

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»  
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ  
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## **ПОЯСНЮВАЛЬНА ЗАПИСКА**

до кваліфікаційної роботи освітнього ступеня «бакалавр»  
за спеціальністю 121 «Інженерія програмного забезпечення»  
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Сайт для створення вебмагазинів»**

Виконав студент групи ПЗ-20-1  
ЛПІНСЬКИЙ Олександр Вікторович

Керівник роботи:  
ГОРДІЄНКО Ірина Валеріївна

Консультант:  
ЧИЖМОТРИЯ Олена Геннадіївна

Рецензент:  
ВАКАЛЮК Тетяна Анатоліївна

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»  
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ  
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

**«ЗАТВЕРДЖУЮ»**

Зав. кафедри інженерії  
програмного забезпечення

Тетяна ВАКАЛЮК

**«26» лютого 2024 р.**

**ЗАВДАННЯ**  
**на кваліфікаційну роботу**

Здобувач вищої освіти: **ЛПІНСЬКИЙ Олександр Вікторович**

Керівник роботи: **ГОРДІЄНКО Ірина Валеріївна**

Тема роботи: **«Сайт для створення вебмагазинів»**,

затверджена Наказом закладу вищої освіти від **«23» лютого 2024 р., №74/с**

Вихідні дані для роботи: **програма для створення вебмагазинів**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів,  
що їх стосуються:

| Розділ | Консультант                     | Завдання видав | Завдання прийняв |
|--------|---------------------------------|----------------|------------------|
| 1      | Гордієнко І.В.<br>Чижмотря О.Г. | 02.03.2024р    | 02.03.2024р      |
| 2      | Гордієнко І.В.<br>Чижмотря О.Г. | 04.04.2024р    | 04.04.2024р      |
| 3      | Гордієнко І.В.<br>Чижмотря О.Г. | 02.05.2024р    | 02.05.2024р      |

## РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра представлена у вигляді розробки програми для створення та налаштування вебмагазинів, яка дозволяє користувачам швидко налаштовувати тему, тексти, адаптивність та зміст сторінок вебмагазину, а також ефективно управляти своїми товарами та замовленнями. Пояснювальна записка містить детальний опис архітектури та алгоритмів програми, а також ключових особливостей системи редагування магазину.

Основною метою роботи є створення зручної та інтуїтивно зрозумілої вебпрограми, що допоможе користувачам швидше запускати власні інтернет-магазини, налаштовувати їх зміст, а також ефективно управляти процесом замовлення товарів.

В роботі наведено порівняльний аналіз з існуючими аналогами програми, які дозволяють налаштовувати зміст та вигляд магазину.

Пояснювальна записка до кваліфікаційної роботи містить 71 сторінку, 28 ілюстрацій, 20 таблиць, 3 додатки, та 30 літературних джерел.

Ця кваліфікаційна робота має на меті забезпечити глибоке розуміння процесів розробки вебмагазинів та поглибити розуміння програмних засобів, що полегшують створення та керування магазинами.

**КЛЮЧОВІ СЛОВА:** ВЕБМАГАЗИН, НАЛАШТУВАННЯ, ДИЗАЙН, ТЕМА, ЗМІСТ, АДАПТИВНІСТЬ

|           |      |                 |        |      |                                    |                   |      |         |
|-----------|------|-----------------|--------|------|------------------------------------|-------------------|------|---------|
|           |      |                 |        |      | ІПЗ.КР.Б-121-24-ПЗ                 |                   |      |         |
|           |      |                 |        |      |                                    |                   |      |         |
| Змн.      | Арк. | № докум.        | Підпис | Дата | Сайт для створення<br>вебмагазинів | Літ.              | Арк. | Аркушів |
| Розроб.   |      | Ліпінський О.В. |        |      |                                    |                   |      |         |
| Керівник  |      | Гордієнко І. В. |        |      |                                    |                   | 3    | 71      |
| Рецензент |      | Вакалюк Т.А.    |        |      |                                    | ФІКТ Гр. ІПЗ-20-1 |      |         |
| Зав. каф. |      | Вакалюк Т.А.    |        |      |                                    |                   |      |         |

## ABSTRACT

The bachelor's thesis is presented in the form of a program development for creating and customizing online stores, allowing users to quickly set up the theme, texts, and content of web store pages, as well as efficiently manage their products and orders. The explanatory note contains a detailed description of the program's architecture and algorithms, as well as the key features of the store editing system.

