ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка Web-застосунку для продажів крафтового шоколаду мовами Java, JavaScript з використанням HTML та CSS»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Сергій САЗОНОВ
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

Сергій САЗОНОВ

Керівник: Владислав ЯСКЕВИЧ
к.т.н.

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Сазонову Сергію Олеговичу

1. Тема кваліфікаційної роботи: «Розробка Web-застосунку для продажів крафтового шоколаду мовами Java, JavaScript з використанням HTML та CSS» керівник кваліфікаційної роботи к.т.н., доцент кафедри ІПЗ Владислав ЯСКЕВИЧ, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи:
Науково-технічна література з питань, пов’язаних з розробкою програмного забезпечення, теоретичні відомості про методи ведення бізнесу продажів крафтового шоколаду, опис методів побудови архітектури web-застосунку, технічна документація з описом інструментів розробки web-застосунків.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд існуючих методів ведення бізнесу продажів крафтового шоколаду.
2. Аналіз та вибір інструментів для реалізації web-застосунку.
3. Проєктування архітектури web-застосунку для продажів крафтового шоколаду.

4. Реалізація та опис функціонування web-застосунку для продажів крафтового шоколаду.
5. Тестування web-застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до застосунку.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Схема клієнт-серверної архітектури web-застосунку.
6. Діаграма класів.
7. Діаграма діяльності.
8. Мапа web-застосунку.
9. Екранні форми.
10. Демонстрація роботи web-застосунку.
11. Апробація результатів дослідження.

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Огляд засобів розробки web-застосунків.	14.03-17.03.2024	
4	Проєктування web-застосунку для продажів крафтового шоколаду	18.03-24.03.2024	
5	Розробка web-застосунку	25.03-21.04.2024	
6	Тестування web-застосунку	22.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Сергій САЗОНОВ
(підпис)

Керівник
кваліфікаційної роботи _____ Владислав ЯСКЕВИЧ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 47 стор., 1 табл., 33 рис., 24 джерела.

Мета роботи – підтримка процесу продажів крафтового шоколаду шляхом створення web-застосунку мовами Java, JavaScript та HTML/CSS.

Об'єкт дослідження – процес продажу крафтового шоколаду.

Предмет дослідження – web-застосунок для продажів крафтового шоколаду.

Короткий зміст роботи: У ході роботи було проведено аналіз предметної галузі та визначено переваги web-застосунку, як спосіб ведення малого бізнесу продажів крафтового шоколаду. Проведено аналіз існуючих web-застосунків для продажів крафтового шоколаду. Проаналізовано існуючі моделі архітектури web-застосунків, засоби та методи розробки. Для розробки web-застосунку було обрано фреймворки Spring та Angular, базу даних PostgreSQL, середовища розробки IntelliJ IDEA та WebStorm та система контролю версій Git. Реалізовано трирівневу архітектуру сервера та створено користувацький інтерфейс. Розроблено клієнт-серверну архітектуру web-застосунку та реалізований ключовий функціонал, зокрема: додавання, зміна, видалення продукції, кошик продуктів, пошук продукту по назві, створення персонального кабінету користувача. Проведено тестування web-застосунку.

Сферою використання застосунку є продажі крафтового шоколаду.

КЛЮЧОВІ СЛОВА: КРАФТОВИЙ ШОКОЛАД, WEB-ЗАСТОСУНОК, КЛІЄНТ, СЕРВЕР.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

СУБД - Система управління базами даних.

IDE - Integrated Development Environment.

HTTP - Hypertext Transfer Protocol.

DTO - Data Transfer Object.

URI - Uniform Resource Identifier.

URL - Uniform Resource Locator.

API - Application Programming Interface.

JPA - Java Persistence API.

SQL - Structured Query Language.

JSON - JavaScript Object Notation.

JWT - JSON Web Token.

HTML - HyperText Markup Language.

CSS - Cascading Style Sheets.

REST - Representational State Transfer.

ВСТУП

Обґрунтування вибору теми та її актуальність: Малий бізнес є невід’ємною частиною сталої економіки, адже він є джерелом надходжень до державного бюджету та створює нові робочі місця, що вирішує не тільки економічні, а й соціальні питання та проблеми [1, 2]. Одним із видів ведення бізнесу є електронна комерція, а web-застосунок стає невід'ємною частиною ведення бізнесу в епоху електронної комерції. Він забезпечує зручність для клієнтів, даючи їм можливість обирати та оплачувати товари чи послуги не виходячи з дому. З точки зору власника бізнесу, web-застосунок економить кошти та автоматизує рутинні завдання, що робить його ключовим фактором для розвитку та прибутковості. Згідно з результатами опитування Індексу настроїв малого бізнесу за 2023 рік, яке провела Європейська Бізнес Асоціація в межах проєкту Unlimit Ukraine, цифровими каналами для продажів користується 67% опитаних підприємців, з яких 15% здійснюють продажі виключно онлайн, а 30% зазначають, що онлайн торгівля приносить їм набагато більше прибутку [3].

Об’єкт дослідження – процес продажу крафтового шоколаду.

Предмет дослідження – web-застосунок для продажів крафтового шоколаду.

Мета роботи – підтримка процесу продажів крафтового шоколаду за допомогою web-застосунку створеного мовами Java, JavaScript та HTML/CSS.

Методи дослідження – методи проєктування та створення трирівнева архітектури web-застосунку, методи проєктування та створення MVC архітектури сервера, методи проєктування та створення користувацького інтерфейсу, методи передачі, зберігання та обробки даних, методи тестування програмного забезпечення.

Для реалізації поставленої мети потрібно вирішити наступні завдання:

1. Проаналізувати вже існуючі аналоги, знайти їх переваги та недоліки.
2. Розробити функціональні та нефункціональні вимоги до web-застосунку, ґрунтуючись на проаналізованих аналогах.

3. Дослідити та вибрати засоби розробки web-застосунку, розробити модель архітектури web-застосунку та його web-застосунку.
4. Розробити web-застосунок на основі обраних інструментів та завчасно визначених вимог.
5. Провести тестування web-застосунку.

Робота пройшла апробацію на Всеукраїнській науково-технічній конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях» за темами «Розробка web-застосунку для продажів крафтового шоколаду» та «Web-застосунок як інструмент для ведення малого бізнесу», м. Київ, ДУІКТ, 24 квітня 2023 року [4, 5].

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Web-застосунок та його застосування для ведення бізнесу

Одним з найпопулярніших видів ведення малого бізнесу є електронна комерція - це реалізація товарів і послуг в інтернет-просторі. Однією зі складових електронної комерції є web-застосунок [4].

Web-застосунок - це програмне забезпечення, яке працює в web-браузері, подібно до web-сайту. На відміну від статичних web-сайтів, web-застосунки динамічні та інтерактивні, що дозволяє користувачам виконувати різні дії, такі як:

- Web-застосунок може мати каталог продуктів з фотографіями, описами та цінами, дозволяючи користувачам легко знаходити те, що їм потрібно.
- Користувачі можуть додавати товари до кошика, порівнювати різні продукти та змінювати замовлення перед оформленням покупки.
- Web-застосунок може пропонувати різні безпечні методи оплати, такі як кредитні картки, електронні гаманці та банківські перекази.
- Користувачі можуть відстежувати статус своїх замовлень та отримувати сповіщення про їхнє оновлення.
- Web-застосунок не має прив’язки до місця та часу роботу, тому клієнт завжди має змогу зробити замовлення в будь-де, у будь-який час.
- У web-застосунку клієнт може залишити свій відгук та читати відгуки інших клієнтів.
- У web-застосунку зберігається історія замовлень клієнта, тому клієнт завжди можете замовити товар, що сподобався.
- Web-застосунок може слугувати певною гарантією відповідальності компанії.

Важливість використання web-застосунку для бізнесу:

- Web-застосунок допомагає охопити ширшу аудиторію та збільшити обсяг продажів, надаючи клієнтам більш зручний спосіб покупки товарів та

послуг будь-де.

- Web-застосунок може допомогти покращити обслуговування клієнтів, надаючи їм цілодобовий доступ до інформації про продукцію, можливість відстежувати свої замовлення та спілкуватися з вами безпосередньо.
- Використовуючи web-застосунок зникає необхідність витрачати кошти на оренду приміщення, торгівельного обладнання, а кількість складу працівників може бути зменшена до мінімуму.
- Web-застосунок може збирати дані про клієнтів, такі як їхні вподобання та звички покупки, що можна використовувати для покращення своїх продуктів та послуг.
- Web-застосунок успішно зарекомендував себе як інструмент проведення маркетингових акцій.
- Глобальна мережа Інтернет охоплює увесь світ, тому маючи web-застосунок бізнес має змогу виходу на міжнародний ринок.

В цілому, web-застосунок є цінним інструментом для малого бізнесу, який може допомогти збільшити продажі, покращити обслуговування клієнтів, заощадити гроші та підвищити конкурентоспроможність.

1.2 Аналіз існуючих аналогів

1.2.1 WANDER

WANDER - це онлайн-магазин, що пропонує широкий вибір ексклюзивного шоколаду та солодощів ручної роботи. Вони пишаються використанням високоякісних інгредієнтів та пропонують унікальні смаки, які не знайти в звичайних магазинах [6].

WANDER пропонують широкий спектр шоколаду з різними смаками, від класичного темного і молочного, до більш екзотичних, таких як перець чилі, малиновий перець, лавандовий шоколад і трюфелі з оливковою олією та морською сіллю. Для своєї продукції вони використовують лише найкращі інгредієнти, такі

як: бельгійський шоколад, вершкове масло та натуральні ароматизатори. WANDER виготовляють шоколад маленькими партіями, щоб гарантувати свіжість та якість. Шоколад WANDER має красиву упаковку, що робить його гарним подарунком. Шоколад можна купити у подарункових коробках або створити свій власний набір подарунків. Замовити шоколад WANDER онлайн, вони пропонують доставку по всій Україні.

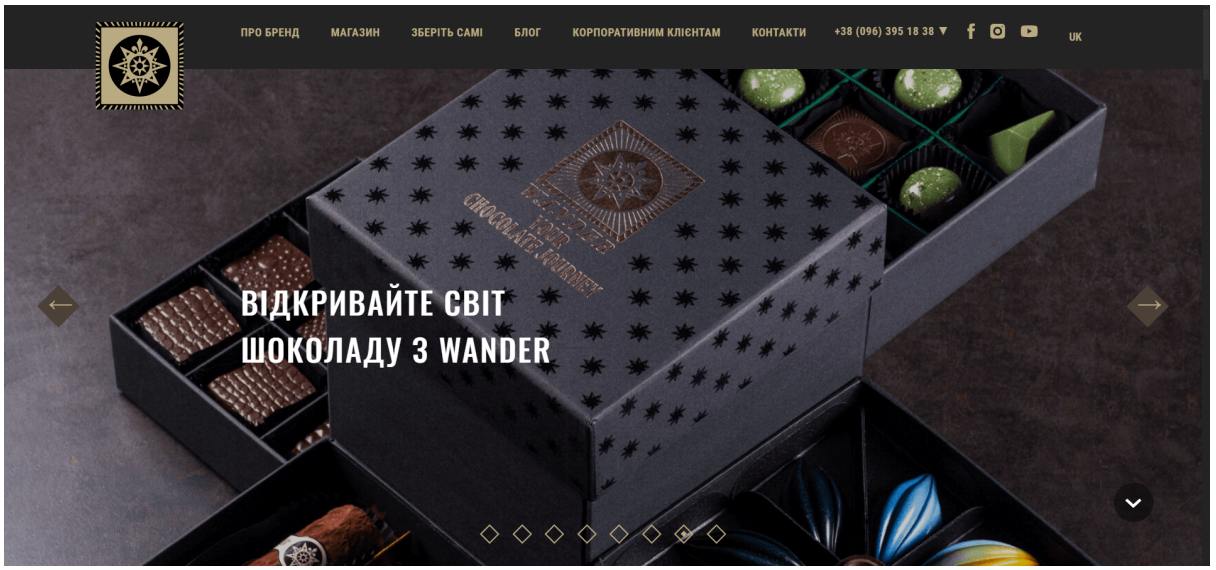


Рис. 1.1 Сайт магазину WANDER (головна строрінка)

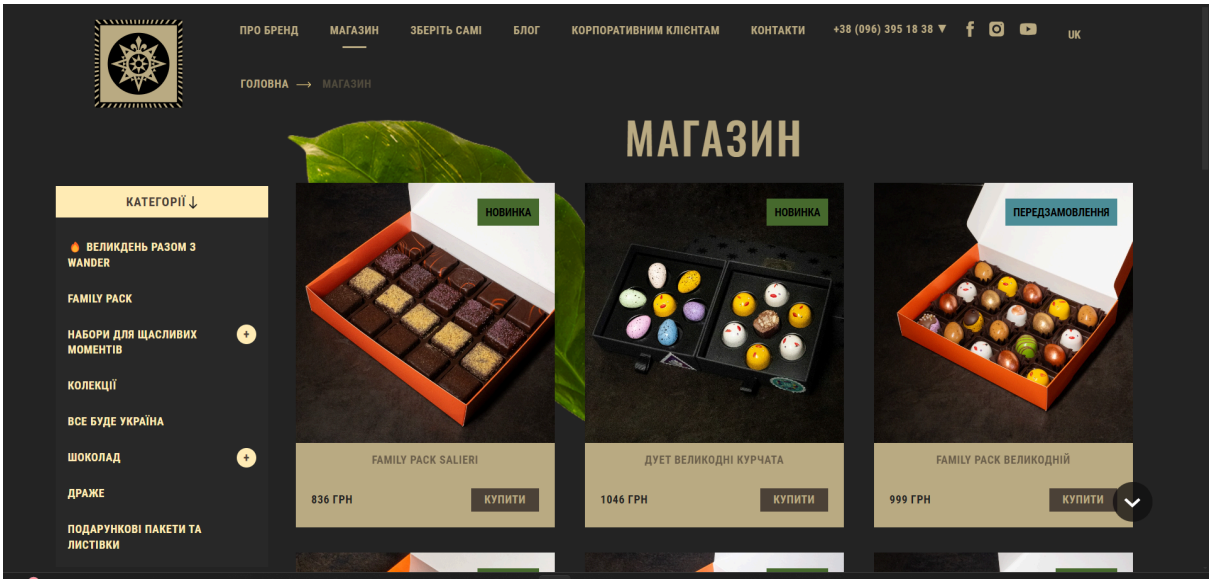


Рис. 1.2 Сайт магазину WANDER (сторінка товару)

У процесі аналізу були знайдені наступні переваги та недоліки:

Переваги:

- Привабливий та візуально приємний дизайн
- Детальна інформація в описі кожного продукту
- Дизайн сайту адаптується до розмірів екрану
- Зрозуміла навігація по сайту

Недоліки:

- Немає можливості створення персонального кабінету
- Немає функції пошуку товару
- Забагато простору екрану займають банери

1.2.2 Spell

Spell - це український виробник бельгійського шоколаду за авторськими рецептами [7]. Їхній шоколад виготовляється з натуральних інгредієнтів за унікальними рецептами. Spell - це чудовий вибір для любителів шоколаду, які шукають високоякісний, унікальний та красиво оформлений шоколад.

Spell пропонує широкий асортимент шоколаду, такі як:

- Класичні смаки: темний шоколад, молочний шоколад, білий шоколад
- Фруктові смаки: полуниця, малина, апельсин, лимон
- Горіхові смаки: фундук, мигдаль, волоський горіх
- Спеціальні смаки: лаванда, імбир, чилі
- Веганські шоколадні цукерки: виготовлені з темного шоколаду та натуральних підсолоджувачів
- Українська колекція: шоколадні цукерки з українськими символами та смаками.

Spell пропонує доставку по всій Україні, ціни на шоколад порівняно невисокі і відповідають високій якості їхніх продуктів. Spell мають багато позитивних відгуків від клієнтів, які хвалять високу якість шоколаду, унікальні смаки та красиве оформлення.

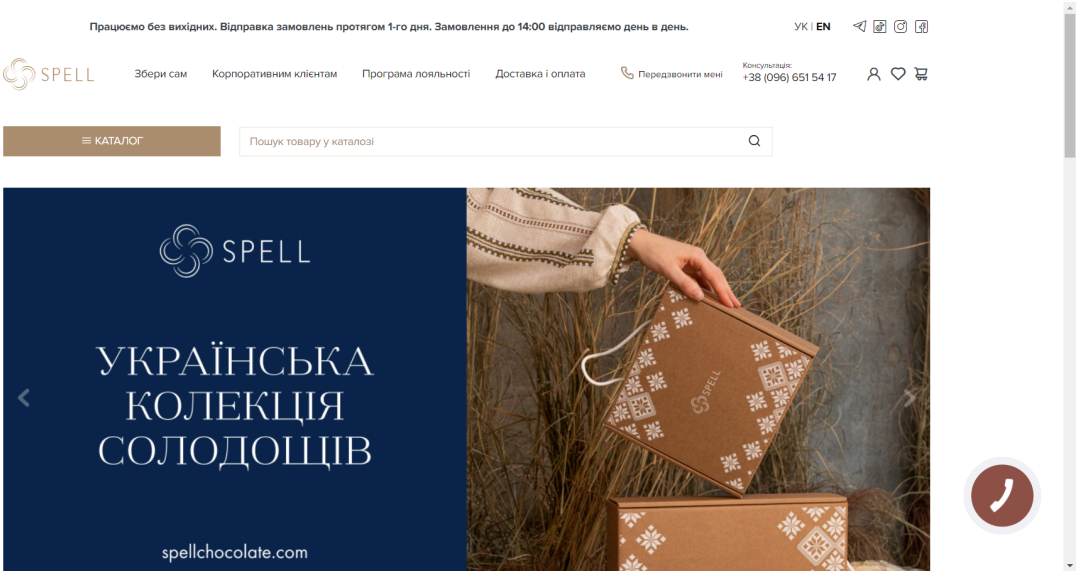


Рис. 1.3 Сайт магазину Spell (головна сторінка)

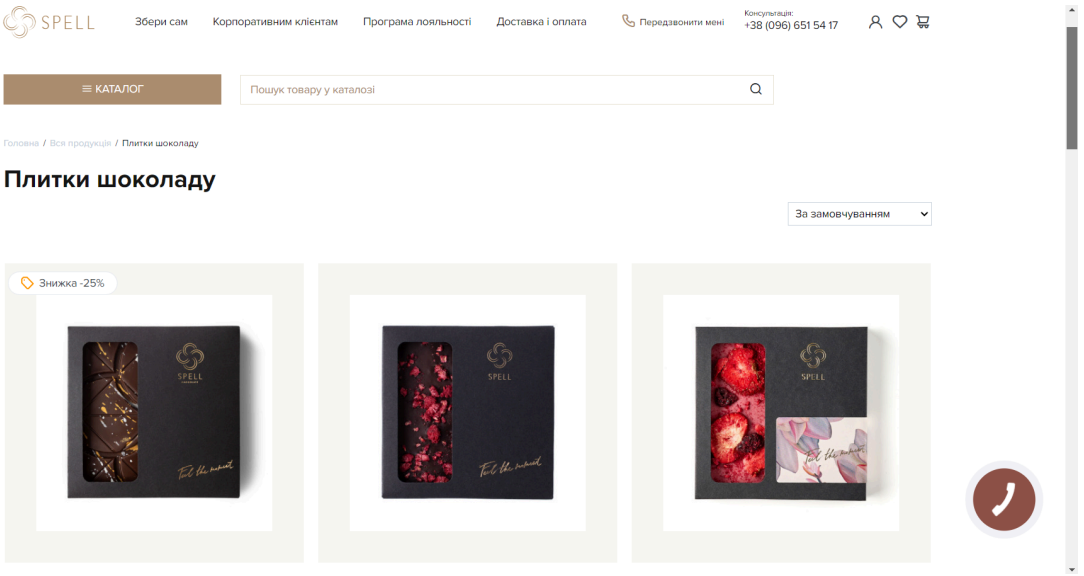


Рис. 1.4 Сайт магазину Spell (сторінка товару)

У процесі аналізу були знайдені наступні переваги та недоліки:

Переваги:

- Сучасний та мінімалістичний дизайн, в ньому багато білого простору, що робить сайт візуально приємним та не перевантаженим
- Зображення шоколаду на сайті високої якості та чіткості, що робить товар візуально привабливим.

