

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСОБІВ
РОЗРОБКИ ВЕБЗАСТОСУНКІВ ЕЛЕКТРОННИХ МАГАЗИНІВ
RESEARCH AND SOFTWARE IMPLEMENTATION OF WEB APPLICATION
DEVELOPMENT TOOLS FOR E-STORES

Виконав: студент(ка) 2 курсу, групи КНТ-110м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Сенічкін А.І.

(прізвище та ініціали)

Керівник Коцур М.І.

(прізвище та ініціали)

Рецензент Гофман Є.О.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
« » 2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Сенічкіна Артура Ігоровича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація засобів розробки вебзастосунків електронних магазинів. Research and Software Implementation of Web Application Development Tools for E-Stores

керівник проєкту (роботи) Коцур Михайло Ігорович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від «05» листопада 2021 року № 435

2. Строк подання студентом проєкту (роботи) 13 грудня 2021 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Коцур М.І., доцент		
Нормоконтроль	Липовець М.В., асистент		

7. Дата видачі завдання “4” вересня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	4-5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	
7	Оформлення пояснювальної записки та документів до неї.	10-13 тиждень	Додатки
8	Нормоконтроль та рецензування.	14-15 тижні	
9	Захист роботи.	16 тиждень	

Студент(ка)

(підпис) Сенічкін А.І.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Коцур М.І.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
112 с., 26 рис., 5 табл., 4 дод., 21 джерело.

ВЕБЗАСТОСУНОК, ЕЛЕКТРОННИЙ МАГАЗИН, ПРОГРАМНЕ
ЗАБЕЗПЕЧЕННЯ, JAVASCRIPT, REACT, REDUX, NODE.JS

Об'єкт дослідження – процес розробки вебзастосунків.

Предмет дослідження – методи та програмні засоби розробки вебзастосунків.

Метою роботи є дослідження та програмна реалізація методів розробки електронних магазинів.

Матеріали, методи та технічні засоби: мова розмітки HTML, CSS для візуального оформлення сторінки, JavaScript для роботи з DOM елементами, React, Redux, Node.js, база даних MongoDB, редактор коду Visual Studio Code, що розповсюджується безкоштовно.

Результати. Розроблено моделі та засоби створення вебзастосунків електронних магазинів.

Практична цінність роботи полягає у розробці вебзастосунку електронного магазину з базовим функціоналом.

Висновки. Розроблено моделі та програмне забезпечення вебзастосунку електронного магазину.

Галузь використання – електронна комерція, інженерія програмного забезпечення вебзастосунків.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 112 pages, 26 figures, 5 tables, 4 appendixes, 21 sources.

WEB APPLICATION, E-SHOP, SOFTWARE, JAVASCRIPT, REACT, REDUX, NODE.JS.

The object of research is the process of web applications developing.

The subject of the research is methods and software tools for web application development.

The purpose of this work is research and software implementation of methods for developing e-shops

Materials, methods and technical tools: HTML markup language, CSS for visual design of the page, JavaScript for work with DOM elements, React, Redux, Node.js, MongoDB database, Visual Studio Code editor, which is distributed free of charge.

Results. The models and tools for creating e-commerce web applications have been developed.

The practical value of the work lies in the development web application e-shop with basic functionality.

Conclusions. Models and software for e-shop web application have been developed.

Scope of use – e-commerce, web application software engineering.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ	9
1 Аналіз предметної області	11
1.1 Дослідження засобів електронної комерції	11
1.2 Класифікація вебсайтів	12
1.3 Дослідження вебзастосунків електронних магазинів	17
1.4 Дослідження функціональних частин вебзастосунків електронних магазинів	20
1.5 Дослідження засобів розробки вебзастосунків електронних магазинів ..	22
1.5.1 Методологія W3DT	22
1.5.2 Метод WSDM	24
1.5.3 Методологія RMM	26
1.5.4 Методологія OOHDM	28
1.5.5 Методологія WISDM	29
1.6 Порівняння методів та засобів розробки вебзастосунків електронних магазинів	31
1.7 Висновки за розділом 1	33
2 Матеріали і методи	35
2.1 Вибір мови програмування та інших технологій розробки	35
2.2 Вибір фреймворку для розробки вебзастосунку електронних магазинів	40
2.3 Вибір серверу для функціонування вебзастосунку електронних магазинів	45
2.4 Вибір та проектування бази даних для реалізації вебзастосунку електронних магазинів	47
2.5 Розробка моделі створення вебзастосунків електронних магазинів	48
2.6 Висновки за розділом 2	51
3 Опис програми	52

3.1	Опис структури вебзастосунку електронного магазину.....	52
3.2	Особливості реалізації архітектури клієнтської частини вебзастосунку електронного магазину.....	53
3.3	База даних вебзастосунку електронного магазину	56
3.4	Опис інтерфейсу користувача та його компонентів	57
3.5	Висновки за розділом 3	60
4	Експлуатація, тестування та експериментальне дослідження програми.....	61
4.1	Призначення й умови застосування програми	61
4.2	Характеристики програми	62
4.3	Інструкція по експлуатації програми.....	62
4.3.1	Звернення до програми.....	62
4.3.2	Вхідні й вихідні дані	62
4.3.3	Повідомлення.....	63
4.4	Тестування та результати роботи вебзастосунку електронного магазину	63
4.5	Висновки за розділом 4	65
	Висновки	66
	Перелік джерел посилання	68
	Додаток А Технічне завдання	70
	Додаток Б Опис програми	74
	Додаток В Текст програми	77
	Додаток Д Слайди презентації	104

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CSS – Cascading Style Sheets (каскадні таблиці стилей);

JS – JavaScript;

HTML – HyperText Markup Language;

HTTP – Hypertext Transfer Protocol;

OOHDM – Object-Oriented Hypermedia Design Methodology;

RMM – Relationship Management Methodology;

W3DT – World Wide Web Design Technique;

WISDM – Web Information System Development Methodology;

ПЗ – програмне забезпечення;

ПК – персональний комп'ютер;

ТЗ – технічне завдання.

ВСТУП

Останнім часом, особливо в умовах карантинних обмежень, велика кількість людей щодня, не виходячи з дому, купують багато товарів за допомогою віртуальних магазинів [1]-[5]. У світі величезними темпами зростає кількість користувачів Internet і, як наслідок, зростає кількість «електронних» покупців, що обумовлює зростання попиту на розробку електронних магазинів. Необхідність зменшення часу створення вебзастосунків електронних магазинів обумовлює доцільність дослідження наявних, а також розробки нових методів та моделей інженерії вебзастосунків електронних магазинів [1]-[5].

Віртуальні магазини сильно зменшують витрати виробника на утримання звичайного магазину, що дає їм можливість розширювати ринки збуту. Також це надає можливість покупцю купувати будь-який товар в будь-який час в будь-якій країні, в будь-якому місті, в будь-який час доби, в будь-який час року. Враховуючи це електронні магазини мають значну перевагу перед звичайними магазинами. Цей момент є істотним при переході виробників з «звичайної» торгівлі на «електронну» [1]-[5].

Тому актуальним є дослідження та програмна реалізація засобів розробки вебзастосунків електронних магазинів

Метою роботи є дослідження та програмна реалізація методів розробки електронних магазинів.

Для досягнення поставленої мети в роботі необхідно розв'язати такі задачі:

- виконати дослідження методів та засобів розробки вебсайтів;
- дослідити існуючі технології, що дозволяють створювати вебсайти;
- розробити модель створення вебзастосунків електронних магазинів;
- створити програмне забезпечення вебсайту для електронного магазину;

– виконати тестування розробленого програмного забезпечення вебсайту електронного магазину.

Об'єкт дослідження – процес розробки вебзастосунків.

Предмет дослідження – методи та програмні засоби розробки вебзастосунків.

Наукова новизна одержаних результатів. Запропоновано інформаційну модель розробки вебзастосунків електронних магазинів у вигляді діаграм в нотації UML, що дозволяють створювати програмне забезпечення у вигляді вебсайтів в залежності від вимог користувачів.

Практичне значення одержаних результатів полягає у тому, що розроблено вебзастосунок електронного магазину з базовим функціоналом.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження засобів електронної комерції

У наш час віртуальні магазини суттєво зменшують витрати виробника на утримання звичайного магазину, що дає їм можливість розширювати ринки збуту. Також це надає можливість покупцю купувати будь-який товар в будь-який час в будь-якій країні, в будь-якому місті, в будь-який час доби, в будь-який час року. Враховуючи це електронні магазини мають значну перевагу перед звичайними магазинами. Цей момент є істотним при переході виробників з «звичайної» торгівлі на «електронну» [1]-[5].

Висока якість продукції, вміння донести інформацію про продукт до споживача і ефективна система збуту, робить підприємство успішним на ринку. У багатьох компаніях зустрічаються проблеми збуту, які заважають ефективно працювати відділу продажів, і не зникають навіть з підбором хороших продавців. Вирішити їх можна лише шляхом автоматизації процесу продажів. У вузькому і технологічному сенсі, під електронним бізнесом раніше розумілося використання інформаційних технологій (в першу чергу пов'язаних з Інтернетом) для організації взаємодії підприємства із зовнішнім середовищем, включаючи постачальників, споживачів, партнерів і т.д. При такому підході електронний бізнес виступає, перш за все, як досить складна прикладна інформаційна система. Більш широкий, або концептуальний, підхід розглядає електронний бізнес як спосіб підприємництва, що сприяє досягненню стратегічного успіху в нову інформаційну епоху. При такому розумінні електронний бізнес аж ніяк не зводиться до інформаційних технологій або активності в Інтернеті. Електронна комерція зачіпає всі аспекти бізнесу, включаючи стратегію, процеси, організацію та технологію, і виводить його далеко за сформовані кордону [1].

Електронна комерція – така форма постачання продукції, при якій вибір і замовлення товарів здійснюється через комп'ютерні мережі, а розрахунки між покупцем і постачальником здійснюються з використанням

електронних платежів. При цьому в якості покупців товарів (або послуг) можуть виступати як приватні особи, так і організації [1]-[5].

1.2 Класифікація вебсайтів

Вебсайт є ефективним засобом торгівлі, який, як і будь-який інший маркетинговий інструмент, заснований на принципі безпосередньої відмови, насамперед, він повинен заінтригувати відвідувача, і має натовкнути його на конкретні дії. Тим не менш, багато ігнорують цю особливість головної сторінки, що часто приводить к тому, що відвідувачі не затримуються на сайті. Такі вебсайти, які містять величезну кількість корисних порад і статей, практично нікогда не досягають очікуваного рівня відвідуваності, не говорячи вже про продажі [1], [2].

Зробивши всього кілька вимірів, просто вебсайт може перетворитись у більш надійний та ефективний інструмент. Важно пам'ятати, що виділяється в день на потенційних клієнтів, обривуючи потік інформації та різних рекламних повідомлень, і те, що заважає їхній увазі, є справжньою найсильнішою конкуренцією. Вебсайт, створив клієнтів не лише для перегляду опублікованих сторінок та досконалих покупок, але й знову відвідав його через невдалий час, а також рекомендує використовувати свої друзям та знайомим [1], [2].

Що бачить користувач при візиті на вебсайт. По-перше, список товарів, що знаходяться на складах. Як правило, привілейований системою автоматизації того або іншого підприємства, цей список міститься у цих виробках, які є у продажу та у звичайних (не віртуальних) журналах. Зберігаючий склад представлений зазвичай у вигляді ієрархічної дерев'яної структури, базових елементів, які використовуються групи товарів [1], [2].

Накопивши корзину, клієнт виконує команду «Виконуйте замовлення» і вибирає зручну для нього форму оплати. Корпоративний покупець повідомляє назву підприємства, номер рахунку, ім'я та телефон контактного

особи. На цьому етапі покупці присвоюється визначений ідентифікаційний код. Це робиться для того, щоб він зайшов у магазин наступного разу, по всій вказаній інформації можна буде не вводити - досить точно вказати свій код. Далі реалізується швидкість і безпосередня передача товару клієнтів [1].

Існують різноманітні форми оплати: оплата кур'єру при доставці, за безналічним розсилом (для організацій), банківським переказом, електронними платежами, оплата розміщеним платежем або поштовим, і телеграфним переводом [1], [2].

Наступні способи доставки замовлення: кур'єрські служби, звичайні поштові, самовивозні та інші способи доставки [1], [2].

Розглянемо типи сайтів у таблиці 1.1.

Таблиця 1.1 – Типи сайтів

Тип сайту	Опис
Сайт візитка	надає інформацію про фірму / послуги / продукти. Як правило, має кінцеве кількість сторінок або одну. Головне завдання – за допомогою візуального оформлення максимально швидко і ефективно донести до відвідувача суть представляється послуги та ступінь її корисності і грандіозності. Такі проекти мають, як правило, яскравий, унікальний дизайн, графічне оформлення переважає перед текстовою інформацією.
Корпоративний сайт	інтернет представництво організації (фірми, корпорації) в мережі, і, отже, в таких проектах найвищі вимоги до оформлення інформації, до балансу графіка / текст. Відповідність фірмовому стилю виражено як у структурі інформації, так і в кольоровій гамі.

Продовження таблиці 1.1

Тип сайту	Опис
Інформаційні ресурси	Це електронні бібліотеки і газети. Інформація, як правило, оновлюється кілька разів на добу. Обсяги інформації дуже великі, і при розробці дизайну таких сайтів слід враховувати той факт, що відвідувачі приходять не подивитися на сайт - вони приходять за змістом.
Віртуальний магазин	При розробці інтерфейсу електронного магазину ставиться мета не тільки швидко і коректно надати інформацію відвідувачеві, але і домогтися того, щоб він став покупцем. Тут, перш за все, слід приділити увагу сервісному обслуговуванню - зручний каталог продукції, пошук по товарам і послугам, ясні і не складні форми, які необхідно заповнити користувачеві для реєстрації, формуванні замовлення і покупки, оперативний зв'язок зі службою тех. підтримки значать більше, ніж яскравий графічний дизайн, однак графічні елементи так само використовуються - це не тільки ілюстрації, фотографії продукції, але і - в першу чергу - піктограми, іконки.

Перші системи і методи електронної комерції зобов'язані появи технологій автоматизації продажів і впровадження автоматизованих систем управління корпоративними ресурсами. У 1960 американські компанії American Airlines і IBM приступають до створення системи автоматизації процедури резервування місць на авіарейси. Це являє собою найперший досвід створення системи електронної комерції [2].

1968 році в Великобританії для обслуговування комерційного безготівкового обороту створена електронна система переказів і клірингу BACS (від англ. Bankers Automated Clearing Services). У 1970 році подібна система під назвою CHIPS (від англ. Clearing House Interbank Payment System) була заснована в США [1], [2].

Стрімке зростання кількості інтернет-користувачів, соціальні мережі, розвиток систем електронних платежів за останні 20 років дозволили Інтернет-комерції стати дуже популярним способом покупок [1], [2].

Віртуальний магазин – це реалізоване в мережі Інтернет представництво шляхом створення Web-сервера для продажу товарів і послуг іншим користувачам мережі Інтернет. Віртуальний магазин називають також Інтернет - магазином. До нього повністю підходить визначення віртуального підприємства. Інакше кажучи, віртуальний магазин - це співтовариство територіально роз'єднаних співробітників магазину (продавців, касирів) і покупців, які можуть спілкуватися і обмінюватися інформацією через електронні засоби зв'язку при повному (або мінімальному) відсутності особистого прямого контакту [1], [2].

Електронна торгівля у віртуальному магазині ґрунтується на тій же структурі, що і традиційна торгівля [1], [2]. Нижче в таблиці 1.2 наведено порівняння традиційного магазину та Інтернет - магазину.

Таблиця 1.2 – Порівняльна характеристика традиційної та електронної торгівлі

Торговельний зал	Віртуальний магазин
Ходьба покупця по торговельному залу і огляд товарів на полицях магазину	Перегляд покупцем веб сторінок
Особистий контакт покупця з продавцем (консультація)	Консультація у продавця (при необхідності) онлайн або по телефону

Продовження таблиці 1.2

Торговельний зал	Віртуальний магазин
Вибір покупцем товару	Вибір покупцем товару
Замовлення товару	Замовлення товару онлайн
Виписка продавцем і вручення покупцеві рахунку на оплату	Пересилання продавцем по комп'ютерних мережах покупцеві рахунку на оплату

Класифікувати електронні магазини можна за різними критеріями. Найцікавішою класифікацією є класифікація за моделлю бізнесу [1], [2]:

- виключно онлайн-магазин;
- суміщення офлайн-бізнесу з онлайн-бізнесом (коли інтернет-магазин був створений на основі вже діючої реальної торгової структури).

По відношенню з постачальниками:

- мають власний склад (наявність реальних товарних запасів);
- працюють за договорами з постачальниками (відсутність значних власних запасів).

Класифікація за товарним асортиментом - книги, аудіо, відеокасети, CD, DVD, комп'ютерна, побутова техніка, мобільні телефони.

Серед методів роздрібного продажу товарів у Мережі можна виділити [1], [2]:

- інтернет - магазини (автоматичні магазини);
- web-вітрини;
- торгові автомати.

Internet-вітрина - швидше це рекламний сервер. На вітрині викладають інформацію про товари, яку постійно оновлюють. Витрати на її створення та адміністрування можуть бути досить низькими, а практична користь такої вітрини очевидна. Але це ще не торгівля. Потенційний покупець, відвідавши вітрину, повинен зателефонувати на фірму, оплатити товар, домовитися про доставку. Тому Internet-вітрина виправдана в тих випадках, коли покупець

треба познайомити зі складною продукцією, на вивчення якої в торговому залі у нього піде дуже багато часу [1], [2].

Internet-вітрина може бути розміщена де завгодно - на власному сервері, на сервері провайдера, на сервері, що надає безкоштовні сторінки. Для роботи з вітриною достатньо мати підключення через телефонну лінію і мінімум навичок роботи з HTML [1], [2], [6], [7].

Торговий автомат може не тільки виконувати функції вітрини, але і приймати замовлення і передавати їх менеджеру, тобто оформляти замовлення і виписувати рахунки на оплату без присутності покупця. Торговий автомат реально торгує і по співвідношенню витрат до результату найбільш кращий для пілотних і тестових проектів з невеликим потоком покупців. Торговий автомат, так само як і Internet-вітрину, можна розмістити і на своєму сервері, і на сервері провайдера. Однак його створення і адміністрування вимагає навичок і певної кваліфікації [1], [2].

Автоматичний магазин - ефективне і комплексне рішення в торговому бізнесі. Він не тільки виписує рахунки, а й відстежує замовлення, приймає електронні платежі і формує заявки на доставку товарів покупцям. Тут завдання менеджера - контролювати роботу системи, складну в обслуговуванні [1], [2].

Оскільки автоматичний магазин повинен мати постійний зв'язок з інформаційною системою компанії, то розміщувати його краще або на корпоративному сервері в локальній мережі, або на віддаленому сервері з постійно діючим каналом зв'язку. Інтерактивні магазини можуть торгувати чим завгодно.

1.3 Дослідження вебзастосунків електронних магазинів

Використання вебзастосунків електронних магазинів дозволяє зменшити чисельність персоналу за рахунок скорочення обсягу взаємодії з клієнтами, оренда дискового простору і розміщення "електронної вітрини"

дешевше і простіше оренди торговельних приміщень і розміщення товарів на полицях, немає потреби в касовому обслуговуванні і т.д. Так само віртуальний магазин можна використовувати як ефективний спосіб маркетингового дослідження, тим більше, що сьогодні ця послуга досить дорога в маркетингових агентствах. Будь-який користувач мережі Інтернет може швидко заповнити анкету, запропоновану йому магазином через комп'ютер. Це дозволяє без особливих витрат вивчити потреби і смаки потенційних покупців і врахувати результати маркетингового обстеження в своїй роботі [1], [2].

Вебзастосунки електронних магазинів дозволяють зробити покупку, не виходячи з офісу в будь-який час, а вибір і замовлення товару займе у нього кілька хвилин, якщо він точно знає, що хоче купити. Служба доставки інтернет-магазину доставить обраний товар у зручний час і місце. Крім цього вибір і оцінка властивостей товару відбувається в інтернет-магазині набагато швидше і зручніше ніж в звичайному магазині. Досить уявити, що покупець прийшов в звичайний магазин побутової техніки і хоче порівняти параметри 20-30 радіотелефонів. Для цього необхідно вивчити всі цінники: запам'ятати характеристики, ціни і назви моделей. Можна звернутися до продавця-консультанта, якщо він не зайнятий, з проханням розповісти про всі ці товари. Але зазвичай жоден продавець не має стільки вільного часу для роботи з одним клієнтом, і немає гарантії, що він володіє всією інформацією [1], [2].

Асортимент інтернет-магазину нічим не обмежений (як, наприклад, асортимент звичайного магазину обмежений площею торгового павільйону). А якщо передбачений пошук по параметрам, то можна вказати характеристики, яким повинен відповідати товар, і потім вже вибирати зі списку моделей, що задовольняють запиту. І ще одне дуже важливе зауваження - ні один менеджер з продажу не в змозі пам'ятати стільки інформації про товар, скільки може надати інтернет-магазин. Інтернет-магазин в змозі видати таку інформацію, як купівельний рейтинг, поради та відгуки про товар, статті про товар, які можуть визначити вибір [1], [2].

