КВАЛІФІКАЦІЙНА РОБОТА

Група ІПЗс-2019
Дронь І.Ю.

2023

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

на правах рукопису

Дронь Ігор Юрійович

УДК 004.378

Розробка веб-застосунку для продажу велосипедів з використанням технологій Spring boot

Спеціальність 121 – «Інженерія програмного забезпечення»

Кваліфікаційна робота на здобуття кваліфікації бакалавра

Нормоконтроль

_____Стисло О.В.

(підпис, дата, розшифрування підпису)

Студент

_____Дронь І.Ю.

(підпис, дата, розшифрування підпису)

Допускається до захисту

Завідувач кафедри

_____к.т.н., доц. Пашкевич О.П.

(підпис, дата, розшифрування підпису)

Керівник роботи

_____ к.т.н., доц. Пашкевич О.П.

(підпис, дата, розшифрування підпису)

Івано-Франківськ – 2023

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

Освітній ступінь: «бакалавр»

Спеціальність: 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____

« __ » _____ 2023 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Дронь Ігор Юрійович

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи

Розробка веб-застосунку для продажу велосипедів з використанням технологій Spring boot

керівник роботи:

Пашкевич Олег Петрович, кандидат технічних наук

затверджена наказом вищого навчального закладу від « 11» листопада 2022 року № 155/1НВ

2. Термін подання студентом роботи 14.06.2023

3. Вихідні дані роботи: Мова програмування Java, HTML, CSS

4. Зміст кваліфікаційної роботи (перелік питань, які потрібно розробити)

1. Аналіз існуючих аналогів та постановка задач

2. Розробка моделі сайту та його функціоналу

3. Реалізація структури та функціоналу вебсайту

5. Дата видачі завдання 10.10.2022

КОНСУЛЬТАНТИ РОЗДІЛІВ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Розділ	Консультант (прізвище, ініціали та посада)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	Огляд програмного забезпечення онлайн магазинів	15.02.23	Виконано
2.	Розробка архітектури клієнт-сервер	14.03.23	Виконано
3.	Програмна реалізація застосунку	07.04.23	Виконано
4.	Формування висновків	28.04.23	Виконано
5.	Оформлення пояснювальної записки	05.05.23	Виконано
6.	Оформлення графічного матеріалу та підготовка до захисту роботи	19.05.23	Виконано

Студент _____ Дронь І.Ю.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Пашкевич О.П.
(підпис) (прізвище та ініціали)

Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Сторінка	Опис графічного матеріалу	Сторінка	Опис графічного матеріалу
27	Діаграма звязків таблиць		
34	Головна сторінка (header)		
34	Головна сторінка (footer)		

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці сайту для продажу велосипедів з використанням мови програмування Java і її фреймворку Spring boot та Spring security.

В першому розділі було розглянуто: існуючі онлайн магазини – їх переваги та недоліки , загальний опис онлайн магазинів – обовязковий функціонал онлайн магазинів , а також були обрані задачі для проекту.

В другому розділі було написано user stories , що потрібно для кращого розуміння зручності та інтуітивності . Також було описано інструментарій Spring і логіка реєстрації і авторизації в Spring security . Крім того було розглянуто структуру бази данних.

В третьму розділі в основному було розглянуто код форм і контроллерів , а також розробку інтерфейсу.

SUMMARY

The qualification work is devoted to the development of a website for the sale of bicycles using the Java programming language and its Spring boot and Spring security frameworks.

The first section considered: existing online stores - their advantages and disadvantages, a general description of online stores - mandatory functionality of online stores, and also selected tasks for the project.

In the second section, user stories were written, which is necessary for a better understanding of convenience and intuitiveness. The Spring toolkit and the logic of registration and authorization in Spring security were also described. In addition, the structure of the database was considered.

In the third section, the code of forms and controllers, as well as the development of the interface, were mainly considere

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі, де інтернет став невід'ємною частиною нашого життя, розробка онлайн магазинів стала надзвичайно актуальною для різних видів бізнесу. Один з таких видів - це магазини велосипедів, які набули значної популярності серед любителів активного способу життя, спорту та просто людей, які цінують здоровий спосіб пересування. Розглянемо деякі з переваг та актуальності розробки онлайн магазину.

По-перше, доступність та зручність покупки. Онлайн магазин велосипедів дозволяє клієнтам придбати потрібний їм велосипед зручно та швидко. Немає необхідності їхати до фізичного магазину, шукати продукт із потрібними характеристиками, а потім стояти в черзі на касі. Замовлення можна зробити за декілька кліків, а доставка може бути організована прямо до дверей клієнта. Це робить процес покупки швидким, зручним та ефективним.

По-друге, широкий вибір товарів. Онлайн магазини велосипедів можуть запропонувати клієнтам набагато більший асортимент товарів порівняно зі звичайними магазинами. Тут можна знайти різні моделі велосипедів, включаючи гірські, шосейні, міські та електричні велосипеди. Клієнти мають можливість порівняти різні моделі, виробників, характеристики, ціни та відгуки користувачів, щоб зробити правильний вибір.

По-третє, інформаційна підтримка. Онлайн магазини велосипедів зазвичай надають детальну інформацію про кожен товар, включаючи його технічні характеристики, фотографії, огляди та відгуки від інших клієнтів. Це дозволяє клієнтам зробити освічений вибір, враховуючи свої потреби та вимоги.

По-четверте, зручність сервісу післяпродажного обслуговування. Багато онлайн магазинів велосипедів пропонують широкий спектр послуг післяпродажного обслуговування, включаючи консультації щодо використання, гарантійні ремонти та сервіс. Клієнти можуть звертатися за допомогою в будь-який час, використовуючи зручні способи зв'язку, такі як електронна пошта або онлайн-чат.

По-п'яте, економія часу та грошей. Онлайн магазин велосипедів дозволяє клієнтам порівнювати ціни та знаходити найкращі пропозиції без необхідності ходити по різних фізичних магазинах. Крім того, великий конкурентний тиск у світі електронної торгівлі може призводити до зниження цін на велосипеди та аксесуари, що дозволяє зекономити гроші.

Узагальнюючи, розробка онлайн магазину велосипедів є надзвичайно актуальною в сучасному світі. Вона забезпечує зручність та широкі можливості для клієнтів, дозволяє знайти оптимальний варіант велосипеда, отримати інформаційну підтримку та сервіс післяпродажного обслуговування. Крім того, розробка онлайн магазину велосипедів пропонує економію часу та грошей. Отже, якщо ви плануєте відкрити магазин велосипедів, необхідно врахувати ці переваги та актуальність онлайн платформи.

Мета роботи. Метою проекту "Онлайн магазин велосипедів" є створення зручної та ефективної електронної платформи, яка надає клієнтам можливість придбати велосипеди та велосипедні аксесуари шляхом онлайн-замовлення .

Головні цілі проекту включають:

- Розробка привабливого та інтуїтивно зрозумілого дизайну веб-сайту.
- Створення бази даних з асортиментом велосипедів та аксесуарів.
- Розробка системи замовлення та оплати через Інтернет.
- Забезпечення безпеки та конфіденційності даних клієнтів.
- Організація швидкої та надійної доставки товарів.

Об’єкт роботи. Продаж велосипедів , деталей та аксесуарів.

Предмет роботи. Продаж товарів через онлайн магазин.

Завдання роботи. Відповідно до обраної теми , робота має такі задачі:

- пошук та аналіз існуючих сайтів з аналогічними технологіями;
- поглиблене вивчення можливостей мови програмування java;
- вивчення можливостей фреймворка Spring boot;
- розробка сучасного та зручного дизайну;

- розробка веб-дизайну та архітектури сайту, відповідаючи принципам доступності;

- проведення тестування продукту.

Методи роботи. Для реалізації проекту та виконання поставлених завдань, було використано мову гіпертекстової розмітки HTML, каскадні таблиці стилів CSS та мову програмування Java, фреймворк до мови Java – Spring boot.

Результати роботи. В результаті виконання кваліфікаційної бакалаврської роботи був розроблений Web-сайт для продажу велосипедів різних типів , з урахуванням принципів доступності та універсальності .

Структура роботи. Розділи – 4. Загальний обсяг основної частини – 60 сторінок. Список використаних джерел – 15

РОЗДІЛ 1. ОПИС НАЯВНИХ ВЕБСАЙТІВ АНАЛОГІВ

1.1 Огляд та аналіз існуючих аналогів

У цьому огляді будуть розглянуті та проаналізовані деякі існуючі аналоги онлайн магазину велосипедів. Метою цього аналізу є отримання уявлення про сильні та слабкі сторони конкурентів, виявлення можливих можливостей для покращення переваг та особливостей вашого магазину.

1. Аналог 1: BikeShop.com

BikeShop.com є одним з провідних онлайн магазинів велосипедів, який пропонує широкий вибір велосипедів та аксесуарів. Сильні сторони BikeShop.com включають:

- велика кількість товарів: BikeShop.com має вражаючий асортимент велосипедів різних типів, моделей, розмірів і виробників;

- доступність і зручність: Клієнти можуть з легкістю переглядати товари, порівнювати характеристики та ціни, а також зробити замовлення безпосередньо на веб-сайті;

- система відгуків: BikeShop.com надає можливість клієнтам залишати відгуки товарів, що допомагає іншим покупцям зробити освідчений вибір.

Однак, слабкі сторони BikeShop.com також варто врахувати:

- конкурентна цінова політика: Ціни на BikeShop.com можуть бути трохи вищими порівняно з іншими конкурентами, що може вплинути на конверсію покупок;

- обмежена кількість інформації: Не всі товари мають детальну інформацію про їх характеристики, що може ускладнити прийняття рішення клієнтами.

2. Аналог 2: CycleWorld

CycleWorld є іншим відомим ігроком у сегменті

онлайн продажу велосипедів. Основні переваги CycleWorld включають:

- професійний дизайн та навігація: CycleWorld пропонує дуже зручний інтерфейс, що полегшує пошук та вибір велосипеда;

- зручна фільтрація: Клієнти можуть швидко фільтрувати товари за різними параметрами, такими як тип, розмір, ціна та бренд;

- персоналізовані рекомендації: CycleWorld використовує алгоритми рекомендацій, що допомагають клієнтам знайти велосипеди, які найбільше підходять їхнім потребам.

Однак, деякі слабкі сторони CycleWorld варто врахувати:

- обмежений вибір: В порівнянні з іншими конкурентами, асортимент велосипедів на CycleWorld може бути меншим;

- обмежені опції доставки: Доставка товарів може бути обмежена за географічним принципом.

3. Аналог 3: Velobay

Velobay - це менш відомий, але досить успішний онлайн магазин велосипедів, який спеціалізується на ексклюзивних та нішевих брендах. Деякі переваги Velobay включають:

- унікальні товари: Velobay пропонує вибір велосипедів, які важко знайти в інших магазинах, що привертає увагу унікальних клієнтів;

- висока якість обслуговування: Velobay прагне забезпечити високий рівень обслуговування клієнтів, надаючи індивідуальні консультації та підтримку.

Однак, Velobay також має свої слабкі сторони:

- вищі ціни: Виробники ексклюзивних велосипедів можуть надавати їх за вищою ціною,

що може обмежити широкий ринок покупців;

- обмежена наявність товарів: Завдяки нішевому спрямуванню, Velobay може мати обмежену кількість одиниць товару.

