ДИПЛОМНА РОБОТА
БАКАЛАВРА

ДРБ. ІП - 22.00.00.000 ПЗ

Група ІП-20-2

Ориняк Олексій

2024

Міністерство освіти і науки України
Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

УДК 004.4

ДИПЛОМНА РОБОТА БАКАЛАВРА

Тема: " Створення інтернет-платформи для комерційного майданчика "

(назва згідно з наказом ректора)

спеціальність – 121 "Інженерія програмного забезпечення"

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДРБ.ІП - 22.00.00.000 ПЗ

(позначення)

Консультанти:

з питань нормоконтролю

асист. /М.М. Піх/
(посада) (підпис) (дата)(розшифровка підпису)

Студент

ІП-20-2 /О.Р.Ориняк/
(шифр групи) (підпис) (дата) (розшифровка підпису)

Керівник роботи

д.т.н., проф. /В.М.Юрчишин/
(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту
Завідувач кафедри

к.т.н., доц. /В.В.Бандура/
(посада) (підпис) (дата) (розшифровка підпису)

2024
(рік)

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітньо-кваліфікаційний рівень бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ ____ ” _____2024 р.

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ БАКАЛАВРА СТУДЕНТОВІ

Ориняку Олексію Романовичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) “ Створення інтернет-платформи для комерційного майданчика”

керівник проекту (роботи) Юрчишин В.М., професор

затверджені наказом закладу вищої освіти від **“** ____ ” травня 2024 р. № _____

2. Строк подання студентом проекту (роботи) 17 червня 2024 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

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

1. Аналіз предметної області та методологій створення веб-додатків
2. Вибір методів і засобів створення веб-додатку для комерційного майданчика
3. Проектування інтернет-платформи для комерційного майданчика
4. Проектування структури веб-платформи
5. Розробка інтерфейсу та функціональності інтернет-платформи для комерційного майданчика

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Перелік основних типів веб-додатків (рис. 1.1)
2. Архітектура Single-Page Application (рис. 1.2)
3. Цикл Multiple-Page Application (рис. 1.3)
4. Типова структура web-додатка (рис. 1.4)
5. Основні види електронної комерції (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання **22 травня 2024 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Аналіз предметної області та методологій створення веб-додатків	24.05.2024	виконано
2	Вибір методів і засобів створення веб-додатку для комерційного майданчика	29.05.2024	виконано
3	Проектування інтернет-платформи для комерційного майданчика	30.05.2024	виконано
4	Проектування структури веб-платформи	04.06.2024	виконано
5	Розробка інтерфейсу та функціональності інтернет-платформи для комерційного майданчика	10.06.2024	виконано
6	Оформлення пояснювальної записки дипломної роботи завідувачем кафедри	17.06.2024	виконано

Студент – дипломник _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Ця робота присвячена розробці інтернет-платформи, спрямованої на створення комерційного майданчика для продажу товарів. У роботі розглянуті основні аспекти розробки та реалізації такої платформи, включаючи аналіз вимог користувачів, проектування інтерфейсу, розробку функціональності, впровадження системи оплати та забезпечення безпеки даних. Також розглянуто питання маркетингової стратегії та підтримки клієнтів. Результатом роботи є інноваційна інтернет-платформа, яка сприяє ефективним торговельним операціям, забезпечує зручність користувачів та підвищує конкурентоспроможність бізнесу на онлайн-ринку.

КЛЮЧОВІ СЛОВА: ЕЛЕКТРОННА КОМЕРЦІЯ, ПРОДАЖ ТОВАРІВ, МОДЕЛЮВАННЯ ВЕБ-РЕСУРСУ, ФРЕЙМВОРК, ВЕБ-РОЗРОБКА, ДІАГРАМА ПОСЛІДОВНОСТІ

ANNOTATION

This work is devoted to the development of an Internet platform aimed at creating a commercial platform for the sale of goods. The paper considers the main aspects of the development and implementation of such a platform, including the analysis of user requirements, interface design, functionality development, payment system implementation and data security. The issue of marketing strategy and customer support was also considered. The result of the work is an innovative Internet platform that facilitates efficient trade operations, provides user convenience and increases business competitiveness in the online market.

KEY WORDS: E-COMMERCE, SELLING, WEB RESOURCE MODELING, FRAMEWORK, WEB DEVELOPMENT, SEQUENCE DIAGRAM

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 8
ВСТУП 9

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДОЛОГІЙ СТВОРЕННЯ ВЕБ-ДОДАТКІВ 11
1.1. Ознайомлення з областю веб-програмування, визначення його задач та цілей 11
1.2. Опис та класифікація веб-додатків 16
1.3. Технології розробки Інтернет-платформ та веб-додатків 22
1.4. Структура веб-додатків та аналіз розробки програмного забезпечення 24
1.5. Опис етапів життєвого циклу програмного забезпечення 27
1.6. Аналіз проблеми створення веб-додатку для комерційних завдань 30
1.7. Висновки до розділу 34

2 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДІВ І ЗАСОБІВ СТВОРЕННЯ ВЕБ-ДОДАТКУ ДЛЯ КОМЕРЦІЙНОГО МАЙДАНЧИКА 35
2.1. Обґрунтування вибору актуальної методології для заданої предметної області 35
2.2. Особливості JavaScript та JS фреймворків 38
2.3. Опис платформи Node 42
2.4. Проектування інтернет-платформи для комерційного майданчика 44
2.5. Розробка діаграм варіантів використання засобами UML 49
2.6. Особливості використання нереляційноїх бази даних MongoDB 52
2.7. Висновки до розділу 54

3 РЕАЛІЗАЦІЯ ІНТЕРНЕТ-ПЛАТФОРМИ ДЛЯ КОМЕРЦІЙНОГО МАЙДАНЧИКА 55
3.1 Проектування структури веб-платформи 55
3.2. Проектування структури бази даних 57
3.3. Програмна реалізація 61
3.4. Розробка інтерфейсу та функціональності інтернет-платформи для комерційного майданчика 67
3.5. Висновки до розділу 77

ВИСНОВКИ 78
СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА 79
ДОДАТКИ
БІБЛІОГРАФІЧНА ДОВІДКА

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ID – Identification – деяке значення, яке ідентифікує певний запис у базі даних.

UDF - Unidirectional Data Flow (односпрямований потік даних)

CBP - Component-Based Programming (компонентно-орієнтоване програмування)

CBFS - Component-Based Folder Structure (структура папок на основі компонентів)

REST - Representational State Transfer (передача репрезентативного стану)

ВСТУП

Безперечно, можна сказати, що на даний період часу веб-розробка стала дуже цікавою сферою, яка дуже активно набирає популярність. У першу чергу це пов’язано з розширенням кількості користувачів Інтернету. Дана мережа з кожним роком розширює власний арсенал можливостей, які включають у себе велику кількість сервісів різного призначення. Серед таких є веб-сайти. Прогрес технологій веб-програмування на даний час вражає, оскільки є дуже різноманітним і дає можливість познайомитися з даною сферою всім розробникам програмного забезпечення. Сама ж сфера вже породила нові спеціальності та вже не обходиться базовим набором технологій для розробки. Завдяки чому самі веб-сайти стали значно складнішими і зараз представляють собою дещо більше ніж просто статичні сторінки. Серед всіх веб-сайтів завжди вирізнялися ті, які працюють як сервіси та надають деякі послуги. Інтернет уже давно став новим ринком для комерційних задач, а застосуванням для таких задач стали веб-сайти. Зараз вони відомі як комерційні веб-додатки, або торгові площадки.

Мета даної роботи дослідити сучасні комерційні веб-додатки, оскільки вони є дуже актуальними на сьогоднішній день. Можливість здійснювати покупки та продаж продукції через мережу Інтернет вже давно не новинка. Даний процес ще називають електронною комерцією, і на даний момент часу потреба у таких послугах колосальна. Даний напрямок не стоїть на місці та задає для розробників веб-додатків все складніші задачі. А про шалену конкуренцію між такими веб-сервісами можна і не казати, оскільки кожен з них реалізує одну і ту саму задачу, але по різному. Реалізація всіх бажань виділитись серед конкурентів лягає на плечі розробників, які на щастя зараз мають всі необхідні для цього інструменти. Сама ж реалізація таких веб-додатків не є простою задачею. Такі веб-сервіси потребують не малих ресурсів та можливості підтримки. А необхідність реалізовувати унікальний функціонал та можливості

додають до базових задач і ще більше завдань. Але не дивлячись на це, такі веб-додатки є цікавими у плані розробки і дозволяють розробникам застосувати всі навички у веб-розробці, не дивлячись на те, який масштаб буде у сервісу. А сучасні веб-технології дозволяють добитися чудового результату, як у плані затрат, так і у плані можливостей. Додатково у даній роботі буде досліджено, як вибраний стек технологій справиться із покладеною задачею і який результат буде у підсумку. Задача даної роботи є дослідити не тільки самі комерційні веб-додатки та вибраний стек технологій для побудови такого додатку, але і показати наглядно, що на даний момент часу представляє собою веб-програмування у плані задач та цілей, які може виконувати, і чому саме веб-додатки є популярним рішенням сучасних потреб користувачів.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДОЛОГІЙ СТВОРЕННЯ ВЕБ-ДОДАТКІВ

1.1. Ознайомлення з областю веб-програмування, визначення його задач та цілей

Для кращого розуміння подальшої роботи, слід визначитись з поняттям веб-розробки, або ж просто веб-програмуванням. Отож, чим воно відрізняється від всім знайомого поняття програмування. Можна сказати, що контекстом виконання, але про це пізніше, спочатку слід дати визначення. Комп'ютер виконує невеликі інструкції, йдучи вниз за списком, роблячи кожен по рядку.

Суть веб-програмування полягає у програмуванні клієнт-серверного додатку. Але як це розуміти? Все досить просто, такий додаток являється розподіленою системою, яку в контексті розробки програмних продуктів називають системою незалежних додатків. Така архітектура додатку характерна тим, що його можна розділити на дві частини – клієнтську та серверну. Ці дві частини вирішують одне спільне завдання, а спілкування між цими частинами відбувається за рахунок мережевих протоколів. А тепер докладніше про обидві частини.

Клієнтська призначена для спілкування з користувачем за допомогою спеціального веб-інтерфейсу, який формує спеціальні запроси, які адресовані серверу, задля виконання потреб користувача.

Для виконання даних частин клієнт-серверного додатку необхідні спеціальні програмні компоненти. Найчастіше для клієнта – це веб-браузер (програмне забезпечення для перегляду веб-сторінок, взаємодії з веб-додатками та іншими задачами, які направлені для роботи з веб-середовищем). По суті, серверна частина це також програма, але яка може виконувати свої функції віддалено, використовуючи для цього ресурси спеціальних систем – серверів.

Можна сформувати такий принцип таких додатків – клієнт замовляє, сервер виконує, все просто.

Але чому саме така архітектура і такий принцип? Адже будь-яке виконання можна організувати у рамках однієї повноцінної програми та системи, але на це є аргумент – а саме, те що виконання даної програми може потребувати великої обчислювальної потужності, тобто потужних характеристик системи, і надавати такі можливості кожному користувачу буде не дуже вигідно. Тому сервер є чудовим варіантом для виконання такої архітектури, адже його потужність може буде в рази вищою за звичайні комп'ютери, які використовують користувачі. Принцип простий – потужна система, яка виконує програму на одній стороні – доступ до менш затратної по потужності клієнтської частини програми зі сторони користувачів, які користуючись ним, просто спілкуються з головною програмою через спеціальні протоколи. Ще можна додати, що на таких серверах можна організовувати бази даних, які можуть мати великі обсяги сховища зберігання даних, тим самим звільняючи користувача від питання зберігання програми та даних. Такі випадки є дуже хорошою практикою.

Отож, уже було сказано про те, що сервер та клієнт спілкуються між собою, але що це означає? Якщо ми говоримо про веб-програмування, то крім серверної та клієнтської частини присутній ще мережевий протокол, завдяки якому клієнт та сервер взаємодіють один з одним. Даний мережевий протокол має назву HTTP, його запроси вміє обробляти веб-сервер, а клієнтська програма створювати запроси завдяки йому.

Отож, як відбувається збереження контексту виконання на стороні клієнта:

- в оперативній пам'яті інтернет-браузера клієнта. Також за допомогою сервісів, які підтримуються інтернет-браузером після виходу HTML п'ятої версії, такими як LocalStorage та SessionStorage. Це досить хороший варіант збереження, але зі своїми недоліками, оскільки не всі дані можуть зберігатися

на стороні клієнта через неможливість перетворити всі дані у текстовий формат і передати на сервер;

- за допомогою невеликих фрагментів текстових даних, що зберігаються на стороні клієнта, всім відомих – cookies. Вони зберігаються, як можна здогадатись, у текстових файлах, у розділах, що виділені операційною системою для зберігання різної інформації користувача. Ці дані передаються кожен раз у заголовках HTTP запитів на сервер. Але є один недолік, користувач може заборонити приймання cookies зі своєї сторони.

Тепер розглянемо, як відбувається збереження контексту на стороні веб-серверу:

- найпростішим варіантом є збереження в області оперативної пам’яті, що виділяється веб-сервером і має назве – стан програми. Дані у такому випадку будуть доступні з усіх веб-сторінок додатку всім його користувачам;

- схожим варіантом з першим є збереження також в оперативній пам’яті і також має свою назву – стан сеансу. У чому тоді відмінність від стану програми? Діло у тому, що дана область виділяється окремо для кожного користувача і зберігається на термін його роботи з веб-додатком;

- зберігання у структурах бази даних. Все просто – він не залежить від того, який веб-сервер. А у чому його надійність? Також нічого складного, просто життєвий цикл і стан сеансу керується веб-сервером, у той час, як інформацію у базі даних контролює веб-додаток.

Отже, розбір поняттям веб-програмування та його відмінність від “класичного” програмування закінчено. Але це ще не все, слід ще розібратись із його складовими. Веб-програмування включає у себе три складові:

- *Веб-дизайн.*

На даний момент часу дана складова не втратила своєї актуальності, але у ній стались значні зміни. Колись спеціаліст у даному напрямку мав володіти вміннями побудови не тільки дизайну графічного користувацького інтерфейсу, але і розробки часткового його функціоналу та структури, що значно

розширювало спектр вмінь для даної складової. Але це було колись, зараз веб-дизайнер виконує лише типові завдання: створення макету або шаблону веб-додатку, його складові частини такі як меню, елементи керування, навігація та інші. Тобто розробка дизайну графічної оболонки користувацького інтерфейсу. Вміння побудови веб-сторінки за допомогою технологій розмітки HTML та стилізації CSS, перейшли до базовий навичок у наступній складовій;

- *Програмування на стороні клієнта*.

Дана складова, як уже було сказано, була пов'язана з веб-дизайном, але зараз у даному напрямку відбувся значний стрибок вперед. Але чому була? Все діло у тому, що програмування на стороні клієнта почало стрімко розвиватись, і перестало включати у себе базовий набір навичок розмітки веб-сторінки за допомогою HTML та стилізації за допомогою CSS, тому веб-дизайнери стали зосереджуватись лише на побудові дизайну веб-додатків, а займатись повноцінною розробкою клієнтської частини почали розробники, які вміли працювати не лише з базовими технологіями, але і складнішими. Але основою залишається HTML, як головна складова веб-сторінки.

Невід'ємною частиною front-end є JavaScript і яка підтримується всіма браузерами. Раніше дана мова застосовувалась лише для написання користувацьких сценаріїв та інших функціональних можливостей графічного інтерфейсу веб-додатку, але зараз дана мова програмування розрослася до набагато більших масштабів і можливостей. Про неї докладніше також буде в іншому розділі. Виділяється також JavaScript-бібліотека jQuery, у склад якої входить велика кількість функцій для роботи з об'єктною моделлю інтернет-сторінки. Варто згадати і про технологію створення веб-сторінок з асинхронним завантаженням гіпертексту в окремій області активної сторінки – AJAX, ну і про різні CSS фреймворки, такі як Bootstrap, Foundation та Materialize CSS, які направлені покращити розробку інтерфейсу користувача та надати нові можливості. Видно, що великий спектр технологій значно розширив можливості даної складової і відділив її свого часу від веб-дизайну,

але і на цьому все не зупинилось. Але і зберігається потреба у простіших інтерфейсах і відповідно потреба у старих технологіях, які також розвиваються. Як завжди все залежить від розробників і спектру їх умінь;

- *Програмування на стороні серверу*.

Одна із найпопулярніших складових веб-програмування, яка на даний момент часу також значно розширила спектр технологій, які можуть тут застосовуватись. Програмування даної частини веб-додатку найбільш нагадує класичне програмування, оскільки тут описується вся логіка веб-додатку, те як він буде відповідати на отримані запити і які операції виконувати. Сервер взаємодіє також із базою даних, тобто він записує у неї дані і отримує, і відповідно надає їх при необхідності користувачу на клієнтській стороні. Раніше дана складова була обмежена у технологіях, які з нею взаємодіяли та на яких можна було писати логіку. Можна виділити PHP, як донедавна найпопулярніша мова веб-програмування та стороні серверу. На даний момент часу такі популярні мови програмування, як Java, Python та C# дозволяють програмувати на стороні серверу. PHP не втратила свою актуальність, але у неї є прямі конкуренти, такі як Ruby та JavaScript. Особливу увагу варто звернути на JS, як на мову програмування, яка ще донедавна вважалась мовою програмування виключно для клієнтської сторони, але зараз дозволяє повноцінно використовувати себе по обидві сторони, тому вона і є зараз найпопулярнішою мовою програмування для веб. Можна ще згадати про мову SQL, яка необхідна для роботи з реляційними базами даних, такими як MySQL, Oracle та Microsoft SQL Server, ну і про нереляційні бази даних, такі як MongoDB. Дана складова веб-програмування з розширенням мов програмування, які можна використовувати, стала робити значні кроки вперед у своєму розвитку, оскільки велика кількість розробників різного профілю почали працювати з нею, ну і складність систем почала зростати, бо потреби до серверної складової веб-додатків почали зростати, тим самим роблячи її актуальною у сфері програмування.

1.2 Опис та класифікація веб-додатків

У попередньому розділі було з’ясовано, що таке веб-програмування і всі його аспекти, але і не один раз було згадано про веб-додатки, веб-сайти та технології їх створення. Тому у даному розділі слід розібратись з самими веб-додатками. Перш за все, слід визначити, що таке веб-сайт та як це поняття пов’язане з веб-додатками.

Веб-сайт – це група або набір веб-сторінок, що взаємопов’язані спільним доменним ім’ям та доступні глобально.

Веб-додаток – це програмне забезпечення або програма, яка доступна для користувача через веб-браузер. Вона складається з інтерфейсу користувача та функціональної частини, яка працює на веб-сервері [2].

Як бачимо, дані поняття відрізняються, але для більшості користувачів між ними немає різниці. Різниця очевидна тим, хто займається розробкою, наприклад для мене дана різниця розвіялась, щойно став до розробки проекту даної роботи.

Тепер розглянемо веб-додатки. На відміну від веб-сайтів, веб-додатки направлені на те, щоб взаємодіяти з користувачем. Вони також можуть доносити потрібну інформацію до користувача, але і надають йому можливість взаємодіяти із собою, тобто обробляти інформацію. Веб-додатки популярні через такі причини:

- прості в обслуговуванні завдяки тому, що один і той самий код у всій програмі;
- відсутні проблеми із сумісністю, для використання достатньо лише веб-браузера;
- відсутність потреби у згоді користувача на обновлення додатку;
- немає прив’язки до платформи – доступ як з комп’ютера, так із мобільного пристрою;
- немає потреби у встановленні такого додатку.

– він завжди доступний через мережу Інтернет;

Отож, можна виділити основні відмінності між ними. Нижче визначені деякі з них:

- Інтерактивність.

Уже було сказано, що веб-додатки цим і відрізняються, що дають можливість користувачу не тільки взаємодіяти зі сторінкою, але здійснювати маніпуляції з інформацією на даній сторінці. Уже було згадано у даній роботі про принцип “запит - відповідь”, адже саме веб-додатки повноцінно реалізують його, користувач взаємодіє з інтерфейсом та отримує відповідь. Але варто зауважити, що і у веб-сайти починають додавати більше інтерактивності, адже користувачам це подобається;

- Аутентифікація.

Для веб-додатків необхідна аутентифікація, оскільки вони пропонують користувачу набагато ширший спектр варіантів дії , ніж веб-сайти. А для веб-сайтів вона не є обов’язковою. Зазвичай користувач може зареєструватись, щоб отримувати якусь інформацію або регулярне обновлення і додаткові функції.

- Функціональність.

Веб-сайти не мають відрізняються відсутністю складних функціональних можливостей, які притаманні для веб-додатків;

- Тип програмного забезпечення.

Як уже було сказано, різниця між ними для звичайних користувачів не очевидна, але насправді вона, адже як уже було зауважено, їхні визначення відрізняються і недарма. Веб-додаток насправді є лише частиною веб-сайту, він не є повноцінним веб-сайтом. А сам веб-сайт – це повноцінний продукт, яким веб-додаток не являється. Отже, за допомогою веб-браузера ви потрапляєте завжди на веб-сайт, але яким він є за типом, очевидно лише розробникам;

- Deployment (розгортання).

У випадку веб-сайту перекомпіляція не потребується при незначних змінах, просто оновлюється HTML-код. А веб-додатки потребують перекомпіляції та нового розгортання всього проекту.

Якщо підводити підсумок, то у випадку з моїм проектом я виберу реалізацію веб-додатку, оскільки мені потрібно, щоб користувач взаємодіяв із сайтом, тобто більша інтерактивність.

Визначають багато різних типів веб-додатків, але я продовжу розбирати ті, які вважаються основними на даний момент і є актуальними (рис. 1.1).



Рисунок 1.1 – Перелік основних типів веб-додатків

- Статичні веб-додатки.

Такі додатки не відрізняються інтерактивністю, а їхня основна функція – донести інформацію. Звучить знайомо, адже вище таке вже згадувалось. Все діло у тому, що веб-сайти і є статичним типом веб-додатків. Їхня мета – представляти і доносити інформацію при мінімальній взаємодії з користувачем.