Витрати на роботу інтернет-магазину, включаючи доставку, істотно нижче ніж у звичайного. На відміну від звичайного магазину інтернет-магазин здатний обслужити кілька сотень клієнтів одночасно. Крім того, якщо покупець проживає в іншому місті, він отримує можливість не витратитися на міжміські телефонні консультації. Всю інформацію можна знайти на сторінках інтернет-магазину [1], [2].

Ціни, як правило, нижче, ніж в звичайних магазинах. Це легко пояснюється тим, що підприємцю не потрібно орендувати торгову площу для магазину і не потрібно наймати на роботу продавців і мерчендайзерів, заробітна плата яких включається до вартості товарів. Для організації продажів в інтернет-магазині потрібен сайт, менеджер і кур'єри, їх утримання обходиться набагато дешевше, ніж утримання персоналу звичайного магазину, що в свою чергу знижує собівартість товару [1], [2].

Віртуальний магазин дає можливість ознайомитися з описом товарів в будь-який час доби без перерв на обід, закриття на ніч або вихідних. Це є черговим перевагою інтернет-магазинів перед звичайними магазинами [3].

Незручна система оплати - іноді покупці відмовляються від здійснення покупки тільки тому, що Інтернет-магазин не запропонував зручну для них форми оплати [1], [2].

Найпоширенішою формою оплати є оплата кур'єру готівкою. Він же зазначається як найбезпечніший спосіб [1], [2].

Найбільш економним способом оплати покупці відзначають передоплату банківським переказом.

Іноді проблемою являється складна система замовлення. Вона полягає в довгому і заплутаному процесі оформлення покупки. Покупцеві необхідно заповнити довгу і часто не зовсім зрозумілу форму, в результаті чого витрачається багато часу [1], [2].

Деякі покупці зазначають повільну швидкість завантаження деяких магазинів, що може послужити причиною відходу з магазину. У деяких магазинах потрібно проходити дуже довгий покроковий процес реєстрації,

щоб зробити замовлення. Це відлякує потенційних покупців здійснювати покупки в даному магазині, особливо коли покупець цінує витрачений їм час.

Існують категорії товарів, які не будуть настільки успішно продаватися в інтернет-магазинах, як в звичайних магазинах, де покупець може спробувати, приміряти, помацати товар, переконатися в його якості. Так, наприклад, взуття, одяг, деякі види продуктів краще продавати в звичайних магазинах. Адже інтернет-магазин не може дати покупцеві можливість доторкнутися, приміряти, спробувати товар, коли йому це просто необхідно для здійснення покупки [1], [2].

Отже, використання вебзастосунків електронних магазинів у наш час є досить ефективним та доцільним, особливо в умовах карантинних обмежень.

1.4 Дослідження функціональних частин вебзастосунків електронних магазинів

Типовий варіант вебзастосунку електронного магазину може бути з наступних функціональних частин [1]-[6]:

- каталог товарів;
- пошукова система;
- користувальницька корзина;
- реєстраційна форма;
- форма відправки замовлення.

Каталог представляє єдину складну і багаторівневу структуру даних, яка є простою і зрозумілою, що виготовляє порівняно виробничі товари. Пропонуємо лише такий каталог представити у вікні дерев'яні предмети, верхній рівень якого є з списки розділених. Ви можете розмістити підрозділи або посилання на конкретний товар і т.д. Таке порівняння просто необхідне для зручного та швидкого пошуку та замовлення товару [1]-[6].

Пошукова система є обов'язковим елементом динамічного електронного каталогу і реалізується на стороні сервера. Незрозумілий на то,

що каталог забезпечує порівняння і групування даних, пошукова система надає користувачеві можливість швидко знайти пошук, що особливо важливо в цій справі, коли каталог представляє достатньо розроблену структуру даних з великим числом розділів, підрозділів і товарів, користувач плоскі представляє в каку Роздільник може знайти цікавий свій товар і є він у каталозі своїх публікацій. Пошукова система в деяких випадках дозволяє значно скоротити кількість переходів між сторінками каталогу для доступу до цікавої інформації [1]-[6].

Індивідуальна реалізація пошуку в Інтернеті передбачає, що тут відбувається вибірка всіх записів, які задовольняють умову запиту (даний механізм шукає пошук із ізбитком). У випадку більших результатів пошуку результатів пошуку, які виконуються по-справжньому для того, щоб відвідувати не надходило довге, щоб встигнути згорнути всі вибори, які можуть включати в себе сотні, тисячі і більші записи. Як правило, відвідувачі не дивляться на всі сторінки вибору, обмежуючи двома або трьома. Таким чином, цей механізм пошуку в багатьох випадках працює краєм медленно і неефективно. Однак він зможе реалізувати виборку однакових товарів від різних поставок, порівняти їх параметри між собою і вибрати найкращий варіант [1]-[6].

Користувальницька корзина представляє собою окремі масивні дані, які обслуговують для зберігання замовленого користувачем товару.

Реєстраційна форма служить для введення персональних даних користувачів. У даній інформації, яка використовується для їх ідентифікації між сеансами роботи з інтернет-магазином. Дана інформація може зберігатись як на сторонній сервері, так і на стороні клієнтів [1]-[6].

Форма відправки заказа відправляє службу для введення контактної інформації контактної інформації та відправляє її та заказує на електронний ящик організацій [1]-[6].

Інтернет-магазин можна реалізувати як на стороні сервера, так і на стороні клієнтів. У першому випадку використовуються серверні сценарії,

побудовані на таких технологіях, як PHP, Perl, ASP, JSP, ColdFusion, JavaScript і т.ін.[1-3].

1.5 Дослідження засобів розробки вебзастосунків електронних магазинів

У наш час існують різні засоби та технології розробки вебзастосунків електронних магазинів, при досить часто з'являються нові технології, що зменшують витрати на розробку та вдосконалення інформаційних систем вебзастосунків. Такі технології дозволяють подавати вебсайт не як засіб подання інформаційного змісту, а як повноцінну платформу для бізнесу, спілкування та соціальної взаємодії. Тому сучасні засоби розробки та підтримки вебзастосунків повинні дозволяти підприємствам безперервно впроваджувати інновації на своїх вебсторінках, щоб вони не застаріли та не втратили зв'язок зі своєю базою користувачів, не відповідаючи їй зростаючим і вимогливим потребам. З еволюцією всесвітньої павутини та зростаючою потребою в інноваціях вебзастосунків електронних магазинів дослідники намагалися запропонувати різні методи та технології, що дозволяють покращити розробку вебсторінок. Проаналізуємо основні методи та засоби розробки вебзастосунків електронних магазинів [1]–[6].

1.5.1 Методологія W3DT

Техніка W3DT (World Wide Web Design Technique) представляє собою підхід, що використовується для проектування вебгіпермедійних застосунків, та був розроблений спеціально для розробки великих вебсайтів. Це техніка, зосереджена на спільній розробці розподілених вебсторінок, що дозволяє моделювати високоструктуровані інформаційні домени, схожі на базу даних, і звичайний гіпертекст. Таким чином, він полегшує процес розробки як структурованих, так і неструктурованих вебсторінок, підтримуючи як

статичний, так і динамічний вміст. Це також полегшує створення уніфікованих вебсайтів, використовуючи підмоделі для кожного місця, що робить його ідеальною методологією для великомасштабних вебсайтів [2], [6].

Методологія W3DT містить інтуїтивно зрозумілу графічну модель, придатну для розробки вебсторінок, яка описує компоненти дизайну, такі як сайти, сторінки, індекс, форми, меню, посилання, динамічні посилання тощо.

Процес розробки вебзастосунків в рамках цієї структури поділяється на такі етапи [2], [6]:

- розробники окреслюють графічне подання вебсайту та відповідних сторінок, яке буде враховувати не тільки естетичне представлення вебсайту, але й його навігаційну структуру. Це досягається шляхом побудови однієї або кількох діаграм. Ці діаграми складаються щонайменше з однієї сторінки з додатковими посиланнями та макетом, який визначатиме специфікації форматування для кожної сторінки. Основна структура кожної сторінки складається з трьох елементів: форми, покажчика та меню;

- розробники створюють і запускають прототипи, використовуючи комп'ютерне середовище. На цьому етапі використовується комп'ютерне середовище проектування, що дає розробнику можливість створити запущений прототип вебсайту. Середовище проектування є CASE-інструментом для методології W3DT, який надає цій методології інтуїтивно зрозумілий браузер, який дозволяє розробникам графічно представляти вебсайт під час його розробки, а також малювати та редагувати всі функції його конструкцій.

W3DT поєднує техніку моделювання з комп'ютерним середовищем проектування. В першу чергу ця методологія була створена для підтримки вимог неструктурованих ієрархічних доменів. Це візуальна методологія високого рівня, яка використовує функціональність HTML та його фундаментальну мову [2], [6].

1.5.2 Метод WSDM

Метод WSDM (Web Site Design Method, метод проєктування вебсайтів) дозволяє проєктувати не тільки традиційні вебзастосунки, але й семантичні вебдодатки, що призвело до того, що його перейменували на метод вебсемантичного дизайну [2], [6].

Метод WSDM є методологією, орієнтованою на користувача, яка ставить користувача в центр розробки вебсайту. Він зосереджується на філософії дизайну, орієнтованій на аудиторію, коли продукт створюється з урахуванням аудиторії. Таким чином, розробник повинен враховувати різні потенційні цільові аудиторії (відвідувачі та користувачі). Їхні потреби та переваги мають бути відправною точкою для процесу проєктування. Це означає, що ключова структура вебсайту впливає з уподобань цільової аудиторії, в результаті чого на домашній сторінці пропонуються різні шляхи навігації (іменовані аудиторії), по одному для кожного різного типу користувачів/відвідувачів. При використанні цього підходу кожен етап проєктування зосереджується на одній конкретній особливості циклу вебдизайну, наприклад, аналіз вимог і завдань, моделювання даних і функціональних можливостей, моделювання навігації, моделювання та впровадження презентації [2], [6].

WSDM – це методологія, яка не тільки пропонує примітиви моделювання, що дозволяє веброзробникам проєктувати та розробляти моделі, які зображують вебсайт/додаток з різних точок зору та на різних рівнях абстракції, але, крім того, вона також пропонує систематичний спосіб розробки Вебдодаток. Оскільки ця методологія не просто прив'язана до технології, вона не тягне за собою дизайн і структуру даних. Головним завданням цієї методології є визначення потенційних користувачів та їхніх інформаційних потреби. Таким чином, можна стверджувати, що управління інформацією знаходиться в центрі цієї методології [2], [6].

WSDM містить п'ять основних етапів. Існує попередній етап, специфікація місії, під час якого окреслюються мета і задачі розробки вебсайту, а також його тематика та цільові користувачі. Другий етап — моделювання користувача. Тут дуже важливо зосередитися на потенційних користувачах вебсайту. Цей етап розділений на два етапи: класифікація користувачів та опис класу користувачів, де користувачі ідентифікуються та класифікуються. Клас користувача — це поділ потенційних користувачів, які є рівними з точки зору їхніх інформаційних вимог. Іншим етапом є концептуальне проектування, яке також поділяється на два етапи: моделювання об'єктів, де інформаційні вимоги кожного окремого класу користувачів та їх перспективи правильно описуються шляхом розробки концептуальної об'єктної моделі для кожного з різних класів користувачів, і розробка дизайну, де розробляється навігаційна модель. Навігаційна модель представляє навігаційну доріжку, яка повідомляє, як користувачі можуть переміщатися по наявній інформації [2], [6].

Третій етап — проектування реалізації. На цьому етапі розробляється «вигляд» вебсайту. Мета — створити надійний, приємний та ефективний «вигляд» концептуального проекту, зробленого на попередніх етапах. Четвертий етап — впровадження, фактична добудова вебсайту та його розміщення в мережі. Розробляючи програму за цією методологією, розробник повинен дотримуватися чітко визначеної філософії дизайну, яка допоможе йому/їй отримати необхідну підтримку для організації вебсайту. З WSDM розвиток включає ланцюг послідовних етапів. Кожен етап має чітко визначений вихід. Таким чином, для кожного етапу пропонується (під) метод, який ілюструє, як отримати вихід з його входу. Вихід однієї фази є входом наступної фази. Реалізація WSDM гарантує регулярний доступ до автономних обчислювальних елементів і чітко визначений життєвий цикл. Крім того, кожен з елементів розглядається як ресурс, який можна адресувати за допомогою стандарту адресації вебсервісів [2], [6]. Таким чином, метод WSDM дозволяє досить на високому рівні розробити інтерфейс сайту. Проте

він не дозволяє виконувати проектування та керування динамічними даними. Тому при розробки методів та засобів створення вебсайтів інтернет-магазинів доцільно поєднати його з іншими методологіями для розробки структури даних та динаміки сайту.

1.5.3 Методологія RMM

Методологія управління відносинами (Relationship Management Methodology, RMM) спочатку була розроблена як основа для проектування та конструювання гіпермедійних додатків. Ця методологія носить таку назву, оскільки вона зосереджена на гіпермедійних додатках як засобах для зв'язків між інформаційними об'єктами [2], [6].

RMM – це структурована методологія, у якій етапу процесу розробки передують ряд досліджень, зосереджених на таких речах, як цілі вебсайту, аналіз ринку та користувачів, джерела інформації та дозволи, канали розподілу та аналіз витрат і вигод. Це призводить до техніко-економічних обґрунтувань, а також до глибокого знання як інформації, так і навігаційних вимог.

Після того, як потреби, цілі та вимоги проекту чітко визначені, виконується процес із семи етапів. Перший етап – E-R дизайн, на якому інформаційна область програми представляється E-R (Entity-Relationship) діаграмою. На цьому етапі виконується дослідження відповідних сутностей і взаємозв'язків області додатка [2], [6].

На наступному етапі виконується проектування об'єктів (сутностей). Цей крок визначає, як інформація у вибраних об'єктах буде представлена користувачам, і як вони можуть отримати до неї доступ. Отримана концепція описується як діаграма E-R+. Це передбачає поділ сутності на значущі частини та організацію їх у гіпертекстову мережу [2], [6].

На етапі навігаційного проектування окреслюється та розробляється модуль, який дозволить здійснювати гіпертекстову навігацію. Кожне

асоціативне відношення, що з'являється на діаграмі E-R+, досліджується та відображається на діаграмі керування відносинами (RM, Relationship Management) [2], [6].

Етап розробки конверсійного протоколу використовує групу правил перетворення для перетворення кожного елемента діаграми моделі даних керування відносинами (RMDM) в об'єкт цільової платформи [2], [6].

На етапі проєктування користувальницького інтерфейсу виконується дизайн макетів екрана для кожного об'єкта, що відображається на діаграмі, отриманій раніше.

Проєктування поведінки під час виконання програми – на цьому етапі розробники приймають рішення про те, які механізми обходу посилання, історії, зворотного відстеження та навігації мають застосовуватися. Важливо враховувати нестабільність і розмірність домену, щоб вибрати, чи потрібно розробляти вміст вузла та кінцеві точки посилань під час програми чи динамічно обчислюватися на вимогу під час виконання [2], [6].

Останній етап пов'язаний безпосередньо з побудовою вебзастосунку та його тестуванням, і полягає в реалізації програми та тестуванні.

Вебзастосунки, розроблені за цією методологією, представлені у вигляді моделей RMDM, заснованих на моделі E-R і методі (HDM, Hypermedia Design Method), який був одним із перших методів, створених для визначення структури та взаємодії гіпермедійних додатків. Методологія RMM поєднує підхід зверху вниз і знизу вгору. Створені таким чином моделі даних дозволяють розбивати атрибути даного об'єкта або сутності на фрагменти та групувати об'єкти. Концепція фрагмента тут призначена для посередництва між логічною архітектурою вебсайту та його презентацією, оскільки кожен фрагмент містить інформацію, яка буде відображатися. RMM особливо підходить для розробки та впровадження вебсайтів, зосереджених на реляційних базах даних. Він дозволяє зображати та розвивати домен програми абстрактним способом, а також описує всі елементи домену програми щодо типів сутностей, атрибутів і зв'язків, за допомогою включення моделей даних

відношень [2], [6]. Таким чином, методологія RMM входить до категорії методологій розробки вебсайтів, які використовують модифіковані моделі, засновані на традиційних методологіях розробки програмного забезпечення. Проте методологія RMM є надзвичайно складною, що робить її менш адаптованою, оскільки веброзробники, як правило, мають мало досвіду в предметній області.

1.5.4 Методологія OOHDM

Методологія об'єктно-орієнтованого гіпермедійного проєктування (Object-Oriented Hypermedia Design Methodology, OOHDM) орієнтована на використання інформації та принципу, згідно з яким інформацію можна розділити на менші частини незалежно від того, як вона представлена або збережена. Цей підхід використовує механізми абстракції та композиції в об'єктно-орієнтованій моделі, щоб дозволити виконати стислий опис складних інформаційних елементів і дозволити специфікувати складні шаблони навігації та перетворення інтерфейсу [2], [6].

Методологія OOHDM по суті базується на моделі Hypermedia Design Model (HDM) та методі, який зосереджується на уявленні про гіпертекст як поєднання сутностей та їхніх відносин, таких як навігаційні шляхи. Однак OOHDM розширює HDM до об'єктно-орієнтованої парадигми, де шаблони проєктування дозволяють механізми композиції (наприклад, класифікацію, агрегацію та ієрархії успадкування), що веде до абстракції та повторного використання шляхом введення кількох візуальних схем для покращення виразності моделі. На відміну від HDM, OOHDM представляє чітко визначений метод розробки гіпермедійних вебзастосунків [2], [6].

Згідно з методологією OOHDM, вебзастосунок гіпермедіа розробляється в чотиристадійному процесі, який підтримує інкрементну або прототипну модель процесу. Кожен етап підкреслює конкретну проблему проєктування, і будується об'єктно-орієнтована модель. Класифікація,

агрегація та узагальнення/спеціалізація використовуються протягом усього процесу для покращення можливостей абстракції та можливостей повторного використання. Основними стадіями розробки вебзастосунків при використанні цієї моделі є концептуальне проєктування, проєктування навігації, проєктування інтерфейсу, імплементація розробленого вебзастосунку [2], [6].

Методологія OOHDM враховує саму природу інформаційних систем управління, де різні користувачі повинні отримати доступ до спільних даних у спосіб, який адаптований для них. Гіпермедійний застосунок розроблено та побудовано як набір навігаційних об'єктів, які працюють як логічні вікна на спільній концептуальній моделі. Використовуючи модель OOHDM, можна створити план для області гіпермедійних застосунків, використовуючи визнані об'єктно-орієнтовані концепції, такі як структура і поведінка, і механізми абстракції, такі як агрегація та узагальнення/спеціалізація [2], [6].

Метою застосування моделі OOHDM є допомога дизайнерам вебзастосунків у розробці однокористувацького гіпермедійного середовища. Це також методологія, орієнтована на сприйнятливую модель гіпермедійних додатків, у якій користувачі/відвідувачі переміщаються по гіпермедійному простору, який був розроблений заздалегідь. Проте в результаті такої орієнтації моделі OOHDM не вистачає деяких можливостей для проєктування функціональних можливостей редагування або створення, які дозволяють маніпулювати гіпермедійним простором.

1.5.5 Методологія WISDM

Методологія розробки вебінформаційних системи (Web Information System Development Methodology, WISDM) призначена для створення вебінформаційних систем, та була розроблена з усвідомлення того, що традиційні методології розробки системи, такі як життєвий цикл розробки системи (SDLC), методологія водоспаду або модель швидкої розробки додатків (RAD), недостатні для опису конкретних особливостей та потреб

веброзробки. Існують три виміри, які відокремлюють методології розробки настільного програмного забезпечення від Інтернет-проектів. По-перше, у традиційній розробці є велика абстракція у вимогах та стратегічних принципах, тоді як у веброботці стратегічний вимір набагато більш очевидний і відчутний. По-друге, у традиційній розробці інформаційних систем типовий користувач — це працівник, якого можна навчати та консультувати безпосередньо, тоді як у веброботці типовий користувач — це, по суті, клієнт, якому не потрібно використовувати продукт чи навчати його використання. Нарешті, дизайн традиційного проекту інформаційної системи зосереджений на чистому зручності використання, тоді як при розробці вебзастосунків продукт повинен бути візуально привабливим, а також зручним для використання [2], [6].

Таким чином, методологія WISDM намагається створити основу для поєднання традиційних методів розробки систем з методами на основі Webтехнологій. Використовуючи усталені методи та прийоми, ця методологія спирається на існуючі найкращі підходи, а не додає нову методологію взагалі.

Методологія WISDM використовує п'ять основних аспектів, які формують її суть. Ці п'ять аспектів є відображенням різних рівнів впливу, які будуть формувати кінцевий продукт відповідно до його відповідних зацікавлених сторін [2], [6].