1.2 Загальний опис онлайн магазинів

Онлайн магазини є одним з найпоширеніших інтернет-проектів в сучасному цифровому світі. Вони надають можливість людям з усього світу здійснювати покупки зручно та ефективно, не виходячи з дому. У цьому тексті буде надано загальний опис онлайн магазинів, їхні переваги та особливості.

Основні характеристики онлайн магазинів

Онлайн магазини є віртуальними платформами, де продавці пропонують товари або послуги, а покупці можуть переглядати асортимент, здійснювати вибір та зробити покупку через Інтернет. Основні характеристики онлайн магазинів включають:

Доступність: Онлайн магазини доступні 24/7, що означає, що клієнти можуть здійснювати покупки в будь-який зручний для них час.

Зручність: Клієнти можуть швидко та з легкістю переглядати товари, порівнювати ціни, зчитувати відгуки, здійснювати покупки та оплату, не виходячи з дому.

Широкий вибір: Онлайн магазини можуть пропонувати широкий асортимент товарів або послуг з різних категорій, що задовольняє потреби різних клієнтів.

Глобальний доступ: Завдяки Інтернету, онлайн магазини можуть працювати на міжнародному рівні, забезпечуючи клієнтам з усього світу можливість здійснювати покупки.

Переваги онлайн магазинів

Онлайн магазини мають ряд переваг порівняно з традиційними крамницями. Деякі з них включають:

Значний економічний виграш: Онлайн магазинам не потрібно платити орендну плату за торгову площу, витрати на персонал та інші витрати, що дозволяє їм пропонувати товари за більш доступними цінами.

Миттєва доступність і швидкість: Клієнти можуть переглядати товари,

здійснювати покупки та отримувати їх без зайвих зусиль та затримок.

Більший вибір: Онлайн магазини мають можливість пропонувати широкий вибір товарів, включаючи ті, які можуть бути складніше знайти в традиційних крамницях.

Зручність: Покупці можуть здійснювати покупки з будь-якого місця та в будь-який зручний для них час, використовуючи комп'ютер або смартфон.

Особливості онлайн магазинів

Онлайн магазини мають свої особливості, які варто враховувати:

Важливість дизайну та користувацького досвіду: Дизайн інтерфейсу магазину повинен бути привабливим та легким у використанні, щоб залучити багато нових клієнтів.

Забезпечення безпеки: Онлайн магазини повинні забезпечувати безпеку особистої інформації та фінансових операцій клієнтів.

Якість зображень та описів: Зображення та описи товарів повинні детальними, щоб дати клієнтам повну інформацію про продукт перед покупкою.

Клієнтська підтримка: Онлайн магазини повинні мати зручні канали комунікації для клієнтів, які шукають допомогу або мають запитання.

Розвиток онлайн магазинів

Онлайн магазини постійно розвиваються, адаптуючись до змінних потреб клієнтів та нових технологій. Деякі тенденції, що впливають на розвиток онлайн магазинів, включають:

Зростання мобільного шопінгу: За останні роки значно зросла популярність мобільних пристроїв для здійснення покупок. Онлайн магазини повинні бути оптимізовані для мобільних пристроїв та мати додатки, які спрощують процес покупок на смартфонах і планшетах.

Персоналізація та рекомендації: Застосування алгоритмів машинного навчання дозволяє онлайн магазинам надавати персоналізовані рекомендації та пропозиції, враховуючи індивідуальні потреби та попередні покупки клієнтів.

Використання відео та віртуальної реальності: Деякі онлайн магазини

використовують відео та віртуальну реальність для показу товарів у деталях та створення більш іммерсивного досвіду покупки.

Загалом, онлайн магазини є важливою складовою сучасного ринку, що надає клієнтам зручність, широкий вибір та ефективність у здійсненні покупок. Їхній постійний розвиток та адаптація до нових тенденцій робить їх ще більш привабливими для споживачів. Онлайн магазини стали популярними завдяки своїм численним перевагам, таким як доступність 24/7, зручність та широкий вибір товарів. Вони забезпечують зручність і ефективність, дозволяючи покупцям придбати товари безпосередньо через Інтернет. Онлайн магазини пропонують різноманітні товари, від електроніки та одягу до косметики та продуктів харчування.

Основною перевагою онлайн магазинів є їхня доступність - вони доступні цілодобово, що дозволяє покупцям здійснювати покупки у зручний для них час. Крім того, онлайн магазини пропонують широкий вибір товарів, а клієнти можуть легко порівнювати ціни та характеристики товарів, переглядаючи відгуки попередніх покупців. Онлайн магазини також зручні для тих, хто не має можливості відвідувати фізичні магазини, наприклад, через велику відстань або фізичні обмеження.

Для того, щоб здійснити покупку в онлайн магазині, покупець зазвичай шукає товар на веб-сайті магазину, додає його до кошика і оформляє замовлення, надаючи необхідну інформацію для доставки та оплати. Після підтвердження замовлення покупець зазвичай отримує підтвердження по електронній пошті та вказівки щодо способу оплати та доставки.

У процесі зростання популярності онлайн магазинів з'явилися різноманітні електронні платіжні системи, які дозволяють здійснювати оплату зручним для покупця способом, таким як кредитні картки, електронні гаманці або платіжні системи, такі як PayPal.

Важливо зазначити, що покупці повинні бути уважними при здійсненні покупок в онлайн магазинах. Рекомендується перевіряти репутацію магазину,

читати відгуки попередніх покупців та бути обережними під час надання особистої інформації та оплати.

1.3 Постановка задач

Розробка онлайн магазину є складним та відповідальним процесом, який потребує чіткого визначення поставлених завдань. Правильна постановка задач є ключовим етапом у створенні успішного та ефективного інтернет-магазину. У даному тексті будуть розглянуті основні аспекти та напрями постановки задач при розробці онлайн магазину.

1.Визначення бізнес-цілей

Першим етапом постановки задач є чітке визначення бізнес-цілей вашого онлайн магазину. Це означає визначення основних завдань, які ви хочете досягти за допомогою свого магазину. Наприклад, це можуть бути такі цілі, як збільшення обсягу продажів, залучення нових клієнтів, покращення взаємодії з клієнтами тощо. Визначення бізнес-цілей допоможе вам зорієнтуватись і визначити основні напрями роботи над розробкою магазину.

2.Аналіз цільової аудиторії

Другим важливим кроком є аналіз вашої цільової аудиторії. Ви повинні зрозуміти, хто є вашими потенційними клієнтами, які їхні потреби та очікування, що їх мотивує до покупки в інтернеті. Це допоможе вам налаштувати ваш магазин на відповідну аудиторію, пропонувати їм релевантні товари та послуги, а також покращити користувацький досвід.

3.Функціональні вимоги та функціонал магазину

На цьому етапі ви повинні визначити функціональні вимоги до вашого онлайн магазину. Це означає складання переліку основних функцій, які повинні бути доступні в вашому магазині. Наприклад, це можуть бути функції, такі як реєстрація користувачів, пошук товарів, додавання товарів до кошика, онлайн-оплата, огляд замовлень, можливість залишати відгуки та рейтинги, підтримка

клієнтів тощо. Важливо врахувати специфіку вашого бізнесу та потреби цільової аудиторії при визначенні функціоналу магазину.

4.Дизайн та користувацький досвід

Дизайн вашого онлайн магазину грає важливу роль у привертанні та утриманні клієнтів. Постановка задач з дизайну повинна включати визначення загального стилю та настрою магазину, розміщення елементів на сторінці, кольорову палітру, вибір шрифтів та графічних елементів. Крім того, важливо забезпечити зручний та інтуїтивно зрозумілий користувацький досвід, що дозволить клієнтам легко здійснювати покупки та навігацію по сайту.

5.Безпека та захист даних

Один з найважливіших аспектів розробки онлайн магазину - це забезпечення безпеки та захисту персональних даних клієнтів. Постановка задач з цього питання повинна включати встановлення надійних механізмів шифрування, захисту від несанкціонованого доступу та злому, а також резервне копіювання даних. Важливо використовувати надійні платіжні шлюзи та додержуватись вимог законодавства щодо захисту персональних даних.

6.Бекенд на Spring Boot:

Spring Boot - це фреймворк для розробки Java додатків, який спрощує і прискорює процес створення високоякісних і масштабованих веб-додатків. Він забезпечує широкий функціонал для розробки серверної частини додатків, такий як керування залежностями, обробка HTTP-запитів, робота з базами даних та безпека.

Вибір Spring Boot для реалізації бекенду вашого онлайн магазину має кілька переваг. Перш за все, він забезпечує ефективне керування залежностями, що дозволяє легко і швидко додавати необхідні модулі та бібліотеки для вашого проекту. Крім того, Spring Boot надає велику кількість вбудованих функцій для розробки веб-додатків, таких як контролери, сервіси, репозиторії, аутентифікація та авторизація, що спрощує розробку бекенду.

7.Фронтенд на HTML і CSS:

HTML (HyperText Markup Language) і CSS (Cascading Style Sheets) є основними мовами для створення та стилізації веб-сторінок. HTML відповідає за структуру та розмітку контенту, включаючи текст, зображення та посилання. CSS використовується для задання зовнішнього вигляду веб-сторінок, таких як кольори, шрифти, розташування елементів та анімація.

Використання HTML і CSS для реалізації фронтенду вашого онлайн магазину є розумним вибором, оскільки ці мови є стандартом для веб-розробки і мають широку підтримку в різних браузерах. HTML надає вам можливість створювати структуровані та доступні веб-сторінки, а CSS дозволяє вам візуально оформити ці сторінки згідно вашого бренду та дизайну.

Однак, варто відзначити, що розробка фронтенду на HTML і CSS може бути вимогливою і вимагати досвіду веб-розробки для досягнення професійного рівня. У разі потреби в складному інтерфейсі або динамічних елементах на сторінці, ви також можете розглянути використання JavaScript, який надає можливості для інтерактивності та динамічної поведінки веб-сторінок.

Важливо також пам'ятати, що розробка бекенду та фронтенду - це взаємопов'язані процеси, які потребують злагодженої роботи. Ви маєте забезпечити правильну взаємодію між бекендом і фронтендом, наприклад, за допомогою API, яке дозволяє обмінюватися даними між сервером і клієнтом.

8.Інтеграція зі сторонніми сервісами

У залежності від ваших бізнес-потреб, можуть бути потрібні інтеграції зі сторонніми сервісами. Наприклад, це можуть бути платіжні системи, системи управління відвантаженнями, системи обліку та аналітики, поштові сервіси тощо. Постановка задач з інтеграції повинна визначити необхідні сервіси та забезпечити їх правильну настройку та взаємодію з магазином.

9.Тестування та вдосконалення

Останнім етапом в постановці задач є розробка плану тестування та вдосконалення магазину. Тестування допоможе виявити та виправити можливі помилки та проблеми перед запуском магазину. Важливо планувати час для

вдосконалення та оновлення функціоналу магазину з урахуванням отриманого фідбеку від користувачів та змін на ринку.

Узагальнюючи, постановка задач при розробці онлайн магазину є важливим етапом, який визначає напрямки роботи та вимоги до магазину. Важливо чітко визначити бізнес-цілі, аналізувати цільову аудиторію, визначати функціонал та дизайн, забезпечувати безпеку даних, інтегрувати зі сторонніми сервісами, а також проводити тестування та вдосконалення. Правильна постановка задач допоможе створити успішний та конкурентоспроможний онлайн магазин.