- Динамічні веб-додатки.

Дані додатки набагато складніші. Це зумовлене тим, що їхня інтерактивність дуже висока і користувач може взаємодіяти з додатком. Зазвичай вони мають панель адміністрування, де адміністратори можуть виправляти або змінювати вміст додатку, будь то текст чи зображення. Даний тип додатків також був розглянутий вище, оскільки динамічний тип притаманний веб-додаткам.

- Комерційні веб-додатки.

Такі веб-додатки ще називають інтернет-магазинами. Їхня суть просто, комерційні задачі. Також може бути присутня панель управління для адміністратора такого веб-додатку для керування сайтом та його вмістом.

- Портали.

Дані веб-додатки досить таки відрізняються від інших, що були перелічені, оскільки доступ до розділів та інших категорій веб-додатку здійснюється через головну сторінку.

- Анімовані веб-додатки.

Анімація або графічний контент є обов'язковим атрибутом таких сайтів. Вони схожі на статичні, але передають інформацію інакше, з різними ефектами та більша частина їх вмісту – це графічний та відео контент.

- Веб-додатки із системою керування вмістом.

У таких веб-додатках для адміністраторів присутня панель, завдяки якій вони можуть керувати вмістом сайту, додавати новий або оновлювати. Приклад такого веб-додатку – WordPress. Він безсумнівно являється популярним і поширеним в Інтернеті. Про нього написано багато матеріалів, тому для тих, кому потрібен такий тип веб-додатків, легко з ним розберуться.

- Односторінкові веб-додатки (Single-Page Application).

Це новий тип додатків, який є дуже популярним, оскільки працює всередині веб-браузера. Це лише одна веб-сторінка, яку ви відвідуєте, а потім завантажує ввесь інший вміст за допомогою JavaScript. Найпопулярнішим JavaScript фреймворком для даного типу веб-додатків є Angular.

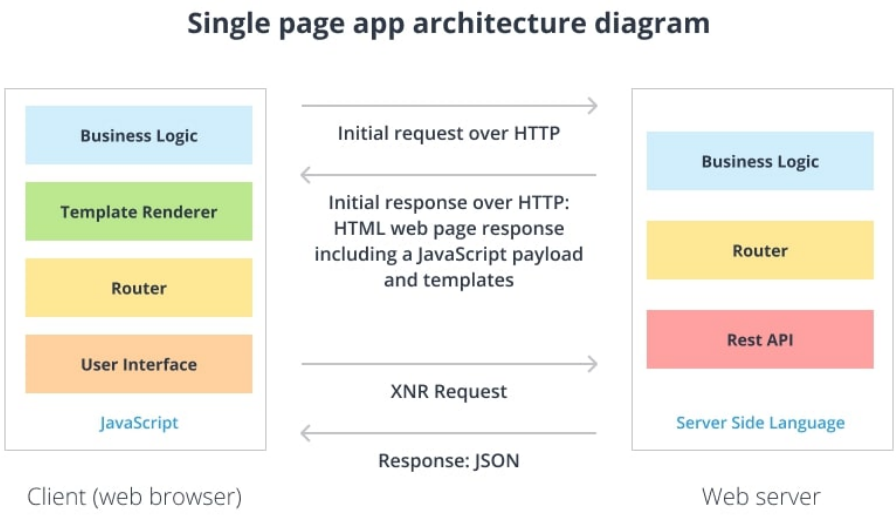


Рисунок 1.2 – Архітектура Single-Page Application

- Багатосторінкові веб-додатки (Multiple-Page Application).

Такі додатки працюють звичним способом. Кожен запит або відображення даних змушують сервер відправляти нову сторінку до веб-браузера. Такі програми набагато більші за розміром та структурою за SPA, завдяки чому мають багато рівнів графічного інтерфейсу користувача. Таку особливість покращує AJAX, завдяки якому не потрібно турбуватися за передачу великої кількості даних між клієнтом та сервером, що воно може займати більше часу. Дана технологія дозволяє оновити лише окремі частини веб-сайту. Але і у розробці такі веб-додатки складніші за SPA.

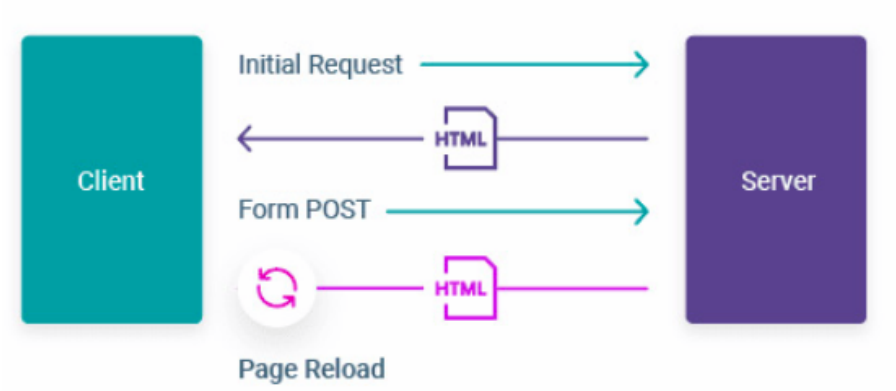


Рисунок 1.3 – Цикл Multiple-Page Application

На рисунку 1.4 представлена типова структура веб-додатка з набором найбільш поширених компонентів, які групуються за функціональними областями.

Шар уявлення зазвичай включає компоненти, які відповідають за користувацький інтерфейс (UI) та логіку представлення. Шар бізнес-логіки включає компоненти, які відповідають за бізнес-логіку, бізнес-процеси та бізнес-сутності. Шар доступу до даних включає компоненти, які забезпечують доступ до необхідних даних та веб-сервісів.

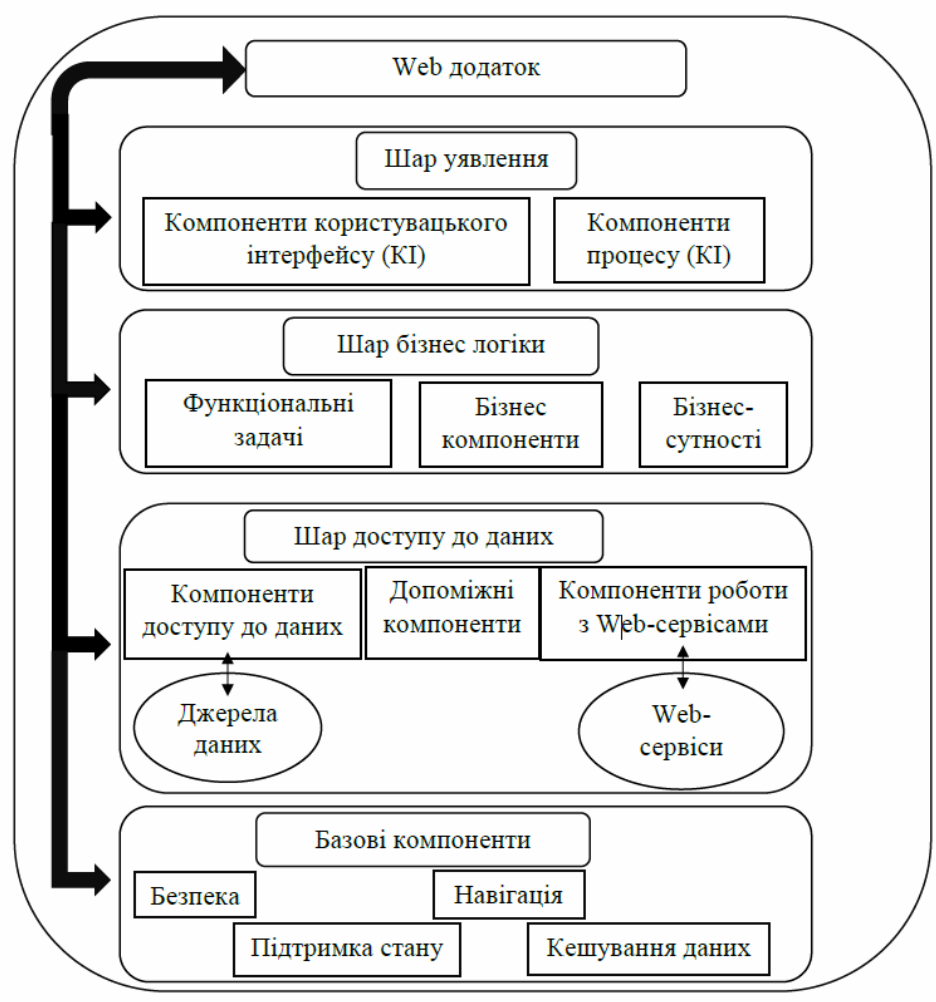


Рисунок 1.4 - Типова структура web-додатка

Отже, кожен тип веб-додатків має власні сильні сторони, але не варто забувати, що все ж вони залишаються веб-сайтами. Вони не є самостійними

додатками, хоча дуже стараються ними здаватися. Але вони мають багато плюсів, що роблять їх цікавими у розробці та використанні. Щодо розробки, то тут є багато різних технологій про які варто сказати. Про те, що веб-додаток ділиться на дві частини – front-end та back-end (сторона клієнта та серверу) уже було сказано, але слід ще раз про це згадати, щоб розділити технології для кожної зі сторін. Раніше також про деякі з них було згадано при докладному розборі кожної зі сторін, але зараз повний перерахунок всіх основних на даний час.

1.3. Технології розробки Інтернет-платформ та веб-додатків

На сьогоднішній день у веб-розробці є багато різноманітних технологій розробки. Нижче перераховані такі популярні рішення:

1) HTML (мова розмітки HyperText, надає структуру веб-сайту);
2) CSS (каскадні таблиці стилів, надає стиль елементам веб-сайту. Для кращої роботи із собою підтримує препроцесори SASS та LESS);
3) Мови програмування:
   - JavaScript – підтримується всіма браузерами і застосовується при розробці як і клієнтської сторони, так і серверної;
   - TypeScript – розширює можливості JavaScript, виділяється тим, що підтримує статичну типізацію (явне визначення типів) та використання повноцінних класів, як в інших об’єктно-орієнтованих мовах;
   - Python – використовує фреймворки Django, Flask та Pylons для веб;
   - Ruby – використовує Ruby on Rails фреймворк для побудови веб-додатків;
   - PHP – мова програмування для веб, яка генерує HTML-сторінки на стороні веб-серверу;

- Java – об’єктно-орієнтована мова програмування, яка використовується для написання програмного забезпечення для будь-якого застосування;

4) Фреймворки:
   - Node.js – це JavaScript фреймворк для серверної сторони веб-додатків;
   - Ruby on Rails – full-stack фреймворк, що побудований на Ruby;
   - Django – full-stack фреймворк, що побудований на Python;
   - Cordova – мобільне середовище розробки для IOS та Android засобами JavaScript, CSS та HTML;
   - Bootstrap – UI фреймворк для побудови інтерфейсу користувача;
   - WordPress – система керування контентом, що побудована на PHP;
   - .NET – full-stack фреймворк для C#;
   - Angular – front-end JavaScript фреймворк;
   - Vue.js - front-end JavaScript фреймворк;
5) Бібліотеки:
   - jQuery – JavaScript бібліотека з великим набором функцій для взаємодії JS та HTML, також для роботи з DOM-елементами;
   - React JS – JavaScript бібліотека для побудови інтерфейсу користувача;
   - Redux – бібліотека для менеджменту та зберігання стану компонентів у сучасних JavaScript додатках;
6) Бази даних:
   - MongoDB – NoSQL база даних;
   - Redis – популярний key-value магазин;
   - PostgreSQL – популярна SQL база даних;
   - MySQL;
   - SQL Server;

7) Протоколи:
   - HTTP;
   - DDP;
   - REST;
   - GraphQL – являє собою мову запитів з відкритим кодом, яка є альтернативою REST;
8) Формати даних:
   - JSON;
   - XML;
   - CSV;
9) Інші інструменти:
   - Webpack – інструмент, що дозволяє скомпілювати модулі веб-додатку, такі як JavaScript, HTML, CSS та інші, в єдині статичні одиночні файли;
   - Gulp – інструмент для автоматизації розробки клієнтської частини веб-додатку;
   - Babel – це транспілятор JavaScript, який дає можливість перевести ES6 та вище код у ES5, тим самим розширить підтримку коду старішими версіями веб-браузерів;
   - Git – система для контролю версій.

Отже, у даному розділі було з’ясовано, що існує велика кількість типів веб-додатків, але все ж веб-додаток насправді є лише частиною веб-сайту, він не є повноцінним веб-сайтом.

1.4. Структура веб-додатків та аналіз розробки програмного забезпечення

Будь-яка програма має свою структуру, або архітектуру і веб-додатки не є виключенням. Про неї вже була згадано у попередніх розділах, але повноцінно

розібрано дане питання буде лише у даному розділі. Отож, про веб-додатки вже було сказано, про їх типи та технології їх створення також, але про те, що є їхньою основою, що їх зв'язує будемо дізнаватися детальніше, оскільки знаємо лише, що веб-додатки складаються з клієнтської частини та серверної.

Користувач Інтернету отримує можливість бачити певну сторінку у веб-браузері завдяки деякій взаємодії між різними компонентами додатків, інтерфейсами користувача, системами середнього програмного забезпечення, базами даних та сервером. Тією основою, що поєднує це все та дозволяє взаємодіяти разом , є саме архітектура веб-додатку. Якщо говорити коротко, то архітектура – це така структура веб-додатку, що включає різні компоненти та зовнішні програми. Це є дуже важливим, оскільки перед тим, як сідати за розробку веб-додатку, важливо вибрати тип його архітектури, а також модель компонентів веб-додатків. Правильний вибір сприятиме кращому розумінню розробки та правильній роботі веб-додатку. Тепер слід зрозуміти, як визначити правильну архітектуру. Про те, як працюють веб-додатки уже було сказано, ми знаємо, що розробники розробляють код на стороні клієнта та сервера, що будь-який код, що пишеться для сервера здатний відповідати на HTTP-запити, що користувач отримує від сервера веб-сторінки та інформацію та велику кількість технологій, що допомагають розробляти такі додатки.

Існує три типи архітектури веб-додатків, про одну з них уже згадувалось у попередньому розділі, але тут ми розглянемо її як архітектуру, а не тип, тому почнемо розгляд саме з неї [4]:

1) Single Page Application (SPA) або односторінковий веб-додаток.

Такий веб-додаток замість завантаження абсолютно нових сторінок із серверу у відповідь на дії користувача здійснює динамічне оновлення вмісту поточної сторінки. Основою такого є технології AJAX та XML і такі додати розроблені таким чином, щоб вони вимагали лише найнеобхідніші змістовні та інформаційні елементи, що призводить до інтерактивного користувацького досвіду;

2) Мікросервіси.

Це невеликі та легкі сервіси, які виконують єдиний функціонал. Такий функціонал виконується за допомогою Microservices Architecture Framework, що надає ряд переваг і дозволяє розробникам не тільки підвищити продуктивність додатку, але і прискорити процес розгортання. Компоненти, що створені такою технологією не залежать безпосередньо один від одного. Таким чином, їх не потрібно створювати за допомогою однієї мови програмування і розробники, що працюють з технологією мікросервісів можуть не залежати від набору технологій і вільно їх обирати;

3) Безсерверні архітектури.

Дана архітектура веб-додатків використовується, коли компанії, що буде використовувати дане програмне забезпечення не хоче керувати або підтримувати сервери, а також пристрої, для яких розробили його. Розробник такого додатку консультується зі стороннім постачальником віддалених інфраструктурних послуг для аутсортингу сервера, а також керування інфраструктурою. Перевага у тому, що це дозволяє виконувати логіку коду, не задумуючись над завданнями, які пов’язані з інфраструктурою.

Тепер розглянемо компоненти архітектури веб-додатків. Архітектура включає у себе такі компоненти:

- *UI/UX компоненти веб-додатку*. Дані компоненти є частиною плану компонування інтерфейсу веб-програми, але не мають нічого спільного з функціонуванням архітектури веб-додатків. До них входять журнали діяльності, інформаційні панелі, сповіщення, налаштування, статистика та інші;
- *Структурні компоненти*. Такі компоненти веб-додатку в основному стосуються функціональності, керування та зберігання бази даних. Основними структурними компонентами веб-додатку є – клієнтська та серверна частини. Також сюди ще можна віднести сервер баз даних. Клієнтський компонент дозволяє взаємодіяти з функціями веб-додатків кінцевому користувачу, розробляється за допомогою технологій HTML, CSS та JavaScript.

Серверний компонент може бути побудований за допомогою різних технологій, список яких був у попередньому розділі, є основним центром керування веб-додатком, який підтримує бізнес-логіку та багатошарові програми. Сервер бази даних пропонує бізнес-логіку та відповідну інформацію або дані, які зберігаються та керуються сервером веб-додатку. Він зберігає, витягує та надає інформацію.

У підсумку можна сказати, що веб-розробка не стоїть на місці, розвиваючись вона замінює застарілі структури та основні компоненти, тому вибір архітектури веб-додатку є дуже важливим, адже визначатиме його програмну реалізацію, його продуктивність та правильну роботу у подальшому використанні. Правильна архітектура відкриває шлях для майбутніх планів розширення та масштабованості веб-додатку. Тут ми плавно переходимо до процесу, що буде після визначення архітектури – розробка програмного забезпечення. Слід провести аналіз його розробки.

При аналізі процесу розробки програмного забезпечення, варто звернути увагу на те, що даний процес має свій життєвий цикл, що об’єднує у собі етапи розробки.

1.5. Опис етапів життєвого циклу програмного забезпечення

Життєвий цикл програмного забезпечення – це процес, який використовується в інформаційних технологіях для процесів аналізу, проектування, програмної розробки та тестування програмного забезпечення. Метою даного циклу є створити програмне забезпечення високої якості, яке відповідає всім вимогам клієнта. Життєвий цикл ще називають процесом розробки ПЗ, що визначає завдання, які мають бути виконані на кожному кроці процесу розробки.

ISO/IEC 12207 – це міжнародний стандарт для процесів життєвого циклу програмного забезпечення. Метою даного стандарту є визначення всіх завдань, що необхідні для розробки та обслуговування ПЗ.

Процес життєвого циклу направлений на застосування до програмного проекту в рамках програмної організації. У ньому виділяють такі складові, як складання детального плану, який має описувати, як розробляти, підтримувати, обновляти та змінювати конкретне програмне забезпечення.

Отже, у життєвому циклі розробки ПЗ виділяють такі основні послідовні етапи:

Етап 1: Планування та аналіз вимог. Аналіз вимог є найважливішим та фундаментальним етапом життєвого циклу програмного забезпечення. Планування проекту проводять старші члени колективу з розробки ПЗ на основі вимог замовника, відділу продажу, аналізу ринку та експертів у галузі. У подальшому зібрану інформацію оцінюють та вкладають у план розвитку з метою проведення дослідження техніко-економічного обґрунтування продукту у доступній, експлуатаційній та технічній сферах. Планування вимог щодо забезпечення якості та визначення ризиків, пов'язаних із проектом, також проводиться на даному етапі. Результатом дослідження технічної обґрунтованості є визначення різних технічних підходів, яких варто дотримуватися для успішної реалізації проекту з високою продуктивністю та мінімальними ризиками для бізнесу;

Етап 2: Системний аналіз та визначення вимог. Після виконання першого етапу, результатом якого буде виконаний аналіз вимог, наступним кроком є визначення та документування вимог до продукту та отримання їх схвалення від замовника. Це робиться через документ SRS (Software Requirement Specification) – специфікації вимог до програмного забезпечення – який складається з усіх вимог до продукту, які повинні бути спроектовані та розроблені протягом життєвого циклу проекту;

Етап 3: Розробка архітектури системи. Якщо з архітектурою проекту ще не було визначено, або плани змінили під час виконання перших двох етапів, то на допомогу приходить SRS (Software Requirement Specification) – так званий орієнтир, щоб визначити архітектуру продукту, який потрібно розробити. Виходячи з вимог, що зазначені в SRS, зазвичай пропонується і документується більше одного підходу до проектування архітектури продукту в специфікації проектного документа або DDS (Design Documentation Specification). DDS переглядається всіма зацікавленими сторонами проекту та ґрунтується на різних параметрах (оцінка ризику, надійність виробу, модульність проектування, бюджет та часові обмеження), тобто для проекту вибирається найкращий підхід до проектування. Підхід до розробки архітектури чітко визначає всі архітектурні модулі проекту разом із його комунікацією та поданням потоку даних із зовнішніми та сторонніми модулями (звісно, якщо такі є). Внутрішня конструкція всіх модулів вибраної архітектури повинна бути визначена у найменших деталях в DDS.

Етап 4: Створення та розробка продукту. На даному етапі життєвого циклу уже відбувається фактична розробка програмного забезпечення. Код програмного продукту пишеться у відповідності до специфікацій, зазначених у документах SRS та DDS. Якщо архітектура проекту була визначена розроблена правильно, детально та організовано, то написання проекту можу проходити плавно, без клопотів. Розробники повинні дотримуватися вказівок кодування, визначених їх керівниками та компанією. Мова програмування обирається відповідно до типу ПЗ що розробляється та вимог до нього. Розробка програмного забезпечення може мати різні етапи розробки, залежно від методологій програмування, якими керуються, такі як SCRUM або AGILE підхід.

Етап 5: Тестування продукту. Один з ключових етапів життєвого циклу, який є підсумком всіх етапів. Однак дана стадія стосується виключно етапу тестування вихідного проекту, де розбираються повідомлення про помилки та

дефекти, здійснюється виправлення коду та його повторне тестування. Даний етап буде тривати до того часу, поки проект не досягне стандартів якості, що були визначені у SRS.