По-перше, виконується аналіз вимог та умов до розробки вебзастосунку. Організаційний аналіз насамперед пов'язаний зі створенням цінності продукту. Вебсайт або вебдодаток повинні відображати бізнес-цілі організації, зосереджуючись на дослідженнях ринку та вимогах до інформації. Аналіз інформації дозволяє веброботнику додатково встановлювати та визначати вимоги користувачів, головним чином шляхом створення документів із графічними позначеннями та/або прототипами програмного забезпечення. WISDM зазвичай використовує модель UML уявлення про функціональність системи [2], [6].

Після етапу аналізу виконується етап проектування. Робочий дизайн (проектування)— це створення та визначення відносин між замовником і співробітником, і це, по суті, тягне за собою принцип проектування сайту або програми з урахуванням замовника. Технічний дизайн – це сама по собі розробка вебсайту. Це стосується програмування та структури даних, необхідних для створення вебсайту [2], [6].

При використанні цього підходу важливу роль відіграє взаємодія людини та комп'ютера, що являє собою взаємодію між технічним і робочим дизайном через дизайн інтерфейсу користувача. Дизайн інтерфейсу користувача є важливою частиною загального процесу розробки вебдизайну. Цей процес повинен поєднувати технічні навички, досвід та очікування замовника/користувача. Веброзробник повинен зосередитися на дизайні інтерфейсу користувача вебсайту, гарантуючи, що він буде легко доступним, зручним, гнучкий, захищений, а також підтримуватиме ергономічні функції [2], [6].

Таким чином, методологія WISDM використовує соціально-технічний погляд на розробку вебзастосунків, проте виконання всіх етапів, запропонованих у цій методології, вимагає багато часу роботи веброботників.

1.6 Порівняння методів та засобів розробки вебзастосунків електронних магазинів

Результати проведеного у попередніх підрозділах аналізу показали, що на відміну від традиційної розробки програмного забезпечення, інженерія вебзастосунків тягне за собою багато аспектів та особливостей, які необхідно враховувати у процесі розробки ПЗ, таких як широке коло можливих користувачів, необхідність представити привабливе зображення та набагато більший акцент на зручності використання. Такі особливі характеристики спонукали дослідників представити методології, розроблені спеціально для Інтернету. Ці методології, як правило, приділяють більше уваги процесам

проектування, а також враховують велику кількість користувачів та більш активний період обслуговування вебзастосунків.

У таблиці 1.3 наведено порівняння відомих методів та засобів розробки вебзастосунків електронних магазинів.

Таблиця 1.3 – Порівняння методів та засобів розробки вебзастосунків електронних магазинів

Критерій порівняння	Методи та засоби розробки вебзастосунків				
	W3DT	WSDM	RMM	OOHDM	WISDM
Зручність використання для розробки вебзастосунків електронних магазинів	+	±	±	±	±
Необхідність спільної розробки вебсайту	+	±	–	–	+
Можливість роботи з великими даними (розробки великих вебсайтів)	+	+	±	±	±
Орієнтованість на кінцевого користувача	±	+	±	±	±
Наявність етапу оцінки економічної доцільності розробки вебсайту	–	–	+	–	–
Робота з базами даних	±	±	+	±	±

Як видно за таблиці 1 та проведеного аналізу, незважаючи на формальні відмінності, всі вебметодології, мають спільну головну мету – покращення та полегшення процесу дизайну та розробки вебсайту. Ці

методології використовуються з метою допомогти веброзробникам у процесі створення та структурування вебсторінки. Однак надзвичайно важливо, щоб методології сприяли оцінці інформації та заохочували ефективні та привабливі форми подання та надання інформації як засобу просування продуктів і послуг. Крім того, розробляючи вебструктуру, розробники та організація повинні гарантувати оптимальну видимість усієї доступної інформації стосовно мети вебсайту, щоб переконатися, що користувач може отримати доступ до всієї інформації, яка йому потрібна на цьому одному вебсайті. Таким чином, наявність недоліків існуючих методологій розробки вебзастосунків обумовлює необхідність розробки нових та модифікації існуючих методів інженерії вебзастосунків електронних магазинів.

1.7 Висновки за розділом 1

Проаналізовано засоби електронної комерції. Визначено, що у наш час віртуальні магазини суттєво зменшують витрати виробника на утримання звичайного магазину, що дає їм можливість розширювати ринки збуту. Також це надає можливість покупцю купувати будь-який товар в будь-який час в у будь-якій точці світу. Виконано класифікацію вебсайтів. Досліджено вебзастосунки електронних магазинів, визначено їх функціональні характеристики. Визначено, що типовий варіант вебзастосунку електронного магазину може бути з наступних функціональних частин: каталог товарів, пошукова система, користувальницька корзина, реєстраційна форма, форма відправки замовлення.

Досліджено засоби розробки вебзастосунків електронних магазинів. Результати проведеного аналізу показали, що на відміну від традиційної розробки програмного забезпечення, інженерія вебзастосунків тягне за собою багато аспектів та особливостей, які необхідно враховувати у процесі розробки ПЗ, таких як широке коло можливих користувачів, необхідність

представити привабливе зображення та набагато більший акцент на зручності використання.

Проаналізовано методології W3DT, WSDM, RMM, OOHDM, WISDM розробки вебзастосунків, які приділяють більше уваги процесам проектування, а також враховують велику кількість користувачів та більш активний період обслуговування вебзастосунків.

Проте при використанні деяких моделей не вистачає певних можливостей для проектування функціональних можливостей редагування або створення, які дозволяють маніпулювати гіпермедійним простором. Інші методології інженерії вебзастосунків є досить складними, внаслідок чого виконання всіх етапів процесу розробки вебсайтів, запропонованих у таких методологіях, вимагає багато часу роботи веброзробників. Таким чином, наявність недоліків існуючих методологій розробки вебзастосунків обумовлює необхідність розробки нових та модифікації існуючих методів інженерії вебзастосунків електронних магазинів.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Вибір мови програмування та інших технологій розробки

Вебсайти подаються за допомогою мови HTML (HyperText Markup Language - «мова гіпертекстової розмітки»), яка є базовим будівельним блоком, з яким можуть співпрацювати усі веббраузери. Він визначає зміст і структуру вебконтенту. Інші технології, крім HTML, зазвичай використовуються для опису зовнішнього вигляду / уявлення (CSS) або функціональності / поведінки (JavaScript) вебсторінки [7], [8].

Існують кілька версій HTML. HTML 2.0 (листопад 1995) був розроблений Internet Engineering Task Force (IETF) для впорядкування загальноприйнятих положень в кінці 1994 року. HTML + (1993) і HTML 3.0 (1995) - це багатші версії мови HTML / 4 /. Більшість людей визнають, що документи HTML повинні працювати в різних браузерах і на різних платформах. Досягнення сумісності знижує витрати авторів, оскільки вони можуть розробляти тільки одну версію документа. В іншому випадку виникає ще більший ризик, що буде являти собою суміш особистих несумісних форматів, що в кінцевому рахунку призведе до зниження комерційного потенціалу для всіх учасників. У кожній версії HTML робилася спроба відобразити все більше число угод між працівниками і користувачами цієї індустрії, щоб зусилля авторів не були витрачені даремно, а їх документи не стали б нечитабельним в короткий термін [7].

Під гіпертекстом ("hypertext") розуміються посилання, які з'єднують вебсторінки один з одним або в межах одного вебсайту, або між вебсайтами. Посилання є фундаментальним аспектом Інтернету. Завантажуючи контент в Інтернет і пов'язуючи його зі сторінками, створеними іншими людьми, ви стаєте активним учасником Всесвітньої павутини.

HTML використовує розмітку ("markup") для відображення тексту, зображень та іншої інформації в веббраузері. HTML-розмітка включає в себе спеціальні "елементи", такі як <head>, <title>, <body>, <header>, <footer>.

<article>, <section>, <p>, <div>, , , <aside>, <audio>, <canvas>, <datalist>, <details>, <embed>, <nav>, <output>, <progress>, <video> і багато інших [7].

HTML-елемент виділяється з іншого тексту в документі за допомогою "тегів", які складаються з імені елемента оточеного "<" і ">". Ім'я елемента всередині тегу не чутливе до регістру. Тобто, воно може бути написано в верхньому або нижньому регістрі, або змішано. Наприклад, тег <title> може бути записаний як <Title>, <TITLE>, або будь-яким іншим способом [7].

Проектування інтерфейсу сайту доцільно виконати за допомогою CSS (Cascading Style Sheets, або каскадні таблиці стилів) - це декларативна мова, яка відповідає за те, як сторінки виглядають в веб браузері. CSS стилі застосовуються до елементів сторінки для їх правильного відображення [8].

CSS одна з ключових Web технологій, поряд з HTML і JavaScript. Як правило CSS використовується для визначення стилів HTML елементів, але також може бути застосований спільно з іншими мовами розмітки, такими як SVG або XML [8].

CSS правило складається з селектора і набору властивостей з їх значеннями [8].

CSS - це фактично мова стилів, який визначає відображення HTML-документів. CSS працює зі шрифтами, з квітами символів і фону, з полями, з рядками, з висотою і з шириною елементів відображення, з фоновими зображеннями, з позиціонуванням елементів і багато з чим іншим [8].

Якщо HTML необхідний для структурування змісту сторінки, то CSS необхідний для того, щоб форматувати це структуроване зміст [8].

Використання CSS полегшує створення якісних сайтів, дозволяючи задати стилі окремих елементів сторінок сайту в особливих css-файлах, щоб в подальшому бути впевненим в тому, що всі сторінки сайту будуть витримані в єдиному стилі [8].

CSS-модулі - це CSS-файли, в яких всі класи і анімації за замовчуванням знаходяться в локальній області видимості. CSS-модулі - це не

офіційна специфікація, вони не імплементовані в браузері, скоріше, це завдання, що запускається на стадії складання проекту (наприклад, за допомогою Webpack або Browserify), в процесі виконання якої імена класів і селектори змінюються так, щоб утворилася свого роду локальна область видимості (щось на зразок простору імен) [8].

CSS-модулі використовують інший підхід ніж стандартний css. Замість того, щоб писати звичайний HTML, нам доведеться написати розмітку в JavaScript-файлі, наприклад, в index.js. У процесі складання компілятор аналізує файл стилів, який ми імпортували, потім аналізує JavaScript і створює клас, який доступний через styles.className. Потім генерує на їх основі нові HTML і CSS-файли, вже з новими класами. CSS-модулі гарантують, що всі стилі одного компонента знаходяться в одному місці і застосовуються тільки до цього компонента і ніякому іншому. Цей підхід був розроблений, щоб вирішити проблему глобальному контексті в CSS [8].

Існує три методи для використання CSS для документа HTML [8]:

- вбудований (Inline). Цей метод дозволяє в будь-який тег HTML додати стиль. Використання вбудованого методу надає максимальний контроль над усіма властивостями документа. Наприклад, якщо необхідно задати зовнішній вигляд окремого абзацу, то можна просто додати атрибут style = x до тегу абзацу, і браузер відобразить цей абзац за допомогою параметрів стилю, доданого в код;

- впроваджений (Embedded). Впровадження дозволяє контролювати всю сторінку HTML. При використанні тега <style>, поміщеного всередині розділу <head> документа, в код вставляються деталізовані атрибути стилю, які будуть застосовуватися до всієї сторінці;

- пов'язаний (Linked або External). Пов'язана таблиця стилів - потужний інструмент, який дозволяє створювати зразки стилів, який потім можна застосовувати до всіх документів HTML одночасно. Основний документ таблиці стилів створюється програмістом. Цей документ містить стилі, які

будуть єдиними для всіх документів. Будь-який документ HTML, пов'язаний з цим файлом, буде використовувати зазначені стилі [7-8].

Синтаксис всіх методів, використовуваних для застосування стилів до документів HTML, практично однаковий. Таблиці стилів, як і пропозиції, складаються з певних частин. Ці частини включають такі елементи [8]:

- показчик (Selector). Показчик є елементом, до якого будуть застосовуватися призначаються атрибути. Це може бути просто тег типу заголовок або абзацу. Таблиці стилів дозволяють використовувати різні об'єкти, включаючи і класи;

- властивість (Property). Властивість визначає показчик. Наприклад, якщо в якості показчика обраний абзац, можна включити властивості, що визначають цей показчик. У властивості входять такі елементи, як поля, шрифти і фонові зображення. У таблицях стилів існує багато властивостей, які можна використовувати для того, щоб визначити показчик;

- значення (Value). Значення визначають властивості. Припустимо є заголовок першого рівня H1 (показчик) і необхідно включити властивість `type-family` (властивість шрифту). Шрифт, який застосовується до зазначеного фрагменту, задається значенням цієї властивості.

JavaScript – мова сценаріїв, або скриптів. Скрипт являє собою програмний код – набір інструкцій, який не вимагає попередньої обробки (наприклад, компіляції) перед запуском. Код JavaScript інтерпретується двигунком браузера під час завантаження вебсторінки. Інтерпретатор браузера виконує порядковий аналіз, обробку і виконання вихідної програми або запиту [9]–[21].

Призначення мови JavaScript – дозволити зробити сторінку активною. Мова JavaScript не має ніякого відношення до мови Java, незважаючи на схожу назву. Java має дуже мало коштів для зв'язку аплетів з браузером, що так необхідно розробникам в їх додатках, в той час як JavaScript розроблявся саме для цього. На відміну від Java, JavaScript є мовою, що інтерпретується. Програма на мові JavaScript розміщується прямо в HTML-документі у вигляді

вихідного тексту і автоматично компілюється в байт-код при завантаженні документа в браузер. JavaScript - об'єктно-орієнтована мова з прототипним спадкуванням. Він підтримує кілька вбудованих об'єктів, а також дозволяє створювати або видаляти свої власні (призначені для користувача) об'єкти. Об'єкти можуть успадковувати властивості безпосередньо один від одного, утворюючи ланцюжок об'єкт-прототип [9]–[21].

Також використовується і в інших програмних продуктах, наприклад, node.js або Apache CouchDB. JavaScript це прототипна-орієнтований, мультипарадигмний мову з динамічною типізацією, який підтримує об'єктно-орієнтована, імперативний і декларативний (наприклад, функціональне програмування) стилі програмування [9]–[21]. У таблиці 2.1 наведено результати аналізу різних мов програмування для веброзробки.

Таблиця 2.1 – Порівняння мов програмування для програмної реалізації вебзастосунку електронних магазинів

Критерій порівняння мов програмування	Мова програмування		
	JavaScript	Java	C#
Зручність використання для програмної реалізації вебзастосунку електронних магазинів	+	±	–
Наявність засобів для розробки вебзастосунків електронних магазинів	+	±	±
Можливість обробки баз даних при створенні електронних магазинів	+	+	+
Засоби створення інтернет-застосувачів	+	±	±
Зручність роботи з HTML-документами	+	±	±

Таким чином для програмної реалізації вебзастосунку електронних магазинів обрано мову програмування JavaScript.

2.2 Вибір фреймворку для розробки вебзастосунку електронних магазинів

Проаналізуємо різні фреймворки, які можуть бути використані для розробки вебзастосунку електронних магазинів.

Bootstrap є найпопулярнішим фреймворком, у його найближчого конкурента в 3-5 разів менше співтовариство. По-друге, це не тільки css, а й js-фреймворк. Тобто в Bootstrap написані готові стилі і скрипти, для застосування яких вам достатньо всього лише прописати необхідні стильові класи і атрибути html-елементів [15], [16].

По суті, якщо говорити простою мовою, це файл або декілька файлів з готовим написаним кодом, які підключаються в сайту в секції head, після чого стає можливим використання можливостей цього фреймворка.

Фреймворки створюють для того, щоб іншим веброзробникам було легше верстати сайти. Сьогодні практично будь-який розробник після створення з нуля парочки сайтів замислюється, як йому прискорити процес розробки[15], [16].

Якщо робити розробку сайту з нуля, то доведеться подбати про дуже багато речей. Все css-стилі, все вебсценарії доведеться писати з нуля, але ж це можуть бути сотні і тисячі рядків коду. Причому ви можете зробити масу помилок у верстці. Наприклад, просто ваш шаблон буде по-різному виглядати в основних браузерах або він буде не адаптивний [15], [16].

Bootstrap варто використовувати, тому що якщо ми говоримо про фіксовані макетах, то їх легко зробити навіть з нуля. Просто створюємо блоки, задаємо їм фіксовану ширину і працюємо по макету. Але у випадку з адаптивною версткою все в рази складніше. Потрібно зробити так, щоб на

будь-яких розмірах екранів ваш сайт відображався добре. Для цього вам доведеться використовувати медіа-запити [15], [16].

Фреймворк бере на себе кроссбраузерність і адаптивність, а це основні речі, про які повинен подбати розробник. Але з bootstrap реалізувати їх дуже просто. Це дозволяє створити html-шаблон навіть людині, який раніше дуже мало займався версткою і особливо не знайомий з css. Також bootstrap ідеально підходить при роботі в команді. Верстка на bootstrap при належному вмінні і розумінні відбувається в 3-5 разів швидше, а однаковість коду дозволить будь-якому вашому колезі внести правки. Якщо ж ми говоримо про верстку без фреймворку, то тут і кожного розробника може бути свій стиль і іншій людині доведеться витратити час на вивчення його коду [15], [16].

Є два основні недоліки цього фреймворку. Перший – коду зазвичай в бібліотеці написано більше, ніж якщо б ви написали при розробці з нуля. Тому що коли ви робите самостійно, ви реалізуєте тільки необхідний функціонал і все. У Bootstrap ж є все на всі випадки життя. Навіть те, що вам може не знадобитися. Але ця проблема дуже легко вирішується тим, що ви можете самі вибирати, які компоненти фреймворку завантажити в css-файл. Наприклад, ви взагалі можете завантажити тільки сітку, а все інше робити самостійно. Другий недолік - шаблонний дизайн. Але і ця проблема легко вирішується, тому що вона буде існувати тільки в тому випадку, якщо ви будете використовувати тільки готові компоненти фреймворку і нічого ніколи не кастомизировать під себе [15], [16].

В останні роки стали популярні односторінкові додатки (single-page application, SPA^{4 6}). Фреймворки, такі як Angular, Ember і Backbone, допомагали розробникам JavaScript створювати сучасні вебдодатки за межами використання чистого (ванільного) JavaScript і jQuery. Список цих популярних рішень далеко не повний. Існує широке коло фреймворків для створення SPA.

Якщо подивитися на дати релізів, то більшість з них відносяться до першого покоління SPA: Angular 2010, Backbone 2010 і Ember 2011 року.

React був спочатку випущений Facebook в 2013 році. React - це не SPA-фреймворк, а бібліотека для розробки призначених для користувача інтерфейсів (UI). Це тільки уявлення, буква V в аббревіатурі MVC (Model View Controller). Вона дозволяє вам малювати (Render) компоненти в якості видимих елементів в браузері. Однак ціла екосистема навколо React дозволяє створювати односторінкові додатки [10].

React - це декларативна, ефективна і гнучка JavaScript бібліотека для створення користувацьких інтерфейсів. Вона дозволяє вам збирати складний UI з маленьких ізольованих шматочків коду, званих «компонентами». Вона була створена в Facebook з метою вирішити ряд проблем, пов'язаних з великомасштабними сайтами, керованими даними. У момент випуску бібліотеки в 2013 році цей проект сприймався з деякою часткою скептицизму, оскільки угоди по React досить незвичайні [10].

React є невеликою бібліотекою, що поставляється без того набору інструментів, який може знадобитися для створення ваших додатків. Але приділіть їй п'ять хвилин [10].

Так, завдяки React прямо в вашому JavaScript створюється код, схожий на HTML. І зрозуміло, всі ці теги вимагають попередньої обробки перед запуском в браузері. А для цього, швидше за все, знадобиться вбудований інструмент, такий як Webpack [10].

За допомогою React ми можемо створювати SPA(Single Page Application) [10].

Односторінковий додаток(SPA) - це програма, яка складається з єдиної HTML-сторінки і інших ресурсів (таких як JavaScript і CSS), необхідних для успішної роботи. Будь-яка взаємодія з головною або наступними її сторінками не вимагає контакту з сервером, що означає - сторінка не вимагає перезавантаження [10].

Варто відзначити, що React не відноситься до фреймворків у чистому вигляді. Це свого роду модифікована бібліотека, що дозволяє розробляти ПЗ на основі моделі MVC (Model-View-Controller), де модель відповідає за

подання даних, вигляд – відображає дані користувача, а контролер інтерпретує дії користувача та змушує модель вносити зміни) [10].

Хоч React і дозволяє створювати односторінкові додатки, це зовсім не обов'язково. React можна використовувати для вдосконалення невеликих частин вже існуючих сайтів, щоб надати додаткову інтерактивність. Код, написаний в React, може мирно співіснувати з розміткою відрендерене на сервері за допомогою PHP, або з будь-якими іншими фроненд-бібліотеками [10].

Основні переваги бібліотеки React [10]:

- в основі прості мови програмування;
- надзвичайна гнучкість програми;
- використання DOM;
- додаток витримує великі навантаження;
- react і SEO добре ладнають. Пошуковим роботам простіше переглядати сайти, поліпшується взаємодія користувачів з Вашим ресурсом;
- забезпечує незмінність батьківських даних;
- має відкриту бібліотеку даних;
- невелика вага бази даних;
- забезпечує просту міграцію між версіями.