- Навігація по сайту проста та зрозуміла.
- Є розділ з відгуками клієнтів
- На сайті є можливість пошуку товару за назвою
- Дизайн сайту адаптується до розмірів екрану

Недоліки:

- Довге завантаження сайту
- Для перегляду асортименту необхідно як мінімум перейти на ще одну сторінку

1.2.3 Львівська майстерня шоколаду

Львівська майстерня шоколаду - це мережа магазинів та кафе, де продають шоколад ручної роботи, цукерки та інші солодощі [8]. Їх перший магазин був відкритий у Львові у 2012 році, і з того часу вони розширилися до інших міст України.

Львівська майстерня шоколаду відома своїми високоякісними солодощами, виготовленими з натуральних інгредієнтів. Вони пропонують широкий асортимент шоколадних цукерок, включаючи класичні смаки, такі як темний шоколад, молочний шоколад та білий шоколад, а також більш унікальні смаки, такі як лаванда, перець чилі та бекон. Вони також продають шоколадні плитки, трюфелі, праліне та інші солодощі.

Окрім солодощів, Львівська майстерня шоколаду також пропонує каву, чай та інші напої. У деяких їхніх магазинах також є кафе, де можна посидіти та насолодитися ласощами.

Львівська майстерня шоколаду дуже відома за використання високоякісних інгредієнтів для створення своїх шоколадних шедеврів. Вони мають дуже широкий асортимент, від класичних плиток шоколаду до вишуканих цукерок з екзотичними начинками. Їхні майстри шоколаду постійно створюють нові та цікаві рецепти, тому тут часто з’являються нові та цікаві позиції. Також Львівська майстерня шоколаду пропонує екскурсії по майстерні, завдяки чому, можна на

власні очі побачити як виготовляється шоколад. У деяких магазинах є кафе, де можна насолодитися чашкою кави або гарячого шоколаду зі смачним десертом. Ціни у Львівській майстерні шоколаду вище середніх, але це виправдано високою якістю їх продукції.

Рис. 1.5 Сайт магазину Львівської майстерні шоколаду (головна сторінка)

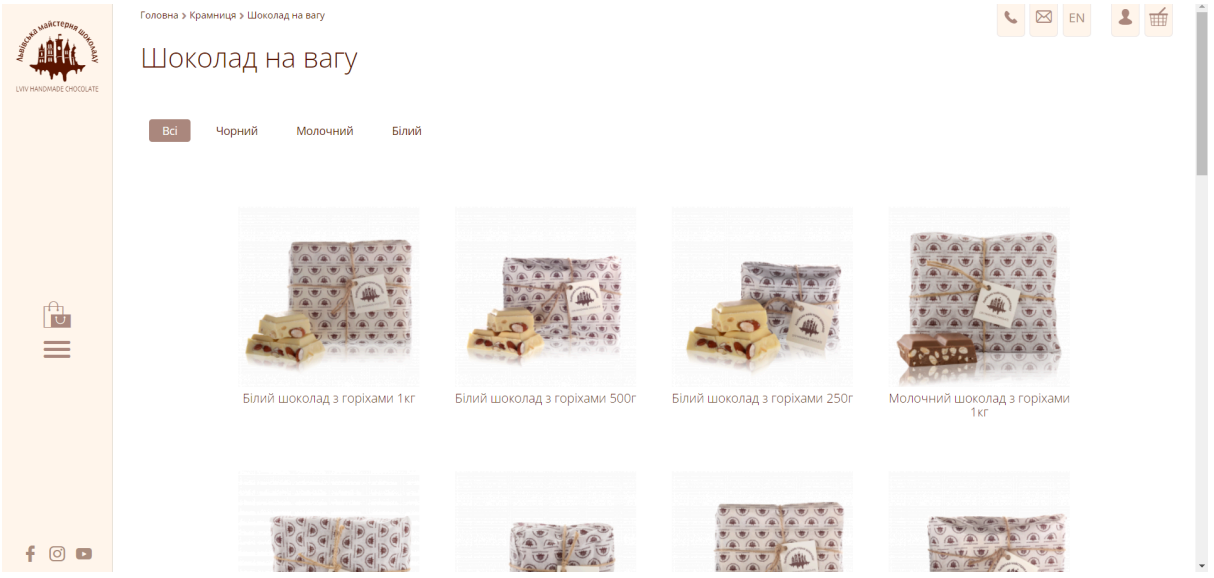


Рис. 1.6 Сайт магазину Львівської майстерні шоколаду (сторінка товару)

У процесі аналізу були знайдені наступні переваги та недоліки:

Переваги:

- Сайт має приємний та елегантний дизайн. Коричнево-золота кольорова гама відповідає темі шоколаду та створює відчуття тепла та розкоші.
- Продукти чітко представлені з якісними фотографіями та описом.
- Широкий вибір продуктів
- Сторінки продуктів містять детальний опис, інгредієнти, строк придатності, умови зберігання та алергени.
- На сайті є можливість пошуку товару за назвою
- На сайті є блог, де можна дізнатися про історію шоколаду, рецепти та новини компанії.
- На сайті є інформація про те, як стати частиною франчайзу Львівської майстерні шоколаду.
- Дизайн сайту адаптується до розмірів екрану

Недоліки:

- Для перегляду асортименту необхідно як мінімум перейти на ще дві сторінки
- Незручна навігація між сторінками
- Довге завантаження сторінок (2,5 секунди в середньому)

1.3 Порівняльна таблиця

Проаналізувавши вже існуючі рішення на ринку, були виділені переваги та недоліки кожного з них, завдяки чому можна чітко зрозуміти яким має бути web-застосунок.

Для більш наочного розуміння була створена таблиця порівняння аналогів та застосунку, що розробляється.

Таблиця 1.1

Таблиця порівняння web-застосунків
для продажів крафтового шоколаду

Критерії порівняння	Львівська майстерня шоколаду	WANDER	Spell Chocolate	DreamChocolatery
Можливість створення кастомної шоколадки	-	-	-	+
Можливість створення персонального кабінету	+	-	+	+
Кількість дій для перегляду асортименту	2	2	3	1
Швидкість завантаження сайту	2140 ms	903 ms	966 ms	900 ms
Пошук товару	+	-	+	+

В результаті аналізу було виділено основний функціонал, що має бути реалізований в застосунку

2 ПРОЄКТУВАННЯ WEB-ЗАСТОСУНКУ

2.1 Технічне завдання та вимоги до web-застосунку

Для виконання мети роботи, необхідно розробити web-застосунок для продажів крафтового шоколаду, що буде мати клієнтську та серверну частини, а також базу даних. Клієнтська частина – це те, з чим взаємодіє користувач через web-браузер. Серверна частина відповідає за обробку даних. База даних буде використовуватися для зберігання інформації.

На основі дослідження методів ведення бізнесу з продажів крафтового шоколаду, а також аналізу вже існуючих рішень на ринку, були сформовані функціональні та нефункціональні вимоги до web-застосунку.

Функціональні вимоги:

- Управління асортиментом: Адміністратор має мати змогу додавати нові позиції, редагувати існуючі, а також видаляти товари.
- Завантаження зображень та описів: Для кожного продукту має бути доступним додавання фотографій, які демонструють зовнішній вигляд продукту, ціни та опису товару, що містить інформацію про склад, смакові характеристики, вагу, розмір, та алергени.
- Відображення асортименту: В інтерфейсі користувача мають бути відображені всі наявні в базі даних продукти.
- Категоризація: Для зручного пошуку та навігації по асортименту має бути реалізована система категорій, яка дозволить користувачам швидко знаходити шоколад, що відповідає їхнім вподобанням.
- Персональний кабінет користувача: Користувачі мають мати можливість створювати персональний кабінет та авторизуватися в системі, використовуючи електронну пошту та пароль.

- Збереження історії замовлень: В персональному кабінеті користувача має відображатися історія всіх його замовлень та такі дані як: номер, товар, дата, час, сума, статус замовлення.
- Оновлення інформації: Користувачі мають мати можливість оновлювати свою особисту інформацію та змінювати пароль.
- Пошук товару: Має бути реалізована система пошуку, яка дозволить користувачам знаходити потрібні продукти за назвою.
- Кошик продуктів: Користувачі мають мати можливість додавати продукти до кошика, переглядати його вміст, міняти кількість товарів та видаляти позиції.
- Розрахунок вартості: автоматичний розрахунок загальної вартості товарів в кошику.
- Можливість створення власних шоколадних плиток: Користувач має мати змогу зібрати свою шоколадку з завчасно доданих інгредієнтів таких як: чорний, білий та молочний шоколад та додаткових інгредієнтів таких як: малина, полуниця, горіхи та інші.

Нефункціональні вимоги:

- Мінімалістичний дизайн: Інтерфейс користувача має бути інтуїтивно зрозумілим, кольорова гама біло-сіра, а використання нефункціональних елементів в дизайні має бути мінімізовано.
- Сумісність з різними браузерами: Web-застосунок має коректно відображатися в таких браузерах: Google Chrome, Firefox, Edge та Opera.
- Масштабованість: Система має бути спроєктована таким чином, щоб можна було легко додавати нові модулі та впроваджувати новий функціонал, без шкоди для продуктивності та стабільності роботи web-застосунку.
- Захищеність персональних даних: Персональні дані мають бути захищені від викрадення та неавторизованого змінення.

- Швидке завантаження сторінок: Швидкість завантаження сторінок має бути не більше 1500 мілісекунд.
- Сортування за замовчуванням: За замовчуванням продукти мають відображатися на сторінці в порядку їхнього додавання в базу даних з найпершого доданого продукту.

2.2 Вибір засобів розробки

2.2.1 Spring

Для розробки серверної частина web-застосунку обрано фреймворк фреймворк Spring, бо він надає великий набір інструментів та бібліотек, які спрощують та прискорюють процес розробки.

Spring Framework, або просто Spring [9] - це відкритий фреймворк для розробки web-застосунків мовою Java [10]. Даний фреймворк полегшує написання коду та робить його більш організованим. Spring є популярним вибором для Java розробників, оскільки він допомагає писати чистий, модульний та масштабований код.

Spring має наступні переваги:

- Spring керує об'єктами програми та їх взаємодією,що допомагає уникати написання великої кількості коду.
- Замість того, щоб об'єкти створювали свої власні залежності, Spring робить це за них. Це робить код більш гнучким та легким для тестування.
- Spring пропонує широкий набір модулів для різних задач, таких як доступ до бази даних, безпека, та розробка web-застосунків.

Рис. 2.1 Логотип Spring

2.2.2 Angular

Для розробки клієнтської частини web-застосунку обрано фреймворк Angular, тому що він відомий як рішення для створення швидких web-застосунків. Даний фреймворк розробила та обслуговує компанія Google, він є надійним і стабільним, із масовою підтримкою спільноти розробників.

Angular – це фреймворк з відкритим кодом для розробки клієнтської частини web-застосунків [11]. Він написаний мовою програмування TypeScript [12], що є надмножиною мови JavaScript [13].

Angular має наступні переваги:

- В Angular використовується компонентний підхід до розробки, що робить код більш модульним та легшим для розуміння.
- Автоматична синхронізація моделі з інтерфейсом користувача, що дозволяє бачити зміни в реальному часі.
- Angular пропонує набір директив, які розширюють можливості HTML. Ці директиви можуть використовуватися для додавання функціональності, такої як обробка форм, валідація даних та анімація.

Рис. 2.2 Логотип Angular

2.2.3 PostgreSQL

Для зберігання даних про товари, користувачів та замовлення в web-застосунку використовується база даних PostgreSQL, яка забезпечує широкий спектр типів даних.

PostgreSQL - це відкрита об'єктно-реляційна система керування базами даних [14]. Вона відома своєю надійністю та гнучкістю, що робить її популярним вибором для широкого спектру задач, від web-розробки та електронної комерції до наукових досліджень та аналітики даних.

PostgreSQL має наступні переваги:

- PostgreSQL підтримує як реляційну, так і об'єктно-орієнтовану модель даних, що дозволяє зберігати та обробляти складні дані у гнучкій та ефективній формі.
- В PostgreSQL використовується стандарт мови запитів SQL, що робить її легкою для вивчення та використання розробниками з досвідом роботи з іншими СУБД, що використовують мову SQL.
- PostgreSQL відома своєю стабільністю та стійкістю до збоїв та пропонує широкий спектр функцій безпеки для захисту даних від несанкціонованого доступу.
- PostgreSQL є програмним забезпеченням з відкритим кодом, що означає, що її можна використовувати, змінювати та розповсюджувати безкоштовно. Це робить її популярним вибором для розробників, які шукають економічно ефективну та надійну СУБД.

Рис. 2.3 Логотип PostgreSQL

2.2.4 IntelliJ IDEA

Середовищем розробки для серверної частини web-застосунку було обрано IntelliJ IDEA

IntelliJ IDEA - це популярне інтегроване середовище розробки для мов програмування Java, Kotlin, Scala, C++, Python, PHP та багато інших. Вона розроблена компанією JetBrains, яка також відома створенням інструментів для розробників [15].

IntelliJ IDEA має наступні переваги:

- Функції автозаповнення коду, які допомагають писати код швидше та з меншою кількістю помилок.
- Інструменти для рефакторингу коду, які допомагають покращити структуру коду.
- Інструмент відлагодження, який допомагає знаходити та виправляти помилки в коді.
- IntelliJ IDEA інтегрується з популярними системами контролю версій, такими як Git, що допомагає контролювати версії проєкту.
- Для IntelliJ IDEA доступні плагіни, які розширюють її функціональність та додають нові можливості.

IntelliJ IDEA використовується розробниками по всьому світу для створення широкого спектру програмного забезпечення, від простих web-застосунків до складних корпоративних систем.

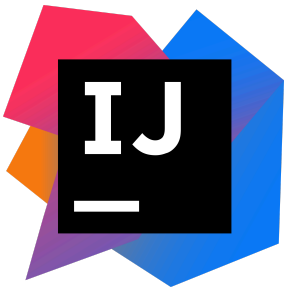


Рис. 2.4 Логотип IntelliJ IDEA

2.2.5 WebStorm

Середовищем розробки для клієнтської частини web-застосунку було обрано WebStorm.

WebStorm – це інтегроване середовище розробки від компанії JetBrains, спеціально створене для web-розробки [16].

Воно ґрунтується на платформі IntelliJ IDEA і пропонує широкий спектр функцій, які полегшують написання, тестування та налагодження коду JavaScript, HTML та CSS .

WebStorm має наступні переваги:

- Всебічна підтримка HTML, CSS та JavaScript, включаючи автодоповнення коду, синтаксичний аналіз та перевірку помилок.
- Інтеграція з популярними JavaScript фреймворками, такими як React, Angular, Vue.js та Node.js, пропонуючи спеціальні інструменти та підказки для їх ефективного використання.
- Інструмент відлагодження, який дозволяє відстежувати виконання JavaScript коду, знаходити та виправляти помилки.
- WebStorm інтегрується з Git та іншими системами контролю версій, полегшуючи спільну роботу над проєктами та відстеження змін.
- WebStorm можна розширювати за допомогою плагінів, які додають нові функції та можливості.

WebStorm використовується розробниками по всьому світу для створення сучасних web-застосунків. Гнучкість та широкий спектр функцій роблять його популярним вибором як для досвідчених, так і для початківців web-розробників.

Рис. 2.5 Логотип WebStorm

2.2.6 Git та GitHub

Для контролю за розробкою та збереження файлів web-застосунку використовується Git та GitHub.

Git - це розподілена система контролю версій та спільної роботи, яка відстежує зміни у файлах проєкту. Вона використовується розробниками для спільної роботи над кодом та відстежування історії зміни файлів [17].

Git має наступні переваги:

- За допомогою Git можна зберігати поточний стан проєкту в будь-який момент часу. Це дозволяє, у разі необхідності, повернутися до попередніх версій файлів.
- Git дозволяє створювати розгалуження для експериментів та об'єднувати їх назад до основної гілки.
- Git захищає цілісність даних за допомогою контрольних сум. Це гарантує, що жоден файл не може бути змінений непомітно.

Рис. 2.6 Логотип Git

GitHub - це онлайн-платформа, розроблена для роботи з Git. Вона надає зручний інтерфейс та функціонал для зберігання, управління та спільної роботи над Git-репозиторіями [18].