Етап 6: Розгортання та обслуговування. Після закінчення тестування, продукт очікує на готовність розгортання і формального виску на відповідному ринку. Іноді розгортання продукту відбувається поетапно, відповідно до визначеної бізнес-стратегії організації. Спочатку продукт може бути випущений в обмеженому сегменті та протестований у реальному бізнес-середовищі. Після отримання зворотного зв'язку, товар може бути визначений як фінальний і бути випущеним, або із запропонованими удосконаленнями він буде доведений до ідеалу і через деякий час випущений повторно в сегменті націлювання на ринок. Після випуску товару на ринок, наступає фаза технічного обслуговування, починається підтримка програмного забезпечення.

Також варто сказати пару слів про безпеку. Через вимогливий характер розробки ПЗ, правильна оцінка безпеки часто залишається непоміченою, і таким чином, залишається велика кількість "дир" у проекті, що є хорошою новиною для хакерів. Важливо, щоб перед випуском, програмний продукт пройшов ретельну оцінку рівня безпеки.

1.6. Аналіз проблеми створення веб-додатку для комерційних завдань

Торгівля, або продаж та купівля товарів, завжди мала великий попит протягом століть. Для одних це був спосіб заробітку, а для інших спосіб отримати бажане. Але з розвитком технологій, з'явилися нові способи здійснювати даний процес.

Електронна комерція [5], або електронний бізнес – це реалізація покупки та продажу товарів через мережу Інтернет. Також передбачається можливість передачі даних та коштів між двома та більше сторонами. Якщо говорити

простіше, то це сучасні інтернет-магазини. На сьогоднішній день, це найпопулярніший спосіб реалізації продукції. Традиційний метод торгівлі уже давно не являється єдиним, для кращої ефективності використовують ще і метод продажу в Інтернеті. Будь-яка людина, яка хоче реалізувати деякий товар, використовує зараз саме Інтернет мережу, оскільки він користується попитом і має ряд переваг, таких як зручність, немає прив'язки до розташування, електронна оплата і масовість, оскільки користувачів в Інтернеті є велика кількість, і знайти саме свого покупця стає простіше. А з прогресом смартфонів, які тепер дозволяють повноцінно виходити у мережу Інтернет та користуватися веб-сайтами, електронна комерція є однією з найпопулярніших напрямків в Інтернеті. Адже в Інтернеті є можливість рекламувати свої послуги та ресурси, та охоплювати великі межі для реалізації товарів. На даний момент існує безліч ресурсів, де можна здійснювати електронну комерцію, для прикладу – соціальні мережі (Instagram, Facebook) зараз дозволяють створювати бізнес-профілі користувачів і викладати пости із товарами. Також є різні площадки, що дозволяють продавати товари некомерційним особам, або просто обмінюватися товарами. Але найпопулярнішими ресурсами для цього все ж залишаються веб-додатки. Про те, що для комерційних задач є окремий тип веб-додатків була розказано у попередньому розділі. Але для електронної комерції можуть використовуватись як статичні веб-додатки, так і комерційні, але задачі тоді відрізнятимуться. Якщо необхідно донести людям інформацію про онлайн-магазин, то веб-сайт чудовий варіант, але для повноцінної реалізації потрібно реалізовувати як комерційний веб-додаток, адже у такому додатку потрібна інтерактивність та реалізація складної бізнес логіки.

Комерційні веб-додатки у свою чергу можна розділити на такі типи, виходячи з функцій, які вони реалізують:

Бізнес для бізнесу. Перший тип реалізує електронні транзакції товарів і послуг між компаніями. Для прикладу: підприємство продає продукцію іншим підприємствам;

Бізнес-споживач. Другий тип реалізує електронні транзакції товарів та послуг між компаніями та споживачем. Даний тип є всім знайомим, оскільки є він є типом всіх знайомих нам інтернет-магазинів, де ми купуємо те, що нам необхідно;

Між споживачами. Третій тип реалізує електронні транзакції товарів та послуг між споживачами, також переважно присутня і третя сторона. Прикладами можуть служити всім знайомі сервіси для продажу товарів користувачів – eBay та OLX;

Споживач бізнес. Четвертий тип реалізує електронні трансакції товарів і послуг, коли фізичні особи пропонують товари або послуги компаніям. Прикладом може служити випадок, коли користувач за оплату рекламує сторінку деякого продукту або компанії для продажу, задля того, щоб його аудиторія відвідала побачила цю рекламу і відвідала сторінку тих, хто замовив рекламу.



Рисунок 1.5 – Основні види електронної комерції

Комерційний тип веб-додатків є дуже складним у плані розробки, оскільки містить велику кількість різноманітних функцій взаємодії з товаром,

фільтрації товарів, реалізацію їхньої покупки, оплати та оформлення замовлення. Також може бути присутня панель управління для адміністратора такого веб-додатку для керування сайтом та його вмістом. Але складність, не відміняє того, що такі додатки дуже актуальні, і на даний момент такий спосіб реалізації товарів є пріоритетним. Складність полягає ще у тому, що такі веб-додатки є дуже масштабними, з великою кількістю роботи, що на клієнтській стороні, що на серверній. Також потрібна правильно організована робота з базами даних. Такий проект вимагає великої кількості ресурсів, як людських, так і матеріальних, оскільки він є багаторівневим. Також сучасні тенденції веб-програмування вимагають використання у такому проекті всіх сучасних технологій.

Проблемою таких комерційних веб-додатків є унікальність. Велика кількість таких онлайн магазинів є схожими за функціоналом та логікою, тому тут доводиться реалізовувати різні унікальні функції, що будуть відрізняти кожен такий веб-додаток від інших. Дана задача покладається на обидві сторони веб-додатку, оскільки на front-end частину покладається задача унікального дизайну і графічних удосконалень, а на back-end частину реалізація унікального функціоналу.

Провівши аналіз веб-додатків для комерційних задач для того, щоб визначитись який тип є кращим рішенням для проекту даної роботи та які задачі покладаються на такі додатки, я зробив вибір на користь на поєднання типів бізнес-споживач та між користувачами. Це зумовлено популярністю веб-сервісів для реалізації товарів між користувачами, а наявність можливості реалізації власної продукції дозволить компанії власнику реалізовувати і власні бізнес цілі. Такий веб-сервіс дозволить охопити велику частину ринку та при зростаючій популярності розширюватись та надавати користувач всі актуальні послуги. Тому список вимог до проекту виглядатиме таким чином:

- Перегляд асортименту площадки та товарів користувачів;
- Перегляд сторінки вибраного товару;

- Сортування товарів;
- Пошук товарів;
- Сторінка уподобаних товарів;
- Кошик (додавання товарів у корзину та вилучення товарів з неї);
- Оформлення замовлення;
- Покупка товарів онлайн;
- Реєстрація у веб-сервісі;
- Вхід у веб-сервіс;
- Сторінка особистого профілю користувача;
- Редагування інформації про користувача;
- Додавання, видалення та редагування власних товарів користувачем;
- Контрактна форма;
- Підписка на розсилку новин;
- Відображення інформації про користувача, що розмістив товар;
- Товар повинен мати стан (новий або вживаний), категорію та інші дані.

У підсумку хочеться сказати, що такі веб-додатки є цікавими для розробників, у плані розробки вони представляють цікаве випробування власних можливостей, і здатність удосконалити власні вміння під час розробки, бо уже було сказано, що такі додатки є складними у реалізації.

1.7. Висновки до розділу

У даному розділі було розглянуто область веб-програмування, визначено види веб-додатків, їхню структуру, їх задачі та цілі. Було проведено аналіз проблеми створення веб-додатку для комерційних завдань. Зроблено аналіз розробки програмного забезпечення та зроблено огляд актуальних на даний момент технологій створення веб-додатків.

2 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДІВ І ЗАСОБІВ СТВОРЕННЯ ВЕБ-ДОДАТКУ ДЛЯ КОМЕРЦІЙНОГО МАЙДАНЧИКА

2.1. Обґрунтування вибору актуальної методології для заданої предметної області

У попередньому розділі було багато сказано за веб-додатки, про їхні особливості та популярність, які типи їх є і для чого використовуються, але також про велику кількість технологій для їх створення. Для тих, хто хоче розпочати займатися розробкою веб-додатків, постає запитання у тому, що використовувати. Таким саме запитання я ставив і собі. Будучи знайомим із основними веб-технологіями для клієнтської сторони додатку, я почав детально дізнаватися про технології для серверної частини. І виявляється, що у даних двох частин веб-додатку є дещо спільне, а саме – JavaScript.

Можна сказати, що зараз настав момент, коли JavaScript можна назвати найпопулярнішою мовою для веб-програмування. А починалось все скромно, мова застосовувалась для написання сценаріїв на клієнтській стороні, а протягом тривалого часу перетворилась на невід'ємну складову сторони клієнта і почала використовуватись у якості мови програмування для серверної сторони, а все завдяки Node. Перспективи цілісного набору технологій, що використовують JavaScript очевидні – більше ніяких змін контексту, тільки одна мова програмування для всього. Це дало перспективу знавцям JavaScript використовувати власні вміння не лише для клієнтської сторони, але і для серверної. Звісно, що тут існують і свої аспекти, що потребуються у вивченні, але бар'єр у вигляді різних мов програмування на обох сторонах – почав зникати. Але можна сказати, що прогрес JavaScript, ще не закінчено, дана мова програмування лише проходить етап становлення, оскільки має ряд власних особливостей, які не дозволяють ставити її у ряд з іншими мовами програмування. Але перспектива є, і зараз можна повністю її оцінити. Детальніше сама мова буде розібрана у подальшому, але зараз про неї потрібно

було сказати, оскільки вона грає велику роль у тому, чому був вибраний ряд технологій, що побудовані саме для її використання.

Про Node уже було згадано. Але що це таке? Це те, завдяки чому JavaScript зробив крок вперед, те що винесло його за рамки веб-браузера. Це платформа для створення додатків завдяки засобам JavaScript. Але що означає винесла за рамки веб-браузера? Раніше, для того, щоб бачити результат виконання JavaScript коду потрібен був веб-браузер, але зараз Node дозволяє виконувати такий код у себе ж на комп'ютері, і бачити результат виконання. Але докладніше про Node буде у подальшому розділі, зараз слід розуміти, що дана платформа дозволяє розробити веб-додаток засобами JavaScript. Саме Node дозволить використовувати веб-фреймворк Express. Даний фреймворк і дозволить повноцінно реалізувати проект даної роботи. Але як саме? Він необхідний для створення сервера та побудови серверної логіки додатку, тобто він надає набір функцій для реалізації веб-додатку.

Отже, на даний момент зрозуміло, що веб-додаток буде розроблятися за допомогою таких засобів як Node та Express, а поєднує їх мова JavaScript. Наступним важливим компонентом являється база даних. Як уже було розібрано у попередніх розділах, невід'ємною частиною будь-якого сучасного веб-додатку є база даних. Існує багато видів баз даних, але є і такі, що краще за інших підходять для платформи Node. У даному питанні не є секретом те, що велику популярність мають реляційні бази даних, що мають велику кількість інтерфейсів, але активна розробка для даної платформи дала поштовх для нового підходу для баз даних – так званих NoSQL. Вони ще відомі як нереляційні бази даних, але коректніше буде їх назвати документ-орієнтованими базами даних або базами типу "ключ - значення"[6]. Вони реалізують більш простий підхід до зберігання даних. Таких баз даних є велика кількість, але найбільше виділяється – MongoDB. Але чому саме така база даних? Все діло у тому, що додатки Node досить просто інтегруються з MongoDB та прекрасно взаємодіють. А саме завдяки типу бази "ключ -

значення”. Такі бази даних краще підходять для зберігання об’єктів, що і робить їх хорошим доповненням для Node та JavaScript. А у даному проекті використовуються схеми даних, що відповідають саме такому типу баз даних.

Ось такий набір технологій буде використовуватись у даному проекті. Даний набір дозволить побудувати сучасний веб-додаток, що буде відповідати всім вимогам. Завдяки платформі Node є можливість використовувати велику кількість модулів, що у процесі розробки значно спростять реалізацію тих чи інших функцій. Мінімалістичний веб-фреймворк Express допоможе організувати серверну логіку додатку, і при потребі доповнюється іншими модулями. А MongoDB забезпечить просту інтеграцію з додатком завдяки модулю mongoose, швидку та коректну роботу з даними. Даний стек технологій прекрасно виконує задачу щодо побудови швидкого та функціонального веб-серверу для веб-додатку, а також хорошу взаємодію з клієнтською стороною, завдяки шаблонізації. Сама ж клієнтська частина буде побудована завдяки стандартним технологіям front-end розробки – HTML, CSS та JavaScript.

Отже, поєднання JavaScript, Node та Express – це чудовий набір для веб-розробників, яким потрібен потужний стек технологій, що швидко розгортається, який встиг добре себе зарекомендувати[6]. Він допоможе створити першокласний веб-додаток з прекрасним набором функцій, що буде зручний в експлуатації і чудовий варіантом для бізнесу, оскільки підтримка та розгортання такого додатку економічно вигідно та ефективно. Важливим моментом JavaScript, який є важливим для кожного веб-розробника, як найпопулярніший веб-інструмент для всіх сучасних веб-браузерів. Практично неможливо зараз не використовувати JavaScript, якщо потребується добитися сучасної функціональності на стороні клієнта. А екосистема Node надає велику кількість фреймворків, бібліотек і застосунків, що прискорюють розробку і надають найкращі практики для написання коду. На даний момент це чудовий вибір, оскільки сучасний JavaScript це прекрасна мова програмування для актуальних задач, його підтримують великі компанії, він легкий у

використанні, підтримується сучасними веб-браузерами, ну і головний аспект – це прекрасні фреймворки і бібліотеки, як на стороні клієнта, так і сервера. Тому даний стек технологій є чудовим вибором у сучасному веб-програмуванні.

2.2. Особливості JavaScript та JS фреймворків

JavaScript – це інтерпретована мова програмування з об'єктно-орієнтованими можливостями. Базовий синтаксис мови нагадує мови програмування C, C++ і Java такими програмними конструкціями, як умовний оператор *if*, цикл *while* та оператор *&&*. Але схожість проявляється лише у синтаксисі, адже в інших аспектах це відмінні від них мова.

JavaScript – це нетипізована мова програмування, що означає відсутність потреби надавати тип змінним. Об'єкти даної мови мають можливість інтроспекції і динамічної зміни типу механізмів прототипів. Механізм наслідування JavaScript сильно відрізняється від C++ та Java, оскільки використовується механізм наслідування прототипів. Регулярними виразами і засобами роботи з масивами зроблені у даній мові за образом і подобою мови Perl.

На даний момент у веб-браузері JavaScript може здійснювати такі операції:

- додавати новий HTML код на веб-сторінку, здійснювати маніпуляції із вмістом, модифікувати стилі;
- обробляти події, тобто реагувати на дії користувача;
- відправляти запити на сервера, скачувати і загружати файли завдяки технологіям AJAX та COMET;
- робота з cookies, запити на введення даних користувача;
- зберігати дані на стороні клієнта завдяки *LocalStorage.*

Але і є можливості JavaScript, які обмежені у веб-браузері. Це зроблено заради безпеки користувача. Мета полягає у запобіганні доступу

недобросовісної веб-сторінки до особистої інформації або нанесення шкоди даним користувача.

Можна сказати, що JavaScript це не типова мова програмування. Її функцій та можливості визначили задачі для яких він розроблявся і у яких зараз працює. Все залежить від середовища виконання. Але є мінімум три сильні сторони JavaScrip:

- повна інтеграція з іншими веб-технологіями для побудови веб-сторінок, а саме HTML та CSS;
- прості речі виконуються просто і не потребують складної реалізації;
- підтримка всіма популярними веб-браузерами, де він включений за замовчуванням.

JavaScript являться єдиною технологією сторони клієнта, що поєднує у собі вище зазначені сторони. Ось чим він особливий[9]. Це на даний момент найпопулярніший інструмент для розробки інтерфейсів користувача для веб-додатків. Але він вже давно цим не обмежується, його можливості поступово ростуть, і задають тенденцію у веб-програмуванні.

JavaScript – це не типова мова програмування, причини цього вже були згадані. Більшість користувачів хочуть використовуючи його отримувати більші можливості, такі що присутні в інших мовах програмування. Це особливо проявляється у розробників, які до JavaScript працювали з іншими мовами програмування, такими як Java, C++, C#, які виділяються більшими можливостями, оскільки спектр їхніх задач охоплює значно більші масштаби. Але не лише потреба розробників, але і необхідність деяких проектів. Тому з’являються і набирають популярності деякі нові мови програмування, які транспіліруються або ж конвертуються у JavaScript, перш ніж почнуть працювати. Все це завдяки сучасним інструментам, що здійснюють даний процес швидко і прозоро, фактично даючи можливість розробникам писати код на іншій мові або на тому самому JavaScript лише з додатковими можливостями. Популярними прикладами є:

CoffeeScript – доповнення до JavaScript, що дозволяє писати компактний код з додатковими синтаксичними додатками, що відсутні у стандартному JavaScript. Такий код має більш короткий синтаксис;

TypeScript – розширює можливості JavaScript, виділяється підтримкою статичної типізації (явне визначення типів) та використання повноцінних класів, як в інших об'єктно-орієнтованих мовах задля розробки та підтримки великих і складних систем.

На даний момент є популярними такі фреймворки та бібліотеки:

- jQuery – найпопулярніша JavaScript бібліотека з великим набором функцій для взаємодії JS та HTML, також для роботи з DOM-елементами, обробки подій, анімації, селектори CSS та AJAX-запити;
- React - JavaScript бібліотека для побудови інтерфейсу користувача. Одна з найпопулярніших бібліотек сьогоднішнього дня. Використовує компонентний підхід побудови інтерфейсів, підтримує стан компонентів, віртуальний DOM-дерево та багато іншого. Плюси даної бібліотеки у компактності, ефективності, високій продуктивності та гнучкості. Також є можливість рендерингу на стороні серверу;
- Lodash та Uinderscore – бібліотеки, що надають можливість використовувати велику кількість інструментів для розширення функціональності додатку. Вони вирішують рутинні задачі щодо роботи зі строками, числами, масивами, об'єктами та іншим. Просто у використанні і мають багату на приклади використання документацію;
- Angular – один з найпопулярніших фреймворків для створення сучасних SPA веб-додатків. Велика підтримка і функціональність. Такий веб-додаток неважкий у підтримці та розгортанні;
- Vue.js – головним плюсом даного фреймворка є його простота та можливість використання в уже готовому додатку для розширення можливостей. Як і React, є чудовим засобом для створення інтерфейсів користувача. Використовує синтаксис шаблону HTML для прив'язки

DOM та даних. Моделі являються простими JS-об'єктами. Також використовує компонентний підхід;

- Ember.js – це чудовий front-end фреймворк на основі шаблону MVVM (Model-View-ViewModel). Реалізує шаблонізацію, зв'язування даних та бібліотеки. Являється комплексним рішенням для клієнтських додатків. Мінусом є його складність та великий дистрибутив;
- Redux – одне з найкращих рішень на сьогоднішній день, коли задача стоїть у контролі стану додатку та компонентів;
- Next.js – мінімалістичний веб-фреймворк для серверного рендерингу React додатків;
- Meteor.js – популярний full stack фреймворк, який призначений для побудови повноцінної інфраструктури веб-додатку та мобільних додатків;
- Express.js – мінімалістичний та гнучкий веб-фреймворк для веб-додатків.
- Node.js. Даний JS фреймворк призначений для реалізації повноцінної серверної логіки веб-додатку. Має великий набір функцій для комфортної розробки, що включає у себе велику кількість методів HTTP і проміжного ПО, призначеного для створення API. Має велику кількість плагінів, що надають додаткові можливості. Утворює з Node.js та MongoDB прекрасний стек для повноцінної веб-розробки[6].

Дивлячись на такий великий список технологій для JavaScript, можна зрозуміти, що дана мова являється універсальною, вона використовується у технологіях різного призначення. Потенціал JavaScript розкривається завдяки всім згаданим фреймворкам та бібліотекам, які уже встигли себе зарекомендувати як чудові інструменти у веб-розробці. Багато з ним прекрасно доповнюють одне одного при розробці і дозволяють розробляти повноцінні веб-додатки і не тільки. Недарма JavaScript являється зараз найпопулярнішою мовою програмування для веб і хорошим вибором для тих, хто хоче працювати

у даній сфері IT, адже вивчивши JavaScript, розробник відкриває для себе великі можливості у виборі технологій, які об’єднує дана мова програмування.

2.3. Опис платформи Node

Безсумнівно, Node можна назвати революцією JavaScript. Про те, яким давним-давно був JavaScript і для чого використовувався, було сказано у попередньому розділі. Звідти відомо, що порівняно з іншими мовами програмування, JavaScript мав вузьке коло застосування, що обмежувалося виконанням у веб-браузері. Але мова програмування мала можливості для серйозніших задач, які притаманні серйознішим мовам, таким як Java та C++. Розробникам хотілося повноцінно використовувати мову, тобто мати змогу писати код, який буде працювати за межами веб-браузера. Саме із такою метою був розроблений у 2009 році Node.

Node – це програмна платформа, що побудова на движку V8, задачею якого є перетворення JavaScript у машинний код[10]. Дана платформа перетворила JavaScript з мови вузькоспеціалізованого напрямку у мову загального призначення. Node додає можливість JavaScript працювати з пристроями вводу та виводу, із зовнішніми бібліотеками, працювати компонентам як модулям, організовувати веб-сервера, можливість розробляти комп’ютерні і мобільні програми, а також програмувати мікроконтроллери. Це була загальна інформація про саме платформу, але цим вона не обмежується.

Node – це середовище виконання JavaScript, яке включає у себе все необхідне для виконання програм, що написані на JS[11]. Хорошим атрибутом Node є підтримка менеджера пакетів npm без додаткових маніпуляцій, він доступний зразу після установки. Також варто відмітити те, що Node підтримується всіма операційними системами і установлюється як звичайний додаток, після чого система буде підтримувати його на системному рівні. Це є незалежність від платформи.