Недоліки бібліотеки React [10]:

- великий вибір інструментів ставить в тупик;
- для освоєння всіх нюансів потрібен тривалий час.

Фреймворк React підтримує односторонню передачу даних. При цьому у React об'єкти інкапсульовані. Говорячи простою мовою, кожен із об'єктів програми відноситься до кінцевих процедур, які не вимагають дій користувача до закінчення роботи. Однак React підтримує копіювання та передачу стану. Тобто властивості прописаних об'єктів можуть бути відновлені на іншому пристрої, якщо запустити програму та повідомити стан компонентів. Отже, рендеринг буде ідентичним, на екранах обох пристроїв буде та сама «картинка» [10].

У таблиці 2.2 наведено порівняння JavaScript фреймворків для розробки вебзастосунків.

Таблиця 2.2 – Порівняння JavaScript-фреймворків для програмної реалізації вебзастосунків електронних магазинів

Критерій порівняння JavaScript-фреймворків	Фреймворки		
	React	Vue.js	Angular
Зручність механізму відображення кінцевого вебсайту	+	±	+
Направленість та класи залежностей	Одностороння передача даних	Одностороння передача даних	Взаємодія об'єктів у модулях (службах)
Зручність оновлення версій розроблюваного програмного забезпечення	+	+	±
Частота використання розробниками вебзастосунків, % [9]	78,1	0,8	21
Зручність використання для реалізації вебзастосунків електронних магазинів	+	±	+

Таким чином для розробки вебзастосунку електронних магазинів обрано JavaScript-фреймворк React.

2.3 Вибір серверу для функціонування вебзастосунку електронних магазинів

Для функціонування вебзастосунку електронних магазинів доцільно обрати сервер Node.js, що являє собою програмну платформу, засновану на движку V8 (здійснює трансляцію JavaScript в машинний код), що перетворює JavaScript з вузькоспеціалізованою мови в мову загального призначення. Node.js додає можливість JavaScript взаємодіяти з пристроями введення-виведення через свій API (написаний на C ++), підключати інші зовнішні бібліотеки, написані на різних мовах, забезпечуючи виклики до них з JavaScript-коду. Node.js застосовується переважно на сервері, виконуючи роль вебсервера, але є можливість розробляти на Node.js і десктопні віконні додатки (за допомогою NW.js, AppJS або Electron для Linux, Windows і macOS) і навіть програмувати мікроконтролери (наприклад, tessel і espruino). В основі Node.js лежить подієво-орієнтоване і асинхронне (або реактивне) програмування з неблокуючим введенням / висновком [13], [14].

Однією з основних привабливих особливостей Node.js є швидкість. JavaScript-код, що виконується в середовищі Node.js, може бути в два рази швидше, ніж код, написаний на компільованих мовах, на зразок C або Java, і на порядки швидше інтерпретованих мов на зразок Python або Ruby. Причиною подібного є неблокуюча архітектура платформи, а конкретні результати залежать від використовуваних тестів продуктивності, але, в цілому, Node.js - це дуже швидка платформа [13], [14].

У середовищі Node.js виконується код, написаний на JavaScript. Це означає, що мільйони фронтенд-розробників, які вже користуються JavaScript в браузері, можуть писати і серверний, і клієнтський код на одному і тому ж мовою програмування без необхідності вивчати абсолютно новий інструмент для переходу до серверної розробки. У браузері і на сервері використовуються однакові концепції мови. Крім того, в Node.js можна оперативно переходити на використання нових стандартів ECMAScript у міру їх реалізації на

платформі. Для цього не потрібно чекати до тих пір, поки користувачі оновлять браузер, так як Node.js - це серверна середовище, яке повністю контролює розробник. В результаті нові можливості мови виявляються доступними при установці підтримуючої їхньої версії Node.js [13], [14].

В Основі Node.js, крім інших рішень, лежить опенсорсний JavaScript-движок V8 від Google, який застосовується в браузері Google Chrome і в інших браузерах. Це означає, що Node.js користується напрацюваннями тисяч інженерів, які зроби середу Виконання JavaScript Chrome Неймовірно швидкої і продовжають працювати в напрямку Вдосконалення V8 [13], [14].

У традиційних мовах програмування (C, Java, Python, PHP) всі інструкції, за замовчуванням, є блокуючими, якщо тільки розробник явно не подбає про асинхронному виконанні коду. В результаті якщо, наприклад, в такому середовищі, зробити мережевий запит для завантаження якогось JSON-коду, виконання потоку, з якого зроблено запит, буде призупинено до тих пір, поки не завершиться отримання і обробка відповіді.

JavaScript значно спрощує написання асинхронного і неблокуючим коду з використанням єдиного потоку, функцій зворотного виклику (коллбеків) і підходу до розробки, заснованої на події. Кожен раз, коли нам потрібно виконати важку операцію, ми передаємо відповідним механізмом коллбек, який буде викликаний відразу після завершення цієї операції. В результаті, для того щоб програма продовжила роботу, чекати результатів виконання подібних операцій не потрібно. Подібний механізм виник в браузерах. Ми не можемо дозволити собі чекати, скажімо, закінчення виконання AJAX-запиту, не маючи при цьому можливості реагувати на дії користувача, наприклад, на клацання по кнопках. Для того щоб користувачам було зручно працювати з вебсторінками, всі, і завантаження даних з мережі, і обробка натискання на кнопки, має відбуватися одночасно, в режимі реального часу. Якщо ви створювали коли-небудь обробник події натискання на кнопку, то ви вже користувалися методиками асинхронного програмування. Асинхронні механізми дозволяють єдиному Node.js-сервера одночасно

обробляти тисячі підключень, не навантажуючи при цьому програміста завданнями з управління потоками і по організації паралельного виконання коду. Подібні речі часто є джерелами помилок. Node.js надає розробнику неблокуючих базові механізми введення виведення, і, в цілому, бібліотеки, які використовуються в середовищі Node.js, написані з використанням неблокуючих парадигм. Це робить блокуючу поведінку коду швидше винятком, ніж нормою. Коли Node.js потрібно виконати операцію вводу-виводу, на зразок завантаження даних з мережі, доступу до бази даних або до файлової системи, замість того, щоб заблокувати очікуванням результатів такої операції головний потік, Node.js ініціює її виконання і продовжує займатися іншими справами до тих пір, поки результати виконання цієї операції не будуть отримані [13], [14].

2.4 Вибір та проєктування бази даних для реалізації вебзастосунку електронних магазинів

Для реалізації вебзастосунку електронних магазинів доцільно обрати MongoDB, яка концептуально являє собою систему керування базами даних. У середині MongoDB може бути нуль або більше баз даних, кожна з яких є контейнером для інших сутностей. База даних може мати нуль або більше «колекцій». Колекція настільки схожа на традиційну «таблицю», що можна сміливо вважати її одним і тим же. Колекції складаються з нуля або більше «документів». Знову ж, документ можна розглядати як «рядок». Документ складається з одного або більше «полів», які подібні до «Колонок». «Індекси» в MongoDB майже ідентичні таким же в реляційних базах даних. «Курсори» відрізняються від попередніх п'яти концепцій, але вони дуже важливі (хоча часом їх обходять увагою) і заслуговують окремого обговорення. Важливо розуміти, що коли ми робимо запит у MongoDB будь яких даних, то вона повертає курсор, з яким ми можемо робити все що завгодно - підраховувати,

пропускати певну кількість попередніх записів - при цьому, не завантажуючи самі дані [9]-[21].

Підводячи підсумок, MongoDB складається з «баз даних», які складаються з «Колекцій». «Колекції» складаються з «документів». Кожен «документ» складається з «полів». «Колекції» можуть бути проіндексовані, що покращує продуктивність вибірки і сортування. І, нарешті, отримання даних з MongoDB зводиться до отримання «курсора», який віддає ці дані за потребою. СУБД MongoDB відноситься до NoSQL баз даних, основною рисою яких є нереляційний характер і відповідно мову запитів, відмінну від SQL. У MongoDB як мова запитів використовується JavaScript і JSON-структури. Вибір настільки невластивої мови запиту пояснюється тим, що ця документо-орієнтована СУБД використовує JSON-формат для подання документів і виведення результатів. Фізично JSON-структури зберігаються в бінарному BSON форматі [9]-[21].

2.5 Розробка моделі створення вебзастосунків електронних магазинів

Як зазначено у першому розділі, існуючі методології розробки вебзастосунків приділяють велику увагу процесам проєктування, а також враховують велику кількість користувачів та більш активний період обслуговування вебзастосунків. Проте при використанні деяких моделей не вистачає певних можливостей для проєктування структури вебзастосунків та функціональних можливостей редагування або створення вебзастосунків з урахуванням потреб користувача.

Тому доцільно розробити модель створення вебзастосунків електронних магазинів у вигляді діаграм, поданих у нотації UML, що дозволяють усувати наявність недоліків існуючих методологій розробки вебзастосунків з урахуванням потреб замовників та кінцевих споживачів.

Запропоновану модель створення вебзастосунків електронних магазинів подамо як кортеж $M_{vz} = \langle M_1, M_2, M_3, \dots \rangle$, що представляє собою сукупність інформаційних моделей у вигляді UML-діаграм, представлених у нотації UML 2.5. Таким чином, запропонована модель являє собою множину абстрактних моделей $M_1, M_2, M_3, \dots$ вебзастосунку на основі структурних діаграм, діаграм поведінки та діаграм взаємодії.

Модель компонентів вебзастосунку електронного магазину подамо у вигляді структурної діаграми компонентів, наведеної на рис. 2.1.

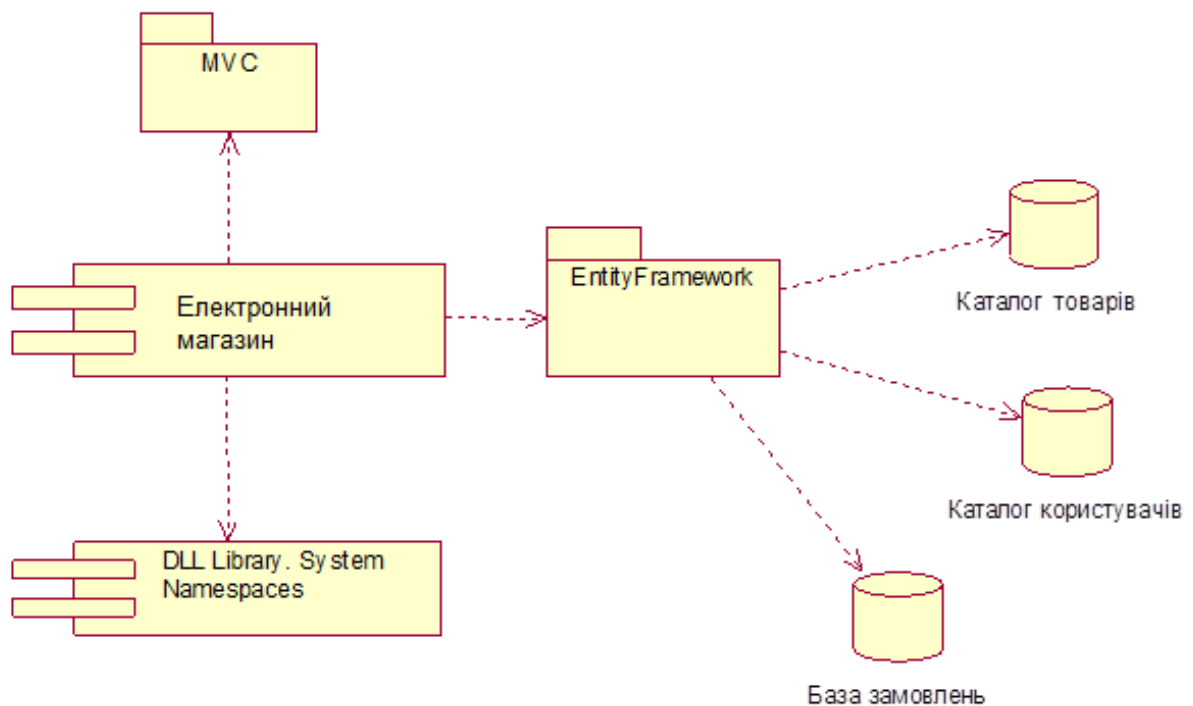


Рисунок 2.1 – Діаграма компонентів вебзастосунку електронного магазину у нотації UML

Можливі варіанти поведінки вебзастосунку електронного магазину подамо у вигляді діаграми станів (рис. 2.2) та діаграми прецедентів (рис. 2.3), представлених у нотації UML 2.5.

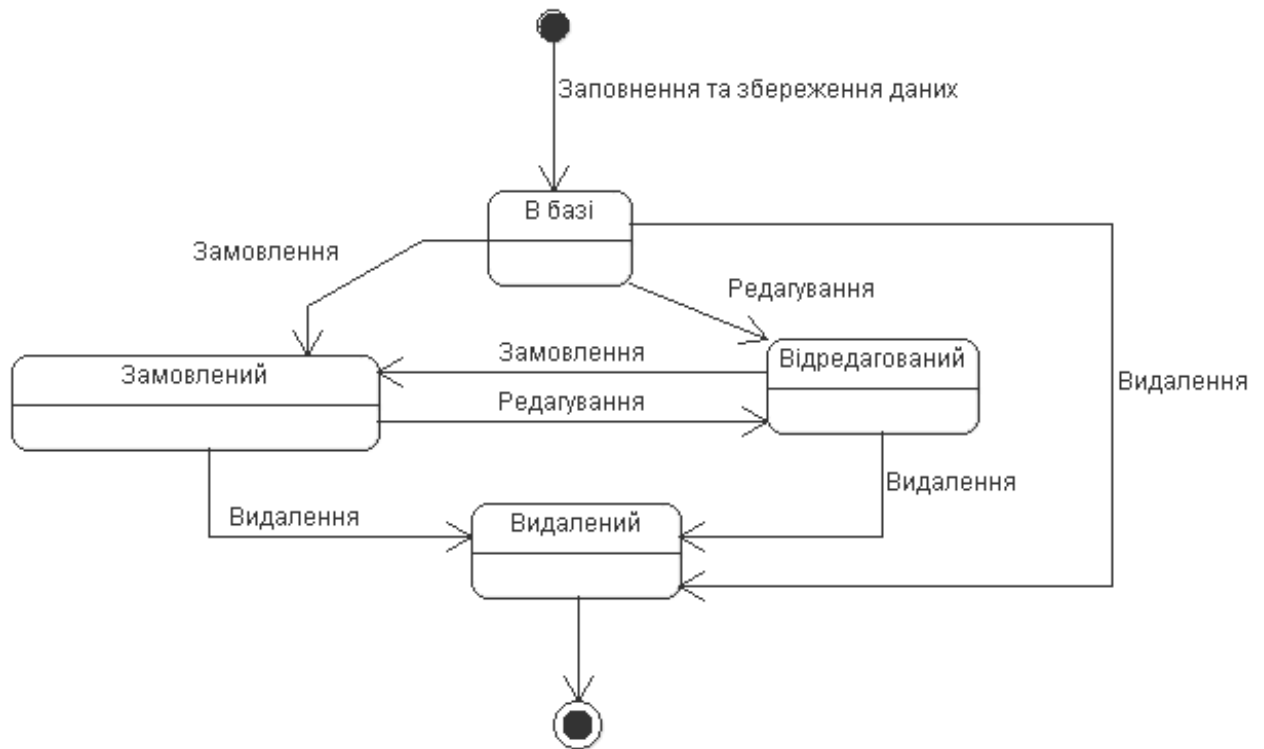


Рисунок 2.2 – Діаграма стану для замовлення товару

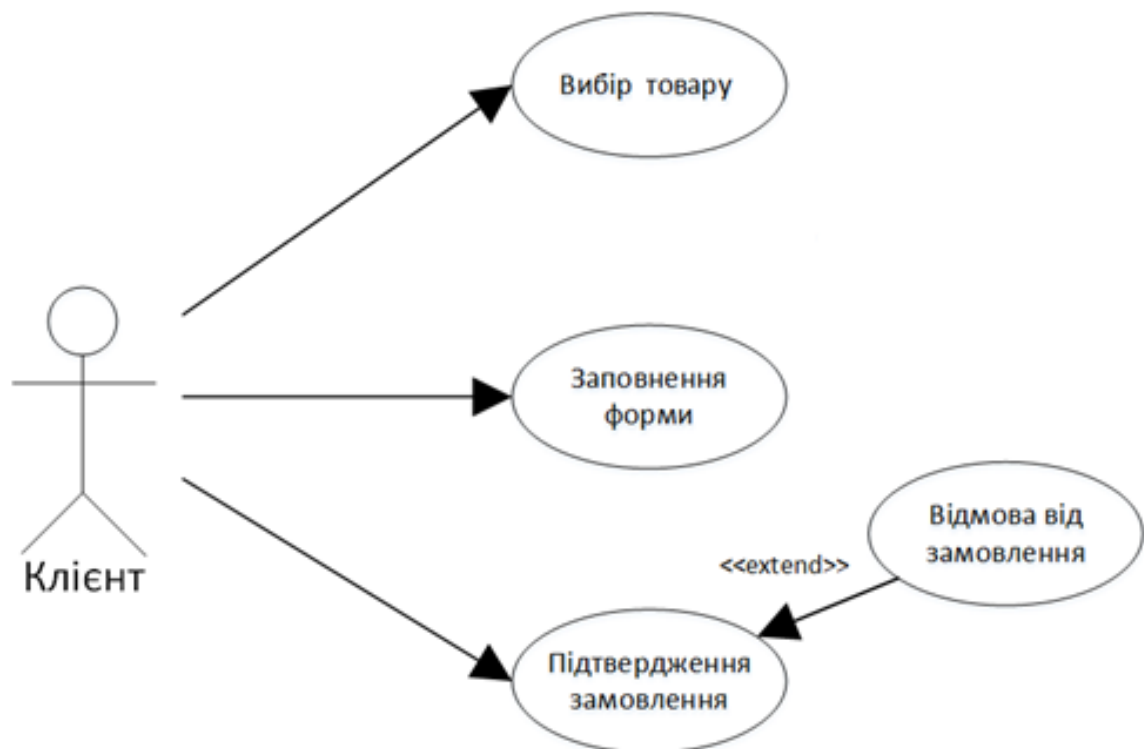


Рисунок 2.3 – Діаграма прецедентів «Замовлення товару»

Таким чином, розроблено модель створення вебзастосунків електронних магазинів у вигляді діаграм, поданих у нотації UML.

Запропонована модель $M_{vz} = \langle M_1, M_2, M_3, \dots \rangle$ являє собою множину абстрактних моделей вебзастосунку на основі структурних діаграм, діаграм поведінки та діаграм взаємодії, що дозволяють розробляти вебзастосунки електронних магазинів з урахуванням потреб замовників та кінцевих споживачів.

2.6 Висновки за розділом 2

Для створення вебзастосунків електронних магазинів було обрано мову програмування JavaScript, яка є досить зручною для розробки програм подібного типу. Як фреймворк було обрано React, який являє собою JavaScript-фреймворк, досить часто використовується розробниками вебзастосунків є досить зручним для створення нових версій розроблюваного програмного забезпечення.

Для реалізації вебзастосунку електронних магазинів обрано MongoDB, яка концептуально являє собою систему керування базами даних.

Розроблено модель створення вебзастосунків електронних магазинів у вигляді діаграм, поданих у нотації UML. Запропонована модель $M_{vz} = \langle M_1, M_2, M_3, \dots \rangle$ являє собою множину абстрактних моделей вебзастосунку на основі структурних діаграм, діаграм поведінки та діаграм взаємодії, що дозволяють розробляти вебзастосунки електронних магазинів з урахуванням потреб замовників та кінцевих споживачів.

3 ОПИС ПРОГРАМИ

3.1 Опис структури вебзастосунку електронного магазину

Структуру розробленого вебзастосунку електронного магазину наведено на рис. 3.1.



Рисунок 3.1 – Структура вебзастосунку електронного магазину

Як видно з рисунку 3.1, вебзастосунок електронного магазину складається з трьох модулів, кожний з яких містить власні підмодулі, які взаємодіють між собою та відповідають за виконання певних функцій програмного забезпечення електронного магазину.

Інтерфейсний модуль містить компоненти, що дозволяють забезпечити взаємодію користувачів програми (осіб, що відвідують сайт електронного магазину) з програмою.

Функціональний модуль забезпечує виконання основних функцій вебзастосунку електронного магазину, пов'язаних з поданням користувачу переліку наявних у магазині товарів, пошуком товарів за критеріями споживача (користувача), здійсненням замовлення певних товарів, скасування замовлення. Тут містяться також функції роботи користувача-адміністратора з програмою (додавання нових товарів, редагування інформації про існуючі товари, зміна статусу замовлення користувача-споживача, видалення

інформації про товари, якими електронний магазин перестає займатися, видалення користувачів та ін.).

Модуль роботи з базою даних призначений для роботи з базою даних програмної системи та пов'язаний з іншими модулями вебзастосунку електронного магазину.

