**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

**Факультет інформаційних технологій**

**Кафедра прикладних інформаційних систем**

# ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

НА ТЕМУ

Інтернет магазин торгівлі товарами ручної роботи

Галузь знань **12 «Інформаційні технології»**

Спеціальність **122 «Комп'ютерні науки»**

Освітня програма «**Прикладне програмування»**

Освітній рівень: бакалавр

Виконав: студент 4 курсу, групи ПП-42 Лепетинський Д.А.

(прізвище та ініціали)

Керівник Булгакова О.С.

(прізвище та ініціали) к.т.н., доцент

(науковий ступінь, звання)

Унікальність тексту 95%

Випускна кваліфікаційна робота бакалавра допущена до захисту рішенням кафедри *прикладних інформаційних систем*

Протокол № 14 від 23.05.2023 р.

зав. кафедри Плескач В.Л.

Київ – 2023

КИЇВСЬКИИ НАЦІОНАЛЬНИИ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

НАЗВА ТЕМИ: Інтернет магазин торгівлі товарами ручної роботи

Освітня програма: Прикладне програмування
Спеціальність: Комп'ютерні науки

| ПІБ | Підпис |
|---|---|
| Лепетинський Данило Андрійович | |

ТЕМА РОБОТИ

| |
|---|
| Інтернет магазин торгівлі товарами ручної роботи<br>E-store for selling handmade goods |

МЕТА БАКАЛАВРСЬКОЇ РОБОТИ, ЗАВДАННЯ

| | |
|---|---|
| Мета бакалаврської роботи: підвищення ефективності торгівлі товарами ручної роботи<br><br>План роботи: | |
| | 1. Загальнотеоретичні засади електронного бізнесу<br>2. Аналіз програмно-технічних рішень інформаційної системи електронної торгівлі виробами ручної роботи<br>3. Розроблення, реалізація веб застосунку з продажу виробів ручної роботи |
| | |

ПІБ, ступінь, звання наукового керівника роботи: Булгакова О.С. к.т.н., доцент

# КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

| №з/п | Назва етапів кваліфікаційної роботи бакалавра | Термін виконання етапів кваліфікаційної роботи бакалавра | Відмітка про виконання |
|---|---|---|---|
| 1. | Вибір теми та наукового керівника кваліфікаційної роботи бакалавра | 14.10.2022 | виконано |
| 2. | Видача завдання кваліфікаційної роботи бакалавра | 24.10.2022 | виконано |
| 3. | Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра | 31.10.2022 | виконано |
| 4. | Затвердження плану кваліфікаційної роботи бакалавра | 01.11.2022 | виконано |
| 5. | Підбір та вивчення літературних та інших джерел з теми дослідження | 08.11.2022 | виконано |
| 6. | Підготовка і подання науковому керівнику першого варіанту І розділу роботи | 21.12.2022 | виконано |
| 7. | Підготовка і подання науковому керівнику | 31.01.2023 | виконано |

| | | | |
|---|---|---|---|
| | першого варіанту II розділу роботи | | |
| 8. | Підготовка і подання науковому керівнику першого варіанту III розділу роботи | 30.03.2023 | виконано |
| 9. | Подання роботи у першому варіанті | 28.04.2023 | виконано |
| 10. | Оформлення пояснювальної записки кваліфікаційної роботи бакалавра | 03.05.2023 | виконано |
| 11. | Подання кваліфікаційної роботи бакалавра на попередній захист | 23.05.2023 | виконано |
| 12. | Врахування зауважень керівника і подання роботи в остаточному варіа нті (з відповідним висновком про допуск) на кафедру | 26.05.2023 | виконано |
| 13. | Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу) | 12.06.2023 | виконано |

| | | | |
|---|---|---|---|
| 14. | Захист кваліфікаційної роботи бакалавра | 28.06.2023 | виконано |

Здобувач вищої осв _____

(підпис)

Керівник _____

(підпис)

# ВІДОМІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

| Складові частини дипломної роботи | Обсяг, арк. |
|---|---|
| Титульний аркуш | 1 |
| Календарний план дипломної роботи | 2 |
| Відомість дипломної роботи | 1 |
| Анотація | 2 |
| Анотація (іноземною мовою-англійською) | 1 |
| Зміст | 1 |
| Вступ | 2 |
| Розділ 1 | 15 |
| Розділ 2 | 35 |
| Розділ 3 | 24 |
| Висновки | 1 |
| Перелік використаних джерел | 4 |

| | | | | ДП XXXX 00.000.00 | | |
|---|---|---|---|---|---|---|
| | ПІБ | Підп. | Дата | | | |
| Розробн. | Лепетинський Д.А | | | Відомість дипломної роботи | Лист | Листів |
| Керівн. | Булгакова О.С. | | | | | |
| Н/контр. | Кравченко К.В | | | | | |
| Зав. каф. | Плескач В.Л. | | | | | |

# АНОТАЦІЯ

Кваліфікаційна робота: 92 с., 30 рис., 41 джерел.

Ця робота присвячена проектуванню та розробленню програмної системи е-комерції для торгівлі товарами ручної роботи. *Метою* цієї роботи є ефективна торгівля товарами ручної роботи за допомогою платформи е-комерції. Для досягнення цієї мети поставлені наступні *завдання*:

1. Дослідити загальнотеоретичні засади цифрової економіки та систем електронної комерції, зосереджуючись на їх застосуванні в контексті торгівлі товарами ручної роботи.

2. Здійснити аналіз програмно-технологічних рішень для побудови інформаційних систем електронної комерції з фокусом на особливості торгівлі товарами ручної роботи.

3. Спроектувати, реалізувати та впровадити програмну систему е-комерції для торгівлі товарами ручної роботи з урахуванням інженерії та із застосуванням таких технологій як Next.js, TypeScript, Firebase та інші відповідних інструментів.

*Об'єктом дослідження* є процеси, пов'язані з веденням електронної торгівлі, а саме товарами ручної роботи.

*Предметом дослідження* є програмно-технічні, організаційні засади, принципи та підходи побудови програмної системи е-комерції для торгівлі товарами ручної роботи.

Для досягнення поставлених цілей використовуються такі методи дослідження:

1. Теорія управління для аналізу теоретичних аспектів електронної комерції.

2. Емпіричний аналіз і синтез систем електронної комерції.

4. Метод порівняння для аналізу наявних ресурсів та програмних систем е-комерції.

Розроблена платформа має *практичне значення*, оскільки її можна використовувати для торгівлі виробами ручної роботи, враховуючи сучасні вимоги ринку щодо взаємодії з користувачами. Це надає значну підтримку

малим бізнесам, які, у зв'язку з пандемією та воєнним станом, змушені шукати способи реалізації своїх товарів через Інтернет.

Ключові слова: система е-комерції, електронна торгівля, товари ручної роботи, Next.js, TypeScript, Firebase.

# ABSTRACT

Thesis: 92 pages, 30 figures, 41 sources.

This work is devoted to the design and development of an e-commerce software system for trade in handmade goods. The purpose of this work is to effectively trade handmade goods using an e-commerce platform. To achieve this goal, the following tasks are set:

1. Explore the general theoretical foundations of the digital economy and e-commerce systems, focusing on their application in the context of trade in handmade goods.

2. To carry out an analysis of software and technological solutions for the construction of electronic commerce information systems with a focus on the peculiarities of trade in handmade goods.

3. Design, implement and implement an e-commerce software system for the trade of handmade goods, taking into account engineering and using technologies such as Next.js, TypeScript, Firebase and other relevant tools.

The object of the research is the processes related to the conduct of electronic trade, namely handmade goods.

The subject of the study is the software and technical, organizational principles, principles and approaches of building an e-commerce software system for trade in handmade goods.

To achieve the set goals, the following research methods are used:

1. Management theory for the analysis of theoretical aspects of e-commerce.

2. Empirical analysis and synthesis of electronic commerce systems.

4. Comparison method for analysis of available resources and e-commerce software systems.

The developed platform is of practical importance as it can be used for the trade of handicrafts, taking into account the modern requirements of the market in terms of user interaction. This provides significant support to small businesses that, due to the pandemic and martial law, are forced to find ways to sell their goods online.

Keywords: e-commerce system, e-commerce, handmade goods, Next.js, TypeScript, Firebase.

ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

API(Application Programming Interface) - програмний інтерфейс програми;

HTTP(HyperText Transfer Protocol) - протокол передачі гіпертексту;

ORM(Object-Relational Mapping) - об'єктно-реляційна проекція;

CORS(Cross-Origin Resource Sharing) - спільне використання ресурсів з різних джерел;

UI(User Interface) - користувацький інтерфейс;

СУБД - Система управління базами даних;

БД - База даних;

# ВСТУП

У сьогоднішньому конкурентному та орієнтованому на комфорт суспільстві, споживачі більше не бажають обходити вулиці з метою знайти необхідний товар, натомість, вони прагнуть здійснювати покупки, не виходячи із дому, цим самим роблячи електронну комерцію, як невід'ємну складову електронного бізнесу, гнучким та вигідним рішенням як для споживачів так і для надавачів послуг.

**Актуальність даної теми** зумовлена тим, що особливо гострою є для малого бізнесу в умовах сучасного ринку, майже монополізованого великими компаніями. Постає питання масштабування аудиторії свого підприємства, що для малого бізнесу можливе лише за рахунок інтеграції у мережу Інтернет. Більше того, останні декілька років показали всім підприємцям світу важливість присутності власного бізнесу на інтернет платформах, оскільки переважна більшість малих підприємств виявилася абсолютно беззахисною із введенням карантинних заходів. Вчасне впровадження систем електронної комерції є життєво необхідним у сучасних суспільно-політичних та ринкових умовах.

**Метою дипломної роботи** є підвищення ефективності роботи магазина виробами ручної роботи, шляхом створення повноцінного та сучасного веб-сайту.

**Завдання дослідження**:

- дослідити загально-теоретичні засади електронного бізнесу;
- здійснити аналіз програмно-технологічних рішень інформаційної системи електронної торгівлі споживчими товарами;
- спроектувати, реалізувати, впровадити Інтернет-магазин товарів ручної роботи.

**Об’єктом дослідження** дипломної роботи є процеси е-торгівлі споживчими товарами.

**Предметом дослідження** дипломної роботи є програмно-технічні, організаційні засади, принципи, підходи побудови програмної системи електронної торгівлі споживчими товарами з використанням Next.JS який базується на React Js мовою програмування Typescript, яка базується на мові JavaScript.

**Практичне значення одержаних результатів** полягає у тому, що розроблена система допоможе реалізовувати товари в інтернеті з подальшою змогою оновлення асортименту, що сприятиме бізнес.

**Структура роботи:**

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, розподілених на підрозділи,висновку та списку використаних джерел.

# РОЗДІЛ 1
# ЗАГАЛЬНОТЕОРЕТИЧНІ ЗАСАДИ ЕЛЕКТРОННОГО БІЗНЕСУ

## 1.1 Стан розвитку цифрової економіки в Україні та світі, основні її поняття та зміст

### 1.1.1 Означення та концепція електронного бізнесу

Електронний бізнес (або е-бізнес) - це форма бізнесу, в якій використовуються електронні технології, особливо Інтернет, для здійснення різних бізнес-операцій, таких як продаж товарів і послуг, маркетинг, обмін даними, комунікація між бізнес-партнерами та інших[3]. Електронний бізнес набуває все більшої популярності та впливу на сучасне суспільство.

Концепція електронного бізнесу базується на використанні електронних засобів комунікації та обробки інформації для забезпечення ефективного та швидкого обміну даними між бізнес-суб'єктами. Інтернет став основною платформою для розвитку електронного бізнесу, пропонуючи безліч можливостей для взаємодії та торгівлі через віртуальний простір.

Одним із ключових аспектів електронного бізнесу є електронна комерція (е-комерція), що охоплює купівлю і продаж товарів та послуг через Інтернет. Завдяки електронній комерції компанії можуть вести бізнес онлайн, дозволяючи клієнтам зробити покупки в будь-який час та з будь-якого місця. Електронна комерція включає в себе такі аспекти, як електронний майданчик (онлайн-магазин), електронні платіжні системи, електронні гроші, електронний обмін даними між партнерами тощо[4].

Другою важливою складовою електронного бізнесу є електронний маркетинг. Електронний маркетинг використовує Інтернет і цифрові канали комунікації для просування товарів і послуг. Це включає в себе розсилку електронних листів, рекламу в Інтернеті, соціальний медіа-маркетинг, пошукову оптимізацію та інші методи, які допомагають привернути увагу цільової аудиторії та збільшити обсяги продажів. Концепція електронного бізнесу також включає в себе поняття електронного управління бізнесом, що передбачає використання електронних систем для оптимізації та автоматизації

бізнес-процесів. Це може включати в себе електронну обробку замовлень, системи управління взаємодією з клієнтами (CRM), електронну систему управління складом, аналітику даних та інші інструменти[2], які допомагають ефективно управляти бізнес-процесами.

Нарешті, електронний бізнес включає в себе аспекти безпеки, оскільки зростання електронного бізнесу супроводжується збільшенням ризиків, пов'язаних з кіберзлочинністю та захистом конфіденційної інформації. Електронний бізнес вимагає впровадження заходів з кібербезпеки, таких як шифрування даних, захист від несанкціонованого доступу, резервне копіювання даних та інші заходи для забезпечення надійності та безпеки бізнес-операцій. Означення та концепція електронного бізнесу відображають сутність і значення цієї форми бізнесу у сучасному світі. Впровадження електронного бізнесу надає компаніям можливість збільшити свою конкурентоспроможність, залучити нових клієнтів та розширити географію свого бізнесу. Процеси електронного бізнесу продовжують розвиватися, що відкриває нові перспективи та можливості для підприємств у всіх галузях.

Глобальний характер: Електронний бізнес забезпечує можливість глобального зв'язку та торгівлі без обмежень, що відкриває нові ринки та клієнтські бази для підприємств[1]. Компанії можуть ефективно вести бізнес з клієнтами та партнерами з усього світу, незалежно від географічних меж.

Електронна аутентифікація та безпека: Захист конфіденційної інформації та персональних даних є одним з ключових аспектів електронного бізнесу. Електронна аутентифікація (використання паролів, шифрування, біометричних технологій тощо) гарантує, що тільки авторизовані користувачі мають доступ до системи.

Мобільний електронний бізнес: Зростання використання мобільних пристроїв (смартфонів, планшетів) відкриває нові можливості для електронного бізнесу. Компанії можуть створювати спеціалізовані застосунки (мобільні застосунки) для зручного доступу клієнтів до продуктів та послуг, покупок, спілкування тощо[8].

Соціальний електронний бізнес: Соціальні медіа мережі (Facebook, Instagram, Twitter тощо) надають платформи для реклами та просування товарів і послуг. Бізнес-підприємства активно використовують соціальні медіа для будівництва бренду, залучення клієнтів, здійснення продажів та спілкування зі своєю аудиторією.

Інтернет речей (Internet of Things - IoT) у електронному бізнесі: IoT відкриває нові можливості для збільшення автоматизації та ефективності електронного бізнесу[12]. За допомогою підключених пристроїв, компанії можуть здійснювати моніторинг, управління запасами, логістику та інші бізнес-процеси в режимі реального часу.

Електронний бізнес у сфері послуг: Крім традиційної продажу товарів, електронний бізнес також розширюється в галузі надання послуг. Наприклад, онлайн-банкінг, онлайн-туризм, онлайн-освіта, онлайн-консультації та багато інших послуг можуть бути надані через електронні канали.

Ці аспекти відображають постійний розвиток та зміну електронного бізнесу. Нові технології, зміни у способах комунікації та споживання інформації впливають на розвиток електронного бізнесу, відкриваючи нові можливості та виклики для компаній у всіх галузях.

### 1.1.2 Роль технологій інформаційного суспільства в розвитку електронного бізнесу

Технології інформаційного суспільства відіграють вирішальну роль у розвитку електронного бізнесу, забезпечуючи потужні інструменти та інфраструктуру для здійснення бізнес-операцій в електронному середовищі. Інновації в галузі інформаційних технологій не тільки змінюють спосіби взаємодії та комунікації[5], але й створюють нові можливості для підприємств у всіх галузях. Одним із ключових факторів, що сприяє розвитку електронного бізнесу, є широке поширення Інтернету. Інтернет став основним каналом комунікації та торгівлі, що дає змогу підприємствам залучати нових клієнтів,

здійснювати електронні продажі та розширювати географію своїх послуг. Інтернет надає доступ до глобального ринку та відкриває безліч можливостей для взаємодії та співпраці.

Іншим важливим аспектом є розвиток електронної комерції. Електронна комерція надає платформу для здійснення електронних транзакцій, продажу товарів і послуг через Інтернет. Технології електронної комерції, такі як електронні майданчики, платіжні системи, електронні гроші, дозволяють підприємствам забезпечити безпечну та зручну торгівлю в електронному середовищі. Вони сприяють автоматизації процесів продажу, ефективному керуванню запасами, а також забезпечують швидку та зручну оплату. Технології інформаційного суспільства також впливають на маркетингові стратегії підприємств. Виникнення соціальних мереж, блогів та інших платформ спілкування в Інтернеті надає можливості для ефективного просування продуктів та послуг. Маркетологи можуть використовувати соціальні мережі для взаємодії зі своєю аудиторією, проведення рекламних кампаній, аналізу споживчих попередніх уподобань та отримання поважання клієнтів[7].

Набуває значення і роль технології Big Data у розвитку електронного бізнесу. Великі обсяги даних, які генеруються в електронному середовищі, можуть бути реалізовані та використовувані для покращення стратегій продажу, виявлення нових ринкових тенденцій та управління взаємодіями з клієнтами. Аналітичні інструменти допомагають зрозуміти споживчі поведінки, створювати персоналізовані пропозиції та підвищувати ефективність маркетингових кампаній.

Крім цього, розвиток мобільних технологій має великий вплив на електронний бізнес. Смартфони та планшети стали невід'ємною частиною нашого повсякденного життя, і компанії використовують мобільні застосунки та адаптивні веб-сайти, щоб забезпечити зручність та доступність для клієнтів. Мобільний електронний бізнес надає можливість здійснювати покупки, взаємодіяти з брендами та здійснювати платежі в будь-якому місці і в будь-

який час. Наголошується також значущість розвитку кібербезпеки в електронному бізнесі[8]. Зростання електронних транзакцій та обміну конфіденційною інформацією ставить підвищені вимоги до захисту даних та забезпечення приватності. Компанії повинні приділяти належну увагу кібербезпеці, використовувати шифрування, багаторівневі системи аутентифікації та інші заходи для запобігання кібератакам та збиткам, пов'язаним з порушеннями безпеки даних.

Технології інформаційного суспільства є необхідною основою для розвитку електронного бізнесу. Вони надають інструменти та можливості для підприємств з усіх галузей, щоб забезпечити ефективну комунікацію, розширити ринки, залучити клієнтів та забезпечити конкурентну перевагу. За використанням сучасних технологій, підприємства можуть інновувати, пристосовуватися до змінних умов ринку та забезпечувати високу якість обслуговування своїм клієнтам.

### 1.1.3 Види та моделі електронного бізнесу

Електронний бізнес (е-бізнес) розгортається в різних сферах діяльності та включає в себе різноманітні види та моделі. Вони відображають різні підходи до здійснення електронних транзакцій, взаємодії з клієнтами та організації бізнес-процесів. Основні види та моделі електронного бізнесу включають[6]:

1. Електронна комерція (e-commerce): Це найбільш відомий та поширений вид електронного бізнесу. Електронна комерція охоплює продаж товарів та послуг через Інтернет. Вона може бути класифікована за різними критеріями, зокрема за типом товарів (фізичні товари, цифрові товари, послуги), за способом здійснення оплати (онлайн-платежі, платежі при отриманні), за типом партнерства (B2C - бізнес-до-споживача, B2B - бізнес-до-бізнесу, C2C - споживач-до-споживача).

2. Електронний майданчик (e-marketplace): Це онлайн-платформа, де різні продавці та покупці зустрічаються для здійснення бізнес-операцій. Електронні майданчики можуть бути загальними, як наприклад Amazon або eBay, або

спеціалізованими, що спеціалізуються на конкретній галузі або типі товарів. На електронному майданчику покупець може порівнювати ціни, здійснювати покупки та отримувати інформацію про продукти та продавців.

3. Електронна аукціонна платформа (e-auction): Цей вид електронного бізнесу передбачає здійснення продажу товарів або послуг через аукціони, де покупець, який пропонує найвищу ціну, отримує право придбати товар або послугу. Електронні аукціони можуть бути публічними (для всіх учасників) або приватними (для обмеженого кола учасників). Вони широко використовуються в багатьох галузях, таких як закупівлі, мистецтво, нерухомість тощо.

4. Електронний банкінг (e-banking): Це електронний спосіб здійснення банківських операцій, таких як перекази коштів, платежі, зарахування та зняття коштів, управління банківськими рахунками тощо. Електронний банкінг дозволяє клієнтам здійснювати операції в режимі реального часу через веб-платформи, мобільні застосунки або телефони.

5. Електронна логістика (e-logistics): Ця модель електронного бізнесу охоплює управління логістичними процесами, такими як складське господарство, управління запасами, доставка товарів тощо, за допомогою електронних систем та технологій. Електронна логістика дозволяє ефективно координувати та оптимізувати ланцюг постачання, знижувати витрати та підвищувати швидкість доставки товарів.