Висновки до розділу 1

У даному розділі було проведено огляд програмного забезпечення, яке використовують онлайн магазини. Вивчені були три сайти для продажу велосипедів, зокрема BikeShop.com, CycleWorld ,Velobay.

Кожен з цих сайтів пропонує детальну інформацію про різні види товарів. Кожен товар має свою окрему сторінку з описом цього товару, фотографіями та цінами , що допомагає користувачам зробити вибір і зрозуміти, чого очікувати.

На кожному сайті є розділи для велосипедів , аксесуарів та деталей. На цих сторінках представлені фотографії та коротка інформація про товар, що дозволяє користувачу зразу зрозуміти чи відповідає товар його вимогам і чи доцільно дізнаватись більше про цей товар.

Порівнявши розглянуті сайти-аналоги, можна зробити висновок, що наявність можливості онлайн покупки є серйозною перевагою, яка покращує зручність та ефективність як для власників сайту так і для клієнтів.

РОЗДІЛ 2. Розробка архітектури клієнт-серверної аплікації для реєстрації пацієнтів

2.1 Створення варіантів використання сайту

Юзер-сторі (англ. User Stories) - це інструмент, що використовується в агільному розробленні програмного забезпечення для опису функціональних вимог до системи з точки зору користувача. Вони допомагають зосередитися на потребах та цілях користувачів і служать основою для розробки продукту.

Юзер-сторіси виражаються у форматі коротких, простих речень, які описують бажані дії користувача та очікуваний результат. Зазвичай використовується наступна структура:

"Як [тип користувача], я хочу [дія], щоб [результат][10]."

Основна мета юзер-сторіс - зосередитися на потребах користувача та описати їх у зрозумілій формі для всіх учасників розробки. Вони забезпечують розуміння того, що система повинна робити, і слугують основою для планування роботи, пріоритезації функціональності та оцінки зусиль, необхідних для реалізації.

Основні переваги використання юзер-сторіс:

Фокус на користувачеві: Юзер-сторіси ставлять користувача в центр розробки, допомагаючи зрозуміти його потреби та очікування.

Простота та зрозумілість: Формат юзер-сторіс дуже простий, що полегшує їх створення і розуміння всіма учасниками проекту.

Покращення комунікації: Використання юзер-сторіс допомагає забезпечити чіткість та однозначність вимог, сприяючи кращій комунікації між командою розробників, бізнес-аналітиками та замовником.

Пріоритезація: Юзер-сторіси допомагають визначити найважливіші функціональність і зробити правильні рішення щодо розподілу ресурсів.

Відстеження прогресу: Кожна юзер-сторі має чіткий критерій завершення, що дозволяє відстежувати прогрес розробки.

Наприклад, припустимо, що у вас є онлайн магазин велосипедів. Один з можливих юзер-сторіс може мати наступний вигляд:

"Як покупець, я хочу мати можливість додати вибраний велосипед у кошик, щоб зберегти його перед оформленням замовлення [2]."

Цей юзер-сторіс вказує на бажання користувача додати велосипед у кошик і підкреслює важливість цієї функції для користувачів вашого магазину.

Юзер-сторіси є потужним інструментом управління вимогами та розробки продукту, що допомагають зосередитися на потребах користувачів та забезпечують більше задоволення від використання системи.

Такі варіанти було створено:

1.Як користувач, я хочу переглядати каталог велосипедів, щоб знайти відповідну модель для мене;

2.Як користувач, я хочу фільтрувати велосипеди за різними параметрами (тип, розмір, колір, ціна тощо), щоб звузити вибір;

3.Як користувач, я хочу переглядати деталі велосипедів, включаючи опис, технічні характеристики та фотографії, щоб отримати інформацію про товар;

4.Як користувач, я хочу додавати велосипеди до кошика, щоб зберігати їх для майбутніх покупок;

5.Як користувач, я хочу здійснювати оплату велосипедів онлайн, використовуючи безпечні платіжні методи;

6.Як користувач, я хочу відстежувати статус своїх замовлень, щоб бути в курсі процесу доставки;

7.Як користувач, я хочу мати можливість звертатися до служби підтримки або задавати питання щодо продуктів чи замовлень;

8.Як адміністратор, я хочу мати доступ до панелі адміністрування, щоб керувати каталогом велосипедів, оновлювати інформацію про товари та виконувати інші адміністративні завдання;

9.Як адміністратор, я хочу отримувати повідомлення про нові замовлення та керувати процесом обробки і доставки;

10.Як адміністратор, я хочу мати можливість керувати користувачами, змінювати їхні дані та вирішувати питання, пов'язані з обліковими записами;

2.2 Інструментарій Spring boot

Spring Boot є потужним фреймворком, який надає розробникам зручність та ефективність під час розробки веб-додатків на мові програмування Java. Однією з ключових особливостей Spring Boot є його інструментарій, який допомагає розробникам створювати додатки швидко та ефективно [7-10].

Один з основних компонентів інструментарію Spring Boot - це Spring Boot Starter. Це набір залежностей, які автоматично налаштовуються для конкретного типу додатка. Spring Boot Starter надає легку і зручну можливість підключати різні модулі Spring Framework, такі як Spring MVC для веб-розробки, Spring Data JPA для роботи з базами даних або Spring Security для забезпечення безпеки додатків. За допомогою Starter'ів, розробникам не потрібно вручну налаштовувати всі необхідні залежності, оскільки це вже зроблено [10,12].

Spring Boot також володіє рядом інструментів, які спрощують розробку та підтримку додатків. Наприклад, Spring Boot Actuator - це модуль, який надає готову функціональність для моніторингу та керування додатком. Завдяки Actuator, розробники можуть отримувати доступ до різноманітної інформації про додаток, такої як стан, метрики використання пам'яті, кількість запитів та інші статистичні дані. Це дозволяє вирішувати проблеми продуктивності, виявляти потенційні уразливості [10].

Інший важливий інструмент - Spring Boot DevTools. Він призначений для покращення процесу розробки, надаючи автоматичне перезавантаження додатку при зміні коду. Це дозволяє розробникам швидко бачити результати своїх змін

без необхідності вручну перезапускати додаток. DevTools також забезпечує автоматичну перекомпіляцію, що допомагає економити час під час розробки.

Spring Boot також надає різні інструменти для тестування додатків. Модуль Spring Boot Test дозволяє розробникам писати як модульні, так і інтеграційні тести з використанням різних технологій. Він забезпечує легкість налаштування тестів, включаючи можливість створювати тестові контейнери та конфігурацію баз даних для тестування.

Інший модуль - Spring Boot Data - спрощує взаємодію з базами даних. Він надає високорівневий підхід до роботи з реляційними та нереляційними базами даних, забезпечуючи гнучку інтеграцію з популярними технологіями, такими як JPA (Java Persistence API) або MongoDB. Spring Boot Data дозволяє розробникам просто визначати моделі даних та робити операції CRUD (створення, читання, оновлення, видалення) без необхідності вручну писати багато коду.

Spring Security є потужним фреймворком для забезпечення безпеки додатків, розроблених на платформі Java з використанням Spring Framework. Його основна мета - захистити додаток від різних загроз безпеки, таких як несанкціонований доступ, аутентифікація, авторизація, атаки злому, XSS (міжсайтовий скриптинг) та CSRF (міжсайтова подвійна заява). Spring Security надає широкий спектр функціональності та інструментів для досягнення цих цілей.

Одним з ключових компонентів Spring Security є аутентифікація і авторизація. Аутентифікація - це процес перевірки та підтвердження ідентичності користувача. Spring Security надає різні способи аутентифікації, включаючи базову аутентифікацію, форму аутентифікацію, аутентифікацію з використанням сторонніх систем (наприклад, OAuth або LDAP) та інші. Крім того, він підтримує налаштування різних механізмів аутентифікації, таких як двухфакторна аутентифікація, використання сертифікатів та інші.

Авторизація визначає, які ресурси та функціональність мають бути доступні для аутентифікованого користувача. Spring Security надає гнучкі засоби

для визначення прав доступу, включаючи базові ролі, ієрархії ролей, вирази SpEL (Spring Expression Language) та анотації. Розробники можуть точно налаштувати, які ресурси доступні для різних ролей користувачів, і встановити детальні правила авторизації [13-17].

Spring Security також підтримує захист від різних атак безпеки. Він надає захист від атак XSS шляхом екранування вхідних даних, захист від атаки CSRF шляхом генерації та перевірки токенів безпеки, контроль доступу до захищених ресурсів, обмеження кількості невдалих спроб входу та багато іншого.

Spring Security може інтегруватися з іншими модулями та інструментами Spring Framework, такими як Spring MVC, Spring Boot, Spring Data і Spring Cloud, що робить його потужним і універсальним рішенням для забезпечення безпеки великих Java-додатків [10].

Узагальнюючи, Spring Security є потужним фреймворком для забезпечення безпеки додатків на платформі Java. Він надає різноманітні можливості для аутентифікації, авторизації та захисту від різних загроз безпеки. За допомогою Spring Security розробники можуть створювати безпечні та надійні додатки, які захищені від вразливостей та атак.

Також варто зазначити Spring Boot Cloud, який дозволяє легко створювати та управляти масштабованими додатками в хмарних середовищах. Він надає інтеграцію з популярними хмарними платформами, такими як AWS, Azure, Google Cloud, і надає засоби для керування конфігурацією та моніторингу у хмарному середовищі. Завдяки Spring Boot Cloud, розробники можуть легко розгортати свої додатки в хмарі та отримувати всі переваги масштабування та надійності, які надаються хмарними платформами [10].

Spring Boot CLI (Command Line Interface) - це інструмент командного рядка, який дозволяє швидко створювати та запускати додатки Spring Boot. Він дозволяє розробникам швидко виконувати рутинні завдання, такі як створення нових проектів, запуск сервера, виконання тестів та багато іншого, без необхідності використання інтегрованого середовища розробки.

Крім того, існує інструмент Spring Boot Admin, який надає можливість моніторити та керувати додатками, побудованими на базі Spring Boot. Він дозволяє отримувати інформацію про стан та працездатність додатків, включаючи метрики, журнали подій та інше. Завдяки Spring Boot Admin, розробники можуть з легкістю відстежувати та аналізувати роботу своїх додатків [10].

Узагалі, Spring Boot надає багатий інструментарій, який спрощує розробку, тестування, моніторинг та розгортання додатків на основі Java. Завдяки своїй простоті використання, конфігурації та інтеграції з іншими модулями Spring, він є одним з найпопулярніших фреймворків для розробки веб-додатків. Величезна кількість вбудованих функцій, стандартизований підхід до розробки та добре співпрацюючі модулі роблять Spring Boot потужним інструментом для будь-якого розробника, який працює з Java [10].

2.3 Авторизація та аутентифікація за допомогою Spring Security

Реєстрація і аутентифікація є важливими компонентами Spring Security. Вони дозволяють користувачам отримати доступ до захищених ресурсів та функціональності додатка. Давайте розглянемо ці аспекти більш детально [11].