The main objective of the work is to create a convenient and intuitive web application that helps users to launch their own online stores more quickly, customize their content, and efficiently manage the product ordering process.

The work provides a comparative analysis with existing analogous programs that allow customization of the store's content and appearance.

The explanatory note for the thesis includes 71 pages, 28 illustrations, 20 tables, 3 appendices, and 30 literature sources.

This thesis project aims to ensure a deep understanding of the web store development processes and to deepen the understanding of software tools that facilitate the creation and management of stores.

**KEYWORDS:** ONLINE STORE, CUSTOMIZATION, DESIGN, THEME, CONTENT, ADAPTABILITY

|  |  |  |  |  |                    |      |
|--|--|--|--|--|--------------------|------|
|  |  |  |  |  | ИПЗ.КР.Б-121-24-ПЗ | Арк. |
|  |  |  |  |  |                    | 4    |
|  |  |  |  |  |                    |      |

## ЗМІСТ

|                                                                                           |           |
|-------------------------------------------------------------------------------------------|-----------|
| <b>РЕФЕРАТ.....</b>                                                                       | <b>3</b>  |
| <b>ABSTRACT .....</b>                                                                     | <b>4</b>  |
| <b>ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ .....</b>                                                    | <b>6</b>  |
| <b>ВСТУП.....</b>                                                                         | <b>7</b>  |
| <b>РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ<br/>ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ.....</b>          | <b>9</b>  |
| 1.1 Постановка задачі.....                                                                | 9         |
| 1.2 Аналіз аналогів програмного продукту .....                                            | 10        |
| 1.3 Вибір архітектури системи.....                                                        | 14        |
| 1.4 Обґрунтування вибору інструментальних засобів .....                                   | 14        |
| Висновки до першого розділу.....                                                          | 17        |
| <b>РОЗДІЛ 2. ПРОЄКТУВАННЯ ПРОГРАМИ.....</b>                                               | <b>19</b> |
| 2.1 Визначення варіантів використання та об'єктно-орієнтованої структури<br>системи ..... | 19        |
| 2.2 Розробка бази даних.....                                                              | 30        |
| 2.3 Проектування та реалізація алгоритмів роботи системи .....                            | 33        |
| 2.4 Реалізація функціоналу програми для створення вебмагазинів.....                       | 35        |
| Висновки до другого розділу .....                                                         | 40        |
| <b>РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПРОГРАМОЮ .....</b>                            | <b>42</b> |
| 3.1 Порядок встановлення та налаштування параметрів платформи .....                       | 42        |
| 3.2 Структура інтерфейсу сайту для створення вебмагазинів.....                            | 44        |
| 3.3 Тестування сайту для конструювання вебмагазинів .....                                 | 51        |
| Висновки до третього розділу.....                                                         | 52        |
| <b>ВИСНОВКИ .....</b>                                                                     | <b>54</b> |
| <b>СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ .....</b>                                                   | <b>56</b> |
| <b>ДОДАТКИ.....</b>                                                                       | <b>59</b> |

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

ООП – об'єктно орієнтоване програмування;

CRUD – Create, Read, Update, Delete (створення, читання, оновлення, видалення);

MVC – Model-View-Controller (модель-вигляд-контролер)

ПЗ – програмне забезпечення;

Рис – рисунок.

|  |  |  |  |  |                    |      |
|--|--|--|--|--|--------------------|------|
|  |  |  |  |  | ІПЗ.КР.Б-121-24-ПЗ | Арк. |
|  |  |  |  |  |                    | 6    |
|  |  |  |  |  |                    |      |

## ВСТУП

**Актуальність теми** полягає в зростаючій потребі у створенні доступних і ефективних інструментів для створення вебмагазинів, особливо з урахуванням зростаючої конкуренції та швидких змін у галузі електронної комерції. Сучасні технології пропонують нові можливості для спрощення процесів створення та управління вебсайтами, що дозволяє підприємцям і малим бізнесам швидше виходити на ринок та ефективно презентувати свої товари онлайн. Розробка конструктора вебмагазинів, який забезпечує легке налаштування та управління сторінок, відповідає актуальним вимогам ринку та сприяє розвитку електронної комерції, забезпечуючи підприємцям інтуїтивно зрозумілий та доступний інструмент для створення вебмагазину.