Головною метою з якою був розроблений Node – це дозволити JavaScript працювати на серверній стороні веб-додатків. Платформа стала новим різновидом веб-серверу. У деякому сенсі у Node багато спільного і іншими популярними веб-серверами. Але все ж , є відмінність. Перша відмінність це підхід. Аналогічно з веб-фреймворком Express, підхід Node до веб-серверу дуже мінімалістичний. Це проявляється у простоті установки та настройки. Це зумовлено простотою та ясністю конфігураційних опцій. Друга відмінність полягає в однопотоковості Node. Такий підхід зумовлений движком V8, на якому побудований Node, оскільки він призначений для перетворення JavaScript у машинний код, а сам JS є однопотоковим.

Отож, Node являється однопотоковим та асинхронним. Це дозволяє Node не потребувати великих апаратних ресурсів, щоб виконувати веб-додаток і виконувати деякі задачі одночасно. Інші веб-сервери можуть працювати багатопоточно, але це вимагає потужних апаратних ресурсів. Це має і власну ціну, оскільки щоб добитися такого, потребуються ресурси, як і кадрові, так і фінансові.

Якщо про написання додатків, то додатки Node мають деяку особливість. Завдяки движку V8, компіляція JavaScript коду у машинний відбувається прозоро. Це означає, що компіляція відбувається моментально, тобто зміни стають зразу видимими або доступними після обновлення файлу JS. Відсутність окремого кроку у компіляції зменшує важкість обслуговування і розгортання.

Як одну із особливостей Node також виділяють і відсутність буферизації. Це означає, що додатки Node ніколи не буферують будь-які дані. Такі програми виводяться дані окремими частинами.

Node поступово почав утворювати власну екосистему. Він став платформою, що об’єднює актуальні JavaScript технології. Він є основною частиною будь-якого додатку, що на ньому розробляється. Все через можливість виконання JavaScript на віддаленому від браузера сервері, що

відповідно дозволяє використовувати JavaScript фреймворки та бібліотеки. Так само з базами даних. Для Node прекрасно підходить MongoDB, як найпопулярніша база даних типу “ключ - значення”. Про стек технологій Node, Express, MongoDB уже було докладно сказано, адже він ліг в основу проекту даної роботи[6]. Але варто сказати, що до даного стеку чудово підходить фреймворк Angular та бібліотека React. Поєднання цих чотирьох технологій дозволяє створити комплексний веб-додаток. Їхнє поєднання ще називають MERN або MEAN. Таких може бути велика кількість, все залежить від того, яку технологію підкласти. Це наглядно показує все різноманіття екосистеми Node.

Також важливою перевагою екосистеми Node є великий вибір доступних пакетів. **NPM** (Node Package Manager) – це менеджер пакетів, тобто готових модулів, що можуть використовуватись при розробці. Це дозволяє одним розробникам випускати готові програмні рішення задач для інших, тим самим уникнути при розробці проекту повторного написання одних і тих самих компонентів.

Node це сучасна платформа, це те, що вивело JavaScript на новий рівень і значно розширило сферу, куди його можна інтегрувати. Як стало очевидним, Node утворив навколо себе цілу екосистему, яка по-максимуму дозволяє використовувати можливості сучасного JavaScript та технологій, що написані на ньому і заточені для веб. Це актуальна платформа для веб-розробників, які хочуть писати програмне забезпечення використовуючи одну мову програмування.

2.4. Проектування інтернет-платформи для комерційного майданчика

Перед тим, як почати програмувати, потрібно підійти до такої ж важливої частини як проектування. Спочатку робимо повний опис проєкту і взаємодії. Потім декомпозиція. Ділимо все на підсистеми і кожну описуємо.

Спочатку іде функціональне проєктування. Це методологія IDEF0 (рис. 2.1). Це побудова ієрархічної системи діаграм. Окрема діаграма і буде описом частинок.

Потрібно зрозуміти для діаграми IDEF0, що є на початку і що виходить в результаті. Спочатку є лише потреба клієнта у чомусь. Його особисті побажання, гроші. Замовлення обробляє менеджер. Для цього йому потрібні або інструкціїї або добре вивчити опис товарів. Не забуваємо і про законі України, та права покупців. Кінцева точка це сформоване замовлення та можливі поради менеджера.

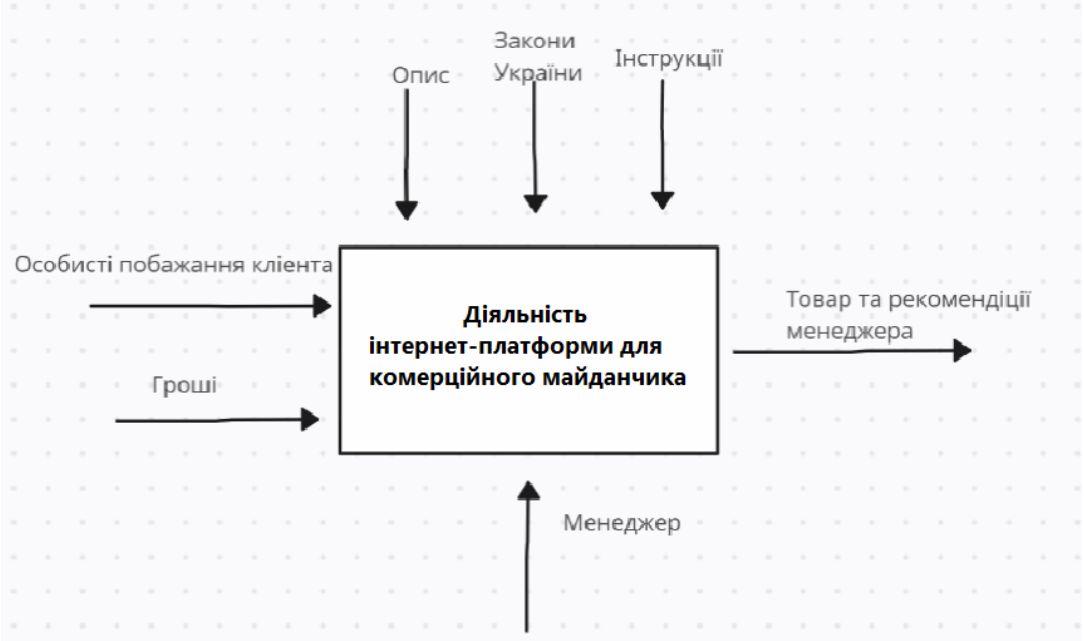


Рисунок 2.1 - Контекстна діаграма IDEF0

Для функціонального проєктування даного проєкту можна виділити кілька підсистем, які допоможуть розбити його на складові частини.

Підсистема вибору товару ця підсистема дозволяє клієнту обирати товари за їхніми побажаннями. Вона має включати функціонал для пошуку, фільтрації та перегляду товарів з урахуванням характеристик, категорій та інших параметрів

Підсистема оплати і купівлі товару. Ця підсистема забезпечує процес оплати і оформлення замовлення. Вона повинна мати інтеграцію з платіжними системами для при йому платежів, а також функціонал для заповнення форми замовлення та обробки платежу

Підсистема формування замовлень. Ця підсистема відповідає за збір і обробку інформації про замовлення. Вона включає функціонал для зберігання даних клієнтів, обробки виб раних товарів, розрахунку загальної вартості замовлення.

Підсистема сповіщення клієнта. Ця підсистема відповідає за надсилання повідомлення клієнту щодо підтвердження замовлення та інших важливих оновлень. Вона може використовувати електронну пошту, SMS або інші канали зв'язку для доставки повідомлень клієнту.

Кожна з цих підсистем має свої функціональні можливості та взаємодії з іншими частинами системи, але разом вони утворюють повну функціональність інтернет магазину. Нижче наведено діаграму декомпозиції інтернет – платформи для електронної комерції (рис. 2.2).

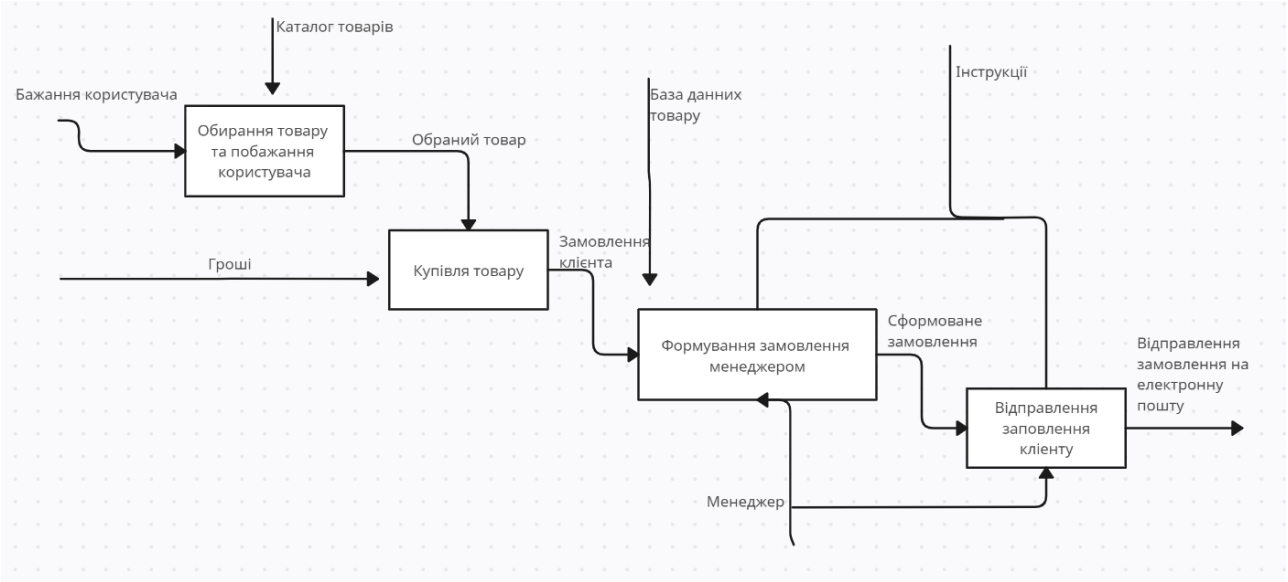


Рисунок 2.2 – Діаграма декомпозиції інтернет-платформи для комерційного майданчика

На рисунку 2.3 представлено діаграму прецедентів. Розберемо акторів згідно таблиці 2.1.

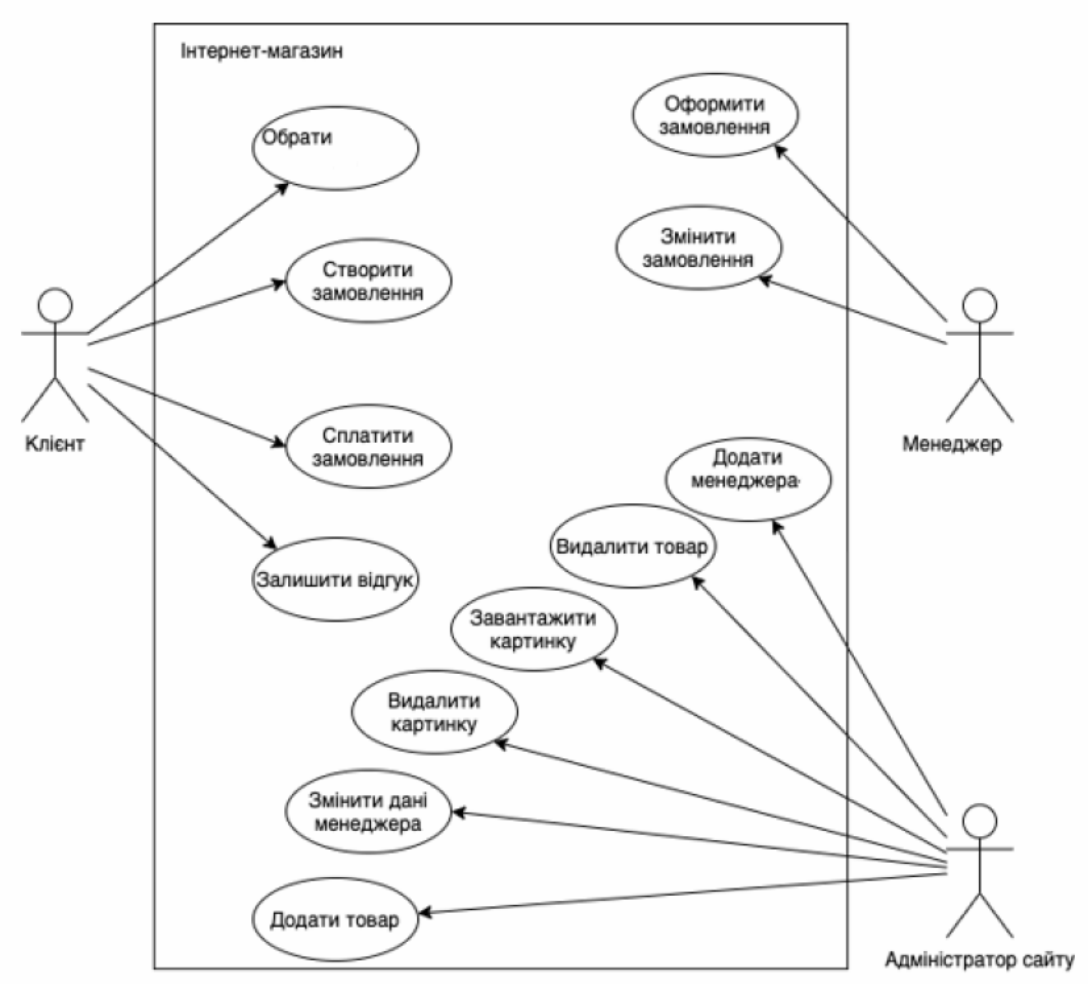


Рисунок 2.3 – Діаграма прецедентів

Таблиця 2.1 – Актори

Назва актора	**Опис актора**
Клієнт	Користується сайтом
Менеджер	Обробляє заявки
Адміністратор сайту	Має права «адміністратор»

Розберемо можливості акторів, що представлені нижче в таблиці 2.2.

Таблиця 2.2 – Можливості акторів

Назва	Опис
Обрати	Функція щоб клієнт міг обрати товар
Створити замовлення	Можливість створення замовлення
Сплатити замовлення	Функція оплати
Залишити відгук	Можливість поділитися відгуком з іншими клієнтами
Оформити замовлення	Функція менеджера прийняти замовлення
Змінити замовлення	Можливість менеджера редагувати замовлення
Додати менеджера	Фукнія адміністратора добавляти чи видаляти менеджерів
Видалити товар	Можливість редагування товару
Завантажити картинку	Можливість змінювати данні товару
Видалити картинку	Можливість змінювати данні товару
Змінити дані менеджера	Функція редагування профілю менеджера
Додати товар	Функція редагування товару

Діаграма послідовності – це графічний інструмент, який дозволяє моделювати взаємодію між об'єктами або компонентами системи впродовж певного часового періоду. Дозволяє зобразити порядок виконання операцій та комунікацію між об'єктами системи. Вона допомагає розібратися в послідовності подій і взаємодії між об'єктами під час виконання певного процесу. Загалом, діаграма послідовності допомагає уявити та зрозуміти, як система працює у реальному часі, відображаючи взаємодію об'єктів та послідовність подій. Вона є потужним інструментом для аналізу та

проєктування систем, а також для комунікації між розробниками та зацікавленими сторонами. Розглянемо діаграму активностей (рис. 2.4).

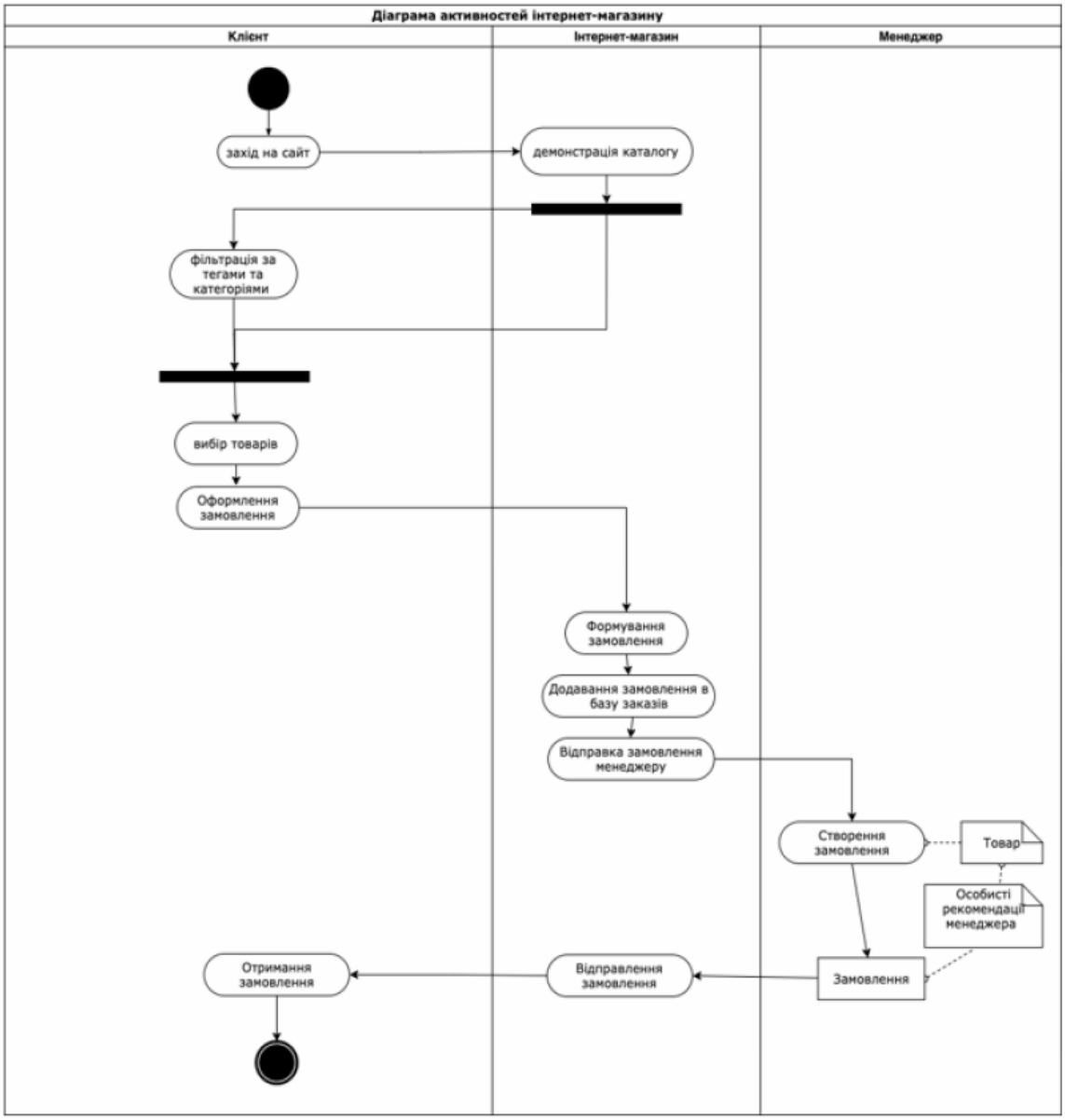


Рисунок 2.4 – Діаграма взаємодії

2.5. Розробка діаграм варіантів використання засобами UML

Розробка діаграми варіантів використання переслідує наступні цілі:

− визначити загальні межі і контекст модельованої предметної області на початкових етапах проєктування системи;

– сформулювати загальні вимоги до функціонального поведінки проєктованої системи;

– розробити вихідну концептуальну модель системи для її подальшої деталізації у формі логічних і фізичних моделей;

– підготувати вихідну документацію для взаємодії розробників системи з її замовниками і користувачами.

Діаграма варіантів використання для ролі адміністратора наведена на рисунку 2.5.

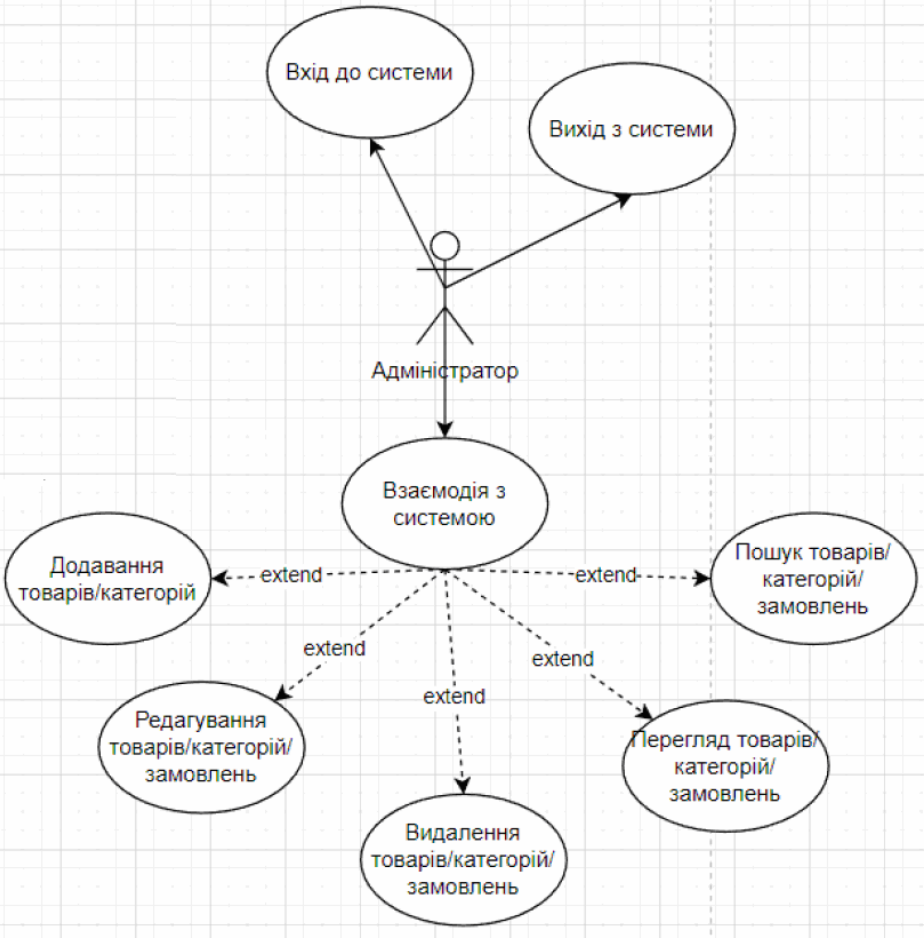


Рисунок 2.5 – Діаграма варіантів використання (роль: адміністратор)

На рисунку 2.6 зображено діаграму послідовності основного бізнес-процесу «Оформлення замовлення».