Реєстрація - це процес, за допомогою якого користувач створює обліковий запис у системі. Spring Security надає можливість налаштувати власний механізм реєстрації, який включає в себе створення користувача в базі даних або використання зовнішніх систем, таких як OAuth або LDAP, для аутентифікації та авторизації користувачів. Під час реєстрації можна вимагати від користувача заповнити обов'язкові поля, такі як ім'я, електронна пошта, пароль тощо. Spring Security забезпечує механізми для перевірки валідності та унікальності введених даних користувача [11].

Після реєстрації користувача, важливим етапом є аутентифікація, яка перевіряє, чи користувач є дійсним і чи має він право доступу до системи. Spring Security надає різні механізми аутентифікації, включаючи форму аутентифікації,

базову аутентифікацію, аутентифікацію на основі токенів та інші. Форма аутентифікації - це найбільш поширений спосіб аутентифікації, де користувач вводить свої облікові дані (наприклад, ім'я користувача та пароль) на веб-сторінці, яку надає додаток. Spring Security бере на себе перевірку цих облікових даних, шифрування пароля та управління процесом аутентифікації.

Після успішної аутентифікації користувача Spring Security надає йому доступ до захищених ресурсів та функціональності додатка. Це може включати обмеження доступу до певних URL-адрес, контроль дозволів на виконання певних операцій або обробку інших обмежень на основі ролей користувачів. Spring Security надає потужні механізми для налаштування прав доступу та управління ролями користувачів, дозволяючи гнучко налаштовувати рівень безпеки вашого додатка [11-12].

Крім того, Spring Security забезпечує підтримку різних додаткових функцій, пов'язаних з реєстрацією і аутентифікацією. Наприклад, можливо використовувати двофакторну аутентифікацію для забезпечення більш високого рівня безпеки, використовувати капчу або обмежити кількість невдалих спроб входу. Spring Security надає гнучкість для налаштування цих функцій залежно від ваших потреб та вимог додатка.

Узагалішуючи, Spring Security надає потужні інструменти для реєстрації і аутентифікації користувачів в Java-додатках. Цей фреймворк дозволяє налаштовувати різноманітні механізми реєстрації та аутентифікації, забезпечуючи безпеку та контроль доступу до захищених ресурсів. За допомогою Spring Security ви можете створювати безпечні та надійні додатки, які задовольняють вимоги безпеки вашого проекту.

2.4 Створення структури таблиць бази данних

MySQL є однією з найпопулярніших систем управління базами даних (СУБД) та використовується в широкому спектрі додатків, від невеликих веб-

сайтів до великих корпоративних систем. Давайте розглянемо переваги та недоліки MySQL порівняно з іншими базами даних.

Переваги MySQL:

Надійність: MySQL є дуже надійною СУБД і довів свою стабільність протягом багатьох років. Вона має високу стійкість до помилок, забезпечує відновлення даних та може працювати в умовах високого навантаження.

Простота використання: MySQL має простий у використанні синтаксис SQL, що робить його доступним для новачків. Легкість установки та налаштування також робить його привабливим для швидкого розгортання додатків.

Широке співробітництво: MySQL підтримує велику спільноту користувачів та розробників, що сприяє швидкому розробленню, вирішенню проблем та обміну знаннями. Існує багато документації, форумів та ресурсів, які допомагають вирішувати проблеми та вдосконалювати навички.

Підтримка реплікації: MySQL надає можливість реплікації даних, що дозволяє створювати резервні копії, забезпечувати високу доступність та розподілення навантаження. Це особливо корисно для великих проектів з високим обсягом трафіку.

Підтримка транзакцій: MySQL підтримує транзакції, що дозволяють забезпечити цілісність даних та відновлювати базу даних в разі виникнення помилок. Використовуючи операції COMMIT та ROLLBACK, можна забезпечити консистентність даних.

Масштабованість: MySQL добре масштабується як вертикально (за рахунок збільшення обсягу ресурсів на сервері), так і горизонтально (за допомогою розподілення бази даних на кілька серверів). Це дозволяє розширювати ресурси для забезпечення ефективності та продуктивності додатка.

Недоліки MySQL:

Обмежена підтримка деяких функцій: MySQL може бути обмежений в деяких аспектах, зокрема в підтримці складних операцій з базами даних, які

присутні в інших СУБД, таких як PostgreSQL або Oracle.

Відсутність деяких продвинутих можливостей: MySQL не має всіх функцій, які присутні в деяких комерційних СУБД. Наприклад, він може не мати вбудованої підтримки аналітичних функцій або географічних даних.

Перформанс: Хоча MySQL є дуже швидким і ефективним, в деяких ситуаціях він може не досягати такої високої продуктивності, як деякі інші СУБД. Залежно від конкретного використання, можуть виникати проблеми зі швидкодією або обробкою великого обсягу даних.

Обмежена масштабованість: Хоча MySQL може бути масштабованим, в нього є певні обмеження щодо масштабування горизонтально. Це може вплинути на його використання в дуже великих проектах зі значним обсягом даних та вимогами до продуктивності.

Відкритий код: Хоча відкритий код є перевагою для багатьох, деякі організації можуть вважати це недоліком, оскільки можуть виникати проблеми щодо підтримки та відповідальності за використання СУБД.

У підсумку, MySQL є популярною СУБД з багатьма перевагами, такими як надійність, простота використання та масштабованість. Однак, в порівнянні з деякими іншими СУБД, він може бути обмеженим у підтримці деяких функцій та не досягати такої високої продуктивності в деяких сценаріях. При виборі СУБД варто враховувати конкретні вимоги проекту та аналізувати переваги та недоліки кожної системи.

На (рис. 2.1) зображена діаграма зв'язків, яка наглядно відображає взаємозв'язки між різними елементами системи або компонентами програмного забезпечення. Ця діаграма графічно демонструє структуру та взаємодії між об'єктами, допомагаючи зрозуміти, як вони взаємодіють один з одним.

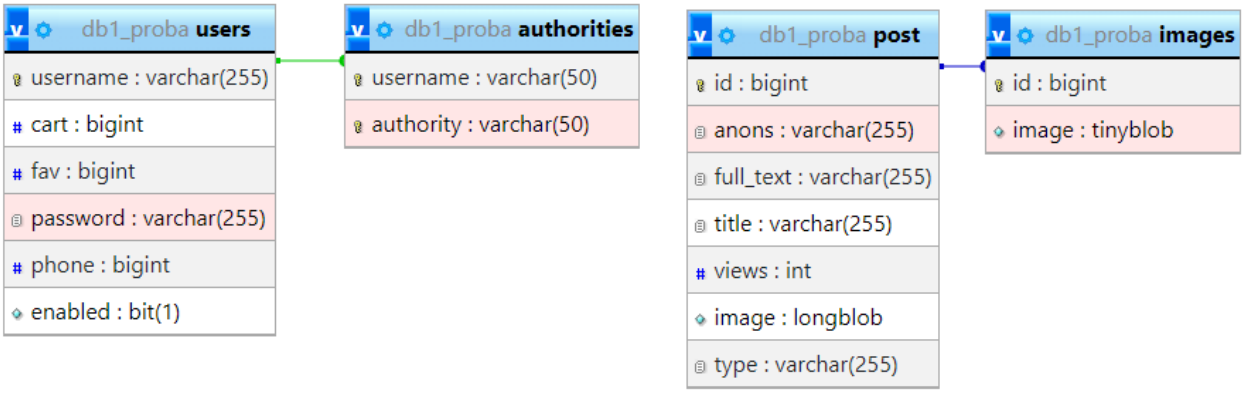

Рисунок 2.1 – Діаграма зв'язків таблиць

Для показу залежностей і зв'язків між об'єктами на діаграмі зв'язків використовуються графічні елементи, такі як стрілки та лінії. Це дозволяє відобразити різні типи зв'язків, такі як асоціації, агрегації, композиції, спадковість та інші.

Діаграма зв'язків допомагає розкрити структуру системи, виділити ключові компоненти та їх взаємодії. Це особливо корисно при проектуванні нової системи або аналізі існуючої, оскільки діаграма допомагає розібратися у складних взаємозв'язках і забезпечує зрозумілість та належне управління системою.

Діаграми зв'язків є важливим інструментом в області моделювання програмного забезпечення та системного аналізу. Вони дозволяють розробникам, аналітикам та зацікавленим сторонам отримати загальне уявлення про структуру системи та взаємозв'язки між її компонентами.

Використання діаграм зв'язків може бути різноманітним, від проектування баз даних та моделювання об'єктно-орієнтованих систем до проектування веб-додатків та іншого програмного забезпечення. Ці діаграми є ефективним засобом комунікації між різними членами команди розробки та сприяють зрозумілості та узгодженості під час роботи над проектом.

У практичному застосуванні розробки програмного забезпечення діаграми зв'язків є важливою частиною процесу аналізу та проектування системи. Вони допомагають зрозуміти потреби та вимоги користувачів, виявити ключові

елементи системи та їх взаємозв'язки, а також забезпечують зрозумілість та належне управління проектом.

У Spring Boot анотація @Entity використовується для позначення класів, які представляють сутності бази даних. Ця анотація вказує, що клас є персистентним і може бути збережений в базі даних. При використанні анотації @Entity, клас набуває можливості зберігати та отримувати об'єкти з бази даних.

Основні поняття, пов'язані з анотацією @Entity, такі:

Таблиця бази даних в мові програмування Java використовується для зберігання та управління даними. У рамках розробки додатків на основі фреймворку Spring Boot, клас, що позначений анотацією @Entity, відповідає таблиці бази даних. Кожен об'єкт цього класу представляє рядок або запис в таблиці;

Для відображення полів класу в колонки таблиці бази даних використовується анотація @Column. Наприклад, якщо у вас є клас User з полями id, name та email, то ці поля відображаються як колонки в таблиці бази даних. Зазвичай в кожній таблиці бази даних є одне або декілька полів, які унікально ідентифікують кожен рядок. Це називається ключовим полем. За замовчуванням, якщо у класу, позначеного анотацією @Entity, є поле з назвою "id", то воно вважається ключовим полем. Ви також можете вказати інше поле як ключове, використовуючи анотацію @Id;

Додатково, ви можете встановлювати зв'язки між різними сутностями бази даних, використовуючи анотації, такі як @OneToOne, @OneToMany, @ManyToOne та @ManyToMany. Ці анотації дозволяють визначати відношення між об'єктами з різних сутностей. Наприклад, анотація @OneToOne вказує на наявність одного-до-одного відношення між двома сутностями;

При запуску додатка Spring Boot автоматично створює таблиці бази даних, використовуючи інформацію з анотацій @Entity. Цей процес називається генерацією таблиць. Ви можете використовувати різні режими генерації таблиць, наприклад, автоматично оновлювати схему бази даних або створювати її заново

при кожному запуску;

Після позначення класу анотацією @Entity і налаштування бази даних, ви можете виконувати операції створення, читання, оновлення та видалення (CRUD) з об'єктами цього класу. Spring Boot автоматично генерує SQL-запити для цих операцій на основі анотацій @Entity. Наприклад, для збереження нового об'єкта в базі даних використовується метод save() репозиторію, а для отримання об'єкта за його ідентифікатором використовується метод findById();

Використання анотації @Entity у спрощеному режимі розробки Spring Boot дозволяє використовувати об'єктно-реляційний відображувач (ORM) для зручної роботи з базою даних. Вона дозволяє працювати з об'єктами, замість безпосередньої роботи з SQL-запитами, що полегшує розробку додатків;

Узагальнюючи, робота з таблицями бази даних у Spring Boot виконується шляхом використання анотацій @Entity для відображення класів у таблиці, @Column для відображення полів у колонки, @Id для визначення ключового поля, а також зв'язків між сутностями за допомогою анотацій @OneToOne, @OneToMany, @ManyToOne та @ManyToMany. Spring Boot автоматично генерує таблиці бази даних та дозволяє виконувати CRUD операції з об'єктами цих таблиць. Використання анотації @Entity спрощує розробку та підтримку додатків, дозволяючи працювати з об'єктами замість безпосередньої роботи з SQL-запитами.