6. Електронний маркетинг (e-marketing): Це використання електронних каналів та інструментів для просування продуктів і послуг. Електронний маркетинг включає в себе такі методи, як електронна реклама, електронна поштова розсилка, контент-маркетинг, соціальні медіа-кампанії тощо. Він дозволяє підприємствам досягати своєї цільової аудиторії, збільшувати свій бренд і продажі, взаємодіяти з клієнтами.

7. Електронна облікова система (e-accounting): Це використання електронних систем та програм для обліку та фінансового управління підприємства. Електронна облікова система дозволяє автоматизувати процеси обліку, контролювати фінансову діяльність, вести електронну звітність та

аналізувати фінансові дані.

Ці види та моделі електронного бізнесу демонструють різноманітність та широкий спектр можливостей, які відкриваються завдяки використанню електронних технологій та Інтернету. Компанії можуть вибирати та комбінувати різні моделі електронного бізнесу в залежності від своїх потреб, цілей та галузей діяльності, що дозволяє їм забезпечити конкурентну перевагу та досягти успіху на ринку[11].

## 1.2 Юридичні аспекти електронного бізнесу

### 1.2.1 Конституційні та правові гарантії електронного бізнесу в Україні

Електронний бізнес (е-бізнес) в Україні розглядається як важливий сектор економіки, що здатний сприяти розвитку підприємництва, торгівлі та інновацій. З метою забезпечення стабільного та ефективного функціонування електронного бізнесу, в Україні прийнято ряд конституційних та правових гарантій, що регулюють цей сектор. Розглянемо детальніше деякі з них[37]:

1. Конституційні гарантії: Основними конституційними принципами, що стосуються електронного бізнесу, є принципи захисту прав власності, свободи підприємництва, свободи споживачів та свободи договору. Згідно з Конституцією України, кожен має право на власність, а держава забезпечує захист права власності. Крім того, гарантується свобода підприємництва, що означає право фізичних та юридичних осіб на провадження підприємницької діяльності. Ці принципи становлять фундаментальну основу для розвитку електронного бізнесу в Україні.

2. Законодавчі акти: Україна прийняла ряд законодавчих актів, спрямованих на регулювання електронного бізнесу та забезпечення правової охорони учасників цього сектору. Одним із ключових документів є Закон України "Про електронний цифровий підпис", який визначає правовий статус електронного підпису та його застосування в електронних документах. Законодавство також включає Закон "Про електронну комерцію", який регулює правові аспекти електронної комерції, включаючи електронні договори, захист

споживачів та електронну платежі.

3. Захист персональних даних: В Україні встановлено відповідні правові норми, що забезпечують захист персональних даних учасників електронного бізнесу. Зокрема, існує Закон "Про захист персональних даних", який встановлює правила збору, зберігання, використання та передачі персональних даних. Цей закон забезпечує конфіденційність та безпеку персональних даних, що є важливим аспектом довіри в електронному бізнесі.

4. Електронний уряд: Україна активно впроваджує концепцію електронного уряду (е-уряду). Це означає перехід від традиційного взаємодії громадян з урядовими структурами до використання електронних каналів комунікації та електронних сервісів. Електронний уряд сприяє полегшенню бізнес-процесів, зменшенню бюрократичних процедур та підвищенню ефективності взаємодії між бізнесом і державою.

5. Кібербезпека: Україна приділяє значну увагу кібербезпеці в електронному бізнесі. В країні існує законодавство, що стосується кібербезпеки, включаючи Закон "Про кібербезпеку". Цей закон визначає правові норми щодо захисту інформації в електронному середовищі, захисту від кібератак та сприяння безпечному функціонуванню електронного бізнесу.

6. Розвиток інфраструктури: Україна активно розвиває інфраструктуру електронного бізнесу, зокрема швидкісні Інтернет-з'єднання, електронні платіжні системи, онлайн-сервіси тощо. Це стимулює зростання електронного бізнесу, забезпечує зручність для користувачів та сприяє інноваціям в галузі електронної комерції та послуг.

Ці конституційні та правові гарантії сприяють створенню сприятливої інвестиційної та підприємницької кліматури в електронному бізнесі в Україні. Вони забезпечують захист прав та інтересів учасників електронного бізнесу, стимулюють інновації та зростання цього сектору економіки. Проте, для досягнення повного потенціалу електронного бізнесу в Україні, важливо продовжувати вдосконалювати законодавство та забезпечувати ефективне виконання правових норм у цій сфері.

### 1.2.2 Законодавство про електронний бізнес: основні положення в Україні

Електронний бізнес (е-бізнес) став неодмінною складовою сучасної економіки, що забезпечує зручність та ефективність взаємодії між підприємствами та споживачами. В Україні прийнято ряд законодавчих актів, що регулюють електронний бізнес та забезпечують правову охорону учасників цього сектору. Давайте розглянемо основні положення законодавства про електронний бізнес в Україні[1].

1. Закон України "Про електронний цифровий підпис" (ЕЦП). Цей закон визначає правовий статус електронного цифрового підпису та його застосування в електронних документах. Згідно з цим законом, електронний цифровий підпис має юридичну силу, якщо він використовується згідно з вимогами закону. Він забезпечує автентичність, цілісність та неденість електронних документів, що використовуються в електронному бізнесі.

2. Закон України "Про електронну комерцію". Цей закон встановлює правила та норми щодо електронної комерції в Україні. Він регулює такі аспекти, як електронні договори, захист прав споживачів, рекламу в електронній комерції, електронні платежі та інші важливі аспекти електронного бізнесу. Закон встановлює вимоги до інформації, що повинна бути надана споживачам в електронному бізнесі, а також регулює відповідальність за порушення правил електронної комерції.

3. Закон України "Про захист персональних даних". Цей закон визначає правовий статус та захист персональних даних в електронному бізнесі. Він встановлює вимоги до збору, зберігання, використання та передачі персональних даних. Закон надає споживачам право контролювати свої персональні дані, включаючи право на доступ, виправлення та видалення своїх даних. Крім того, він встановлює вимоги до захисту персональних даних та відповідальність за їхнє порушення.

4. Закон України "Про електронні платежі". Цей закон регулює питання,

пов'язані з електронними платежами в електронному бізнесі. Він встановлює вимоги до проведення електронних платежів, включаючи використання безпеки платежів, електронних грошей та платіжних систем. Закон також визначає права та обов'язки учасників електронних платежів та регулює відповідальність за порушення правил електронних платежів.

5. Закон України "Про кібербезпеку". Цей закон встановлює вимоги щодо захисту інформації в електронному бізнесі та забезпечення кібербезпеки. Він визначає правові норми щодо захисту від кібератак, забезпечення безпеки інформаційних систем та мереж, використання шифрування та інших засобів захисту. Закон також встановлює відповідальність за порушення правил кібербезпеки та визначає механізми співробітництва з іншими державами у сфері кібербезпеки.

6. Закон України "Про торговельні майданчики". Цей закон регулює діяльність електронних торговельних майданчиків, де проводяться електронні торги та здійснюється електронна комерція. Закон встановлює вимоги до реєстрації та ліцензування торговельних майданчиків, правила проведення торгів, включаючи захист прав покупців та продавців, і визначає відповідальність за порушення правил торговельних майданчиків.

Ці законодавчі акти в Україні створюють основну правову основу для розвитку електронного бізнесу та забезпечують захист прав та інтересів учасників цього сектору. Проте, важливо враховувати, що електронний бізнес є динамічною галуззю, і законодавство повинно адаптуватися до нових технологій та викликів, що виникають. Тому, постійне вдосконалення законодавства та сприяння інноваціям є важливими завданнями для розвитку електронного бізнесу в Україні.

### 1.2.3 Захист прав споживачів у електронному бізнесі

Захист прав споживачів у електронному бізнесі є важливим аспектом, оскільки він забезпечує довіру споживачів до електронних послуг та товарів, що продаються в онлайн-середовищі. Україна приділяє значну увагу

законодавчому регулюванню цієї сфери для забезпечення прав споживачів та регуляції відносин між споживачами та підприємствами в електронному бізнесі. Давайте розглянемо основні аспекти захисту прав споживачів у електронному бізнесі в Україні.

1. Закон "Про електронну комерцію". Цей закон встановлює важливі правила та норми, що стосуються електронної комерції та захисту прав споживачів. Він визначає обов'язки підприємців у сфері електронної комерції, зокрема, надання інформації про товари та послуги, правила відмови від угоди, регулювання реклами та багато іншого. Закон також встановлює вимоги до якості товарів і послуг, права споживачів на повернення товарів та відшкодування збитків у разі їх невідповідності.

2. Закон "Про захист прав споживачів". Цей закон встановлює загальні правила та принципи захисту прав споживачів у всіх сферах, включаючи електронний бізнес. Він надає споживачам право на інформацію, яка дозволяє здійснювати обґрунтований вибір товарів і послуг. Закон встановлює також право споживачів на безпеку товарів та послуг, на якість обслуговування, на захист від недобросовісної реклами та інші права, що забезпечують їх інтереси в електронному бізнесі.

3. Право на відмову від угоди. Законодавство передбачає право споживача на відмову від угоди, у тому числі в електронному бізнесі. Згідно з цим правом, споживач може відмовитися від придбання товару або послуги протягом певного періоду після його отримання. Таке право дозволяє споживачам здійснювати свободний вибір та захищає їх від некоректних дій підприємств.

4. Регулювання реклами. Україна має законодавство, яке регулює рекламу в електронному бізнесі. Реклама повинна бути правдивою та несе відповідальність за дотримання норм права на рекламу. Законодавство встановлює обмеження щодо некоректної реклами, включаючи недостовірну інформацію про товари та послуги, порушення правил конкуренції тощо. Це допомагає захищати споживачів від маніпулятивної та некоректної реклами в

електронному середовищі.

5. Механізми вирішення спорів. У разі виникнення спорів між споживачами та підприємствами в електронному бізнесі, в Україні існують механізми вирішення таких спорів. Наприклад, споживачі мають можливість звертатися до захисних органів прав споживачів або до суду для захисту своїх прав. Такі механізми дозволяють споживачам отримати компенсацію, відшкодування збитків та інші форми захисту їхніх прав у разі порушення угоди в електронному бізнесі.

Захист прав споживачів у електронному бізнесі є важливою складовою для забезпечення довіри та стабільності цього сектору. Законодавчі акти в Україні встановлюють правила та механізми захисту споживачів, а також відповідальність підприємств за порушення прав споживачів. Проте, важливо забезпечувати ефективне виконання законодавства, посилювати контроль та популяризацію прав споживачів в електронному бізнесі для створення довіри та стимулювання розвитку цього сектору економіки.

## 1.3 Безпека систем е-комерції

На цьому етапі стає очевидною необхідність шифрування даних на ресурсі. Перш за все, з метою убезпечити дані, паролі, кредитні картки та іншу інформацію користувачів ресурсу. Головною умовою широкого використання мережі Інтернет у сфері торгівлі є питання забезпечення адекватного рівня безпеки для всіх транзакцій, проведених через мережу.

З цією метою сучасними компаніями використовуються різні програмні та технічні засоби. Одним із способів, який належить до засобів вирішення проблеми безпеки в мережі є криптографія. За допомогою криптографії були розроблені такі основні методи забезпечення безпеки в Інтернеті: шифрування, електронний підпис, протоколи та сертифікати. Для повного впровадження названих рішень вони використовуються не поодинці, а в комплексі[15].

Шифрувальні алгоритми (симетричні та асиметричні) покликані переконатися, що важлива інформація надійно схована від сторонніх очей.

Найчастіше вони полягають у поєднанні ключа (у симетричних шифрувальних алгоритмах) або пари ключів (у асиметричних) з текстами таким чином, щоб вхідна інформація перетворюється у набір символів, які можна прочитати лише використавши ключ для дешифрування. Приклади симетричних алгоритмів шифрування: DES (Data Encryption Standard), RC4, RC5, RC6, IDEA (International Data Encryption Algorithm).

Проте ці алгоритми мають свої недоліки, такі як поширення ключів, для вирішення яких були розроблені асиметричні алгоритми шифрування, які полягають у використанні двох ключів (відкритого, відомого кожному та особистого, відомого лише власнику) та які пов’язані за певним правилом, таким чином, що, хоча кожний з пари ключів підходить[17] як для шифрування так і для дешифрування, дані, зашифровані одним ключем, можуть бути розшифровані тільки іншим. Прикладом найпоширенішого асиметричного алгоритму шифрування є алгоритм RSA. Цей алгоритм використовується як для шифрування даних так і для створення електронних підписів. Тоді як шифрувальні алгоритми дають змогу захистити дані від сторонніх втручань, електронний підпис покликаний переконатися, що інший учасник транзакції є саме тією особою, за яку себе видає. Електронний підпис також заснований на використанні концепції відкритого та особистого ключів. На основі них розроблений дуже важливий елемент криптосистем – хеш-функції, які використовуються для виявлення факту втручання та зміни повідомлень, тобто для електронного підпису. Прикладами захищених хеш-функцій є SHA (Secure Hash Algorithm) та MD-5 (Message Digest 5), які є гарантами того, що навіть найменші зміни в документі будуть викликати зміну його зведення (дайджест) та те, що різні документи матимуть різні цифрові підписи. Таким чином, електронний підпис є гарантом того, що дійсність та цілісність повідомлення не буде порушена ззовні.

Також, поширеними механізмами шифрування даних та захисту транзакцій в Інтернет є стандарт SET (Secure Electronic Transactions) та протокол SSL (Secure Socket Layer). Стандарт SET, що також використовує

цифрові сертифікати за стандартом X.509, призначений для переказу коштів та транзакцій між суб’єктами е-торгівлі.[19]

Протокол SSL, протокол електронного обміну даними, покликаний забезпечувати шифрування інформації. У реаліях сьогоднішньої мережі Інтернет онлайн-магазин не може обійтися без впровадження до своєї системи цього протоколу. Окрім переходу на HTTPS у контексті основного призначення SSL-сертифікату – забезпечення інформації користувачів, є й інші вагомі причини. Корпорація Google прагне якомога активніше дбати про безпеку користувачів, і тому з січня 2017 року діє наступне рішення: всі сайти, які не використовують HTTPS-протокол (відповідно, SSL-сертифікат), позначатимуться як незахищені[22].

Представники Google заявили, що відповідна відмітка буде з’являтися поруч з адресами сайтів, які не дбають про безпеку особистої інформації користувачів. Для відвідувачів сайтів ці позначки будуть сигналізувати про те, що їх паролі, номери кредитних карток тощо можуть бути викрадені. Очевидно, що купувати щось в такому інтернет-магазині користувачі навряд чи захочуть.

## 1.4 Аналіз готових програмних рішень

Аналіз програмних систем з продажу виробів ручної роботи в Україні показує наявність різноманітних платформ та сервісів, які сприяють розвитку та популяризації цього сектору. Розглянемо деякі з них:

"Handmade.ua": Це одна з найпопулярніших платформ в Україні для продажу виробів ручної роботи. Сервіс надає можливість майстриням та виробникам презентувати свої вироби, створювати власний магазин та здійснювати продажі онлайн. "Handmade.ua" забезпечує зручний інтерфейс, можливості пошуку, категоризації та фільтрації товарів, а також сприяє взаємодії між покупцями та продавцями.

"Etsy": Це світово-відома платформа для продажу виробів ручної роботи та унікальних товарів. В Україні також активно використовується як інструмент для просування виробників ручної роботи. "Etsy" пропонує різноманітні можливості для створення власного магазину, виставлення товарів на продаж, а

також має велику базу активних користувачів та інструменти маркетингу.

Соціальні мережі: В Україні виробники ручної роботи активно використовують соціальні мережі, такі як Instagram та Facebook, для продажу своїх виробів. Вони створюють власні сторінки, демонструють свої творіння, отримують замовлення та взаємодіють зі своїми клієнтами.

Ці програмні системи та платформи надають споживачам можливість знайти унікальні вироби ручної роботи, спілкуватися з виробниками, здійснювати покупки онлайн та отримувати задоволення від унікальних та авторських товарів. Вони також сприяють розвитку підприємництва в сфері ручної роботи та стимулюють креативність та мистецтво в Україні.

Електронний бізнес є важливим аспектом сучасної глобалізованої економіки і продовжує розвиватися зі значним темпом. Загально-теоретичні засади електронного бізнесу включають розуміння його основних компонентів, методів, технологій та стратегій, що використовуються для досягнення бізнес-цілей в цифровому середовищі.

1. Компоненти електронного бізнесу. Електронний бізнес включає такі ключові компоненти, як електронна комерція (продаж товарів та послуг через Інтернет), електронна взаємодія з клієнтами, електронне управління ланцюгом поставок та інші аспекти, що використовують цифрові технології для підтримки бізнес-процесів.

2. Методи та технології. Електронний бізнес використовує різні методи та технології, включаючи веб-сайти, мобільні застосунки, соціальні медіа, обробку великих даних, штучний інтелект та інші сучасні технології, щоб спростити бізнес-процеси та підвищити ефективність.

3. Стратегії. Електронний бізнес вимагає розробки ефективних стратегій, що включають визначення цільових ринків, побудову відносин з клієнтами,

оптимізацію операцій, інноваційні підходи до маркетингу та продажу, а також врахування законодавчих і безпекових аспектів.

Підсумовуючи, можна сказати, що електронний бізнес має величезний потенціал для розвитку і постійного вдосконалення бізнесу. Зростаюча доступність та проникнення Інтернету, нарощування швидкостей передачі даних та постійна еволюція технологій, все це створює нові можливості для бізнесу. Втім, це також створює виклики, такі як необхідність захисту даних, підвищення конкуренції та постійне оновлення технологій. Щоб бути успішними, компанії повинні розуміти і пристосовуватися до цих змін.

# РОЗДІЛ 2
# АНАЛІЗ ПРОГРАМНО-ТЕХНІЧНИХ РІШЕНЬ ІНФОРМАЦІЙНОЇ СИСТЕМИ ЕЛЕКТРОННОЇ ТОРГІВЛІ ВИРОБАМИ РУЧНОЇ РОБОТИ

## 2.1 Технології створення систем е-бізнесу

Для реалізації програмної системи кваліфікаційної роботи були обрані такі технології створення веб-застосунків: Next.JS, Typescipt, бібліотеки: MaterialUI, Axios, React-hook-form, Yup, база даних використовувалась Firebase.

Next.js і TypeScript - це два популярних і потужних інструменти для розробки веб-застосунків. Next.js є фреймворком реактивного програмування, побудованого на основі платформи Node.js, а TypeScript - це мова програмування зі статичним типізацією, яка розширює JavaScript.

Next.js надає зручний набір інструментів для створення універсальних або ізоморфних веб-застосунки, що працюють як на клієнтській, так і на серверній стороні. Його основна перевага - це можливість використання рендерингу на стороні сервера (SSR) і статичного генерації (Static Site Generation - SSG) для покращення швидкодії та оптимізації роботи застосунку. З Next.js ви можете створювати розширені застосунки з маршрутизацією, підтримкою стилів, інтеграцією бази даних та багато іншого.

TypeScript, з іншого боку, є мовою програмування, яка додає статичну типізацію до JavaScript, що полегшує розробку великих та складних проектів. TypeScript дозволяє визначати типи змінних, функцій та об'єктів, що сприяє покращенню контролю над програмним кодом, зменшенню помилок та полегшенню співпраці в команді. Він також надає розширені функції автодоповнення та визначення типів, що полегшують роботу з кодом.

Коли поєднується Next.js з TypeScript, отримується потужний стек технологій для розробки сучасних веб-застосунків. TypeScript надає статичну типізацію для Next.js, що полегшує відлагодження коду, забезпечує більшу стабільність та безпеку в процесі розробки. Крім того, TypeScript підтримує

розширену систему модулів, що полегшує імпорт та використання залежностей.

Next.js разом з TypeScript забезпечує зручну інтеграцію, завдяки чому можна легко створювати розширені веб-застосунки з багатим функціоналом, високою продуктивністю та надійністю. Використання цього стеку технологій дозволяє розробникам зосередитися на створенні великих та потужних проектів, забезпечуючи при цьому зручну та безпечну розробку.

Material-UI, Axios, React Hook Form і Yup - це популярні бібліотеки та інструменти, які використовуються для розробки веб-застосунків з використанням React.js. Давайте розглянемо кожну з цих технологій детальніше:

1. Material-UI є бібліотекою компонентів із графічним дизайном у стилі Material Design. Вона надає готові компоненти, такі як кнопки, форми, навігаційні елементи, таблиці та багато інших, що допомагають створювати привабливий та консистентний інтерфейс користувача. Material-UI також пропонує можливості настроювання теми, стилізацію компонентів та підтримку розширення.

2. Axios є клієнтом HTTP, який дозволяє здійснювати запити до сервера з використанням методів HTTP, таких як GET, POST, PUT, DELETE тощо. Він є простим у використанні та має широку функціональність, таку як підтримка обіцянок (promises), перехоплення помилок, встановлення заголовків запиту тощо. Axios робить взаємодію з сервером простою та зручною для розробників React-застосунків.

3. React Hook Form - це бібліотека для управління формами в React-застосунках. Вона пропонує простий та потужний спосіб збирання та валідації даних форм. React Hook Form використовує підхід на основі хуків (hooks), що полегшує роботу з формами шляхом використання функціональних компонентів та стану. Бібліотека також надає функціональності, такі як валідація даних, управління станом форми та обробка подій.

4. Yup - це бібліотека для валідації даних в JavaScript. Вона надає простий та декларативний спосіб визначення правил валідації для об'єктів, строкових значень, чисел та багатьох інших типів даних. Yup дозволяє виконувати перевірку на коректність даних, таку як обов'язкові поля, правильний формат електронної пошти, максимальну та мінімальну довжину рядка тощо.

