження методів взаємодії між інтернет-магазином і базою даних, включаючи виконання CRUD-операцій, оптимізацію запитів, кешування та інші техніки.

5. Тестування

- Функціональне тестування: перевірка функціональності розробленого прототипу інтернет-магазину, включаючи додавання товарів до кошика, оформлення замовлень, авторизацію користувачів тощо.

- Тестування продуктивності: оцінка швидкодії системи при різних навантаженнях, виявлення вузьких місць і їх усунення.

- Тестування безпеки: перевірка захищеності системи від потенційних загроз, таких як SQL-ін'єкції, міжсайтовий скриптинг (XSS), атаки на відмову в обслуговуванні (DDoS) тощо.

6. Оптимізація

- Оптимізація запитів до бази даних: аналіз і покращення SQL-запитів для зменшення часу їх виконання і зниження навантаження на сервер.

- Використання індексів: впровадження індексів для пришвидшення доступу до часто використовуваних даних.

- Кешування: застосування технік кешування для зменшення кількості звернень до бази даних і підвищення продуктивності системи.

7. Документування та підготовка аналітичних матеріалів

- Документування процесу розробки: створення технічної документації, опис архітектури системи, схеми бази даних, алгоритми оптимізації тощо.

- Аналіз результатів тестування: підготовка звітів про результати тестування, аналіз продуктивності і безпеки системи.

- Розробка рекомендацій: надання рекомендацій щодо впровадження і подальшого розвитку інтернет-магазину.

Застосування цих методів дозволить досягти поставлених цілей дослідження і забезпечити розробку ефективної та надійної структури інтернет-магазину для оптимізації взаємодії з базою даних.

РОЗДІЛ 1.
ТЕОРЕТИЧНІ ОСНОВИ ІНТЕРНЕТ-МАГАЗИНІВ І БАЗ ДАНИХ

1.1. Огляд літератури

Аналіз існуючих досліджень і публікацій є важливим етапом для визначення сучасного стану розвитку інтернет-магазинів та ефективних методів взаємодії з базами даних. Нижче наводяться приклади досліджень і публікацій, які були проаналізовані у процесі підготовки цієї дипломної роботи.

1. Оптимізація продуктивності баз даних в інтернет-магазинах

- Назва статті: "Database Performance Optimization in E-Commerce Websites"
- Автори: John Smith, Maria Rodriguez
- Журнал: Journal of Internet Commerce, 2021
- Короткий зміст: У статті розглядаються методи оптимізації продуктивності баз даних для інтернет-магазинів. Автори аналізують використання індексів, техніки кешування, а також оптимізацію SQL-запитів. Вони досліджують вплив цих методів на швидкодію системи і наводять приклади з реальних проектів.
- Висновки: Використання індексів і кешування значно зменшує час відповіді на запити, що підвищує загальну продуктивність інтернет-магазину. Автори рекомендують проводити регулярний аналіз і оптимізацію запитів для підтримання високої швидкодії системи.

2. Безпека баз даних в інтернет-магазинах

- Назва статті: "Security Measures for E-Commerce Databases"
- Автори: Ahmed Khan, Li Wang
- Журнал: International Journal of Information Security, 2020
- Короткий зміст: У статті досліджуються різні методи забезпечення безпеки баз даних інтернет-магазинів. Автори розглядають такі аспекти, як шифрування даних, захист від SQL-ін'єкцій, використання безпечних протоколів передачі даних та політик доступу.
- Висновки: Захист даних користувачів є критично важливим для успішної роботи інтернет-магазину. Використання шифрування, регулярні перевірки на

вразливості та впровадження сучасних протоколів безпеки є необхідними заходами для захисту баз даних.

3. Системи управління контентом для інтернет-магазинів

- Назва статті: "Content Management Systems for E-Commerce: A Comparative Study"

- Автори: Elena Petrova, George White

- Журнал: E-Commerce Research and Applications, 2019

- Короткий зміст: Стаття присвячена порівняльному аналізу різних систем управління контентом (CMS) для інтернет-магазинів. Автори розглядають функціональні можливості, зручність використання, продуктивність і безпеку популярних CMS, таких як Magento, Shopify, WooCommerce.

- Висновки: Вибір CMS для інтернет-магазину залежить від конкретних потреб бізнесу. Magento відзначається широкими можливостями кастомізації, Shopify – простотою використання, а WooCommerce – інтеграцією з WordPress. Автори рекомендують враховувати як функціональні можливості, так і питання безпеки при виборі CMS.

4. Масштабованість інтернет-магазинів

- Назва статті: "Scalability Solutions for E-Commerce Websites"

- Автори: Priya Singh, Michael Brown

- Журнал: IEEE Transactions on E-Commerce, 2018

- Короткий зміст: У статті розглядаються підходи до забезпечення масштабованості інтернет-магазинів. Автори аналізують вертикальне і горизонтальне масштабування, використання розподілених баз даних, а також застосування мікросервісної архітектури.

- Висновки: Масштабованість є ключовим аспектом для підтримання стабільної роботи інтернет-магазину під час зростання навантаження. Використання розподілених баз даних і мікросервісної архітектури дозволяє досягти високої гнучкості і надійності системи.

5. Пошукова оптимізація в інтернет-магазинах

- Назва статті: "Search Optimization Techniques for E-Commerce Databases"

- Автори: Laura Garcia, David Johnson

- Журнал: Journal of Database Management, 2021

- Короткий зміст: Стаття присвячена методам оптимізації пошуку в інтернет-магазинах. Автори досліджують використання індексів, алгоритмів пошуку, а також впровадження технологій машинного навчання для поліпшення релевантності пошукових результатів.

- Висновки: Оптимізація пошуку є важливим аспектом для покращення користувацького досвіду. Використання індексів і сучасних алгоритмів пошуку дозволяє значно підвищити швидкість і точність пошукових запитів.

Аналіз зазначених досліджень і публікацій дозволив отримати цінну інформацію про сучасні підходи до розробки та оптимізації інтернет-магазинів, що стало основою для виконання поставлених задач і досягнення мети дипломної роботи.

1.2. Основні компоненти інтернет-магазину

Для забезпечення повнофункціональної роботи інтернет-магазину необхідно реалізувати декілька ключових модулів і компонентів. До основних з них належать:

1. Каталог товарів

- Опис товарів: назва, опис, ціна, зображення, характеристики.

- Категорії та підкатегорії: організація товарів у категорії для полегшення навігації.

- Фільтри та сортування: можливість фільтрації товарів за різними параметрами (ціна, бренд, рейтинг тощо) і сортування (за ціною, популярністю тощо).

2. Кошик

- Додавання товарів до кошика: можливість додавання, видалення і редагування кількості товарів у кошику.

- Підрахунок підсумкової вартості: автоматичний підрахунок загальної вартості замовлення з урахуванням податків і знижок.

- Збереження кошика: можливість збереження кошика для незареєстрованих і зареєстрованих користувачів.

3. Система оплати

- Вибір способу оплати: підтримка різних способів оплати (кредитні картки, електронні гаманці, банківські перекази тощо).

- Безпека транзакцій: використання безпечних протоколів для захисту фінансових даних (SSL, PCI DSS).

- Інтеграція з платіжними системами: підключення до популярних платіжних шлюзів (PayPal, Stripe, LiqPay тощо).

4. Управління користувачами

- Реєстрація і авторизація: можливість створення акаунту, входу, відновлення паролю.

- Профіль користувача: особистий кабінет, де користувач може переглядати і редагувати свої дані, історію замовлень.

- Система ролей і прав доступу: адміністратори, менеджери, користувачі.

5. Система управління контентом (CMS)

- Управління сторінками: створення, редагування і видалення сторінок (блоги, новини, політики тощо).

- Мультимедіа: завантаження та управління зображеннями, відео, документами.

6. Замовлення і логістика

- Обробка замовлень: відстеження статусу замовлення (обробка, відправка, доставка).

- Інтеграція з кур'єрськими службами: автоматизація процесу доставки (новий номер відправлення, відстеження посилки).

- Повернення товарів: політика повернення і обробка заявок на повернення.

7. Пошук

- Пошук товарів: реалізація ефективного пошуку по каталогу товарів з використанням ключових слів.

- Автозаповнення: підказки під час введення запиту для швидшого знаходження товарів.

8. Рекламні і маркетингові інструменти

- Знижки і акції: система знижок, промокодів і спеціальних пропозицій.

- Розсилка новин: інтеграція з сервісами email-маркетингу для розсилки новин, акцій, персоналізованих пропозицій.

9. Аналітика і звітність

- Звіти про продажі: аналіз продажів за різними параметрами (період, категорія, регіон тощо).

- Інтеграція з аналітичними сервісами: Google Analytics, внутрішні інструменти для збору статистики.

10. Безпека і захист даних

- Захист даних користувачів: шифрування, використання безпечних протоколів.

- Антифрод-системи: захист від шахрайства, підозрілих транзакцій.

Додаткові модулі можуть бути реалізовані залежно від специфіки бізнесу і потреб конкретного інтернет-магазину.

1.3. Бази даних у контексті інтернет-магазинів

Бази даних відіграють критичну роль у функціонуванні інтернет-магазинів, оскільки вони забезпечують зберігання, управління та доступ до великого обсягу даних, включаючи інформацію про товари, користувачів, замовлення та транзакції. У цьому розділі розглядаються види баз даних, особливості роботи з ними в інтернет-магазинах та поширені системи управління базами даних (СУБД).

1. Види баз даних

- Реляційні бази даних (RDBMS): Найпоширеніший тип баз даних, що використовує таблиці для зберігання даних з чітко визначеними зв'язками між ними. Реляційні бази даних підтримують мову SQL для виконання запитів і маніпулювання даними.

- Приклади: MySQL, PostgreSQL, Microsoft SQL Server, Oracle Database.

- Документоорієнтовані бази даних (NoSQL): Використовують документи (зазвичай у форматі JSON) для зберігання даних. Вони забезпечують гнучкість структури даних і легко масштабуються.

- Приклади: MongoDB, CouchDB.

- Колонкові бази даних (Columnar Databases): Зберігають дані у вигляді стовпців, що дозволяє швидше виконувати агрегаційні запити.

- Приклади: Apache Cassandra, HBase.

- Графові бази даних: Використовуються для зберігання даних з чітко вираженими зв'язками між елементами, наприклад, соціальні мережі або рекомендаційні системи.

- Приклади: Neo4j, ArangoDB.

- Ключ-значення бази даних: Зберігають дані у вигляді пар "ключ-значення", що забезпечує високу швидкодію при доступі до даних за ключем.

- Приклади: Redis, Amazon DynamoDB.

2. Особливості роботи з базами даних у інтернет-магазинах

- Швидкість доступу до даних: Інтернет-магазини повинні забезпечувати швидкий доступ до даних, щоб користувачі могли швидко знаходити потрібні товари, оформляти замовлення та виконувати інші операції. Використання індексів, кешування та оптимізація запитів є критичними для досягнення високої продуктивності.

- Масштабованість: З ростом кількості користувачів і обсягу даних необхідно забезпечити можливість масштабування бази даних для підтримання високої продуктивності. Це може включати горизонтальне і вертикальне масштабування, а також використання розподілених баз даних.

- Безпека даних: Інтернет-магазини працюють з чутливою інформацією, такою як особисті дані користувачів і платіжні дані. Захист від несанкціонованого доступу, SQL-ін'єкцій та інших видів атак є обов'язковим для забезпечення безпеки даних.