Висновок до розділу 2

В розділі 2 мого досліджування я зосередився на кількох важливих аспектах, що стосуються розробки веб-додатків. Конкретно, я розглянув User stories, Інструментарій Spring Boot, реєстрацію і автентифікацію Spring Security, а також структуру бази даних.

Почавши з User stories, я проаналізував цей методологічний підхід, який дозволяє описати функціональність веб-додатку через короткі історії

користувачів. Я розглянув процес складання User stories, їх структуру та призначення. Такий підхід допомагає команді розробників краще зрозуміти потреби користувачів і належним чином впровадити функціональність веб-додатку.

Далі, я розглянув Інструментарій Spring Boot, який є потужним фреймворком для розробки веб-додатків на мові Java. Я описав основні переваги використання Spring Boot, такі як автоматична конфігурація, спрощення розробки, підтримка вбудованого сервера та багато іншого. З'ясував, як Spring Boot сприяє покращенню продуктивності розробників і забезпечує ефективну структуру проекту.

Одним з важливих аспектів розробки веб-додатків є реєстрація і автентифікація користувачів. В цьому контексті, я детально розглянув Spring Security - потужний модуль Spring Framework, який дозволяє забезпечити безпеку веб-додатків. Я пояснив процес налаштування Spring Security для реєстрації користувачів, аутентифікації, авторизації та захисту ресурсів. Такий підхід допомагає забезпечити безпеку веб-додатку і захистити конфіденційні дані користувачів.

Нарешті, я зайнявся структурою бази даних у контексті веб-додатків. Я розглянув різні типи баз даних, такі як реляційні та нереляційні, та їх особливості. Подальше, я дослідив методи моделювання даних та зв'язки між сутностями бази даних. Описав приклади використання SQL та ORM (Object-Relational Mapping) для роботи з базою даних. В результаті, я надав загальний огляд структури бази даних і показав, як вона використовується для зберігання та отримання даних у веб-додатку.

Таким чином, у розділі 2 я детально розглянув User stories, Інструментарій Spring Boot, реєстрацію і автентифікацію Spring Security і структуру бази даних.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СТРУКТУРИ ТА ФУНКЦІОНАЛУ ВЕБ-АПЛІКАЦІЇ

3.1 Розробка інтерфейсу сайту

Розробка дизайну веб-сайту є важливим етапом в процесі створення сучасних і привабливих веб-проектів. Вона поєднує в собі естетичні аспекти, функціональність та користувацький досвід для створення привабливого та легкого у використанні інтерфейсу. Давайте розглянемо детальніше, що таке розробка дизайну сайту та які принципи та елементи вона включає.

Розробка дизайну сайту - це процес створення зовнішнього вигляду веб-сайту, його інтерфейсу та користувацького досвіду. Вона починається з аналізу вимог і цілей проекту, а також вивчення цільової аудиторії. Розробник дизайну співпрацює з командою проекту, включаючи дизайнерів, розробників та замовника, для визначення потреб та очікувань стосовно веб-сайту.

Один з головних елементів розробки дизайну - це створення макетів (wireframes) та прототипів веб-сайту. Макети допомагають візуалізувати структуру та розташування елементів на сторінці. Вони можуть бути створені за допомогою спеціальних програм, таких як Adobe Photoshop, Sketch або Figma. Прототипи дозволяють інтерактивно тестувати функціональність та навігацію сайту перед його реалізацією.

Дизайн веб-сайту включає такі елементи, як кольорова палітра, шрифти, графіка, кнопки, іконки, фотографії тощо. Всі ці елементи об'єднуються в єдиний гармонійний стиль, відображаючи бренд та цінності компанії. Кольори та шрифти відіграють важливу роль у створенні настрою та сприйняття сайту користувачами. Графічні елементи, такі як логотипи та ілюстрації, допомагають визначити унікальний стиль і впізнаваність сайту.

Розробка дизайну також включає вирішення питань щодо користувацького досвіду (UX). Користувацький досвід визначає, наскільки легким та зручним є використання веб-сайту для його цільової аудиторії. Він залежить від таких

факторів, як навігація, швидкість завантаження, доступність інформації та зручність використання функцій. Розробник дизайну має забезпечити, щоб веб-сайт був зрозумілим та привабливим для користувачів.

Окрім вигляду та функціональності, розробка дизайну також враховує адаптивність веб-сайту під різні пристрої. З мобільними пристроями, такими як смартфони та планшети, все більше користувачів отримують доступ до веб-сайтів. Тому важливо, щоб дизайн сайту був адаптований під різні екрани та розміри пристроїв, забезпечуючи оптимальний користувацький досвід незалежно від типу пристрою.

Розробка дизайну сайту вимагає вмінь інтегрувати естетику, функціональність та користувацький досвід. Вона включає співпрацю з командою проекту, дослідження цільової аудиторії, створення макетів та прототипів, вибір кольорової палітри, шрифтів, графіки та інших елементів дизайну. Ключовою метою розробки дизайну є створення привабливого, зручного у використанні та ефективного веб-сайту, який відповідає цілям бізнесу та потребам користувачів.

Узагалішуючи, розробка дизайну сайту - це складний процес, який поєднує естетику, функціональність та користувацький досвід. Вона включає створення макетів, вибір кольорової палітри, шрифтів та графіки, а також забезпечує адаптивність під різні пристрої. Розробка дизайну веб-сайту вимагає вмінь та співпраці з командою проекту для створення привабливого та ефективного інтерфейсу, що задовольняє потреби користувачів і сприяє досягненню бізнес-цілей.

На рисунку 3.1 зображено зовнішній вигляд головної сторінки, яка є ключовим елементом сайту. Цей знімок екрана демонструє вигляд та структуру головної сторінки. Елементи інтерфейсу на цьому зображенні надають уявлення про те, яку функцію вони виконують.

Додатково, після зображення головної сторінки, наведений фрагмент коду, який відповідає за створення цієї сторінки. Цей фрагмент коду демонструє

технічну реалізацію та структуру, яка допомогла створити зображений вигляд домашньої сторінки.

Комбінація зображення вигляду сторінки та фрагменту коду надає повнішу картину про те, як була розроблена та реалізована домашня сторінка проекту. Вона дозволяє глибше зрозуміти як зовнішній вигляд, так і технічну структуру цієї важливої частини проекту.

Розглянувши зображення вигляду сторінки, ми можемо оцінити її дизайн, кольорову палітру, розташування елементів та загальний вигляд. Вона відображає те, як користувачі бачитимуть домашню сторінку та взаємодіятимуть з нею. Зображення допомагає зрозуміти, як розташовані різні елементи, такі як заголовки, кнопки, зображення та текст, і як вони співпрацюють один з одним для створення зручного та привабливого інтерфейсу.

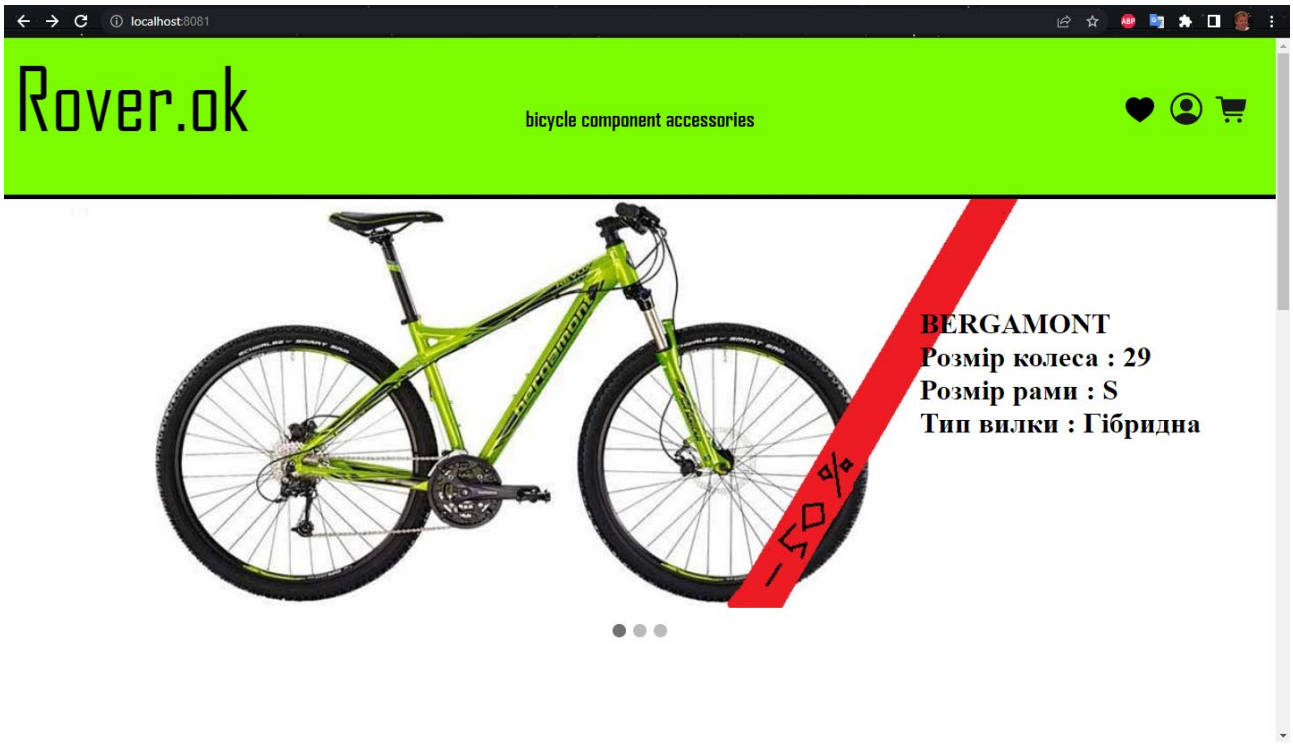

Рисунок 3.1 – Головна сторінка (header)