3.2 Особливості реалізації архітектури клієнтської частини вебзастосунку електронного магазину

При розробці архітектури клієнтської частини було застосовано підхід компоненти-контейнери. Він заснований на розділенні компонентів залежно від того для чого вони призначені.

На рисунку 3.2 наведено схему взаємодії між компонентами (елементами) клієнтської частини вебзастосунку електронного магазину



Рисунок 3.2 – Схема взаємодії між елементами клієнтської частини вебзастосунку електронного магазину

Компоненти-контейнери (Containers) є батьками для компонентів-уявлень і забезпечують зв'язок між уявленнями і іншими частинами

вебзастосунку електронного магазину. Компонент-контейнер створюється як будь-який інший компонент React. Він має метод `render`, але не створює елементів DOM, а тільки повертає компонент-уявлення. Вони застосовуються для роботи з даними. Можуть робити запити на сервер, оброблювати отримані дані та розподіляти їх між компонентами [14]-[17].

Компоненти-уявлення (Components) не представляють, звідки беруться дані, вони нічого не знають про стан програми. Компоненти-уявлення вебзастосунку електронного магазину не змінюють дані. Фактично, будь-який компонент, який одержує властивості від батька, повинен тримати їх незмінними. У той же час, вони можуть будь-яким чином формувати дані (наприклад, конвертувати Unix timestamp в читану дату-час).

Основні контейнери вебзастосунку електронного магазину:

- Auth – контейнер, що виконує функції авторизації та реєстрації. У ньому ми оброблюємо дані, які користувач вводить при реєстрації або логіні та передає їх у `redux-saga` для відправлення на сервер;

- Header – оброблює логіку пошуку товарів. Містить у собі інформацію про користувача, що ввійшов у систему, та оброблює функцію `logout`;

- ManageProducts – відповідає за роботу з продуктами. Виконує запит на сервер за списком товарів, що були додані користувачем. Має логіку додавання, видалення та редагування продуктів;

- Products – оброблює запит за товарами;

- Single Product – контейнер додаткової інформації про товар;

- UnauthorizedPage – контейнер виконує функцію заглибки, коли виконується перехід до сторінок доступних тільки авторизованим користувачам.

- UserPage – функції оновлення даних користувача.

Основні компоненти вебзастосунку електронного магазину:

- Error Boundaries - відловлюють помилки JavaScript в будь-якому місці дерев їх дочірніх компонентів, зберігають їх в журналі помилок і виводять запасний UI замість звалився дерева компонентів. Запобіжники відловлюють

помилки при рендеринге, в методах життєвого циклу і конструкторах дерев компонентів, розташованих під ними.

- MainPage – головна сторінка;
- ProductForm - форма редагування / додавання продуктів;
- ProductList – відображення списку продуктів;
- SignPage – сторінка реєстрації / авторизації ;
- UnauthorizedBlock – заглушка для неавторизованих користувачів ;
- UserForm – форма редагування інформації користувача;
- UserInfo - сторінка користувача.

Redux використовується для управління даними і передачі їх між React-компонентами. Дозволяє зберігати отримані дані з сервера і при перемиканні між сторінками або при виконанні дій, які можуть викликати перезавантаження сторінки, нам не потрібно кожен раз робити новий запит, якщо дані лежать в сховище Redux. Також, якщо нам потрібно відображати отримані дані в декількох місцях інтерфейсу або на різних сторінках, у нас є можливість використовувати дані з Store [21].

Store являє собою сховище даних у Redux, у якому ми за допомогою actions та reducers ми зберігаємо, або оновлюємо дані. Для роботи зі сховищем було використано бібліотеку redux-saga. Вона покликана спростити і покращити побічні ефекти (тобто такі дії, як асинхронні операції, наприклад, завантаження даних, і "брудні" дії, такі, як доступ до браузерних кешу), зробити легкими в тестуванні і краще справлятися з помилками. Асинхронні запити – такі запити, при яких браузер не блокує взаємодію користувача з сайтом [21].

У розробленому вебзастосунку використовуються також компоненти LocalStorage – об'єкти, які можуть містити у собі дані у форматі ключ-значення. У розроблюваному вебзастосунку в LocalStorage записується jwtToken, який використовується для передачі даних аутентифікації. Коли користувач успішно авторизується ми зберігаємо токен у LocalStorage. Далі коли користувач заходить на сайт відправляється запит на сервер, який

перевіряє токен на валідність. Якщо токен валідний, ця інформація записується в стор і всі наступні перевірки, наприклад спроба перейти на сторінку доступну тільки авторізованим користувачам, перевірка виконується на стороні клієнта. І якщо користувач авторизований ми виконуємо запит користувача, якщо немає - то якимось чином обробляємо помилку - наприклад, перенаправляємо на сторінку-заглушку [21].

3.3 База даних вебзастосунку електронного магазину

База даних вебзастосунку електронного магазину зберігається на сервері. На рисунку 3.3 наведено концептуальну модель бази даних розробленого вебзастосунку електронного магазину.

Як видно з рисунку, база даних складається з таких сутностей:

- користувач (user) – містить інформацію про користувача (клієнта) електронного магазину;
- замовлення товару (product_order) – зберігає інформацію про замовлення товару клієнтом електронного магазину;
- замовлення (order) – зберігає інформацію про оброблення замовлення електронним магазином;
- товар (product) – містить інформацію про товар електронного магазину;
- категорія (categorie) – містить інформацію про категорію (тип) товару електронного магазину.

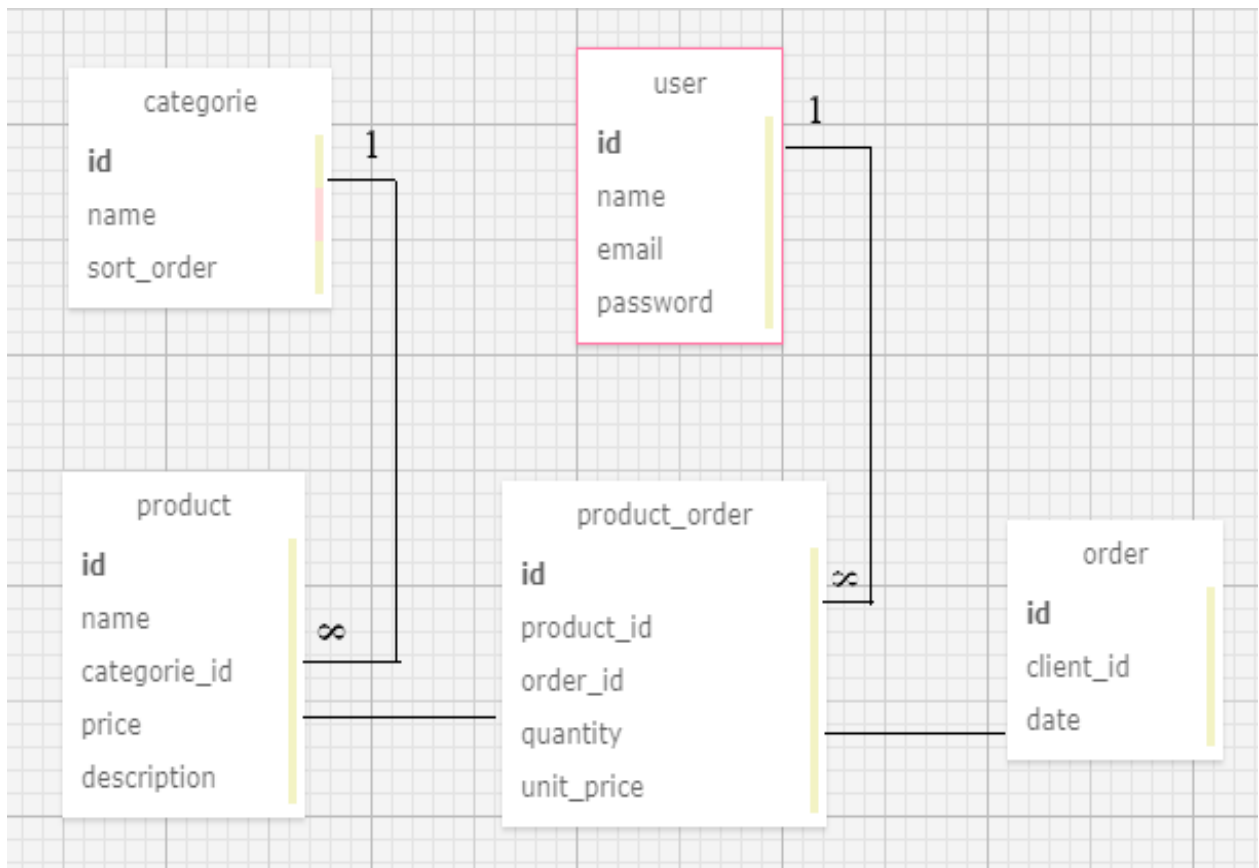


Рисунок 3.3 – Концептуальна модель бази даних розробленого вебзастосунку електронного магазину

3.4 Опис інтерфейсу користувача та його компонентів

Інтерфейс користувача розробленого вебзастосунку електронного магазину представляє собою елементи та компоненти, які можуть виявити вплив на взаємодію користувача з програмним забезпеченням електронного магазину. У розробленій програмі використовуються такі засоби інтерфейсу користувача засоби відображення інформації, показана інформація, формати та коди; командні режими, мова користувацького інтерфейсу; обладнання та технології введення даних; діалоги, взаємодія та транзакції між користувачами та комп'ютерами; зворотній зв'язок з користувачем; підтримка прийнятих рішень у конкретній предметній області; порядок використання додатку [7]–[21].

При розробленні інтерфейсу користувача вебзастосунку електронного магазину враховувався принцип «чим простіше, тим краще», який полягає у тому, що на екрані повинен бути лише інформативний, корисний для користувача, текст і контактна інформація, а шрифт повинен легко читатися. Графічні елементи є чіткими, виразними та швидко завантажуються [11-12]. Оскільки людський глаз переглядає сторінку сайту зверху вниз, то найбільша увага концентрується на верхній частині лівої частини. Тому на цьому місці у розробленому вебзастосунку розміщується найважливіша інформація: назва фірми, логотип, саме назва сайту та ін. [7]–[21].

З метою спрощення роботи користувача з вебзастосунком електронного магазину у верхній частині вебсторінки розробленого вебзастосунку не використовується авангардні та складні композиції з багатьох рисунків. Використовуються інтерфейсні компоненти, які дозволяють покупцю повинен без проблем шукати цікавий товар та мати можливість полегшити пошук інформації про потрібний товар, крім того в опис товару крім тексту додається до кількох фотографій. На вітрині розміщуються дані про діяльність фірм-виробників товарів [7]–[21].

Дизайн бланкового вебзастосунку електронного магазину був створений в м'яких кольорах. Кольорова палітра зображена на рисунку 3.4.

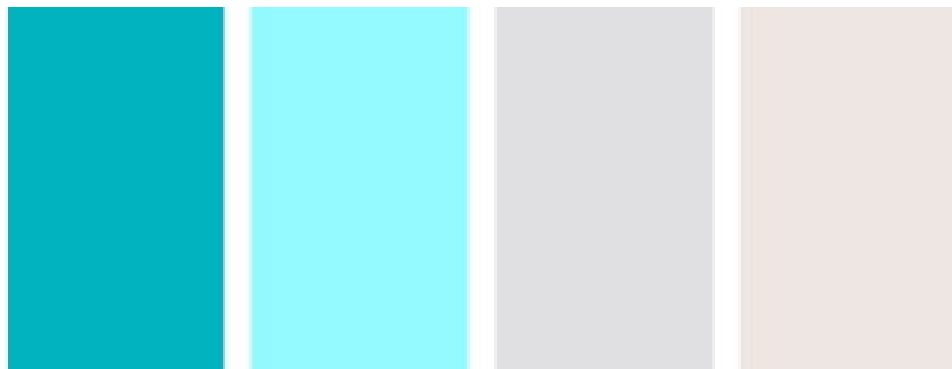


Рисунок 3.4 – Палітра кольорів вебзастосунку електронного магазину

У ході роботи було створено гумовий (адаптивний) дизайн, що представляє собою табличний дизайн, в якому ширина однієї або декількох

областей задається у відсотках від ширини вікна, що надає можливість користувачеві користуватися додатком з пристроїв будь яких розмірів [7]–[21].

Такий дизайн був створений за допомогою сітки Bootstrap, яка являє собою систему гнучких сіток для мобільних пристроїв, для зручної верстки та розташування елементів будь-яких видів і розмірів. У програмі використовується система «12 колонок», 5-ти адаптивних ярусів, схема сітки зазначена на рисунку 3.5. Система сіток Bootstrap 4 використовує контейнери, ряди і колонки, щоб зручно розташовувати вміст. Бутстрап реалізований за допомогою флексбокса і повністю адаптивний до різних пристроїв [7]–[21].

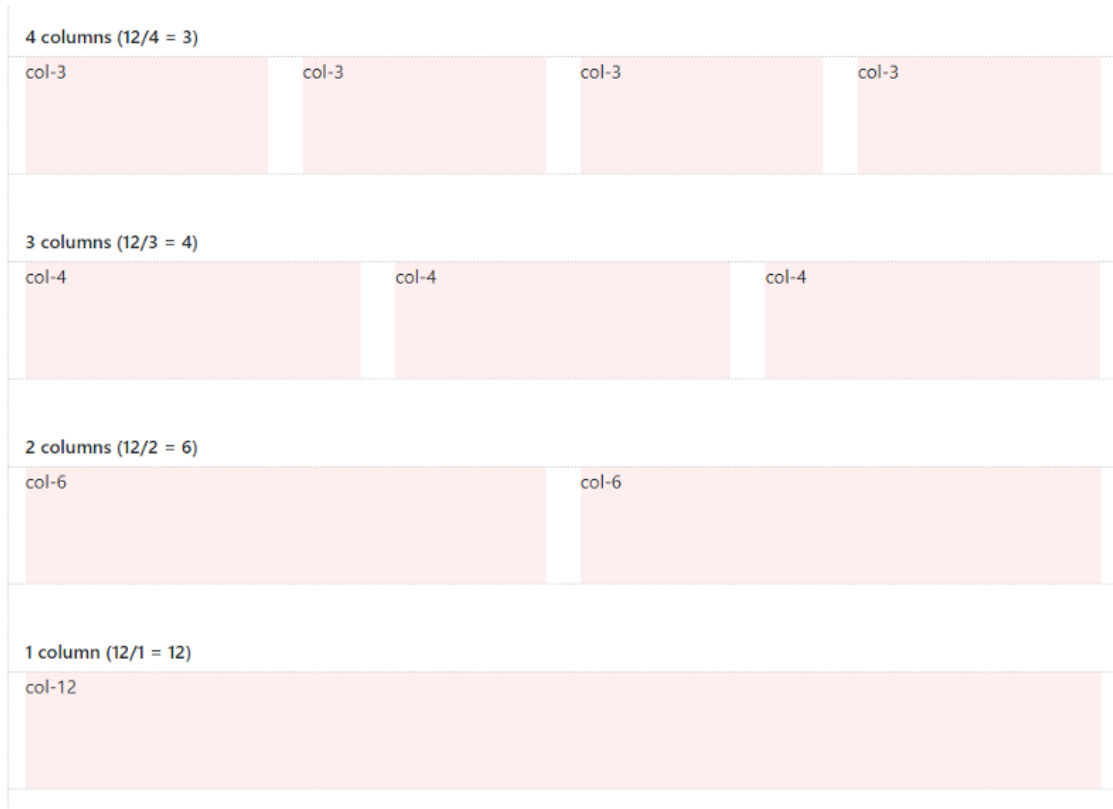


Рисунок 3.5 – Використовувана сітка Bootstrap у вебзастосунку електронного магазину

У створеному вебзастосунку електронних магазинів є такі форми: реєстрація, авторизація, додавання продукту, редагування продукту, редагування даних користувача. Введені користувачем дані проходять валідацію на стороні клієнта перед відправкою на сервер, що дозволяє

зменшити кількість непотрібних запитів на сервер, якщо були заповнені не всі поля, або вони були заповнені невірно. При перевірці виводиться повідомлення вірні введені дані чи ні. Валідація реалізована за допомогою RegExp, який є конструктором регулярного виразу для зіставлення тексту з шаблоном [7]–[21].

3.5 Висновки за розділом 3

В цьому розділі описано структуру вебзастосунку електронного магазину. Визначено, що вебзастосунок електронного магазину складається з трьох модулів, кожний з яких містить власні підмодулі, які взаємодіють між собою та відповідають за виконання певних функцій програмного забезпечення електронного магазину. Описано особливості реалізації архітектури клієнтської частини вебзастосунку електронного магазину.

Наведено концептуальну модель бази даних розробленого вебзастосунку електронного магазину. Надано опис інтерфейсу користувача та його компонентів.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програми

Програмний продукт у вигляді вебзастосунку призначений для організації роботи електронних магазинів.

Вебзастосунок електронних магазинів забезпечує виконання функцій:

- ведення бази даних про товари електронного магазину;
- реєстрація користувачів;
- авторизація зареєстрованих користувачів;
- можливість пошуку товарів за критеріями;
- додавання обраних товарів у корзину;
- оформлення замовлення.

Для експлуатації вебзастосунку електронних магазинів необхідні такі апаратні засоби комп'ютера на стороні клієнта:

- процесор 1.4 гігагерц (ГГц) і вище;
- оперативна пам'ять 2 гігабайти (Гб) ОЗУ;
- наявність вільного місця на жорсткому диску не менше 800 мегабайт (МБ);
- екран з роздільною здатністю 1366 x 768 пікселів.

У клієнта повинна бути наявність вільного доступу до мережі Інтернет та веббраузер для відображення вебсторінки електронного магазину.

До складу технічних засобів при роботі з вебзастосунком електронних магазинів повинен входити сервер, що задовольняє таким вимогам:

- процесор 2 гігагерц (ГГц) і вище;
- оперативна пам'ять об'ємом 4 гігабайти, або більше;
- накопичувач об'ємом не менше 120 Гб;
- наявність монітору з роздільною здатністю 1366 x 768 пікселів.

Функціональні характеристики розробленого вебзастосунку електронних магазинів відповідають технічному завданню, наведеному у

додатку А. Опис програми вебзастосунку електронних магазинів та текст програми вебзастосунку електронних магазинів наведені у додатках Б та В.

4.2 Характеристики програми

Програма вебзастосунку електронних магазинів розроблена за допомогою мови програмування JavaScript. Для забезпечення можливостей роботи з даними електронного магазину, зокрема функції створення, редагування та видалення інформації про певні товари магазину тощо, необхідна наявність на сервері системи керування базами даних MongoDB.

Час роботи вебзастосунку електронних магазинів залежить від обсягу бази даних про товари підприємства, конфігурації комп'ютера та пропускної здатності мережі від сервера до клієнта.

4.3 Інструкція по експлуатації програми

4.3.1 Звернення до програми

Для завантаження вебзастосунку електронного магазину необхідно запустити веббраузер та перейти за відповідною адресою вебсайту інтернет магазину.

4.3.2 Вхідні й вихідні дані

Вхідними даними до програми вебзастосунку електронного магазину є інформація щодо товарів, які споживач хоче придбати, інформація про споживача (при реєстрації та ідентифікації), а також інша інформація, що використовується в магазині при обробленні даних про товари.

Вихідними даними програми вебзастосунку електронного магазину є заповнена корзина товарів користувача, інформація щодо статусу замовлення (при роботі з клієнтською частиною програми), а також інформація про

клієнтів магазину, товари, що містяться на складі, замовлення споживачів (при роботі с серверною частиною програмою в якості адміністратору електронного магазину).

4.3.3 Повідомлення

Розроблена програма вебзастосунку електронного магазину забезпечує стійке функціонування у різних браузерх користувачів, що досягається за рахунок використання адаптивного підходу при розробці інтерфейсу сайту вебзастосунку. У випадку виникнення виняткових ситуацій, наприклад, коли користувач задає вхідні дані, що не відповідають кількості (при необхідності ввести саме кількість товару), вебзастосунок електронного магазину видає відповідні повідомлення.

4.4 Тестування та результати роботи вебзастосунку електронного магазину

Розроблена система (вебзастосунок електронного магазину) вирішує задачі, пов'язані з обробленням даних, що виникають в процесі роботи електронного магазину.

Тестування розробленого вебзастосунку електронного магазину буде відбуватися за таким сценарієм:

- користувач запускає різні веббраузери (Google Chrome, Mozilla та інші) і вводить у рядок адреси відповідне посилання на вебсайт електронного магазину та натискає клавішу enter;
- користувач (клієнт електронного магазину) натискає кнопку зареєструватися;
- клієнт обирає необхідний товар або набір товарів;
- користувач додає обрані товари у корзину;
- користувач видаляє інформацію про деякі з обраних товарів;

- користувач виходить з вебзастосунку електронного магазину;
- користувач заходить від імені адміністратора електронного магазину;
- користувач (адміністратор) переглядає інформацію про зареєстрованих клієнтів магазину;
- користувач (адміністратор) переглядає інформацію про замовлення клієнтів;
- користувач (адміністратор) змінює статус замовлення на «виконано»;
- користувач (адміністратор) виходить з системи.