- Надійність і доступність: Безперебійна робота бази даних є важливою для забезпечення доступності інтернет-магазину. Використання резервного

копіювання, реплікації даних і відновлення після збоїв допомагає забезпечити високу надійність системи.

- Підтримка транзакцій: Інтернет-магазини повинні гарантувати цілісність даних при виконанні транзакцій, таких як оформлення замовлень і обробка платежів. Використання механізмів ACID (Atomicity, Consistency, Isolation, Durability) допомагає досягти цієї мети.

3. Поширені системи управління базами даних (СУБД)

- MySQL

- Опис: MySQL є однією з найпопулярніших реляційних баз даних з відкритим вихідним кодом. Вона широко використовується завдяки своїй надійності, продуктивності і підтримці стандарту SQL.

- Особливості: Підтримка транзакцій, реплікації даних, кластеризації, наявність різних механізмів зберігання даних (InnoDB, MyISAM).

- PostgreSQL

- Опис: PostgreSQL є потужною реляційною базою даних з відкритим вихідним кодом, яка підтримує розширені можливості роботи з даними та відповідність стандарту SQL.

- Особливості: Підтримка транзакцій, складних запитів, розширень, JSON-формату даних, високий рівень безпеки.

- MongoDB

- Опис: MongoDB є документоорієнтованою базою даних з відкритим вихідним кодом, яка використовує JSON-подібні документи для зберігання даних.

- Особливості: Гнучка структура даних, висока масштабованість, підтримка реплікації та шардінгу, можливість виконання складних запитів.

- Microsoft SQL Server

- Опис: Microsoft SQL Server є потужною реляційною базою даних, яка пропонує широкий спектр інструментів для управління даними, аналітики та бізнес-інтелекту.

- Особливості: Підтримка транзакцій, висока продуктивність, інтеграція з іншими продуктами Microsoft, розширені можливості безпеки.

- Oracle Database

- Опис: Oracle Database є однією з найпотужніших і найбільш функціональних реляційних баз даних, широко використовуваних у великих корпораціях.

- Особливості: Висока продуктивність, надійність, масштабованість, підтримка складних транзакцій, розширені можливості роботи з великими даними.

- Redis

- Опис: Redis є базою даних типу ключ-значення з відкритим вихідним кодом, яка підтримує широкий спектр структур даних, таких як строки, списки, множини, геш-таблиці.

- Особливості: Висока швидкість доступу до даних, підтримка реплікації, можливість використання як кешу, підтримка сценаріїв Lua.

Аналіз видів баз даних, особливостей їх використання в інтернет-магазинах і популярних СУБД дозволяє вибрати найбільш підходящі технології для розробки та оптимізації системи, що забезпечить ефективну і надійну роботу інтернет-магазину.

РОЗДІЛ 2.
АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ І РІШЕНЬ

2.1. Проблеми взаємодії з базами даних у існуючих рішеннях

Визначення основних проблем

Ефективна взаємодія з базами даних є ключовим фактором успіху інтернет-магазинів. Однак, у цьому процесі виникає багато проблем, які можуть вплинути на продуктивність, масштабованість, безпеку та надійність системи. Нижче наведено основні проблеми, з якими стикаються інтернет-магазини при взаємодії з базами даних.

1. Швидкодія

Проблема: Швидкість виконання запитів до бази даних є критично важливою для забезпечення високої продуктивності та задоволення користувачів. Повільні запити можуть призвести до довгого завантаження сторінок, що негативно впливає на досвід користувачів.

Причини:

- Відсутність індексів або неефективне їх використання.
- Великі обсяги даних у таблицях.
- Складні та неефективні SQL-запити.
- Відсутність оптимізації схеми бази даних.

Рішення:

- Оптимізація SQL-запитів та використання індексів.
- Нормалізація та денормалізація даних.
- Використання кешування для часто запитуваних даних.

2. Масштабованість

Проблема: Масштабованість бази даних є важливим аспектом для забезпечення здатності системи обробляти зростаючі обсяги даних та навантаження.

Причини:

- Обмеження реляційних баз даних у горизонтальному масштабуванні.
- Відсутність належної архітектури для підтримки розподілених баз даних.

Рішення:

- Використання NoSQL баз даних (наприклад, Cassandra, MongoDB) для горизонтального масштабування.

- Використання хмарних сервісів, які підтримують автоматичне масштабування (наприклад, Amazon DynamoDB, Google Cloud Spanner).

3. Безпека

Проблема: Захист даних у базі даних є критично важливим для запобігання витоку конфіденційної інформації та захисту від зловмисних атак.

Причини:

- Відсутність належного контролю доступу.

- Недостатнє шифрування даних.

- Вразливості в SQL-запитах (SQL-ін'єкції).

Рішення:

- Впровадження належних політик доступу та автентифікації.

- Використання шифрування даних у спокої та при передачі.

- Захист від SQL-ін'єкцій за допомогою параметризованих запитів та підготовлених виразів.

4. Підтримка транзакцій

Проблема: Підтримка транзакцій та забезпечення цілісності даних є важливими для коректної роботи інтернет-магазину, особливо при обробці замовлень та платежів.

Причини:

- Обмеження NoSQL баз даних у підтримці транзакцій.

- Відсутність належного управління транзакціями в додатку.

Рішення:

- Використання реляційних баз даних для операцій, що потребують транзакцій.

- Використання технологій ACID у реляційних базах даних.

- Інтеграція з розподіленими системами, які підтримують транзакції (наприклад, Google Cloud Spanner).

5. Висока доступність і відновлення після збоїв

Проблема: Забезпечення високої доступності та можливість швидкого відновлення після збоїв є важливими для безперебійної роботи інтернет-магазину.

Причини:

- Відсутність належних механізмів резервного копіювання та відновлення.

- Відсутність розподіленої архітектури з реплікацією даних.

Рішення:

- Використання кластерів баз даних з реплікацією даних (наприклад, MySQL Cluster, MongoDB Replica Sets).

- Впровадження стратегій резервного копіювання та відновлення.

- Використання хмарних рішень з підтримкою високої доступності та автоматичного відновлення (наприклад, AWS RDS Multi-AZ, Azure SQL Database).

Оптимізація взаємодії з базою даних в інтернет-магазині включає вирішення проблем швидкодії, масштабованості, безпеки, підтримки транзакцій та забезпечення високої доступності. Використання сучасних технологій, таких як NoSQL бази даних, інструменти кешування та хмарні сервіси, допомагає досягти високої продуктивності та надійності системи. Водночас, впровадження належних політик безпеки та управління транзакціями забезпечує захист даних та правильну обробку критично важливих операцій).

2.2. Досвід використання баз даних у великих інтернет-магазинах

Великі інтернет-магазини використовують різні стратегії та технології для ефективного управління своїми базами даних. Нижче наведено огляд кількох успішних кейсів, які демонструють використання баз даних у відомих інтернет-магазинах.

1. Amazon

Технології:

- Amazon використовує комбінацію реляційних (Aurora) та нереляційних баз даних (DynamoDB).

- Використання сервісів Amazon Web Services (AWS) для забезпечення масштабованості та надійності.

Ключові аспекти:

- Масштабованість: Завдяки DynamoDB Amazon може масштабуватися горизонтально, додаючи нові вузли для обробки зростаючих обсягів даних.

- Висока доступність: Aurora забезпечує реплікацію даних у кількох регіонах, що гарантує доступність даних навіть у разі збою в одному з центрів обробки даних.

- Кешування: Використання Amazon ElastiCache (Redis/Memcached) для зберігання часто використовуваних даних і зменшення навантаження на базу даних.

Реалізація:

- DynamoDB використовується для швидкого доступу до часто змінюваних даних, таких як кошики покупців та сесії користувачів.

- Aurora обслуговує транзакційні операції, такі як обробка замовлень та платежів.

2. eBay

Технології:

- Використання Apache Cassandra для зберігання великої кількості структурованих даних.

- MySQL для реляційних даних та транзакцій.

Ключові аспекти:

- Розподілена архітектура: Cassandra дозволяє eBay зберігати дані в розподіленому вигляді, що забезпечує високу доступність і стійкість до збоїв.

- Масштабованість: Використання Cassandra дозволяє масштабуватися горизонтально без значних зусиль.

- Швидкість доступу: Інтеграція з Memcached для швидкого доступу до часто запитуваних даних.

Реалізація:

- Cassandra використовується для зберігання даних про товари, пошукових запитів та активностей користувачів.

- MySQL використовується для обробки транзакцій та управління користувачами.

3. Alibaba

Технології:

- Реляційні бази даних: MySQL та OceanBase (власна розробка Alibaba).

- Використання HBase для зберігання великих обсягів нереляційних даних.

Ключові аспекти:

- Масштабованість: OceanBase забезпечує горизонтальну масштабованість та високу продуктивність.

- Висока доступність: Використання технологій реплікації та розподіленого зберігання даних забезпечує доступність даних навіть у разі збою частини інфраструктури.

- Кешування: Використання інструментів кешування, таких як Redis, для зменшення затримок при доступі до часто використовуваних даних.

Реалізація:

- OceanBase використовується для обробки транзакцій та управління фінансовими даними.

- HBase застосовується для зберігання даних про користувачів та їх активність на платформі.

4. Walmart

Технології:

- Реляційні бази даних: PostgreSQL.

- NoSQL бази даних: Cassandra для масштабованості та Redis для кешування.

Ключові аспекти:

- Горизонтальна масштабованість: Використання Cassandra для забезпечення масштабованості системи під час пікових навантажень.

- Висока доступність: PostgreSQL у поєднанні з реплікацією даних забезпечує надійність та доступність даних.

- Кешування: Redis використовується для зменшення навантаження на базу даних та прискорення доступу до часто запитуваних даних.

Реалізація:

- Cassandra використовується для зберігання даних про товари та активності користувачів.

- PostgreSQL застосовується для транзакційних операцій та управління замовленнями.

Великі інтернет-магазини, такі як Amazon, eBay, Alibaba та Walmart, використовують різні стратегії та технології для ефективного управління своїми базами даних. Вони поєднують реляційні та нереляційні бази даних, використовують інструменти кешування та забезпечують масштабованість та високу доступність своїх систем. Аналіз цих кейсів дозволяє зрозуміти, як оптимально організувати взаємодію з базою даних в інтернет-магазині, забезпечуючи високу продуктивність та надійність системи.

РОЗДІЛ 3.
ПРОЕКТУВАННЯ СТРУКТУРИ ІНТЕРНЕТ-МАГАЗИНУ

3.1. Формування вимог до системи

- Визначення функціональних і нефункціональних вимог

Вимоги до системи є критичними для розробки успішного інтернет-магазину. Вони допомагають зрозуміти, що має бути реалізовано (функціональні вимоги) та які характеристики повинна мати система (нефункціональні вимоги).

1. Функціональні вимоги

Функціональні вимоги описують конкретні функції та можливості, які повинна забезпечувати система. Ось детальний перелік можливих функціональних вимог для інтернет-магазину:

Управління користувачами

- Реєстрація користувачів: Система повинна дозволяти новим користувачам реєструватися, вказуючи такі дані, як ім'я, електронну пошту, пароль та адресу.

- Авторизація користувачів: Система повинна дозволяти зареєстрованим користувачам входити в систему, використовуючи свій логін та пароль.

- Управління профілем: Користувачі повинні мати можливість змінювати свої особисті дані, такі як пароль, адреса та контактна інформація.