**Метою** випускної кваліфікаційної роботи бакалавра є розробка програми для створення та налаштування вебмагазинів, який включає інтуїтивно зрозумілі інструменти для управління дизайном, змістом, адаптивністю та функціональністю магазину, а також ефективні засоби для управління товарами та замовленнями.

**Завдання** розробки програми веб-магазинів є наступними:

1. Аналіз існуючих рішень — дослідження характеристик існуючих конструкторів вебсайтів та їх функціональних можливостей для ідентифікації корисних властивостей, які можна інтегрувати у власний проєкт.
2. Проектування архітектури — розробка структури програми, яка дозволить ефективно створювати та налаштовувати вебмагазин з урахуванням різних потреб користувачів.
3. Реалізація основних модулів програми — створення модулів для редагування дизайну, управління змістом, адаптивністю та інтеграції функцій пов'язаних з товарами та замовленнями.
4. Тестування конструктора — проведення тестів для перевірки зручності використання, надійності та ефективності роботи створеної системи.

**Об'єктом дослідження** є процес створення та управління веб-магазинами, який включає складові такі як налаштування структури та змісту сторінки, налаштування дизайну, адаптивності, текстів, а також операції з товарами від їх перегляду до замовлення. Особлива увага приділяється зручності редагування магазину.

**Предметом дослідження** є алгоритми та структура конструктора вебмагазину, який розробляється для забезпечення інтуїтивно зрозумілої роботи по налаштуванню сайту. Досліджуються методи оптимізації процесу створення вебмагазинів.

# РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

## 1.1 Постановка задачі

В кваліфікаційній роботі, виходячи з мети, потрібно реалізувати веборієнтовану систему для створення та налаштування веб-магазинів, яка дозволить користувачам швидко налаштовувати тему, тексти та зміст сторінок веб-магазину, а також ефективно управляти своїми товарами та замовленнями. Цей інструмент дозволить користувачам без технічних навичок легко створювати та налаштовувати свій магазин, швидко виходити на ринок, представляючи свої товари чи послуги онлайн.

1. Вивчення вимог і розробка технічного завдання.
  - 1.1. Дослідження наявних рішень.
  - 1.2. Аналіз прикладів використання або впровадження.
  - 1.3. Визначення ключових задач і можливостей продукту.
2. Вибір архітектури та технологій для реалізації.
  - 2.1. Порівняння можливих варіантів впровадження.
  - 2.2. Вибір та обґрунтування відповідних технологій.
  - 2.3. Інтеграція необхідних стандартних рішень і фреймворків.
3. Розробка та реалізація системи.
  - 3.1. Розробка архітектури серверної частини системи.
  - 3.2. Створення модулю користувачів.
  - 3.3. Створення редактора сторінок.
  - 3.4. Надання можливості налаштування теми.
  - 3.5. Надання можливості налаштування текстів.
  - 3.6. Забезпечення підтримки адаптивності для різних пристроїв.
  - 3.7. Розробка модулю повного циклу замовлення продукту.
4. Перевірка та тестування системи.
  - 4.1. Тестування редактора сторінок, а також всіх операцій пов'язаних з ним.
  - 4.2. Тестування всіх елементів, які можливо додати на сторінку.

#### 4.3. Виправлення недоліків, виявлених під час тестування.

В результаті буде створено повнофункціональний інструмент, який дозволить користувачам легко налаштовувати веб-магазини, редагувати сторінки, змінювати теми та тексти, а також ефективно управляти процесом замовлення товарів.

### 1.2 Аналіз аналогів програмного продукту

Розробка платформи для створення веб-магазинів вимагає порівняння з іншими існуючими рішеннями на ринку. Це дозволяє виявити сильні та слабкі сторони продуктів-конкурентів та визначити, які функції варто інтегрувати у власну систему. Розглянемо порівняння з трьома популярними платформами: WordPress, Squarespace та Wix.