Ці технології можуть бути комбіновані для розробки потужних та ефективних веб-застосунків. Наприклад, Material-UI може використовуватись для створення привабливого інтерфейсу користувача, Axios - для здійснення запитів до сервера та отримання даних, React Hook Form - для управління формами та їх валідації, а Yup - для визначення правил валідації даних. Використання цих технологій дозволяє покращити швидкодію розробки, забезпечити зручну та безпечну роботу з формами та створити привабливий інтерфейс для користувачів.

Firebase є платформою розробки мобільних та веб-застосунків, яка надає різноманітні сервіси для спрощення роботи зі зберіганням та обробкою даних. Один з цих сервісів - Firebase Firestore Database - є потужною розподіленою базою даних, яка дозволяє зберігати, організовувати та синхронізувати дані між різними клієнтськими пристроями.

Ось кілька ключових особливостей та можливостей Firebase Firestore Database:

1.Документ-орієнтована модель даних. Firestore використовує модель документів для зберігання даних. Документи представляють собою колекції ключ-значення, де значення можуть бути різних типів, таких як рядки, числа, булеві значення, об'єкти та масиви. Ця модель дозволяє гнучко організовувати дані та виконувати запити до них.

2. Розподіленість та синхронізація даних. Firestore автоматично реплікує дані на серверах Firebase, що забезпечує швидкий доступ та надійність. Він також підтримує режим офлайну, що дозволяє працювати з даними навіть без з'єднання з Інтернетом. Коли підключення відновлюється, дані автоматично синхронізуються між клієнтами та серверами.

3. Розширена підтримка запитів. Firestore надає потужні можливості для виконання запитів до даних. Ви можете використовувати різні умови фільтрації, сортування, обмеження кількості результатів та розширені запити, такі як "з'єднання" та "групування". Це дозволяє ефективно отримувати потрібні дані та оптимізувати роботу з базою даних.

4. Інтеграція з іншими сервісами Firebase. Firestore інтегрується з іншими сервісами Firebase, такими як Firebase Authentication (аутентифікація користувачів), Firebase Cloud Functions (функції на основі хмари) та Firebase Storage (зберігання файлів). Це дозволяє легко поєднувати різні сервіси Firebase для створення повноцінних застосунків.

5. Безпека та права доступу. Firestore надає механізми безпеки та прав доступу до даних. Ви можете використовувати правила безпеки (security rules) для обмеження доступу до певних частин даних на основі умов та ролей користувачів. Це дозволяє забезпечити конфіденційність та безпеку даних у вашому застосунку.

Firebase Firestore Database є потужним інструментом для розробки застосунків зі зберіганням та синхронізацією даних. Він дозволяє швидко створювати масштабовані та ефективні застосунки з використанням гнучкої моделі даних та багатьох додаткових можливостей, які надає Firebase платформ.

## 2.2 Структура веб-системи

### 2.2.1 Визначення інтерфейсу користувача та принципи дизайну

Інтерфейс користувача (ІК) - це точка взаємодії між користувачем і комп'ютерною системою, де користувач може вводити команди, передавати дані та отримувати відповіді. Визначення ІК полягає в розробці способу представлення інформації користувачеві та забезпеченні зручного та ефективного способу взаємодії з системою.

Принципи дизайну ІК допомагають забезпечити користувачам зрозумілий, доступний та приємний досвід взаємодії з програмними

продуктами, веб-сайтами або мобільними застосунками. Детально розглянемо деякі ключові принципи дизайну ІК:

1. Простота та зрозумілість. ІК повинен бути простим і легким у використанні. Користувачам має бути легко орієнтуватися в системі та здійснювати необхідні дії без зайвих зусиль. ІК повинен мати логічну структуру і зрозумілу навігацію.

2. Консистентність. Дизайн ІК повинен бути консистентним у всій системі або застосунку. Елементи управління, іконки, кольори та функціонал мають мати однаковий вигляд і взаємодіють однаково в різних частинах системи. Це допомагає знизити плутанину та сприяє швидкому навчанню інтерфейсу.

3. Зручність використання. ІК повинен бути зручним для використання різними категоріями користувачів, включаючи людей з обмеженими можливостями. Наприклад, використання поняття "Drag and Drop" (перетягнути та відпустити) може спростити роботу з програмними продуктами.

4. Відповідність завданням. ІК повинен відповідати конкретним завданням, які виконує користувач. Наприклад, для програми редагування фотографій можуть бути спеціалізовані панелі з інструментами, що дозволяють легко маніпулювати зображеннями.

5. Візуальна привабливість. ІК повинен мати привабливий зовнішній вигляд, використовувати зручні шрифти, кольори та графічні елементи. Естетичний дизайн сприяє покращенню загального враження користувача від продукту.

6. Застосування відповідних метафор. Використання відомих метафор допомагає користувачам легше зрозуміти, як взаємодіяти з системою. Наприклад, використання іконки кошика для видалення елементів є загальноприйнятим стандартам.

7. Забезпечення зворотного зв'язку. ІК повинен надавати користувачу відповідну інформацію про те, що відбувається в системі після виконання дії. Наприклад, підтвердження про успішне виконання команди або повідомлення

про помилку.

8. Адаптивність. ІК повинен бути адаптивним до різних пристроїв і розмірів екранів. Наприклад, веб-сайт повинен коректно відображатися як на комп'ютері, так і на мобільному телефоні.

Ці принципи дизайну ІК є лише загальними орієнтирами, і їх застосування може варіюватися залежно від контексту і вимог конкретного проекту. Професіоналізми з дизайну інтерфейсів стежать за цими принципами, щоб забезпечити зручну та задоволену взаємодію користувачів з продуктом.

### 2.2.2 Технології розробки інтерфейсу користувача

Технології розробки інтерфейсу користувача для веб-сайтів можуть включати різні інструменти та мови програмування. Опишемо основні технології, які використовуються для розробки ІК веб-сайтів детально:

1. HTML (HyperText Markup Language). HTML є основним будівельним блоком веб-сторінок. Використовуючи HTML-теги, розробники визначають структуру сторінки, розміщують текст, зображення, посилання та інші елементи.

2. CSS (Cascading Style Sheets). CSS використовується для визначення зовнішнього вигляду веб-сторінок. За допомогою CSS можна задати кольори, шрифти, розташування, розміри елементів, а також застосовувати анімацію та переходи між станами.

3. JavaScript. JavaScript є мовою програмування, яка використовується для надання веб-сайтам інтерактивності. За допомогою JavaScript можна створювати різні динамічні ефекти, валідацію форм, робити запити до сервера без перезавантаження сторінки (AJAX) та багато іншого.

4. Responsive Web Design (RWD). RWD є підходом до розробки веб-сайтів, який забезпечує їх адаптивність до різних розмірів екранів. За допомогою CSS-медіа-запитів та гнучкого розташування елементів, розробники можуть забезпечити, щоб веб-сайт коректно відображається як на комп'ютерах, так і на мобільних пристроях.

5. UI фреймворки. Існують різні фреймворки, такі як Bootstrap, Foundation, Materialize CSS, які надають готові компоненти та стилі для швидкої розробки ІК веб-сайту. Вони містять набір готових елементів, таких як кнопки, форми, навігаційні панелі, які можна використовувати для побудови інтерфейсу.

6. UX/UI дизайн. UX (User Experience) та UI (User Interface) дизайн відіграють важливу роль у розробці ІК веб-сайту. UX-дизайнери вивчають та аналізують поведінку користувачів, розробляють прототипи та забезпечують зручну навігацію та функціональність. UI-дизайнери відповідають за зовнішній вигляд, кольори, шрифти та інші візуальні аспекти ІК.

7. Front-end фреймворки. Такі фреймворки, як React, Angular і Vue.js, дозволяють розробникам будувати складні веб-застосунки з високою продуктивністю та повторним використанням коду. Вони забезпечують організацію коду, стану застосунку та динамічне оновлення ІК.

Це лише основні технології, які використовуються для розробки ІК веб-сайтів. Розробники також можуть використовувати інші інструменти, бібліотеки та фреймворки в залежності від конкретних потреб проекту.

### 2.2.3 Побудова бази даних

Firebase Firestore Database є однією з ключових технологій, що використовуються для розробки інтерфейсу користувача веб-сайтів. Ця гнучка NoSQL база даних, розроблена компанією Google, надає розробникам потужні та зручні засоби для зберігання та синхронізації даних. У цьому розділі розглянемо детальніше принципи роботи Firebase Firestore Database та його інтеграцію зі фреймворком розробки веб-застосунків Next.js.

Firebase Firestore Database заснований на моделі даних, яка складається з колекцій (collections) та документів (documents). Колекції є контейнерами, які зберігають групи документів, а документи містять поля зі значеннями. Кожен документ має унікальний ідентифікатор (ID) та може містити різні типи полів, включаючи рядки, числа, об'єкти та масиви.

Одним з ключових принципів роботи Firebase Firestore Database є

реалтайм синхронізація даних. Це означає, що коли дані змінюються на одному пристрої, вони автоматично оновлюються на всіх підключених пристроях без необхідності оновлення сторінки. Це надає користувачам можливість спостерігати за змінами в реальному часі та взаємодіяти з актуальними даними. Для отримання даних з Firestore використовується метод get(). Цей метод дозволяє отримати дані з колекцій або документів та здійснювати їх подальшу обробку. Крім того, Firestore надає можливість підписки на зміни даних за допомогою функції onSnapshot(). При підписці на зміни, коли дані в базі даних змінюються, викликається зареєстрований обробник, який дозволяє оновити інтерфейс користувача в реальному часі.

Next.js є потужним фреймворком для розробки універсальних веб-застосунків, який забезпечує високу продуктивність та швидкість розробки. Інтеграція Firebase Firestore з Next.js дозволяє розробникам зручно використовувати Firebase Firestore Database для зберігання та синхронізації даних у веб-застосунках. Основні кроки для інтеграції включають:

Налаштування Firebase проекту:

Перш ніж розпочати роботу з Firebase Firestore Database, розробнику потрібно створити проект в консолі Firebase та отримати необхідні конфігураційні дані проекту, такі як ключі доступу та параметри.

Встановлення та налаштування пакету Firebase:

У Next.js застосунку необхідно встановити пакет firebase за допомогою npm або yarn, і імпортувати необхідні модулі для налаштування з'єднання з Firebase проектом.

Налаштування з'єднання з Firebase:

Використовуючи отримані конфігураційні дані Firebase, розробник повинен викликати функцію initializeApp() для налаштування з'єднання з Firebase проектом. Це зазвичай робиться в файлі _app.js або відповідному місці застосунку, щоб забезпечити доступ до Firebase в усьому застосунку.

Використання Firebase Firestore в Next.js застосунку:

Після налаштування з'єднання з Firebase, розробник може використовувати модулі Firebase Firestore для зчитування, запису та оновлення даних у застосунку. Наприклад, для зчитування даних з колекції "users" можна використовувати метод get(), передаючи посилання на відповідну колекцію.

Реалізація реалтайм оновлень:

Одним з переваг Firebase Firestore є можливість реалтайм синхронізації даних. Розробник може підписатися на зміни даних за допомогою функції onSnapshot(), яка викликається при кожній зміні в базі даних. Це дозволяє оновлювати інтерфейс користувача в реальному часі та надати користувачам актуальні дані.

Цей розділ надає детальний огляд технології Firebase Firestore Database та його інтеграції з фреймворком Next.js. Детальна розробка та реалізація цих технологій вимагає більш глибокого дослідження та розуміння. Розробники можуть використовувати цей опис як вихідний пункт для подальшої наукової роботи та детальнішого вивчення Firebase Firestore Database та його використання в реальних проектах Next.js.

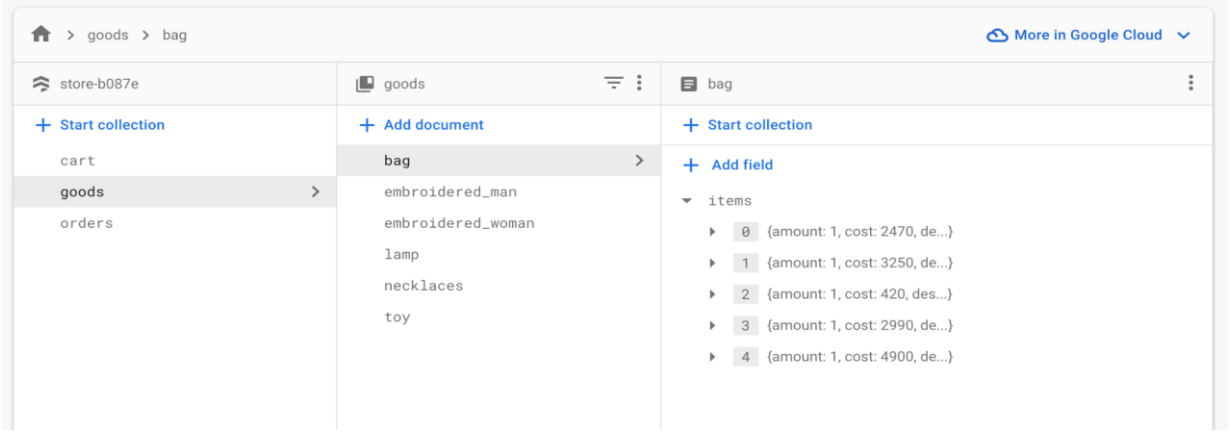


Рисунок 2.1 – Зберігання товарів

На рис 2.1. зображено колекцію товарів в ній назва типу товару а в назві масив товарів з їхніми даними.

На рис 2.2. зображено колекцію корзини в якій зберігаються унікальні номера користувачів, в документах зберігається масив з даними про товари які додані у корзину.

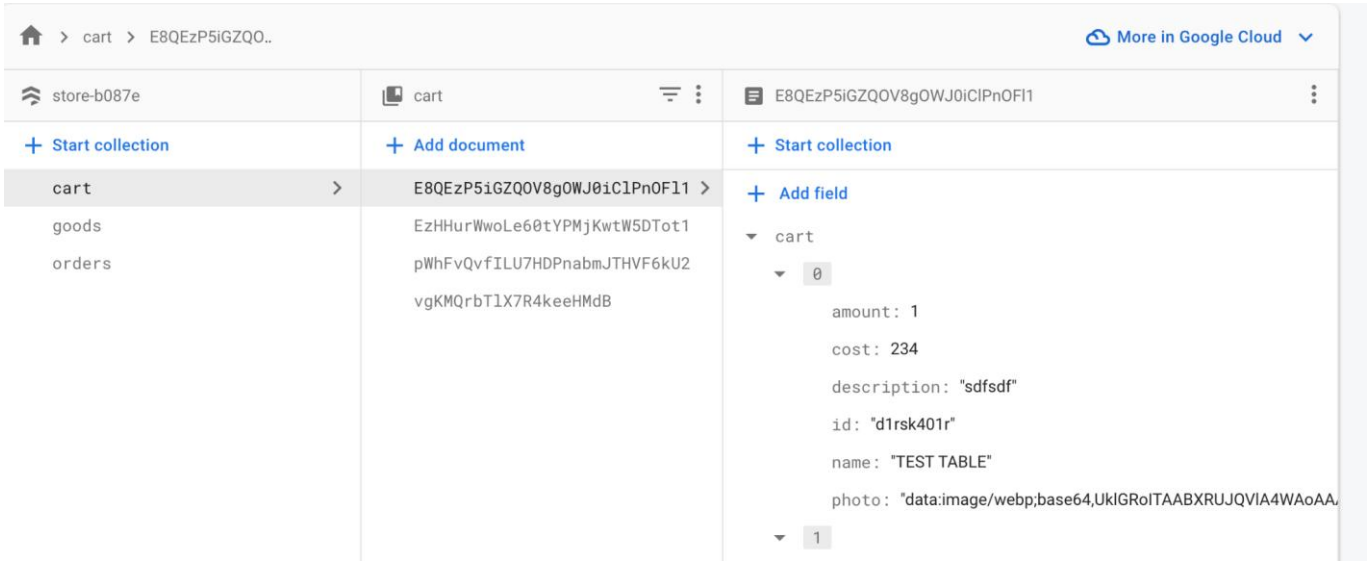


Рисунок 2.2 – Зберігання корзини

На рис 2.3. зображено колекцію замовлень в якій зберігаються унікальні унікальні номера користувачів, в документах зберігається масив з даними про замовлення в яких є масив замовленими товарами.

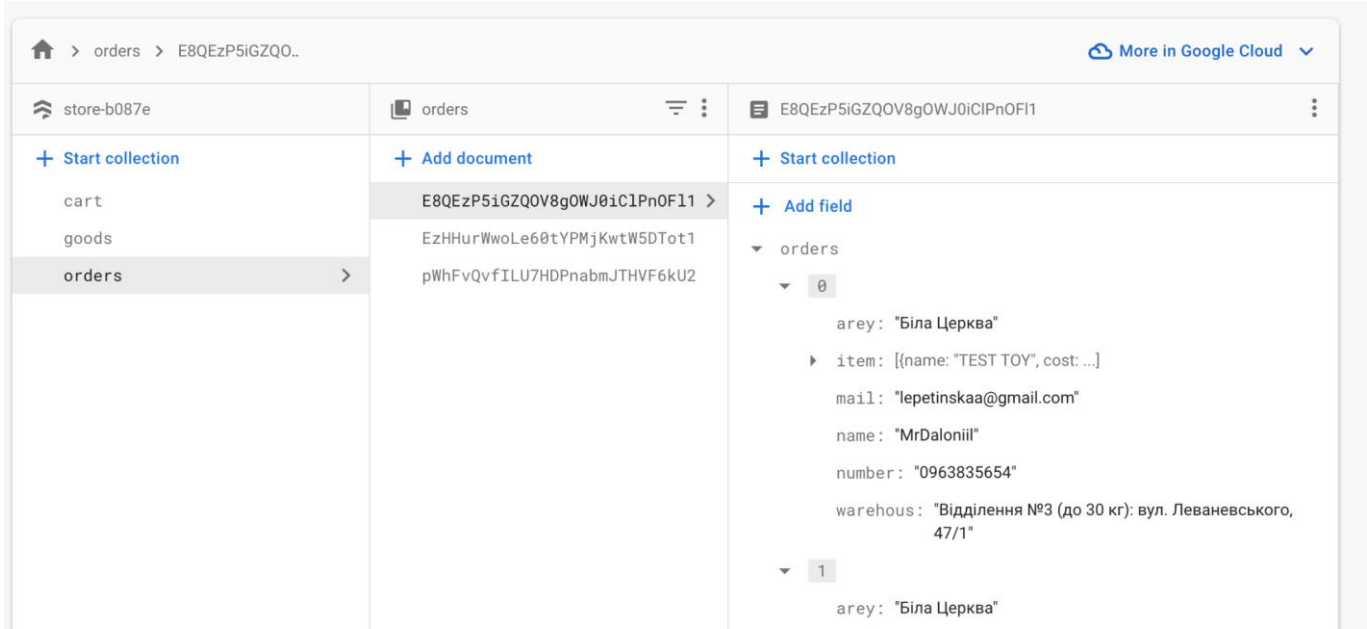


Рисунок 2.3 – Зберігання замовлень

### 2.2.4 Аутефікація

Аутентифікація є важливою складовою розробки багатьох веб-застосунках, яка дозволяє користувачам виконувати вхід у систему та забезпечує безпеку і конфіденційність даних. Firebase, платформа розробки веб-застосунків від Google, надає зручний і потужний механізм аутентифікації, що легко інтегрується з Next.js - популярним фреймворком розробки універсальних веб-застосунків. У цьому розділі ми детально розглянемо процес аутентифікації через Firebase у Next.js.

Перш ніж розпочати процес аутентифікації, необхідно налаштувати проект у консолі Firebase та отримати конфігураційні дані проекту. Ці дані включають ідентифікатор проекту, ключі доступу та інші параметри.

У Next.js застосунку потрібно встановити пакет firebase за допомогою npm або yarn, і імпортувати необхідні модулі для аутентифікації в файлі, де ви плануєте використовувати аутентифікацію.

Приклад коду:

```
import firebase from 'firebase/app';
import 'firebase/auth';
```

Після встановлення пакету Firebase та імпорту модулів, необхідно ініціалізувати Firebase у вашому Next.js застосунку, використовуючи конфігураційні дані проекту Firebase.

Приклад коду:

```
if (!firebase.apps.length) {
  firebase.initializeApp(firebaseConfig);
}
```

У Firebase існує кілька методів аутентифікації, таких як електронна пошта та пароль, Google, Facebook, Twitter та інші. Подивимося на приклад аутентифікації за допомогою електронної пошти та пароля:
Приклад коду:

```
// Аутентифікація за допомогою електронної пошти та пароля
const email = 'example@email.com';
const password = 'password';

firebase
  .auth()
  .signInWithEmailAndPassword(email, password)
  .then((userCredential) => {
    // Користувач успішно аутентифікований
    const user = userCredential.user;
    console.log('Користувач аутентифікований:', user);
  })
  .catch((error) => {
    // Помилка під час аутентифікації
    const errorCode = error.code;
    const errorMessage = error.message;
    console.error('Помилка аутентифікації:', errorCode, errorMessage);
  });
```

Цей код виконує аутентифікацію користувача за допомогою заданої електронної пошти та пароля. У випадку успішної аутентифікації, ви отримаєте об'єкт user, який містить інформацію про аутентифікованого користувача. У разі помилки під час аутентифікації, ви отримаєте відповідні дані про помилку.