GitHub має наступні переваги:

- GitHub дозволяє зберігати Git-репозиторії на їх серверах. Це робить репозиторії доступними з будь-якого місця та дозволяє легко працювати з іншими розробниками над одним проєктом.

- GitHub використовується для розміщення проєктів з відкритим кодом розробниками по всьому світу.
- GitHub має увесь функціонал Git, такий як відстеження історії версій, розгалуження та об'єднання коду. Це полегшує керування змінами в проєкті та повернення до попередніх версій.
- Крім основних функцій, GitHub пропонує багато додаткових можливостей, таких як сторінки для документації проєкту, система відстеження помилок, огляд коду та інтеграції з середовищами розробки.

Рис. 2.7 Логотип Github

2.3 Моделювання архітектури web-застосунку

Для розуміння того, як має бути розроблений та як в подальшому має функціонувати web-застосунок, створено наступні схеми на діаграми:

2.3.1 Схема архітектури web-застосунку

Схема загальної архітектури web-застосунку демонструє функціонування системи в цілому. Web-застосунок має трирівневу клієнт-серверну архітектуру.

Клієнт складається з користувацького інтерфейсу, з яким взаємодіє користувач через web-браузер. Сервер відповідає за обробку даних та виконання бізнес-логіки. База даних використовується для зберігання інформації про користувачів, товари, замовлення, та інших даних, необхідних для роботи web-застосунку [5].

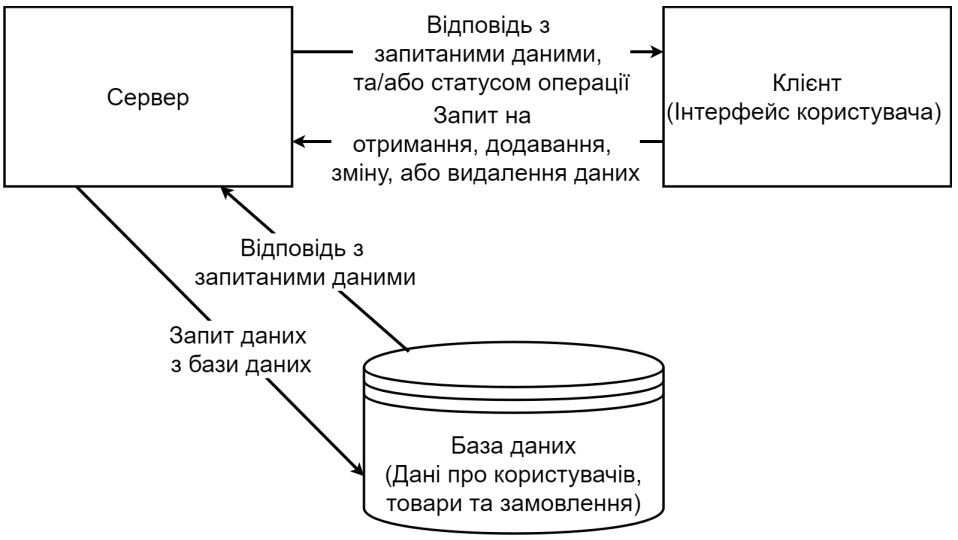


Рис. 2.8 Схема клієнт-серверної архітектури web-застосунку

2.3.2 Мапа web-застосунку

Для розуміння того, які сторінки web-застосунку мають бути розроблені, створено мапу web-застосунку, на якій зображено всі сторінки та зв’язки між ними. Дана мапа дає розуміння того, з яких сторінок можна потрапити на інші.

Рис. 2.9 Мапа web-застосунку

2.3.3 Діаграма класів

Діаграма класів – це тип статичної UML-діаграми, яка використовується для моделювання об'єктно-орієнтованих систем. Діаграма класів візуалізує внутрішню структуру web-застосунку, показує класи та зв'язки між ними. На даній діаграмі зображені основні класи серверної частини, поля класів та їх типи даних, та відношення між класами. Цифри 1 - 0..n, 0..n - 0..n, 1 - 1 позначають відношення один до багатьох, багато до багатьох та один до одного відповідно. В подальшому, ці класи будуть застосовуватись для створення таблиць в базі даних.

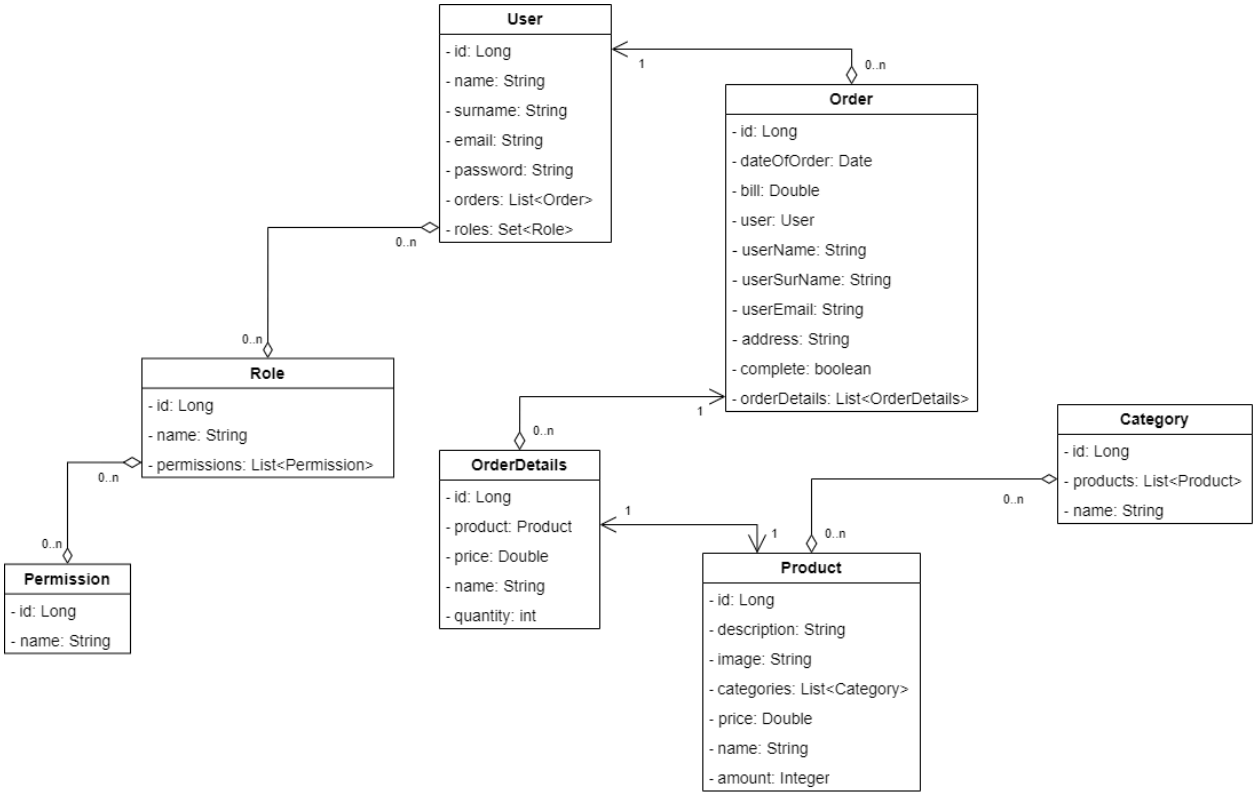


Рис. 2.10 Діаграма класів

2.3.4 Діаграма варіантів використання

Діаграма варіантів використання візуалізує взаємодію між акторами (користувачами, або зовнішніми системами) та функціональними можливостями системи. Дана діаграма описує як система має функціонувати з точки зору користувача, не вдаючись до технічних деталей. Стрілкою include позначається необхідність виконання однієї дії перед іншою. Стрілкою extend позначається функція, що доповнює іншу.

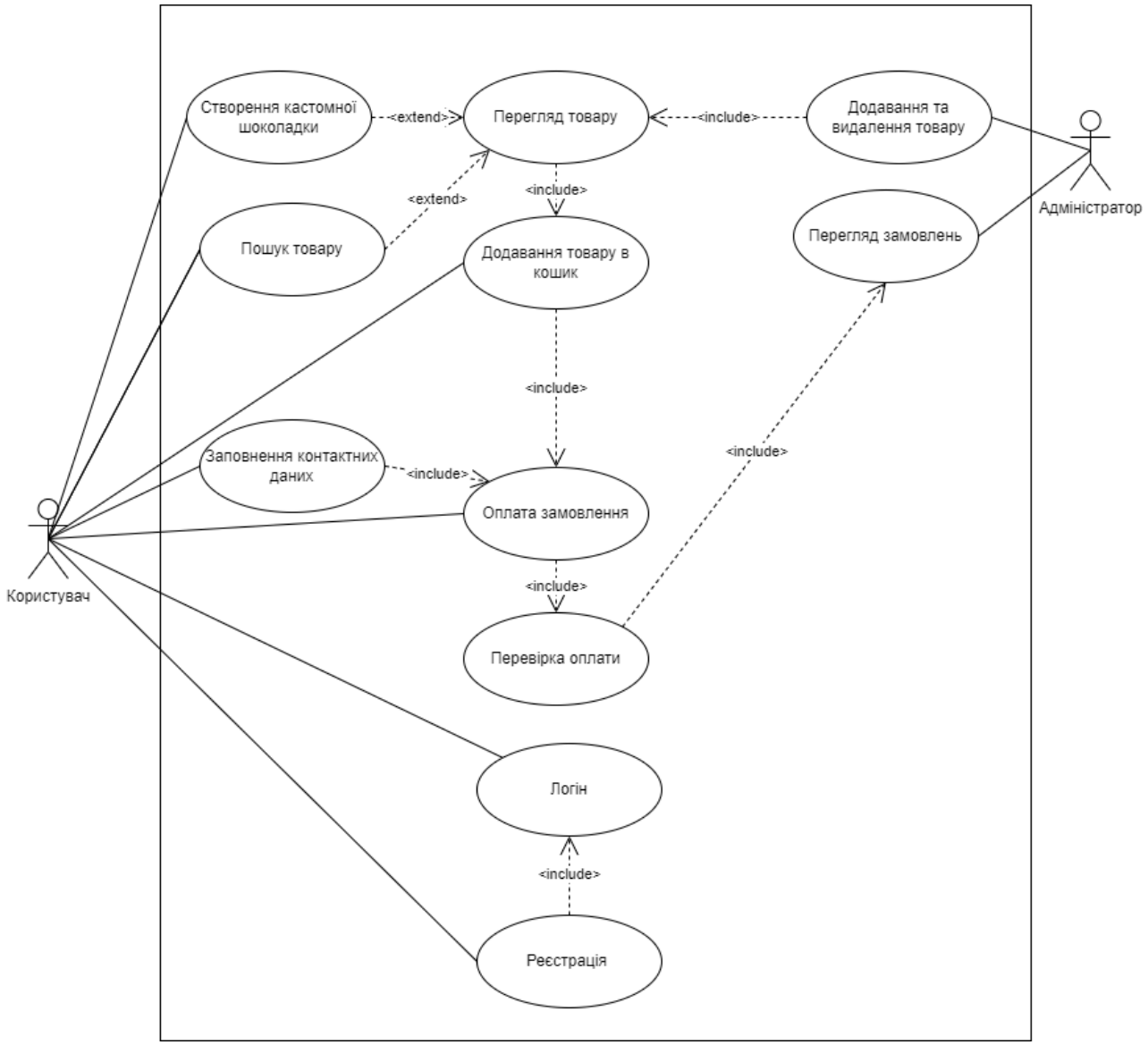


Рис. 2.11 Діаграма варіантів використання

2.3.5 Діаграма варіантів діяльності

Діаграма діяльності візуалізує порядок виконання операцій та ілюструє послідовність дій від однієї до наступної. Вона демонструє цілісну роботу системи та вважається розширеним варіантом блок-схеми. Дії можуть виконуватися людьми, програмними компонентами або комп'ютерами. Потік на діаграмі діяльності переходить від однієї операції до іншої та може бути послідовним, розгалуженим, або одночасним.

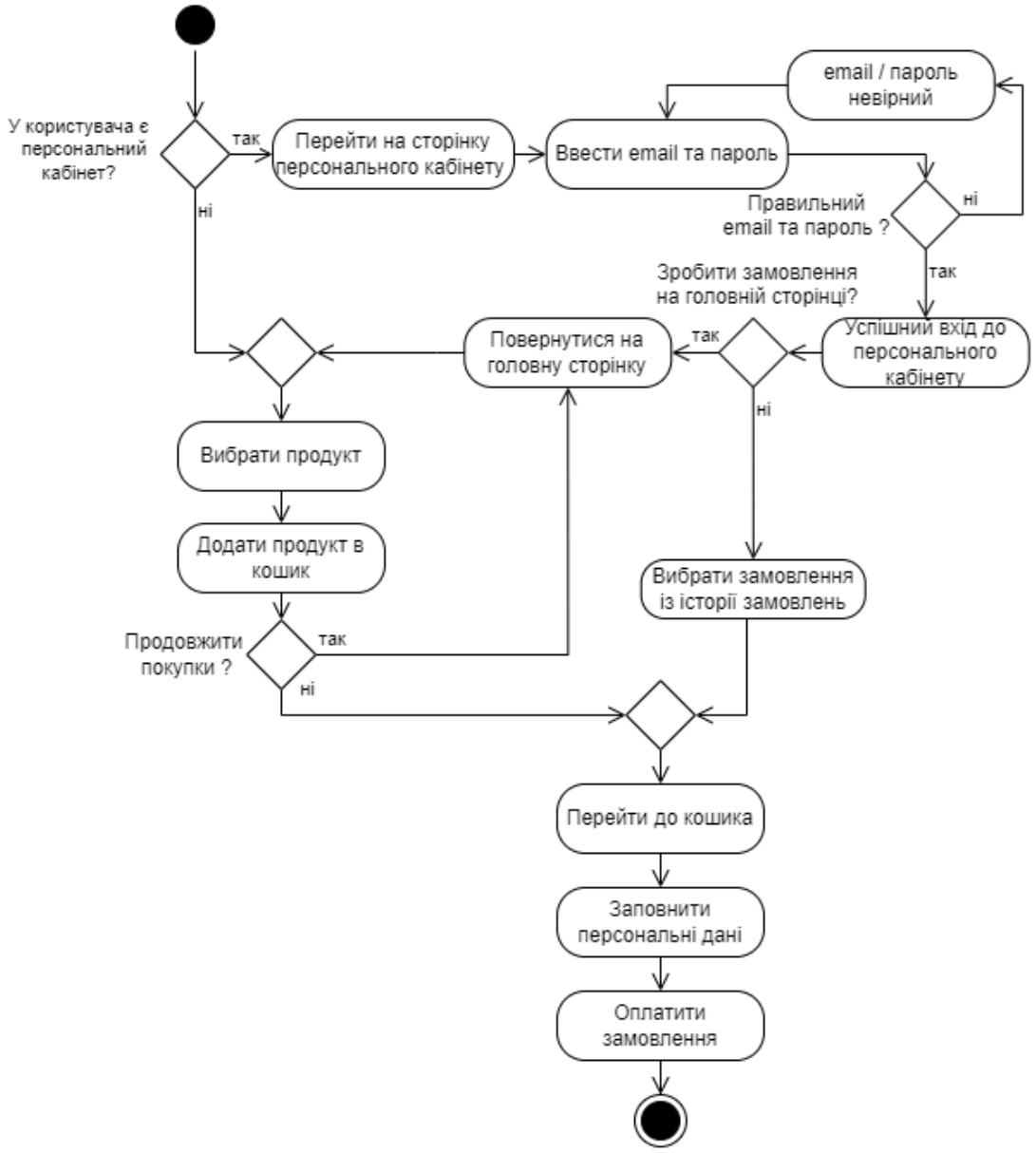


Рис. 2.12 Діаграма діяльності (дії для оформлення замовлення)

3 ОПИС РОЗРОБКИ ТА ТЕСТУВАННЯ WEB-ЗАСТОСУНКУ

3.1 Серверна частина web-застосунку

Для розробки серверної частини web-застосунку використовувався патерн Controller-Service-Repository. Для того щоб визначити яку роль в системі відіграє клас, використовуються анотації - спеціальна форма метаданих в Java.

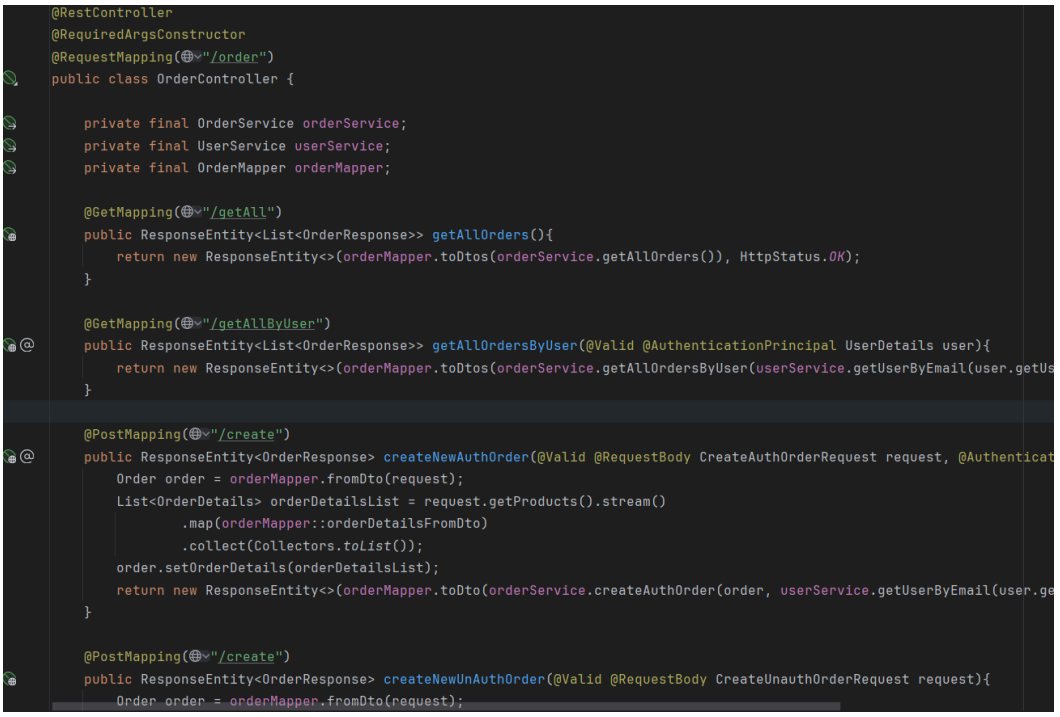
3.1.1 Контролер

Контролер - це клас за яким дані з HTTP-запитів надходять до бізнес-логіки програми та повертаються результати. Контролер не містить ніякої бізнес-логіки, функція це маршрутизація запитів до відповідних сервісів та підготовка відповіді, він діє як інтерфейс між користувачем та іншими шарами програми. Контролер отримує HTTP-запити від клієнта, після чого аналізує їх та витягує інформацію, необхідну для обробки, такі як, URI, заголовок та тіло запиту, що перетворюється на DTO за допомогою маперів.