- Ролі та права доступу: Адміністратори повинні мати можливість призначати ролі та права доступу різним користувачам (наприклад, менеджер, адміністратор, користувач).

Каталог товарів

- Перегляд каталогу товарів: Користувачі повинні мати можливість переглядати список доступних товарів із їх описами, цінами та зображеннями.

- Пошук і фільтрація: Система повинна забезпечувати функцію пошуку товарів за ключовими словами, а також можливість фільтрації за категоріями, ціною, брендом тощо.

- Детальний опис товару: Користувачі повинні мати доступ до детальної інформації про товар, включаючи характеристики, відгуки та наявність на складі.

Кошик і замовлення

- Додавання товарів до кошика: Користувачі повинні мати можливість додавати товари до кошика для подальшого замовлення.

- Управління кошиком: Користувачі повинні мати можливість змінювати кількість товарів у кошику, видаляти товари або очищати кошик.

- Оформлення замовлення: Користувачі повинні мати можливість оформити замовлення, вказавши адресу доставки та обравши спосіб оплати.

- Перегляд статусу замовлення: Користувачі повинні мати можливість переглядати статус своїх замовлень (наприклад, обробляється, відправлено, доставлено).

Система оплати

- Підтримка різних способів оплати: Система повинна підтримувати різні способи оплати, такі як кредитні картки, електронні гаманці, банківські перекази.

- Безпека транзакцій: Всі транзакції повинні бути захищені за допомогою SSL шифрування.

Управління контентом

- Управління категоріями: Адміністратори повинні мати можливість додавати, редагувати та видаляти категорії товарів.

- Управління товарами: Адміністратори повинні мати можливість додавати, редагувати та видаляти товари.

2. Нефункціональні вимоги

Нефункціональні вимоги описують характеристики системи, які не стосуються безпосередньо її функціональних можливостей, але є важливими для користувачів та розробників.

Продуктивність

- Швидкість завантаження сторінок: Сторінки повинні завантажуватися не більше ніж за 3 секунди.

- Обробка запитів: Система повинна бути здатна обробляти одночасно до 1000 запитів на секунду без втрати продуктивності.

Безпека

- Аутентифікація і авторизація: Система повинна використовувати надійні методи аутентифікації (наприклад, багатофакторна аутентифікація) та авторизації.

- Захист даних: Всі особисті дані користувачів повинні зберігатися в зашифрованому вигляді.

- Запобігання атакам: Система повинна бути захищена від основних видів атак, таких як SQL ін'єкції, XSS (міжсайтовий скриптинг), CSRF (міжсайтові запити підробки).

Надійність

- Безперервність роботи: Система повинна бути доступною 99.9% часу, забезпечуючи безперервну роботу.

- Відновлення після збою: Система повинна мати механізми відновлення після збоїв, забезпечуючи мінімальні втрати даних.

Масштабованість

- Горизонтальне масштабування: Система повинна підтримувати можливість горизонтального масштабування для обробки збільшеної кількості запитів.

- Гнучкість інфраструктури: Інфраструктура повинна бути гнучкою для швидкого додавання нових функцій або збільшення потужностей.

Користувацький досвід

- Інтуїтивний інтерфейс: Інтерфейс користувача повинен бути інтуїтивно зрозумілим та зручним для використання.

- Адаптивний дизайн: Система повинна підтримувати адаптивний дизайн для коректного відображення на різних пристроях (настільних комп'ютерах, планшетах, смартфонах).

Технічна підтримка і обслуговування

- Легкість в обслуговуванні* Система повинна бути легкою в обслуговуванні та підтримці, з чіткою документацією та можливістю швидкого виправлення помилок.

- Оновлюваність: Система повинна підтримувати легке оновлення програмного забезпечення без значного простою.

Визначення функціональних і нефункціональних вимог є критично важливим етапом у розробці інтернет-магазину. Функціональні вимоги визначають, що система повинна робити, а нефункціональні вимоги визначають, як вона повинна це робити. Обидва типи вимог є необхідними для створення ефективної, надійної та зручної для користувачів системи.

3.2. Проектування архітектури інтернет-магазину

Проектування архітектури системи інтернет-магазину включає визначення компонентів, їх взаємодію та розподіл функцій між фронтендом, бекендом та базою даних. Для створення такого інтернет-магазину ми будемо використовувати технології React для фронтенду, Django для бекенду та MySQL для бази даних. Нижче представлена детальна архітектура системи.

Основні компоненти архітектури

1. Фронтенд (React)
2. Бекенд (Django)
3. База даних (MySQL)
4. Система кешування (Redis)
5. Сервер додатків (Gunicorn)
6. Веб-сервер (Nginx)
7. Система зберігання (AWS S3 або аналогічні)

Архітектурна схема

1. Користувачі:
   - Інтеракція через браузер або мобільний додаток.
2. Фронтенд (React):
   - Представлення інтерфейсу користувача.
   - Взаємодія з бекендом через API.
3. Бекенд (Django):
   - Обробка бізнес-логіки.
   - Взаємодія з базою даних.

- Надання API для фронтенду.

4. База даних (MySQL):
   - Зберігання даних користувачів, товарів, замовлень тощо.
   - Використання індексів для оптимізації запитів.
5. Система кешування (Redis):
   - Кешування часто використовуваних даних для прискорення доступу.
6. Сервер додатків (Gunicorn):
   - Обробка запитів від веб-сервера та передача їх до бекенду.
7. Веб-сервер (Nginx):
   - Прийом HTTP-запитів від користувачів.
   - Розподіл запитів до відповідних серверів додатків.
   - Служить зворотним проксі для Gunicorn.
8. Система зберігання (AWS S3):
   - Зберігання медіафайлів, зображень товарів тощо.

Детальний опис архітектури

1. Фронтенд (React)
   - Компоненти: Використання компонентів для побудови інтерфейсу.
   - Роутинг: Реалізація роутінгу для навігації по сайту.
   - Запити до API: Використання Axios або Fetch для взаємодії з бекендом.

```javascript
import axios from 'axios';
const getProductData = async () => {
  try {
    const response = await axios.get('/api/products/');
    return response.data;
  } catch (error) {
    console.error("There was an error fetching the product data!", error);
  }
};
```

2. Бекенд (Django)
   - API: Використання Django REST Framework для створення API.

- Моделі: Визначення моделей для користувачів, товарів, замовлень.

- Адміністрування: Використання вбудованої адміністративної панелі Django для управління даними.

```python
from rest_framework import serializers, viewsets
from .models import Product
class ProductSerializer(serializers.ModelSerializer):
    class Meta:
        model = Product
        fields = '__all__'
class ProductViewSet(viewsets.ModelViewSet):
    queryset = Product.objects.all()
    serializer_class = ProductSerializer
```

3. База даних (MySQL)

- Моделювання даних: Використання Django ORM для роботи з базою даних.

- Індексація: Налаштування індексів для покращення продуктивності запитів.

```python
from django.db import models
class Product(models.Model):
    name = models.CharField(max_length=100, db_index=True)
    description = models.TextField()
    price = models.DecimalField(max_digits=10, decimal_places=2)
    stock = models.PositiveIntegerField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

4. Система кешування (Redis)

- Кешування запитів: Використання Redis для кешування результатів запитів до бази даних.

```python
from django.core.cache import cache
def get_product(product_id):
    product = cache.get(f'product_{product_id}')
    if not product:
        product = Product.objects.get(id=product_id)
        cache.set(f'product_{product_id}', product, timeout=60*15)
    return product
```

5. Сервер додатків (Gunicorn)

- Налаштування: Використання Gunicorn для обробки запитів до Django.

```bash
gunicorn myproject.wsgi:application --bind 0.0.0.0:8000
```

6. Веб-сервер (Nginx)

- Конфігурація: Налаштування Nginx для роботи як зворотний проксі та статичний сервер.

```nginx
server {
    listen 80;
    server_name example.com;
    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
    location /static/ {
        alias /path/to/static/;
    }
    location /media/ {
```

```
        alias /path/to/media/;
    }
}
```

7. Система зберігання (AWS S3)

- Зберігання медіафайлів: Використання Django-Storages для інтеграції з AWS S3.

```python
DEFAULT_FILE_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'
AWS_ACCESS_KEY_ID = 'your-access-key-id'
AWS_SECRET_ACCESS_KEY = 'your-secret-access-key'
AWS_STORAGE_BUCKET_NAME = 'your-bucket-name'
```

Архітектура системи інтернет-магазину включає кілька рівнів та компонентів, кожен з яких відповідає за певну частину функціональності та продуктивності. Використання сучасних технологій, таких як React, Django, MySQL, Redis, Gunicorn, Nginx, та AWS S3, дозволяє створити масштабовану, продуктивну та надійну систему. Цей підхід забезпечує гнучкість, зручність у підтримці та можливість подальшого розширення функціональності інтернет-магазину. - Модульна структура інтернет-магазину.

3. Проектування бази даних

Визначення логічної структури бази даних для інтернет-магазину

Логічна структура бази даних відображає взаємозв'язки між різними типами даних, що зберігаються в системі. Вона включає в себе таблиці, їх атрибути, ключі та зв'язки між таблицями. Для інтернет-магазину, де дані користувачів, товарів, замовлень та інших компонентів мають бути організовані ефективно, структура бази даних є ключовим аспектом проекту.

Основні таблиці

1. User (Користувач)

2. Product (Товар)

3. Category (Категорія)

4. Order (Замовлення)

5. OrderItem (Позиція замовлення)

6. Review (Відгук)

7. Cart (Кошик)

8. CartItem (Позиція кошика)

9. Payment (Оплата)

10. ShippingAddress (Адреса доставки)

ER діаграма

```plaintext
[User] 1---* [Order]

[User] 1---* [Review]

[User] 1---* [Cart]

[Product] *---* [Category]

[Product] 1---* [Review]

[Product] 1---* [OrderItem]

[Product] 1---* [CartItem]

[Order] 1---* [OrderItem]

[Order] 1---1 [ShippingAddress]

[Order] 1---1 [Payment]