Форму сайту електронного магазину з переліком товарів наведено на рисунку 4.1.

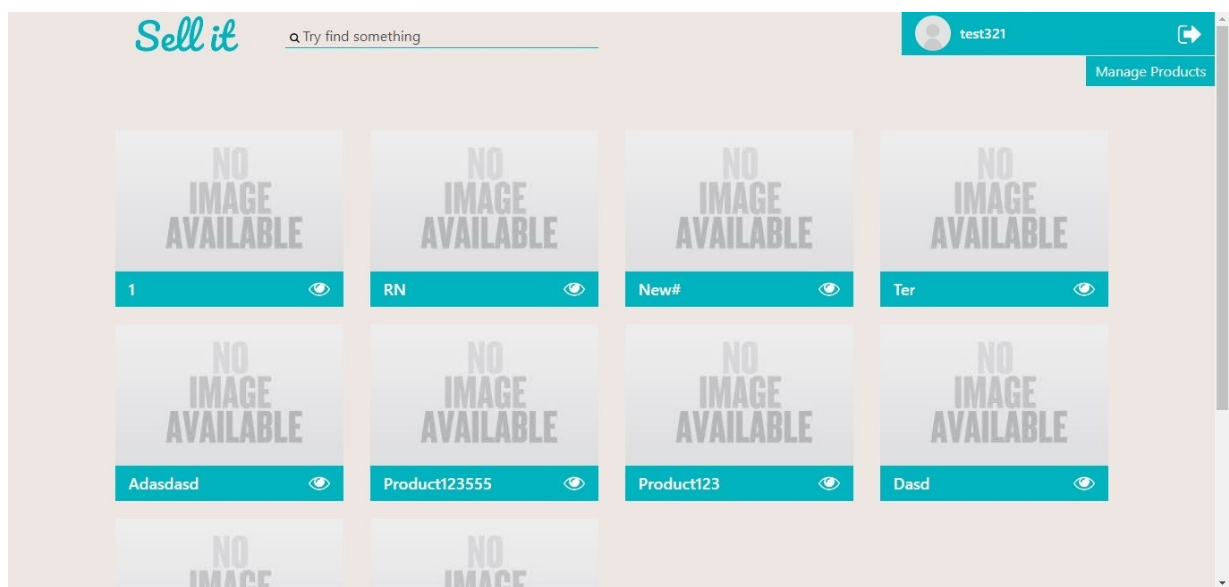


Рисунок 4.1 – Форма сайту електронного магазину з переліком товарів

Користувачу надається можливість обрати потрібний товар, переглянути інформацію про нього, додати товар у корзину.

Приклад сторінки електронного магазину, що містить опис певного товару, наведено на рисунку 4.2.

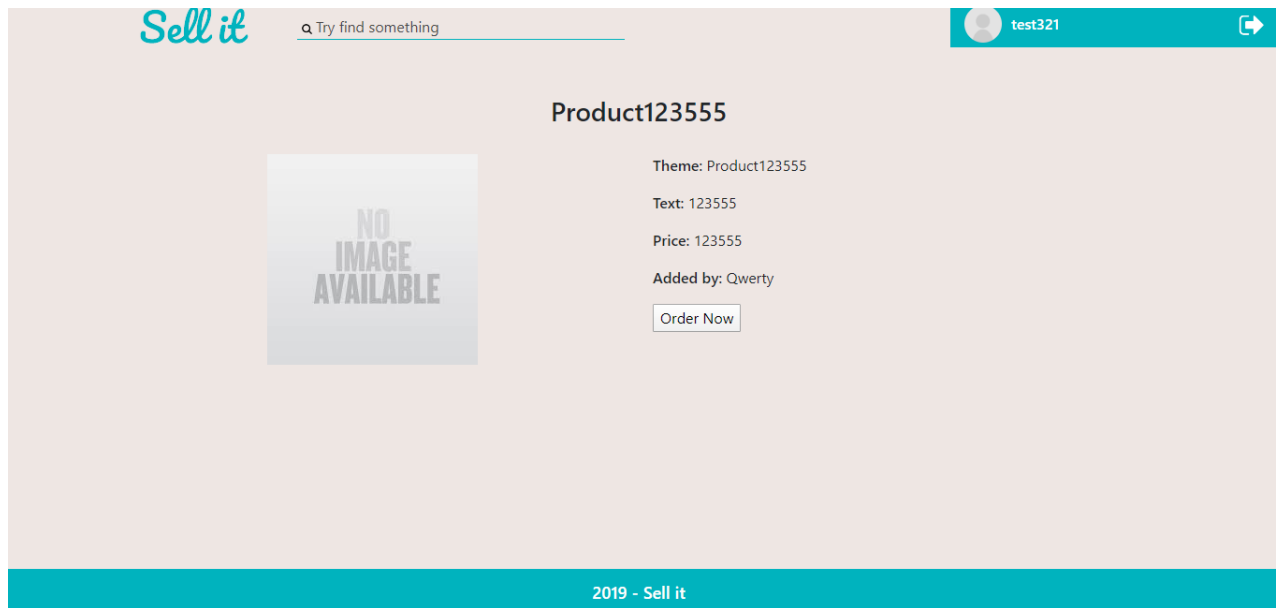


Рисунок 4.2 – Приклад сторінки електронного магазину, що містить опис певного товару

На основі проведеного аналізу за результатами тестування виявлено, що розроблений вебзастосунок електронного магазину відповідає технічному завданню та дозволяє виконувати завдання, пов'язані з обробкою інформації в процесі функціонування магазину.

4.5 Висновки за розділом 4

Описано та досліджено функціональні можливості вебзастосунку електронного магазину. Для оцінки якості роботи вебзастосунку електронного магазину було проведено його тестування, яке показало, розроблений вебзастосунок електронного магазину відповідає технічному завданню та дозволяє виконувати завдання, пов'язані з обробкою інформації в процесі функціонування магазину.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи магістра було проаналізовано та досліджено процес та методи розробки вебзастосунків електронних магазинів.

Проаналізовано засоби електронної комерції. Визначено, що у наш час віртуальні магазини суттєво зменшують витрати виробника на утримання звичайного магазину, що дає їм можливість розширювати ринки збуту. Також це надає можливість покупцю купувати будь-який товар в будь-який час в у будь-якій точці світу. Виконано класифікацію вебсайтів. Досліджено вебзастосунки електронних магазинів, визначено їх функціональні характеристики. Визначено, що типовий варіант вебзастосунку електронного магазину може бути з наступних функціональних частин: каталог товарів, пошукова система, користувальницька корзина, реєстраційна форма, форма відправки замовлення.

Досліджено засоби розробки вебзастосунків електронних магазинів. Результати проведеного аналізу показали, що на відміну від традиційної розробки програмного забезпечення, інженерія вебзастосунків тягне за собою багато аспектів та особливостей, які необхідно враховувати у процесі розробки ПЗ, таких як широке коло можливих користувачів, необхідність представити привабливе зображення та набагато більший акцент на зручності використання.

Для створення вебзастосунків електронних магазинів було обрано мову програмування JavaScript, яка є досить зручною для розробки програм подібного типу. Як фреймворк було обрано React, який являє собою JavaScript-фреймворк, досить часто використовується розробниками вебзастосунків є досить зручним для створення нових версій розроблюваного програмного забезпечення. Для реалізації вебзастосунку електронних магазинів обрано MongoDB, яка концептуально являє собою систему керування базами даних.

Наукова новизна роботи полягає у тому, що розроблено модель створення вебзастосунків електронних магазинів у вигляді діаграм, поданих у нотації UML. Запропонована модель являє собою множину абстрактних моделей вебзастосунку на основі структурних діаграм, діаграм поведінки та діаграм взаємодії, що дозволяють розробляти вебзастосунки електронних магазинів з урахуванням потреб замовників та кінцевих споживачів.

Практичне значення роботи полягає у тому, що на основі запропонованої моделі створення вебзастосунків електронних магазинів у вигляді діаграм, поданих у нотації UML, розроблено вебсайт електронного магазину. У роботі описано структуру вебзастосунку електронного магазину. Визначено, що вебзастосунок електронного магазину складається з трьох модулів, кожний з яких містить власні підмодулі, які взаємодіють між собою та відповідають за виконання певних функцій програмного забезпечення електронного магазину. Описано особливості реалізації архітектури клієнтської частини вебзастосунку електронного магазину. Наведено концептуальну модель бази даних розробленого вебзастосунку електронного магазину. Надано опис інтерфейсу користувача та його компонентів

Описано та досліджено функціональні можливості вебзастосунку електронного магазину. Для оцінки якості роботи вебзастосунку електронного магазину було проведено його тестування, яке показало, розроблений вебзастосунок електронного магазину відповідає технічному завданню та дозволяє виконувати завдання, пов'язані з обробкою інформації в процесі функціонування магазину.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ахромов, Я. В. Системы электронной коммерции [Текст]: учеб. пособие для вузов / Я. В. Ахромов. – М. : Оникс, 2007. - 416 с. – ISBN 978-5-488-00970-7.
2. Isaias P. Web Site Development Methodologies [Text] / Pedro Isaias, Tomayess Issa // High Level Models and Methodologies for Information Systems. – 2014. – P. 63–81. DOI: 10.1007/978-1-4614-9254-2_4.
3. Балабанов И. Т. Торговля через виртуальный магазин [Текст] / И. Т. Балабанов // Электронная коммерция, 2004. – С. 195-197.
4. Интернет-магазин: о каких функциях и модулях нужно не забыть на первом этапе [Электрон. ресурс]. – Режим доступа: <https://readyscript.ru/text-article/internetmagazin-o-kakih-funkciyah-i-modulyah-nuzhno-ne-zabyt-na-pervom-etape/>
5. Интернет-магазин: преимущества и недостатки [Электрон. ресурс]. – Режим доступа: http://xbb.uz/web/Internet-magazin_preimucshestva_i_nedostatki.
6. Kratov S. On the Approaches to the Development of a Scientific Organization Website [Text] / S. Kratov // 3rd Conference on Computer Technology and Applications, Vladivostok, 18-25 Aug. 2018. – DOI: 10.1109/RPC.2018.8482177.
7. HTML [Электрон. ресурс]. – Режим доступа: <https://developer.mozilla.org/ru/docs/Web/HTML>.
8. CSS [Electronic resource]. – Access mode: <https://developer.mozilla.org/en-US/docs/Glossary/CSS>.
9. Сравнение JavaScript фреймворков Vue.js, React и Angular [Электрон. ресурс]. – Режим доступа: <https://mkdev.me/posts/sravnenie-javascript-freymvorkov-vue-js-react-i-angular-2019>.
10. Алекс Бэнкс, Ева Порселло React и Redux. Функциональная вебразработка [Текст], 2018. – 336 с.

11. Redux: шаг за шагом [Electronic resource]. – Access mode: <https://medium.com/devschacht/redux-step-by-step-e6c42a9b00cd>.
12. Robin Wieruch and Alexey Pyltsyn. The Road to Learn React [Text], 2019. – 229 p.
13. Node.js [Электрон. ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Node.js>.
14. Руководство по Node.js [Электрон. ресурс]. – Режим доступа: <https://habr.com/ru/company/ruvds/blog/422893/>.
15. Что такое Bootstrap? [Электрон. ресурс]. – Режим доступа: <https://webformyself.com/chto-takoe-bootstrap/>.
16. Что такое Bootstrap? [Электрон. ресурс]. – Режим доступа: <https://bootstrap-4.ru/docs/4.0/layout/grid/>.
17. CSS Modules [Electronic resource]. – Access mode: <https://github.com/css-modules/css-modules>.
18. Марк Хасслер. Вебаналитика = Web Analytics [Текст]. – М.: Эксмо, 2010. – 432 с. – ISBN 978-5-699-36444-2.
19. Эрик А. Мейер. CSS. Карманный справочник [Текст]. – М.: Диалектика-Вильямс, 2019. – 208 с. – ISBN 978-5-6041394-1-7.
20. Этан Браун Изучаем JavaScript: руководство по созданию современных вебсайтов [Текст]. – М.: Диалектика-Вильямс, 2016. – 368 с. – ISBN 978-5-9908463-9-5.
21. redux-saga [Электрон. ресурс]. – Режим доступа: <https://ru.redux-saga.js.org/>.

ДОДАТОК А
Технічне завдання

Вступ

Програмне забезпечення у вигляді вебзастосунку електронного магазину може використовуватися для вирішення задач, пов'язаних з діяльністю продажу товарів через мережу Інтернет.

А.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Дослідження та програмна реалізація засобів розробки вебзастосунків електронних магазинів», затверджене наказом Національного університету «Запорізька політехніка» № 435 від 5 листопада 2021 р.

А.2 Призначення розробки

Програмний продукт у вигляді вебзастосунку призначений для організації роботи електронних магазинів. Розроблена програма здатна виконувати ведення бази даних про товари магазину та споживачів (клієнтів) магазину, а також підтримувати процес онлайн придбання товарів магазину клієнтами.

А.3 Основні вимоги до програми, що розробляється

А.3.1 Вимоги до функціональних характеристик

Вебзастосунок електронного магазину повинна забезпечувати виконання наступних функцій:

- ведення бази даних про товари електронного магазину;
- реєстрація користувачів;
- авторизація зареєстрованих користувачів;
- можливість пошуку товарів за критеріями;

- додавання обраних товарів у корзину;
- оформлення замовлення.

А.3.2 Вимоги до інтерфейсу програми

Комп'ютерна програма у вигляді вебзастосунку електронного магазину повинна забезпечувати можливість відображення інформації за допомогою html-сторінок у різних браузерях. Взаємодія користувача з програмою у вигляді вебзастосунку електронного магазину можлива за допомогою веббраузера.

А.3.3 Вимоги до надійності

Програма у вигляді вебзастосунку електронного магазину повинна обробляти виняткові ситуації.

А.3.4 Умови експлуатації

Експлуатація програми у вигляді вебзастосунку електронного магазину повинна здійснюватися відповідно до документації на розроблене програмне забезпечення електронного магазину. Програма у вигляді вебзастосунку електронного магазину може бути записана як на диску.

А.3.5 Вимоги до складу та параметрів технічний засобів

Для експлуатації програмного продукту у вигляді вебзастосунку електронного магазину необхідні такі програмно-технічні засоби:

- процесор 1.4 гігагерц (ГГц) і вище;
- оперативна пам'ять 2 гігабайти (Гб) ОЗУ;

- наявність вільного місця на жорсткому диску не менше 800 мегабайт (МБ);
- екран з роздільною здатністю 1366 x 768 пікселів.

А.3.6 Вимоги до маркування і пакування

Програма у вигляді вебзастосунку електронного магазину може поставлятися на різних носіях. На упакуванні повинна бути назва програми «Вебзастосунок електронного магазину».

А.4 Вхідні дані до роботи

Вхідними даними до програми вебзастосунку електронного магазину є інформація щодо товарів, які споживач хоче придбати, інформація про споживача (при реєстрації та ідентифікації), а також інша інформація, що використовується в магазині при обробленні даних про товари.

Вихідними даними програми вебзастосунку електронного магазину є заповнена корзина товарів користувача, інформація щодо статусу замовлення (при роботі з клієнтською частиною програми), а також інформація про клієнтів магазину, товари, що містяться на складі, замовлення споживачів (при роботі з серверною частиною програмою в якості адміністратору електронного магазину).

ДОДАТОК Б
Опис програми

Б.1 Загальні відомості

Програма у вигляді вебзастосунку електронного магазину розроблена за допомогою фреймворку React та мови програмування JavaScript.

Б.2 Функціональне призначення

Програмний продукт у вигляді вебзастосунку призначений для організації роботи електронних магазинів.

Програма у вигляді вебзастосунку електронного магазину забезпечує виконання наступних функцій:

- ведення бази даних про товари електронного магазину;
- реєстрація користувачів;
- авторизація зареєстрованих користувачів;
- можливість пошуку товарів за критеріями;
- додавання обраних товарів у корзину;
- оформлення замовлення.

Б.3 Опис логічної структури

Програма у вигляді вебзастосунку електронного магазину представляє застосунок, що складається з різних програмних модулів.

Для реалізації програми у вигляді вебзастосунку електронного магазину було створено програмні модулі. У процесі розробки вони були розподілені по файлам по відповідним каталогам та файлам.

Б.4 Використані технічні засоби

Для експлуатації програмного продукту у вигляді вебзастосунку електронного магазину необхідні такі засоби:

- процесор 1.4 гігагерц (ГГц) і вище;
- оперативна пам'ять 2 гігабайти (Гб) ОЗУ;
- наявність вільного місця на жорсткому диску не менше 800 мегабайт (МБ);
- екран з роздільною здатністю 1366 x 768 пікселів.

Б.5 Виклик і завантаження

Для завантаження вебзастосунку електронного магазину необхідно запустити веббраузер та перейти за відповідною адресою вебсайту інтернет магазину.

Для початку роботи з програмою та здійснення придбань товарів у магазині необхідно ідентифікувати клієнта.

Б.6 Вхідні та вихідні дані

Вхідними даними до програми вебзастосунку електронного магазину є інформація щодо товарів, які споживач хоче придбати, інформація про споживача (при реєстрації та ідентифікації), а також інша інформація, що використовується в магазині при обробленні даних про товари.

Вихідними даними програми вебзастосунку електронного магазину є заповнена корзина товарів користувача, інформація щодо статусу замовлення (при роботі з клієнтською частиною програми), а також інформація про клієнтів магазину, товари, що містяться на складі, замовлення споживачів (при роботі з серверною частиною програмою в якості адміністратору електронного магазину).

ДОДАТОК В
Текст програми

```

products.js
export const fetchProducts = (page) => {
  return {
    type: "FETCH_PRODUCTS_REQUEST",
    payload: page
  }
}

export const fetchSingleProduct = (id) => {
  return {
    type: "FETCH_SINGLE_PRODUCT_REQUEST",
    payload: id
  }
}

export const searchValueChange = (value) => {
  return {
    type: "SEARCH_PRODUCTS_REQUEST",
    payload: value
  }
}

export const clearSingleProductState = () => {
  return {
    type: "CLEAR_SINGLE_PRODUCT_STATE",
  }
}

export const addProduct = (data) => {
  return {
    type: "ADD_PRODUCT_REQUEST",
    payload: data
  }
}

export const deleteProduct = (id) => {
  return {
    type: "DELETE_PRODUCT_REQUEST",
    payload: id
  }
}

export const fetchOwnProducts = () => {
  return {
    type: "FETCH_OWN_PRODUCTS_REQUEST"
  }
}

export const updateProduct = (data) => {
  return {
    type: "UPDATE_PRODUCT_REQUEST",
    payload: data
  }
}

export const userLogin = (data) => {
  return {
    type: "USER_LOGIN_REQUEST",
    payload: data
  }
}

export const userSignUp = (data) => {

```

```

        return {
          type: "USER_SIGN_UP_REQUEST",
          payload: data
        }
      }

export const userLogout = () => {
  return {
    type: "USER_LOGOUT",
  }
}

export const errorClear = () => {
  return {
    type: "ERROR_CLEAR"
  }
}

export const updateUser = (data) => {
  return {
    type: "USER_UPDATE_REQUEST",
    payload: data
  }
}

export const getUserInfo = () => {
  return {
    type: "USER_INFO_REQUEST"
  }
}
}
base.js
import axios from 'axios';

const request = ({ method, url, data, headers }) => {
  return axios({
    method,
    url,
    data,
    headers
  })
}

export const get = (path, headers) => {
  return request({
    method: 'GET',
    url: BASE_URL + path,
    headers
  });
}

export const post = (path, data, headers) => {
  return request({
    method: 'POST',
    url: BASE_URL + path,
    data,
    headers
  });
}

export const deleteRequest = (path, headers) => {
  return request({
    method: 'DELETE',
    url: BASE_URL + path,
    headers
  })
}

```

```

    });
  };

export const patch = (path, data, headers) => {
  return request({
    method: 'PATCH',
    url: BASE_URL + path,
    data,
    headers
  });
}
interceptors.js
import axios from 'axios';

export default {
  setupInterceptors: (store, history) => {

    axios.interceptors.response.use(response => {
      return response;
    }, error => {
      if (error.response.status === 401) {
        store.dispatch({ type: 'USER_LOGOUT' });
      }

      if (error.response.status === 404) {
        history.push('/not-found');
      }

      if (error.response.status === 500) {
        history.push('/server-error');
      }

      return Promise.reject(error);
    });
  },
};
products.js
import { get, post, deleteRequest, patch } from './base';

export function getProducts(page = 1) {
  return get('/posters/?page=' + page);
}

export function getSingleProduct(id) {
  return get('/posters/' + id + '/');
}

export function getSearchProducts(value) {
  return get('/posters/?search=' + value);
}

export function addProduct(data, headers) {
  return post('/posters/', data, headers);
}

export const deleteProduct = (id, headers) => {
  return deleteRequest('/posters/' + id + '/', headers);
}

export const fetchOwnProducts = (headers) => {
  return get('/posters/own/', headers)
}

export const updateProducts = (id, data, headers) => {