**WordPress** (рис. 1.1) – це високо гнучка та розширювана платформа, широко відома своєю здатністю задовольняти різноманітні потреби вебсайтів, включаючи електронну комерцію. Вона пропонує величезний вибір плагінів і тем, що дозволяє користувачам налаштовувати свої сайти відповідно до конкретних вимог. Однією з її значних переваг є велика спільнота розробників і користувачів, які сприяють постійному вдосконаленню та забезпечують підтримку. Крім того, WordPress оснащений численними SEO-плагінами, які підвищують видимість сайту в пошукових системах. Однак її гнучкість супроводжується складністю, яка може вимагати певних технічних навичок для належного налаштування та обслуговування. Безпека також може бути проблемою через велику кількість плагінів, які потрібно управляти. Незважаючи на ці виклики, WordPress залишається економічно ефективним рішенням для тих, хто готовий інвестувати час у вивчення та обслуговування системи.



Рис. 1.1 – Інтерфейс WordPress [1]

#### Переваги:

- WordPress пропонує широкий спектр плагінів і тем, що дозволяють розширити функціональність сайту;
- Велика спільнота розробників і користувачів забезпечує постійну підтримку та оновлення;
- Багато плагінів для SEO, що допомагають покращити видимість сайту в пошукових системах.

#### Недоліки:

- Потребує певних технічних навичок для налаштування і підтримки;
- Велика кількість плагінів може створювати вразливості в системі;
- Деякі плагіни та теми потребують додаткових витрат.

**Squarespace** (рис. 1.2) - Squarespace відомий своїм зручним інтерфейсом і професійними дизайнерськими шаблонами, що робить його відмінним вибором для користувачів без технічної експертизи. Він пропонує інтуїтивно зрозумілий редактор з функцією перетягування, який спрощує процес створення та налаштування вебсайтів. Squarespace також безперешкодно інтегрується з різними інструментами для маркетингу та продажів, підвищуючи його корисність для малих підприємств, які прагнуть швидко створити онлайн-присутність.

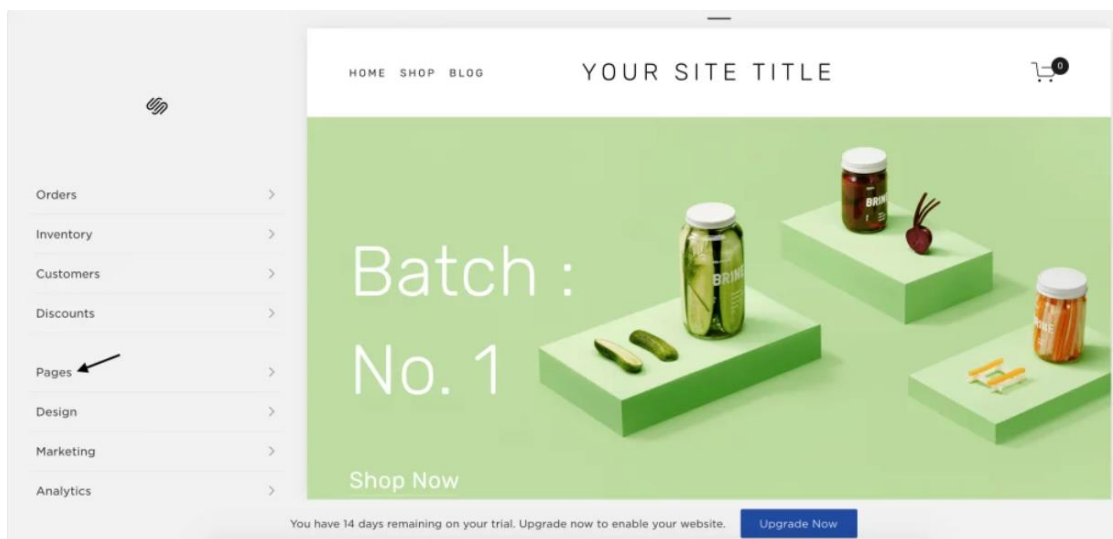


Рис. 1.2 – Інтерфейс Squarespace [2]

#### Переваги:

- Інтуїтивно зрозумілий інтерфейс, що не потребує технічних знань;
- Широкий вибір високоякісних шаблонів;
- Легка інтеграція з різними інструментами для маркетингу та продажів.

#### Недоліки:

- Менше можливостей для розширення функціональності порівняно з WordPress;
- Вищі витрати на підписку порівняно з іншими платформами;
- Можливості інтеграції обмежені екосистемою Squarespace.