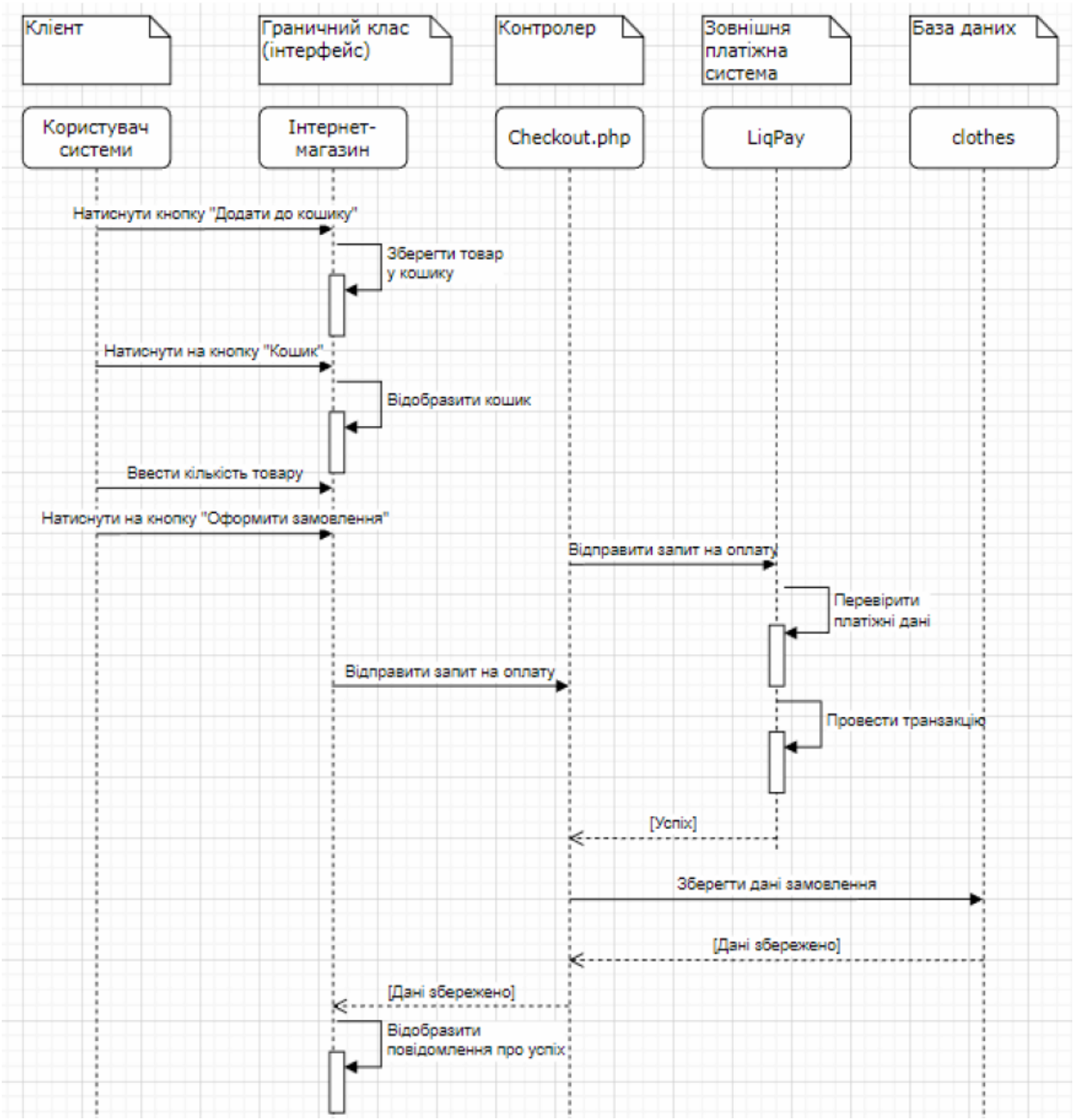


Рисунок 2.6 – Діаграма послідовності процесу «Оформлення замовлення»

Діаграма послідовності (Sequence Diagram) є одним з видів UML-діаграм і використовується для моделювання взаємодії між об’єктами або компонентами системи впродовж певного часового інтервалу. Вона дозволяє візуалізувати послідовність викликів, повідомлень та обміну даними між об’єктами та компонентами системи під час виконання певного сценарію.

Основною метою діаграми послідовності є зрозуміння та візуалізація логіки взаємодії між об’єктами в рамках конкретної функціональності або сценарію використання. Вона допомагає ілюструвати порядок викликів методів, передачу повідомлень, обробку подій та інші аспекти взаємодії між об’єктами.

2.6. Особливості використання нереляційноїх бази даних MongoDB

База даних – це структурний, організований набір однорідних та пов’язаних між собою даних, який зберігається на комп’ютері або може зберігатися на віддаленому сервері. Це певний контейнер для зберігання впорядкованих даних.

СУБД (Система управління базами даних) – спеціальне програмне забезпечення для організації супроводу і надання доступу до бази даних користувачу.

Про бази даних уже було згадано у даній роботі, але у даному розділі варто детально розглянути вибрану базу даних для проекту даної роботи.

MongoDB – це найпопулярніша NoSQL база даних та СУБД з відкритим кодом, орієнтована на документи або база даних типу “ключ-значення” і яка не потребує опису схеми таблиць.

Термін “NoSQL” – означає, що дана база даних нереляційна і не використовує мову SQL. Вона не базується на настільній структурі реляційних баз даних, але забезпечує зовсім інший механізм зберігання та пошуку даних. Цей формат називається BSON, тобто JSON-подібні документи і схему бази даних. Сама вона написана на мові програмування C++ і є популярним рішенням для платформи Node.

Система керування реляційними базами даних не є хорошим вибором, коли мова йде про обробку великих даних. Це є особливістю їхньої конструкції через що вони не мають горизонтального масштабування. Якщо база даних працює на одному сервері, то вона досягне межі масштабування.

Нереляційні бази даних є більш масштабованими та забезпечують високу продуктивність. MongoDB масштабує додаючи як найбільше серверів і збільшує продуктивність завдяки своїй гнучкій моделі документу.

Відмінності SQL та NoSQL баз даних:

1) SQL база даних має типовий дизайн схеми, який показує кількість таблиць та взаємозв'язок між цими таблицями. MongoDB документоорієнтована, тому тут немає поняття схеми та взаємозв'язку;
2) Складні транзакції не підтримуються в MongoDB через недоступність складних операцій з'єднання;
3) MongoDB дозволяє отримати дуже гнучку та масштабовану структуру документів;
4) MongoDB набагато швидша за SQL бази даних завдяки ефективним методам індексації та зберігання;
5) Дані типи баз даних поєднують деякі пов'язані терміни. Що у SQL базах даних називається таблицею, у MongoDB називається колекцією. Аналогічно, кортеж називається документом, а стовпець – полем. MongoDB кожному об'єкту колекції надає "_id" за замовчуванням, що забезпечує унікальність кожного документу та можливість у подальшому шукати його по id. Він подібний до первинного ключа у SQL базах даних.

У MongoDB можна виділити такі особливості:

- Орієнтована на документи або документоорієнтована. MongoDB зберігає основну тему в мінімальній кількості документів. На відміну від SQL баз даних вона не розбиває її на кілька реляційних структур;

- Індексація. Відсутність індексації призводить до того, що базі даних доводиться сканувати кожен документ колекції, щоб вибрати ті, які відповідають запиту. Це не є ефективно, тому для ефективного пошуку індексація необхідна. MongoDB використовує її для обробки великих обсягів даних за короткий термін;

- Масштабованість. MongoDB здійснює масштабування горизонтально за допомогою сегментування (розділення даних по різних вузлах кластера). Дані у такому випадку розділяються на фрагменти даних за допомогою ключа сегментування, через що фрагменти даних рівномірно будуть рознесені по вузлах, що розташовані на багатьох різних фізичних серверах;

- Реплікація та висока доступність. MongoDB збільшує доступність даних за допомогою декількох копій даних на різних серверах. База працює лише з оригінальним екземпляром, а інші потрібні для страхування основного.;

- Агрегація. Операції здійснення агрегації обробляють записи даних та повертають готові результати.

На даний час MongoDB підтримує роботу з всіма популярними мовами програмування, такими як C, C++, C#, Java, Node, Perl, PHP, Python, Ruby, Scala, Go та іншими. MongoDB є чудовим вибором на сьогоднішній день. Вона забезпечує сучасний підхід до зберігання даних та має широкий спектр застосувань.

2.7. Висновки до розділу

У даному розділі було обґрунтовано вибір методології для заданої предметної області та визначення необхідних технологій. Було проведено аналіз кожної технології, що буде застосована у проекті даної роботи. Зроблено аналіз того, чому саме вибрані технології є актуальними на сьогоднішній день для розробки веб-додатків та області їх застосування.

3 РЕАЛІЗАЦІЯ ІНТЕРНЕТ-ПЛАТФОРМИ ДЛЯ КОМЕРЦІЙНОГО МАЙДАНЧИКА

3.1 Проектування структури веб-платформи

У даному розділі буде здійснено реалізацію програмного продукту. Обґрунтування вибраної методології його реалізації було здійснено у попередньому розділі 2.1. За типом даний веб-додаток є комерційним з моделлю компонентів веб-додатків – один сервер та одна база даних. Задачі він буде виконувати типові для свого типу. Архітектура даного веб-додатку буде типовою для вибраних технологій – MVC (Model-View-Controller).

Реалізація програмного продукту починається з проектування структури веб-додатку. Архітектура для веб-додатків створених засобами Node та Express.js продемонстрована на рисунку 3.1.

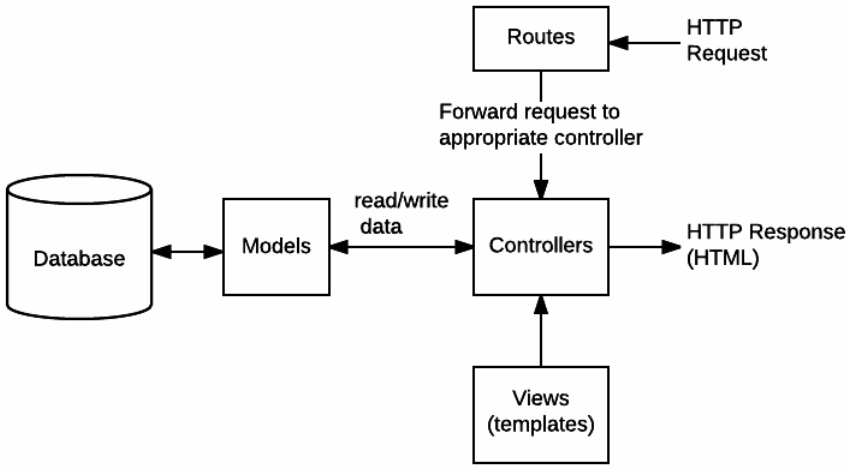


Рисунок 3.1 – Архітектура веб-платформи

Виходячи з даної архітектури, стає зрозумілим те, що повинно бути присутнім у веб-додатку. Проект повинен містити моделі, що будуть здійснювати взаємодію з базою даних, представлення клієнтської частини, що

будуть відображатись користувачу, маршрути, що будуть пересилати запит до відповідного контроллера та самі контроллери,, що будуть обробляти запити на сервер. Структура даного програмного продукту зображена на рисунку 3.2.

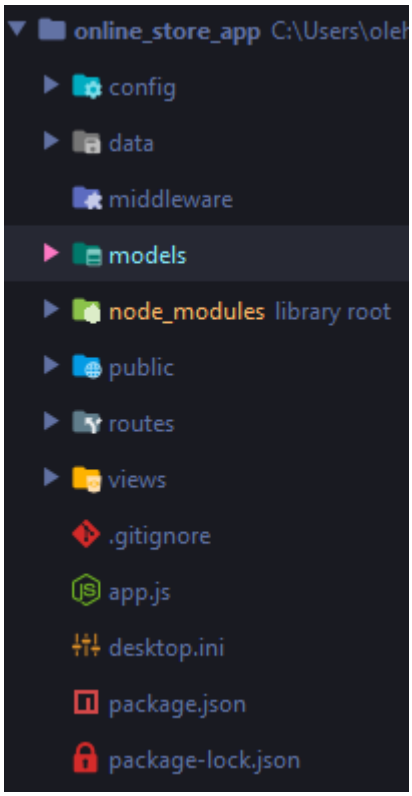


Рисунок 3.2 – Структура проекту

Структура проекту:

- config – містить файли конфігурації проекту;
- data – зберігаються файли керування даними проекту;
- middleware – проміжне програмне забезпечення;
- models – моделі бази даних;
- node_modules – папка зберігання бібліотек проекту;
- public – папка представлень статичних файлів проекту;
- routes – маршрутизація;
- views – представлення клієнтської частини.

Детальна структура на подана на рисунку 3.3.

– app.js – реалізація серверної частини веб-додатку;
– package.json – файл опису залежностей пакетів проекту.

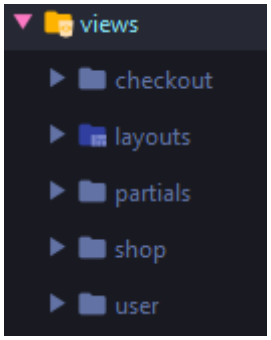


Рисунок 3.3 – Папка Views

Отже, структуру проекту було визначено. На черзі проектування структури бази даних, що є не менш важливим етапом розробки ПЗ.

3.2. Проектування структури бази даних

Базою даних проекту є MongoDB. Для взаємодії з нею у проекті реалізовані моделі. За допомогою даних моделей здійснюється читання сервером необхідної інформації з бази даних та запис у неї відповідної інформації. Кожна модель означає наявність у базі даних відповідної колекції, де зберігаються дані. Веб-додаток включає у себе 7 моделей.

Модель User зображена на рисунку 3.4.

```
{
    _id:<ObjectId1>,
    firstName:String,
    lastName:String
    phone:String,
    username:String,
    password:String,
    __v:0
}
```

Рисунок 3.4 – Модель User

Дана модель містить унікальне для кожного документа поле *_id* та поля, що використовується для роботи з користувачами. Поле паролю шифрується за допомогою бібліотеки *bcrypt*. Модель створить у базі даних колекцію *users*.

Модель **Product** зображена на рисунку 3.5.

```
{
 _id:<ObjectId1>,
 imagePath:String,
 title:String
 category:String,
 description:String,
 price:Number,
 rating:Array,
 state:String,
   __v:0
}
```

Рисунок 3.5 – Модель **Product**

Дана модель містить унікальне для кожного документа поле *_id* та поля необхідні для роботи з товарами у веб-додатку. Колекція – *products*. Модель **UserProduct** зображена на рисунку 3.6.

```
{
 _id:<ObjectId1>,
 imagePath:String,
 title:String
 category:String,
 description:String,
 price:Number,
 user:<ObjectIdUser>,
 state:String,
   __v:0
}
```

Рисунок 3.6 – Модель **UserProduct**

Дана модель містить унікальне для кожного документа поле *_id* та поля, що необхідні для роботи з товарами користувачів. Варто виділити поле *user*,

яке буде містити *id* відповідного користувача. Модель створить у базі даних колекцію *userproducts*. Модель **Subscribe** зображена на рисунку 3.7.

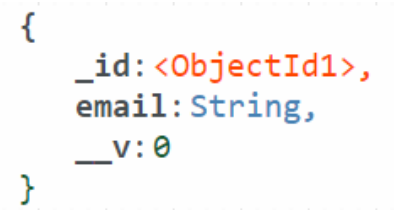


Рисунок 3.7 – Модель **Subscribe**

Дана модель містить унікальне для кожного документа поле *_id* та поле електронної пошти. Модель створить у базі даних колекцію *subscribes*. Використовується для підписки на розсилку. Модель **Contact** зображена на рисунку 3.8.

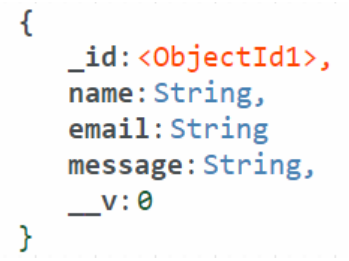


Рисунок 3.8 – Модель **Contact**

Дана модель містить унікальне для кожного документа поле *_id* та поля, що необхідні для підтримки контакту з користувачами. Модель створить у базі даних колекцію *contacts*.

Модель **Cart.** Дана модель необхідна для роботи з корзиною покупок. У ній реалізовані функції додавання до корзини, її очищення та вилучення товарів. Вона формує поле *cart*, яке буде використане у подальшому для оформлення замовлення. Працює з сесією.

Модель **Wish.** Дана модель необхідна для роботи з уподобаними товарами. У ній реалізовані функції додавання до списку бажань та вилучення. Працює з сесією.

Модель **Order** зображена на рисунку 3.9.

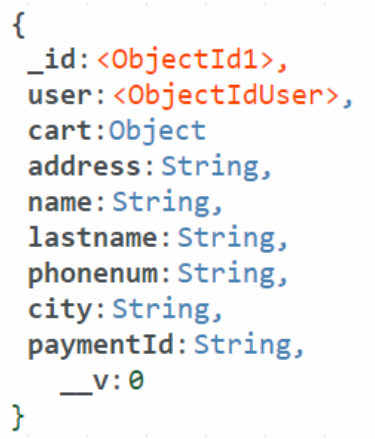


Рисунок 3.9 – Модель **Order**

Дана модель містить унікальне для кожного документа поле *_id* та поля, що необхідні для оформлення замовлення користувачем. Виділяються поле *cart*, яке формується моделлю корзини, поле *paymentId*, яке містить *id* здійсненої онлайн оплати та поле *user*, яке буде містити *id* відповідного користувача. Модель створить у базі даних колекцію *orders*.

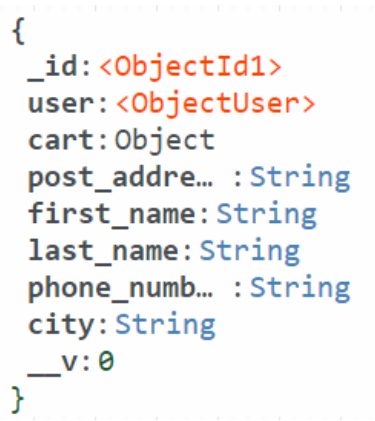


Рисунок 3.10 – Модель **Contact**

Модель **PostOrder** зображена на рисунку 3.10. Дана модель містить унікальне для кожного документа поле *_id* та поля, що необхідні для оформлення замовлення користувачем, яке не буде оплачене онлайн. Виділяються поле *cart*, яке формується моделлю корзини та поле *user*, яке буде містити *id* відповідного користувача. Модель створить у базі даних колекцію post*orders*.

Для роботи з базою даних використовується MongoDB Atlas. База даних проекту знаходиться на віддаленому кластері під назвою ShopAppCluster. Колекції веб-додатку зображені нижче на рисунку 3.11.

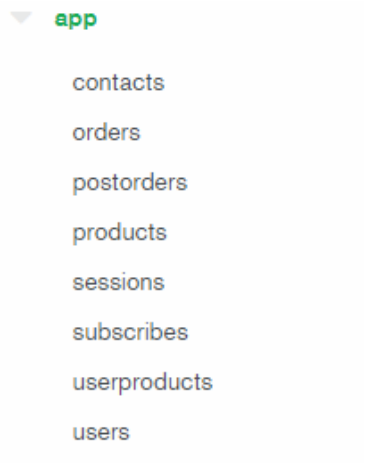


Рисунок 3.11 – Колекції бази даних MongoDB

Отже, використовуючи можливості та особливості бази даних MongoDB були створені такі моделі, що забезпечать взаємодію веб-серверу з самою базою даних при записі даних та їх читанню.

3.3. Програмна реалізація

Програмна реалізація веб-додатку на платформі Node починається з ініціалізації проекту. Після чого буде створений файл *package.json*, який буде містити інформацію про проект та список залежностей пакетів, що

використовуються. Частина вмісту файлу *package.json* нижче на рисунках 3.12. На цьому рисунку зображені залежності пакетів, які зберігає файл *package.json*.

```
"license": "ISC",
"dependencies": {
  "bcrypt": "^4.0.1",
  "body-parser": "^1.19.0",
  "config": "^3.3.1",
  "connect-flash": "^0.1.1",
  "connect-mongo": "^3.2.0",
  "cookie-parser": "^1.4.5",
  "csurf": "^1.11.0",
  "express": "^4.17.1",
  "express-handlebars": "^4.0.3",
  "express-session": "^1.17.1",
  "express-validator": "^4.3.0",
  "lodash": "^4.17.15",
  "mongoose": "^5.9.10",
  "passport": "^0.4.1",
  "passport-local": "^1.0.0",
  "stripe": "^8.49.0"
},
"devDependencies": {
  "jest": "^26.0.1",
  "nodemon": "^2.0.3"
}
```

Рисунок 3.12 – Файл *package.json* (залежності)

Нижче на рисунку 3.13 можна побачити назву проекту, його версію та опис. Важливу роль грають строки “main” – вказує на головний файл проекту та “scripts” – скрипти для роботи з проектом.

```
{
  "name": "online_shore_app",
  "version": "1.0.7",
  "description": "Graduate  work. Online store app project. /NodeJS/ExpressJS/MongoDB",
  "main": "app.js",
  "scripts": {
    "start": "node app",
    "dev": "nodemon app",
    "test": "jest"
  },
```

Рисунок 3.13 – Файл *package.json*

Після ініціалізації створюється головний файл веб-додатку, який і буде запускати ввесь проект. Таким файлом є *app.js* у якому реалізована серверна частина веб-додатку за допомогою Express.js та підключення до бази даних MongoDB. Додаткові конфігурації для даного файлу зберігаються у папці *config* у файлі *default.json.*

Коли настройка веб-серверу та бази даних завершена, слід організувати взаємодію серверу з базою даних. Для цього у папці *models* створюються моделі взаємодії з базою даних. Про моделі веб-додатку було докладно сказано у попередньому розділі.