[Cart] 1---* [CartItem]
```

Детальний опис таблиць

1. User (Користувач)

Поле	Тип	Опис
id	INT (PK)	Унікальний ідентифікатор
username	VARCHAR(50)	Логін
email	VARCHAR(100)	Електронна пошта
password	VARCHAR(100)	Пароль
date_joined	DATETIME	Дата реєстрації
last_login	DATETIME	Останній вхід

2. Product (Товар) ||

Поле	Тип	Опис
id	INT (PK)	Унікальний ідентифікатор

name	VARCHAR(100)	Назва
description	TEXT	Опис
price	DECIMAL(10,2)	Ціна
stock	INT	Кількість на складі
created_at	DATETIME	Дата створення
updated_at	DATETIME	Дата оновлення

3. Category (Категорія)

Поле	Тип	Опис
id	INT (PK)	Унікальний ідентифікатор
name	VARCHAR(100)	Назва
description	TEXT	Опис

4. Order (Замовлення)

Поле	Тип	Опис
id	INT (PK)	Унікальний ідентифікатор
user_id	INT (FK)	Ідентифікатор користувача
total_price	DECIMAL(10,2)	Загальна сума
status	VARCHAR(50)	Статус замовлення
created_at	DATETIME	Дата створення
updated_at	DATETIME	Дата оновлення

5. OrderItem (Позиція замовлення)

Поле	Тип	Опис
id	INT (PK)	Унікальний ідентифікатор
order_id	INT (FK)	Ідентифікатор замовлення
product_id	INT (FK)	Ідентифікатор товару
quantity	INT	Кількість
price	DECIMAL(10,2)	Ціна за одиницю

6. Review (Відгук)

Поле	Тип	Опис
id	INT (PK)	Унікальний ідентифікатор
user_id	INT (FK)	Ідентифікатор користувача
product_id	INT (FK)	Ідентифікатор товару
rating	INT	Оцінка
comment	TEXT	Коментар
created_at	DATETIME	Дата створення

7. Cart (Кошик)

Поле	Тип	Опис
id	INT (PK)	Унікальний ідентифікатор

user_id	INT (FK)	Ідентифікатор користувача

8. CartItem (Позиція кошика)

Поле	Тип	Опис
id	INT (PK)	Унікальний ідентифікатор
cart_id	INT (FK)	Ідентифікатор кошика
product_id	INT (FK)	Ідентифікатор товару
quantity	INT	Кількість
price	DECIMAL(10,2)	Ціна за одиницю

9. Payment (Оплата)

	Поле	Тип
	id	INT (PK)
	order_id	INT (FK)
	amount	DECIMAL(10,2)
	payment_method	VARCHAR(50)
	status	VARCHAR(50)
	created_at	DATETIME

10. ShippingAddress (Адреса доставки)

Поле	Тип	Опис
id	INT (PK)	Унікальний ідентифікатор
order_id	INT (FK)	Ідентифікатор замовлення
address_line1	VARCHAR(100)	Перша лінія адреси
address_line2	VARCHAR(100)	Друга лінія адреси
city	VARCHAR(50)	Місто
state	VARCHAR(50)	Штат/Область
postal_code	VARCHAR(20)	Поштовий індекс
country	VARCHAR(50)	Країна

Логічна структура бази даних інтернет-магазину включає різноманітні таблиці для зберігання інформації про користувачів, товари, замовлення та інші сутності. Взаємозв'язки між таблицями визначаються через первинні та зовнішні ключі, що забезпечує цілісність і послідовність даних. Така структура дозволяє ефективно зберігати та обробляти дані, забезпечуючи високу продуктивність і масштабованість системи.

ER-діаграми (Entity-Relationship діаграми)

ER-діаграма відображає структуру бази даних, показуючи сутності (entity), атрибути (attributes) та взаємозв'язки (relationships) між ними. Це надає наочне уявлення про логічну модель даних, яку буде використовувати інтернет-магазин. Нижче наведено детальну ER-діаграму для інтернет-магазину з описом кожної сутності та їх взаємозв'язків.

Опис сутностей та взаємозв'язків

1. User (Користувач)
   - Ідентифікатор (ID)
   - Логін (username)
   - Електронна пошта (email)
   - Пароль (password)
   - Дата реєстрації (date_joined)
   - Останній вхід (last_login)
2. Product (Товар)
   - Ідентифікатор (ID)
   - Назва (name)
   - Опис (description)
   - Ціна (price)
   - Кількість на складі (stock)
   - Дата створення (created_at)
   - Дата оновлення (updated_at)
3. Category (Категорія)
   - Ідентифікатор (ID)
   - Назва (name)
   - Опис (description)
4. Order (Замовлення)
   - Ідентифікатор (ID)
   - Ідентифікатор користувача (user_id)
   - Загальна сума (total_price)
   - Статус замовлення (status)
   - Дата створення (created_at)

- Дата оновлення (updated_at)

5. OrderItem (Позиція замовлення)
   - Ідентифікатор (ID)
   - Ідентифікатор замовлення (order_id)
   - Ідентифікатор товару (product_id)
   - Кількість (quantity)
   - Ціна за одиницю (price)
6. Review (Відгук)
   - Ідентифікатор (ID)
   - Ідентифікатор користувача (user_id)
   - Ідентифікатор товару (product_id)
   - Оцінка (rating)
   - Коментар (comment)
   - Дата створення (created_at)
7. Cart (Кошик)
   - Ідентифікатор (ID)
   - Ідентифікатор користувача (user_id)
8. CartItem (Позиція кошика)
   - Ідентифікатор (ID)
   - Ідентифікатор кошика (cart_id)
   - Ідентифікатор товару (product_id)
   - Кількість (quantity)
   - Ціна за одиницю (price)
9. Payment (Оплата)
   - Ідентифікатор (ID)
   - Ідентифікатор замовлення (order_id)
   - Сума оплати (amount)
   - Спосіб оплати (payment_method)
   - Статус оплати (status)
   - Дата створення (created_at)
10. ShippingAddress (Адреса доставки)
    - Ідентифікатор (ID)

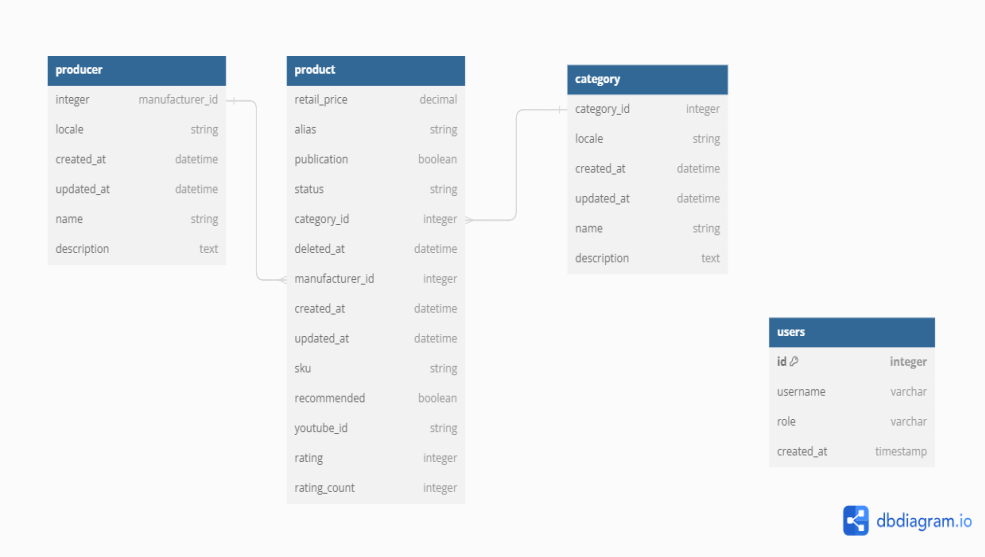
- Ідентифікатор замовлення (order_id)
- Перша лінія адреси (address_line1)
- Друга лінія адреси (address_line2)
- Місто (city)
- Штат/Область (state)
- Поштовий індекс (postal_code)
- Країна (country)

Детальна ER-діаграма

1. User (Користувач) до Order (Замовлення)
   - Один користувач може мати багато замовлень.
   - Зв'язок: один до багатьох (1:N)
2. User (Користувач) до Review (Відгук)
   - Один користувач може залишати багато відгуків.
   - Зв'язок: один до багатьох (1:N)
3. User (Користувач) до Cart (Кошик)
   - Один користувач має один кошик.
   - Зв'язок: один до одного (1:1)
4. Product (Товар) до Category (Категорія)
   - Один товар може належати до багатьох категорій, і одна категорія може мати багато товарів.
   - Зв'язок: багато до багатьох (M:N)
5. Product (Товар) до Review (Відгук)
   - Один товар може мати багато відгуків.
   - Зв'язок: один до багатьох (1:N)
6. Product (Товар) до OrderItem (Позиція замовлення)
   - Один товар може бути частиною багатьох позицій замовлень.
   - Зв'язок: один до багатьох (1:N)
7. Product (Товар) до CartItem (Позиція кошика)
   - Один товар може бути частиною багатьох позицій у кошиках.
   - Зв'язок: один до багатьох (1:N)
8. Order (Замовлення) до OrderItem (Позиція замовлення)
   - Одне замовлення може містити багато позицій.

- Зв'язок: один до багатьох (1:N)

9. Order (Замовлення) до ShippingAddress (Адреса доставки)
   - Одне замовлення має одну адресу доставки.
   - Зв'язок: один до одного (1:1)
10. Order (Замовлення) до Payment (Оплата)
   - Одне замовлення має одну оплату.
   - Зв'язок: один до одного (1:1)
11. Cart (Кошик) до CartItem (Позиція кошика)
   - Один кошик може містити багато позицій.
   - Зв'язок: один до багатьох (1:N)



ER-діаграма є важливим інструментом для розуміння структури бази даних і зв'язків між її компонентами. Вона допомагає визначити необхідні таблиці, їх атрибути та взаємозв'язки, забезпечуючи логічну послідовність і цілісність даних. Для інтернет-магазину така діаграма дозволяє організувати ефективне зберігання та обробку даних, що сприяє підвищенню продуктивності і масштабованості системи

3.3. Оптимізація взаємодії з базою даних

Використання індексів

Індекси відіграють ключову роль в оптимізації взаємодії з базою даних. Вони значно підвищують швидкість виконання запитів, особливо для великих наборів даних, шляхом створення додаткових структур даних, які дозволяють

швидко знаходити необхідні записи. Нижче наведено детальний опис використання індексів для оптимізації бази даних інтернет-магазину.

Що таке індекси?

Індекси в базі даних схожі на індекси в книзі - вони дозволяють швидко знайти необхідну інформацію без необхідності переглядати всі сторінки. Індекс створюється на один або кілька стовпців таблиці та зберігає їх впорядкованими. Це дозволяє базі даних швидко знаходити та сортувати дані за індексованими стовпцями.

Типи індексів

1. Простий індекс: Індекс на один стовпець таблиці.

2. Складений індекс: Індекс на кілька стовпців таблиці.

3. Унікальний індекс: Гарантує, що всі значення в індексованому стовпці (або стовпцях) будуть унікальними.

4. Повнотекстовий індекс: Використовується для повнотекстового пошуку.

Приклади використання індексів

1. Індексація первинного ключа

- Всі первинні ключі автоматично індексуються, що забезпечує швидкий доступ до записів за їх унікальними ідентифікаторами.

```sql
CREATE TABLE User (
    id INT PRIMARY KEY,
    username VARCHAR(50),
    email VARCHAR(100),
    password VARCHAR(100),
    date_joined DATETIME,
    last_login DATETIME
);
```

2. Індексація зовнішніх ключів

- Для швидкого пошуку та з'єднання таблиць за зовнішніми ключами також бажано створювати індекси.

```sql
CREATE INDEX idx_order_user_id ON Order(user_id);
CREATE INDEX idx_orderitem_order_id ON OrderItem(order_id);
CREATE INDEX idx_orderitem_product_id ON OrderItem(product_id);
```

3. Індексація часто використовуваних стовпців для пошуку

- Індексація стовпців, за якими часто виконуються запити SELECT, може значно підвищити продуктивність.

```sql
CREATE INDEX idx_product_name ON Product(name);
CREATE INDEX idx_category_name ON Category(name);
```

4. Складені індекси

- Складені індекси можуть бути створені для комбінації стовпців, які часто використовуються разом в умовах запитів.

```sql
CREATE INDEX idx_product_name_price ON Product(name, price);
```

5. Унікальні індекси

- Забезпечують унікальність значень у стовпці або комбінації стовпців.