DTO - це об'єкти, які використовуються для передачі даних між різними частинами системи і містять лише ті дані, які необхідні для конкретного випадку їх використання.

```
13  @Data
14  @AllArgsConstructor
15  @NoArgsConstructor
16  @Builder
17  public class CreateAuthOrderRequest {
18
19      @NotNull(message = "id cannot be null")
20      private Long id;
21
22      @NotNull(message = "address cannot be null")
23      private String address;
24
25      @NotEmpty(message = "Products cannot be null")
26      private List<OrderDetails> products;
27      private boolean complete;
28  }
```

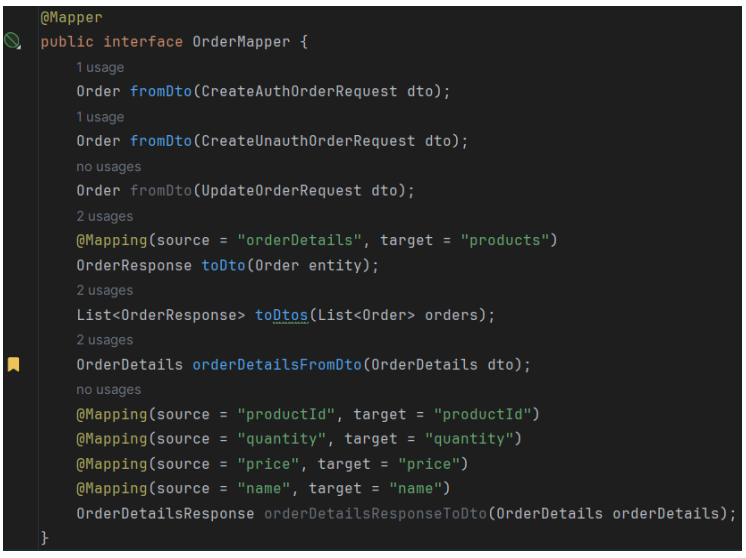
Рис. 3.1 DTO “CreateAuthOrderRequest”



```
@RestController
@RequiredArgsConstructor
@RequestMapping("/order")
public class OrderController {

    private final OrderService orderService;
    private final UserService userService;
    private final OrderMapper orderMapper;

    @GetMapping("/getAll")
    public ResponseEntity<List<OrderResponse>> getAllOrders(){
        return new ResponseEntity<>(orderMapper.toDtos(orderService.getAllOrders()), HttpStatus.OK);
    }

    @GetMapping("/getAllByUser")
    public ResponseEntity<List<OrderResponse>> getAllOrdersByUser(@Valid @AuthenticationPrincipal UserDetails user){
        return new ResponseEntity<>(orderMapper.toDtos(orderService.getAllOrdersByUser(userService.getUserByEmail(user.getUs
    }

    @PostMapping("/create")
    public ResponseEntity<OrderResponse> createNewAuthOrder(@Valid @RequestBody CreateAuthOrderRequest request, @Authenticat
        Order order = orderMapper.fromDto(request);
        List<OrderDetails> orderDetailsList = request.getProducts().stream()
                .map(orderMapper::orderDetailsFromDto)
                .collect(Collectors.toList());
        order.setOrderDetails(orderDetailsList);
        return new ResponseEntity<>(orderMapper.toDto(orderService.createAuthOrder(order, userService.getUserByEmail(user.ge
    }

    @PostMapping("/create")
    public ResponseEntity<OrderResponse> createNewUnAuthOrder(@Valid @RequestBody CreateUnauthOrderRequest request){
        Order order = orderMapper.fromDto(request);
```

Рис. 3.2 Контролер “OrderController”

Мапер - це клас, або інтерфейс, який відповідає за перетворення даних з одного формату в інший. Для цього він використовує прописані правила того, як перетворювати кожне поле.



```
@Mapper
public interface OrderMapper {
    1 usage
    Order fromDto(CreateAuthOrderRequest dto);
    1 usage
    Order fromDto(CreateUnauthOrderRequest dto);
    no usages
    Order fromDto(UpdateOrderRequest dto);
    2 usages
    @Mapping(source = "orderDetails", target = "products")
    OrderResponse toDto(Order entity);
    2 usages
    List<OrderResponse> toDtos(List<Order> orders);
    2 usages
    OrderDetails orderDetailsFromDto(OrderDetails dto);
    no usages
    @Mapping(source = "productId", target = "productId")
    @Mapping(source = "quantity", target = "quantity")
    @Mapping(source = "price", target = "price")
    @Mapping(source = "name", target = "name")
    OrderDetailsResponse orderDetailsResponseToDto(OrderDetails orderDetails);
}
```

Рис. 3.3 Мапер “OrderMapper”

Після обробки запиту контроллер делегує роботу відповідному сервісу, який відповідає за конкретну функціональність.

3.1.2 Сервіс

Сервіс - це клас, що містить бізнес-логіку системи. Він містить логіку, необхідну для виконання конкретних функціональних вимог та відповідає за виконання дій, таких як автентифікація користувачів, управління даними та обробка транзакцій, а також за генерацію повідомлень у разі виникнення помилки під час виконання операції. Для доступу до даних, необхідних для виконання дії, сервіс використовує репозиторій.

```java
@Service
@RequiredArgsConstructor
public class OrderService {
    private final ProductService productService;
    private final OrderRepository orderRepository;

    1 usage
    @Transactional(readOnly = true)
    public List<Order> getAllOrders() {
        return orderRepository.findAll();
    }

    1 usage
    @Transactional(readOnly = true)
    public List<Order> getAllOrdersByUser(User user) {
        return orderRepository.findAllByUser(user);
    }

    1 usage
    @Transactional
    public Order createAuthOrder(Order order, User user) {
        order.setUser(user);
        order.setUserName(user.getName());
        order.setUserSurName(user.getSurname());
        order.setUserEmail(user.getEmail());
        List<OrderDetails> orderDetails = order.getOrderDetails();
        Double bill = 0.0;
        for (OrderDetails orderDetail : orderDetails) {
            Product product = orderDetail.getProduct();
            orderDetail.setName(product.getName());
            orderDetail.setPrice(product.getPrice());
            bill += product.getPrice() * orderDetail.getQuantity();
```

Рис. 3.4 Сервіс “OrderService”

3.1.3 Репозиторій

Репозиторій - це інтерфейс, що надає доступ до даних за допомогою абстракцій для доступу до бази даних. Репозиторій надає методи для створення, читання, оновлення та видалення даних з бази даних і приховує деталі реалізації джерела даних, що робить систему більш гнучкою. Також репозиторій має обробляти помилки пов'язані з даними та генерувати повідомлення у разі виникнення помилки. Репозиторії не містять жодної бізнес-логіки, адже їхня єдина мета - надання даних для сервісів. В Spring репозиторії реалізований за допомогою Spring Data JPA.

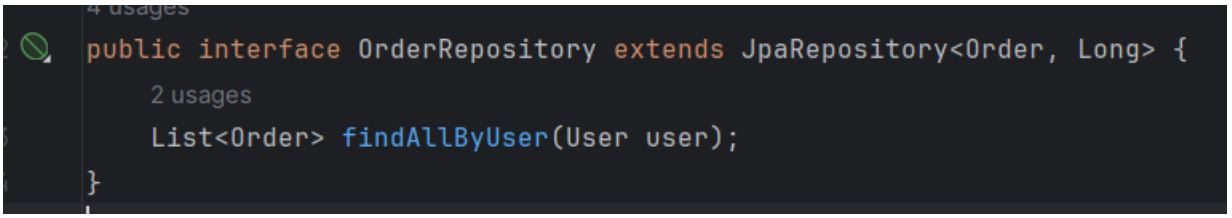


Рис. 3.5 Репозиторій “OrderRepository”

Spring Data JPA - це API, що ґрунтується на Hibernate, котрий в свою чергу є імплементацією JPA - стандартного інтерфейсу для об'єктно-реляційного відображення в Java [19, 20]. Spring Data JPA надає зручний спосіб для збереження та отримання об'єктів Java з реляційних баз даних. Він перетворює об'єкти Java на таблиці і виконує запити до базі даних, завдяки чому немає потреби писати SQL-код для доступу до даних. Для цього Spring Data JPA використовує JPQL, який є декларативною мовою запитів, схожою на SQL.

Після отримання даних від репозиторія сервісом та здійснення ним бізнес-логіки, контроллер готує дані для клієнта, так само через мапер у вигляді TDO, після чого перетворює дані у HTTP-відповідь.

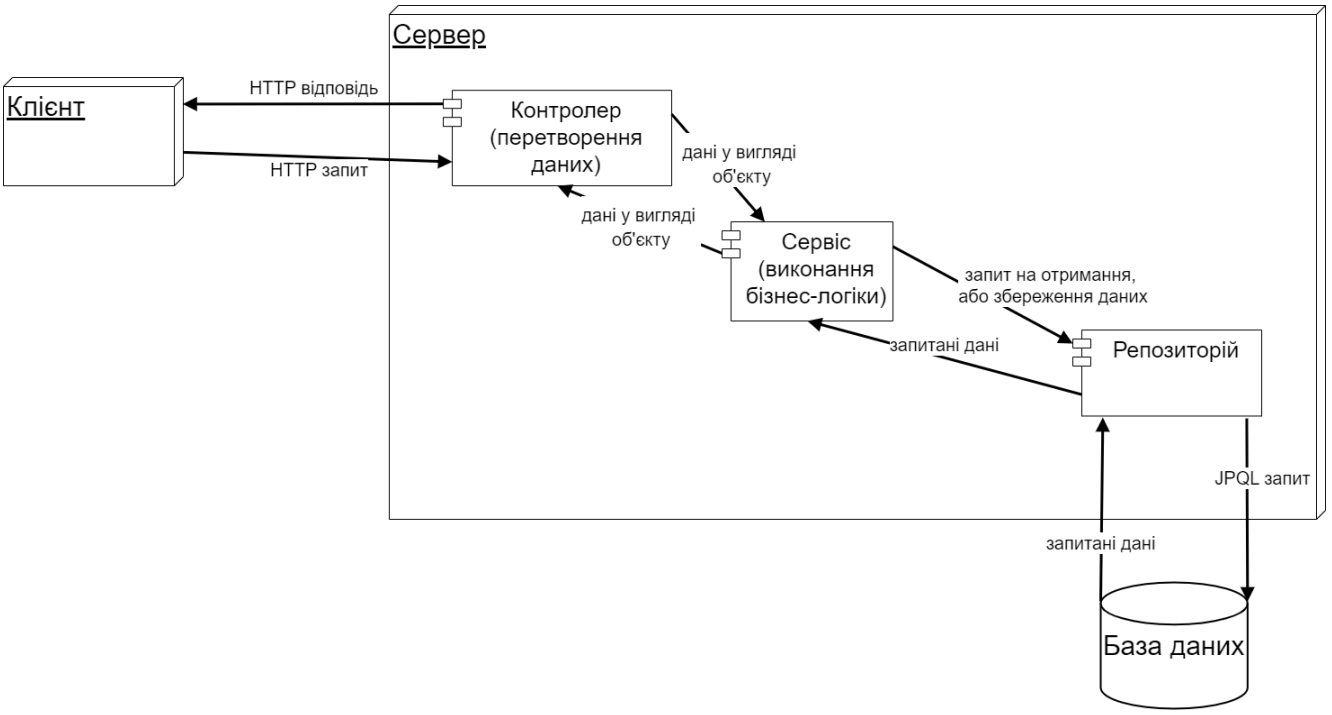


Рис. 3.6 Схема роботи сервера

3.1.4 Модель

Моделі - це класи, що використовуються для створення таблиць в базі даних. Поля, що прописуються в класі, будуть перетворені в поля відповідної таблиці в базі даних. Для більш гнучкого налаштування таблиці використовуються наступні анотації:

@Id: позначення поля, що буде первинним ключем таблиці.

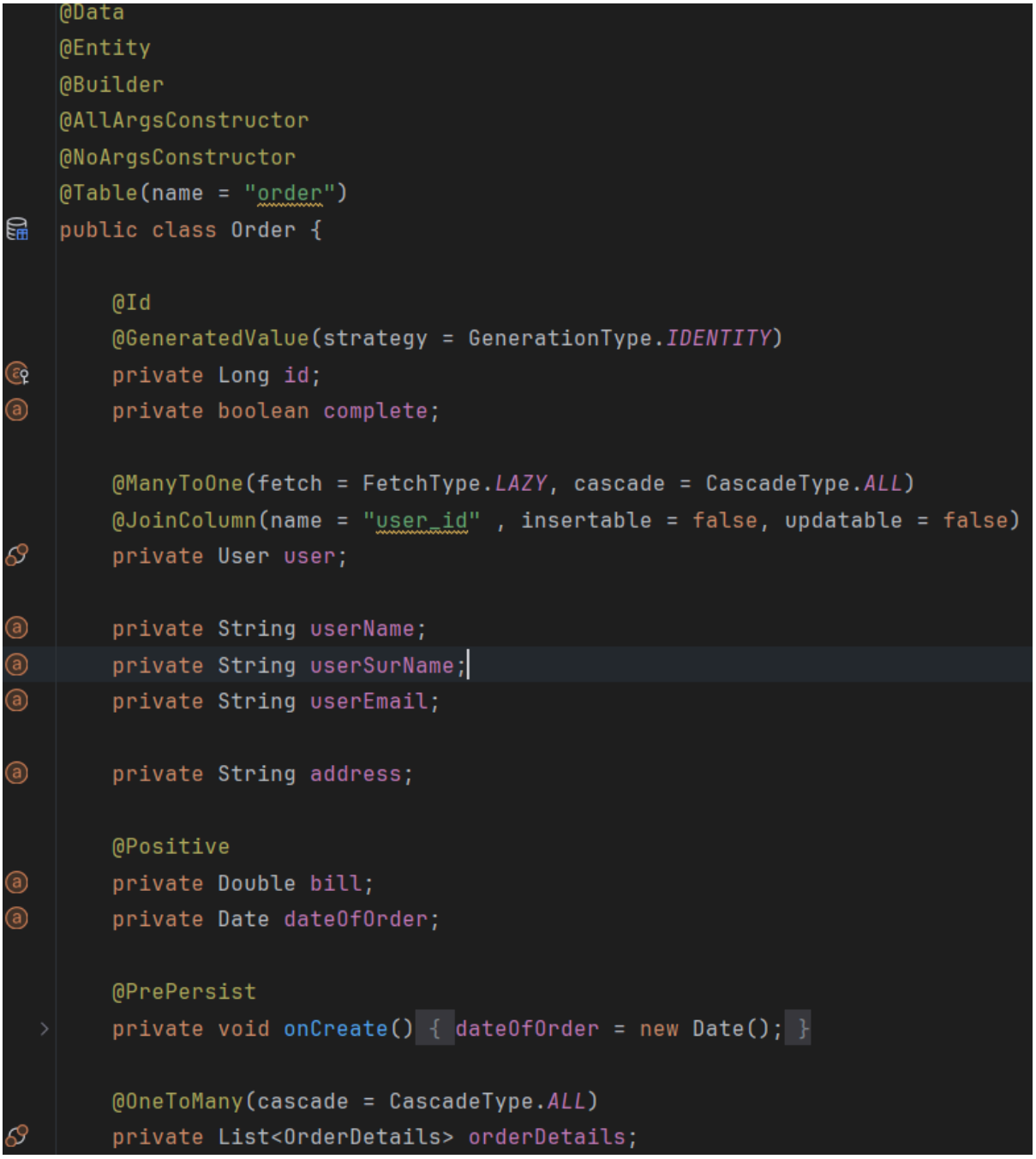
@GeneratedValue: визначення того, як буде генеруватися значення первинного ключа.

@Table: визначення властивостей таблиці. За допомогою цієї анотації можна дати назву таблиці в базі даних.

@Column: визначення властивостей поля таблиці. За допомогою цієї анотації можна також визначити такі характеристики стовпчика, як ім'я, тип даних, довжина.

@JoinColumn: використовується для визначення зв'язків між таблицями.

@OneToMany, @ManyToMany: використовуються для визначення різних типів зв'язку між таблицями.



```java
@Data
@Entity
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "order")
public class Order {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private boolean complete;

    @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @JoinColumn(name = "user_id" , insertable = false, updatable = false)
    private User user;

    private String userName;
    private String userSurName;
    private String userEmail;

    private String address;

    @Positive
    private Double bill;
    private Date dateOfOrder;

    @PrePersist
    private void onCreate() { dateOfOrder = new Date(); }

    @OneToMany(cascade = CascadeType.ALL)
    private List<OrderDetails> orderDetails;
```

Рис. 3.7 Модель “Order”

Для кожної таблиці був створений контролер, сервіс, репозиторій, та мапер.

За автентифікацію та безпеку відповідає модуль Spring Security [21]. Він надає засоби побудови захисту від багатьох поширених вразливостей, таких як: SQL-ін'єкції, XSS, CSRF, Clickjacking, Session Fixation Доступ до ресурсів відбувається на основі ролей та дозволів для них, що гарантує доступ до даних

лише тим користувачам, які мають дозвіл ними оперувати. Для авторизації користувачів використовується JWT.

JWT - це формат для передачі інформації між сторонами як частина системи авторизації [22]. Він використовується для удостовірення особистості користувача або сервера і надання доступу до захищених ресурсів. JWT складається з трьох частин, розділених крапками:

- Заголовок: містить інформацію про тип токена, алгоритм шифрування та метадані.
- Корисне навантаження: містить фактичні дані, які передаються, такі як електронна пошта, пароль, роль, та час створення.
- Підпис: використовується для перевірки автентичності та цілісності токена, за допомогою криптографічного алгоритму.