Next.js дозволяє легко відстежувати стан аутентифікації користувача, використовуючи API useEffect та useState.

Приклад коду:

```
import { useEffect, useState } from 'react';

function AuthStatus() {
  const [user, setUser] = useState(null);

  useEffect(() => {
    const unsubscribe = firebase.auth().onAuthStateChanged((currentUser) => {
      setUser(currentUser);
    });

    return () => {
      unsubscribe();
    };
  }, []);

  if (user) {
    // Користувач аутентифікований
    return <div>Привіт, {user.email}!</div>;
  } else {
    // Користувач не аутентифікований
    return <div>Будь ласка, увійдіть у свій обліковий запис.</div>;
  }
}
```

У цьому прикладі ми використовуємо useEffect для підписки на зміни стану аутентифікації користувача. Коли стан аутентифікації змінюється, викликається функція `onAuthStateChanged, і ми отримуємо поточного користувача в аргументі currentUser. За допомогою useStateми оновлюємо станusersзалежно від наявності аутентифікованого користувача. У залежності від станyuser`, ми рендеримо відповідне повідомлення.

Firebase також надає можливість вийти з облікового запису користувача. Для цього використовуйте метод signOut():

Приклад коду:

```
firebase
  .auth()
  .signOut()
  .then(() => {
    // Користувач вийшов з облікового запису
    console.log('Користувач вийшов з облікового запису');
  })
  .catch((error) => {
    // Помилка під час виходу з облікового запису
    console.error('Помилка виходу з облікового запису:', error);
  });
```

Аутентифікація через Firebase у Next.js дозволяє розробникам легко інтегрувати потужну систему аутентифікації у свої веб-застосунки. За допомогою Firebase API та модулів firebase/auth, розробники можуть реалізувати різні методи аутентифікації, включаючи електронну пошту та пароль, соціальні мережі та інші. Крім того, вони можуть легко відстежувати стан аутентифікації користувача та забезпечити вихід з облікового запису. Аутентифікація через Firebase у Next.js надає потужний механізм аутентифікації, що полегшує розробку безпечних та функціональних веб-застосунків.

На основі цього я зареєструвався та обрав свій ідентифікатор як адмін і також використовуючи цей айді я створюю персональні документи для кожного

користувача.

### 2.2.5 Оптимізація зображень

Оптимізація зображення при завантаженні на сервер через веб-сайт відноситься до процесу зменшення розміру та ваги зображення, зазвичай з метою поліпшення продуктивності веб-сторінок та зниження часу завантаження. Оптимізація зображень може включати різні техніки та підходи для зменшення розміру файлу зображення, зберігаючи при цьому якість візуального відтворення.

Основні причини для оптимізації зображень під час завантаження на сервер через сайт:

1. Зниження розміру файлу: Зменшення розміру зображення дозволяє зменшити обсяг даних, що передається через мережу. Це знижує час завантаження сторінки та поліпшує продуктивність.

2. Покращення швидкості завантаження: Зменшення розміру зображення допомагає зменшити час завантаження сторінки, що впливає на користувацький досвід. Швидкість завантаження є важливим фактором для залучення і утримання відвідувачів на сайті.

3. Економія пропускної здатності: Зменшення розміру зображень також допомагає економити пропускну здатність сервера та мережі, особливо в разі великого обсягу відвідувачів або при роботі з обмеженими ресурсами.

4. Підтримка різних пристроїв та мереж: Оптимізовані зображення дозволяють краще пристосовуватись до різних пристроїв та мереж з обмеженими можливостями, забезпечуючи оптимальний досвід для користувачів.

Техніки оптимізації зображень можуть включати:

- Стиснення. Використання алгоритмів стиснення, таких як Lossless або Lossy, для зменшення розміру файлу зображення без помітного впливу на якість.

- Зменшення роздільної здатності. Зменшення фактичних розмірів зображення (ширина та висота) до необхідних розмірів, що використовуються на веб-сторінці.

- Видалення метаданих. Видалення непотрібних метаданих, які можуть бути вбудовані у файл зображення, таких як EXIF-дані, що не впливають на відображення.

- Використання відповідного формату. Вибір оптимального формату зображення, такого як JPEG, PNG або WebP, в залежності від типу зображення та вимог до якості.

- Lazy Loading. Використання методу "лінивого" завантаження зображень, коли зображення завантажуються лише при прокручуванні або показі на екрані, забезпечуючи ефективніше використання ресурсів.

- CDN. Використання Content Delivery Network (CDN) для поширення зображень, що дозволяє їх ефективну доставку до користувачів з різних регіонів.

Правильна оптимізація зображень може допомогти забезпечити швидке та ефективне завантаження веб-сторінок, знижуючи розмір файлів зображень і покращуючи користувацький досвід.

```
const imageUploaded = (): void => {
   const file = document.querySelector('input[type=file]').files[0];
   const reader = new FileReader();
   reader.onload = () => {
      if (!reader || !reader.result) return;
      if (file == null) {
         return;
      }
      const ready = new FileReader();
      ready.readAsDataURL(file);
      ready.onload = function () {
```

```
            const re = this.result as string;
            const img = new Image();
            img.src = re;
            img.onload = function () {
               const that = this;
               const w = '300';
               const h = '300';
               const quality = 1; // quality
               const canvas = document.createElement('canvas');
               const ctx = canvas.getContext('2d');
               const anw = document.createAttribute('width');
               anw.nodeValue = w;
               const anh = document.createAttribute('height');
               anh.nodeValue = h;
               canvas.setAttributeNode(anw);
               canvas.setAttributeNode(anh);
               ctx?.drawImage(that, 0, 0, w, h);
               const base64 = canvas.toDataURL('image/webp', quality);
               setImageToUpload(base64);
            };
         };
      };
      reader.readAsDataURL(file);
};
```

Цей код виконує оптимізацію розміру зображення, що завантажується користувачем, за допомогою HTML5 Canvas. Розглянемо детальніше, як працює оптимізація розміру зображення в даному коді:

1. Вибір та читання файлу зображення:

- Код розпочинається з вибору файлу зображення за допомогою `<input type="file">`.

- При виборі файлу, використовується `FileReader`, щоб прочитати його вміст.

- Читання файлу відбувається за допомогою `readAsDataURL()`, що перетворює вміст файлу в Data URL.

2. Оптимізація розміру зображення:

- Після успішного читання файлу викликається `onload` функція `FileReader`, яка обробляє результат читання.

- Створюється новий екземпляр `Image`, який буде використовуватись для завантаження зображення з Data URL.

- Після завантаження зображення викликається `onload` функція `Image`, де відбувається оптимізація розміру зображення.

- Створюється HTML-елемент `<canvas>`, на якому буде виконуватись оптимізація.

- Встановлюються розміри (`width` та `height`) для створеного елемента `<canvas>`.

- Використовуючи контекст рендерингу `getContext('2d')`, зображення масштабується до встановлених розмірів.

- Результат оптимізації отримується за допомогою `toDataURL()`, де зображення перетворюється у Data URL зі зазначеним форматом (`image/webp`) та якістю (`quality`).

- Оптимізоване зображення у форматі Data URL призначається для подальшої обробки або відображення.

3. Застосування оптимізованого зображення:

- Оптимізоване зображення (`base64`) може бути збережено або використано для подальшої обробки, наприклад, завантаження на сервер або відображення на веб-сторінці.

Цей код демонструє простий спосіб оптимізації розміру зображення на клієнтському боці перед подальшим використанням. Оптимізація розміру зображення дозволяє знизити обсяг даних, переданих через мережу, і покращити продуктивність завантаження сторінки.

Для покращення зображень в веб-інтерфейсах включає різні техніки та методи, спрямовані на поліпшення візуального відтворення зображень та підвищення їх якості. Ось кілька популярних підходів до покращення зображень в веб-інтерфейсах:

1. Використання високоякісних зображень. Важливо мати доступ до оригінальних зображень високої якості. Це дозволяє забезпечити гарне візуальне відтворення навіть при збільшенні масштабу зображення.

2. Респонсивне зображення. Використовуйте тег `<img>` з атрибутом `srcset` для вибору та завантаження відповідного розміру зображення, залежно від розміру екрану пристрою. Це допомагає забезпечити оптимальне завантаження та відображення зображень на різних пристроях.

3. Оптимізація розміру зображення. Зменшення розміру файлу зображення, зберігаючи при цьому якість, допомагає покращити швидкість завантаження сторінки. Використовуйте техніки стиснення, зменшення роздільної здатності та оптимізації формату зображення.

4. Кропінг зображень. Вирізання непотрібних або недоречних областей зображення допомагає зосередитись на важливих деталях та покращити візуальний вигляд.

5. Кешування зображень. Використання кешування зображень на стороні клієнта або сервера допомагає зменшити час завантаження та прискорити відображення зображень при повторних відвідуваннях.

6. Прогресивне завантаження. Використання прогресивного формату зображень, такого як JPEG або WebP, дозволяє зображенням поступово завантажуватись та покращує сприйняття користувача.

7. Заміна зображень на SVG або CSS. Використовуйте векторні зображення (наприклад, SVG) або CSS для створення графічних ефектів та іконок, що поліпшує роздільну здатність та масштабованість.

8. Адаптація до теми. Забезпечуйте зображенням можливість адаптуватись до теми та фону сторінки. Наприклад, використовуйте прозорість

або кольорові фільтри, щоб зображення легше співвідносилися з різними фонами.

9. Анімація та інтерактивність. Використовуйте CSS анімацію або JavaScript бібліотеки для створення динамічних та інтерактивних ефектів зображень, що привертають увагу користувачів.

Покращення зображень в веб-інтерфейсах варіюється залежно від вимог проекту та може включати комбінацію різних технік. Важливо збалансувати якість та продуктивність, забезпечуючи максимально якісне візуальне відтворення зображень при оптимальному завантаженні сторінки.

```
const canvas = document.createElement('canvas');
const ctx = canvas.getContext('2d');
const img = new Image();
img.src = parsedItem.photo;
img.onload = () => {
   canvas.width = img.width * 2;
   canvas.height = img.height * 2;
   //@ts-ignore
   ctx.imageSmoothingEnabled = true;
   //@ts-ignore
   ctx.drawImage(img, 0, 0, canvas.width, canvas.height);
   const newBase64 = canvas.toDataURL("image/jpeg", 1);
   setImageSrc(newBase64);
};
```

Цей код використовує HTML5 Canvas для поліпшення зображення, збільшуючи розмір зображення в два рази. Давайте розглянемо кожну частину коду, щоб зрозуміти, як він працює:

1. Створення елементу Canvas:

```
const canvas = document.createElement('canvas');
```

```
const ctx = canvas.getContext('2d');
```

Ці рядки коду створюють новий елемент `<canvas>` та отримують контекст малювання 2D (`ctx`) для нього. Елемент `<canvas>` використовується для малювання графіки на веб-сторінці.

2. Завантаження зображення:

```
const img = new Image();
img.src = parsedItem.photo;
img.onload = () => {
};
```

У цій частині коду створюється новий об'єкт `Image`, і йому призначається джерело зображення (`parsedItem.photo`). Після завантаження зображення (`img.onload`), виконується оптимізація зображення за допомогою HTML5 Canvas.

3. Оптимізація зображення:

```
canvas.width = img.width * 2;
canvas.height = img.height * 2;
ctx.imageSmoothingEnabled = true;
ctx.drawImage(img, 0, 0, canvas.width, canvas.height);
const newBase64 = canvas.toDataURL("image/jpeg", 1);
setImageSrc(newBase64);
```

В цій частині коду виконується оптимізація зображення за допомогою Canvas. Спочатку встановлюються розміри елементу `<canvas>` вдвічі більше розмірів початкового зображення (`img.width * 2` і `img.height * 2`).

`ctx.imageSmoothingEnabled = true;` вмикає згладжування зображення, що допомагає зробити його більш виглядом природним та плавним.

`ctx.drawImage(img, 0, 0, canvas.width, canvas.height);` малює зображення на елементі `<canvas>` за допомогою методу `drawImage()`. Зображення розміщується з координатами (0, 0) і масштабується до розмірів елементу `<canvas>`. `const newBase64 = canvas.toDataURL("image/jpeg", 1);` перетворює вміст елементу <canvas> в Data URL у форматі "image/jpeg" з якістю 1 (100%).

Отриманий Data URL містить оптимізоване зображення.

setImageSrc(newBase64); оновлює зображення на веб-сторінці, встановлюючи нове значення для imageSrc за допомогою функції setImageSrc().

Цей код бере початкове зображення, збільшує його розмір у два рази за допомогою Canvas та повертає оптимізоване зображення у форматі Data URL. Такий підхід може бути використаний для ретушування зображень та забезпечення кращої якості візуального відображення.

### 2.2.6 Локальне веб сховище

Local Storage - це механізм, який дозволяє веб-застосункам зберігати дані локально в браузері. Він надає простий спосіб зберігання ключ-значення безстроково, що дозволяє веб-застосунку зберігати стан, налаштування, кешовані дані та іншу інформацію на боці клієнта.

Давайте розглянемо детальніше, як працює Local Storage:

1. Збереження даних в Local Storage:
   - Щоб зберегти дані в Local Storage, використовується об'єкт `localStorage`.
   - Дані зберігаються у форматі ключ-значення.
   - Як приклад, розглянемо збереження значення "John" з ключем "name":

     localStorage.setItem("name", "John");

   У цьому прикладі, значення "John" зберігається в Local Storage з ключем "name".

2. Отримання даних з Local Storage:
   - Для отримання даних з Local Storage використовується метод `getItem()`.
   - Наприклад, отримаємо значення з ключем "name":

     const name = localStorage.getItem("name");

     console.log(name); // Виведе "John"

   У цьому прикладі, значення з ключем "name" ("John") отримується з Local Storage і виводиться в консоль.

3. Оновлення даних в Local Storage:
   - Для оновлення значення даних в Local Storage використовується метод `setItem()` з вказанням того ж ключа.

- Наприклад, оновимо значення ключа "name" на "Peter":

localStorage.setItem("name", "Peter");

У цьому прикладі, значення з ключем "name" оновлюється на "Peter" в Local Storage.

4. Видалення даних з Local Storage:

- Для видалення даних з Local Storage використовується метод `removeItem()`.
- Наприклад, видалимо значення з ключем "name":

localStorage.removeItem("name");

У цьому прикладі, значення з ключем "name" видаляється з Local Storage.

5. Очищення Local Storage:

- Якщо потрібно видалити всі дані з Local Storage, можна використати метод clear()
- Наприклад, очистимо Local Storage:

localStorage.clear();

У цьому прикладі, всі дані в Local Storage будуть видалені.

6. Ліміти та обмеження:

- У Local Storage є обмеження на обсяг даних, які можна зберегти.
- Зазвичай ліміт складає 5 МБ для більшості браузерів.
- Також важливо пам'ятати, що дані в Local Storage зберігаються локально на пристрої користувача і можуть бути доступні тільки на тому самому домені, на якому були збережені.

Local Storage є потужним інструментом для збереження даних на боці клієнта. Він може бути використаний для зберігання різних типів інформації, таких як налаштування, стан застосунків, вибрані варіанти та багато іншого. Проте варто бути обережними з обсягом даних, що зберігаються, оскільки Local Storage має свої ліміти.

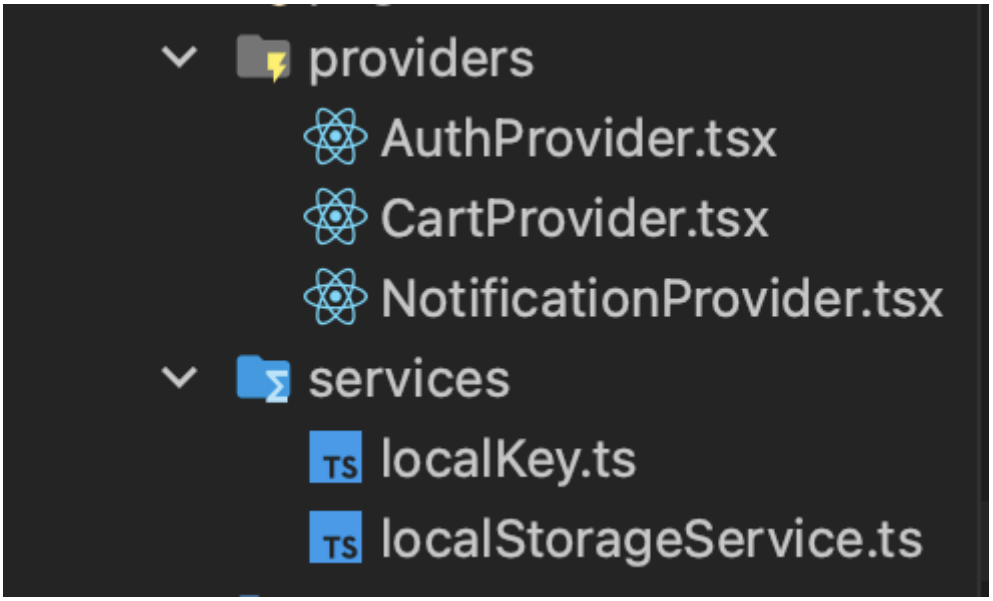


Рисунок 2.4 – Файлова система локального веб сховища

У папці services представлені два файли localStorageService та localKey

1) localStorageService:

Цей файл представляє клас `LocalStorageService`, який надає методи для зберігання та отримання даних в локальному сховищі (`localStorage` або `sessionStorage`). Розглянемо детальніше, як працюють методи цього класу:

1. `getItem<T>(key: ItemType)`. Цей метод отримує значення з локального сховища за вказаним ключем `key`. Він спочатку перевіряє наявність значення в `localStorage`, а якщо його немає, то перевіряє в `sessionStorage`. Метод повертає отримане значення, яке парситься з JSON у вказаний тип `T`.

2. `setAuth(auth: Auth, session = false)`. Цей метод встановлює дані аутентифікації `auth` в локальному сховищі. Залежно від значення параметру `session`, дані зберігаються в `localStorage` або `sessionStorage`. Попередні дані авторизації видаляються за допомогою `localStorage.clear()`. Аутентифікаційні дані перетворюються в JSON та зберігаються з використанням `setItem()`.

3. `removeAuth(session = false)`. Цей метод видаляє дані аутентифікації з локального сховища. Залежно від значення параметру `session`, дані видаляються з `localStorage` або `sessionStorage`. Дані аутентифікації замінюються на пусті дані (пустий об'єкт користувача).

4. `setCart(item: Item, session = false)`. Цей метод додає товар до кошика.

Залежно від значення параметру `session`, дані зберігаються в `localStorage` або `sessionStorage`. Він спочатку отримує поточний список товарів з `ContextKey.CART`, перевіряє, чи вже є товар з таким самим `id` в кошику. Якщо такий товар вже є, то збільшується значення `amount`. В іншому випадку, товар додається до списку. Оновлений список зберігається в локальному сховищі.

5. `setCarts(item: Item[], session = false)`. Цей метод встановлює список товарів в кошику. Залежно від значення параметру `session`, дані зберігаються в `localStorage` або `sessionStorage`. Він отримує поточний список товарів з `ContextKey.CART` і перевіряє кожен новий товар зі списку. Якщо такий товар вже є, то він не додається до списку. В іншому випадку, новий товар додається до списку. Оновлений список зберігається в локальному сховищі.

6. `removeItemCart(id: number, session = false)`. Цей метод видаляє товар з кошика за його `id`. Залежно від значення параметру `session`, дані видаляються з `localStorage` або `sessionStorage`. Він отримує поточний список товарів з `ContextKey.CART`, знаходить індекс товару з вказаним `id` та видаляє його зі списку. Оновлений список зберігається в локальному сховищі.

7. `removeCart(session = false)`. Цей метод видаляє всі товари з кошика. Залежно від значення параметру `session`, дані видаляються з `localStorage` або `sessionStorage`. Він замінює список товарів на пустий список та зберігає оновлений список в локальному сховищі.

8. `minusItemCart(id: number, session = false)`. Цей метод зменшує кількість товару в кошику за його `id`. Залежно від значення параметру `session`, дані зберігаються в `localStorage` або `sessionStorage`. Він отримує поточний список товарів з `ContextKey.CART`, знаходить індекс товару з вказаним `id` та зменшує значення `amount` на 1. Оновлений список зберігається в локальному сховищі.

9. `plusItemCart(id: number, session = false)`. Цей метод збільшує кількість товару в кошику за його `id`. Залежно від значення параметру `session`, дані зберігаються в `localStorage` або `sessionStorage`. Він отримує поточний список

товарів з `ContextKey.CART`, знаходить індекс товару з вказаним `id` та збільшує значення `amount` на 1. Оновлений список зберігається в локальному сховищі. Цей клас `LocalStorageService` надає зручний інтерфейс для роботи з локальним сховищем та зберігання різних типів даних, зокрема даних аутентифікації та товарів в кошику. Він дозволяє легко отримувати, зберігати, оновлювати та видаляти дані з локального сховища в залежності від вказаного контексту.

2) localKey:

У файлі localKey.ts зберігаються ключі для зручного доступу до локального сховища

У папці providers представлені 3 файли AuthProvider, CartProvider, NotificationProvider

1) AuthProvider:

Цей файл використовується для створення контексту автентифікації (`AuthContext`) і надає провайдер (`AuthProvider`), який управляє станом автентифікації та зберіганням даних автентифікації в локальному сховищі.