Важливим компонентом веб-додатку є організація логіки веб-серверу. Вона реалізовується з допомогою маршрутів звернення до серверу та контроллерів, що виконують запити. Даний компонент реалізовано у папці *routes*, вміст якої нижче на рисунку 3.14. У даній папці присутні чотири основні файли маршрутів, у яких і прописана вся логіка веб-серверу.

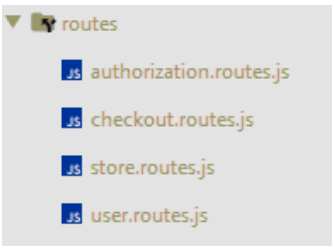


Рисунок 3.14 – Папка *routes*

Дані файли підключаються до основного файлу *app.js* та реалізують основні URL-маршрути веб-додатку. Те, як вони підключаються нижче на рисунку 3.15.

```
// Routes
app.use('/store/authorization', AuthorizationRouter); // authorization uri
app.use('/store/user', UserRouter); // user uri
app.use('/store/checkout', CheckoutRouter); // checkout uri
app.use('/store', StoreRouter); // store uri
```

Рисунок 3.15 – Routes

Маршрут **/store/authorization** відповідає за організацію реєстрації та входу у веб-додатку.

Маршрут **/store/user** відповідає за організацію маршрутів для користувача веб-додатку, таких як сторінка профілю, редагування інформації, додавання товарів та інших.

Маршрут **/store/checkout** відповідає за організацію оформлення замовлення та покупки товарів у веб-додатку.

Маршрут **/store** відповідає за організацію основної частини веб-додатку (домашня сторінка, сторінка товарів та товару та інші).

З серверною частиною веб-додатку завершено. Тепер переходимо до сторони клієнта. За роботу даної частини у Express.js відповідає шаблонізатор **Handlebars**, який є прекрасно організовує взаємодію між клієнтом та сервером.

Всі клієнтські сторінки реалізовані у папці *views*, а статичні файли **CSS** та **JavaScript** у папці *public*.

Вміст папки *public* зображений нижче на рисунку 3.16.

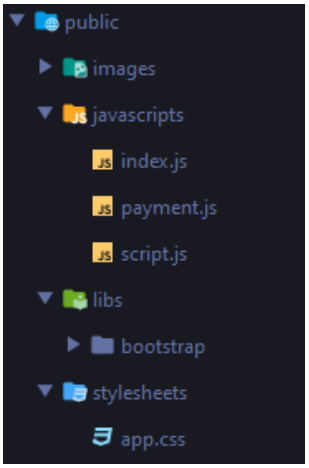


Рисунок 3.16 – Папка *public*

У даній папці зберігаються файли, що необхідні для реалізації веб-сторінок клієнтської частини веб-додатку. Файл *app.css* є основним **CSS** файлом у якому реалізовані стилі компонентів веб-сторінок. Варто виділити файл *payment.js*, який необхідний для здійснення покупки товарів, решту JS файлів необхідні для front-end. У папці *libs* зберігають файли фреймворку **Bootstrap.**

Вміст папки *views* представляє собою набір папок, що розбиті на компонентні частини клієнтської сторони веб-додатку (рис.3.17). Важливу роль тут відіграє шаблонізатор **Handlebars**, який дозволяє розбивати HTML-код на компоненти та використовувати за потреби у HTML-документах. Також він допомагає працювати з даними, що передаються з серверу. Його настройка та підключення було здійснено у файлі *app.js*. А основний файл, що організовує його роботу знаходить у папці *views/layouts/layout.hbs.*

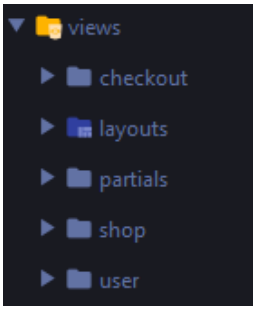


Рисунок 3.17 – Папка *views*

У папці *partials* присутні основні компоненти веб-сторінок, які показані на рисунку 3.18.

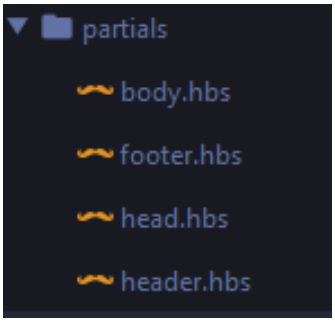


Рисунок 3.18 – Компоненти папки *partials*

У папці *checkout* присутні основні HTML-документи, які відповідають за представлення веб-сторінок оформлення замовлення. Вони показані нижче на рисунку 3.19.

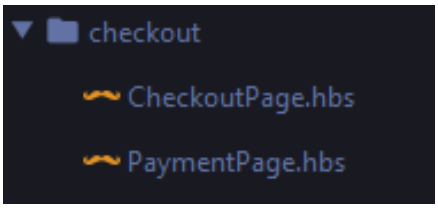


Рисунок 3.19 – HTML-документи папки *checkout*

У папці *user* присутні основні HTML-документи, які відповідають за представлення веб-сторінок для користувачів та показані на рисунку 3.20.

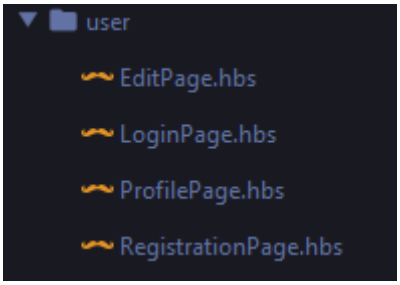


Рисунок 3.20 – HTML-документи папки *user*

У папці *shop* присутні основні HTML-документи, які відповідають за представлення веб-сторінок торгової площадки. Показані дані документи нижче на рисунку 3.21.

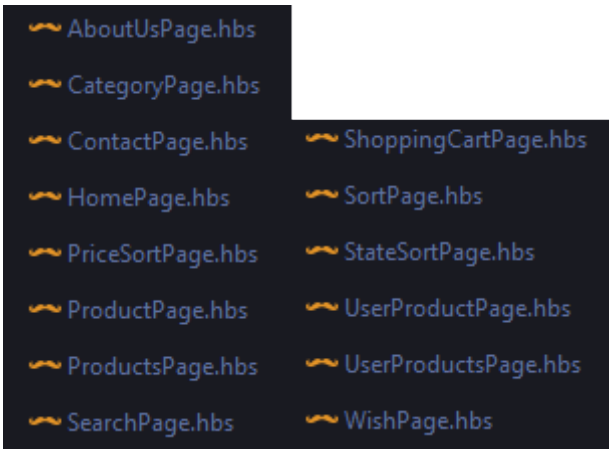


Рисунок 3.21 – HTML-документи папки *shop*

Програмна реалізація веб-додатку завершена. Були реалізовані серверна сторона веб-додатку та сторона клієнта.

3.4. Розробка інтерфейсу та функціональності інтернет-платформи для комерційного майданчика

Для того, щоб перевірити роботу веб-додатку, потрібно проаналізувати цілі, які ми хотіли досягти, розробляючи дану торгову площадку. Виходячи з аналізу вимог до проекту, який був сформований у розділі 1.4, варто скласти чек-лист для тестування веб-додатку:

1) Користувач відвідує домашню сторінку веб-сервісу;
2) Переходить на сторінку асортименту та додає уподобаний товар у корзину;
3) Після чого може зайти на сторінку корзини або зразу на сторінку оформлення замовлення;

4) Якщо користувач не авторизований, то його переадресує на сторінку Входу, звідки він також може перейти на сторінку реєстрації;
5) Після здійснення даної операції, користувача переадресовує на сторінку оформлення замовлення, де він може здійснити оформлення замовлення без онлайн оплати, або перейти на сторінку онлайн оплати;.
6) Після оформлення замовлення користувач буде переадресований на домашню сторінку, звідки може перейти на сторінку особистого профілю;
7) Сторінка профілю дозволяє користувачу переглянути оформлене замовлення та здійснювати вже власну реалізацію товарів через веб-сервіс.

Проведення тестування по кожному пункту чек-листа гарантуватиме успішну реалізацію всіх основних компонентів веб-додатку.

Головна сторінка веб-додатку показана на рисунку 3.22. Це є домашня сторінка, яка презентує веб-додаток та розказує про його можливості (як ним користуватись). На домашній сторінці є все необхідне для навігації по веб-додатку. Навігація здійснюється за допомогою навігаційного бару (header) веб-додатку та нижньої панелі веб-додатку (footer). Дані навігаційні компоненти веб-додатку будуть доступні на кожній веб-сторінці, оскільки здійснюючи переходи на інші веб-сторінки змінюється лише наповнення у залежності від призначення.

Навігаційний бар також містить поле для пошуку, яке буде доступне з будь-якої частини веб-додатку. Пошук здійснюється серед всього асортименту веб-сервісу та товарів користувачів. Дана функція реалізована для швидкого пошуку потрібних товарів користувача, що значно покращить їхню взаємодію з веб-додатком. Достатньо ввести ключове слово і здійснити пошук по ньому. Після чого будуть представлені результати пошуку.

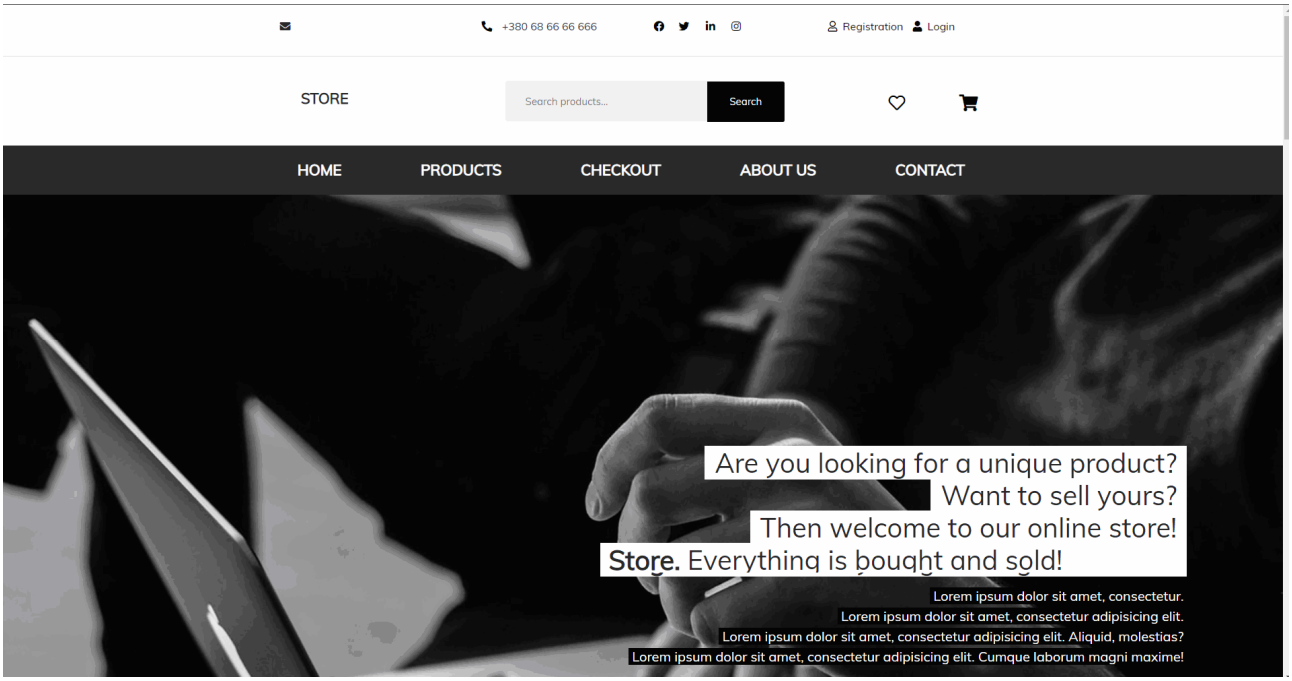


Рисунок 3.22 – Головна сторінка сторінка інтернет-платформи для комерційного майданчика

Навігаційна верхня навігаційна панель дозволяє отримати доступ до інших веб-сторінок. Звіди можна здійснити перехід на сторінку: авторизації, реєстрації, корзини, списку бажань, домашню сторінку, сторінку з продукцією, оформлення замовлення, про торгову площадку та сторінку для контакту з сервісом. Нижня навігаційна панель надає те саме, але також і деяку інформацію для користувача.

Сторінка продукції представляє собою список асортименту торгової площадки і зображена на рисунку 3.23. Вона дає можливість здійснювати маніпуляції з товарами. Можна показати товари за категорією, відсортувати за вибраними параметрами, показати товари за їхнім станом (нові або вживані) та відфільтрувати за ціновими рамками. Товари представляють собою список карточок. Карточка товару містить зображення товару, його назву, ціну та позначку, що він є товаром веб-сервісу. Також карточка має кнопки, що надають додати товар у корзину або список бажань.

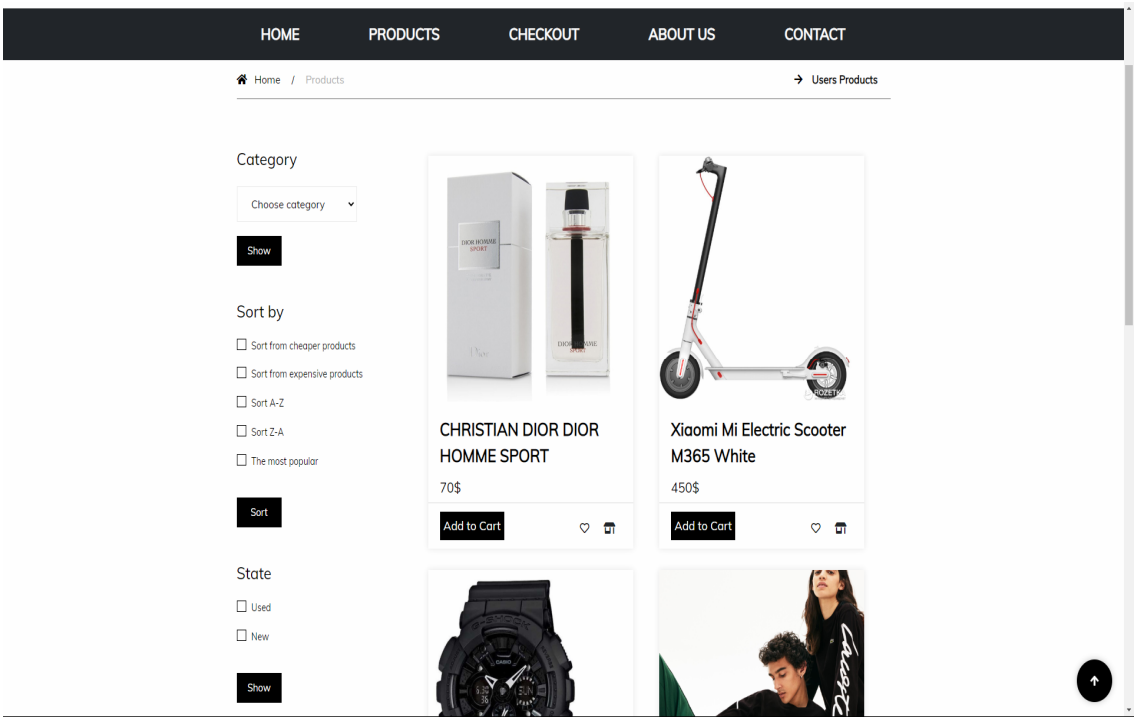


Рисунок 3.23 – Сторінка з переліком товарів

Також карточка дає можливість детальніше оглянути товар. Для цього слід натиснути на назву товару і перейти на його сторінку, де можна прочитати опис, оцінити товар, побачити його стан та наявність. Продемонстровано на рисунку 3.24.

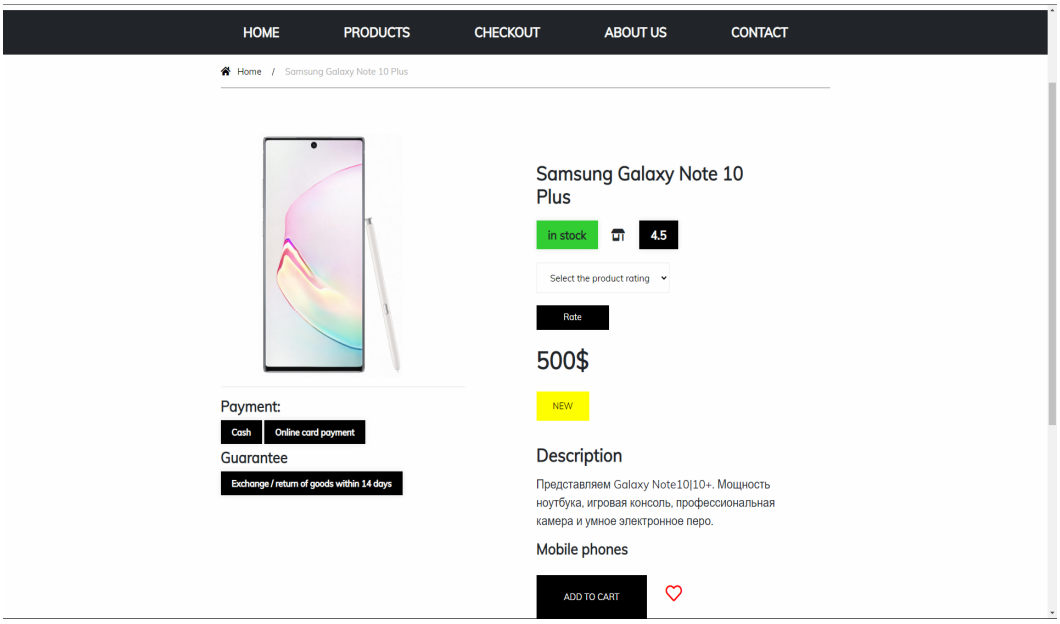


Рисунок 3.24 – Сторінка товару

Також можна здійснити перехід на сторінку з товарами від користувачів веб-сервісу, де будуть доступні ті самі можливості. Дана сторінка зображена на рисунку 3.25 нижче.

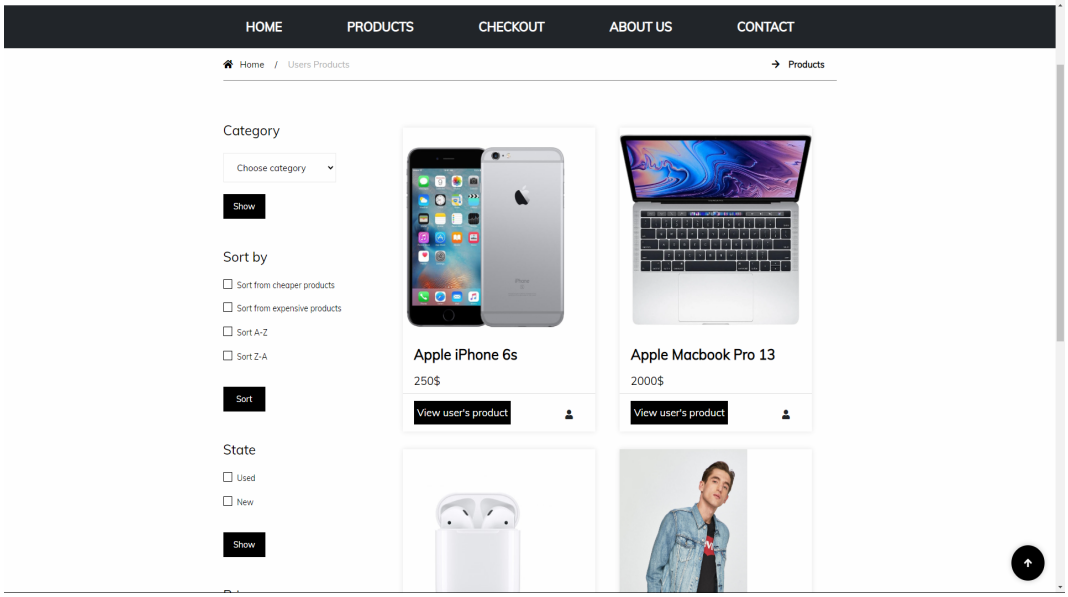


Рисунок 3.25 – Сторінка товарів користувачів

Здійснивши перехід на сторінку товару від користувача, показано нижче на рисунку 3.26, можна побачити інформацію про власника товару (дозволить зв’язатися з ним для покупки) та сам товар.