```

```

        return patch('/posters/' + id + '/', data, headers)
    }

user.js
import { post, patch, get } from './base';

export function signIn(data) {
    return post('/login/', data);
};

export function signUp(data) {
    return post('/registration/', data);
}

export const updateUser = (data, headers) => {
    return patch('/profile/', data, headers);
}

export const getUserInfo = (headers) => {
    return get('/profile/', headers);
}
ErrorBoundary.jsx
import React from 'react';
import style from './ErrorBoundary.module.css';

class ErrorBoundary extends React.Component {
    constructor(props) {
        super(props);
        this.state = { hasError: false };
    }

    static getDerivedStateFromError() {
        return { hasError: true };
    }

    render() {
        if (this.state.hasError) {
            return (
                <div className={` ${style.error_message} flex-grow`} >
                    <h5>Something went wrong.</h5>
                </div>
            );
        }
        return this.props.children;
    }
}

export default ErrorBoundary;

NotFound.jsx
import React from 'react';
import { Link } from 'react-router-dom';
import styles from './errors.module.css';

const NotFound = () => {
    return (
        <div className={styles.error_page_wrap}>
            <h1>404 - PAGE NOT FOUND</h1>
            <p className={styles.error_page_text}>The page you are
looking for might have been removed had it's name changed or is temporary
unavailable</p>
            <Link to="/" className={styles.error_page_link}>GO TO HOME
PAGE</Link>
        </div>
    );
}

```

```

        </div>
    )
}

export default NotFound;

ServerError.jsx
import React from 'react';
import styles from './errors.module.css';

const ServerError = () => {
    return (
        <div className={styles.error_page_wrap}>
            <h1>500 - SERVER ERROR</h1>
            <h5 className={styles.error_page_text}>It's not you, it's
me</h5>
            <p>Refresh the page or try again later</p>
        </div>
    )
}

export default ServerError;

Footer.jsx
import React from 'react';
import styles from './Footer.module.css';

const Footer = () => {
    return (
        <footer className={styles.footer}>
            <p className={styles.footer__text}>2019 - Sell it</p>
        </footer>
    )
};

export default Footer;

Search.jsx
import React from 'react';
import styles from './Search.module.css';

const Search = ({ searchValue, searchValueChange }) => {
    return (
        <div className={styles.search_block + " col-lg-4 col-md-8
offset-md-2 offset-lg-0 "}>
            <input
                value={searchValue}
                onChange={ (e) =>
searchValueChange(e)}
                type="search"
                className={styles.search_block__input}
                placeholder="Try find something" />
            </div>
        )
    }

export default Search;

UserBlock.jsx
import React from 'react';
import { Link } from 'react-router-dom';
import styles from './UserBlock.module.css';
import userPhotoDefault from '../../../assets/images/user-photo.jpg';
import logoutIcon from '../../../assets/images/logout-icon.png';

const UserBlock = ({ isLogin, userInfo, userLogout }) => {
    return (
        <React.Fragment>

```

```

        <div className={styles.user_block}>
          {!isLogin ?
            <p className={styles.user_block__text}>
              Welcome, <Link to="/sign-in"
className={styles.user_block__link}>login</Link> or <Link to="/sign-up"
className={styles.user_block__link}> register </Link> for start !
            </p>
            :
            <div className={styles.user_login}>
              <Link to={`/userdetails/${userInfo.id}`}
className={styles.user_login_link}>
                <img src={userInfo.avatar ||
userPhotoDefault} alt=""/>
                <span>{userInfo.username ||
userInfo.email}</span>
              </Link>
            <div>
              <img src={logoutIcon} alt=""/> onClick={userLogout}
            </div>
          </div>
        }
      </div>
    </React.Fragment>
  )
};

export default UserBlock;

HeaderBlock.jsx
import React from 'react';
import { Link } from 'react-router-dom';
import UserBlock from '../UserBlock/UserBlock';
import Search from '../Search/Search';

import styles from '../HeaderBlock.module.css';
import logo from '../../assets/images/logo.png';

const HeaderBlock = ({ data }) => {
  return (
    <header className={styles.header}>
      <div className="container">
        <div className="row">
          <div className={styles.logo + " col-lg-2 col-md-6"}>
            <Link to="/"><img src={logo} alt=""/> </Link>
          </div>
          <Search
            searchValue={data.searchValue}
            searchValueChange={data.searchValueChange}
          />
          {data.userInfo && !data.isLoading
            ? <UserBlock
              isLogin={data.isLogin}
              userInfo={data.userInfo}
              userLogout={data.userLogout}
            />
            : null
          }
        </div>
      </div>
    </header>
  )
};

```

```

export default HeaderBlock;

MainPage.jsx
import React from 'react';
import { Link } from 'react-router-dom';

import Header from '../containers/Header';
import Footer from '../Footer/Footer';
import Products from '../containers/Products';
import ErrorBoundary from '../ErrorBoundaries/ErrorBoundary';
import eyeIcon from '../assets/images/eye.png';
import styles from './MainPage.module.css';

export const MyContext = React.createContext();

const MainPage = () => {
  return (
    <React.Fragment>
      <Header />
      <MyContext.Provider value={eyeIcon}>
        <ErrorBoundary>
          <Link
            to='manage-products'
            className={styles.manage_products}
          >
            <button>Manage Products</button>
          </Link>
          <Products />
        </ErrorBoundary>
      </MyContext.Provider>
      <Footer />
    </React.Fragment>
  )
}

export default MainPage;

ManageProductsPage.jsx
import React, { useState } from 'react';
import Popup from 'reactjs-popup';

import ProductForm from '../ProductForm/ProductForm';
import ProductList from '../ProductsList/ProductsList';
import styles from './ManageProductsPage.module.css';

const ManageProductsPage = ({ isLogin, addProduct, deleteProduct,
ownProductsList, searchValue, updateProduct }) => {

  const [isOpenModal, setModal] = useState(false);

  const handleSubmit = (data) => {
    setModal(false);
    addProduct(data);
  }

  return (
    <React.Fragment>
      <h5>Your Products</h5>
      <button
        onClick={() => setModal(true)}
        className={styles.popup_trigger}>add product</button>
      <Popup
        open={isOpenModal}
        closeOnDocumentClick

```

```

        onClose={() => setModal(false)}
      >
        <React.Fragment>
          <div className={styles.close + ' close'} onClick={()
=> setModal(false)}>
            &times;
          </div>
          <h5>New Product</h5>
          <ProductForm isLogin={isLogin}
onSubmit={handleSubmit}/>
        </React.Fragment>
      </Popup>

      {ownProductsList
        ? <ProductList
            productList={ownProductsList}
            searchValue={searchValue}
            deleteProduct={deleteProduct}
            updateProduct={updateProduct}
          />
        : null
      }
    </React.Fragment>
  )
}

export default ManageProductsPage;

ProductForm.jsx
import React from 'react';
import { Field, reduxForm } from 'redux-form';

import styles from './ProductForm.module.css';

const renderField = ({ input, label, type }) => {
  return (
    <div>
      <label>{label}</label>
      <input {...input} type={type} />
    </div>
  )
}

let NewProduct = ({ handleSubmit }) => {
  return (
    <div className="flex-grow">
      <div className="container">
        <div className="row">
          <div className={styles.form_wrapper}>
            <form
onSubmit={handleSubmit}>
              <Field name="theme" type="text"
label="Theme" component={renderField}/>
              <Field name="text" type="text" label="Text"
component={renderField}/>
              <Field name="price" type="number"
label="Price" component={renderField}/>
              <Field name="category" type="text"
label="Category" component={renderField}/>
              <input type="submit" value="Submit"/>
            </form>
          </div>
        </div>
      </div>
    </div>
  )
}

```

```

        </div>
      </div>
    </div>
  </div>
)
}

export default reduxForm({
  form: 'newProduct'
})(NewProduct);

ProductPage.jsx
import React from 'react';

import Header from '../containers/Header';
import Footer from '../Footer/Footer';
import productDefaultImg from '../assets/images/product-default.jpg';
import styles from './ProductPage.module.css';

const ProductPage = ({ singleProduct }) => {
  return(
    <React.Fragment>
      <Header />
      <div className="container flex-grow">
        <div className="row">
          {singleProduct ?
            <div className={styles.productPage}>
              <h3>{singleProduct.theme}</h3>
              <div className={styles.productInfo}>
                <div className="col-lg-6">
                  <img
src={singleProduct.images.length !== 0 ? singleProduct.images[0].file :
productDefaultImg } alt=""/>
                  </div>
                  <div className={styles.productText + "
col-lg-6">>
                    <p><span>Theme:
</span>{singleProduct.theme || 'Product Title'}</p>
                    <p><span>Text:
</span>{singleProduct.text}</p>
                    <p><span>Price:
</span>{singleProduct.price}</p>
                    <p><span>Added by:
</span>{singleProduct.owner.username}</p>
                    <button>Order Now</button>
                  </div>
                </div>
              </div>
            : null
          )
        </div>
      </div>
    </div>
    <Footer />
  </React.Fragment>
)
}

export default ProductPage;

ProductList.jsx
import React, { useState } from 'react';
import ReactPaginate from 'react-paginate';

```

```

import styles from './ProductsList.module.css';
import Product from './Product';

const ProductList = ({ productList, productsMeta, searchValue,
errorTrigger, fetchProducts, deleteProduct, updateProduct }) => {

  const mapProducts = () => {
    return productList.map((item) => {

      return (
        searchValue.length === 0 || (item.theme !== null &&
item.theme.toLowerCase().indexOf(searchValue.toLowerCase()) !== -1) ?
          <Product
            data={item}
            key={item.pk}
            deleteProduct={deleteProduct}
            updateProduct={updateProduct}
          />
          : null
        )
      ))
    }

    const [activePage, setActivePage] = useState(0);

    const handlePageClick = ({ selected }) => {
      let currentPage = selected + 1;
      fetchProducts(currentPage);
      setActivePage(selected);
    }

    return (
      <div className={styles.products + " flex-grow"}>
        <div className="container">
          { /* <button onClick={() => errorTrigger()}
className={styles.error_trigger}>Error Trigger</button> */ }
          <div className="row">
            {productList
              ? mapProducts()
              : null
            }
          </div>
          {productsMeta && productsMeta.total > 1
            ? <ReactPaginate
              previousLabel={'Prev'}
              nextLabel={'Next'}
              breakLabel={'...'}
              breakClassName={'break-me'}
              pageCount={productsMeta ? productsMeta.total :
1}

              forcePage={activePage}
              marginPagesDisplayed={2}
              pageRangeDisplayed={5}
              onPageChange={handlePageClick}
              containerClassName={styles.pagination}
              subContainerClassName={'pages pagination'}
              activeClassName={styles.pagination_active}
              previousClassName={styles.previous}
              nextClassName={styles.next}
              disabledClassName={styles.disabled}
            />
            : null
          }
        </div>
      )
    )
  }
}

```

```

        </div>
      </div>
    )
  }

  export default ProductList;

  SignPage.jsx

  import React, { Suspense } from 'react';
  import { Link, Redirect } from 'react-router-dom';
  import styles from './SignPage.module.css';
  import logo from "../../assets/images/logo-sign.png";

  const SignIn = React.lazy(() => import('./SignForms/SignIn'));
  const SignUp = React.lazy(() => import('./SignForms/SignUp'));

  const ErrorRender = ({ error }) => {
    let errorsArr = [];
    for(let key in error) {
      errorsArr.push(error[key])
    }
    return (
      <div className={styles.error_wrap}>
        {errorsArr.map((item, index) => {
          return <p id="error" key={index}>{item}</p>
        })}
      </div>
    )
  }

  const SignInBlock = (signUpDone, signInError, handleSignIn) => {
    return (
      <React.Fragment>
        {signUpDone ?
          <p
            className={styles.registration_done}>Registration done! Please login</p> :
          null}
        <ErrorRender error={signInError} />
        <Suspense fallback={<div>Loading...</div>}>
          <SignIn onSubmit={handleSignIn} />
        </Suspense>
      </React.Fragment>
    )
  }

  const SignUpBlock = (signUpError, handleSignUp) => {
    return (
      <React.Fragment>
        <ErrorRender error={signUpError} />
        <Suspense fallback={<div>Loading...</div>}>
          <SignUp onSubmit={handleSignUp} />
        </Suspense>
      </React.Fragment>
    )
  }

  const SignPage = ({ activeTab, signUpDone, signInError, signUpError,
    errorClear, handleSignIn, handleSignUp }) => {
    if (signUpDone && activeTab === "signup") return <Redirect
      to="/sign-in" />
  }

```

```

    return (
      <section className={styles.main}>
        <div className={styles.sign_block_wrap}>
          <Link to="/">
            <img src={logo} alt="" />
          </Link>

          <div className={styles.tabs_select}>
            <Link
              to="/sign-in"
              className={` ${styles.tabs_select__button}
                ${activeTab === "signin" ? styles.tabs_select__button_active : ""} `}
              onClick={() => errorClear()}
            >
              Sign In
            </Link>

            <Link
              to="/sign-up"
              className={` ${styles.tabs_select__button}
                ${activeTab === "signup" ? styles.tabs_select__button_active : ""} `}
              onClick={() => errorClear()}
            >
              Sign Up
            </Link>
          </div>

          {activeTab === "signin"
            ? SignInBlock(signUpDone, signInError, handleSignIn)
            : SignUpBlock(signUpError, handleSignUp)
          }

        </div>
      </section>
    )
  }

export default SignPage;

UnauthorizedBlock.jsx

import React from 'react';

import styles from '../UnauthorizedBlock.module.css';
import SignIn from '../SignPage/SignForms/SignIn';

const ErrorRender = ({ error }) => {
  let errorsArr = [];
  for(let key in error) {
    errorsArr.push(error[key])
  }
  return (
    <div className={styles.error_wrap}>
      {errorsArr.map((item, index) => {
        return <p id="error" key={index}>{item}</p>
      })}
    </div>
  )
}

const UnauthorizedBlock = ({ handleSignIn, signInError }) => {
  return (
    <div className='flex-grow'>

```

```

        <div className='container'>
          <div className="row">
            <div className={styles.unauth_wrap}>
              <h5>Unfortunately, the page you are trying to
view is available only to the logged in users.</h5>
              <h5>After successful login you will be
redirected to the requested page.</h5>
              <ErrorRender error={signInError}/>
              <SignIn onSubmit={handleSignIn} />
            </div>
          </div>
        </div>
      </div>
    )
  }

```

```
export default UnauthorizedBlock;
```

```
UserForm.jsx
```

```

import React from 'react';
import { Field, reduxForm } from 'redux-form';

import styles from './UserForm.module.css';

const renderField = ({ input, label, type }) => {
  return (
    <div>
      <label>{label}</label>
      <input {...input} type={type} />
    </div>
  )
}

let UserForm = ({ handleSubmit }) => {
  return (
    <div className="flex-grow">
      <div className="container">
        <div className="row">
          <div className={styles.form_wrapper}>
            <form
              className={styles.form}
              onSubmit={handleSubmit}>
              <Field name="username" type="text"
label="Username" component={renderField}/>
              <Field name="first_name" type="text"
label="First name" component={renderField}/>
              <Field name="last_name" type="text"
label="Last name" component={renderField}/>
              <input type="submit" value="Submit"/>
            </form>
          </div>
        </div>
      </div>
    </div>
  )
}

export default reduxForm({
  form: 'userForm'
})(UserForm);

UserInfo.jsx

```

```

import React, { useState } from 'react';
import Popup from 'reactjs-popup';
import styles from './UserInfo.module.css';
import userPhotoDefault from '../../../assets/images/user-photo.jpg';
import UserForm from '../UserForm/UserForm';

const UserInfo = ({ user, updateUser }) => {

  const [isOpenModal, setModal] = useState(false);

  const handleSubmit = (data) => {
    setModal(false);
    updateUser(data);
  }

  return (
    <React.Fragment>
      <Popup
        open={isOpenModal}
        closeOnDocumentClick
        onClose={() => setModal(false)}
      >
        <React.Fragment>
          <div className={styles.close + ' close'} onClick={() => setModal(false)}>
            &times;
          </div>
          <h5>Edit User info</h5>
          <UserForm
            onSubmmit={handleSubmit}
            initialValues={user}/>
        </React.Fragment>
      </Popup>
      <div className="flex-grow">
        {user
          ? <div className="container">
              <div className="row">
                <div className={styles.user_page}>
                  <img
                    src={user.avatar ||
userPhotoDefault} alt="" className={styles.user_page_image}/>
                  <div className={styles.user_page_info}>
                    <p>User Name:
<span>{user.username}</span></p>
                    <p>User ID: <span>{user.user_id ||
user.id}</span></p>
                    <p>First name:
<span>{user.first_name}</span></p>
                    <p>Last name:
<span>{user.last_name}</span></p>
                    <button
                      onClick={() =>
setModal(true)} className={styles.profile_btn}>Edit</button>
                  </div>
                </div>
              </div>
            : null
          }
        </div>
      </React.Fragment>
    )
  }

export default UserInfo;

```

```

Auth.jsx
import React from 'react';
import { Redirect } from 'react-router-dom';
import { connect } from 'react-redux';

import SignPage from '../components/SignPage/SignPage';
import Footer from '../components/Footer/Footer';
import { userLogin, userSignUp, errorClear } from '../actions/user';

const mapStateToProps = (state) => ({
  isLogin: state.userReducer.isLogin,
  signUpDone: state.userReducer.signUpDone,
  signInError: state.userReducer.signInError,
  signUpError: state.userReducer.signUpError
});

const mapDispatchToProps = (dispatch) => ({
  userLogin: (values) => {
    dispatch(userLogin(values));
  },
  userSignUp: (values) => {
    dispatch(userSignUp(values));
  },
  errorClear: () => {
    dispatch(errorClear());
  }
});

const Auth = (props) => {
  const {
    activeTab,
    userLogin,
    userSignUp,
    isLogin,
    signUpDone,
    signInError,
    signUpError,
    errorClear
  } = props;

  const handleSignIn = (values) => {
    userLogin(values);
  };

  const handleSignUp = (values) => {
    userSignUp(values);
  };

  if (isLogin) return <Redirect to="/" />;

  return (
    <React.Fragment>
      <SignPage
        activeTab={activeTab}
        signUpDone={signUpDone}
        signInError={signInError}
        signUpError={signUpError}
        errorClear={errorClear}
        handleSignIn={handleSignIn}
        handleSignUp={handleSignUp}
      />
      <Footer />
    </React.Fragment>
  );
};

```

```

    )
  }

export default connect(mapStateToProps, mapDispatchToProps)(Auth);

ManageProducts.jsx

import React, { useEffect } from 'react';
import { connect } from 'react-redux';
import { Redirect } from 'react-router-dom';

import Header from '../Header';
import Footer from '../components/Footer/Footer'
import ManageProductPage from '../components/ManageProductsPage/ManageProductsPage';
import { addProduct, deleteProduct, fetchOwnProducts, updateProduct }
from '../actions/products';

const mapStateToProps = (state) => ({
  isLogin: state.userReducer.isLogin,
  ownProductsList: state.productsReducer.ownProductList,
  searchValue: state.productsReducer.searchValue
})

const mapDispatchToProps = (dispatch) => ({
  addProduct: (data) => {
    dispatch(addProduct(data));
  },
  deleteProduct: (id) => {
    dispatch(deleteProduct(id));
  },
  fetchOwnProducts: () => {
    dispatch(fetchOwnProducts());
  },
  updateProduct: ({ id, data }) => {
    dispatch(updateProduct({ id, data }));
  }
})

let ManageProduct = (props) => {
  const {
    isLogin,
    addProduct,
    deleteProduct,
    ownProductsList,
    fetchOwnProducts,
    searchValue,
    updateProduct
  } = props;

  useEffect(() => {
    fetchOwnProducts();
  }, [fetchOwnProducts]);

  if (!isLogin) return <Redirect to="/" />

  return (
    <React.Fragment>
      <Header />
      <ManageProductPage
        isLogin={isLogin}
        addProduct={addProduct}
        deleteProduct={deleteProduct}
        ownProductsList={ownProductsList}
      />
    </React.Fragment>
  )
}

```

```

        searchValue={searchValue}
        fetchOwnProducts={fetchOwnProducts}
        updateProduct={updateProduct}
      />
      <Footer />
    </React.Fragment>
  )
}

export default connect(mapStateToProps,
mapDispatchToProps)(ManageProduct);

```

Index.js Reducer

```

import { combineReducers } from 'redux'
import productsReducer from '../products';
import userReducer from '../user';
import { reducer as formReducer } from "redux-form";

export default combineReducers({
  userReducer,
  productsReducer,
  form: formReducer
});

```

Routes.jsx

```

import React from 'react'
import { Route, Switch, Redirect } from 'react-router'

import MainPage from "../components/MainPage/MainPage";
import SingleProduct from "../containers/SingleProduct";
import Auth from "../containers/Auth";
import UserPage from '../containers/UserPage';
import ManageProducts from '../containers/ManageProducts';
import NotFound from '../components/Errors/NotFound';
import ServerError from '../components/Errors/ServerError';
import UnauthorizedPage from '../containers/UnauthorizedPage';

import { store } from '../store/store';

const PrivateRoute = ({ render: Component, ...rest }) => {
  let isLogin = store.getState().userReducer.isLogin;
  let path = rest.location.pathname;
  return <Route {...rest} render={(props) => (
    isLogin === true
      ? <Component {...props} />
      : <Redirect to={{
        pathname: "/unauthorized-page",
        state: {
          path
        }
      }} />
  )} />
}

const routes = () => (
  <Switch>
    <Route exact path="/" component={MainPage} />
    <Route path="/sign-in" render={() => (<Auth
activeTab={"signin"} />)} />
    <Route path="/sign-up" render={() => (<Auth
activeTab={"signup"} />)} />
    <Route path="/unauthorized-page" component={UnauthorizedPage} />
  </Switch>
)