При кожній реєстрації нового користувача, або вході в персональний кабінет вже існуючого, створюється новий токен та шифрується пароль

3.2 Клієнтська частина web-застосунку

Інтерфейс користувача розробленого web-застосунку містить наступні сторінки та елементи на них:

- Головна сторінка:
    - Назва магазину у верхньому колонтитулі.
    - Перехід на сторінку оформлення замовлення у вигляді кошику.
    - Перехід на сторінку персонального кабінету у вигляді користувача.
    - Навігаційна панель з категоріями товару.
    - Перехід на сторінку створення персональної шоколадки.
    - Пошук товару.
    - Каталог у вигляді карток товару.

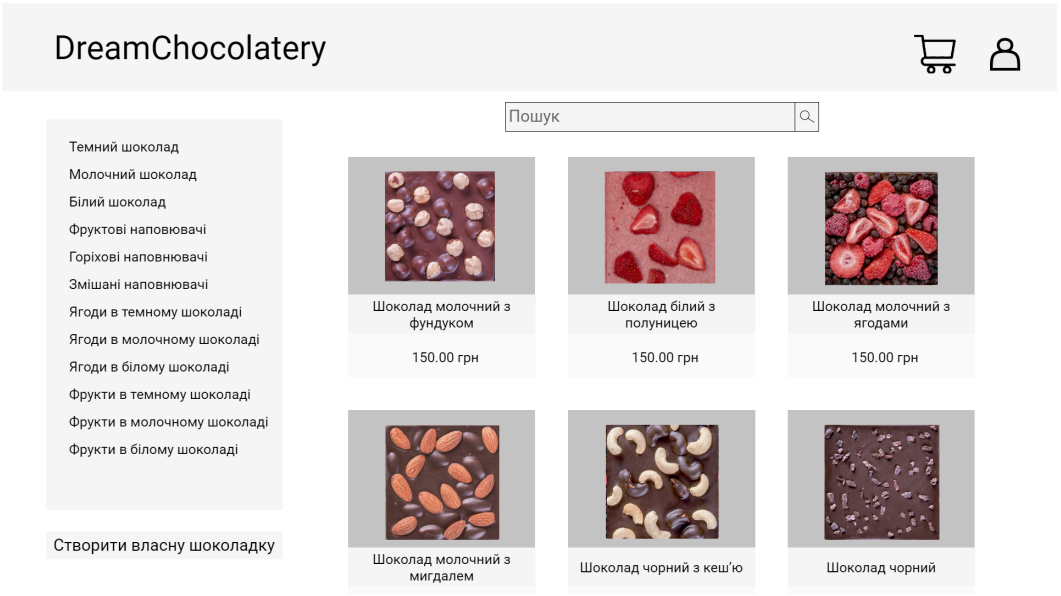


Рис. 3.8 дизайн головної сторінки

- Сторінка продукту:
  - Назва товару.
  - Зображення товару.
  - Опис.
  - Ціна товару.
  - Кнопка “Додати в кошик”.

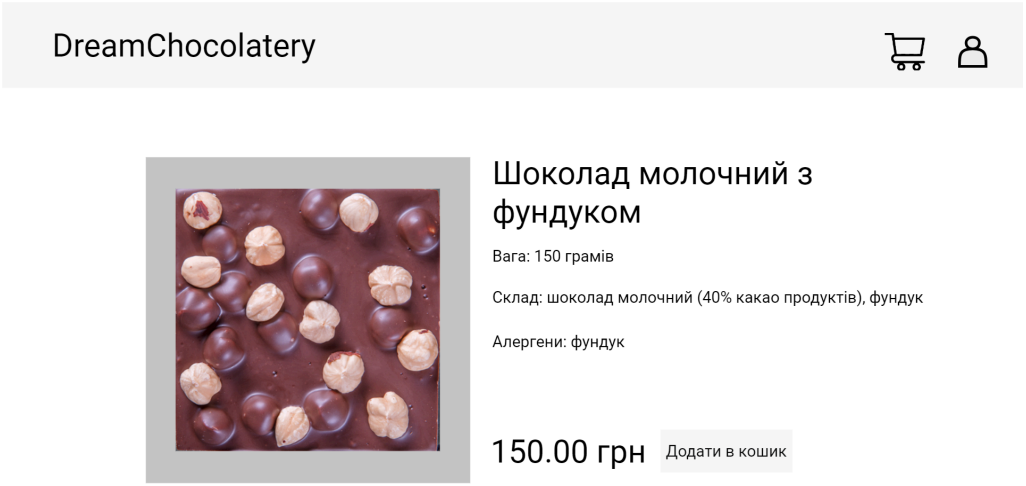


Рис. 3.9 дизайн сторінки продукту

- Сторінка створення персональної шоколадки:
    - Вибір основи шоколадки та її ціна.
    - Вибір наповнювачів, їх назва, ціна та кнопки додати (+) та прибрати (-).
    - Ціна товару.
    - Кнопка “Додати в кошик”.

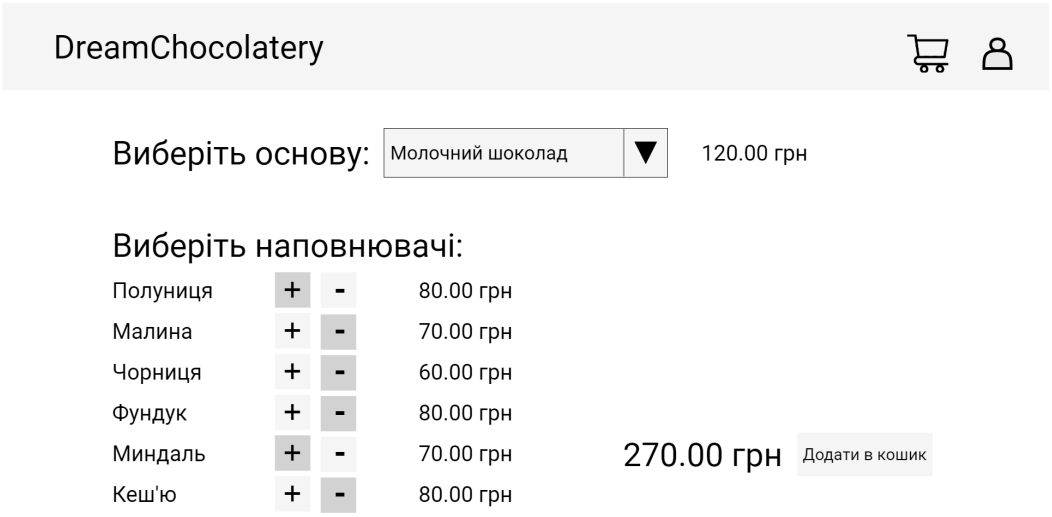


Рис. 3.10 дизайн сторінки створення персональної шоколадки

- Сторінка оформлення замовлення:
    - Обрані товари: кожен обраний товар, його зображення, назва, ціна, кількість та кнопки додати (+) та прибрати (-) 1 одиницю даного товару.
    - Поля для введення імені, прізвища, електронної пошти та адреси доставки. Якщо авторизований, поля імені, прізвища, електронної пошти заповнюються автоматично.
    - Сума замовлення.
    - Кнопка “Оплатити”

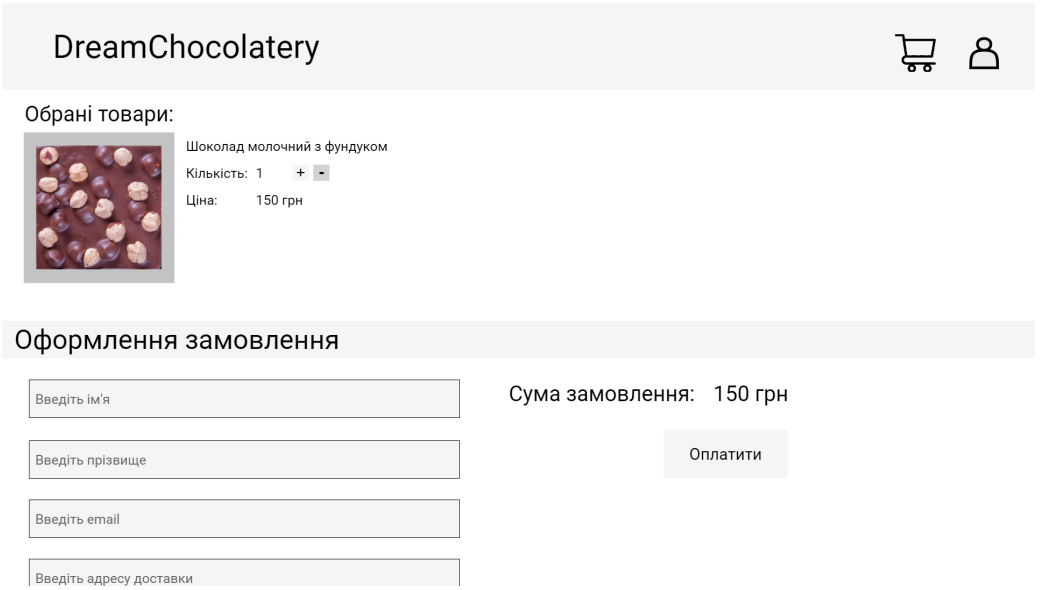


Рис. 3.11 дизайн сторінки оформлення замовлення

- Сторінка входу до персонального кабінету:
  - Поля для вводу електронної пошти та паролю.
  - Кнопка “Увійти”.
  - Кнопка переходу на сторінку реєстрації.

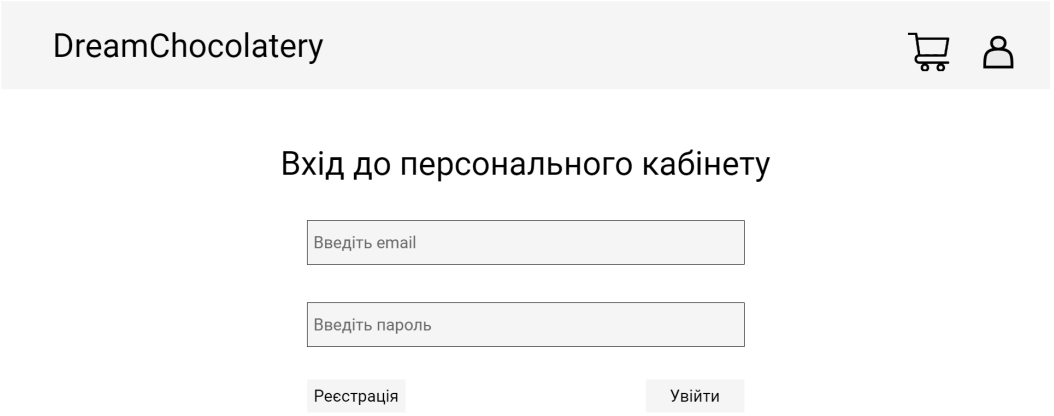


Рис. 3.12 дизайн сторінки входу до персонального кабінету

- Сторінка реєстрації:
    - Поля для вводу електронної пошти, імені, прізвища та паролю.
    - Кнопка “Зареєструватись”.

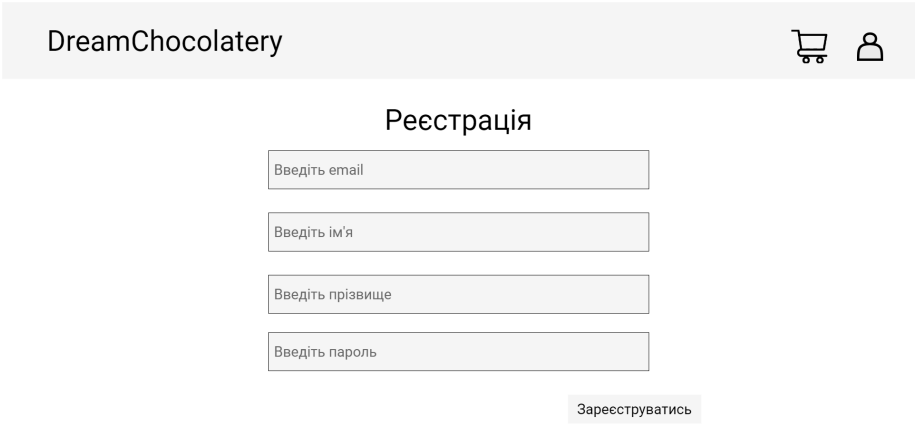


Рис. 3.13 дизайн сторінки реєстрації

- Сторінка персонального кабінету:
    - Відображення імені, прізвища та електронної пошти.
    - Кнопка “Редагувати”, по натисканню на яку відкривається модальне вікно з можливістю редагувати дані.
    - Історія всіх замовлень користувача

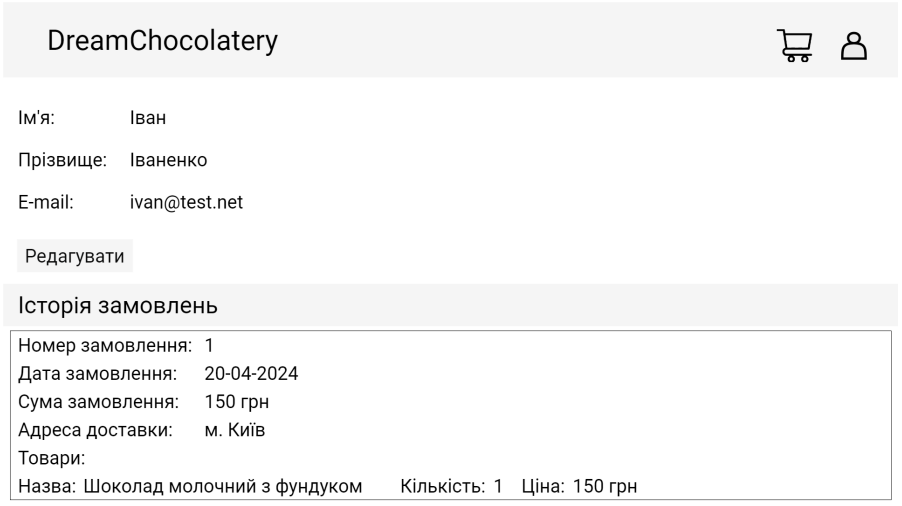


Рис. 3.14 дизайн сторінки персонального кабінету

Кожна сторінка складається з трьох модулів: каркас, стиль та функціонал.

3.2.1 HTML

Каркасом для сторінок web-застосунку слугує HTML - це мова розмітки web-сторінок, яка використовується з моменту створення всесвітньої мережі Інтернет. За її допомогою web-браузери здатні інтерпретувати текст, зображення, посилання та інші елементи, відображаючи їх на сторінках web-застосунків. HTML описує їхню структуру та вміст, визначає ієрархію та логічні зв'язки між елементами сторінки, розбиваючи її на чіткі секції, такі як верхній колонтитул, основний контент, навігаційна панель та нижній колонтитул. HTML-документ має чітку структуру, що створюється за допомогою тегів:

- <!DOCTYPE html> - повідомляє web-браузеру, що цей файл є HTML-документом.
- <html> - позначає початок та кінець сторінки.
- <head> - містить метадані про сторінку, такі як заголовок, опис та посилання на зовнішні файл.
- <body> - позначає вміст сторінки.
- <header> - позначає верхній колонтитул сторінки.
- <article> - позначає самостійний розділ контенту сторінки.
- <nav> - позначає для навігаційну панель.
- <footer> - позначає нижній колонтитул сторінки.
- - універсальний контейнерний блок, який використовується для групування елементів HTML. Він не має власної семантики, але може використовуватися разом з класами CSS для надання йому семантичного значення.
- <link> - використовується для застосування CSS на сторінці.
- <script> - використовується для застосування JavaScript коду на сторінці.

- <h1> - позначає заголовки різних рівнів, цифра позначає рівень заголовка, від 1 до 6.
- <p> - створює абзац в тексті.
- <img> - вставляє зображення на сторінку.
- <a> - створює гіперпосилання.
- та <ol> - створює несортований та сортований список відповідно.
- <table> - створює таблицю.
- <select> - створює випадаючий список.

Всередині тегів прописуються атрибути, вони використовуються для надання додаткової інформації про HTML елементи, змінюючи їх поведінку або зовнішній вигляд. Найпоширеніші атрибути це:

- id - визначає унікальний ідентифікатор елемента.
- class - визначає один або декілька класів CSS, які будуть застосовуються до елемента.
- href - визначає посилання для тегів <a> та <link>.
- src - визначає місцезнаходження файлу для тегів <img> та <script>.
- alt - Визначає альтернативний текст для зображень, який буде відображатися, якщо зображення не завантажиться або буде недоступне для людей з порушеннями зору.

Дані теги та атрибути надають чіткі інструкції web-браузерам, як візуалізувати текст, зображення, посилання, та інші елементи сторінки для користувачів. Але HTML - це лише основа сторінок, для їх дизайну та інтерактивності необхідні CSS та JavaScript.

3.2.2 CSS

CSS - це мова опису зовнішнього вигляду сторінок web-застосунку, вона використовується для стилізації HTML-елементів. CSS робить сторінки візуально привабливими та зручними для користувачів. CSS складається з селекторів та декларацій. Селектори визначають слугують для точного вибору елементи HTML,

до яких необхідно застосувати стиль. Для точного вибору елементів HTML використовують наступні селектори:

- Селектори елементів - визначають елементи за їхнім HTML тегом (наприклад: p).
- Селектори класів - визначають елементи за CSS класами (наприклад: .product_card).
- Селектори ID - визначають елементи за id (наприклад: #nav_element).
- Псевдокласи - визначають які стилі мають застосовуватись до елементів за їхнім станом або поведінкою (наприклад: :hover, якщо на елемент наведено курсор).
- Дочірні селектори - визначають елементи, що знаходяться всередині інших елементів (наприклад, div a).