Розглянемо детальніше, як працює цей код:

1. Імпорт залежностей:

- Цей код починається з імпорту необхідних залежностей, таких як інтерфейси (`Auth`, `AuthContextType`), константи (`ContextKey`), сервіси (`LocalStorageService`) та React компоненти.

2. Створення контексту:

- За допомогою `React.createContext()` створюється контекст автентифікації - `AuthContext`.

- Початкове значення контексту передається в метод `React.createContext()`.

3. Логіка провайдера (`AuthProvider`):

- Компонент `AuthProvider` використовується для надання контексту автентифікації дочірнім компонентам.

- Приймає `children` як вхідний параметр, який представляє дочірні компоненти, обернуті в провайдер.

- Внутрішній стан `authContext` створюється за допомогою `useState()` і отримує значення з локального сховища за ключем `ContextKey.AUTH` за допомогою `appGetAuth()`.

- Метод `updateState()` використовується для оновлення стану `authContext` згідно зі значеннями з локального сховища.

- Метод `handleSetAuth()` використовується для встановлення даних автентифікації в локальному сховищі за допомогою `LocalStorageService.setAuth()` та оновлення стану `authContext`.

- Метод `handleRemoveAuth()` використовується для видалення даних автентифікації з локального сховища за допомогою `LocalStorageService.removeAuth()` та оновлення стану `authContext`.

- За допомогою `useCallback()`, методи `handleSetAuth()` та `handleRemoveAuth()` мемоізуються, що дозволяє уникнути зайвих перерендерів.

- Об'єкт `value` містить значення, які передаються в контекст: `authContext`, `setAuth` та `removeAuth`.

- За допомогою `<AuthContext.Provider>`, значення контексту передаються дочірнім компонентам (`children`).

Цей код дозволяє дочірнім компонентам отримати доступ до стану автентифікації та методів для встановлення та видалення даних автентифікації. Використовуючи `AuthContext.Provider`, значення контексту доступні усередині дерева компонентів, яке включає `AuthProvider` і його дочірні компоненти.

2) CartProvider:

Цей файл використовується для створення контексту кошика (`CartContext`) і надає провайдер (`CartProvider`), який управляє станом кошика та зберіганням даних товарів в локальному сховищі.

Давайте розглянемо детальніше, як працює цей код:

1. Імпорт залежностей:

- Код починається з імпорту необхідних залежностей, таких як інтерфейси (`CartContextType`), константи (`ContextKey`), сервіси (`LocalStorageService`) та React компоненти.

2. Створення контексту:

- За допомогою `React.createContext()` створюється контекст кошика - `CartContext`.

- Початкове значення контексту передається в метод `React.createContext()`.

3. Логіка провайдера (`CartProvider`):

- Компонент `CartProvider` використовується для надання контексту кошика дочірнім компонентам.

- Приймає `children` як вхідний параметр, який представляє дочірні компоненти, обернуті в провайдер.

- Внутрішній стан `cartContext` створюється за допомогою `useState()` і отримує значення з локального сховища за ключем `ContextKey.CART` за допомогою `appGetAuth()`.

- Метод `updateState()` використовується для оновлення стану `cartContext` згідно зі значеннями з локального сховища.

- Методи `handleSetCart()`, `handleSetsCart()`, `handleRemoveCart()`, `handleMinusCart()`, `handlePlusCart()`, `handleRemoveCarts()` використовуються для збереження, оновлення та видалення даних кошика в локальному сховищі за допомогою `LocalStorageService`.

- За допомогою `useCallback()`, методи мемоізуються, що дозволяє уникнути зайвих перерендерів.

- Об'єкт `value` містить значення, які передаються в контекст: `cartContext`, `addItemCart`, `setItemCart`, `removeItemCart`, `minusItemCart`, `plusItemCart`, `removeCart`.

- За допомогою `<CartContext.Provider>`, значення контексту передаються дочірнім компонентам (`children`).

Цей код дозволяє дочірнім компонентам отримати доступ до стану кошика та методів для додавання, видалення, оновлення та очищення товарів у

кошику. Використовуючи `CartContext.Provider`, значення контексту доступні усередині дерева компонентів, яке включає `CartProvider` і його дочірні компоненти.

3) NotificationProvider:

Цей файл використовується для створення контексту повідомлень (`NotificationContext`) і надає провайдер (`NotificationProvider`), який управляє станом повідомлень і надає методи для додавання та видалення повідомлень.

Давайте розглянемо детальніше, як працює цей код:

1. Створення контексту:

- За допомогою `React.createContext()` створюється контекст повідомлень - `NotificationContext`.
- Початкове значення контексту передається в метод `React.createContext()`.

2. Логіка провайдера (`NotificationProvider`):

- Компонент `NotificationProvider` використовується для надання контексту повідомлень дочірнім компонентам.
- Приймає `children` як вхідний параметр, який представляє дочірні компоненти, обернуті в провайдер.
- Внутрішні стани `notification` і `statusNotification` створюються за допомогою `useState()` і мають початкове значення `null`.
- Метод `removeNotification()` встановлює значення `notification` на `null`, що видаляє поточне повідомлення.
- Метод `addNotification()` приймає параметри `message` (повідомлення) і `status` (статус) і встановлює значення `notification` на передане повідомлення і значення `statusNotification` на переданий статус.
- За допомогою `useCallback()`, методи `addNotification()` і `removeNotification()` мемоізуються, що дозволяє уникнути зайвих перерендерів.
- Об'єкт `contextValue` містить значення, які передаються в контекст: `notification`, `statusNotification`, `addNotification` і `removeNotification`.
- За допомогою `<NotificationContext.Provider>`, значення контексту

передаються дочірнім компонентам (`children`).

Вище згаданий код дозволяє дочірнім компонентам отримати доступ до стану повідомлень та методів для додавання та видалення повідомлень. Використовуючи `NotificationContext.Provider`, значення контексту доступні усередині дерева компонентів, яке включає `NotificationProvider` і його дочірні компоненти.

### 2.2.7 Використання сторонніх API

При розробці продукту я використав API нової пошти

API (Application Programming Interface) - це набір правил та протоколів, які визначають, як різні програми та компоненти програмного забезпечення можуть взаємодіяти між собою. API визначає, яким чином програми можуть запитувати та обмінюватися даними, виконувати функції та отримувати результати.

Основні аспекти API:

1. Визначення інтерфейсу. API визначає доступні функції, методи та параметри, які можуть бути використані для взаємодії з певним компонентом або сервісом. Інтерфейс API описує, яким чином розробники можуть взаємодіяти з ним, які параметри потрібно передати та які результати можна отримати.

2. Протоколи комунікації. API визначає протоколи та формати обміну даними між програмами. Це може бути HTTP, REST, SOAP, WebSocket, GraphQL та інші протоколи. Кожен протокол має свої правила та стандарти для взаємодії між клієнтом та сервером.

3. Типи запитів та відповідей. API визначає, які типи запитів можуть бути виконані до сервісу або компонента, і які типи відповідей можуть бути отримані. Це можуть бути GET, POST, PUT, DELETE запити для отримання, створення, оновлення або видалення даних.

4. Формати даних. API описує формати даних, які використовуються для передачі та обміну інформацією між програмами. Це можуть бути JSON, XML,

CSV, Protobuf та інші формати, які використовуються для структурування та представлення даних.

5. Автентифікація та авторизація. API може включати механізми аутентифікації та авторизації, щоб забезпечити безпеку та обмежити доступ до певних функцій або даних. Це можуть бути токени доступу, ключі API, OAuth, JWT та інші методи аутентифікації та авторизації.

6. Документація та SDK: Для використання API розробники часто мають доступ до документації, яка описує функціональність, параметри та приклади використання API. Крім цього, можуть бути надані SDK (Software Development Kit) для певних мов програмування, які спрощують взаємодію з API, надаючи готові класи та методи для виконання запитів та обробки відповідей.

API можуть бути побудовані для різних цілей, включаючи взаємодію з веб-сервісами, створення розширень програмного забезпечення, інтеграцію з платформами соціальних медіа, розробку мобільних застосунків та багато іншого. API дозволяють розробникам використовувати функціональність інших програм або сервісів, не реалізуючи її заново, що полегшує розробку програмного забезпечення та сприяє інтеграції між різними системами.

API Нової пошти (New Post API) - це набір веб-сервісів, які надають доступ до функціональності та інформації, пов'язаних з послугами доставки та логістики, наданими компанією "Нова пошта" в Україні. Це дозволяє розробникам використовувати ці сервіси для створення власних застосунків, інтеграції з електронними магазинами, розробки функціоналу веб-сайтів та інших застосувань, які пов'язані з доставкою товарів.

Основні принципи роботи з API Нової пошти:

1. Реєстрація та отримання доступу:

- Для використання API Нової пошти розробнику необхідно зареєструватися в системі та отримати ключ доступу (API ключ). Цей ключ буде використовуватися для аутентифікації та авторизації запитів до API.

2. Документація та опис доступних сервісів:

- API Нової пошти має докладну документацію, яка описує доступні сервіси,

параметри запитів, формати даних, можливі відповіді та іншу важливу інформацію для використання API.

3. Аутентифікація та авторизація:

- Для забезпечення безпеки та обмеження доступу до функціональності API Нової пошти використовується аутентифікація та авторизація. Зазвичай це здійснюється шляхом передачі API ключа у заголовку запиту або використання токенів доступу.

4. Використання HTTP методів:

- API Нової пошти використовує стандартні HTTP методи, такі як GET, POST, PUT, DELETE, для взаємодії з ресурсами та виконання різних операцій. Запити виконуються до відповідних URL-адрес, які вказують на конкретний ресурс або сервіс.

5. Обробка запитів та відповідей:

- При взаємодії з API Нової пошти розробник формує HTTP запити з необхідними параметрами, які передаються до сервера API. Відповіді на запити повертаються у форматі JSON або іншому вказаному форматі, який містить необхідну інформацію про результати запиту.

6. Функціональність API:

- API Нової пошти надає різноманітну функціональність, таку як створення замовлень, відстеження вантажу, отримання інформації про вартість доставки, розрахунок терміну доставки, створення накладних та багато іншого. Розробник може вибрати необхідний сервіс або комбінацію сервісів для виконання своїх завдань.

7. Обробка помилок:

- API Нової пошти може повертати різні типи помилок, які розробник повинен коректно обробляти. Це може бути пов'язано з некоректними параметрами запиту, недоступністю сервісу, обмеженнями швидкості або іншими факторами.

В цілому, API Нової пошти дозволяє розробникам використовувати різноманітну функціональність, пов'язану з доставкою та логістикою, та

інтегрувати її в свої застосунки та сервіси. Це дозволяє автоматизувати процеси доставки, відстеження посилок, розрахунок вартості доставки та інші операції, що пов'язані з послугами Нової пошти.

Для роботи коду я використовую бібліотеку Axios для виконання HTTP запитів та надає функції для взаємодії з API Нової пошти (Nova Poshta API).

Основною функцією є `useNovaPoshta` яка експортує дві функції: `getCities` та `getWarehouses`. При виклику цих функцій будуть виконані відповідні запити до API Нової пошти.

1. Функція `getCities(city: string)`:

- Виконує POST запит до URL `'https://api.novaposhta.ua/v2.0/json/'` з наступними параметрами:

- `apiKey`: Ключ доступу API Нової пошти.

- `modelName`: Назва моделі, в даному випадку `'AddressGeneral'`.

- `calledMethod`: Назва методу, в даному випадку `'searchSettlements'`.

- `methodProperties`: Властивості методу, у цьому випадку передається `CityName`, який визначає назву міста.

- Повертає результат запиту у форматі Promise. У разі успішного виконання, дані відповіді будуть доступні у полі `response.data`. У разі помилки, виводиться повідомлення про помилку у консоль.

2. Функція `getWarehouses(city: string)`:

- Виконує POST запит до URL `'https://api.novaposhta.ua/v2.0/json/'` з наступними параметрами:

- `apiKey`: Ключ доступу API Нової пошти.

- `modelName`: Назва моделі, в даному випадку `'AddressGeneral'`.

- `calledMethod`: Назва методу, в даному випадку `'getWarehouses'`.

- `methodProperties`: Властивості методу, у цьому випадку передаються `CityName` (назва міста) та `TypeOfWarehouseRef` (референс типу відділення).

- Повертає результат запиту у форматі Promise. У разі успішного виконання, дані відповіді будуть доступні у полі `response.data`. У разі помилки,

виводиться повідомлення про помилку у консоль.

Загальною метою цього коду є спрощення використання API Нової пошти, надаючи функції, які виконують потрібні запити та повертають результати. Розробник може імпортувати цей код та використовувати функції `getCities` та `getWarehouses` для отримання інформації про міста та відділення Нової пошти на основі переданого міста.

# РОЗДІЛ 3
# РОЗРОБЛЕННЯ, РЕАЛІЗАЦІЯ ВЕБ ЗАСТОСУНКУ З ПРОДАЖУ ВИРОБІВ РУЧНОЇ РОБОТИ

## 3.1 Реалізація веб-сайту

Весь сайт полностью адаптований до всіх можливих розмірів екрану

Адаптивний дизайн (або просто адаптивність) веб-сайту означає його здатність адаптуватися та відображатися на різних пристроях та розмірах екрану, забезпечуючи оптимальний користувацький досвід. Адаптивний дизайн дозволяє веб-сторінкам гнучко змінювати свою структуру, розмір та вигляд, щоб належним чином відображатися на різних пристроях, таких як комп'ютери, планшети, смартфони та інші мобільні пристрої.

Принципи роботи адаптивного дизайну:

1. Медіа-запити (Media Queries) - це CSS-правила, які дозволяють застосовувати стилі до веб-сторінок в залежності від характеристик пристрою, на якому вони відображаються, наприклад, ширини екрану. Використовуючи медіа-запити, можна змінювати розміри, розташування, шрифти та інші атрибути для різних пристроїв.

2. Гнучкі сітки (Flexible Grids) використовуються для розміщення та організації контенту на сторінці. Замість фіксованих ширин блоків, використовуються відносні одиниці вимірювання, такі як відсотки або flexbox, щоб контент могло адаптуватися до різних екранів. Це дозволяє елементам веб-сторінки займати необхідну кількість простору, залежно від доступного розміру екрану.

3. Резиновий дизайн (Fluid Design) означає використання відносних одиниць вимірювання та властивостей, що дозволяють елементам веб-сторінки пропорційно розтягуватися або стискатися залежно від розміру екрану. Це забезпечує плавний перехід із великих екранів на менші без втрати зручності використання.

4. Зображення з адаптивним розміром (Responsive Images). Використання адаптивних зображень дозволяє завантажувати зображення з оптимальним

розміром для конкретного пристрою. За допомогою атрибутів HTML та CSS, таких як `srcset`, `sizes` та `max-width`, можна вказати різні варіанти зображень для різних розмірів екрану, щоб забезпечити ефективне завантаження та відображення зображень на різних пристроях.

5. Тач-дружність (Touch-Friendly). Адаптивний дизайн також враховує взаємодію з дотиковими екранами. Він забезпечує достатній розмір кнопок, елементів керування та інтерактивних елементів, щоб їх було зручно торкатися та взаємодіяти з ними на пристроях з дотиковим екраном.

Реалізація адаптивного дизайну передбачає використання CSS-медіа-запитів для зміни стилів залежно від розміру екрану та пристрою, а також гнучких сіток, резинових властивостей та адаптивних зображень. Це можна зробити вручну, застосовуючи правила CSS, або використовуючи CSS-фреймворки, такі як Bootstrap або Foundation, які вже мають вбудовану підтримку адаптивного дизайну. Додатково, розробники також можуть використовувати JavaScript для динамічного маніпулювання елементами сторінки, залежно від розміру екрану.

Material-UI - це популярна бібліотека компонентів для React, яка пропонує готові стилізовані елементи і компоненти інтерфейсу користувача. Вона дозволяє розробникам швидко створювати сучасний та привабливий дизайн веб-застосунків.

Основні переваги використання Material-UI:

1. Готові компоненти. Material-UI надає широкий набір готових компонентів, таких як кнопки, форми, таблиці, модальні вікна, навігаційні панелі та інші. Це значно спрощує процес розробки, оскільки ви можете використовувати ці компоненти безпосередньо, замість написання стилів та коду з нуля.

2. Персоналізація. Material-UI надає можливості для налаштування та персоналізації компонентів. Ви можете змінювати кольори, типографіку, розміри та інші властивості компонентів, щоб вони відповідали вашому дизайну та стилю.

3. Матеріальний дизайн. Material-UI базується на концепції Матеріального дизайну, який рекомендується Google. Це означає, що ваші застосунки матимуть сучасний та стильний вигляд, використовуючи зрозумілі та інтуїтивно зрозумілі елементи і принципи дизайну.

4. Респонсивний дизайн. Material-UI має вбудовану підтримку адаптивного дизайну, що дозволяє вам легко адаптувати ваші компоненти до різних розмірів екрану. Ви можете використовувати медіа-запити або використовувати вбудовані класи для приховування, показу або зміни компонентів залежно від розміру екрану.

5. Розширення та спільнота. Material-UI має велику активну спільноту, яка надає багато розширень, додаткових компонентів, тем та інших ресурсів. Ви можете знайти готові рішення для багатьох стандартних завдань, а також отримати підтримку та поради від інших розробників.

Використання Material-UI для адаптивного дизайну полягає використанні вбудованих класів та компонентів, які забезпечують адаптацію до різних розмірів екрану. Деякі основні підходи до адаптивного дизайну з використанням Material-UI включають:

1. Використання Grid компонента дозволяє легко створювати розташування елементів у сітці. Ви можете використовувати різні класи Grid для вирівнювання елементів у залежності від розміру екрану. Наприклад, ви можете встановити `xs={12}` для елемента, щоб він займав всю ширину екрану на найменших розмірах.

2. Використання Hidden компонента дозволяє приховувати або показувати елементи залежно від розміру екрану. Ви можете використовувати Hidden компонент з різними значеннями `only`, `xsDown`, `smUp` тощо, щоб визначити, коли елемент має бути видимим або прихованим на різних розмірах екрану.

3. Використання Media Query Hook надає вбудований хук `useMediaQuery`, який дозволяє вам застосовувати стилі або логіку залежно від розміру екрану. Ви можете використовувати цей хук, щоб управляти станом

або класами компонентів залежно від адаптивних потреб вашого дизайну.

Ці підходи до адаптивного дизайну з використанням Material-UI дозволяють легко створювати зручний та гнучкий інтерфейс користувача, який працює добре на різних пристроях та розмірах екрану. Вони спрощують розробку адаптивного дизайну та забезпечують консистентність та ефективність розробки.