```

```

        <Route path="/not-found" component={NotFound} />
        <Route path="/server-error" component={ServerError} />
        <PrivateRoute path="/manage-products" render={() =>
(<ManageProducts />)} />
        <PrivateRoute path="/products/:productId" render={({ match }) =>
(<SingleProduct productId={Number(match.params.productId)} />)} />
        <PrivateRoute path="/userdetails/:userId" render={({ match }) =>
(<UserPage userId={match.params.userId} />)} />
    </Switch>
  )

  export default routes;

products.js Saga

import { takeEvery, put } from 'redux-saga/effects';
import {
  getProducts,
  getSingleProduct,
  getSearchProducts,
  addProduct,
  deleteProduct,
  fetchOwnProducts,
  updateProducts
} from '../api_client/products';

function* watchFetchProducts() {
  yield takeEvery("FETCH_PRODUCTS_REQUEST", fetchProducts);
}

function* fetchProducts(action) {
  try {
    const result = yield getProducts(action.payload);
    yield put({type: "FETCH_PRODUCTS_SUCCESS", payload:
result.data});
  } catch(error) {
    yield put ({ type: "FETCH_PRODUCTS_ERROR", payload: error });
  }
}

function* watchFetchSingleProduct() {
  yield takeEvery("FETCH_SINGLE_PRODUCT_REQUEST", fetchSingleProduct);
}

function* fetchSingleProduct(action) {
  try {
    const result = yield getSingleProduct(action.payload);
    yield put({type: "FETCH_SINGLE_PRODUCT_SUCCESS", payload:
result.data});
  } catch(error) {
    yield put ({ type: "FETCH_SINGLE_PRODUCT_ERROR", payload: error
});
  }
}

function* watchSearchProducts() {
  yield takeEvery("SEARCH_PRODUCTS_REQUEST", searchProducts);
}

function* searchProducts(action) {
  try {
    const result = yield getSearchProducts(action.payload);

```

```

        yield      put({type:      "FETCH_PRODUCTS_SUCCESS",      payload:
result.data}))
      } catch(error) {
        yield put ({ type: "FETCH_PRODUCTS_ERROR", payload: error });
      }
    }

function* watchAddProduct() {
  yield takeEvery("ADD_PRODUCT_REQUEST", addProductWorker);
}

function* addProductWorker(action) {
  try {
    const token = yield localStorage.getItem('jwtToken');
    const headers = { Authorization: `JWT ${token}` };
    let result = yield addProduct(action.payload, headers);
    yield put({ type: "ADD_PRODUCT_SUCCESS", payload: result.data
});
  } catch(error) {
    yield put ({ type: "ADD_PRODUCT_ERROR", payload: error });
  }
}

function* watchDeleteProduct() {
  yield takeEvery("DELETE_PRODUCT_REQUEST", deleteProductWorker);
}

function* deleteProductWorker(action) {
  try {
    const token = yield localStorage.getItem('jwtToken');
    const headers = { Authorization: `JWT ${token}` };
    yield deleteProduct(action.payload, headers);
    yield      put({      type:      "DELETE_PRODUCT_SUCCESS",      payload:
action.payload });
  } catch(error) {
    yield put ({ type: "DELETE_PRODUCT_ERROR", payload: error });
  }
}

function* watchFetchOwnProducts() {
  yield      takeEvery("FETCH_OWN_PRODUCTS_REQUEST",
fetchOwnProductsWorker);
}

function* fetchOwnProductsWorker() {
  try {
    const token = yield localStorage.getItem('jwtToken');
    const headers = { Authorization: `JWT ${token}` };
    const result = yield fetchOwnProducts(headers);
    yield      put({type:      "FETCH_OWN_PRODUCTS_SUCCESS",      payload:
result.data.data});
  } catch(error) {
    yield put ({ type: "FETCH_OWN_PRODUCTS_ERROR", payload: error
});
  }
}

function* watchUpdateProduct() {
  yield takeEvery("UPDATE_PRODUCT_REQUEST", updateProductWorker);
}

```

```

function* updateProductWorker(action) {
  try {
    const token = yield localStorage.getItem('jwtToken');
    const headers = { Authorization: `JWT ${token}` };
    let result = yield updateProducts(action.payload.id,
action.payload.data, headers);
    yield put({ type: "UPDATE_PRODUCT_SUCCESS", payload: result.data
});
  } catch(error) {
    yield put ({ type: "UPDATE_PRODUCT_ERROR", payload: error });
  }
}

```

```

export const productSagas = [
  watchFetchProducts(),
  watchFetchSingleProduct(),
  watchSearchProducts(),
  watchAddProduct(),
  watchDeleteProduct(),
  watchFetchOwnProducts(),
  watchUpdateProduct()
];

```

validate.js

```

const validate = (values) => {
  const errors = {};

  if (!values.username) {
    errors.username = 'Required'
  }

  if (!values.email) {
    errors.email = 'Required'
  } else if ( (/^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$/i.test(values.email)) ) {
    errors.email = 'Invalid email address'
  }

  if (!values.password) {
    errors.password = 'Required';
  } else if (values.password.length < 8) {
    errors.password = 'Must be 8 characters or more';
  } else if ( (/^(?=.*[a-zA-Z])(?=.*\d).*$/i.test(values.password)) ) {
    errors.password = 'Must contain letters and numbers';
  } else if ( (/^(?!.*\s).*$/i.test(values.password)) ) {
    errors.password = 'Cannot contain whitespaces';
  }

  if (!values.password1) {
    errors.password1 = 'Required';
  } else if (values.password1.length < 8) {
    errors.password1 = 'Must be 8 characters or more';
  } else if ( (/^(?=.*[a-zA-Z])(?=.*\d).*$/i.test(values.password1)) ) {
    errors.password1 = 'Must contain letters and numbers';
  } else if ( (/^(?!.*\s).*$/i.test(values.password1)) ) {
    errors.password1 = 'Cant contain whitespaces';
  }

  if (!values.password2) {
    errors.password2 = 'Required'
  } else if (values.password2 !== values.password1) {

```

```

        errors.password2 = 'Passwords must match'
    }

    return errors;
}

export default validate;

productController.js
const {check, validationResult} = require('express-validator/check');
const {matchedData, sanitize} = require('express-validator/filter');
const Product = require('../models/product');
const Category = require('../models/category');
const fs = require('fs-extra');
const mkdirp = require('mkdirp');
const resizeImg = require('resize-img');
var unqiud = require('unqiud');
require('express-async-errors');

module.exports = {
  getAllProduct: async (req, res) => {

    try {
      const [count, products] = await Promise.all([
        Product.count(),
        Product.find({})
      ]);
      res.render('product/pages', {count, products});
    } catch (err) {
      console.log("some error");
    }
  },

  productCreateGet: async (req, res) => {
    var title = "";
    var desc = "";
    var price = "";
    var categories = await Category.find({});
    res.render('product/add_page', {
      title,
      desc,
      price,
      categories
    });
  },

  productCreatePost: async (req, res) => {

    let {title, desc, price, category} = req.body;
    let slug = title.replace(/\s+/g, '-').toLowerCase();

    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      var categories = await Category.find({});
      res.render('product/add_page', {
        title, desc, price, slug, category, categories, errors:
errors.array()
      });
    } else {
      Product.findOne({'slug': slug}, (err, page) => {
        if (page) {

```



```

        // do your thang
        res.redirect(backURL);
    } else {
        try {

            var product = await Product.findOne({slug, _id: {'$ne':
id}}});

            if (product) {
                res.flash('info', 'product slug exists, choose
another');

                // do your thang
                // console.log(req);
                res.redirect('/admin/product');
            } else {
                var nowProduct = await Product.findById(id);
                nowProduct.title = title;
                nowProduct.slug = slug;
                await nowProduct.save();
                res.flash('success', 'Product update successfully');
                res.redirect('/admin/product');
            }

        } catch (err) {
            res.flash('danger', err.message);
            // do your thang
            res.redirect(backURL);
        }
    }
},

productDeletePost: async (req, res) => {
    try {
        var slug = req.params.slug;
        var product = await Product.findOne({slug});
        if (product) {
            await product.remove();
            res.json({success: true, message: "product delete
successfully"});
        }
    } catch (err) {
        res.json({success: false, message: "product delete not
success"});
    }
},

ProductFileUpload: async (req, res) => {
    try {

        var id = req.body.product_id;
        var product = await Product.findById(id);
        if (product) {
            let sampleFile = req.files.file;
            var fileName = uniqid() + ".png";
            sampleFile.mv('public/product_images/' + fileName,
function (err) {

                if (err)
                    return res.status(500).send(err);

                product.similar_image.push({url: fileName, thumb:
fileName});

                product.save();

```

```

        res.json({success: true, message: "product image
upload success"}));
    });
    }
  } catch (err) {
    res.json({success: false, message: "product delete not
success"});
  }
},

ProductFileDelete: async (req, res) => {
  try {

    var id = req.body.product_id;
    var fileName = req.body.filename;
    var info = await Product.update({_id: id}, {$pull:
{similar_image: {url: fileName}}});
    res.json({success: false, message: info});
  } catch (err) {
    res.json({success: false, message: err.message});
  }
}

};

ProductMainController.js
const {check, validationResult} = require('express-validator/check');
const {matchedData, sanitize} = require('express-validator/filter');
const Product = require('../models/product');
const Category = require('../models/category');
const fs = require('fs-extra');
const mkdirp = require('mkdirp');
const resizeImg = require('resize-img');
var unqid = require('unqid');
require('express-async-errors');

module.exports = {
  getAllProduct: async (req, res) => {

    try {
      const [count, products] = await Promise.all([
        Product.count(),
        Product.find({})

      ]);
      res.render('ProductHome/allproduct', {title:'all
products',count, products});

    } catch (err) {
      console.log("some error");
    }
  },

  categoryWiseProduct: async (req, res) => {
    try {
      var nowSlug = req.params.slug;
      const [count, products] = await Promise.all([
        Product.count(),
        Product.find({category:nowSlug})

      ]);
      res.render('ProductHome/allproduct', {title:'all
products',count, products});

```

```

        } catch (err) {
            console.log("some error");
        }
    },

    ProductDetails: async (req, res) => {
        try {
            var nowSlug = req.params.slug;
            var product = await Product.findOne({slug:nowSlug});
            res.render('ProductHome/product_single', {title:'all
products',product});

            } catch (err) {
                console.log("some error");
            }
        },
    };

Database.js
module.exports = {
    database : 'mongodb://localhost/cmscart',
}

Product.js model
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var ProductImageSchema = new Schema({
    url: {
        type: String,
        required: true
    },
    thumb: {
        type: String,
        required: true
    },
});

var ProductSchema = new Schema({
    title: {
        type: String,
        required: true
    },
    slug: {
        type: String,
        required: true
    },
    desc: {
        type: String,
        required: true
    },
    category: {
        type: String,
        required: true
    },
    price: {
        type: Number,
        required: true
    },
    image: {
        type: String
    },
});

```

```

        similar_image: {
            type: [ProductImageSchema],
            'default': []
        }
    });
    var Product = mongoose.model('Product', ProductSchema);

    // make this available to our users in our Node applications
    module.exports = Product;

products_route.js
var express = require('express');
var router = express.Router();
const {check, validationResult} = require('express-validator/check');
const {matchedData, sanitize} = require('express-validator/filter');
const ProductController = require('../controllers/productMainController');
var path = require("path");

// show all categories
router.get('/', ProductController.getAllProduct);

// insert a new product get request
router.get('/category/:slug', ProductController.categoryWiseProduct);
router.get('/:slug', ProductController.ProductDetails);

module.exports = router;

```

ДОДАТОК Д
Слайди презентації

Дослідження та програмна реалізація засобів розробки вебзастосунків електронних магазинів

Виконав: ст. гр. КНТ-110м

Сенічкін А. І.

Керівник:
доцент

Коцур М.І.

1

Рисунок Д.1 – Слайд 1

Об'єкт дослідження – процес розробки
вебзастосунків.

Предмет дослідження – методи та програмні
засоби розробки вебзастосунків.

Метою роботи є дослідження та програмна
реалізація методів розробки електронних
магазинів.

2

Рисунок Д.2 – Слайд 2

Задачи роботи:

- виконати дослідження методів та засобів розробки вебсайтів;
- дослідити існуючі технології, що дозволяють створювати вебсайти;
- розробити модель створення вебзастосунків електронних магазинів;
- створити програмне забезпечення вебсайту для електронного магазину;
- виконати тестування розробленого програмного забезпечення вебсайту електронного магазину.

3

Рисунок Д.3 – Слайд 3

Порівняння традиційної та електронної торгівлі

Традиційний магазин	Віртуальний магазин
Ходіння покупця по торговельному залу	Перегляд покупцем веб сторінок
Особистий контакт покупця з продавцем (консультація)	Консультація у продавця онлайн або по телефону
Обмежений асортимент товарів	Більш широкий асортимент
Покупка товару	Замовлення товару на доставку

4

Рисунок Д.4 – Слайд 4

Порівняння методів та засобів розробки вебзастосунків електронних магазинів

Критерій порівняння	Методи та засоби розробки вебзастосунків				
	W3DT	WSDM	RMM	OOHDM	WISDM
Зручність використання для розробки вебзастосунків електронних магазинів	+	+-	+-	+-	+-
Необхідність спільної розробки вебсайту	+	+-	-	-	+
Можливість роботи з великими даними (розробки великих вебсайтів)	+	+	+-	+-	+-
Орієнтованість на кінцевого користувача	+-	+	+-	+-	+-
Наявність етапу оцінки економічної доцільності розробки вебсайту	-	-	+	-	-
Робота з базами даних	+-	+-	+	+-	5

Рисунок Д.5 – Слайд 5

Вибір мови програмування для програмної реалізації вебзастосунку електронних магазинів

Критерій порівняння мов програмування	Мова програмування		
	JavaScript	Java	C#
Зручність використання для програмної реалізації вебзастосунку електронних магазинів	+	+-	-
Наявність засобів для розробки вебзастосунків електронних магазинів	+	+-	+-
Можливість обробки баз даних при створенні електронних магазинів	+	+	+
Засоби створення інтернет-застосунків	+	+-	+-
Зручність роботи з HTML-документами	+	+-	+-

Рисунок Д.6 – Слайд 6

Вибір фреймворку для програмної реалізації вебзастосунків електронних магазинів

Критерій порівняння фреймворків	Фреймворки		
	React	Vue.js	Angular
Зручність механізму відображення кінцевого вебсайту	+	+—	+
Направленість та класи залежностей	Одностороння передача даних	Одностороння передача даних	Взаємодія об'єктів у модулях (службах)
Зручність оновлення версій розроблюваного програмного забезпечення	+	+	+—
Частота використання розробниками вебзастосунків, %	78,1	0,8	21
Зручність використання для реалізації вебзастосунків електронних магазинів	+	+—	+

Рисунок Д.7 – Слайд 7

Діаграма компонентів вебзастосунку електронного магазину у нотації UML

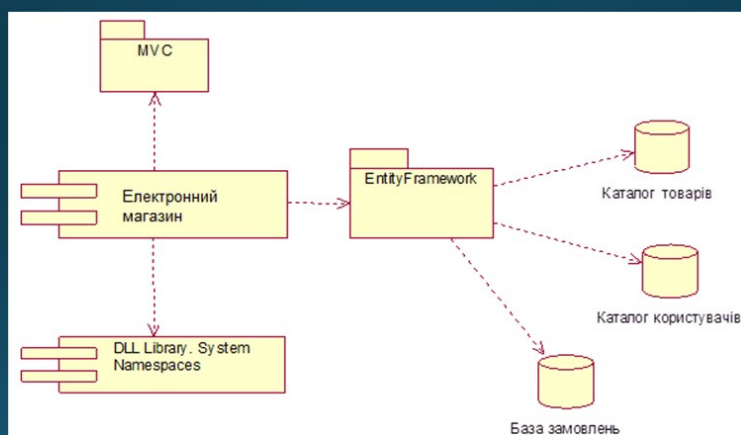


Рисунок Д.8 – Слайд 8

Діаграма стану для замовлення товару



9

Рисунок Д.9 – Слайд 9

Діаграма прецедентів «Замовлення товару»



10

Рисунок Д.10 – Слайд 10

Структура вебзастосунку електронного магазину



11

Рисунок Д.11 – Слайд 11

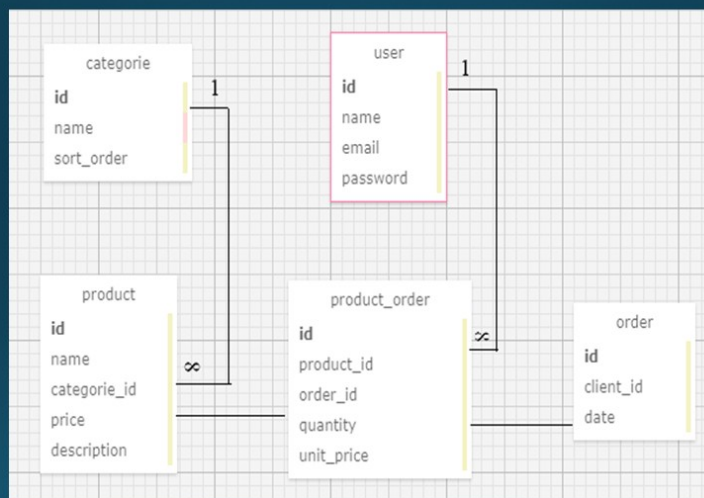
Схема взаємодії між елементами клієнтської частини вебзастосунку електронного магазину



12

Рисунок Д.12 – Слайд 12

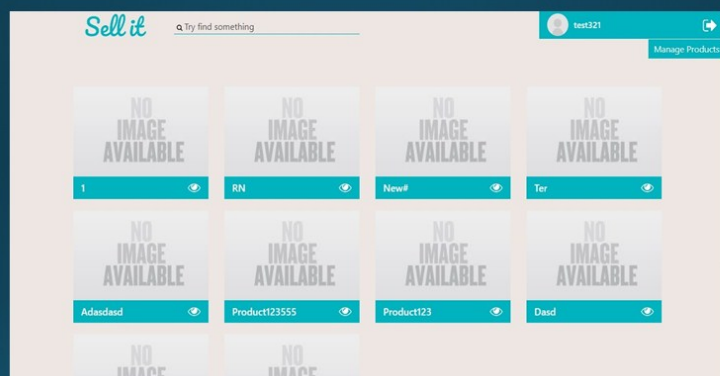
Концептуальна модель бази даних



13

Рисунок Д.13 – Слайд 13

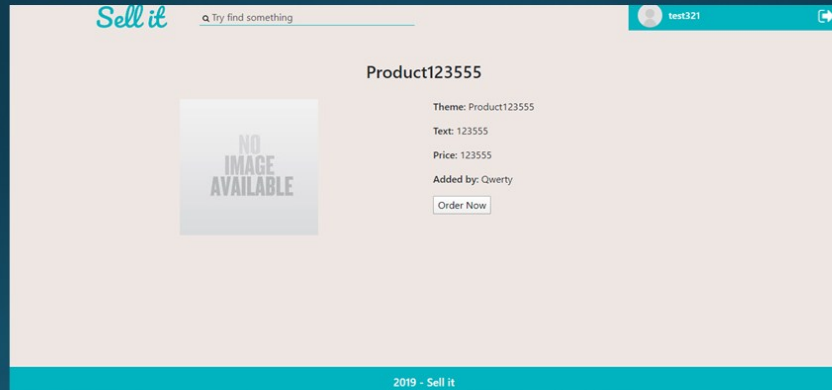
Робота з програмою



14

Рисунок Д.14 – Слайд 14

Робота з програмою. Приклад сторінки електронного магазину, що містить опис певного товару



15

Рисунок Д.15 – Слайд 15

Висновок

В ході виконання дипломної кваліфікаційної роботи магістра було проаналізовано та досліджено процес та методи розробки вебзастосунків електронних магазинів.

Наукова новизна роботи полягає у тому, що розроблено модель створення вебзастосунків електронних магазинів у вигляді діаграм, поданих у нотації UML. Запропонована модель являє собою множину абстрактних моделей вебзастосунку на основі структурних діаграм, діаграм поведінки та діаграм взаємодії, що дозволяють розробляти вебзастосунки електронних магазинів з урахуванням потреб замовників та кінцевих споживачів.

Практичне значення роботи полягає у тому, що на основі запропонованої моделі створення вебзастосунків електронних магазинів у вигляді діаграм, поданих у нотації UML, розроблено вебсайт електронного магазину.

16

Рисунок Д.16 – Слайд 16