Декларації визначають властивості стилю, які будуть застосовуватися до елементів. Існує дуже велика кількість властивостей, які можна розділити на наступні категорії:

- Колір - колір тексту, фону, кордонів (color, background-color, border-color).
- Шрифт - назва шрифта, розмір шрифта, формат шрифта (font-family, font-size, font-weight, line-height).
- Розміри - ширина, довжина, внутрішні та зовнішні відступи елементів (width, height, padding, margin).
- Розташування - розташування елементу на сторінці (display, position, top, left, right, bottom).
- Кордони - стиль кордонів елементу (border-width, border-style, border-color).
- Фон - стиль фону елементу (background-image, background-repeat, background-position, background-size).
- Функціональні - видимість елементи на сторінці, прозорість, переміщення(text-decoration, visibility, opacity, transition).

Тип значення властивості залежить від типу властивості. Одні властивості приймають текстові значення, наприклад, color: red, інші - числові значення, наприклад, font-size: 24px, а деякі можуть приймати одночасно декілька типів, наприклад, border: 2px solid black.

HTML елементи та CSS властивості можуть бути додані, або змінені за допомогою JavaScript коду.

3.2.3 JavaScript

Функціональні можливості створюються за допомогою JavaScript, а точніше його надбудови TypeScript. Головні елементи кожного файлу функціоналу це:

- NgOnInit - метод, що викликається після того, як компонент ініціалізований і всі його дочірні компоненти створені. Це гарантує, що код ініціалізації буде виконаний після того, як усі залежності компонента будуть доступні і перш ніж він стане видимим користувачеві.
- Імпорт - механізм, який імпортує модулі, сервіси та інші ресурси, необхідні для роботи web-застосунку. Імпорти використовуються для підключення внутрішніх модулів, або сторонніх бібліотек, таких як @angular/core, @angular/forms.
- Маршрутизація - це механізм, що надає можливість користувачу переходити між різними сторінками web-застосунку за допомогою URL-адрес. Маршрути описують відповідність між URL-адресами та компонентами. Angular інтерпретує URL-адреси, активує відповідні компоненти та оновлює інтерфейс користувача.
- Конструктор - це функція в класі, яка викликається автоматично, коли створюється новий екземпляр даного класу. Конструктор ініціалізує властивості, необхідні для відображення контенту на сторінці. Дані конструктор отримує за допомогою сервісів

- Сервіси — це класи, які надають доступ до даних, отримуючи дані через API та обробляючи їх перед наданням компонентам.

Для кожної сторінки було розроблено HTML каркас, CSS дизайн та JavaScript код функціоналу.

3.3 Зв’язок між серверною та клієнтською частинами web-застосунку

Для зв'язку клієнта та сервера використовується REST API.

API - це набір правил якими користуються різні програми для обміну даними.

REST - це стиль архітектури API, він ґрунтується на протоколі HTTP.

HTTP - це транспортний протокол для передачі даних, на основі якого побудована всесвітня мережа Інтернет. HTTP спочатку був призначений для передачі гіпертекстових документів, але на теперішній час за його допомогою передають будь-які дані. Відповідно до специфікації OSI, HTTP є протоколом прикладного рівня.

При певних діях користувача клієнт робить HTTP запит до сервера для виконання відповідних дій (додавання, оновлення, видалення). Сервер в свою чергу робить запит до бази даних для отримання даних необхідних для виконання операції. Після отримання даних з бази даних та проведення операції, сервер в надсилає HTTP відповідь клієнту.

HTTP запит складається з трьох частин:

1. Стартовий рядок: містить метод запиту, URI запитуваного ресурсу та версію HTTP протоколу. Ресурсом можуть бути будь-які дані, наприклад текст, числа, зображення, набір інших ресурсів. Метод визначає тип запиту, який надсилає клієнт. П’ять основних методів, пов’язані з REST API:

- GET: Отримання ресурсу.
- POST: Створення нового ресурсу.
- PUT: Оновлення ресурсу.
- PATCH: Частове змінення ресурсу.

- DELETE: Видалення ресурсу.

2. Заголовок: в заголовку надається додаткова інформація про запит або відповідь. Наприклад: який сервер обслуговує запит, який браузер використовує клієнт, який тип даних, що надсилається або отримується.

3. Тіло: містить дані, які передаються, у REST API зазвичай використовують JSON. У GET та DELETE запиті тіло зазвичай порожнє, а POST, PUT, PATCH запити мають містити дані що мають бути завантажені.

HTTP відповідь складається з наступних частин:

1. Статусний рядок: містить код статусу відповіді HTTP, який вказує на результат запиту. Статус-код складається з трьох цифр, перша з них відповідає за:

- 100: запит отримано і обробляється, але результату ще немає.
- 200: запит було успішно оброблено і ресурс доступний.
- 300: клієнт буде перенаправлений до іншого ресурсу для отримання інформації.
- 400: запиті від клієнта є помилка і його неможливо обробити.
- 500: на сервері сталася помилка, що робить неможливим обробку запиту.

2. Заголовок: надає метадані про відповідь, такі як тип контенту, дата та час, довжина контенту. в заголовку надається додаткова інформація про запит або відповідь.

3. Тіло: містить запитаний ресурс або дані, що були запитані для обробки клієнтом.

Стиль REST має наступні принципи:

- Основою будь-якого REST web-застосунку є єдиний інтерфейс. Сервер передає, а клієнт відправляє дані у єдиному форматі.

  Єдиний інтерфейс накладає чотири архітектурні обмеження:

  1. Запити мають ідентифікувати ресурси, це відбувається за допомогою URI.

2. Клієнт повинен передати достатньо даних у запиті, щоб додати, змінити, або видалити ресурс. Сервер виконує цю умову, відправляючи метадані, які додатково описують ресурс.
3. Клієнт має отримати інформацію про подальшу обробку даних, сервер виконує це, відправляючи метадані того, як клієнт може використовувати їх.
4. Клієнт має отримати інформацію про всі пов'язані ресурси, необхідні виконання операції, сервер виконує це, надсилаючи гіперпосилання у відповіді.

- Сервер виконує кожен запит клієнта незалежно від усіх попередніх запитів.Це передбачає, що сервер зможе щоразу повністю зрозуміти та виконувати запит.
- REST web-застосунки можуть зберігати деякі відповіді на клієнтській частині у вигляді кешу для скорочення часу відповіді сервера..
- Клієнт може підключатися до інших авторизованих посередників між клієнтом та сервером і отримувати відповіді від сервера. Сервери також можуть надсилати запити іншим серверам.

Ці принципи забезпечують API незалежність клієнта та сервера, що дозволяє легко впроваджувати новий функціонал. Відсутність збереження стану зменшує навантаження сервера, адже йому не потрібно зберігати інформацію про попередні запити клієнта. Кешування частково, або повністю усуває деякі взаємодії між клієнтом та сервером, що збільшує швидкодію web-застосунку та забезпечує масштабованість. REST API дозволяє використовувати різні мови для розробки клієнта та сервера, або змінити базову технологію будь-якої частини, залишаючи структуру API єдиною.

3.4 Тестування web-застосунку

Для перевірки виконання функціональних та нефункціональних вимог до web-застосунку, було проведено модульне та інтеграційне тестування.

3.4.1 Модульне тестування

Модульне тестування - це рівень тестування на якому перевіряється функціонування окремих модулів web-застосунку.

Для модульного тестування серверної частини використовуються unit-тести, для їх написання використовують фреймворк TestNG [23].

TestNG - це фреймворк для автоматизованого тестування мовою Java, натхненний фреймворками JUnit та NUnit. Він пропонує можливості для тестування різних видів програмного забезпечення.

TestNG містить набір тверджень для написання перевірок:

- assertEquals/assertNotEquals - перевірка двох значень на однаковість/неоднаковість
- assertTrue/assertFalse - перевірка значення на істинність/хибність
- assertNull/assertNotNull - перевірка значення на рівність/не рівність нулю

Керування тестуванням відбувається за допомогою анотацій. Найголовніша анотація - @Test, нею позначають всі методи тестів. Процес тестування розбивається на етапи, для контролю послідовності виконання тестів використовують наступні анотації:

- @BeforeSuite/@AfterSuite - метод буде виконуватись до/після виконання всіх інших методів.
- @BeforeTest/@AfterTest - метод буде виконуватись до/після виконання всіх методів позначених анотацією @Test.
- @BeforeClass/@AfterClass - метод буде виконуватись до/після всіх інших методів в цьому класі.
- @BeforeGroups/@AfterGroups - за допомогою атрибуту group в аннотації @Test можна об’єднувати тести в групи. Метод буде виконуватись до/після всіх методів об’єднаних в групу.
- @BeforeMethod/@AfterMethod - метод буде виконуватись до/після кожного методу позначеного анотацією @Test.

Після успішного модульного тестування проводиться інтеграційне тестування.

3.4.2 Інтеграційне тестування

Інтеграційне тестування - це рівень тестування на якому перевіряється те, як різні компоненти системи взаємодіють один з одним, в даному випадку як взаємодіють клієнт та сервер.

Для інтеграційного тестування використовується платформа для тестування API Postman.

Postman - це платформа для розробки, використання, документування та тестування API. Він використовується розробниками, тестувальниками та менеджерами для налаштування та обміну запитами API між учасниками команди [24].

Рис. 3.15 Інтерфейс Postman

На рис. 3.15 зображено інтерфейс Postman та пронумеровано необхідні для тестування поля:

1. URL запиту: для кожного запиту, потрібна URL-адреса кінцевої точки API. Кожна операція, яку можна виконати, зазвичай пов’язана з кінцевою точкою.
2. Метод запиту: з випадаючого списку необхідно вибрати метод запиту.\
3. Заголовок: при необхідності, можна додати заголовок.
4. Тіло запиту: якщо в запиті мають передаватись дані, необхідно заповнити тіло в форматі JSON.
5. Кнопка відправки запиту: після заповнення всіх необхідних полів, натискається кнопка “Send”.
6. Статус: тут можна побачити статус відповіді.
7. Тіло відповіді: якщо у відповіді передаються дані, вони будуть відображені в цьому полі в форматі JSON.

Таким чином перевіряється правильність відправки та отримання даних між клієнтом і сервером.

ВИСНОВКИ

У результаті виконання дипломної роботи розроблено web-застосунок для продажів крафтового шоколаду. Актуальність даної роботи полягає в тому, що наразі web-застосунок є одним за найбільш ефективних способів ведення малого бізнесу, який в свою чергу є неймовірно важливою частиною економіки.

1. Проведено аналіз предметної галузі, згідно теми дипломної роботи та визначено переваги web-застосунку, як спосіб ведення малого бізнесу продажів крафтового шоколаду. Проаналізовано три існуючих web-застосунки для продажів крафтового шоколаду: Львівська майстерня шоколаду, WANDER, Spell Chocolate, виділено їхні основні переваги та недоліки.
2. На основі аналізу предметної галузі та аналогів, визначено технічне завдання та розроблено функціональні та нефункціональні вимоги до web-застосунку. Ключевими них є:
    - Додавання, редагування та видалення продуктів.
    - Відображення асортименту
    - Персональний кабінет користувача.
    - Пошук товару.
    - Зберігання продуктів в кошику.
    - Можливість створення власних шоколадних плиток.
    - Сумісність з різними браузерами.
    - Масштабованість.
    - Захищеність персональних даних.
    - Швидке завантаження сторінок.
3. Проаналізовано існуючі засоби розробки web-застосунків та обрано наступні: фреймворки Spring та Angular, база даних PostgreSQL, середовища розробки IntelliJ IDEA та WebStorm, Git та GitHub для контролю версій.

4. Спроєктовано клієнт-серверну архітектуру web-застосунку, що ділить застосунок на три частини: серверну, клієнтську та базу даних. Архітектура web-застосунку була спроектована з урахуванням всіх функціональних та нефункціональних вимог.
5. Розроблено web-застосунок для продажів крафтового шоколаду із застосуванням патерну Controller-Service-Repository для розробки серверної частини, модульного підходу для розробки клієнтської частини та REST API для зв’язку між серверною та клієнтською частинами.
6. Проведено модульне на інтеграцій тестування web-застосунку та перевірено виконання функціональних та нефункціональних вимог.

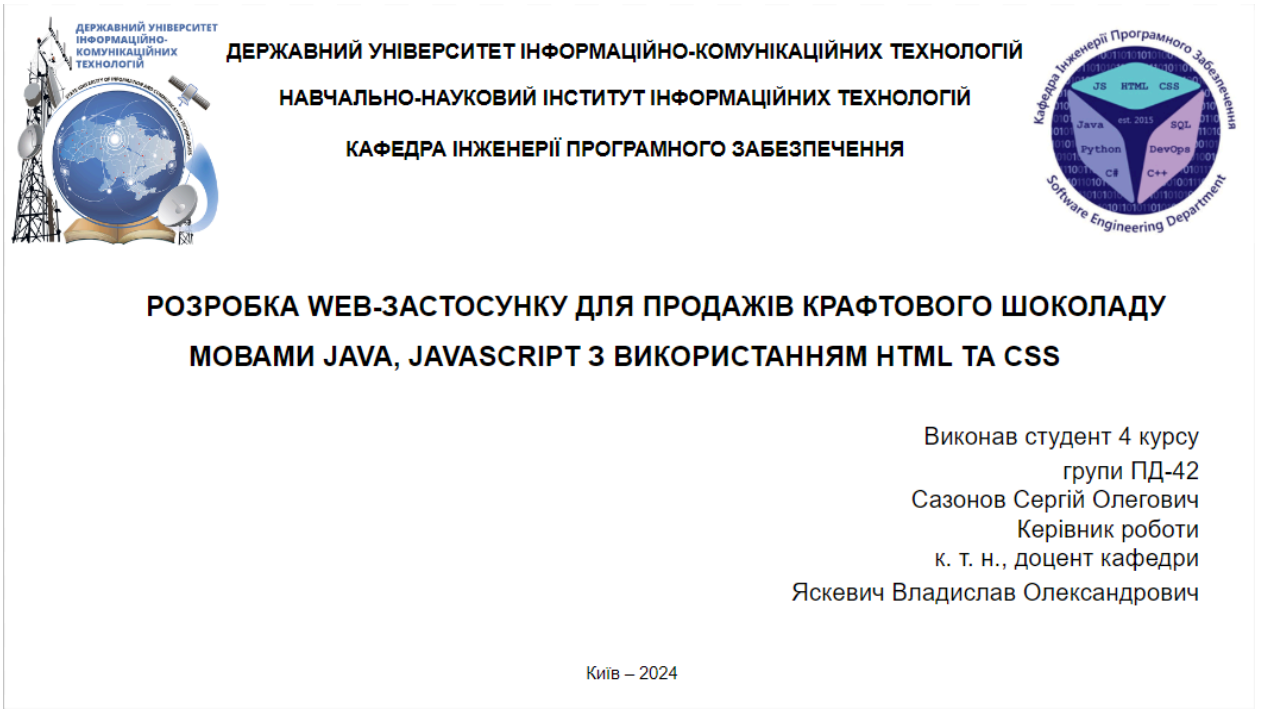
ПЕРЕЛІК ПОСИЛАНЬ

1. Miroshnyk R., Prokopieva U. DEVELOPMENT OF SMALL AND MEDIUM-SIZES BUSINESS IN UKRAINE: PROBLEMS AND POSSIBILITIES. *Journal of Lviv Polytechnic National University. Series of Economics and Management Issues*. 2020. Vol. 4, No. 1. P. 63–71. URL: https://doi.org/10.23939/semi2020.01.063
2. Козак А. Р., Гевлич Л. Л. РОЛЬ ТА МІСЦЕ МАЛОГО БІЗНЕСУ В ЕКОНОМІЦІ УКРАЇНИ ТА СВІТУ. *Вісник студентського наукового товариства ДонНУ імені Василя Стуса*. 2021. Том 2, № 13. с. 236-240. URL: https://jvestnik-sss.donnu.edu.ua/article/view/11281
3. У 2024 році майже 80% МСБ планують розширювати бізнес та залучати кошти у розвиток. *European Business Association*. URL: https://eba.com.ua/u-2024-rotsi-majzhe-80-msb-planuyut-rozshyryuvaty-biznes-ta-zaluchaty-koshty-u-rozvytok/ (дата звернення 10.04.2024).
4. Сазонов С. О., Яскевич В. О. Web-застосунок як інструмент для ведення малого бізнесу. *Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях»*, 24 квітня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С.449-450.
5. Сазонов С. О., Яскевич В. О. Розробка web-застосунку для продажів крафтового шоколаду. *Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях»*, 24 квітня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С.443-444.
6. Сайт магазину WANDER. URL: https://wander.boutique/ (дата звернення 17.04.2024).

7. Сайт магазину Spell. URL: https://spellchocolate.com/ (дата звернення 17.04.2024).
8. Сайт магазину Львівська Майстерня Шоколаду. URL: https://www.chocolate.lviv.ua/ (дата звернення 17.04.2024).
9. Official Spring Documentation. URL: https://docs.spring.io/spring-framework/reference/index.html (дата звернення 20.04.2024).
10. Official Java Documentation. URL: https://docs.oracle.com/en/java/ (дата звернення 17.04.2024).
11. Official Angular Documentation. URL: https://angular.io/docs (дата звернення 20.04.2024).
12. Official TypeScript Documentation. URL: https://www.typescriptlang.org/docs/ (дата звернення 20.04.2024).
13. JavaScript Documentation. URL: https://devdocs.io/javascript/ (дата звернення 20.04.2024).
14. Official PostgreSQL Documentation. URL: https://www.postgresql.org/docs (дата звернення 20.04.2024).
15. Official IntelliJ IDEA Documentation. URL: https://www.jetbrains.com/help/idea/getting-started.html (дата звернення 20.04.2024).
16. Official WebStorm Documentation. URL: https://www.jetbrains.com/help/webstorm/getting-started-with-webstorm.html (дата звернення 20.04.2024).
17. Official Git Documentation. URL: https://git-scm.com/docs (дата звернення 20.04.2024).
18. Official GitHub Documentation. URL: https://docs.github.com (дата звернення 20.04.2024).
19. Official Spring Data JPA Documentation. URL: https://docs.spring.io/spring-data/jpa/reference/index.html (дата звернення 21.04.2024).