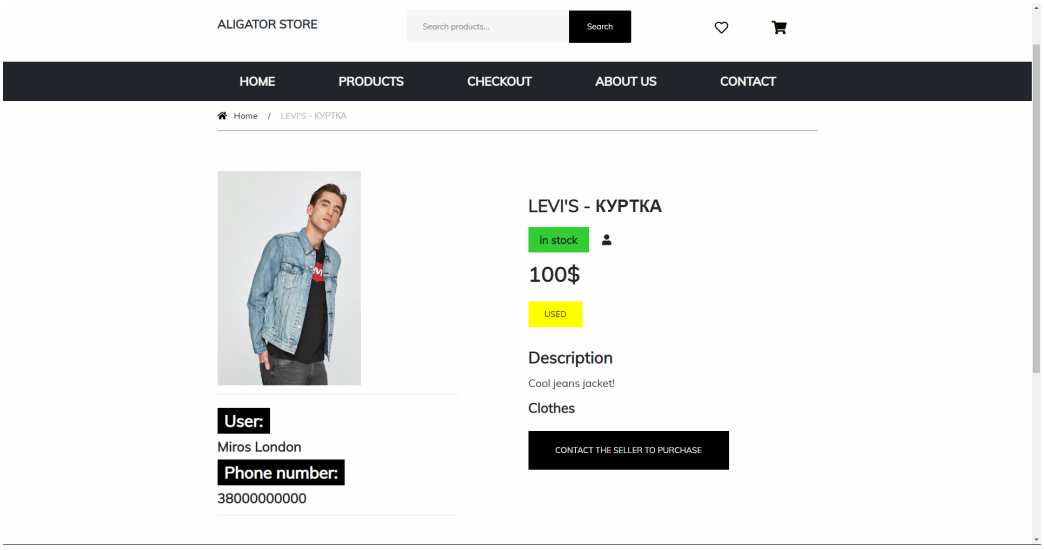


Рисунок 3.26 – Сторінка товару користувача

Кошик, який показано на рисунку 3.27, виконує функцію зберігання товарів, які користувач має намір купити. Користувач додає потрібний товар(ри) у корзину, де вони відображаються як елементи таблиці. Тут показується кількість товарів, їх ціна та загальна ціна товарів корзини. Товари з корзини можна вилучати по одному або вилучити всі разом. Звідси можна перейти на сторінку оформлення замовлення.

Список бажань являється веб-сторінкою на якій відображаються товари, які уподобав користувач.

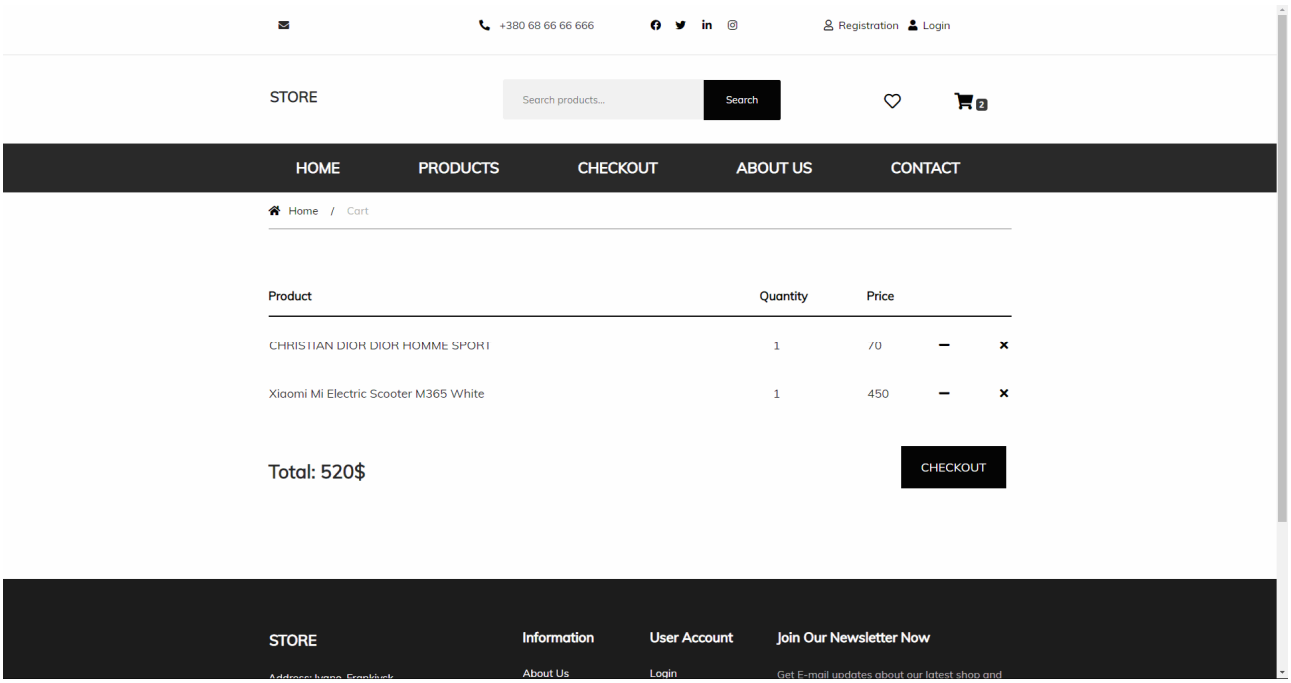


Рисунок 3.27 – Вигляд кошика з товарами

Сторінка оформлення замовлення показана на рисунку 3.28. Вона надає можливість оформити два типи замовлення. Замовлення, яке оформлюється без онлайн оплати і буде доставлене на поштове відділення та замовлення та яке буде оплачене через онлайн платіж за допомогою кредитної карти. Реалізація двох способів є необхідною для того, щоб задовільнити потреби всіх користувачів. Для контролю здійснених у веб-сервісі покупок використовується

сервіс для здійснення електронних платежів – Stripe. Він є популярним рішенням задач пов’язаних з електронною оплатою.

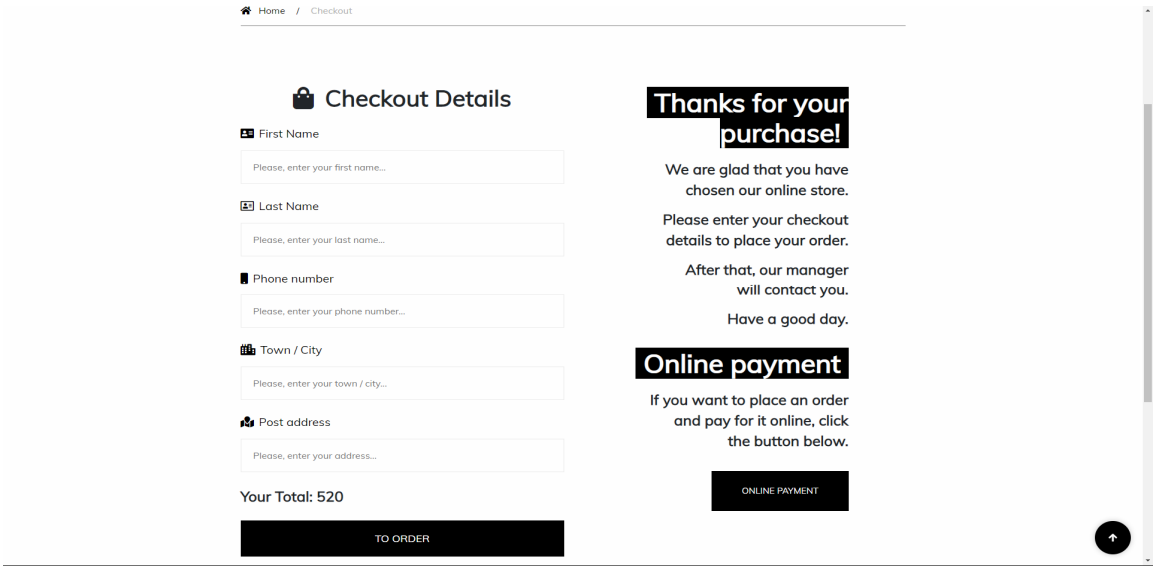


Рисунок 3.28 – Сторінка оформлення замовлення

Зі сторінки оформлення замовлення можна перейти на сторінку онлайн оплати. Після оформлення замовлення та оплати, користувача переведе на домашню сторінку. Після чого він зможе переглянути замовлення у власному профілі. Демонстрація даної сторінки нижче на рисунку 3.29.

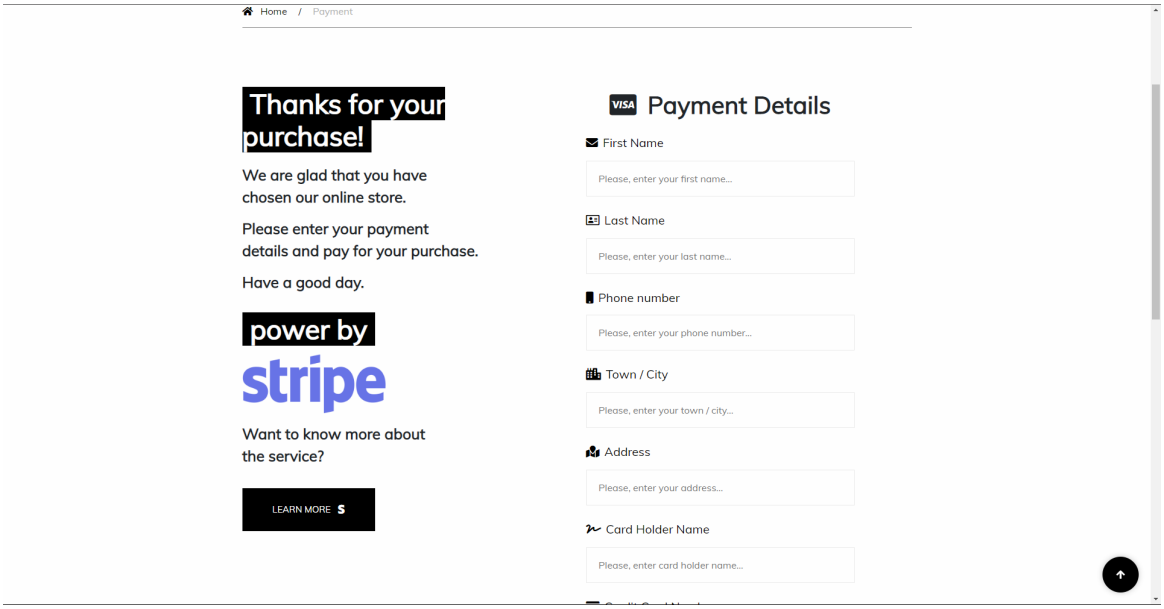


Рисунок 3.29 – Сторінка онлайн оплати

Для здійснення покупок необхідно бути зареєстрованим у веб-сервісі. Для цього реалізовані сторінка реєстрації, яка показана на рисунку 3.30.

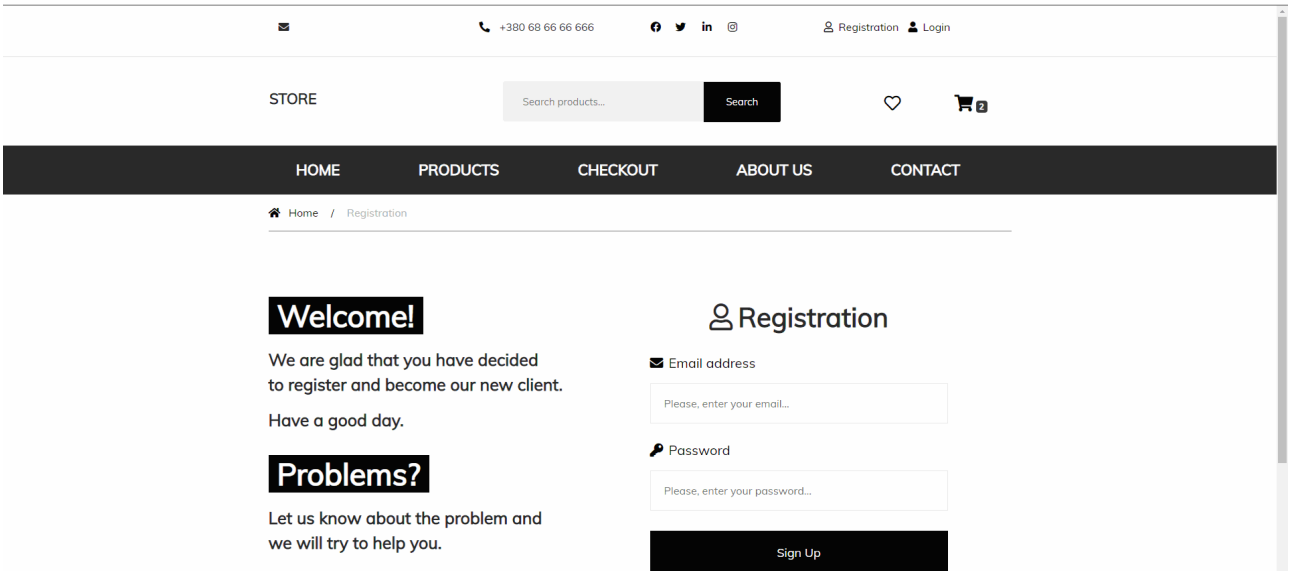


Рисунок 3.30 – Сторінка реєстрації

Якщо користувач вже зареєстрований у веб-сервісу, він може здійснити авторизацію зі сторінки входу, яка продемонстрована нижче на рисунку 3.31.

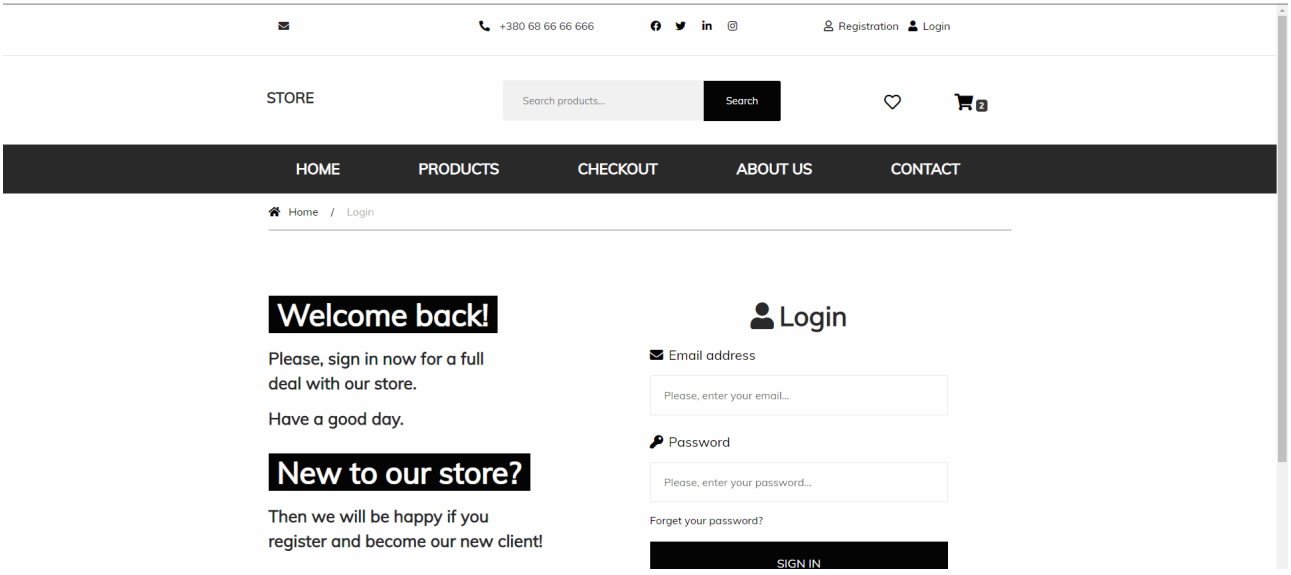


Рисунок 3.31 – Сторінка входу

Після авторизації користувачу буде доступна сторінка профілю, яка показана на рисунку 3.32.

Тут можна переглянути список товарів, які було замовлено та товари, які користувач уже додав до веб-сервісу. Користувач може додати товар, видали, редагувати його.

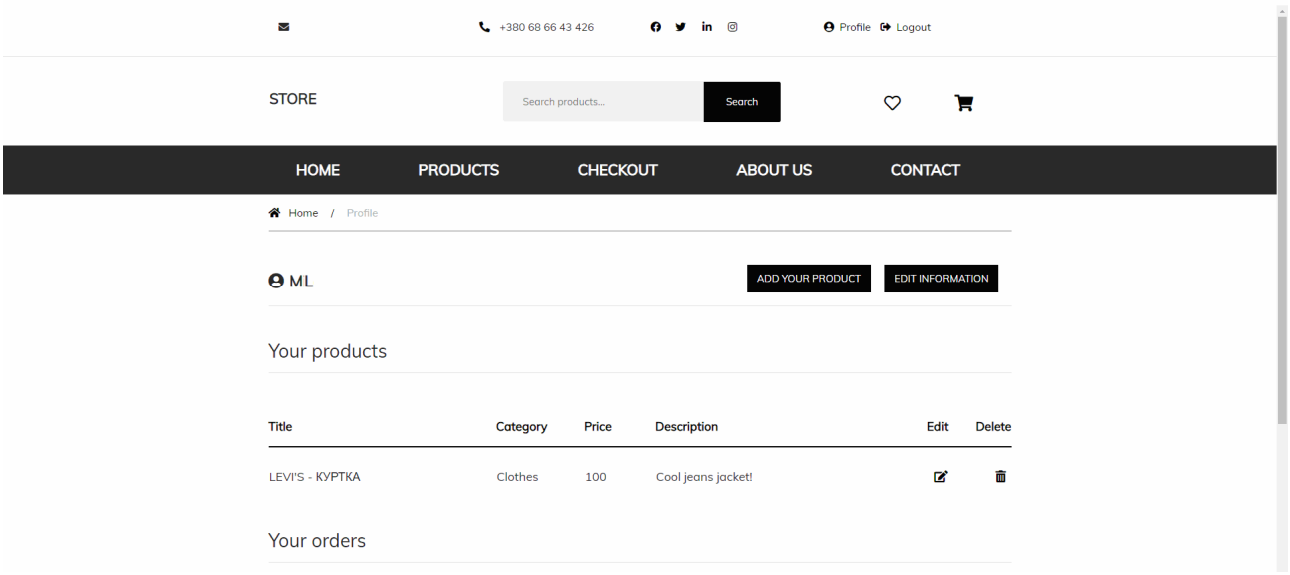


Рисунок 3.32 – Профіль користувача

Також користувачу доступна сторінка редагування інформації, де він може після реєстрації додати такі дані, як ім’я, прізвище та номер телефону, які будуть необхідні йому для успішного продажу власних товарів на торговій площадці. Наявність форми для додавання нової інформації зумовлена тим, що при реєстрації користувач використовує лише власну електронну пошту та вибраний ним пароль. Також можливості здійснити редагування вже існуючої інформації. Це є хорошою практикою, адже користувач у будь-який момент може здійснити редагування потрібних йому даних у веб-сервісі. Показана дана сторінка нижче на рисунку 3.33.

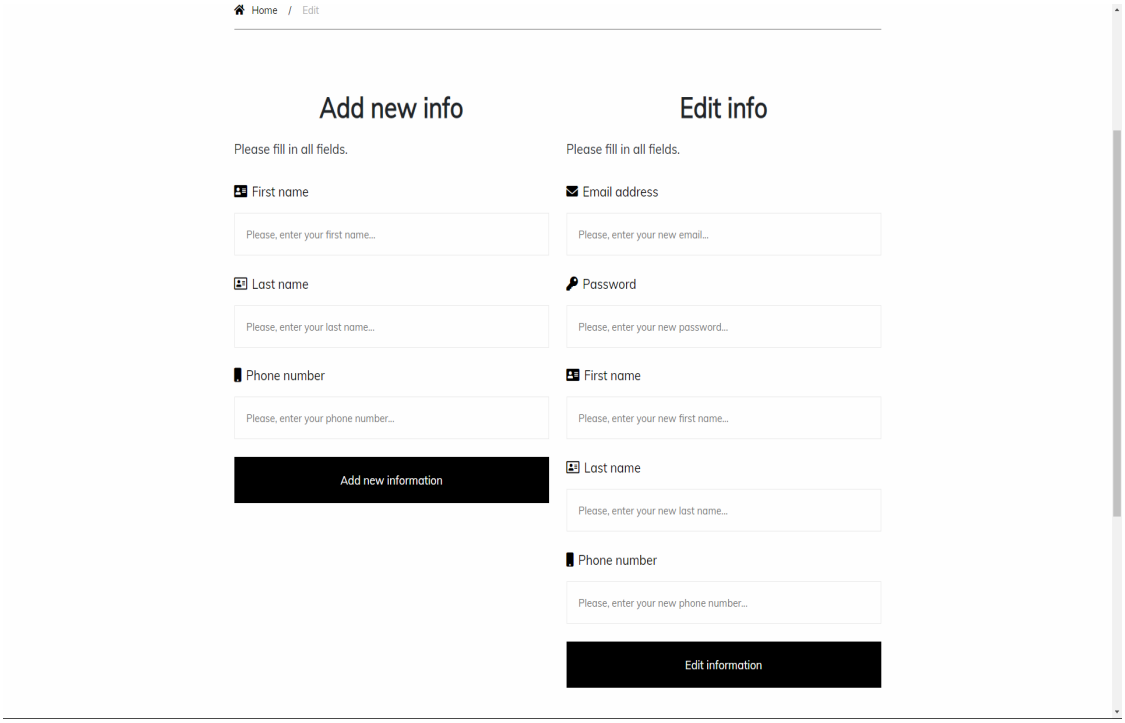


Рисунок 3.33 – Сторінка редагування

Також для користувачів є можливість зв’язатися з веб-сервісом відправивши лист через форму, яка продемонстрована на рисунку 3.34.