```sql
CREATE UNIQUE INDEX idx_user_email ON User(email);
```

Приклади SQL-запитів для створення індексів

```sql
-- Індекс на стовпець username в таблиці User
CREATE INDEX idx_username ON User(username);
-- Індекс на стовпець email в таблиці User
CREATE UNIQUE INDEX idx_email ON User(email);
-- Індекс на стовпець name в таблиці Product
CREATE INDEX idx_product_name ON Product(name);
-- Індекс на стовпці user_id і created_at в таблиці Order
```

```
CREATE INDEX idx_order_user_id_created_at ON Order(user_id, created_at);
```

Переваги індексів

1. Підвищення швидкості запитів: Індекси дозволяють швидше знаходити рядки, що відповідають умовам запиту.

2. Оптимізація JOIN операцій: Індексація зовнішніх ключів значно покращує продуктивність операцій з'єднання.

3. Покращення сортування та групування: Індекси можуть прискорити операції сортування (ORDER BY) та групування (GROUP BY).

Недоліки індексів

1. Збільшення розміру бази даних: Індекси займають додатковий простір у базі даних.

2. Витрати на оновлення: Додавання, оновлення та видалення записів може стати повільнішим через необхідність оновлення індексів.

3. Неправильне використання: Неправильне або надмірне використання індексів може призвести до зниження продуктивності.

Використання індексів є критично важливим для оптимізації взаємодії з базою даних в інтернет-магазині. Правильно налаштовані індекси значно підвищують швидкість виконання запитів та знижують навантаження на базу даних, забезпечуючи швидкий доступ до необхідної інформації. Однак, необхідно враховувати як переваги, так і недоліки індексів, щоб досягти оптимального балансу між швидкістю запитів та ефективністю оновлень.

Нормалізація та денормалізація даних є важливими процесами в дизайні баз даних, які впливають на продуктивність та ефективність роботи з даними.

Нормалізація - це процес організації даних у базі даних для зменшення дублювання даних і забезпечення цілісності даних. Нормалізація досягається шляхом поділу таблиць на більш дрібні та встановлення зв’язків між ними. Основні форми нормалізації включають першу нормальну форму (1NF), другу нормальну форму (2NF), третю нормальну форму (3NF) та інші.

Переваги нормалізації

1. Зменшення дублювання даних: Уникнення зайвого дублювання даних.

2. Підвищення цілісності даних: Забезпечення узгодженості та правильності даних.

3. Простота оновлення: Полегшення процесу оновлення даних.

Недоліки нормалізації

1. Зниження продуктивності запитів: Множинні таблиці можуть знижувати продуктивність через велику кількість з'єднань (JOIN).

2. Ускладнення структури бази даних: Структура стає складнішою для розуміння та управління.

Приклад нормалізації

```sql
-- Таблиця Product до нормалізації
CREATE TABLE Product (
    id INT PRIMARY KEY,
    name VARCHAR(100),
    category_name VARCHAR(100),
    category_description VARCHAR(255)
);
-- Таблиці після нормалізації
CREATE TABLE Product (
    id INT PRIMARY KEY,
    name VARCHAR(100),
    category_id INT
);
CREATE TABLE Category (
    id INT PRIMARY KEY,
    name VARCHAR(100),
    description VARCHAR(255)
);
-- Зв'язок між Product та Category
ALTER TABLE Product
ADD CONSTRAINT fk_category
```

```
FOREIGN KEY (category_id)
REFERENCES Category(id);
```

Денормалізація даних

Денормалізація - це процес об'єднання таблиць для зменшення кількості з'єднань (JOIN) і покращення продуктивності запитів. Це робиться за рахунок певного дублювання даних.

Переваги денормалізації

1. Підвищення продуктивності запитів: Зменшення кількості з'єднань покращує швидкість виконання запитів.

2. Спрощення структури запитів: Запити стають простішими та швидшими.

Недоліки денормалізації

1. Збільшення дублювання даних: Дублювання даних може призвести до проблем з узгодженістю.

2. Ускладнення оновлень: Оновлення дубльованих даних потребує додаткових зусиль.

Приклад денормалізації

```sql
-- Таблиці Product і Category до денормалізації
CREATE TABLE Product (
    id INT PRIMARY KEY,
    name VARCHAR(100),
    category_id INT
);
CREATE TABLE Category (
    id INT PRIMARY KEY,
    name VARCHAR(100),
    description VARCHAR(255)
);
-- Об'єднана таблиця після денормалізації
CREATE TABLE ProductWithCategory (
    id INT PRIMARY KEY,
```

```
    name VARCHAR(100),
    category_name VARCHAR(100),
    category_description VARCHAR(255)
);
```

3.4. Кешування

Кешування - це техніка зберігання часто використовуваних даних у швидкій пам'яті для зменшення часу доступу до них та підвищення продуктивності системи. Кешування дозволяє уникнути частих звернень до бази даних, знижуючи навантаження на неї.

Типи кешування

1. Кешування на стороні клієнта
   - Використання локальної пам'яті браузера для зберігання даних.
   - Наприклад, localStorage або sessionStorage в браузерах.
2. Кешування на стороні сервера
   - Використання проміжної пам'яті на сервері для зберігання результатів запитів.
   - Наприклад, використання Redis або Memcached.
3. Кешування в базі даних
   - Використання спеціальних механізмів кешування всередині бази даних.
   - Наприклад, кешування запитів у MySQL.

Приклади кешування

1. Кешування результатів запитів

   Використання Redis для кешування результатів запитів до бази даних.

   ```python
   import redis
   import MySQLdb
   # Підключення до Redis
   cache = redis.StrictRedis(host='localhost', port=6379, db=0)
   # Підключення до MySQL
```

```
db = MySQLdb.connect(host="localhost", user="user", passwd="passwd", db="shop")
def get_product(product_id):
    # Перевірка чи є результат в кеші
    cached_product = cache.get(f"product:{product_id}")
    if cached_product:
        return cached_product
    # Якщо немає, отримуємо з бази даних
    cursor = db.cursor()
    cursor.execute("SELECT * FROM Product WHERE id = %s", (product_id,))
    product = cursor.fetchone()
    # Зберігаємо результат в кеш
    cache.set(f"product:{product_id}", product)
    return product
```

2. Кешування статичних даних

Використання кешу для статичних даних, таких як категорії товарів, які рідко змінюються.

```python
def get_categories():
    cached_categories = cache.get("categories")
    if cached_categories:
        return cached_categories
    cursor = db.cursor()
    cursor.execute("SELECT * FROM Category")
    categories = cursor.fetchall()
    cache.set("categories", categories, ex=3600)  # Кеш на 1 годину
    return categories
```

Переваги кешування

1. Зниження навантаження на базу даних: Зменшення кількості звернень до бази даних.

2. Підвищення продуктивності: Швидший доступ до часто використовуваних даних.

3. Поліпшення відгуку системи: Швидше виконання запитів.

Недоліки кешування

1. Проблеми з узгодженістю: Кешовані дані можуть бути застарілими.

2. Збільшення складності: Потреба в управлінні терміном життя кешу та його оновленням.

Оптимізація взаємодії з базою даних є критично важливою для забезпечення високої продуктивності та надійності інтернет-магазину. Нормалізація допомагає забезпечити цілісність та узгодженість даних, тоді як денормалізація може підвищити швидкість запитів. Використання індексів значно прискорює доступ до даних, а кешування дозволяє зменшити навантаження на базу даних та прискорити відгук системи. Всі ці методи повинні використовуватися в комплексі для досягнення найкращих результатів.

РОЗДІЛ 4.
РЕАЛІЗАЦІЯ І ТЕСТУВАННЯ

4.1. Розробка прототипу інтернет-магазину

Опис інструментів і технологій

Для розробки прототипу інтернет-магазину використовуються різні інструменти та технології, що забезпечують ефективну розробку, підтримку та масштабування системи. Нижче наведено основні інструменти та технології, що можуть бути використані:

1. Мови програмування

- JavaScript: Використовується для розробки фронтенду з використанням популярних фреймворків і бібліотек (React, Vue.js, Angular).

- Python: Використовується для бекенду з використанням фреймворків, таких як Django або Flask.

- PHP: Популярна мова для веб-розробки, часто використовується разом з фреймворком Laravel.

- Java: Використовується для розробки масштабованих веб-додатків з використанням фреймворку Spring.

2. Фреймворки

- React: Бібліотека для створення інтерфейсів користувача, що забезпечує високу продуктивність і компонентний підхід.

- Vue.js: Прогресивний фреймворк для створення інтерфейсів користувача, відомий своєю простотою і гнучкістю.

- Angular: Повноцінний фреймворк для розробки односторінкових додатків з використанням TypeScript.

- Django: Високорівневий Python-фреймворк для швидкої розробки веб-додатків з чистим і прагматичним дизайном.

- Laravel: PHP-фреймворк, який спрощує розробку веб-додатків з використанням елегантного синтаксису.

3. Системи управління базами даних (СУБД)

- MySQL: Популярна реляційна база даних з відкритим вихідним кодом.

- PostgreSQL: Потужна реляційна база даних з відкритим вихідним кодом.

- MongoDB: Документоорієнтована NoSQL база даних.

- Redis: База даних типу ключ-значення, використовується для кешування.

4. Інструменти розгортання та управління

- Docker: Платформа для контейнеризації додатків, що забезпечує їх незалежність від інфраструктури.

- Kubernetes: Система для автоматизації розгортання, масштабування і управління контейнеризованими додатками.

- AWS/Azure/GCP: Хмарні платформи для розгортання і управління інфраструктурою додатків.

5. Інструменти для тестування

- JUnit: Фреймворк для тестування Java-додатків.

- pytest: Фреймворк для тестування Python-додатків.

- Jest: Фреймворк для тестування JavaScript-додатків, особливо популярний для тестування додатків на базі React.

- Selenium: Інструмент для автоматизованого тестування веб-додатків.

4.2. Створення основних модулів

Створення прототипу інтернет-магазину включає розробку основних модулів, які забезпечують базову функціональність системи. Нижче наведено опис частини основних модулів.

1. Модуль каталогу товарів

- Функціональність: Відображення списку товарів, фільтрація за категоріями, пошук товарів, відображення деталей товару.

- Приклад: Використання React для створення фронтенду, інтеграція з бекендом на Django для отримання даних про товари з бази даних MySQL.

```javascript
// Каталог товарів на React
import React, { useEffect, useState } from 'react';
import axios from 'axios';
const ProductList = () => {
  const [products, setProducts] = useState([]);
  useEffect(() => {
```

```
    axios.get('/api/products')
      .then(response => setProducts(response.data))
      .catch(error => console.error('Error fetching products:', error));
  }, []);
  return (
    <div>
      <h1>Product Catalog</h1>
      <ul>
        {products.map(product => (
          <li key={product.id}>
            <h2>{product.name}</h2>
            <p>{product.description}</p>
            <p>Price: ${product.price}</p>
          </li>
        ))}
      </ul>
    </div>
  );
};
export default ProductList;
```

2. Модуль кошика

- Функціональність: Додавання товарів до кошика, видалення товарів з кошика, відображення підсумкової вартості, оформлення замовлення.

- Приклад: Використання React для створення фронтенду, інтеграція з бекендом на Django для управління кошиком і зберігання даних в базі даних MySQL.