20. Official Java Persistence API URL: https://docs.oracle.com/javaee/7/tutorial/persistence-intro.htm (дата звернення 21.04.2024).
21. Official Spring Security Documentation. URL: https://docs.spring.io/spring-security/reference/index.html (дата звернення 21.04.2024).
22. Official JWT Documentation. URL: https://jwt.io/introduction (дата звернення 21.04.2024).
23. Official TestNG Documentation. URL: https://testng.org/ (дата звернення 23.04.2024).
24. Official Postman Documentation. URL: https://learning.postman.com/docs/introduction/overview/ (дата звернення 26.04.2024).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

РОЗРОБКА WEB-ЗАСТОСУНКУ ДЛЯ ПРОДАЖІВ КРАФТОВОГО ШОКОЛАДУ МОВАМИ JAVA, JAVASCRIPT З ВИКОРИСТАННЯМ HTML ТА CSS

Виконав студент 4 курсу
групи ПД-42
Сазонов Сергій Олегович
Керівник роботи
к. т. н., доцент кафедри
Яскевич Владислав Олександрович

Київ – 2024

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – підтримка процесу продажів крафтового шоколаду шляхом створення web-застосунку мовами Java, JavaScript та HTML/CSS.
- **Об'єкт дослідження -** процес продажу крафтового шоколаду.
- **Предмет дослідження -** web-застосунок для продажів крафтового шоколаду.

2

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати існуючі аналоги web-застосунків для продажів крафтового шоколаду, визначити переваги та недоліки.
2. Розробити функціональні та нефункціональні вимоги до web-застосунку.
3. Дослідити та вибрати засоби розробки web-застосунку, розробити модель архітектури web-застосунку та його дизайн.
4. Розробити web-застосунок на основі обраних інструментів та визначених вимог.
5. Провести тестування web-застосунку.

3

АНАЛІЗ АНАЛОГІВ

Критерії порівняння	Львівська майстерня шоколаду	WANDER	Spell Chocolate	DreamChocolatery
Можливість створення кастомної шоколадки	-	-	-	+
Можливість створення персонального кабінету	+	-	+	+
Каталог доступний на головній сторінці	-	-	-	+
Швидкість завантаження головної сторінки	2140 ms	903 ms	966 ms	900 ms
Пошук товару	+	-	+	+

4

ВИМОГИ ДО ЗАСТОСУНКУ

Функціональні вимоги:

1. Додавання, редагування та видалення продуктів.
2. Категоризація для зручного пошуку та навігації по каталогу продукції.
3. Персональний кабінет користувача.
4. Пошук товару.
5. Зберігання продуктів в кошику.
6. Можливість створення власних шоколадних плиток.

Нефункціональні вимоги:

1. Мінімалістичний дизайн.
2. Сумісність з різними браузерами (Google Chrome, FireFox).
3. Маштабованість.
4. Захищеність персональних даних.
5. Швидке завантаження сторінок (<1500 мс).

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

spring

ANGULAR

IJ

PostgreSQL

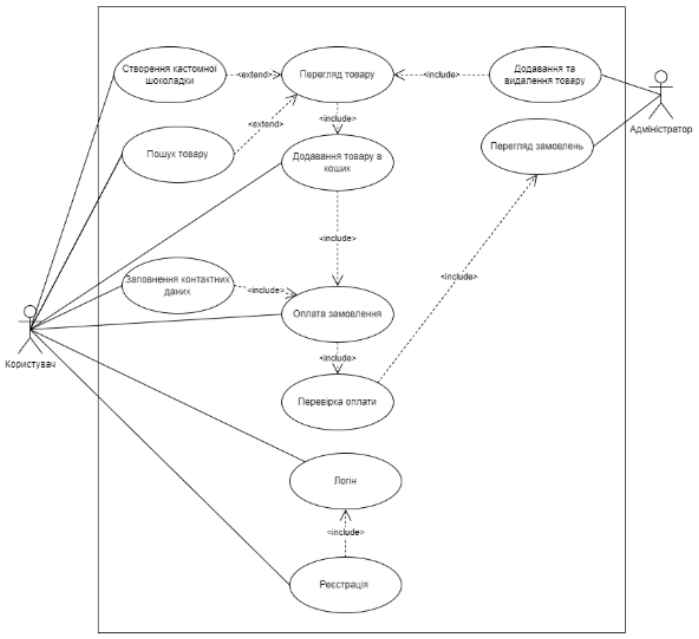
WS

git

GitHub

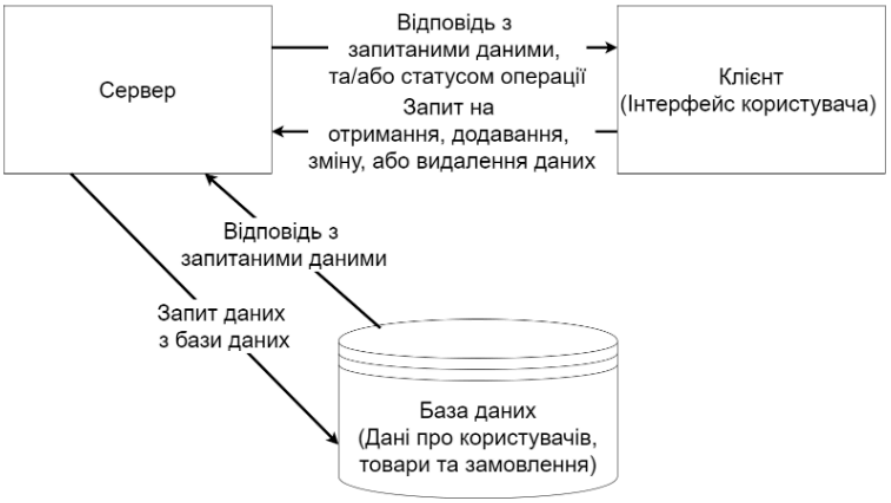
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



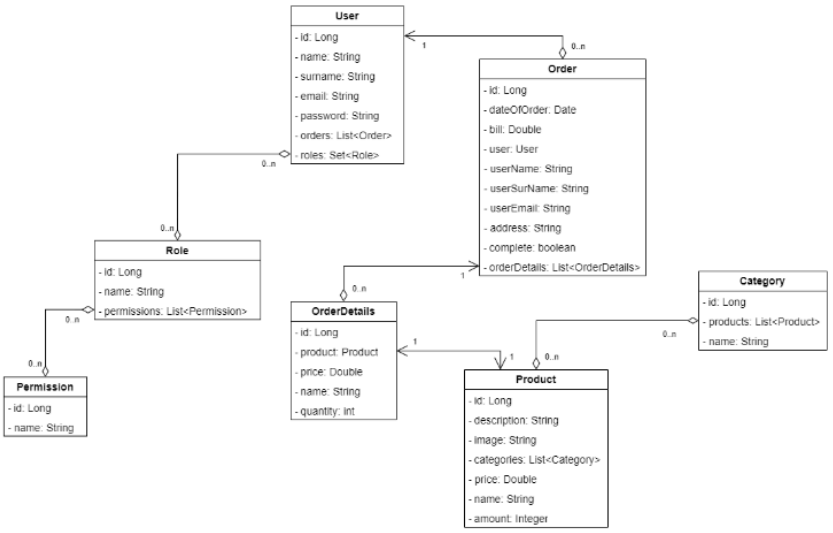
7

СХЕМА КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ WEB-ЗАСТОСУНКУ



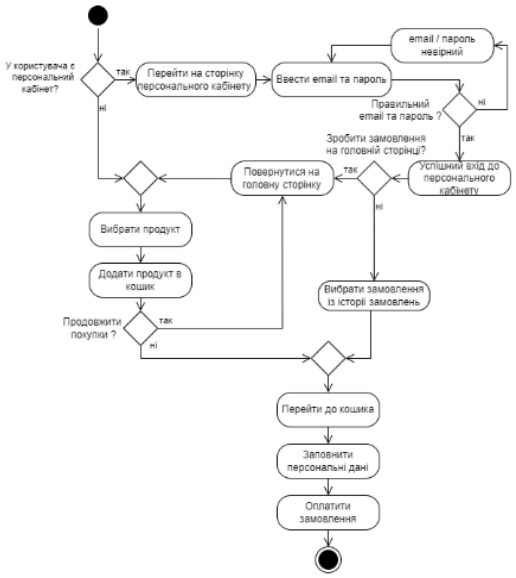
8

ДІАГРАМА КЛАСІВ



9

ДІАГРАМА ДІЯЛЬНОСТІ
(Дії для оформлення замовлення)

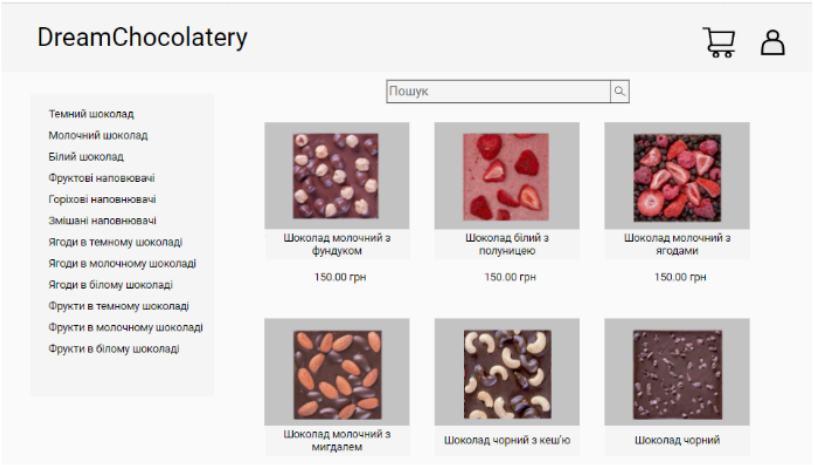


10

МАПА WEB-ЗАСТОСУНКУ

11

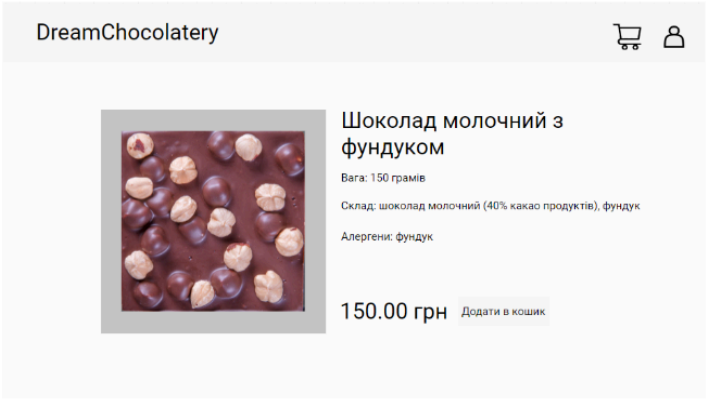
ЕКРАННІ ФОРМИ



Головний екран web-застосунку

12

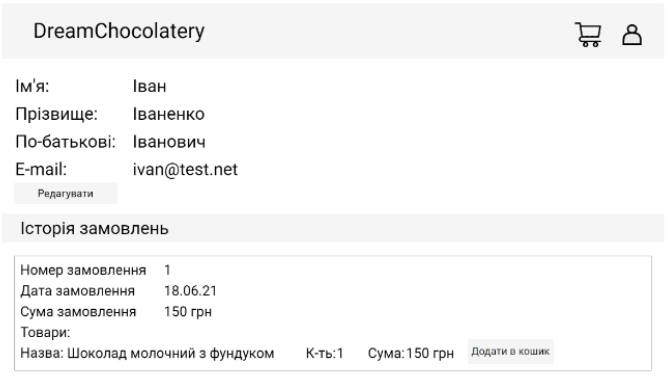
ЕКРАННІ ФОРМИ



Сторінка продукту

13

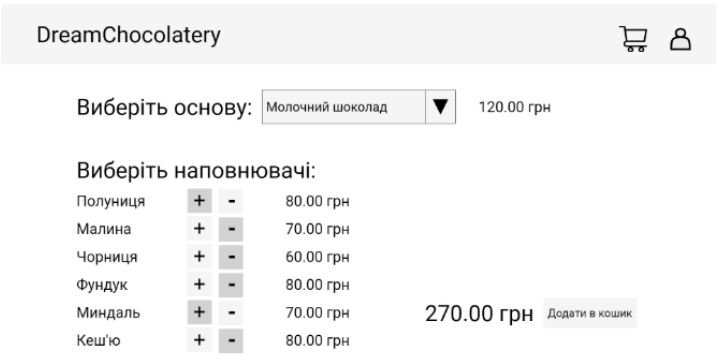
ЕКРАННІ ФОРМИ



Сторінка персонального кабінету користувача

14

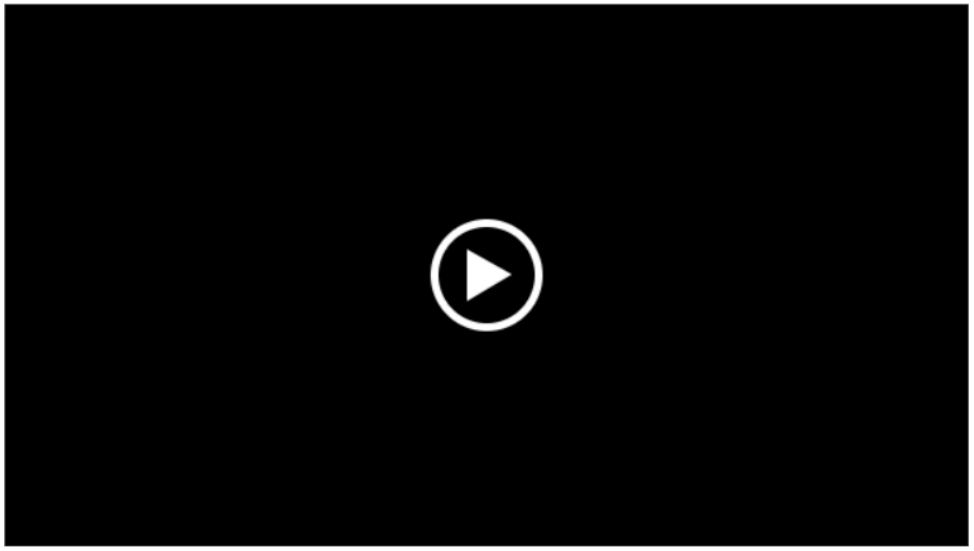
ЕКРАННІ ФОРМИ



Сторінка створення персональної шоколадки

15

ДЕМОНСТРАЦІЯ РОБОТИ WEB-ЗАСТОСУНКУ



16

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Сазонов С. О., Яскевич В. О. Розробка web-застосунку для продажів крафтового шоколаду. *Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях»*, 24 квітня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С.443-444.
2. Сазонов С. О., Яскевич В. О. Web-застосунок як інструмент для ведення малого бізнесу. *Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях»*, 24 квітня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С.449-450.

17

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
package com.dream_chocolatery.entity;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.*;
import
javax.validation.constraints.Positive;
import java.util.Date;
import java.util.List;

@Data
@Entity
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "order")
public class Order {

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;
    private boolean complete;

    @ManyToOne(fetch =
FetchType.LAZY, cascade =
CascadeType.ALL)
    @JoinColumn(name = "user_id" ,
insertable = false, updatable = false)
    private User user;
    private String userName;
    private String userSurName;
    private String userEmail;

    private String address;

    @Positive
    private Double bill;
    private Date dateOfOrder;

    @PrePersist
    private void onCreate(){
       dateOfOrder = new Date();
    }

    @OneToMany(cascade =
CascadeType.ALL)
    private List<OrderDetails>
orderDetails;
}
```

```
package
com.dream_chocolatery.controller;