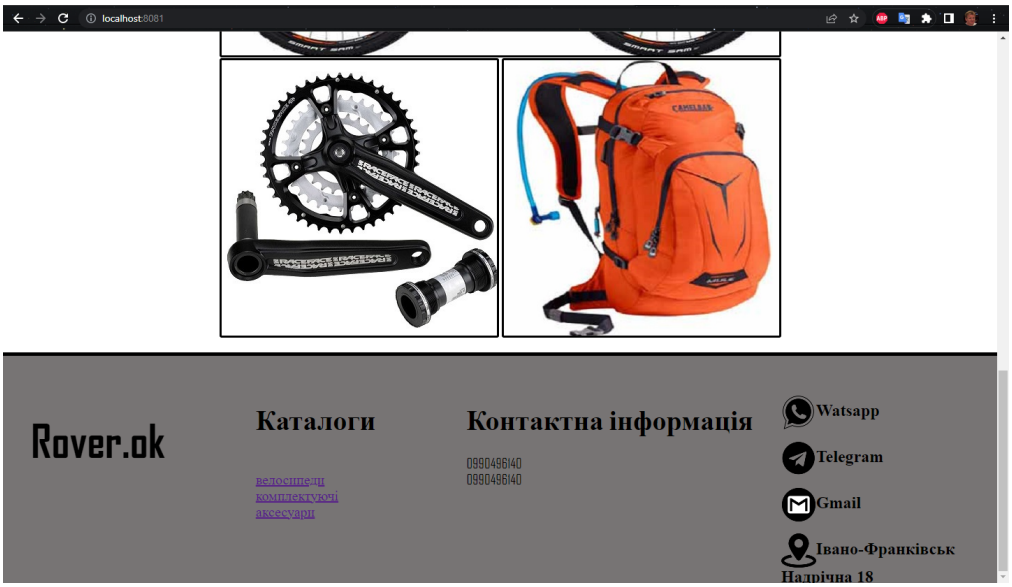

Рисунок 3.2 – Головна сторінка (footer)

Код головної сторінки:

```
<div class="slider-position">
  <div class="slideshow-container">
    <div class="mySlides fade">
      <div class="numbertext">1 / 3</div>
      <img th:src="@{/images/bik2.jpg}" style="width:100%">
      <div class="text">BERGAMONT<br>
        Розмір колеса : 29<br>
        Розмір рами : S<br>
        Тип вилки : Гібридна<br></div></div>
    <div class="mySlides fade">
      <div class="numbertext">2 / 3</div>
      <img th:src="@{/images/bik2.jpg}" style="width:100%">
      <div class="text">Caption Two</div></div>
    <div class="mySlides fade">
      <div class="numbertext">3 / 3</div>
      <img th:src="@{/images/bik2.jpg}" style="width:100%">
      <div class="text">Caption Three</div></div>
    <a class="prev" onclick="plusSlides(-1)">□</a>
    <a class="next" onclick="plusSlides(1)">□</a>
  </div><br>
  <div style="text-align:center">
    <span class="dot" onclick="currentSlide(1)"></span>
    <span class="dot" onclick="currentSlide(2)"></span>
    <span class="dot" onclick="currentSlide(3)"></span></div><script>
```

```
    let slideIndex = 1;
    showSlides(slideIndex);
    function plusSlides(n) {
        showSlides(slideIndex += n);}
    function currentSlide(n) {
        showSlides(slideIndex = n); }
    function showSlides(n) {
        let i;
        let slides = document.getElementsByClassName("mySlides");
        let dots = document.getElementsByClassName("dot");
        if (n > slides.length) {slideIndex = 1}
        if (n < 1) {slideIndex = slides.length}
        for (i = 0; i < slides.length; i++) {
            slides[i].style.display = "none"; }
        for (i = 0; i < dots.length; i++) {
            dots[i].className = dots[i].className.replace(" active", "");}
        slides[slideIndex-1].style.display = "block";
        dots[slideIndex-1].className += " active"}
</script></div>
```

Цей код створює слайдер зображень, а також додає кнопки для перемикання між слайдами.

Основні елементи коду:

: Це контейнер, який утримує весь слайдер.

: Це контейнер, який містить всі слайди.

: Це окремий слайд, що з'являється під час прокручування.

: Відображає номер поточного слайда відносно загальної кількості слайдів.

<img th:src="@{/images/bik2.jpg}" style="width:100%">: Зображення, яке відображається у слайді.

: Текстовий опис або підпис до зображення у слайді.

□ і □: Кнопки для перемикання до попереднього або наступного слайда відповідно.

: Кнопки-індикатори, які дозволяють переходити безпосередньо до певного слайда.

Нижче наведено також JavaScript-код, який виконує логіку перемикання слайдів:

let slideIndex = 1;: Змінна, що відстежує поточний слайд.

showSlides(n): Функція, яка відображає потрібний слайд на основі значення slideIndex.plusSlides(n): Функція, яка зміщує slideIndex на n і викликає showSlides з оновленим значенням slideIndex.

currentSlide(n): Функція, яка встановлює slideIndex на n і викликає showSlides з оновленим значенням slideIndex.

```
<h1 style="text-align: center">Товари</h1>
<div class="nav3">
  <div class="img_button1">
    <button class="velobutt1" onclick="window.location.href='/second/bicycle'">
      <img th:src="@{/images/bik3.jpg}" alt="Button Image">
      <span class="text">Деякий текст</span>
    </button></div>
  <div class="small_img">
    <div class="img_button">
      <button class="velobutt" onclick="window.location.href='/second/component'">
        <img th:src="@{/images/p6.jpg}" alt="Button Image">
        <span class="text">Деякий текст</span>
      </button></div>
    <div class="img_button">
      <button class="velobutt" onclick="window.location.href='/second/accessories'" >
        <img th:src="@{/images/a4.jpg}" alt="Button Image">
        <span class="text">Деякий текст</span>
      </button></div></div></div>
```

Рядок <h1 style="text-align: center">Товари</h1> встановлює стиль заголовка <h1>, де текст "Товари" буде вирівняний по центру сторінки.

Рядок визначає контейнер з класом "nav3", який обгортає наступні елементи.

Рядок визначає контейнер з класом "img_button1", який обгортає кнопку із зображенням.

Рядок<buttonclass="velobutt1"onclick="window.location.href='/second/bicycle'"> визначає кнопку з класом "velobutt1", при натисканні на яку виконується перенаправлення на сторінку "/second/bicycle".

Рядок <img th:src="@{/images/bik3.jpg}" alt="Button Image"> встановлює зображення з шляхом "/images/bik3.jpg" і альтернативним текстом "Button Image".

Рядок Деякий текст визначає спан-елемент з класом "text", який містить текст "Деякий текст".

3.2 Огляд html форм

Елемент header **:**

```
<!-- header start--> <div class="header"> <div class="lable"> <a
href="/">Rover.ok</a> </div> <div class="nav1"> <b><a
href="/second/bicycle">bicycle</a> <a href="/second/component">component</a> <a
href="/second/accessories">accessories</a></b> </div> <div class="nav2"> <button
class="butt" onclick="location.reload()"><img th:src="@{/images/heart.png}"></button>
<div> <div sec:authorize="isAuthenticated()"><button class="butt"
onclick="window.location.href='/userInfo'"><img
th:src="@{/images/user.png}"></button></div> <div
sec:authorize="!isAuthenticated()"><button class="butt"
onclick="window.location.href='/login'"><img
th:src="@{/images/user.png}"></button></div> </div> <button class="butt"
onclick="location.reload()"><img th:src="@{/images/cart.png}"></button> </div> </div>
<!--header end-->
```

Наведений код представляє собою HTML-розмітку, яка створює заголовок (header) для веб-сторінки. Давайте розглянемо його детальніше.

Код починається з коментаря <!-- header start-->, який вказує на початок блоку заголовка. Після цього ми маємо , що створює контейнер для заголовка.

Усередині контейнера є два додаткових контейнера і , а також .

В ми маємо посилання Rover.ok, яке представляє логотип або назву сайту "Rover.ok". При кліці на це посилання буде здійснено перехід на головну сторінку.

У ми маємо навігаційне меню, яке складається з трьох елементів <a>, які мають посилання на "/second/bicycle", "/second/component" і "/second/accessories". Ці посилання, ймовірно, вказують на різні розділи або сторінки сайту.

У ми маємо додаткові елементи управління, які розташовані праворуч. Перший елемент - це кнопка <button> з класом "butt", яка має обробник події onclick="location.reload()". При кліці на цю кнопку сторінка буде перезавантажена. Усередині кнопки є <img>, який має атрибут th:src="@{/images/heart.png}" і відповідає за відображення зображення серця.

Після цього ми маємо два блоки з атрибутами sec:authorize, які перевіряють автентифікацію користувача. Перший блок відображатиметься, якщо користувач аутентифікований, і містить кнопку з обробником події onclick="window.location.href='/userInfo'". При кліці на цю кнопку відбудеться перехід на сторінку "/userInfo". Усередині кнопки є <img>, який має атрибут th:src="@{/images/user.png}" і відповідає за відображення зображення користувача.

Другий блок відображатиметься, якщо користувач не аутентифікований, і містить кнопку з обробником події onclick="window.location.href='/login'". При кліці на цю кнопку відбудеться перехід на сторінку "/login". Усередині кнопки також є <img>, який відповідає за відображення зображення користувача.

Нарешті, ми маємо третю кнопку <button class="butt" onclick="location.reload()">, яка також має обробник події onclick="location.reload()". При кліці на цю кнопку сторінка буде

перезавантажена. Усередині кнопки є <img>, який має атрибут th:src="@{/images/cart.png}" і відповідає за відображення зображення кошика.

Врешті-решт, код закінчується коментарем <!-- header end-->, що вказує на кінець блоку заголовка.

Цей код створює зручний заголовок для веб-сторінки, який містить логотип, навігаційне меню, елементи управління та зображення. Він також має функціонал перезавантаження сторінки та переходу на інші сторінки в залежності від автентифікації користувача.

Елемент footer:

```
<footer class="footer">
```

Початок елемента `footer` з класом "footer".

```
<divclass="footer_text1">
<h1>Rover.ok</h1></div>
```

Внутрішній контейнер `<div>` з класом "footer_text1", який містить заголовок першого рівня `<h1>` з текстом "Rover.ok".

```
<div class="footer_text2">
<h1>Каталоги</h1> <br>
```

: Внутрішній контейнер `<div>` з класом "footer_text2", який містить заголовок першого рівня `<h1>` з текстом "Каталоги" і розрив рядка `<br>`.

```
<a class="a" href="/second"> велосипеди
<br>комплектуючі
 <br>аксесуари </a>
```

Елемент `<a>` з класом "a" та атрибутом `href` вказує на "/second". Внутрішній текст елемента `<a>` містить рядки "велосипеди", "комплектуючі" та "аксесуари", розділені розривами рядка `<br>`.

```
</div>
```

Закриваючий тег `</div>` для контейнера з класом "footer_text2".

```
 <div class="footer_text3">
<h1> Контактна інформація</h1>
 0990496140<br>
0990496140 </div>
```

Внутрішній контейнер `<div>` з класом "footer_text3", який містить заголовок першого рівня `<h1>` з текстом "Контактна інформація". Після

заголовка є два рядки з номерами телефону "0990496140", розділені розривами рядка`<br>`.

```
<divclass="footer_text4">  <h1><imgth:src="@{/images/b1.png}">Watsapp</h1>
<h1><imgth:src="@{/images/b2.png}">Telegram</h1>
<h1><imgth:src="@{/images/b3.png}">Gmail</h1>
<h1><imgth:src="@{/images/b4.png}">Івано-Франківськ <br> Надрічна 18</h1> </div>
```

Опис рядка: Внутрішній контейнер `<div>` з класом "footer_text4", який містить чотири заголовки першого рівня `<h1>`. Кожен заголовок має зображення `<img>` з вказаним шляхом до зображення (використовується Thymeleaf для відображення шляху) та текстовий контент: "Watsapp", "Telegram", "Gmail", "Івано-Франківськ" та "Надрічна 18". Рядок "Надрічна 18" розташований під заголовком і розділений розривом рядка `<br>`. </footer> Опис рядка: Кінець елемента `footer`.