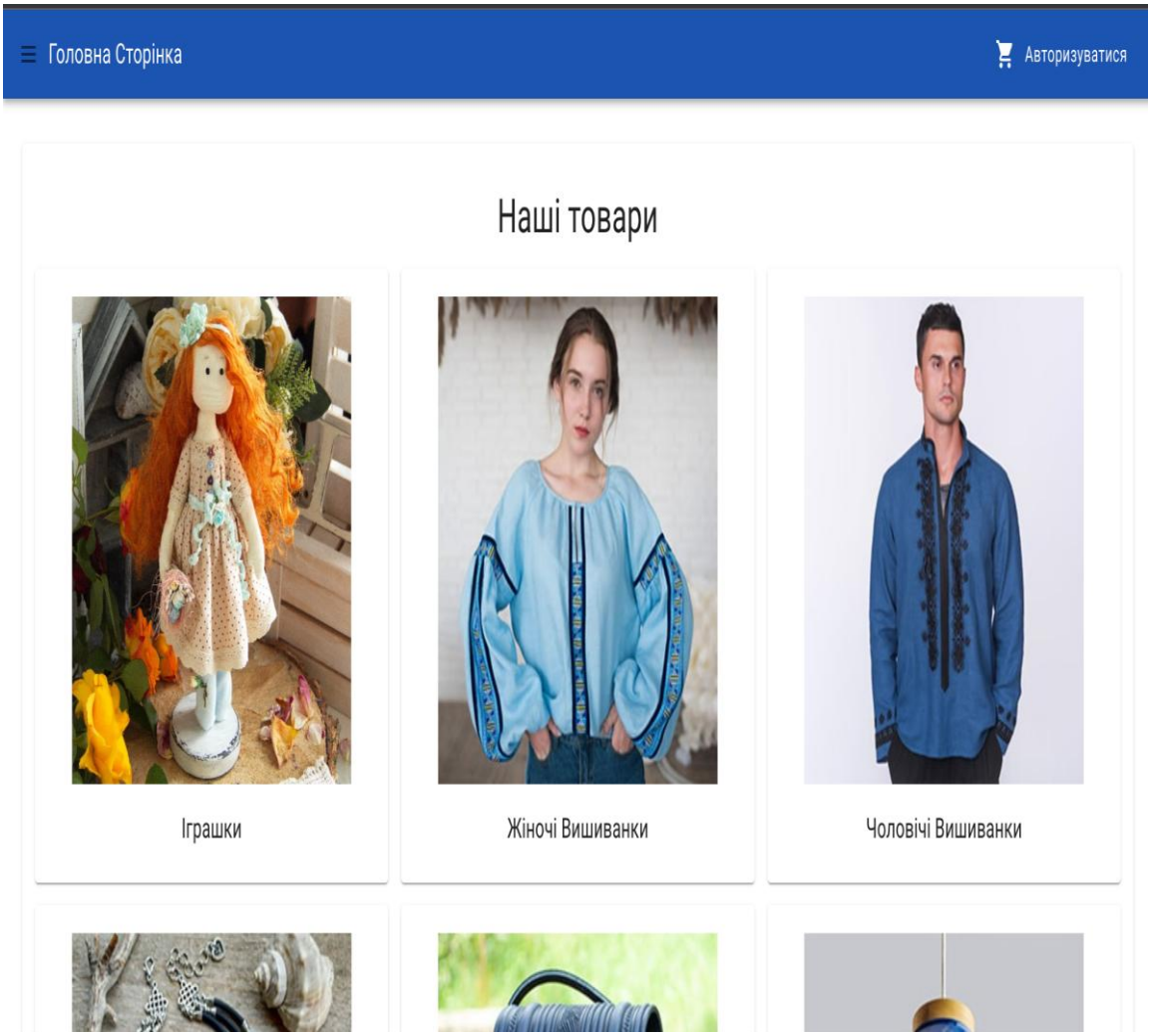


Рисунок 3.1 – Головна сторінка


Рисунок 3.2 – Головна сторінка (адаптив)

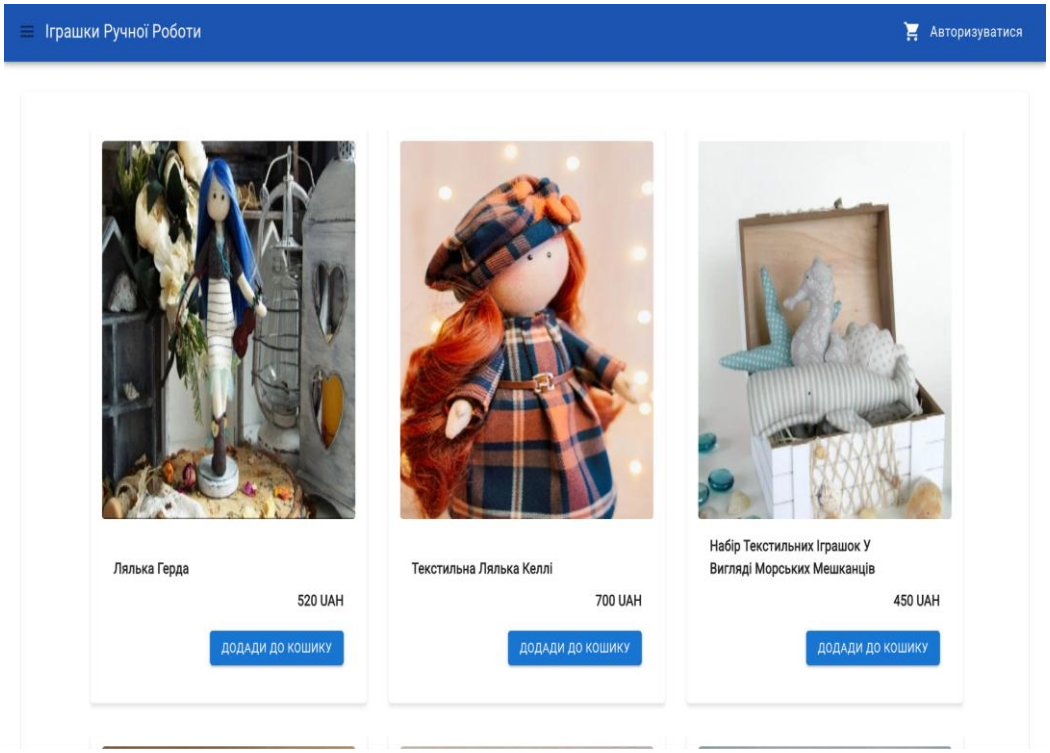


Рисунок 3.3 – Одна з сторінок типів товару (адаптив)


Рисунок 3.4 – Одна з сторінок типів товару (адаптив)

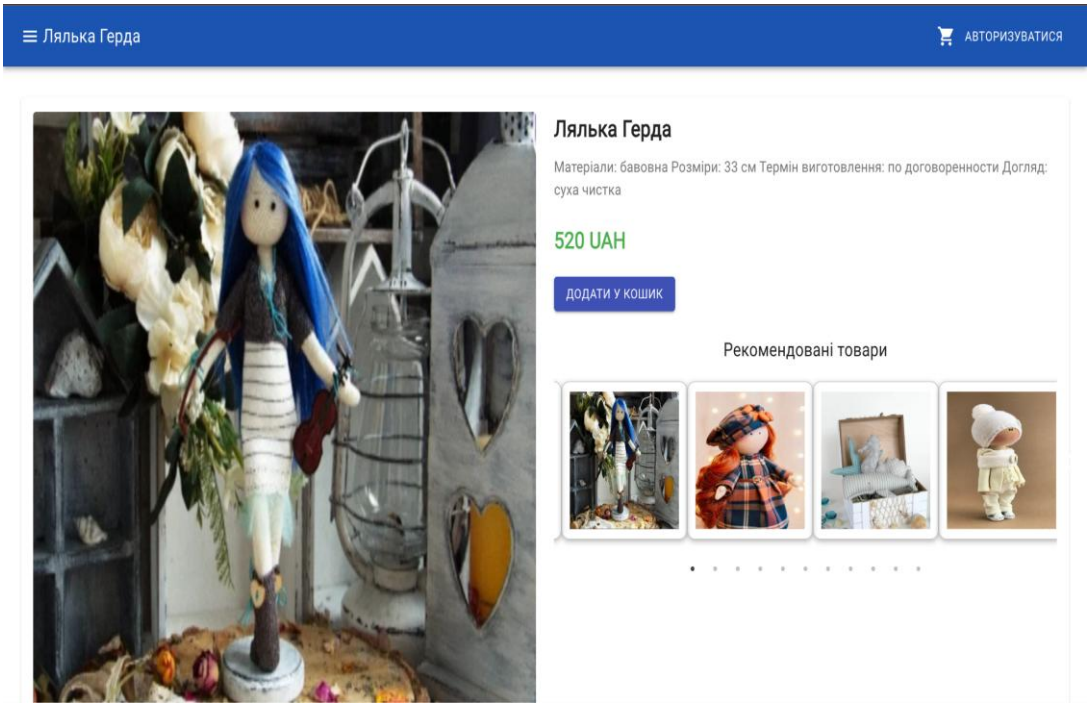


Рисунок 3.5 – Сторінка товару

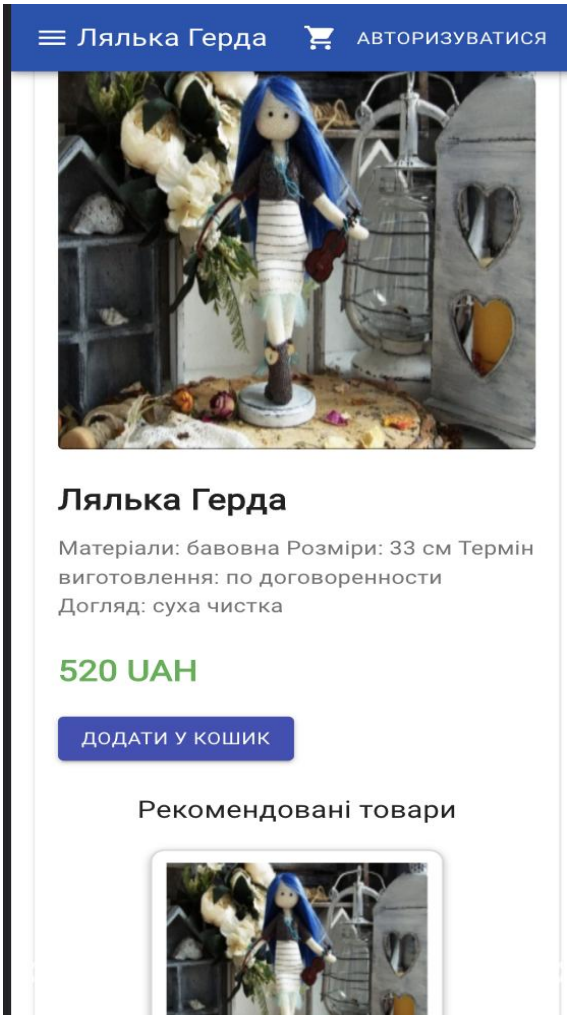


Рисунок 3.6 – Сторінка товару (адаптив)

Корзина на сайті є динамічною вона синхронізується з базою даних.

Робота синхронізації даних у корзині при вході користувача реалізована наступним чином:

1. Збереження товарів у локальному сховищі:

- Коли не зареєстрований користувач додає товари до корзини, ці товари зберігаються у локальному сховищі браузера, наприклад, в `localStorage` або `sessionStorage`. Кожен товар може бути представлений об'єктом з різними властивостями, такими як ідентифікатор, назва, ціна тощо.

2. Авторизація користувача:

- Коли користувач авторизується на сайті, його дані аутентифікації та ідентифікатор користувача зберігаються в базі даних. В результаті авторизації користувач отримує унікальний токен або сесійний ідентифікатор, що дозволяє

серверу ідентифікувати його під час взаємодії.

3. Синхронізація даних з базою даних:

- При авторизації користувача сервер перевіряє, чи є товари у локальному сховищі, пов'язаному з цим користувачем. Якщо такі дані існують, сервер отримує ці дані та зберігає їх у базі даних, пов'язаній з цим користувачем. Товари у локальному сховищі можуть бути пов'язані з ідентифікатором користувача або сесійним ідентифікатором.

- Після синхронізації даних з базою даних локальні дані у сховищі можуть бути очищені або оновлені з бази даних для підтвердження успішної синхронізації.

4. Об'єднання товарів:

- Після синхронізації користувач може продовжити використовувати сайт, але вже зареєстрований. Всі його додані товари в корзину, які були збережені у базі даних, об'єднуються з поточним станом корзини на стороні клієнта.

- Товари, що зберігаються в базі даних, додаються до корзини, якщо вони відсутні у поточному стані корзини. Якщо товар вже присутній у корзині, можуть бути виконані різні дії, такі як оновлення кількості товару або збереження його оригінальних параметрів.

Цей процес синхронізації даних у корзині забезпечує зручну та безперебійну переходов користувачів з незареєстрованого стану до зареєстрованого, дозволяючи їм зберігати свої обрані товари навіть після авторизації.

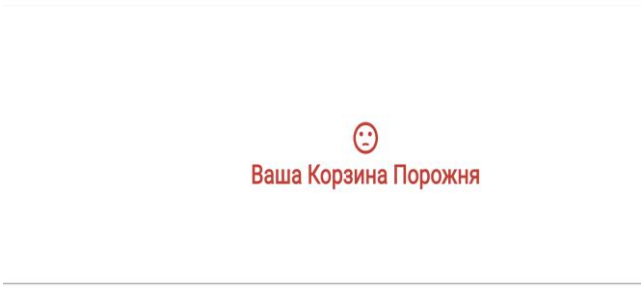


Рисунок 3.7 – Сторінка порожньої корзини

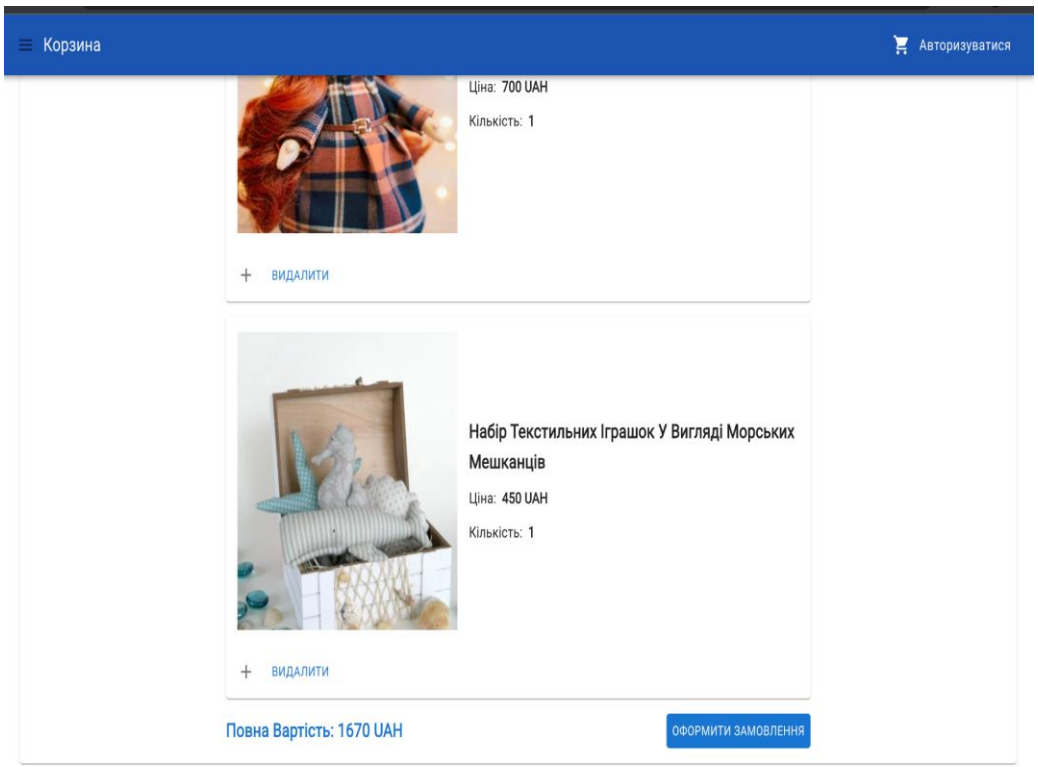


Рисунок 3.8 – Сторінка корзини

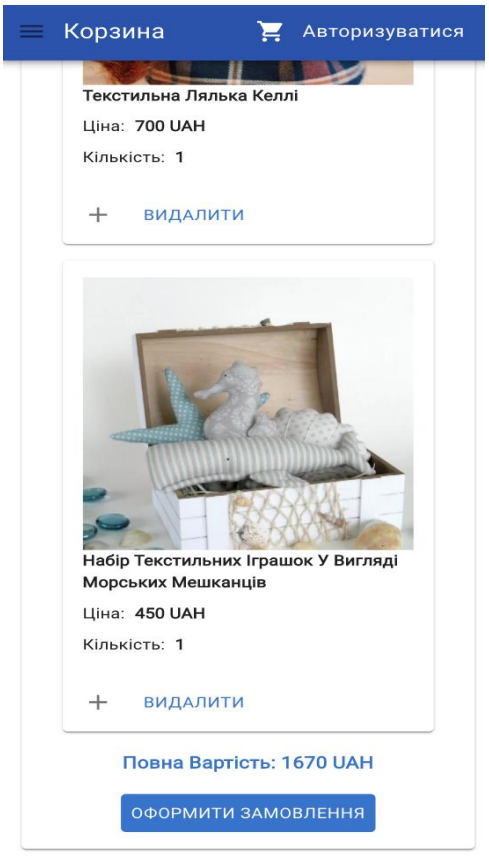


Рисунок 3.9 – Сторінка корзини (адаптив)

В корзині користувач має можливість змінити кількість певного товару та в залежності від кількості товару зміниться повна вартість.

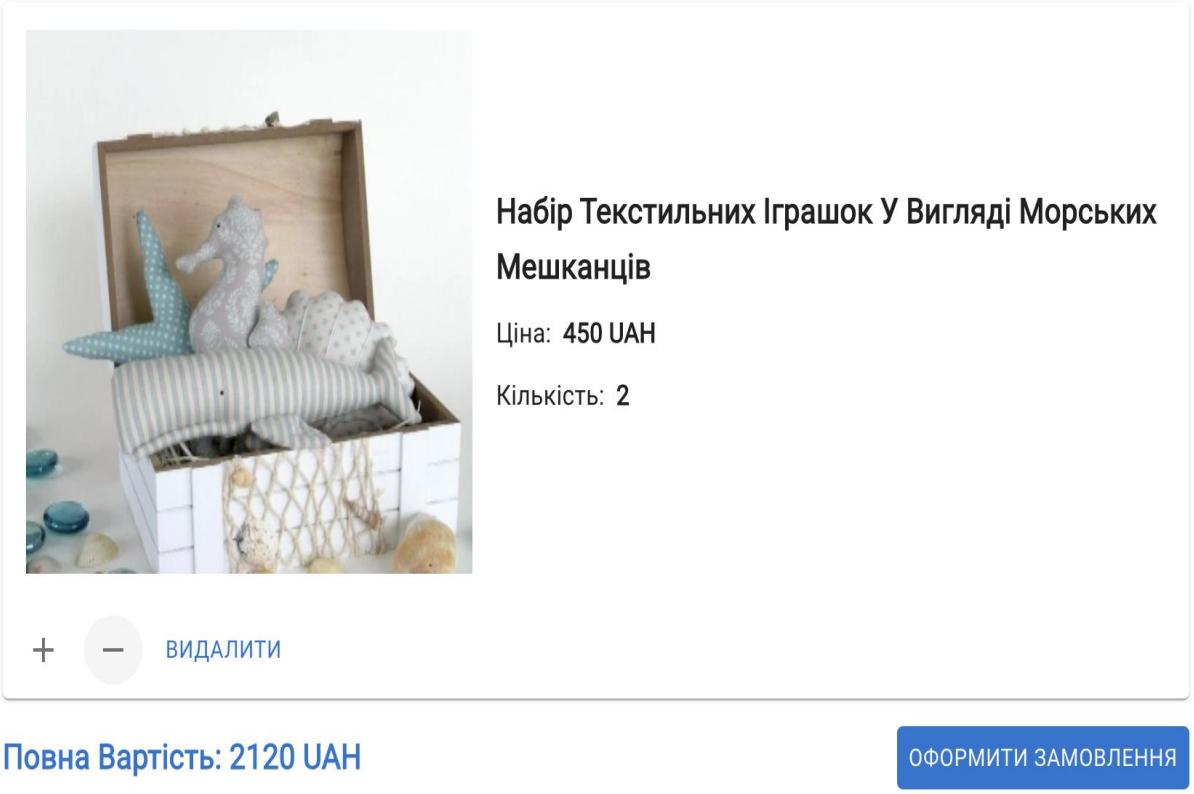


Рисунок 3.10 – Зміна кількості товару

Щоб оформити замовлення користувач повинен бути авторизован у системі. Якщо він не авторизован він буде бачити повідомлення.

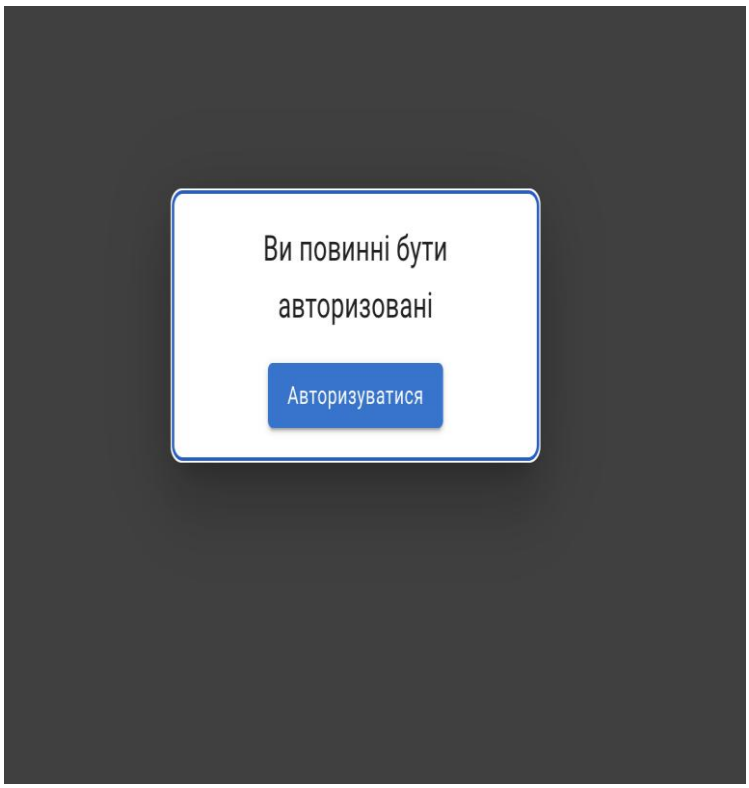


Рисунок 3.11 – Вікно для авторизації

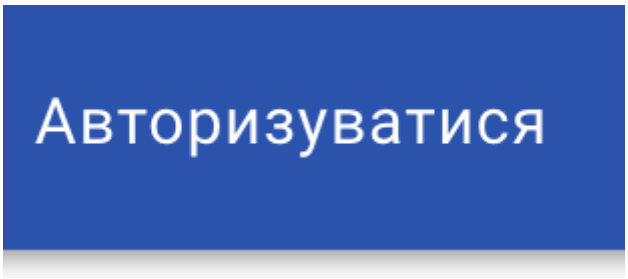


Рисунок 3.12 – Кнопка для авторизації на сайті

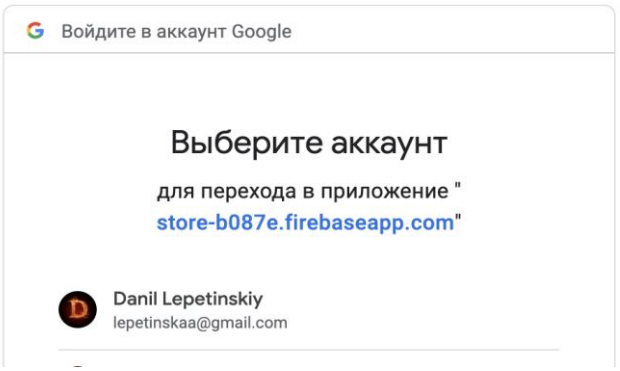


Рисунок 3.13 – Процесс авторизації через гугл аккаунт

Оскільки авторизація проходити через гугл аккаунт я маю можливість відобразити на сайті відкриті дані користувача, щоб це було більш інтерактивно.