**Wix** (рис 1.3) - забезпечує просту та доступну платформу для створення вебсайтів завдяки легкому у використанні редактору з функцією перетягування. Вона особливо підходить для початківців, які хочуть створити функціональний вебсайт без занурення в код. Платформа також пропонує безліч дизайнерських шаблонів і дозволяє значне налаштування.

|  |  |  |  |  |
|--|--|--|--|--|
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

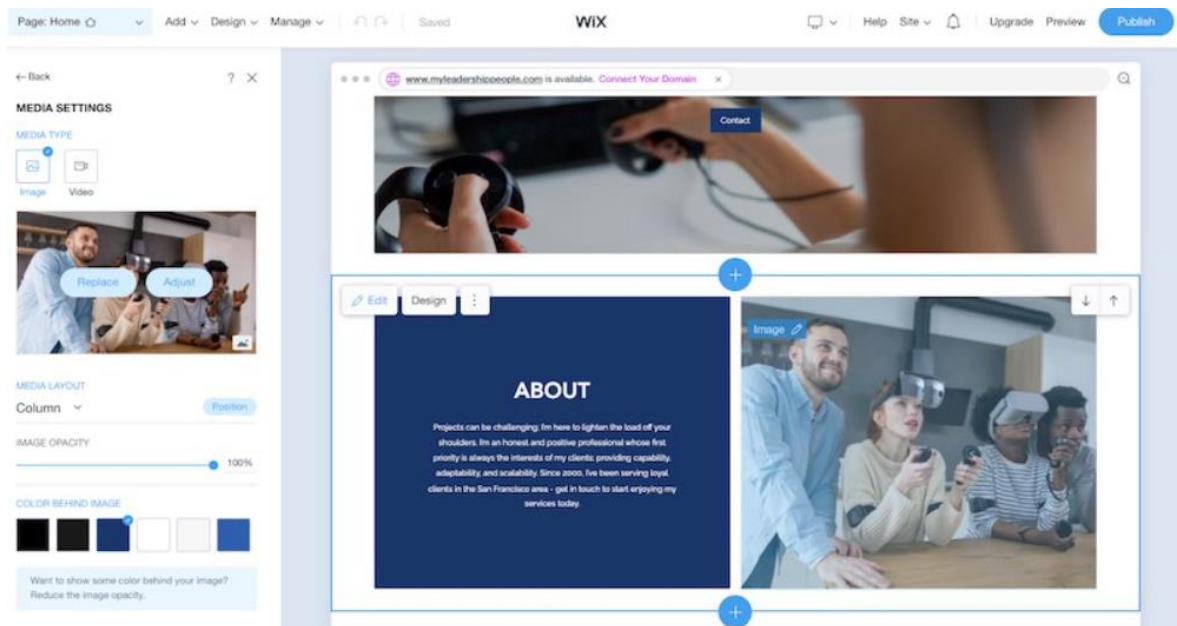


Рис. 1.3 – Інтерфейс Wix [3]

Переваги:

- Простота створення сайту завдяки drag-and-drop редактору;
- Простий інтерфейс;
- Вбудовані інструменти для SEO та маркетингу;
- Інтеграція з штучним інтелектом.

Недоліки:

- Менша гнучкість у порівнянні з WordPress;
- Деякі функції доступні лише у преміум-планах.

Після проведення аналізу аналогів стає очевидним, що кожна платформа має свої сильні і слабкі сторони. WordPress пропонує гнучкість і розширюваність, але потребує технічних знань для налаштування. Squarespace забезпечує простоту використання і професійний дизайн, але має обмежену гнучкість та високу вартість. Інші аналоги, такі як Wix, забезпечують простоту створення сайту, але мають свої обмеження у функціональності та налаштуванні.

Розробка нової платформи для створення веб-магазинів повинна враховувати ці фактори та надавати користувачам оптимальне поєднання гнучкості, простоти використання та доступної вартості.

### 1.3 Вибір архітектури системи

Архітектура додатку для створення веб-магазину буде побудована за патерном MVC (Model-View-Controller) [4], який забезпечує розділення логіки програми на три основні компоненти: модель, вид і контролер. Кожен з цих компонентів виконує окрему функцію, забезпечуючи модульність та полегшуючи підтримку і розширення системи.

Для забезпечення ефективної обробки запитів, всі контролери будуть наслідуватися від одного базового контролера, який надасть захищені методи для обробки HTTP-запитів POST, GET, DELETE [5]. Базовий контролер забезпечить основну функціональність, таку як перевірка авторизації та обробка помилок.