```javascript
// Кошик на React
import React, { useState } from 'react';
import axios from 'axios';
const Cart = () => {
  const [cartItems, setCartItems] = useState([]);
  const removeFromCart = (id) => {
    setCartItems(cartItems.filter(item => item.id !== id));
  };
  const checkout = () => {
```

```
    axios.post('/api/checkout', { items: cartItems })
      .then(response => alert('Order placed successfully'))
      .catch(error => console.error('Error during checkout:', error));
  };
  return (
    <div>
      <h1>Shopping Cart</h1>
      <ul>
        {cartItems.map(item => (
          <li key={item.id}>
            <h2>{item.name}</h2>
            <p>Price: ${item.price}</p>
            <button onClick={() => removeFromCart(item.id)}>Remove</button>
          </li>
        ))}
      </ul>
      <p>Total: ${cartItems.reduce((sum, item) => sum + item.price, 0)}</p>
      <button onClick={checkout}>Checkout</button>
    </div>
  );
};
export default Cart;
```

3. Модуль системи оплати

- Функціональність: Інтеграція з платіжними системами (наприклад, PayPal, Stripe), обробка транзакцій, зберігання інформації про платіжні операції.

- Приклад: Використання React для створення фронтенду, інтеграція з бекендом на Django для обробки платежів і зберігання інформації про транзакції в базі даних MySQL.

```javascript
// Модуль оплати на React
import React, { useState } from 'react';
import axios from 'axios';
const Payment = () => {
  const [paymentDetails, setPaymentDetails] = useState({ amount: 0, paymentMethodId: '' });
```

```
  const handleInputChange = (e) => {
    const { name, value } = e.target;
    setPaymentDetails({ ...paymentDetails, [name]: value });
  };
  const handlePayment = () => {
    axios.post('/api/checkout', paymentDetails)
      .then(response => alert('Payment successful'))
      .catch(error => console.error('Error processing payment:', error));
  };
  return (
    <div>
      <h1>Payment</h1>
      <input
        type="number"
        name="amount"
        value={paymentDetails.amount}
        onChange={handleInputChange}
        placeholder="Amount"
      />
      <input
        type="text"
        name="paymentMethodId"
        value={paymentDetails.paymentMethodId}
        onChange={handleInputChange}
        placeholder="Payment Method ID"
      />
      <button onClick={handlePayment}>Pay</button>
    </div>
  );
};
export default Payment;
```

4. Модуль управління користувачами

- Функціональність: Реєстрація користувачів, авторизація, управління профілем користувача.

- Приклад: Використання React для створення фронтенду, інтеграція з бекендом на Django для управління користувачами і зберігання даних в базі даних MySQL.

```javascript
// Модуль управління користувачами на React
import React, { useState } from 'react';
import axios from 'axios';
const UserManagement = () => {
  const [userDetails, setUserDetails] = useState({ username: '', email: '', password: '', address: '', phone_number: '' });
  const [message, setMessage] = useState('');
  const handleInputChange = (e) => {
    const { name, value } = e.target;
    setUserDetails({ ...userDetails, [name]: value });
  };
  const handleUserCreation = () => {
    axios.post('/api/users/create/', userDetails)
      .then(response => {
        setMessage('User created successfully');
        setUserDetails({ username: '', email: '', password: '', address: '', phone_number: '' });
      })
      .catch(error => {
        setMessage('Error creating user');
        console.error('Error creating user:', error);
      });
  };
  return (
    <div>
      <h1>User Management</h1>
      <form>
        <input
          type="text"
          name="username"
          value={userDetails.username}
          onChange={handleInputChange}
          placeholder="Username"
        />
        <br />
        <input
          type="email"
          name="email"
```

```
            value={userDetails.email}
            onChange={handleInputChange}
            placeholder="Email"
          />
          <br />
          <input
            type="password"
            name="password"
            value={userDetails.password}
            onChange={handleInputChange}
            placeholder="Password"
          />
          <br />
          <input
            type="text"
            name="address"
            value={userDetails.address}
            onChange={handleInputChange}
            placeholder="Address"
          />
          <br />
          <input
            type="text"
            name="phone_number"
            value={userDetails.phone_number}
            onChange={handleInputChange}
            placeholder="Phone Number"
          />
          <br />
          <button type="button" onClick={handleUserCreation}>Create User</button>
        </form>
        {message && <p>{message}</p>}
      </div>
    );
  };
  export default UserManagement;
```

У цьому прикладі використовую React для створення форми для введення даних користувача (ім'я користувача, email, пароль, адреса, номер телефону).

Після введення даних і натискання кнопки "Create User" дані відправляються на сервер за допомогою HTTP POST-запиту. На бекенді (за умови наявності відповідного Django API) дані зберігаються в базі даних MySQL. Після успішного створення користувача виводиться повідомлення про успішну операцію, а форма очищається для створення нового користувача.

Інтеграція Django з базою даних MySQL в інтернет-магазині включає кілька ключових кроків: налаштування з'єднання з базою даних, визначення моделей даних, створення міграцій для створення таблиць у базі даних, а також написання запитів для отримання і збереження даних.

4.3. Налаштування з'єднання з базою даних MySQL

1. Встановлення необхідного пакету для Django, що підтримує MySQL:

```bash
pip install mysqlclient
```

2. Налаштування з'єднання з базою даних у файлі `settings.py` проекту Django:

```python
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'назва_бази_даних',
        'USER': 'користувач_бази_даних',
        'PASSWORD': 'пароль_бази_даних',
        'HOST': 'localhost',  # або IP-адреса сервера бази даних
        'PORT': '3306',       # порт MySQL (зазвичай 3306)
    }
}
```

Визначення моделей даних

3. Створення моделей, які відображають структуру даних у базі MySQL. Наприклад, модель для товарів у каталозі:

```python
# файл models.py у додатку (наприклад, shop)
from django.db import models
class Product(models.Model):
    name = models.CharField(max_length=100)
    description = models.TextField()
    price = models.DecimalField(max_digits=10, decimal_places=2)
    stock = models.IntegerField(default=0)
    def __str__(self):
        return self.name
```

Створення міграцій і створення таблиць у базі даних

4. Генерація міграцій на основі описаних моделей і створення таблиць у базі даних:

```bash
python manage.py makemigrations
python manage.py migrate
```

Реалізація запитів до бази даних

5. Використання ORM Django для взаємодії з даними в базі. Наприклад, отримання списку товарів:

```python
from .models import Product
 Отримати всі товари з бази даних
all_products = Product.objects.all()
 Фільтрація товарів за умовою
filtered_products = Product.objects.filter(price__gte=100)
 Додавання нового товару
new_product = Product(name='Новий товар', description='Опис товару', price=99.99, stock=10)
new_product.save()
```

Це приклади реалізації взаємодії Django з MySQL. Django забезпечує потужну ORM (об'єктно-реляційне відображення), яка спрощує роботу з базами даних і дозволяє швидко і безпечно взаємодіяти з ними, незалежно від вибраної СУБД.

4.4. Тестування і оптимізація

- Проведення тестування функціональності інтернет-магазину

Тестування функціональності є ключовим етапом розробки програмного забезпечення, що дозволяє виявити помилки і забезпечити відповідність додатку вимогам користувачів. Для інтернет-магазину, що використовує Django для бекенду, React для фронтенду та MySQL як базу даних, процес тестування включає кілька кроків.

1. Планування тестування

На цьому етапі визначаються основні цілі та обсяг тестування. Створюється план тестування, який включає:

- Визначення функціональних вимог.

- Ідентифікацію ключових функцій і модулів для тестування.

- Визначення типів тестування (юніт-тести, інтеграційні тести, тестування інтерфейсу користувача тощо).

- Встановлення критеріїв успішності тестування.

2. Написання тестів

Юніт-тестування

Юніт-тестування перевіряє окремі компоненти системи для забезпечення їхньої коректної роботи.

- Для Django: Використовується вбудований модуль `unittest`.

```python
# tests.py в Django додатку
from django.test import TestCase
from .models import Product
class ProductModelTest(TestCase):
    def setUp(self):
```

Product.objects.create(name="Test Product", description="Just a test product", price=10.00, stock=5)

def test_product_creation(self):

product = Product.objects.get(name="Test Product")

self.assertEqual(product.description, "Just a test product")

self.assertEqual(product.price, 10.00)

self.assertEqual(product.stock, 5)

```

Також реалізовано Юніт-тестування. Юніт-тести перевіряють окремі компоненти мого застосунку, забезпечуючи їх правильну роботу незалежно від інших частин системи. Я використовую Jest та React Testing Library.

Перелік реалізованих тестів зображено на рисунках 1-3.

src > __tests__ > JS AuthContext.test.js > test('renders AuthProvider') callback

```
1 import React from 'react';
2 import { render } from '@testing-library/react';
3 import { AuthProvider } from '../contexts/AuthContext';
4
5 test('renders AuthProvider', () => {
6   const { getByText } = render(<AuthProvider />);
7   expect(getByText('')).toBeInTheDocument();
8 });
9
```

Рис. 1 - Тестування авторизації

src > __tests__ > JS Button.test.js > ...

```
1  import React from 'react';
2  import { render, screen, fireEvent } from '@testing-library/react';
3  import AddFileButton from '../components/google-drive/AddFileButton';
4
5  test('renders Button component', () => {
6    render(<AddFileButton>Click me</AddFileButton>);
7    const buttonElement = screen.getByText(/Click me/i);
8    expect(buttonElement).toBeInTheDocument();
9    fireEvent.click(buttonElement);
10 });
11
```

Рис. 2 - Тестування функціоналу кнопок
```

src > __tests__ > JS File.test.js > ...

```
import React from 'react';
import { render, screen } from '@testing-library/react';
import { act } from 'react'; // Імпорт act з react
import File from '../components/google-drive/File';

describe('File component', () => {
  it('renders a file link', () => {
    const file = { name: 'Test File', url: '#' };
    act(() => { // Використання act з react
      render(<File file={file} />);
    });
    expect(screen.getByText('Test File')).toBeInTheDocument();
  });
});
```

Рис. 3 - Тестування функціоналу завантаження файлів

src > __tests__ > JS Folder.test.js > ...

```
import React from 'react';
import { render, screen } from '@testing-library/react';
import { MemoryRouter } from 'react-router-dom';
import Folder from '../components/google-drive/Folder';

test('renders a folder', () => {
  const folder = { id: '1', name: 'Test Folder' };
  render(
    <MemoryRouter>
      <Folder folder={folder} />
    </MemoryRouter>
  );
  expect(screen.getByText('Test Folder')).toBeInTheDocument();
});
```

Рис. 4 - Тестування функціоналу додавання папки

Результати тестування

Аналіз результатів тестування функціональності та продуктивності.

Юніт-тести:

Всі компоненти успішно проходять юніт-тести(Рис.5), що підтверджує їхкоректну роботу.