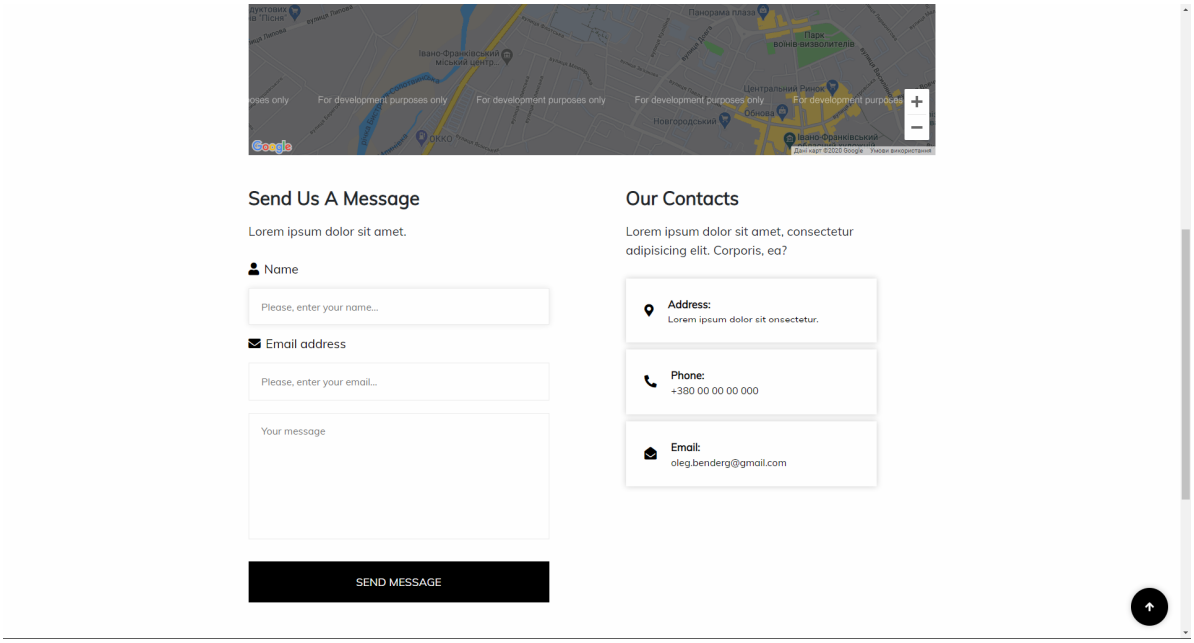


Рисунок 3.34 – Сторінка контакту з веб-сервісом

Якщо користувач має бажання отримувати новини від торгової площадки, то може підписатись на розсилку, ввівши потрібну електронну пошту у відповідну форму, що показана на рисунку 3.35.

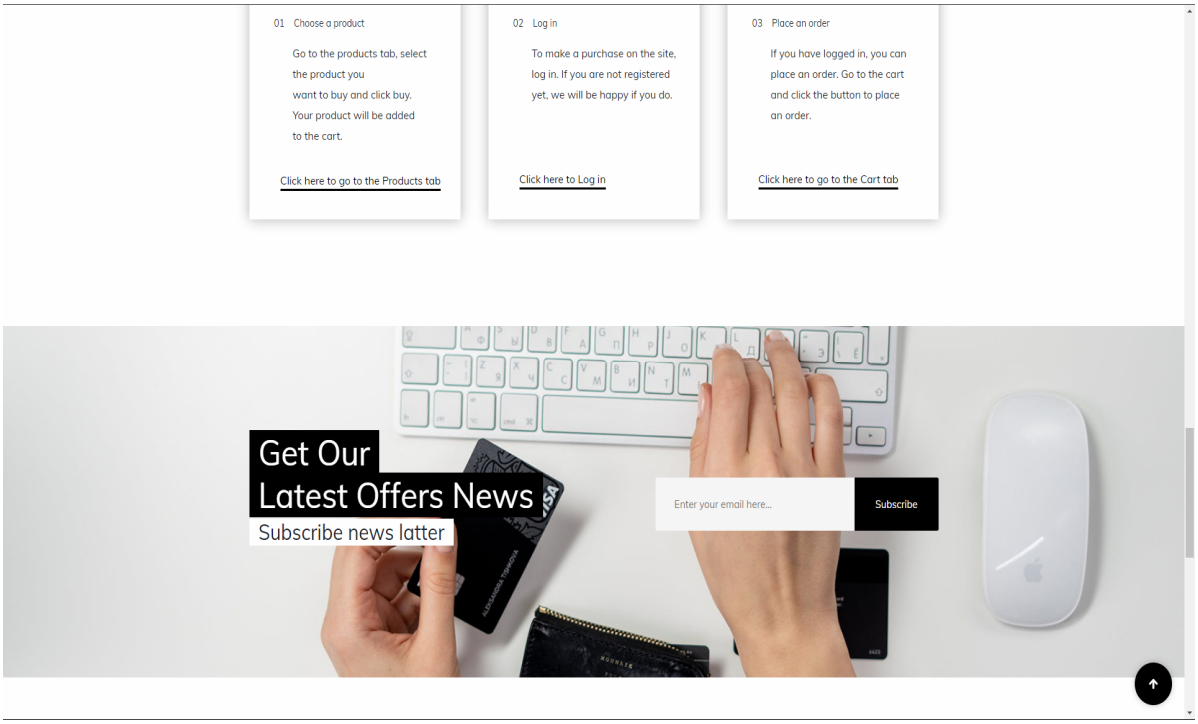


Рисунок 3.35 – Підписка на розсилку новин

Отже, тестування веб-сервісу показало, що вимоги, які були зазначено у списку виконані, а результати виконання було продемонстровано з детальним описом взаємодії з веб-додатком.

3.5. Висновки до розділу

У даному розділі було здійснено алгоритмічне та програмне рішення задачі щодо проекту даної роботи. Було здійснено проектування структури веб-додатку та структури бази даних. Була показана програмна реалізація веб-сервісу за допомогою вибраних технологій. Також був складений чек-лист тестування веб-додатку, що показав успішну реалізацію поставлених цілей.

ВИСНОВКИ

В представленій дипломній роботі виконано створення інтернет-платформи для комерційного майданчика. Було розглянуто велику кількість вирішень поставленої задачі і вибрано актуальний напрям сучасних комерційних майданчиків. Було визначено, які потреби користувачів мають виконувати такі веб-сервіси і який функціонал надавати. Було розроблено вимоги до проекту і те, яку цільову функцію він повинен виконувати, вибрано ринок та цільову аудиторію, яку дана торгова площадка повинна охопити. Було визначено поступові кроки реалізації та ресурси, що для цього необхідні. Виходячи з цього, було розроблено графічний інтерфейс, який у подальшому слід реалізувати і який відповідає всім сучасним вимогам і стандартам. Також визначено необхідні моделі та архітектуру, які використовуватимуться під час розробки.

В роботі досліджено сучасні технології веб-програмування і вибрано ті, які відповідають потребам веб-сервісу. Для реалізації проекту були вибрані актуальні на даний момент часу технології: мова програмування JavaScript та її багатий вибір серед фреймворків та інструментів, таких як Express.js, Lodash, Jest та інші, платформа Node, NoSQL база даних MongoDB, HTML, CSS та Bootstrap. Після чого було розроблено готовий веб-сервіс з повноцінною клієнтською та серверною сторонами. Вибраний стек технологій реалізації задовольняє потреби у хорошій масштабованості та підтримці.

Розроблений веб-сервіс пройшов перевірку тестуванням на рахунок реалізації визначених вимог. Було визначено, що вибраний стек технологій задовольняє всі потреби під час розробки, він надає всі необхідні інструменти та способи реалізації поставлених задач. Тому виконання даного завдання можна вважати успішним, а отриманий результат є прикладом хорошого використання сучасних веб-технологій та прогресу веб-програмування в цілому.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Веб-програмування [Електронний ресурс]: http://apeps.kpi.ua/web-programuvania

2. Single-page application vs. multiple-page application. - URL: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58

3. Що таке електронна комерція?: https://www.interkassa.com/ua/blog/chto-takoe-elektronnaya-kommerciya-e-commerce-dlya-nachinayushchih/

4. Клімушин П.С. Інформаційні системи та технології в економіці : навч. посіб. / П.С. Клімушин, О.В. Орлов, А.О. Серенок. – Х.: Вид-во ХарРІ НАДУ „Магістр”, 2011. –448 с.

5. Макарова М.В. Тенденції розвитку цифрової економіки: Монографія. – Полтава: РВВ ПУСКУ, 2004. – 240 с.

6. Суркова Н. Е., Остроух А. В. Методологія структурного проєктування інформаційних систем. Харків: Науково-інноваційний центр, 2014. 184 с.

7. Тітов, С. В., & Тітова, О. В. (2015). Оцінка юзабіліті освітніх сайтів: методи і технології. Вісник Харківської державної академії культури. Серія: Соціальні комунікації, (47), 127-134.

8. Iryna, T., & Heorhii, M. (2021). Research of regression and modular testing of web applications. Editorial Board, 406.

9. Tvoroshenko, I. S., & Maksimenko, H. (2021). To the question of analysis of existing mechanisms of web application testing.

10. Kuzomin, O., Tolmachova, T., & Astappiev, O. (2017). Analysis of Web user activity data. International Journal of Information Models and Analyses, 6(2), 108-118.

11. Rawat, P., & Mahajan, A. N. (2020). ReactJS: A modern web development framework. International Journal of Innovative Science and Research Technology, 5(11), 698-702.

12. Ranjan, A., Sinha, A., & Battewad, R. (2020). JavaScript for Modern Web Development: Building a Web Application Using HTML, CSS, and JavaScript. BPB Publications.

13. Makris, A., Tserpes, K., Spiliopoulos, G., Zissis, D., & Anagnostopoulos, D. (2021). MongoDB Vs PostgreSQL: A comparative study on performance aspects. GeoInformatica, 25, 243-268.

14. Жулкевський, В. (2020). Використання архітектурного підходу Flux для побудови веб-застосунків на прикладі бібліотеки Redux.

15. Hoque, S. (2020). Full-Stack React Projects: Learn MERN stack development by building modern web apps using MongoDB, Express, React, and Node. js. Packt Publishing Ltd.

16. Єльчанінов, Д. Б., Косіло, М. С., & Бєлова, Н. В. (2014). Технології автоматизації проектування програмних систем. Системи обробки інформації, (6), 135-140.

17. Iryna, T., & Maksym, K. (2021). Research results of functional, white box and smoke testing methods for mobile applications. Trends in science and practice of today, 5, 418.

18. Mohammadi, R. Performance, and Efficiency based Comparison of Angular and React in a case study of Single page application (SPA) (Doctoral dissertation, Dissertation 2020, Herat University, https://scholar. google. com/scholar)

19. Kornienko, D. V., Mishina, S. V., & Melnikov, M. O. (2021, November). The Single Page Application architecture when developing secure Web services. In Journal of Physics: Conference Series (Vol. 2091, No. 1, p. 012065). IOP Publishing

ДОДАТКИ

Додаток А

Лістинг програми

```
const express = require('express'); // express.js library
const mongoose = require('mongoose'); // mongoose for work with
MongoDB
const config = require('config'); // config library
const cookieParser = require('cookie-parser'); // cookie-parser
library to work with cookie
const bodyParser = require('body-parser');
const session = require('express-session'); // express-session
library to work with session
const passport = require('passport'); // Passport is
authentication middleware for Node.js.
const flash = require('connect-flash'); // flash messages
const expressValidator = require('express-validator'); // forms
validation
const expressHandlebars = require('express-handlebars'); //
presentation engine for best working with views
const MongoStore = require('connect-mongo')(session); // for
session
// Routes initialization
const StoreRouter = require('./routes/store.routes');
const AuthorizationRouter =
require('./routes/authorization.routes');
const UserRouter = require('./routes/user.routes');
const CheckoutRouter = require('./routes/checkout.routes');
require('./config/passport'); // passport config

const app = express();
const PORT = config.get('port') || 5000; // server port
const handlebars = expressHandlebars.create({
    defaultLayout: 'layout',
    extname: 'hbs'
}); // Express Handlebars initialization

app.engine('hbs', handlebars.engine);
app.set('view engine', 'hbs');
app.set('views', 'views');

app.use(express.json({ extended: true })); // for using json
app.use(express.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(expressValidator()); // forms validation
app.use(cookieParser()); // for using cookies
app.use(session({ // session settings
    secret: 'storeappsecret',
    resave: false,
    saveUninitialized: false,
```

```
store: new MongoStore({ mongooseConnection: mongoose.connection
}), // connect session to the database
    cookie: { maxAge: 180 * 60 * 1000 }
}));
app.use(flash()); // flash messages
app.use(passport.initialize()); // passport initialize
app.use(passport.session()); // passport access to the session
// public folder connection
app.use(express.static(__dirname + '/public'));

// global property which available in views
app.use((req, res, next) => {
   res.locals.login = req.isAuthenticated();
   res.locals.session = req.session;
   next();
});
// Routes
app.use('/store/authorization', AuthorizationRouter); //
authorization uri
app.use('/store/user', UserRouter); // user uri
app.use('/store/checkout', CheckoutRouter); // checkout uri
app.use('/store', StoreRouter); // store uri

// async function start to connect to the database and start
server
const start = async () => {
    try {
        await mongoose.connect(config.get('mongoURI'), {
            useNewUrlParser: true,
            useUnifiedTopology: true,
            useCreateIndex: true,
            useFindAndModify: false
        });
        app.listen(PORT, () => console.log(`App has been started
on port ${PORT}...`));
    } catch (e) {
        console.log('Server Error', e.message);
        process.exit(1);
    }
};
// call function which start server
start();
const { Router } = require('express'); // express router for work
with routes
const router = Router();
const Product = require('../models/Product'); // import Product
schema
const UserProduct = require('../models/UserProduct'); // import
User schema
const Cart = require('../models/Cart'); // import Cart schema
const Wish = require('../models/Wish');
const User = require('../models/User');
```

```
const _ = require('lodash');
const Contact = require('../models/Contact');
const Subscribe = require('../models/Subscribe');

// get home page // /store/home
router.get('/home', async (req, res) => {
    try {
        const successMsg = req.flash('success')[0];
        const errMsg = req.flash('error')[0];
        await res.render('shop/HomePage', { title: ' Store |
Home', errMsg: errMsg, noError: !errMsg, successMsg: successMsg,
noMessages: !successMsg });
    } catch (e) {
        console.log({ message: e });
    }
});
// get products from db and products page // /store/products
router.get('/products', async (req, res, next) => {
    try {
        const products = await Product.find({}).lean();
        await res.render('shop/ProductsPage', { title: ' Store |
Products', products });
    } catch (e) {
        console.log({ message: e });
    }
});
// get user products page // /store/users-products
router.get('/users-products', async (req, res) => {
    try {
        const userProducts = await UserProduct.find({}).lean();
        await res.render('shop/UserProductsPage', { title: ' Store
| Users Products', userProducts });
    } catch (e) {
        console.log({ message: e });
    }
});
router.get('/product/:id', async (req, res) => {
    try {
        const successMsg = req.flash('success')[0];
        const errMsg = req.flash('error')[0];

        const avarageRating = await Product.find({ _id:
req.params.id }).lean();
        const getAvarager = () => {
            for (let keys in avarageRating) {
                let rating = avarageRating[keys]
                return rating.rating
            }
        }
        const avarage = _(getAvarager()).mean(function (r) {
   return r.toFixed(1)
        })
```

```
        const singleProduct = await Product.find({ _id:
req.params.id }).lean();
        res.render('shop/ProductPage', { title: ' Store |
Product', avarage ,singleProduct, errMsg: errMsg, noError:
!errMsg, successMsg: successMsg, noMessages: !successMsg });
    } catch (e) {
        console.log({ message: e });
    }
});
router.post('/product-rating/:id', async (req, res) => {
    try {
        const productID = req.params.id;
        const query = req.body.rating_select;
        const productRating = {
            $push: {
                "rating": query
            }
        };
        await Product.findByIdAndUpdate(productID,
productRating,(err) => {
            if (err) {
                return req.flash('error', 'Сталася помилка при
оцінюванні, спробуйте ще раз');
            }
            req.flash('success', 'Товар оцінено');
            res.redirect(req.get('referer'));
        });
    } catch (e) {
        console.log({ message: e });
    }
});
router.get('/user-product/:id', async (req, res) => {
    try {
        const singleUserProduct = await UserProduct.find({ _id:
req.params.id }).lean();
        const getUserId = () => {
            for (let keys in singleUserProduct) {
                let user = singleUserProduct[keys]
                return user.user
            }
        }
        const userID = getUserId();
        const userProductInfo = await User.find({ _id: userID
}).lean();
        res.render('shop/UserProductPage', { title: ' Store | User
Product', userProductInfo, singleUserProduct });
    } catch (e) {
        console.log({ message: e });
});
// get about-us page // /store/about-us
router.get('/about-us', async (req, res) => {
```

```
    try {
        await res.render('shop/AboutUsPage', { title: ' Store |
About Us' });
    } catch (e) {
        console.log({ message: e });
    }
});

// get blog page // /store/blog
router.get('/contact', async (req, res) => {
    try {
        const successMsg = req.flash('success')[0];
        const errMsg = req.flash('error')[0];
        await res.render('shop/ContactPage', { title: ' Store |
Contact', errMsg: errMsg, noError: !errMsg, successMsg:
successMsg, noMessages: !successMsg });
    } catch (e) {
        console.log({ message: e });
    }
});

router.post('/contact', async (req, res) => {
    try {
        const userMessage = new Contact({
            name: req.body.name,
            email: req.body.email,
            message: req.body.message
        });

        await userMessage.save((err) => {
            if (err) {
                return req.flash('error', 'Сталася помилка при
відправленні, спробуйте ще раз');
            }
            req.flash('success', 'Дякуємо за ваше повідомлення, ми
постараємся дати Вам відповідь у найближчий час');
            res.redirect('/store/contact');
        });
    } catch (e) {
        console.log({ message: e });
    }
});
router.post('/subscribe', async (req, res) => {
    try {
        const subNews = new Subscribe({
            email: req.body.email
        });
```

Продовження додатку

```
 await subNews.save((err) => {
            if (err) {
                return req.flash('error', 'Сталася помилка при
відправленні запиту на підписку, спробуйте ще раз');
```

```
            }
            req.flash('success', 'Дякуємо за вашу підписку');
            res.redirect('/store/home');
        });
    } catch (e) {

    }
});
router.get('/add-to-wish/:id', async (req, res) => {
    try {
        const id = req.params.id;
        const wish = new Wish(req.session.wish ? req.session.wish
: {});

        await Product.findById(id, (err, product) => {
            if (err) {
                return res.redirect('/store/home');
            }

            wish.addToWish(product, product.id);
            req.session.wish = wish;
            console.log(req.session.wish);
            res.redirect(req.get('referer'));
        });
    } catch (e) {
        console.log({ message: e });
    }
});

router.get('/wish', async (req, res) => {
    try {
        if (!req.session.wish) {
            return res.render('shop/WishPage', { title: ' Store |
Wish', products: null });
        }

        const wish = await new Wish(req.session.wish);
        res.render('shop/WishPage', { title: ' Store | Wish',
products: wish.generateArray(), totalPrice: wish.totalPrice });
    } catch (e) {
        console.log({ message: e });
    }
});
router.get('/remove-wish/:id', async (req, res, next) => {
    try {
        const id = req.params.id;
        const wish = new Wish(req.session.wish ? req.session.wish
: {});

        await wish.removeItem(id);
        req.session.wish = wish;
        res.redirect('/store/wish');
```

```
    } catch (e) {
        console.log({ message: e });
    }
});

// get route for add product to shopping cart by id // /store/add-
to-cart
router.get('/add-to-cart/:id', async (req, res) => {
     try {
         const productId = req.params.id;
         const cart = new Cart(req.session.cart ? req.session.cart
: {});

         await Product.findById(productId, (err, product) => {
             if (err) {
                 return res.redirect('/store/home');
             }

             cart.addToCart(product, product.id);
             req.session.cart = cart;
             console.log(req.session.cart);
             res.redirect(req.get('referer'));
         });
     } catch (e) {
        console.log({ message: e });
     }
});

// get reduce method which reduce one item from cart by id //
/store/reduce
router.get('/reduce/:id', async (req, res) => {
    try {
        const productId = req.params.id;
        const cart = new Cart(req.session.cart ? req.session.cart
: {});

        await cart.reduceByOne(productId);
        req.session.cart = cart;
        res.redirect('/store/shopping-cart');
    } catch (e) {
        console.log({ message: e });
    }
});
router.get('/remove/:id', async (req, res, next) => {
    try {
const productId = req.params.id;
        const cart = new Cart(req.session.cart ? req.session.cart
: {});

        await cart.removeItem(productId);
        req.session.cart = cart;
        res.redirect('/store/shopping-cart');
```

```
    } catch (e) {
        console.log({ message: e });
    }
});

// get shopping-cart page // /store/shopping-cart
router.get('/shopping-cart', async (req, res) => {
    try {
        if (!req.session.cart) {
            return res.render('shop/ShoppingCartPage', { title: '
Store | Cart', products: null });
        }

        const cart = await new Cart(req.session.cart);
        res.render('shop/ShoppingCartPage', { title: ' Store |
Cart', products: cart.generateArray(), totalPrice: cart.totalPrice
});
    } catch (e) {
        console.log({ message: e });
    }
});
// get search page where we can see search results //
/store/search
router.get('/search', async (req, res) => {
    try {
        const successMsg = req.flash('success')[0];
        const errMsg = req.flash('error')[0];

        if(req.query.search) {
            const regex = new
RegExp(escapeRegex(req.query.search), 'gi');
            const filteredProducts = await Product.find({ title:
regex }, (err, res) => {
                if (err) {
                    console.log(err);
                    res.redirect('/store/search');
                    return req.flash('error', 'Сталася помилка при
пошуку, спробуйте ще раз');
                }
            }).lean();
            const filteredUserProducts = await UserProduct.find({
title: regex }, (err, res) => {
                if (err) {
                    console.log(err);
                    res.redirect('/store/search');
                    return req.flash('error', 'Сталася помилка при
пошуку, спробуйте ще раз');
                }
            }).lean();

            res.render('shop/SearchPage', { title: ' Store |
Search', filteredProducts, filteredUserProducts, errMsg: errMsg,
```

```
noError: !errMsg, successMsg: successMsg, noMessages: !successMsg
})
        }
    } catch (e) {
        console.log({ message: e });
    }
});

router.get('/products-category', async (req, res) => {
    try {
        const query = req.query.category_select;
        const productCategory = await Product.find({ category:
query }).lean();
        res.render('shop/CategoryPage', { title: 'Store |
Results', productCategory });
    } catch (e) {
        console.log({ message: e });
    }
```

БІБЛІОГРАФІЧНА ДОВІДКА

Індивідуальна тема дипломної роботи бакалавра: «Створення інтернет-платформи для комерційного майданчика»

Обсяг пояснювальної записки: 80 аркушів

Кількість таблиць 3

Кількість рисунків 37

Кількість додатків 1

Дата закінчення дипломної 17 червня 2024 р.

Підпис студента _____