Від базового контролера будуть наслідуватися два контролери, які забезпечуватимуть авторизацію та автентифікацію користувачів двох типів: користувачів магазину та адміністраторів.

Контролери міститимуть захищені методи, які забезпечать обробку дій при неуспішній авторизації. Це дозволить централізовано керувати помилками та знижувати дублікацію коду.

Застосування патерну MVC та побудова контролерів з використанням базового класу забезпечить високу гнучкість та зручність у підтримці додатку. Це дозволить легко розширювати функціональність, додавати нові модулі та забезпечувати безпеку і ефективність обробки запитів.

### 1.4 Обґрунтування вибору інструментальних засобів

При виборі інструментальних засобів для розробки веб-додатку, особливу увагу було приділено платформі Node.js. Цей вибір підкріплюється низкою значущих переваг, що роблять Node.js особливо придатною для цього проекту. Важливими компонентами обраної технологічної стеку є Express, TypeORM, SQL Server, Pug, чистий JavaScript і Passport.js з локальною стратегією. Розглянемо ці інструменти детальніше.

**Node.js** - це середовище виконання JavaScript, яке дозволяє запускати код JavaScript на стороні сервера [6].

Основні переваги використання:

- Node.js працює на подієвій моделі з неблокуючим ввідом/виводом, що дозволяє обробляти велику кількість одночасних з'єднань з високою продуктивністю;
- Однорідність технологій: Використання JavaScript як на клієнтській, так і на серверній стороні дозволяє використовувати одну і ту ж мову для всієї програми, що спрощує процес розробки.

**Express.js** - це легкий та гнучкий веб-фреймворк для Node.js, який значно спрощує процес створення серверних додатків.

Основні переваги Express.js:

- Простота використання: Express.js надає мінімалістичну структуру для створення веб-додатків і API, що дозволяє швидко налаштувати і запустити сервер;
- Масштабованість: Завдяки своїй модульності, Express.js дозволяє легко додавати нові функції та розширювати додаток без значних зусиль;
- Обробка запитів: Express.js забезпечує ефективну обробку HTTP-запитів і підтримує використання середніх шарів для обробки запитів на різних етапах.

**TypeORM** - це ORM (Object-Relational Mapping) для TypeScript та JavaScript, що працює з Node.js і підтримує різні бази даних, включаючи SQL Server [7].

Основні переваги TypeORM:

- Типізація: TypeORM написаний на TypeScript, що забезпечує статичну типізацію і допомагає уникнути багатьох помилок, що виникають під час компіляції [8];
- Підтримка різних баз даних: TypeORM підтримує різні бази даних, такі як MySQL, PostgreSQL, SQLite, SQL Server та інші, що робить його універсальним рішенням для різних проектів;

- Маппінг об'єктів: TypeORM дозволяє зручно маппити об'єкти JavaScript на таблиці бази даних, що спрощує роботу з даними та зменшує кількість коду, необхідного для взаємодії з базою даних.

**SQL Server** — потужна реляційна система управління базами даних, що надає широкий спектр функцій для зберігання та управління даними [9, 10].

Основні переваги:

- Надійність та безпека: SQL Server забезпечує високий рівень безпеки даних та надійність роботи, що є критично важливим для бізнес-додатків;
- Інтеграція з інструментами: SQL Server має потужні інструменти для аналітики, реплікації та резервного копіювання даних, що спрощує управління базою даних;
- Висока продуктивність: Оптимізація запитів та ефективне управління ресурсами дозволяють досягти високої продуктивності при обробці великих обсягів даних.

**Pug** (раніше Jade) - це шаблонізатор для Node.js, який використовується для генерації HTML на серверній стороні.

Основні переваги:

- Простота синтаксису: Pug використовує інтуїтивно зрозумілий та компактний синтаксис, що дозволяє швидко створювати та підтримувати шаблони HTML;
- Інтерактивність: Pug підтримує динамічні конструкції та змінні, що дозволяє створювати складні сторінки з мінімальними зусиллями.