Форма Second:

```
<div class="tovaru">
  <div class="lot" th:each="el : ${posts}">
    <a th:href="@{/lot/{im}(im=${el.id})}">
      <div class="lot_content">
        <div class="afaf">
      <img th:src="@{/images/{im}.jpg (im=${el.id})}" >
        </div>
        <div class="afa">
      <h3 th:text="${el.title}" class="el"></h3><br>
      <h3 th:text="${el.id}" class="el"></h3>
        </div>
      </div>
    </a>
  </div>
</div>
<div class="tovaru">
```

У даному рядку ми маємо контейнер `<div>` з класом "tovaru". Цей контейнер ізолює групу товарів або елементів.

```
<div class="lot" th:each="el : ${posts}">
```

У цьому рядку ми також маємо контейнер `<div>` з класом "lot". До цього контейнера застосовується директива `th:each`, яка слугує для ітерації по

кожному елементу у змінній `${posts}`. Це означає, що код всередині контейнера `<div class="lot">` буде виконуватись для кожного елементу зі списку .

`${posts}`. <a th:href="@{/lot/{im}(im=${el.id})}">

У цьому рядку ми маємо посилання `<a>` з атрибутом `th:href`, який містить шаблон URL-адреси `/lot/{im}`. Вираз `(im=${el.id})` використовує значення `el.id` для підстановки в шаблон URL-адреси. Це означає, що кожне посилання буде містити унікальний ідентифікатор `el.id`.

У цьому рядку ми маємо контейнер `<div>` з класом "lot_content". Цей контейнер представляє собою блок змісту товару або елементу.

У цьому рядку ми маємо контейнер `<div>` з класом "afaf". Цей контейнер, ймовірно, використовується для стилізації або розміщення зображення товару.

<img th:src="@{/images/{im}.jpg (im=${el.id})}">

У цьому рядку ми маємо тег `<img>` з атрибутом `th:src`, який вказує шлях до зображення товару. Шлях буде формуватись за допомогою шаблону `/images/{im}.jpg`, де значення `im` використовується для підстановки значення `el.id`.

У цьому рядку ми маємо контейнер `<div>` з класом "afa". Цей контейнер, ймовірно, використовується для розміщення текстової інформації про товар. <h3 th:text="${el.title}" class="el"></h3>

У цьому рядку ми маємо заголовок `<h3>`, який має текстовий контент зі значенням `${el.title}`. Значення `el.title` буде підставлено в місці вказаного атрибута `th:text`. Заголовок також має клас "el".

<h3 th:text="${el.id}" class="el"></h3>

У цьому рядку ми також маємо заголовок `<h3>`, який має текстовий контент зі значенням `${el.id}`. Значення `el.id` буде підставлено в місці вказаного атрибута `th:text`. Заголовок також має клас "el". Закриваючий тег `</div>` для контейнера `<div class="afa">`.

Цей код використовується для генерації динамічного HTML-коду, який має містити інформацію про товари або елементи зі списку ${posts}. Кожен елемент списку ${posts} буде відображатись у вигляді окремого блоку з зображенням, назвою та ідентифікатором. Класи і атрибути використовуються для стилізації та надання певної функціональності цим елементам. Посилання також додається для кожного елементу, щоб перенаправити користувача на окрему сторінку, яка відповідає вибраному елементу.

Форма Lot:

```
<div class="content">
 <div class="in_content1">
 <img th:src="@{/images/{it}.jpg (it=${id_lot})}" />
 </div> <div class="in_content2">
 <h3 th:text="${post.full_text}" class="el"></h3>
<h3 th:text="${id_lot}"></h3> </div> </div>

<div class="content">
```

Цей код представляє собою HTML-розмітку, яка створює контейнер з класом "content". Усередині цього контейнера є два додаткові контейнери: "in_content1" та "in_content2".

```
<div class="in_content1">
```

Цей контейнер "in_content1" містить зображення, яке завантажується з використанням атрибуту `th:src`. Зображення має шлях `/images/{it}.jpg`, де `{it}` підставляється значенням змінної `id_lot`, отриманої змінною `it`.

```
<img th:src="@{/images/{it}.jpg (it=${id_lot})}" />
```

Відповідає за вставку зображення в контейнер

```
</div> <div class="in_content2">
```

Цей контейнер "in_content2" містить два заголовки `<h3>`. Перший заголовок має текст, який витягується змінною `post.full_text` і відображається за допомогою атрибуту `th:text`. Другий заголовок має текст, отриманий змінною `id_lot`.

<h3th:text="${post.full_text}"class="el"></h3><h3 th:text="${id_lot}"></h3>
Рядок відповідає за вставку тексту post.full_text у тег заголовку <h3> з класом el , ат акож за вставку значення id_lot у тег заголовку <h3>.

Отже, код створює контейнер з двома додатковими контейнерами всередині. Перший контейнер містить зображення, яке залежить від значення змінної id_lot. Другий контейнер містить два заголовки, які відображають текст змінних post.full_text та id_lot.

Форма regist:

Перевіряє, чи існує параметр "error" в запиті. Якщо параметр "error" існує, то цей рядок відображає блок з текстом "Invalid username and password.", який повідомляє про невірне ім'я користувача або пароль.

<h1>Registration Form</h1>

Відображає заголовок "Registration Form", який слугує назвою форми реєстрації.

<form th:action="@{/regist}" method="post" th:object="${user}">

Визначає форму реєстрації. Він встановлює атрибути action для вказання URL-адреси, до якої буде відправлено дані форми, method для вказання методу передачі даних (у цьому випадку POST), і th:object для вказання об'єкта "user", з якого братимуться дані для заповнення полів форми.

<label> User Name : <input type="text" id="user_name" name="username" th:field="*{username}"/> </label> , <label> Password: <input type="password" id="password" name="password" th:field="*{password}"/> </label> , <label> Phone: <input type="text" id="phone" name="phone" th:field="*{phone}"/> </label>

Визначають рядки форми для введення користувача, пароля та номеру телефону. Вони містять відповідні елементи <input>, які дозволяють користувачеві вводити значення. Атрибут name вказує на ім'я поля, а th:field зв'язує поле з відповідним полем об'єкта "user".

<input type="submit" value="Sign In"/>

Визначає кнопку "Sign In" для надсилання форми.

Форма login:

```
<div th:if="${param.error}">
   Invalid username and password.
  </div>
   <h1>Registration Form</h1>
  <form th:action="@{/login}" method="post">
   <div><label> User Name : <input type="text" id="user_name" name="username"/>
</label></div>
   <div><label> Password: <input type="password" id="password" name="password"/>
</label></div>
   <div><input type="submit" value="Sign In"/></div>
  </form>
<a th:href="@{/regist}">Зареєструватись</a>
```

```
<div th:if="${param.error}">
   Invalid username and password.
</div>
```

Цей рядок перевіряє, чи існує параметр "error" у запиті. Якщо такий параметр існує, то він виводить повідомлення "Invalid username and password." у тегу `<div>`.

```
<h1>Registration Form</h1>
```

Цей рядок відображає заголовок "Registration Form" у розмітці сторінки.

```
<form th:action="@{/login}" method="post">
  <div><label>UserName:<inputtype="text"id="user_name"name="username"/>
</label></div>
<div><label>Password:<inputtype="password"id="password"name="password"/>
</label></div><div><input type="submit" value="Sign In"/></div></form>
```

Цей рядок відображає форму для входу. Властивість `th:action` вказує URL-адресу `/login`, до якої буде відправлено дані форми після натискання кнопки "Sign In". Форма включає два поля для введення: "User Name" (ім'я користувача) та "Password" (пароль), які вводяться у відповідні поля `<input>`. Кнопка "Sign In" виконує відправку форми.

3.3 Огляд контроллерів

MainController – Відповідає за головну сторінку.

@Controller

Цей код вказує, що клас є контролером веб-додатка і буде відповідати за обробку HTTP-запитів.

public class MainController {

Цей клас представляє собою контролер, який оброблятиме HTTP-запити та виконуватиме відповідні дії.

@Autowired

Цей анотація використовується для автоматичного впровадження (внедрення) залежностей. В даному випадку, поле "postRepository" буде автоматично ініціалізовано інстацією, яка реалізує інтерфейс "PostRepository". Це забезпечить доступ до функцій репозиторію, таких як збереження, отримання або видалення даних.

@GetMapping("/")

Цей анотований метод вказує, що він буде обробляти HTTP GET-запити, спрямовані на шлях "/". Це називається методом-обробником запиту.

public String greeting(@RequestParam(name="name", required=false, defaultValue="Word") String name, Model model) {

Цей метод приймає параметри запиту, такі як "name", і модель "Model". Параметр "name" може бути відсутнім, оскільки він має значення за замовчуванням "Word". Модель використовується для передачі даних у представлення.

model.addAttribute("name", name);

Цей рядок додає атрибут "name" до моделі, щоб передати його в представлення. Значення атрибуту встановлюється на основі параметра "name", переданого у запиті.

Iterable<Post> posts = postRepository.findAll();

Цей рядок отримує всі пости (записи) з репозиторію, використовуючи функцію "findAll()". Результат зберігається у змінній "posts", яка є ітерованою колекцією.

model.addAttribute("posts", posts);

Цей рядок додає атрибут "posts" до моделі, щоб передати його в представлення. Значення атрибуту встановлюється на основі змінної "posts", що містить всі пости.

return "greeting";

Цей оператор повертає ім'я представлення "greeting", яке буде відображати сторінку вітання.

SecondController – Відповідає за наповнення сторінки з вибраним типом товарів.

public class SecondController {

Цей клас представляє собою контролер, який оброблятиме HTTP-запити та виконуватиме відповідні дії.

@Autowired

Ця анотація використовується для автоматичного впровадження (внедрення) залежностей. В даному випадку, поле "postRepository" буде автоматично ініціалізовано інстацією, яка реалізує інтерфейс "PostRepository". Це забезпечить доступ до функцій репозиторію, таких як збереження, отримання або видалення даних.

@GetMapping("/second/{type}")

Ця анотована метода вказує, що вона буде обробляти HTTP GET-запити, спрямовані на шлях "/second/{type}". "{type}" є шляховою змінною, яка буде мати значення, передане у запиті.

public String Second (@PathVariable("type") String type, Model model) {

Цей метод приймає шляхову змінну "type" з URL та модель "Model". Значення шляхової змінної привласнюється параметру "type". Модель використовується для передачі даних у представлення.

Iterable<Post> posts = postRepository.getByType(type);

Цей рядок отримує всі пости (записи) з репозиторію, використовуючи функцію "getByType()", яка отримує тип посту. Результат зберігається у змінній "posts", яка є ітерованою колекцією.

model.addAttribute("posts", posts);

Цей рядок додає атрибут "posts" до моделі, щоб передати його в представлення. Значення атрибуту встановлюється на основі змінної "posts", що містить всі пости.

return "second";

Цей оператор повертає ім'я представлення "second", яке буде відображати сторінку "second".