import
com.dream_chocolatery.dto.request.order.Creat
eAuthOrderRequest;
import
com.dream_chocolatery.dto.request.order.Creat
eUnauthOrderRequest;
import
com.dream_chocolatery.dto.response.order.Ord
erResponse;
import
com.dream_chocolatery.entity.Order;
import
com.dream_chocolatery.entity.OrderDetails;
import
com.dream_chocolatery.mapper.OrderMapper;
import
com.dream_chocolatery.service.OrderService;
import
com.dream_chocolatery.service.UserService;
import
lombok.RequiredArgsConstructor;
import
org.springframework.http.HttpStatus;
import
org.springframework.http.ResponseEntity;
import
org.springframework.security.core.annotation.
AuthenticationPrincipal;
import
org.springframework.security.core.userdetails.
UserDetails;
import
org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.util.List;
import java.util.stream.Collectors;
```

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/order")
public class OrderController {

    private final OrderService
orderService;
    private final UserService
userService;
    private final OrderMapper
orderMapper;

    @GetMapping("/getAll")
    public
ResponseEntity<List<OrderResponse>>
getAllOrders(){
        return new
ResponseEntity<>(orderMapper.toDtos(orderS
ervice.getAllOrders()), HttpStatus.OK);
    }

    @GetMapping("/getAllByUser")
    public
ResponseEntity<List<OrderResponse>>
getAllOrdersByUser(@Valid
@AuthenticationPrincipal UserDetails user){
        return new
ResponseEntity<>(orderMapper.toDtos(orderS
ervice.getAllOrdersByUser(userService.getUse
rByEmail(user.getUsername()))),
HttpStatus.OK);
    }

    @PostMapping("/create/authOrder")
    public
ResponseEntity<OrderResponse>
createNewAuthOrder(@Valid @RequestBody
CreateAuthOrderRequest request,
@AuthenticationPrincipal UserDetails user){
        Order order =
orderMapper.fromDto(request);
        List<OrderDetails>
orderDetailsList =
request.getProducts().stream()

.map(orderMapper::orderDetailsFromDto)
            .collect(Collectors.toList());

order.setOrderDetails(orderDetailsList);
        return new
ResponseEntity<>(orderMapper.toDto(orderSe
rvice.createAuthOrder(order,
userService.getUserByEmail(user.getUsername
()))), HttpStatus.OK);
    }


@PostMapping("/create/unauthOrder")
    public
ResponseEntity<OrderResponse>
createNewUnAuthOrder(@Valid
@RequestBody CreateUnauthOrderRequest
request){
        Order order =
orderMapper.fromDto(request);
        List<OrderDetails>
orderDetailsList =
request.getProducts().stream()

.map(orderMapper::orderDetailsFromDto)
            .collect(Collectors.toList());

order.setOrderDetails(orderDetailsList);
        return new
ResponseEntity<>(orderMapper.toDto(orderSe
rvice.createUnAuthOrder(order)),
HttpStatus.OK);
    }

    @PatchMapping("/update")
    public ResponseEntity<Void>
endOrder(@Valid @RequestParam Long
orderId){
        orderService.endOrder(orderId);
        return ResponseEntity.ok().build();
    }
}



package
com..dream_chocolatery.service;

import com.dream_chocolatery.entity.*;
import
com.dream_chocolatery.repository.OrderReposi
tory;
import
lombok.RequiredArgsConstructor;
import
org.springframework.stereotype.Service;
import
org.springframework.transaction.annotation.Tra
nsactional;
```

```
import java.util.List;

@Service
@RequiredArgsConstructor
public class OrderService {
   private final ProductService
productService;
   private final OrderRepository
orderRepository;

   @Transactional(readOnly = true)
   public List<Order> getAllOrders() {
      return orderRepository.findAll();
   }

   @Transactional(readOnly = true)
   public List<Order>
getAllOrdersByUser(User user) {
      return
orderRepository.findAllByUser(user);
   }

   @Transactional
   public Order createAuthOrder(Order
order, User user) {
      order.setUser(user);

order.setUserName(user.getName());

order.setUserSurName(user.getSurname());

order.setUserEmail(user.getEmail());
      List<OrderDetails> orderDetails =
order.getOrderDetails();
      Double bill = 0.0;
      for (OrderDetails orderDetail :
orderDetails) {
         Product product =
orderDetail.getProduct();

orderDetail.setName(product.getName());

orderDetail.setPrice(product.getPrice());
         bill += product.getPrice() *
orderDetail.getQuantity();

productService.reduceQuantity(orderDetail.get
Product(), orderDetail.getQuantity());
      }
      order.setBill(bill);
      return orderRepository.save(order);
   }

   @Transactional
   public Order
createUnAuthOrder(Order order) {
      List<OrderDetails> orderDetails =
order.getOrderDetails();
      Double bill = 0.0;
      for (OrderDetails orderDetail :
orderDetails) {
         Product product =
orderDetail.getProduct();

orderDetail.setName(product.getName());

orderDetail.setPrice(product.getPrice());
         bill += product.getPrice() *
orderDetail.getQuantity();

productService.reduceQuantity(orderDetail.get
Product(), orderDetail.getQuantity());
      }
      order.setBill(bill);
      return orderRepository.save(order);
   }

   @Transactional
   public void endOrder(Long id) {
      Order currentOrder =
orderRepository.getById(id);
      if (!currentOrder.isComplete()) {
         currentOrder.setComplete(true);
      }

orderRepository.save(currentOrder);
   }

}



package
com.dream_chocolatery.repository;

import
com.dream_chocolatery.entity.Order;
import
com.dream_chocolatery.entity.User;
import
org.springframework.data.jpa.repository.JpaRe
pository;
```

```
        import java.util.List;

        public interface OrderRepository
extends JpaRepository<Order, Long> {
            List<Order> findAllByUser(User
user);
        }
```

```
package com.dream_chocolatery.mapper;

import
com.dream_chocolatery.dto.request.order.Creat
eAuthOrderRequest;
import
com.dream_chocolatery.dto.request.order.Creat
eUnauthOrderRequest;
import
com.dream_chocolatery.dto.request.order.Upda
teOrderRequest;
import
com.dream_chocolatery.dto.response.order.Ord
erDetailsResponse;
import
com.dream_chocolatery.dto.response.order.Ord
erResponse;
import com.dream_chocolatery.entity.Order;
import
com.dream_chocolatery.entity.OrderDetails;
import org.mapstruct.Mapper;
import org.mapstruct.Mapping;

import java.util.List;

@Mapper
public interface OrderMapper {
   Order fromDto(CreateAuthOrderRequest
dto);
   Order fromDto(CreateUnauthOrderRequest
dto);
   Order fromDto(UpdateOrderRequest dto);
   @Mapping(source = "orderDetails", target =
"products")
   OrderResponse toDto(Order entity);
   List<OrderResponse> toDtos(List<Order>
orders);
   OrderDetails
orderDetailsFromDto(OrderDetails dto);
   @Mapping(source = "id", target =
"products")
   @Mapping(source = "quantity", target =
"quantity")
   @Mapping(source = "price", target =
"price")
   @Mapping(source = "name", target =
"name")
   OrderDetailsResponse
orderDetailsResponseToDto(OrderDetails
orderDetails);
}
```

```
package
com.dream_chocolatery.dto.request.order;

import
com.dream_chocolatery.entity.OrderDetails;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.NotNull;
import java.util.List;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class CreateAuthOrderRequest {
   @NotNull(message = "address cannot be
null")
   private String address;

   @NotEmpty(message = "Products cannot be
null")
   private List<OrderDetails> products;
   private boolean complete;
}
```

```
package
com.dream_chocolatery.dto.request.order;

import
com.dream_chocolatery.entity.OrderDetails;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
```

```
import lombok.NoArgsConstructor;

import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.NotNull;
import java.util.List;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class CreateUnauthOrderRequest {
   @NotNull(message = "name cannot be
null")
   private String userName;

   @NotNull(message = "name cannot be
null")
   private String userSurname;

   @NotNull(message = "name cannot be
null")
   private String userEmail;

   @NotNull(message = "address cannot be
null")
   private String address;

   @NotEmpty(message = "Products cannot be
null")
   private List<OrderDetails> products;
   private boolean complete;
}



package
com.dream_chocolatery.dto.response.order;

import
com.fasterxml.jackson.annotation.JsonFormat;
import
com.dream_chocolatery.entity.OrderDetails;
import com.dream_chocolatery.entity.User;
import lombok.Data;

import javax.persistence.*;
import javax.validation.constraints.Positive;
import java.util.Date;
import java.util.List;

@Data
public class OrderResponse {
   private Long id;
   private boolean complete;
   private Long userId;
   private String userName;
   private String userSurName;
   private String userEmail;
   private String address;

   @JsonFormat(shape =
JsonFormat.Shape.STRING, pattern =
"dd-MM-yyyy HH:mm",
         timezone = "Europe/Kiev")
   private Date dateOfOrder;

   private Double bill;
   private List<OrderDetailsResponse>
products;
}

package com.dream_chocolatery.entity;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.*;
import java.util.List;

@Data
@Entity
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class Category {
   @Id
   @GeneratedValue(strategy =
GenerationType.IDENTITY)
   private Long id;
   private String name;

   @ManyToMany(fetch = FetchType.LAZY)
   @JoinTable(
         name = "products",
         joinColumns = @JoinColumn(
               name = "product_id",
referencedColumnName = "id"),
         inverseJoinColumns = @JoinColumn(
               name = "category_id",
referencedColumnName = "id"))
   private List<Product> products;
}
```

```
package com.dream_chocolatery.controller;

import
com.dream_chocolatery.dto.request.category.Cr
eateCategoryRequest;
import
com.dream_chocolatery.dto.request.category.U
pdateCategoryRequest;
import
com.dream_chocolatery.dto.response.category.
CategoryResponse;
import
com.dream_chocolatery.mapper.CategoryMapp
er;
import
com.dream_chocolatery.service.CategoryServic
e;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import
org.springframework.http.ResponseEntity;
import
org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.util.List;

@RestController
@RequiredArgsConstructor
@RequestMapping("/category")
public class CategoryController {

   private final CategoryService
categoryService;
   private final CategoryMapper
categoryMapper;

   @GetMapping("/getAll")
   public
ResponseEntity<List<CategoryResponse>>
findAllCategory(){
      return new
ResponseEntity<>(categoryMapper.toDtos(cate
goryService.getAllCategories()),
HttpStatus.OK);
   }

   @PostMapping("/create")
   public ResponseEntity<CategoryResponse>
createCategory(@Valid @RequestBody
CreateCategoryRequest
createCategoryRequest){
      return new
ResponseEntity<>(categoryMapper.toDto(cate
goryService.createCategory(categoryMapper.fr
omDto(createCategoryRequest))),
HttpStatus.OK);
   }

   @PatchMapping("/updatee")
   public ResponseEntity<CategoryResponse>
updateCategory(@Valid @RequestBody
UpdateCategoryRequest
updateCategoryRequest){
      return new
ResponseEntity<>(categoryMapper.toDto(cate
goryService.updateCategory(categoryMapper.fr
omDto(updateCategoryRequest))),
HttpStatus.OK);
   }

   @DeleteMapping("/{id}")
   public void deleteCategory(@PathVariable
Long id){
      categoryService.deleteCategory(id);
   }
}
```

```
package com.dream_chocolatery.service;

import com.dream_chocolatery.entity.Category;
import
com.dream_chocolatery.repository.CategoryRe
pository;
import lombok.RequiredArgsConstructor;
import
org.springframework.stereotype.Service;
import
org.springframework.transaction.annotation.Tra
nsactional;

import
javax.persistence.EntityNotFoundException;
import java.util.List;

@Service
@RequiredArgsConstructor
public class CategoryService {
```

```
    private final CategoryRepository
categoryRepository;

    @Transactional(readOnly = true)
    public List<Category> getAllCategories(){
        return categoryRepository.findAll();
    }

    @Transactional
    public Category createCategory(Category
category){
        return categoryRepository.save(category);
    }

    @Transactional
    public Category updateCategory(Category
category){
        if
(categoryRepository.existsById(category.getId(
))){
            return
categoryRepository.save(category);
        }
        else throw new
EntityNotFoundException("Category not
found");
    }

    @Transactional
    public void deleteCategory(Long id){
        if (categoryRepository.existsById(id)){
            categoryRepository.deleteById(id);
        }
        else throw new
EntityNotFoundException("Category not
found");
    }
}
```

```
package com.dream_chocolatery.repository;

import com.dream_chocolatery.entity.Category;
import
org.springframework.data.jpa.repository.JpaRe
pository;


public interface CategoryRepository extends
JpaRepository<Category, Long> {
}
```

```
package com.dream_chocolatery.mapper;

import
com.dream_chocolatery.dto.request.category.Cr
eateCategoryRequest;
import
com.dream_chocolatery.dto.request.category.U
pdateCategoryRequest;
import
com.dream_chocolatery.dto.response.category.
CategoryResponse;
import com.dream_chocolatery.entity.Category;
import org.mapstruct.Mapper;

import java.util.List;

@Mapper
public interface CategoryMapper {
    CategoryResponse toDto(Category entity);
    List<CategoryResponse>
toDtos(List<Category> entities);
    Category fromDto(CreateCategoryRequest
dto);
    Category fromDto(UpdateCategoryRequest
dto);

}
```

```
package
com.dream_chocolatery.dto.request.category;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.validation.constraints.NotEmpty;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class CreateCategoryRequest {
    @NotEmpty(message = "Name cannot be
null!")
    private String name;
}
```

```
package
com.dream_chocolatery.dto.request.category;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.NotNull;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class UpdateCategoryRequest {
   @NotNull(message = "Id cannot be null")
   private Long id;
   @NotEmpty(message = "Name cannot be
null")
   private String name;
}
```

```
package
com.dream_chocolatery.dto.response.category;

import lombok.Data;

@Data
public class CategoryResponse {
   private Long id;
   private String name;
}
```

```
package com.dream_chocolatery.entity;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.*;
import java.util.List;

@Data
@Entity
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class Product {
      @Id
      @GeneratedValue(strategy =
GenerationType.IDENTITY)
      private Long id;
      private String name;
      private String description;
      private Double price;
      private Integer amount;
      private String image;

      @ManyToMany(fetch =
FetchType.LAZY)
      @JoinTable(
            name = "product_categories",
            joinColumns = @JoinColumn(
                  name = "category_id",
referencedColumnName = "id"),
            inverseJoinColumns =
@JoinColumn(
                  name = "product_id",
referencedColumnName = "id"))
      private List<Category> categories;


      @Override
      public String toString() {
         return "Product{" +
               "id=" + id +
               ", name='" + name + '\'' +
               ", description='" + description + '\''
+
               ", price=" + price +
               ", amount=" + amount +
               ", image='" + image + '\'' +
               '}';
      }
   }
```

```
package com.dream_chocolatery.controller;

import
com.dream_chocolatery.dto.request.product.Cr
eateProductRequest;
```

```
import
com.dream_chocolatery.dto.request.product.Up
dateProductRequest;
import
com.dream_chocolatery.dto.response.product.P
roductResponse;
import
com.dream_chocolatery.mapper.ProductMappe
r;
import
com.dream_chocolatery.service.ProductService
;
import lombok.RequiredArgsConstructor;
import
org.springframework.data.domain.PageRequest
;
import org.springframework.http.HttpStatus;
import
org.springframework.http.ResponseEntity;
import
org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import javax.validation.constraints.NotNull;
import java.util.List;

@RestController
@RequiredArgsConstructor
@RequestMapping("/product")
public class ProductController {

   private final ProductService productService;
   private final ProductMapper productMapper;

   @GetMapping("/getAll")
   public
ResponseEntity<List<ProductResponse>>
getAllProducts(
         @RequestParam(required = false,
defaultValue = "0") int page,
         @RequestParam(required = false,
defaultValue = "6") int size){
      return new
ResponseEntity<>(productMapper.toDtos(prod
uctService.getAllProducts(PageRequest.of(pag
e, size))), HttpStatus.OK);
   }

   @GetMapping("/get/{id}")
   public ResponseEntity<ProductResponse>
getProduct(@NotNull @PathVariable Long
id){
      return new
ResponseEntity<>(productMapper.toDto(produ
ctService.getProduct(id)), HttpStatus.OK);
   }


@GetMapping("/get/category/{categoryId}")
   public
ResponseEntity<List<ProductResponse>>
getAllProductsByCategory(@NotNull
@PathVariable Long id,

@RequestParam(required = false, defaultValue
= "0") int page,

@RequestParam(required = false, defaultValue
= "6") int size){
      return new
ResponseEntity<>(productMapper.toDtos(prod
uctService.getAllProductsByCategory(id,
PageRequest.of(page, size))), HttpStatus.OK);
   }

   @PostMapping("/create")
   public ResponseEntity<ProductResponse>
createProduct(@Valid @RequestBody
CreateProductRequest createProductRequest){
      return new
ResponseEntity<>(productMapper.toDto(produ
ctService.createProduct(productMapper.fromDt
o(createProductRequest))), HttpStatus.OK);
   }

   @PatchMapping("/update")
   public ResponseEntity<ProductResponse>
updateProduct(@Valid @RequestBody
UpdateProductRequest
updateProductRequest){
      return new
ResponseEntity<>(productMapper.toDto(produ
ctService.updateProduct(productMapper.fromD
to(updateProductRequest))), HttpStatus.OK);
   }

   @DeleteMapping("/{id}")
   public void deleteProduct (@NotNull
@PathVariable Long id){
      productService.deleteProduct(id);
   }
}
```

```
package com.dream_chocolatery.service;

import com.dream_chocolatery.entity.Product;
import
com.dream_chocolatery.repository.ProductRep
ository;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Page;
import
org.springframework.data.domain.PageRequest
;
import
org.springframework.stereotype.Service;
import
org.springframework.transaction.annotation.Tra
nsactional;

import
javax.persistence.EntityNotFoundException;
import java.util.List;

@Service
@RequiredArgsConstructor
public class ProductService {

   private final ProductRepository
productRepository;

   @Transactional(readOnly = true)
   public List<Product>
getAllProducts(PageRequest pageRequest){
      Page<Product> page =
productRepository.findAll(pageRequest);
      return page.getContent();
   }

   @Transactional(readOnly = true)
   public Product getProduct(Long id){
      return
productRepository.findById(id).orElseThrow(()
->new EntityNotFoundException("Product not
found"));
   }

   @Transactional(readOnly = true)
   public List<Product>
getAllProductsByCategory(Long id,
PageRequest pageRequest){
      return
productRepository.findAllByCategory(id,
pageRequest);
   }

   @Transactional
   public void reduceQuantity(Product product,
int amount){
      if
(productRepository.existsById(product.getId())
){

product.setAmount(product.getAmount() -
amount);
         productRepository.save(product);
      }
      else throw new
EntityNotFoundException("Product not
found");
   }

   @Transactional
   public Product createProduct(Product
product){
      //List <Long> category =
product.getCategories();


      return productRepository.save(product);
   }

   @Transactional
   public Product updateProduct(Product
product){
      if
(productRepository.existsById(product.getId())
){
         return productRepository.save(product);
      }
      else throw new
EntityNotFoundException("Product not
found");

   }

   @Transactional
   public void deleteProduct(Long id){
      if (productRepository.existsById(id)){
         productRepository.deleteById(id);
      }
      else throw new
EntityNotFoundException("Product not
found");
   }
}
```

```
package com.dream_chocolatery.repository;

import com.dream_chocolatery.entity.Product;
import
org.springframework.data.domain.PageRequest
;
import
org.springframework.data.jpa.repository.JpaRe
pository;

import java.util.List;


public interface ProductRepository extends
JpaRepository<Product, Long> {
   List<Product> findAllByCategory(Long id,
PageRequest pageRequest );

}
```

```
package com.dream_chocolatery.mapper;

import
com.dream_chocolatery.dto.request.product.Cr
eateProductRequest;
import
com.dream_chocolatery.dto.request.product.Up
dateProductRequest;
import
com.dream_chocolatery.dto.response.product.P
roductResponse;
import com.dream_chocolatery.entity.Product;
import org.mapstruct.Mapper;

import java.util.List;
import java.util.stream.Collectors;

@Mapper
public interface ProductMapper {
   Product fromDto(CreateProductRequest
dto);
   Product fromDto(UpdateProductRequest
dto);
   ProductResponse toDto(Product product);
   default List<ProductResponse>
toDtos(List<Product> products){
      return
products.stream().map(this::toDto).collect(Coll
ectors.toList());
   }


}
```

```
package
com.dream_chocolatery.dto.request.product;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.validation.constraints.*;
import java.util.List;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class CreateProductRequest {
   @NotEmpty(message = "Name cannot be
null")
   private String name;

   @NotEmpty(message = "Description cannot
be null")
   private String description;

   @NotNull(message = "Price cannot be null")
   @DecimalMin("0.01")
   @Positive
   private Double price;

   @Positive
   private Integer amount;

   private String image;

   @NotEmpty(message = "Category cannot be
null")
   private List<Long> categories;
}
```

```
package
com.dream_chocolatery.dto.response.product;

import com.dream_chocolatery.entity.Category;
import lombok.Data;

import java.util.List;

@Data
public class ProductResponse {
    private Long id;
    private String name;
    private String description;
    private Double price;
    private Integer amount;
    private String image;
    private List<Category> categories;
}
```