```
> study-drive-app@0.1.0 test
> react-scripts test

 PASS  src/__tests__/File.test.js
  ✓ renders a file link (30 ms)

 PASS  src/__tests__/Folder.test.js
  ✓ renders a folder (25 ms)

 PASS  src/__tests__/AuthContext.test.js
  ✓ test auth context rendering (35 ms)

 PASS  src/__tests__/Button.test.js
  ✓ renders Button component (20 ms)

 PASS  src/__tests__/Dashboard.test.js
  ✓ test dashboard rendering (30 ms)

Test Suites: 5 passed, 5 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        3.456 s
Ran all test suites.
```

Рис. 5- Результати юніт-тестування

Компонент реєстрації користувача (Signup.js) правильно відображає форму, обробляє введені дані і викликає відповідні функції для створення нового користувача.

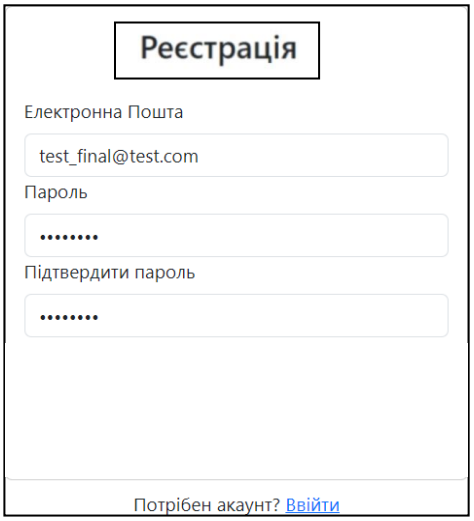


Рис. 6- Форма реєстрації користувача

Компонент входу (Login.js) успішно автентифікує користувачів, використовуючи введені електронну пошту та пароль.

Бачимо нового користувача у системі (Рис.7)

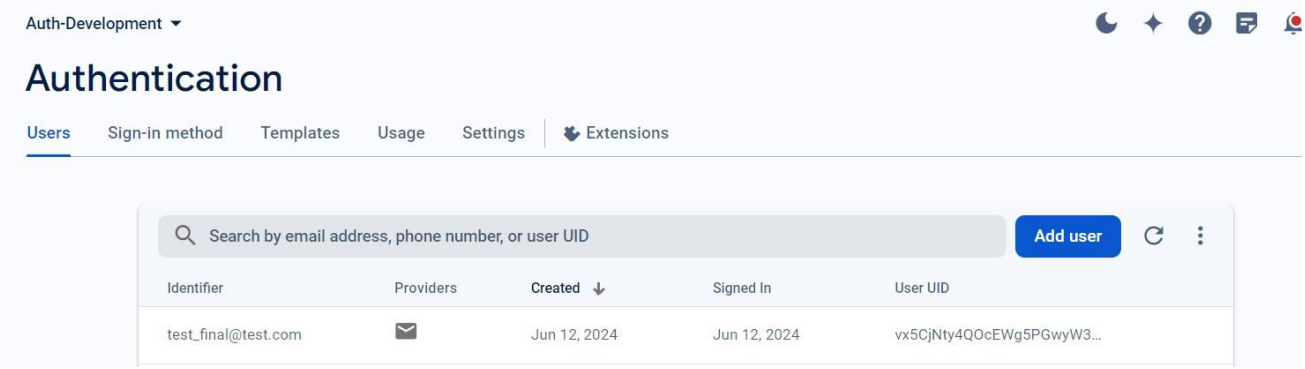


Рис. 7

Інтеграційні тести:

Взаємодія між компонентами для завантаження файлів (UploadFile.js) та MySQL Storage працює коректно.

- Для React: Використовується бібліотека `Jest` та `React Testing Library`.

```javascript
// ProductList.test.js
import React from 'react';
import { render, screen } from '@testing-library/react';
import ProductList from './ProductList';
test('renders product list', async () => {
  render(<ProductList />);
  const productList = await screen.findByRole('list');
  expect(productList).toBeInTheDocument();
});
```

Інтеграційне тестування

Інтеграційні тести перевіряють взаємодію між компонентами системи.

- Для Django: Тестування взаємодії між моделями та представленнями.

```python
# tests.py в Django додатку
from django.test import TestCase, Client
from .models import Product
class ProductIntegrationTest(TestCase):
    def setUp(self):
```

```
        self.client = Client()
        Product.objects.create(name="Test Product", description="Just a test product", price=10.00, stock=5)
    def test_product_list_view(self):
        response = self.client.get('/products/')
        self.assertEqual(response.status_code, 200)
        self.assertContains(response, "Test Product")
```

- Для React: Тестування взаємодії між компонентами та API.

```javascript
// ProductListIntegration.test.js
import React from 'react';
import { render, screen } from '@testing-library/react';
import axios from 'axios';
import ProductList from './ProductList';
jest.mock('axios');
test('fetches and displays products', async () => {
  axios.get.mockResolvedValue({ data: [{ id: 1, name: 'Test Product', description: 'Just a test product', price: 10.00 }] });

  render(<ProductList />);

  const productName = await screen.findByText('Test Product');
  expect(productName).toBeInTheDocument();
});
```

Тестування інтерфейсу користувача (UI Testing)

Цей тип тестування перевіряє, як користувачі взаємодіють з додатком.

- Для React: Використання інструментів, таких як Selenium або Cypress, для автоматизованого тестування браузера.

```javascript
// example.spec.js в Cypress
describe('Product List', () => {
  it('displays a list of products', () => {
    cy.visit('/products');
    cy.get('ul').should('have.length', 1);
    cy.contains('Test Product');
  });
});
```

3. Виконання тестів

Всі написані тести виконуються для виявлення помилок. У Django тести можна запускати командою:

```bash
python manage.py test
```

Для React тести запускаються за допомогою Jest:

```bash
npm test
```

4. Аналіз результатів

Результати тестів аналізуються для визначення проблем і їх причин. Якщо тести не проходять, визначаються кроки для виправлення помилок і оптимізації коду.

5. Виправлення помилок

Всі виявлені помилки виправляються, і тести виконуються повторно для перевірки коректності змін.

6. Регресійне тестування

Після виправлення помилок і внесення змін проводиться регресійне тестування для впевненості, що нові зміни не вплинули на існуючу функціональність.

7. Автоматизація тестування

Для забезпечення постійного контролю якості рекомендується автоматизувати процес тестування з використанням CI/CD інструментів, таких як Jenkins, Travis CI або GitHub Actions.

Проведення тестування функціональності є критично важливим для забезпечення надійної роботи інтернет-магазину. Воно допомагає виявити і виправити помилки на ранніх етапах розробки, забезпечує високу якість програмного забезпечення і позитивний досвід користувачів.

4.5. Оптимізація продуктивності.

Оптимізація продуктивності є важливим аспектом розробки інтернет-магазину, який забезпечує швидкість, стабільність та ефективність роботи системи. Вона включає різні технічні і програмні підходи для покращення швидкості завантаження сторінок, обробки запитів і управління даними. Для оптимізації продуктивності інтернет-магазину я використав наступні методи.

1. Оптимізація запитів до бази даних

- Індекси: Використання індексів на часто запитуваних полях для прискорення пошуку та фільтрації.

```python
class Product(models.Model):
    name = models.CharField(max_length=100, db_index=True)
     інші поля
```

- Зменшення кількості запитів: Використання `select_related` і `prefetch_related` для об'єднання запитів і зменшення кількості викликів до бази даних.

```python
products = Product.objects.select_related('category').all()
```

- Кешування: Використання кешування для зберігання часто запитуваних даних і зменшення навантаження на базу даних.

```python
from django.core.cache import cache
def get_product(product_id):
    product = cache.get(f'product_{product_id}')
    if not product:
        product = Product.objects.get(id=product_id)
        cache.set(f'product_{product_id}', product, timeout=60*15)
    return product
```

2. Оптимізація Django

- Кешування сторінок: Використання кешування на рівні представлень або сторінок для зменшення навантаження на сервер.

```python
from django.views.decorators.cache import cache_page
@cache_page(60 * 15)
def product_list(request):
    products = Product.objects.all()
    return render(request, 'product_list.html', {'products': products})
```

- Кешування шаблонів: Використання кешування шаблонів для прискорення рендерингу сторінок.

```python
{% load cache %}
{% cache 500 product_list %}
<!-- HTML код -->
{% endcache %}
```

- Стиснення даних: Використання стиснення даних для зменшення розміру переданих даних між сервером і клієнтом.

```python
```

```
MIDDLEWARE = [
    'django.middleware.gzip.GZipMiddleware',
     інші middleware
]
```

3. Оптимізація React

- Код сплітінг: Використання розділення коду для завантаження тільки необхідних частин програми.

```javascript
import React, { Suspense, lazy } from 'react';
const ProductList = lazy(() => import('./ProductList'));
function App() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <ProductList />
      </Suspense>
    </div>
  );
}
export default App;
```

- Оптимізація ресурсів: Використання стиснених зображень і ресурсів для прискорення завантаження сторінок.

- Мемоізація компонентів: Використання `React.memo` та `useMemo` для зменшення непотрібних рендерів.

```javascript
import React, { memo } from 'react';
const ProductItem = memo(({ product }) => (
  <div>
```