Рисунок 3.14 – Відображення авторизованого користувача

В поля для оформлення замовлення я передаю інформацію з гугл акаунта користувача.

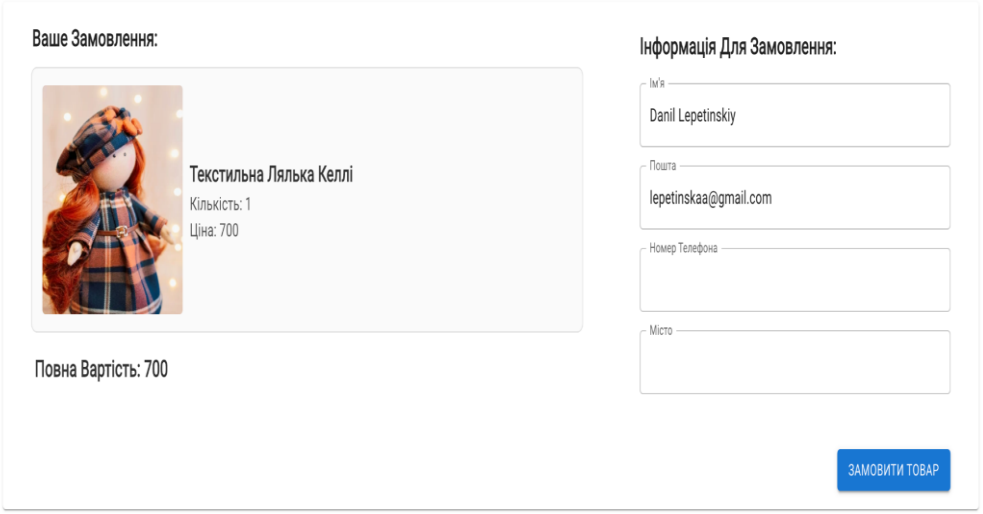


Рисунок 3.15 – Сторінка оформлення замовлення

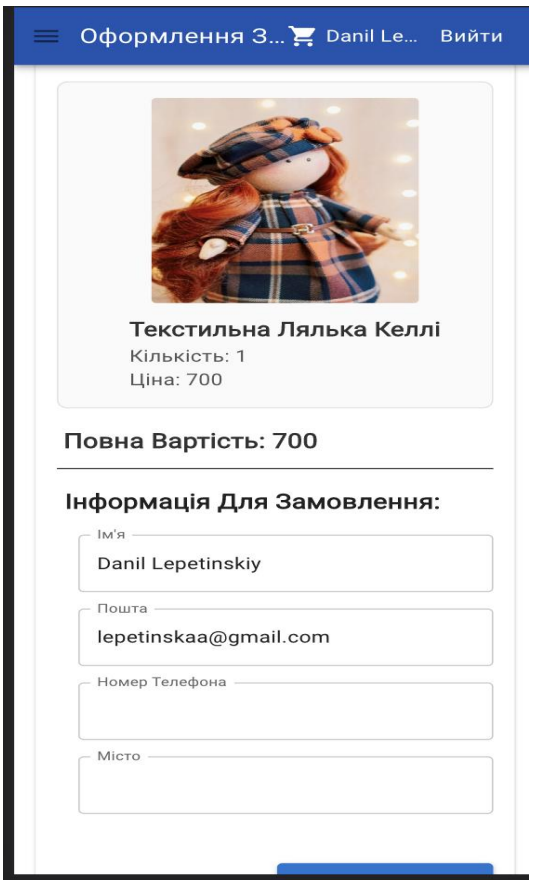


Рисунок 3.16 – Сторінка оформлення замовлення(адаптив)

Валідація - це процес перевірки введених користувачем даних з метою визначення їх відповідності вимогам або правилам, встановленим для конкретного поля або форми. Валідація допомагає переконатися, що дані, введені користувачем, є коректними та придатними для подальшої обробки або збереження.

React Hook Form (реактивна форма з використанням хуків) та Yup - це дві популярні бібліотеки для валідації форм у React-застосунках.

Реалізація валідації за допомогою React Hook Form та Yup включає наступні кроки:

1. Визначення форми:

- Створіть компонент форми та використовуйте хук `useForm` з `react-hook-form` для ініціалізації форми та отримання необхідних методів та стану.

2. Визначення схеми валідації:

- Створіть схему валідації за допомогою `yup`, визначивши правила для

кожного поля форми. Наприклад, ви можете визначити, що поле є обов'язковим, має відповідати певному типу даних або має певну мінімальну/максимальну довжину.

3. Прикріплення валідації до полів форми:

- Додайте `ref` до кожного поля форми та використовуйте методи `register` з `react-hook-form` для прикріплення цього поля до форми та валідації.

4. Відображення помилок:

- Використовуйте метод `errors` з `react-hook-form` для отримання інформації про помилки валідації. Відобразіть ці помилки поряд з відповідними полями форми для повідомлення користувачеві про помилкові дані.

5. Обробка подання форми:

- Використовуйте метод `handleSubmit` з `react-hook-form` для обробки подання форми. Перевірте, чи всі дані валідні, і виконайте відповідні дії, такі як надсилання даних на сервер або збереження в локальному стані.

Загальною ідеєю використання React Hook Form та Yup є поєднання простоти та ефективності для реалізації валідації форм в React-застосунках. Вони допомагають спростити процес валідації, забезпечуючи зручний та надійний спосіб перевірки введених даних та управління формою.

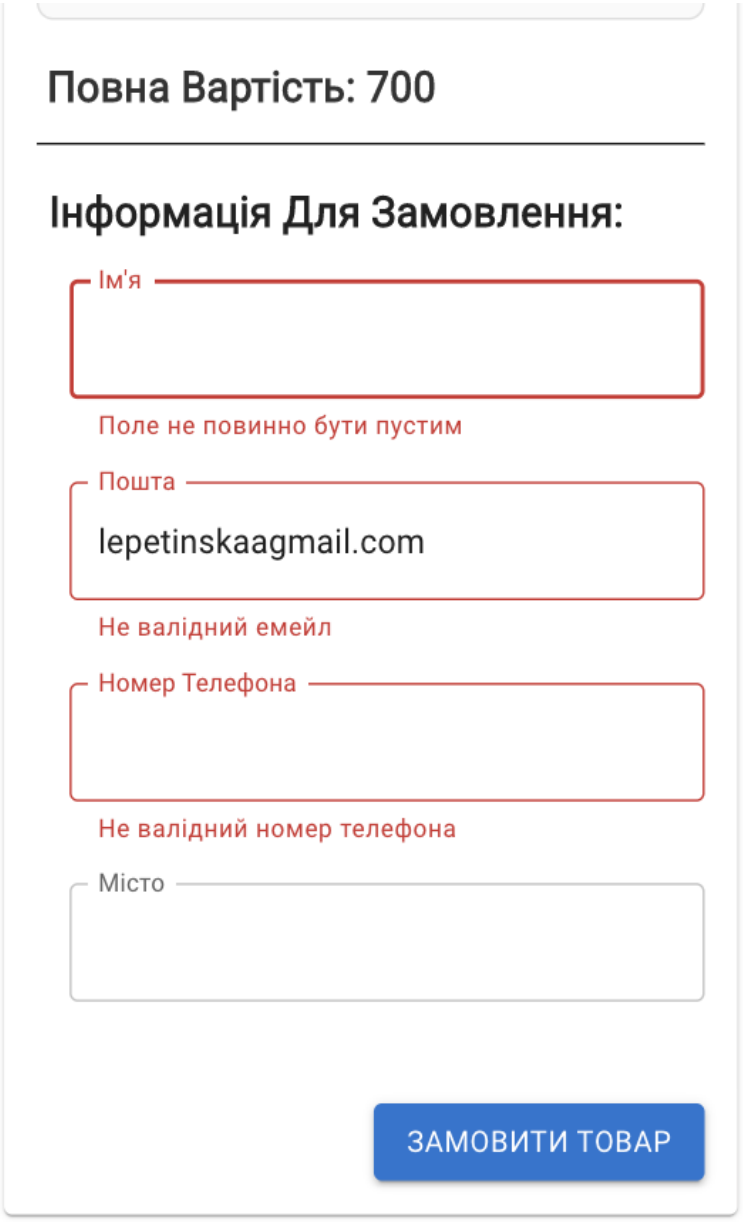


Рисунок 3.17 – Демонстрація роботи валідації


Рисунок 3.18 – Вікно прийнятого замовлення

Сторінка додавання товару доступна лише адміну. НА цій сторінці адмін обирає до якого типу товару він хоче додати свій товар, заповнює поле з іменем, ціною та описом. Також він повинен додати картинку, коли картинка додана відображається зелена галочка.

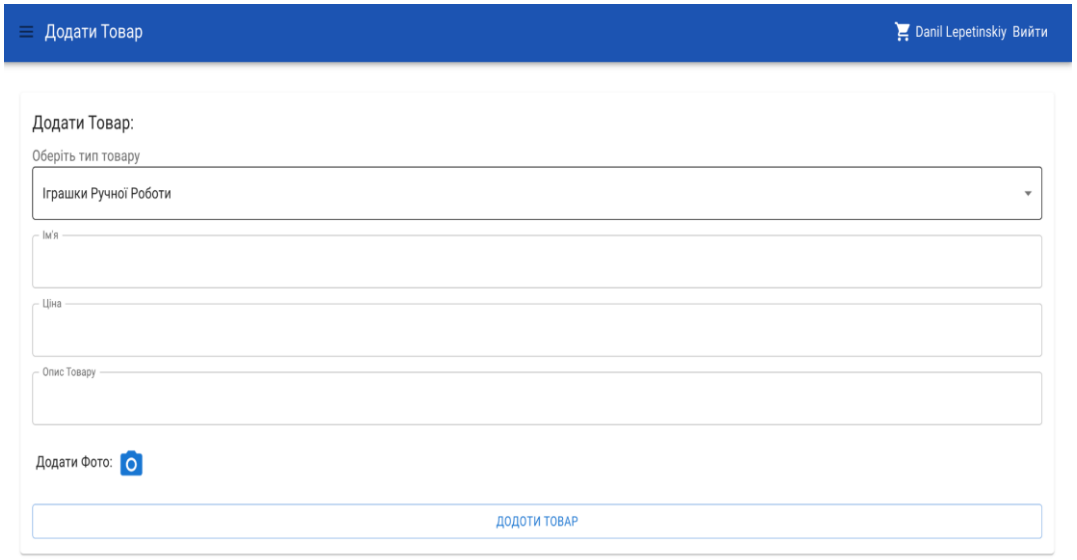


Рисунок 3.19 – Сторінка додавання товару

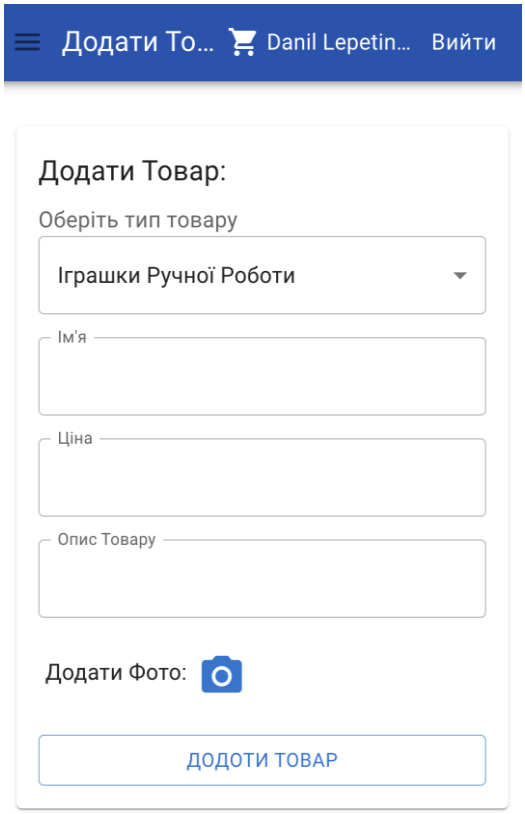


Рисунок 3.20 – Сторінка додавання товару(адаптив)

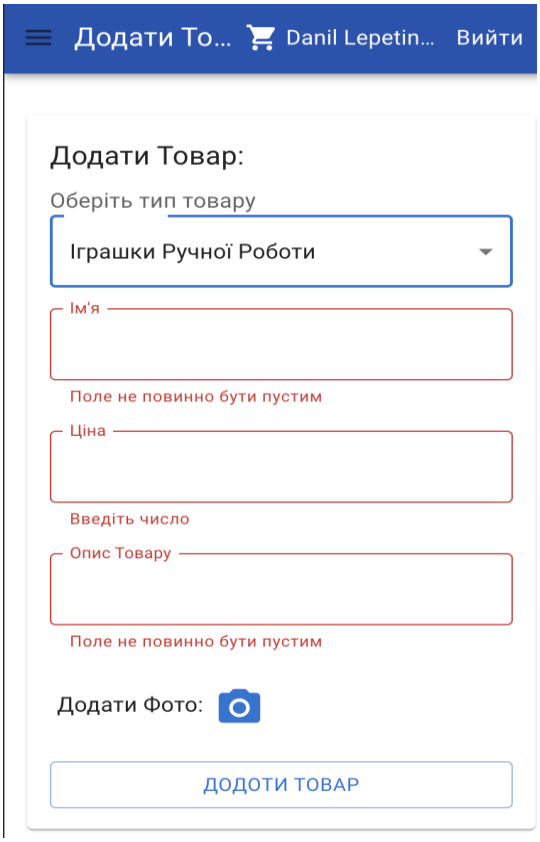


Рисунок 3.20 – Демонстрація валідації

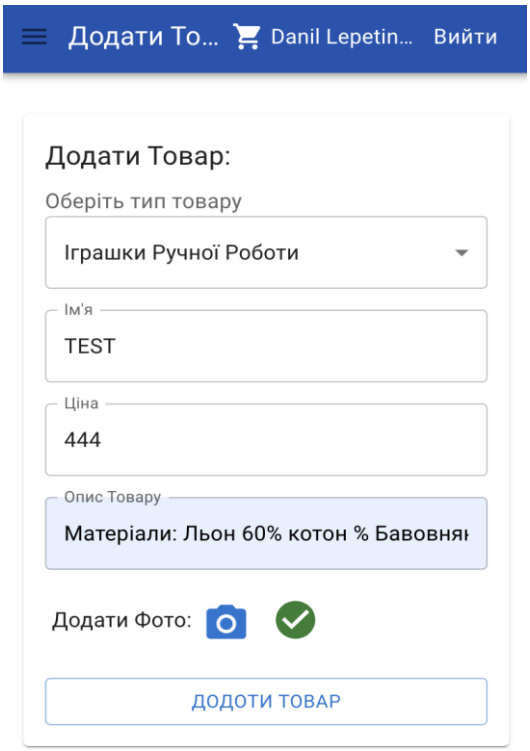


Рисунок 3.20 – Демонстрація заповненої форми

На веб-сайті існує особлива сторінка під назвою "Замовлення", яка є динамічною та надає різні функціональності залежно від ролі користувача, який її відкриває. Ця сторінка має декілька важливих особливостей.

1. Доступ для адміністратора:

- Коли адміністратор відкриває сторінку "Замовлення", він спочатку бачить список всіх покупців, які здійснювали замовлення на нашому сайті. Цей список містить інформацію про кожного покупця, таку як їх ім'я, контактні дані тощо.

2. Перегляд замовлень покупця:

- Адміністратор має можливість вибрати певного покупця зі списку і переглянути всі замовлення, які були здійснені цим покупцем. Ця функціональність дозволяє адміністратору отримати повну картину про активність кожного покупця на сайті.

3. Перегляд деталей окремого замовлення:

- Після вибору певного замовлення від обраного покупця, адміністратор може переглянути всі деталі цього замовлення. Це включає інформацію про товари, які були замовлені, кількість, ціну, адресу доставки тощо. Ця функціональність дозволяє адміністратору отримати повний огляд процесу замовлення та забезпечити якість обслуговування покупців.

4. Доступ для покупця:

- Коли покупець відкриває сторінку "Замовлення", він бачить лише свої замовлення, які він здійснив на нашому сайті. Це дозволяє покупцю переглянути всі свої попередні замовлення та отримати інформацію про них.

5. Перегляд окремого замовлення покупця:

- Покупець має можливість відкрити будь-яке своє замовлення та переглянути всі його деталі. Це включає інформацію про замовлені товари, кількість, ціну, адресу доставки тощо. Ця функціональність допомагає покупцю відстежувати статус і деталі своїх замовлень.

6. Авторизація:

- Доступ до сторінки "Замовлення" доступний лише авторизованим користувачам. Це забезпечує конфіденційність та безпеку інформації, пов'язаної з замовленнями та покупцями.

Таким чином, сторінка "Замовлення" на нашому веб-сайті надає зручний та простий спосіб перегляду та керування замовленнями для адміністраторів та покупців. Вона дозволяє отримати повну інформацію про замовлення та забезпечує зручну навігацію між різними рівнями деталізації.

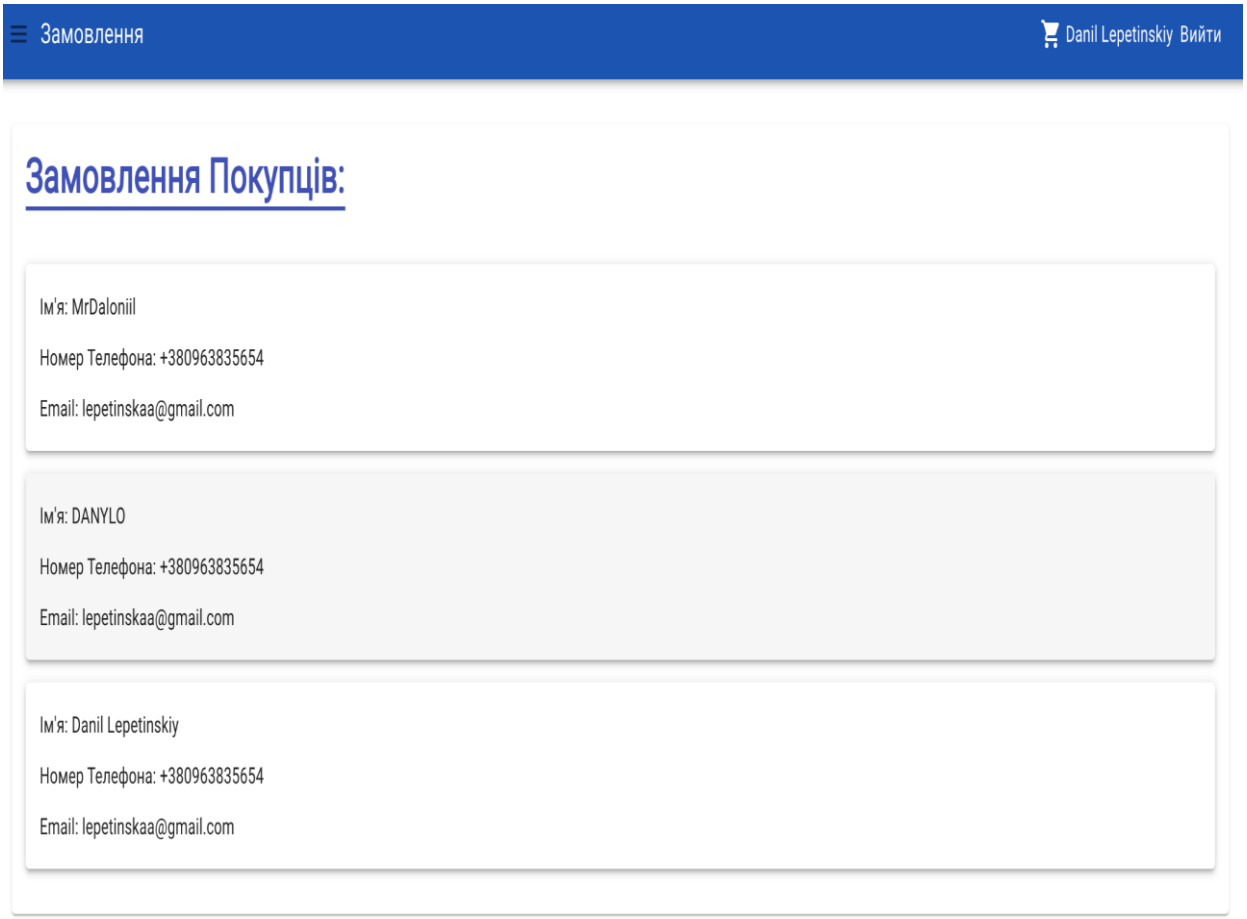


Рисунок 3.21 – Сторінка покупців

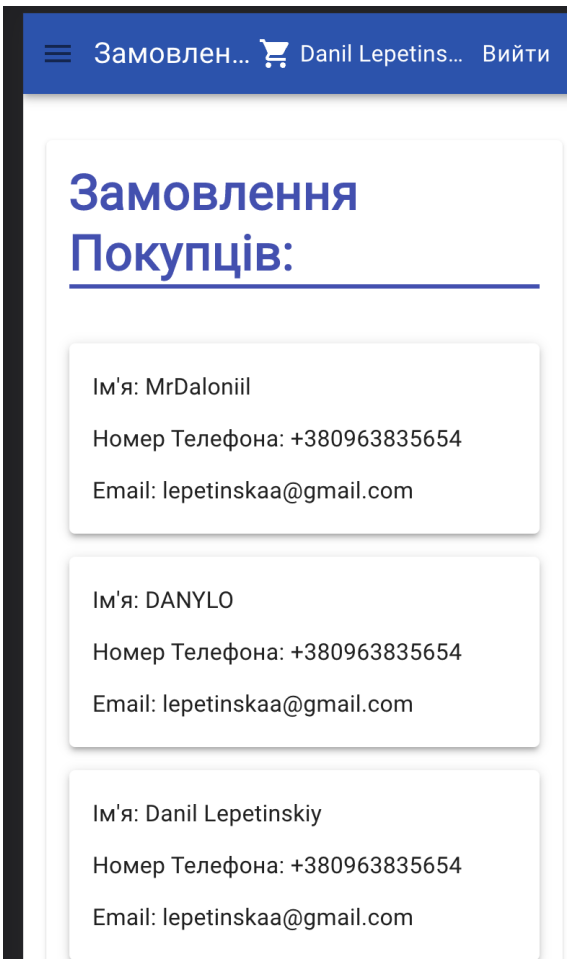


Рисунок 3.22 – Сторінка покупців(адаптив)