LotController – Відповідає за відображення товару та інформації про нього на окремій сторінці.

```
@Controller
public class LotController {
   @Autowired
   private PostRepository postRepository;
   @GetMapping("/lot/{id}")
   public String Lot (@PathVariable("id") Long id, Model model){
      Optional<Post> post= postRepository.findById(id);
      if (post.isPresent()) {
         model.addAttribute("post", post.get());
         model.addAttribute("id_lot", id);
      }

      return "lot";
   }
}
```

```
public class LotController {
```

Цей клас представляє собою контролер, який оброблятиме HTTP-запити та виконуватиме відповідні дії.

@Autowired

Ця анотація використовується для автоматичного впровадження (внедрення) залежностей. В даному випадку, поле "postRepository" буде автоматично ініціалізовано інстацією, яка реалізує інтерфейс "PostRepository". Це забезпечить доступ до функцій репозиторію, таких як збереження, отримання або видалення даних.

@GetMapping("/lot/{id}")

Ця анотована метода вказує, що вона буде обробляти HTTP GET-запити, спрямовані на шлях "/lot/{id}". "{id}" є шляховою змінною, яка буде мати значення, передане у запиті.

public String Lot (@PathVariable("id") Long id, Model model) {

Цей метод приймає шляхову змінну "id" з URL та модель "Model". Значення шляхової змінної привласнюється параметру "id". Модель використовується для передачі даних у представлення.

Optional<Post> post = postRepository.findById(id);

Цей рядок шукає пост (запис) з репозиторію за його ідентифікатором, використовуючи функцію "findById()". Результат зберігається у змінній "post", яка є обгорткою (wrapper) для можливого значення.

if (post.isPresent()) {

Ця умова перевіряє, чи присутнє значення в змінній "post".

model.addAttribute("post", post.get());

Цей рядок додає атрибут "post" до моделі, щоб передати його в представлення. Значення атрибуту отримується зі змінної "post", використовуючи метод "get()" для отримання значення обгортки "Optional".

model.addAttribute("id_lot", id);

Цей рядок додає атрибут "id_lot" до моделі, щоб передати його в представлення. Значення атрибуту встановлюється на основі змінної "id".

return "lot";

Цей оператор повертає ім'я представлення "lot", яке буде відображати сторінку "lot" на якій будуть обрані товари.

LoginController – Відповідає за реєстрацію та аутентифікацію користувачів.

public class LoginController {

Цей клас представляє собою контролер, який оброблятиме HTTP-запити та виконуватиме відповідні дії.

@Autowired

Ця анотація використовується для автоматичного впровадження (внедрення) залежностей. В даному випадку, поля "authorityRepository", "userRepository" і "passwordEncoder" будуть автоматично ініціалізовані

відповідними залежностями. Це забезпечить доступ до функцій репозиторіїв та коду для шифрування паролів.

```
@GetMapping("/login")
```

Ця анотована метода вказує, що вона буде обробляти HTTP GET-запити, спрямовані на шлях "/login".

```
public String login(Model model) {
```

Цей метод приймає модель "Model" для передачі даних у представлення.

```
Iterable<User> users = userRepository.findAll();
```

Цей рядок отримує всіх користувачів з репозиторію, використовуючи функцію "findAll()". Результат зберігається у змінній "users".

```
model.addAttribute("users", users);
```

Цей рядок додає атрибут "users" до моделі.

```
return "login";
```

Цей оператор повертає ім'я представлення "login", яке буде відображати сторінку для входу.

```
@PostMapping("/login")
```

Ця анотована метода вказує, що вона буде обробляти HTTP POST-запити, спрямовані на шлях "/login".

```
public String postlogin(@ModelAttribute("user") User user, BindingResult result) {
```

Цей метод приймає об'єкт "User" (отриманий з форми входу) та об'єкт "BindingResult", який містить результати валідації форми.

```
if (result.hasErrors()) {
```

Ця умова перевіряє, чи є помилки в результаті валідації форми.

```
return "registration";
```

Якщо є помилки, метод повертає ім'я представлення "registration", яке буде відображати сторінку реєстрації.

```
return "/greeting";
```

Якщо немає помилок, метод повертає ім'я представлення "/greeting", яке буде відображати сторінку привітання.

```
@GetMapping("/regist")
```

Ця анотована метода вказує, що вона буде обробляти HTTP GET-запити, спрямовані на шлях "/regist".

public String regist(Model model) {

Цей метод приймає модель "Model" для передачі даних у представлення.

UserDto user = new UserDto();

Цей рядок створює новий об'єкт "UserDto".

model.addAttribute("user", user);

Цей рядок додає атрибут "user" до моделі.

return "regist";

Цей оператор повертає ім'я представлення "regist", яке буде відображати сторінку реєстрації.

@PostMapping("/regist")

Ця анотована метода вказує, що вона буде обробляти HTTP POST-запити, спрямовані на шлях "/regist".

public String postregist(@ModelAttribute("user") UserDto userDto, BindingResult result) {

Цей метод приймає об'єкт "UserDto" (отриманий з форми реєстрації) та об'єкт "BindingResult", який містить результати валідації форми.

if (result.hasErrors()) {

Ця умова перевіряє, чи є помилки в результаті валідації форми.

return "registration"; Якщо є помилки, метод повертає ім'я представлення "registration", яке буде відображати сторінку реєстрації.

User user = new User();

Цей рядок створює новий об'єкт "User".

user.setUsername(userDto.getUsername()); user.setPassword(passwordEncoder.encode(userDto.getPassword())); user.setPhone(userDto.getPhone()); user.setEnabled(true); userRepository.save(user);

Ці рядки встановлюють значення властивостей об'єкта "User" на основі даних з об'єкта "UserDto". Пароль шифрується перед збереженням.

Authorities authorities = new Authorities(); authorities.setUsername(authorities.getUsername()); authorities.setAuthority("ROLE_USER"); authorityRepository.save(authorities);

Ці рядки створюють об'єкт "Authorities" та встановлюють значення його властивостей. Об'єкт "Authorities" зберігає інформацію про ролі користувачів.

return "/greeting";

Якщо немає помилок, метод повертає ім'я представлення "/greeting", яке буде відображати сторінку привітання.

```
@GetMapping("/userInfo")
```

Ця анотована метода вказує, що вона буде обробляти HTTP GET-запити, спрямовані на шлях "/userInfo".

```
public String userInfo(Principal principal, Model model) {
```

Цей метод приймає об'єкт "Principal" та модель "Model" для отримання інформації про користувача та передачі її у представлення.

```
Authentication authentication = (Authentication) principal; org.springframework.security.core.userdetails.User user = (org.springframework.security.core.userdetails.User) authentication.getPrincipal(); final String username = user.getUsername();
```

Ці рядки отримують ім'я користувача з об'єкта "Principal".

```
User user1 = userRepository.findById(username).get();
```

Цей рядок отримує об'єкт користувача з репозиторію на основі його імені.

```
model.addAttribute("user", user1);
```

Цей рядок додає атрибут "user" до моделі, щоб передати його в представлення.

```
return "userInfo";
```

Цей оператор повертає ім'я представлення "userInfo", яке буде відображати сторінку з інформацією про користувача.

Висновки до розділу 3

У розділі 3дипломної роботи, я вирішив звернути увагу на розгляд дизайну головної сторінки, в якому я детально проаналізував його особливості, компоненти та вплив на користувачів. Цей розділ присвячений вивченню HTML-коду, що використовується на всіх сторінках сайту, а також структурі контролерів і задач, які вони виконують.

Почавши з дизайну головної сторінки, я розглянув різні аспекти, такі як розташування елементів, використання кольорів, шрифтів, графіки та інших

дизайнерських елементів. Я описав, як ці компоненти взаємодіють між собою, створюючи зручне та естетичне візуальне сприйняття для відвідувачів.

Наступною частиною розділу є аналіз HTML-коду, який використовується на всіх сторінках сайту. Я детально розглянув структуру і синтаксис HTML-тегів, їх використання для розмітки тексту, створення посилань, таблиць, форм та інших важливих елементів веб-сторінок. Також було вивчено принципи доступності та семантичної розмітки, які допомагають поліпшити взаємодію з сайтом для різних користувачів.

Після цього я детально розглянув структуру контролерів, які відповідають за обробку запитів користувачів. Вони виконують важливу роль у веб-додатку, забезпечуючи правильну маршрутизацію та передачу даних між фронтендом і бекендом. Описуючи функціональність кожного контролера, я звернув увагу на їх завдання і функції, які вони виконують у процесі обробки запитів.

Завершуючи розділ 3, я також проаналізував різноманітні задачі, які використовуються контролерами. Ці задачі можуть включати автентифікацію користувачів, валідацію даних, роботу з базою даних, відправку повідомлень і багато іншого. Я розглянув приклади коду та пояснив їх роль у веб-додатку.

Таким чином, у розділі 3 я зосередився на докладному аналізі дизайну головної сторінки, HTML-коду всіх сторінок та структурі контролерів і задач.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи створено веб-сайт, за допомогою якого можна купити велосипеди , аксесуари та деталі і отримати консультацію що до вибраних товарів . Одним з важливих компонентів сайту є можливість отримати онлайн консультацію. Користувачі мають можливість звернутися до кваліфікованих фахівців.

В процесі розробки було:

- проведено аналіз переваг та недоліків уже існуючих сайтів-аналогів;
- зроблено вибір технологій для реалізації веб-сайту, а саме (front-end:Html,CSS; back-end:Java(Spring Boot,Spring Security); database:MySQL);
- розроблено сучасний та простий для сприйняття дизайн веб-сайту;
- проведено тестування продукту;

розроблено заходи з охорони праці.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Oracle Java Tutorials: https://docs.oracle.com/javase/tutorial/ (дата звернення:07.04.2023)

2. W3Schools: https://www.w3schools.com/(дата звернення:07.04.2023)

3. Codecademy: https://www.codecademy.com/(дата звернення:07.04.2023)

4. Udemy: https://www.udemy.com/(дата звернення:07.04.2023)

5. Coursera: https://www.coursera.org/(дата звернення:08.04.2023)

6. Pluralsight: https://www.pluralsight.com/(дата звернення:08.04.2023)

7. Baeldung: https://www.baeldung.com/(дата звернення:08.04.2023)

8. JavaPoint: https://www.javatpoint.com/(дата звернення:08.04.2023)

9. Spring Framework Reference Documentation: https://docs.spring.io/spring-framework/docs/current/reference/html/(дата звернення:09.04.2023)

10. Spring Boot Documentation: https://docs.spring.io/spring-boot/docs/current/reference/html/(дата звернення:09.04.2023)

11. Spring Guides: https://spring.io/guides(дата звернення:09.04.2023)

12. HTML.com: https://html.com/(дата звернення:09.04.2023)

13. Mozilla Developer Network (MDN): https://developer.mozilla.org/(дата звернення:09.04.2023)

14. CSS-Tricks: https://css-tricks.com/(дата звернення:09.04.2023)

15. Learn HTML & CSS - Shay Howe: https://learn.shayhowe.com/(дата звернення:10.04.2023)

16. Java Code Geeks: https://www.javacodegeeks.com/(дата звернення:10.04.2023)

17. JavaWorld: https://www.javaworld.com/(дата звернення:10.04.2023)

18. DZone Java Zone: https://dzone.com/java-jdk-development-tutorials-tools-news(дата звернення:10.04.2023)

19. Java Ranch: https://coderanch.com/(дата звернення:11.04.2023)

20. The Java™ Tutorials: https://docs.oracle.com/javase/tutorial/(дата звернення:07.04.2023)