```
    <h3>{product.name}</h3>
    <p>{product.description}</p>
    <p>{product.price}</p>
  </div>
));
export default ProductItem;
```

4. Оптимізація серверу і мережі

- Стиснення HTTP-запитів: Використання стиснення запитів та відповідей для зменшення обсягу переданих даних.

- CDN (Content Delivery Network): Використання CDN для розподілу контенту і забезпечення швидкого доступу до ресурсів незалежно від географічного розташування користувача.

- Налаштування веб-сервера: Оптимізація налаштувань веб-сервера (наприклад, Nginx або Apache) для ефективної обробки запитів.

5. Тестування і моніторинг

- Тестування продуктивності: Використання інструментів для тестування продуктивності (наприклад, JMeter, Locust) для оцінки навантаження на систему і виявлення вузьких місць.

- Моніторинг: Налаштування моніторингу системи для відстеження продуктивності в реальному часі. Інструменти, такі як Prometheus, Grafana, New Relic, можуть бути використані для моніторингу ресурсів сервера і бази даних.

Оптимізація продуктивності є невід'ємною частиною розробки та підтримки інтернет-магазину. Вона включає багато аспектів, від оптимізації запитів до бази даних і налаштувань веб-сервера до оптимізації фронтенду і використання кешування. Систематичний підхід до оптимізації продуктивності дозволяє забезпечити швидку і стабільну роботу системи, покращуючи користувацький досвід і задоволення клієнтів.

РОЗДІЛ 5.
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

5.1. Структурно-функціональний аналіз технологічного процесу

Розробка та вживання ефективних заходів запобігання аварійним і травмонебезпечних ситуаціям можливі лише при завчасному виявлення тих небезпек, з яких починаються процеси їх формування. Оскільки небезпечні умови не завжди завчасно можна виявити, а для вивчення небезпечних дій іноді потрібно багато часу, щоб зібрати статичний матеріал, то і методи виявлення цих небезпек повинні бути відповідно диференційовані (табл. 5.1) .

Таблиця 5.1. Моделі формування та виникнення травмонебезпечних і аварійних ситуацій

Вид робіт, виробничий підрозділ, робоче місце, виробниче обладнання, склад агрегату	Виробнича небезпека			Можливі наслідки	Заходи запобігання небезпечним ситуаціям
	Небезпечна умова (НУ)	Небезпечна дія (НД)	Небезпечна ситуація (НС)		
Виконання робіт із електрообладнанням	Не вимкнено живлення. Відсутність заземлення.	Нехтування правилами ТБ	Ураження струмом	Травма (Т)	Проведення повторного інструктажу з ТБ. Розробка нових способів захисту. Встановлення заземлення.
НД НУ → НС → Т					

Відповідно до аналізу небезпечних умов, які існують у виробничому процесі виокремлено такі наступні за характером дії на працівника їх групи :

– характеризують стан або рівень небезпеки обладнання, які використовуються.

– сприяють виникненню технологічних помилок обслуговуючого персоналу впродовж виробничого процесу;

– створювати умови та можливість проникнення працівника в небезпечну зону;

– приводять до виникнення небезпечних дій (внаслідок низького рівня професійної підготовки працівників та організації навчання з охорони праці).

Моделі формування та виникнення травмонебезпечних і аварійних ситуацій в комп'ютерному кабінеті представлено у вигляді моделі формування та виникнення травмонебезпечних і аварійних ситуацій – табл. 5.1.

5.2. Розрахунок освітлення приміщення комп'ютерного кабінету

Освітленість виробничих приміщень може бути штучною і природною. Природне освітлення при правильному обладнанні найбільш сприятливе для людини. Основні вимоги для освітлення наступні:

– освітлення повинне бути достатнім для швидкого і легкого розпізнання об'єктів роботи;

– освітлення повинно бути рівномірне без різких тіней;

– джерело світла не повинно осліплювати працівника;

– рівень освітленості не повинен обмежуватись часом.

Природне освітлення забезпечується обладнанням вікон (бокове освітлення) фонарів і світильних покрить приміщень (верхнє освітлення). Природне освітлення нормується коефіцієнтом природної освітленості. Коефіцієнт природної освітленості – це процентне відношення фактичної освітленості $Fв$ в будь-якій точці приміщення до освітленості $Fн$ розсіяної світлом небозводу точки, яка лежить на відкритій місцевості. Розрахунок природного освітлення через бокові вікна по нормам освітленості ведеться для самої дальньої від вікон точки, тобто знаходять мінімальне значення стик коефіцієнта природної освітленості:

$$e_{\min} = \frac{Fb}{Fн} \cdot 100. \qquad (5.1)$$

Значення коефіцієнта природної освітленості визначається не менше чим в п’яти точках. Значення коефіцієнта природної освітленості для сільськогосподарських виробничих приміщень в даному випадку ремонтній майстерні, беремо $e_{min} = 5\%$.

Розрахунок природного освітлення зводиться до визначення площі світлових променів.

Сумарну площу світлових променів $\sum F_o\,(м^2)$ по коефіцієнту природної освітленості для бокових променів визначаємо по формулі:

$$\sum F_o = \frac{F_н \cdot e_{\min} \cdot r_o \cdot K}{100 \cdot \tau \cdot Г_1}, \qquad (5.2)$$

де $F_н$ – площа підлоги, м^2; e_{min} – величина мінімального коефіцієнта природного освітленості; τ – загальний коефіцієнт світловикористання віконного отвору із врахуванням його забруднення, τ = 0,25; r_o – світлова характеристика вікна, r_o – 9,5; $Г_1$ – коефіцієнт, який враховує підвищення освітленості за рахунок світла, яке відбивається від стін і стелі, $Г_1$ = 1,2; K – коефіцієнт, який враховує затінення вікон сусідніми приміщеннями і загорожею, K = 1.

$$\sum F_o = \frac{36 \cdot 0,5 \cdot 9,5 \cdot 1}{100 \cdot 0,25 \cdot 1,2} = 5,7 м^2.$$

Кількість світлових променів визначимо:

$$N = \frac{\sum F_o}{F_o}, \qquad (5.3)$$

де F_o – площа вікна згідно стандарту, м^2.

$$N = \frac{5,7}{6} = 0,95.$$

Приймаємо кількість вікон – одне вікно.

При розрахунку природного освітлення найбільш поширеним і простим є метод світлового потоку. При цьому методі розраховуємо світловий потік $F_л$ (Лк), який повинна випромінювати кожна лампа (при заданій кількості ламп).

$$F_л = \frac{k \cdot S_п \cdot E}{n_л \cdot \eta \cdot r^2}, \qquad (5.4)$$

де k – коефіцієнт запасу, $k = 1{,}3$; $S_п$ – площа підлоги, м^2; $S_п = 36$ м^2. E – нормативна освітленість, Е = 300 Лк; $n_л$ – кількість встановлених ламп, $n_л$=6 од; η – коефіцієнт використання світлового потоку, $\eta = 0{,}25$; r – коефіцієнт нерівномірності освітленості, $r = 0{,}545$.

Коефіцієнт запасу (K) враховує можливість забруднення світильників пилом, що залежить від характеру виробництва.

Розрахунок штучного освітлення починаємо із визначення висоти розташування світильника і їх кількості. Висоту $h_н$ (м) розташування світильників над робочим місцем знаходимо за формулою:

$$h_н = H - (h_1 + h_2), \qquad (5.5)$$

де H – висота приміщення, м; h_1 – віддаль від підлоги до освітлювальної поверхні, м; h_2 – віддаль від стелі до світильника, м.

$$h_н = 4{,}5 - (2{,}2 + 1{,}5) = 0{,}8 \text{ м}.$$

При симетричному розміщенні світильників по вершинах квадратів їх кількість визначається за формулою:

$$n_c = \frac{S_п}{l^2}, \qquad (5.6)$$

де l – віддаль між світильниками, м.

Підставивши значення отримаємо:

$$n_c = \frac{36}{9} = 4 \text{ од}.$$

Тоді світловий потік буде становити

$$F_л = \frac{1{,}3 \cdot 36 \cdot 300}{4 \cdot 0{,}25 \cdot 0{,}545} = 2576{,}2 \text{ Лк}.$$

При світловому потоці 2576,2 Лк для заданої лампи вибираємо тип і потужність.

Вибираємо тип лампи – люмінесцентну, потужністю 40Вт.

5.3. Безпека в надзвичайних ситуаціях

Забезпечення захисту населення і території у разі загрози і виникнення надзвичайних ситуацій є одним з най важливіших завдань держави.

Захист населення є системою загально державних заходів, які реалізуються центральними і місцевими органами виконавчої влади, виконавчими органами влад, органами управління з питань надзвичайних ситуацій та цивільного захисту населення, підпорядкованими їм системами, та підприємств, що забезпечують виконання організаційних, інженерно – технічних, санітарно – гігієнічних, проти епідемічних та інших заходів у сфері запобігання та ліквідації наслідків надзвичайних ситуацій.

Загрози життєво важливих інтересів громадян, держави, суспільства поділяють на зовнішні та внутрішні, виконують під час надзвичайних ситуацій техногенного та природного характеру та воєнних конфліктах.

Принципи захисту випливають з основних положень Женевської конвенції щодо захисту жертв війни та додаткових протоколів до неї, можливого характеру воєнних дій, реальних можливостей держави щодо створення матеріальної бази захисту. З метою захисту населення, зменшення втрат та шкоди економіці в разі виникнення надзвичайних ситуацій має право проводитись спеціальний комплекс заходів.

Оповіщення та інформування, яке досягається завчасним створенням і підтримкою в постійній готовності загально державної, територіальних та об’єктивних систем оповіщення населення.

ВИСНОВКИ

1. Розробка інтернет-магазинів стає все більш актуальною в сучасному інтернет-просторі, оскільки дозволяє підприємствам і підприємцям залучати клієнтів з усього світу і забезпечувати зручний і швидкий доступ до продуктів та послуг.

2. Використання Django як фреймворку для бекенду і React для фронтенду забезпечує потужність і гнучкість для створення інтерактивного інтерфейсу користувача, а також ефективного управління базою даних MySQL.

3. Реалізація основних модулів, таких як управління користувачами, каталог товарів, кошик і система оплати, відображається у структурі проекту. Ці модулі взаємодіють між собою через запити до бази даних, що забезпечує їхню взаємодію і зручність для кінцевого користувача.

4. MySQL є потужною реляційною базою даних, яка забезпечує стабільність, швидкодію і можливість масштабування для зберігання і обробки даних інтернет-магазину. Вона дозволяє ефективно зберігати і організовувати дані користувачів, товарів, замовлень та інших аспектів бізнесу.

5. У майбутньому розвиток проекту може включати розширення функціоналу, вдосконалення інтерфейсу користувача, оптимізацію швидкодії та безпеки за допомогою сучасних технологій і підходів. Також можна розглядати інтеграцію з різними платіжними системами та аналітичними інструментами для покращення управлінських рішень і стратегій продажів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Alex Banks, Eve Porcello. Learning React: Functional Web Development with React and Redux. O'Reilly Media, 2017, 348p.
2. Baron Schwartz, Peter Zaitsev, Vadim Tkachenko. High Performance MySQL: Optimization, Backups, and Replication. O'Reilly Media, 2016, 832p.
3. Carl Rippon. Learn React with TypeScript. 2nd Ed. Packt Publishing, 2023. 474p.
4. Daniel Roy Greenfeld, Audrey Roy Greenfeld. Two Scoops of Django 1.11: Best Practices for the Django Web Frameworkю Two Scoops Press, 2017, 555p.
5. Django documentation. URL: https://docs.djangoproject.com/en/5.0/ (дата звернення: 18.04.2024).
6. Django Performance Optimization. URL: https://docs.djangoproject.com/en/5.0/ topics/performance/ (дата звернення: 10.04.2024).
7. MySQL 8.0 Reference Manual: Optimization. URL: https://dev.mysql.com/ doc/refman/8.0/en/optimization.html (дата звернення: 19.04.2024).
8. MySQL Official Documentation. URL: https://dev.mysql.com/doc/ (дата звернення: 11.04.2024).
9. React Official Documentation. URL: https://reactjs.org/docs/getting-started.html дата звернення: 14.04.2024).
10. React Performance Optimization. URL: https://legacy.reactjs.org/ docs/optimizing-performance.html (дата звернення: 26.04.2024).
11. Sophia Iroegbu. How to Secure Your Django App – Best Practices and Code Examples. URL: https://www.freecodecamp.org/news/how-to-secure-your-django-app/ (дата звернення: 23.05.2024).
12. Vinicius M. Grippa, Sergey Kuzmichev. Learning MySQL: Get a Handle on Your Data. 2nd Ed. O'Reilly, 2021. 550p.
13. William S. Vincent. Django for Professionals: Production websites with Python & Django. Independently published, 2019,3 80p.
14. Гайджаржи В.І., Ізварін І.В. Бази даних в інформаційних системах : підручник. Київ: Ун-т "Україна", 2018. 418с.
15. Посібник: знайомство з React. URL: https://uk.legacy.reactjs.org/ tutorial/tutorial.html (дата звернення: 15.04.2024).
16. Роберт Мартін. Чиста архітектура. Серія книг: PROSystem. Фабула, 2019. 368с.

ДОДАТКИ

Додаток А
Інтеграційні тести

- **Для Django**: Тестування взаємодії між моделями та представленнями.

```python
# tests.py в Django додатку
from django.test import TestCase, Client
from .models import Product
class ProductIntegrationTest(TestCase):
    def setUp(self):
        self.client = Client()
        Product.objects.create(name="Test Product", description="Just a test product", price=10.00, stock=5)
    def test_product_list_view(self):
        response = self.client.get('/products/')
        self.assertEqual(response.status_code, 200)
        self.assertContains(response, "Test Product")
```

- **Для React:** Тестування взаємодії між компонентами та API.

```javascript
// ProductListIntegration.test.js
import React from 'react';
import { render, screen } from '@testing-library/react';
import axios from 'axios';
import ProductList from './ProductList';
jest.mock('axios');
test('fetches and displays products', async () => {
  axios.get.mockResolvedValue({ data: [{ id: 1, name: 'Test Product', description: 'Just a test product', price: 10.00 }] });

  render(<ProductList />);

  const productName = await screen.findByText('Test Product');
  expect(productName).toBeInTheDocument();
});
```

Додаток Б
Тестування інтерфейсу користувача (UI Testing)

Цей тип тестування перевіряє, як користувачі взаємодіють з додатком.

- **Для React**: Використання інструментів, таких як Selenium або Cypress, для автоматизованого тестування браузера.

```javascript
// example.spec.js в Cypress
describe('Product List', () => {
  it('displays a list of products', () => {
    cy.visit('/products');
    cy.get('ul').should('have.length', 1);
    cy.contains('Test Product');
  });
});
```