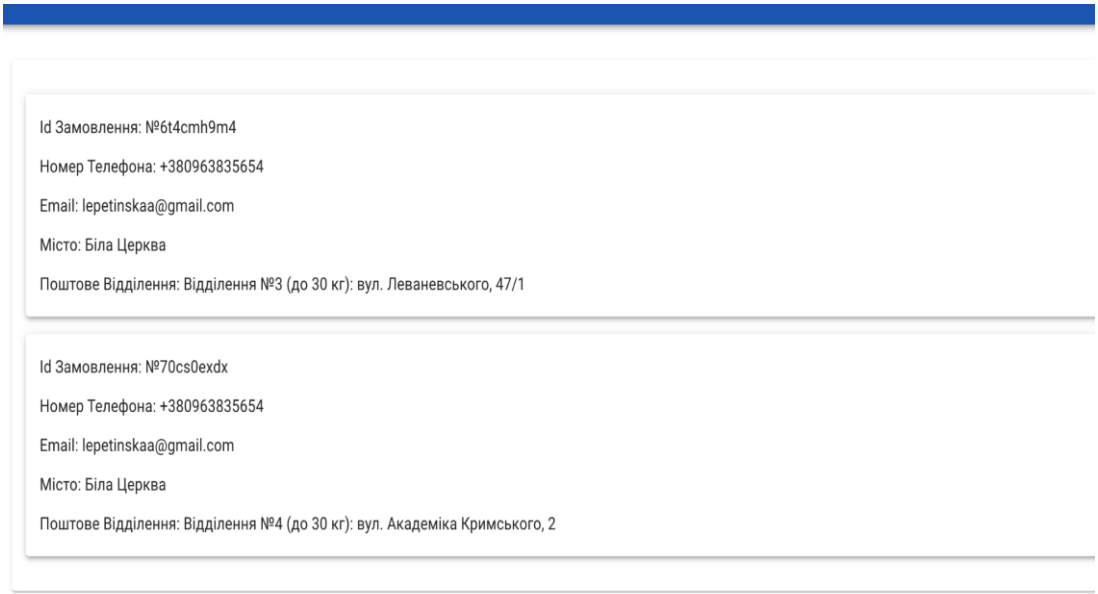


Рисунок 3.23 – Сторінка замовлень покупця

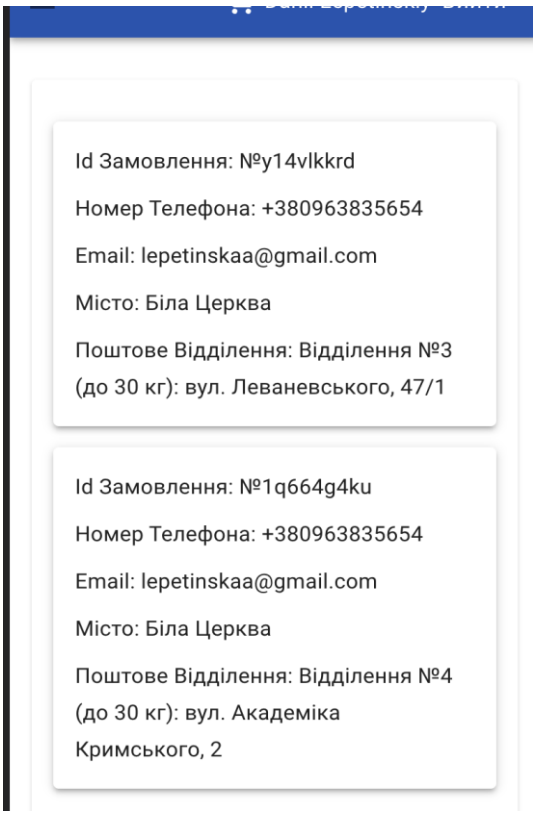


Рисунок 3.24 – Сторінка замовлень покупця(адаптив)

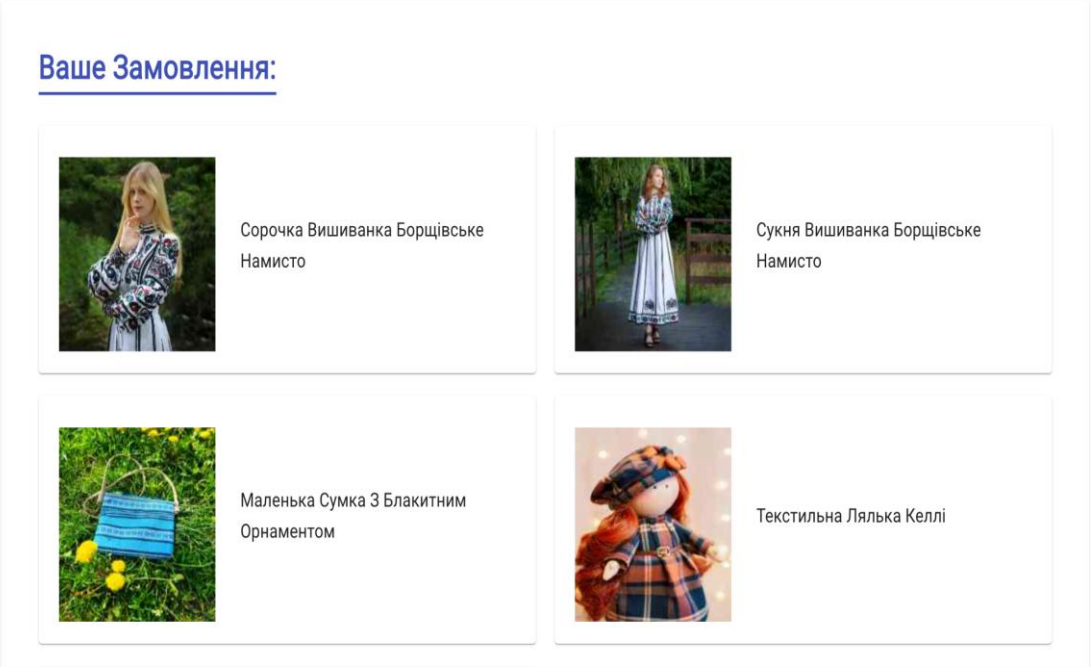


Рисунок 3.25 – Сторінка замовлення покупця

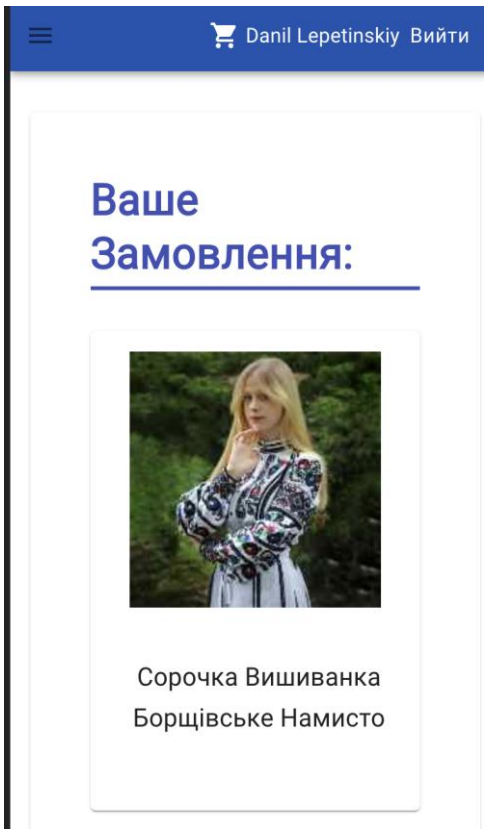


Рисунок 3.26 – Сторінка замовлення покупця(адаптив)

3.2 Захист персональних даних користувачів

Захист персональних даних користувачів є критично важливим аспектом розробки веб-застосунків. Firebase, як хмарна платформа розробки веб-застосунків, надає потужні інструменти для авторизації та забезпечення безпеки персональних даних користувачів. Нижче описані основні принципи та можливості захисту персональних даних через авторизацію за допомогою Firebase.

1. Аутентифікація та авторизація:

- Firebase надає механізми аутентифікації, такі як електронна пошта та пароль, соціальні мережі (Google, Facebook, Twitter і т.д.), а також інші методи, які дозволяють користувачам створювати облікові записи та входити в систему.

- Після аутентифікації Firebase забезпечує систему авторизації, яка дозволяє керувати правами доступу користувачів до різних ресурсів та функціональності веб-застосунку. Ви можете визначати права доступу на

основі ролей або інших критеріїв та забезпечувати безпеку даних, обмежуючі доступ до конфіденційної інформації.

2. Захист даних:

- Firebase забезпечує захищені механізми збереження та передачі даних, зокрема використовуючи шифрування на рівні пересилання (SSL/TLS) для захисту даних, які передаються між клієнтом і сервером.

- Firebase також надає можливість шифрування даних на сервері, що додатково забезпечує конфіденційність інформації.

3. Моніторинг активності та аудит:

- Firebase дозволяє вам вести моніторинг активності користувачів, щоб виявити можливі порушення безпеки. Ви можете відстежувати логи входу, дії користувачів та інші події для виявлення несподіваних або підозрілих активностей.

- Також можна налаштувати систему аудиту, яка записувати події, пов'язані з доступом до даних або зміною прав доступу, що дозволяє вам встановлювати контроль над діями користувачів та виявляти можливі загрози безпеці.

4. Захист від злому:

- Firebase надає різні механізми для захисту від зламу, такі як захист від введення неправильних даних, обмеження запитів з небезпечними параметрами, захист від перехоплення сесії та багато іншого. Вона також надає оновлення безпеки та патчі для виявлених вразливостей.

Загалом, захист персональних даних користувачів через авторизацію за допомогою Firebase є складним процесом, що включає такі аспекти, як аутентифікація, авторизація, захист даних, моніторинг та захист від зламу. Firebase надає потужні інструменти, які допомагають розробникам забезпечити безпеку та конфіденційність персональних даних, забезпечуючи надійність та довіру користувачів.

# ВИСНОВОК

У ході виконання кваліфікаційної роботи було проведено ретельне дослідження теоретичних аспектів та основ побудови системи продажу товарів в Інтернеті. Аналізуючи програмно-технологічні рішення інформаційних систем електронної торгівлі споживчими товарами, було виявлено ключові відмінності та особливості електронної комерції та електронного бізнесу. Це дало змогу встановити належні орієнтири для розробки власного проекту, а також визначити необхідний функціонал для подальшої реалізації.

На основі обраних програмно-технологічних рішень, було розроблено структуру системи, включаючи файлову систему та локальне зберігання даних користувачів. Це дало можливість належно організувати зберігання та доступ до інформації, що стосується користувачів та їх взаємодії з системою.

Основним результатом проекту став розроблений прототип програмної системи для онлайн-продажу товарів. Цей прототип вміщує в собі всі необхідні функціональні можливості, які вимагаються від подібних систем. Впровадження такої системи дозволить користувачам легко та зручно здійснювати покупки в Інтернеті.

Під час розробки програмного забезпечення для проекту було використано передові технології, такі як TypeScript, Next.js, Firebase, Material UI та інші допоміжні бібліотеки. Це дозволило створити ефективну та сучасну систему, яка забезпечує високу продуктивність та зручний інтерфейс користувача.

В результаті кваліфікаційної роботи було досягнуто підвищення ефективності роботи магазина виробами ручної роботи, шляхом створення повноцінного та сучасного веб-сайту. Досліджено теоретичні основи, проаналізовано програмно-технологічні рішення та успішно реалізовано функціонал системи. Цей проект відображає необхідність та переваги електронної торгівлі у сучасному світі та надає зручний та безпечний спосіб здійснення покупок для користувачів.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. ЗУ «Про платіжні системи та переказ коштів в Україні» (Відомості Верховної Ради України (ВВР), 2001, № 29, ст.137) [Електронний ресурс] // Zakon.rada.gov.ua – URL: https://zakon.rada.gov.ua/laws/show/2346-14#Text (дата звернення: 15.12.2022)
2. Ecommerce business models and how to take your business online [Електронний ресурс] // Adobe – URL: https://business.adobe.com/blog/basics/ecommerce-business-models (дата звернення: 16.12.2022)
3. Understanding the Different Types of Ecommerce Business Models [Електронний ресурс] // BigCommerce – URL: https://www.bigcommerce.com/articles/ecommerce/types-of-business-models/ (дата звернення: 16.12.2022)
4. What Is C2C? How Does the Customer-to-Customer Model Work? [Електронний ресурс] // Investopedia – URL: https://www.investopedia.com/terms/c/ctoc.asp (дата звернення: 17.12.2022)
5. Business-to-government [Електронний ресурс] // Вікіпедія – URL: https://en.wikipedia.org/wiki/Business-to-government (дата звернення: 20.12.2022)
6. E-Governance Models: G2C & G2B - A brief introduction [Електронний ресурс] // Linkedin – URL: https://www.linkedin.com/pulse/e-governance-models-g2c-g2b-brief-introduction-falak-dutta/ (дата звернення: 20.12.2022)
7. What is business to employee (B2E) and how can it impact e‑ commerce? [Електронний ресурс] // Sana-commerce – URL: https://www.sana-commerce.com/blog/what-does-b2e-mean-for-ecommerce/ (дата звернення: 20.12.2022)
8. The World's Largest Online Marketplaces [Електронний ресурс] // Statista – URL: https://www.statista.com/chart/24405/top-5-online-market-places-by-gmv/ (дата звернення: 22.12.2022)

9. Charts: Top Global Marketplaces by Number of Visits [Електронний ресурс] // PracticalEcommerce – URL: https://www.practicalecommerce.com/charts-top-global-marketplaces-by-number-of-visits (дата звернення: 23.12.2022)
10. Projected retail e-commerce GMV share of Amazon in the United States from 2016 to 2021 [Електронний ресурс] // Statista – URL: https://www.statista.com/statistics/788109/amazon-retail-market-share-usa/ (дата звернення: 24.12.2022)
11. EBay's Market Share Slips to Below 5% [Електронний ресурс] // Marketplace Pulse – URL: https://www.marketplacepulse.com/articles/ebays-market-share-slips-to-below-5 (дата звернення: 27.12.2022)
12. 25+ Top eBay Statistics 2023 (User And Revenue Data) [Електронний ресурс] // Marketplace Pulse – URL: https://startupbonsai.com/ebay-statistics/ (дата звернення: 29.12.2022)
13. Artificial Intelligence in Ecommerce [Електронний ресурс] // BigCommerce – URL: https://www.bigcommerce.com/articles/ecommerce/ecommerce-ai/ (дата звернення: 29.12.2022)
14. Client-Server Architecture [Електронний ресурс] // BigCommerce – URL: https://medium.com/codex/client-server-architecture-5e103aa0106d (дата звернення: 30.12.2022)
15. Architectural characteristics of web-based applications [Електронний ресурс] // BigCommerce – URL:https://www.ibm.com/docs/en/db2-for-zos/11?topic=environment-architectural-characteristics-web-based-applications (дата звернення: 30.12.2022)
16. Ковтун Л. О., Ткачук В. М., Франчук Р. Вибір архітектури програмного забезпечення інформаційної навчальної системи [Електронний ресурс] // Вісник Хмельницького національного університету, №5 (277). 2019 – URL: http://journals.khnu.km.ua/vestnik/wp-content/uploads/2021/01/42-6.pdf (дата звернення: 04.01.2023)

17. Supply Chain Management, SCM [Електронний ресурс] // IT Enterprice – URL: https://www.it.ua/knowledge-base/technology-innovation/supply-chain-management-scm (дата звернення: 05.01.2023)
18. Why Is Customer Relationship Management So Important? [Електронний ресурс] // Forbes – URL: https://www.forbes.com/sites/forbesagencycouncil/2017/10/24/why-is-customer-relationship-management-so-important/ (дата звернення: 05.01.2023)
19. 85+ Impressive CRM Statistics You Need to Know in 2023 [Електронний ресурс] // WebFX – URL: https://www.webfx.com/blog/marketing/crm-statistics/ (дата звернення: -06.01.2023)
20. Graphical user interface [Електронний ресурс] // Britannica – URL: https://www.britannica.com/technology/graphical-user-interface (дата звернення: 08.01.2023)
21. Mobile retail commerce sales as percentage of retail e-commerce sales worldwide from 2016 to 2021 [Електронний ресурс] // Statista – URL: https://www.statista.com/statistics/806336/mobile-retail-commerce-share-worldwide/ (дата звернення: 10.01.2023)
22. Top 10+ Mobile Commerce Statistics for 2023 [Електронний ресурс] // Tidio – URL: https://www.tidio.com/blog/mobile-commerce-statistics/ (дата звернення: 11.01.2023)
23. Front-End Development: The Complete Guide [Електронний ресурс] // Cloudinary – URL: https://cloudinary.com/guides/front-end-development/front-end-development-the-complete-guide (дата звернення: 11.01.2023)
24. HTML: HyperText Markup Language [Електронний ресурс] // MDN Web Docs – URL: https://developer.mozilla.org/en-US/docs/Web/HTML (дата звернення: 12.01.2023)
25. CSS [Електронний ресурс] // MDN Web Docs – URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 12.01.2023)

26. JavaScript [Електронний ресурс] // MDN Web Docs – URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 12.01.2023)
27. What Is Back-End Development? [Електронний ресурс] // BuiltIn – URL: https://builtin.com/software-engineering-perspectives/back-end-development (дата звернення: 14.01.2023)
28. What's the Difference? Relational vs Non-Relational Databases [Електронний ресурс] // Insightsoftware – URL: https://insightsoftware.com/blog/whats-the-difference-relational-vs-non-relational-databases/ (дата звернення: 14.01.2023)
29. Hypertext Transfer Protocol [Електронний ресурс] // Oracle – URL: https://docs.oracle.com/cd/E19857-01/817-6247-10/agaphttp.html (дата звернення: 15.01.2023)
30. The Rise of Cryptocurrency and What It Means for Ecommerce [Електронний ресурс] // BigCommerce – URL: https://www.bigcommerce.com/articles/ecommerce/cryptocurrency-ecommerce/ (дата звернення: 17.01.2023)
31. Number of Email Users Worldwide 2022/2023: Demographics & Predictions [Електронний ресурс] // FinancesOnline – URL: https://financesonline.com/number-of-email-users/ (дата звернення: 18.01.2023)
32. Amazon.com Top Marketing Channels [Електронний ресурс] // Similarweb – URL: https://www.similarweb.com/website/amazon.com/#traffic-sources (дата звернення: 19.01.2023)
33. The Role of Search Engine Optimization in Search Marketing [Електронний ресурс] // ResearchGate – URL: https://www.researchgate.net/publication/228294453_The_Role_of_Search_Engine_Optimization_in_Search_Marketing (дата звернення: 21.01.2023)
34. Магазин Amazon.com [Електронний ресурс] // Amazon – URL: https://www.amazon.com/ (дата звернення: 26.01.2023)
35. Магазин Ecolaglobal.com [Електронний ресурс] // Ecolaglobal – URL: https://www.ecolaglobal.com/shop (дата звернення: 10.02.2023)

36. Магазин Vito.store [Електронний ресурс] // Vito – URL: https://vito.store/ (дата звернення: 11.02.2023)
37. ЗУ «Про захист персональних даних» (Відомості Верховної Ради (ВВР), 2010, №34, ст.481) [Електронний ресурс] // Zakon.rada.gov.ua – URL: https://zakon.rada.gov.ua/laws/show/2297-17#Text (дата звернення: 15.02.2023)
38. Бем М. В., Городиський І. М., Саттон Г., Родіоненко О. М. Захист персональних даних: Правове регулювання та практичні аспекти: науково-практичний посібник. – К.: К.І.С., 2015. – 220 с.
39. E-commerce: Business, Technology, Society by Kenneth C. Laudon, Carol Guercio Traver. URL: https://www.amazon.com/E-commerce-Business-Technology-Society-15th/dp/0134601566
40. The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses by Eric Ries. URL: https://www.amazon.com/Lean-Startup-Entrepreneurs-Continuous-Innovation/dp/0670921602
41. Плескач В. Л. Електронна комерція: Підручник / В. Л. Плескач, Т. Г. Затонацька. К.: Знання, 2007, 535 с.