**Passport.js** — це middleware для Node.js, яке забезпечує автентифікацію у веб-додатках [11, 12].

Основні переваги:

- Простота використання: Інтеграція Passport.js з Express.js забезпечує простий та зрозумілий процес налаштування автентифікації;

- Безпека: Використання локальної стратегії дозволяє забезпечити високий рівень безпеки для користувачів, обробляючи автентифікацію безпосередньо на сервері [13-15].

На клієнтській стороні для адміністративної частини буде використовуватися Pug у поєднанні з чистим JavaScript. Для адміністративної частини, де наповнення сторінок відносно просте, використання Pug дозволяє швидко створювати сторінки без зайвого коду, а для клієнтської частини, де необхідне динамічне наповнення, чистий JavaScript забезпечить високу гнучкість та продуктивність, дозволяючи ефективно маніпулювати DOM.

Загальна архітектура додатку буде побудована таким чином, щоб всі компоненти працювали злагоджено і забезпечували необхідну функціональність. Express.js буде використовуватися як основний фреймворк для обробки HTTP-запитів [16] і управління маршрутизацією. TypeORM забезпечить взаємодію з базою даних SQL Server, що дозволить зберігати та обробляти дані. Pug буде використовуватися для створення серверних шаблонів, а чистий JavaScript — для динамічного наповнення клієнтської частини [17]. Passport.js з локальною стратегією забезпечить безпечну автентифікацію користувачів.

### **Висновки до першого розділу**

У першому розділі цієї роботи було визначено основні напрямки та цілі розробки вебдодатку для створення та управління вебмагазинами. Основною метою є створення ефективного інструменту, який дозволяє користувачам швидко налаштовувати тему, тексти та зміст сторінок вебмагазину, а також ефективно управляти своїми товарами та замовленнями. Ретельно плановані етапи аналізу, вибору технологій, розробки архітектури та тестування є критичними для досягнення цих цілей.

Аналіз аналогів показав, що, враховуючи сильні та слабкі сторони існуючих платформ для створення вебмагазинів, таких як WordPress, Squarespace та Wix, нова платформа може запропонувати свої переваги, інтегруючи найкращі практики і функціональні можливості для задоволення

|  |  |  |  |  |                    |      |
|--|--|--|--|--|--------------------|------|
|  |  |  |  |  | ІПЗ.КР.Б-121-24-ПЗ | Арк. |
|  |  |  |  |  |                    | 17   |
|  |  |  |  |  |                    |      |

потреб сучасних підприємців. Вибір архітектури системи спрощує розробку та дозволяє забезпечити високу гнучкість та масштабованість продукту.

Застосування платформи Node.js у комбінації з Express, TypeORM, SQL Server, Pug, чистим JavaScript та Passport.js для розробки веб-додатку виправдається завдяки їх потужним можливостям.

Таким чином, підготовка та планування, проведені у цьому розділі, закладають міцний фундамент для подальшої розробки веб-додатку, що стане надійним інструментом для підприємців. Цей додаток дасть гідну відповідь вимогам сучасних технологічних та бізнесових викликів.

## РОЗДІЛ 2. ПРОЄКТУВАННЯ ПРОГРАМИ

### 2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

Для чіткого визначення використання веб-додатку і досягнення найкращих результатів, наведемо перелік основних вимог до системи. Ці вимоги допоможуть встановити основні напрямки взаємодії з користувачами та дозволять спланувати роботу над проектом. Виконання всіх вимог забезпечить стабільну роботу додатку та успішне виконання всіх функцій.

#### Функціональні вимоги

- Створення та редагування сторінок: адміністратори повинні мати можливість легко створювати нові сторінки веб-магазину, вказуючи унікальний URL та редагуючи їхній зміст. Це дозволяє адміністратору гнучко управляти вмістом магазину та швидко адаптувати його під потреби бізнесу;
- Наявність шаблонів: адміністратори повинні мати набір базових сторінок сайту, які можна редагувати та які складають невід'ємну частину будь-якого вебмагазину, а також базовий шаблон для кожної з таких сторінок;
- Редагування теми: адміністратори повинні мати зручний інтерфейс для редагування графічного змісту сторінок, а саме кольорів верхньої та нижньої панелі сайту і окремо для всіх елементів сторінки;
- Редагування текстів: адміністратори повинні мати зручний інтерфейс для редагування текстової складової сайту, а саме всіх текстів, які бачить звичайний користувач;
- Редагування сайту для мобільного розширення: адміністратори повинні мати зручний інтерфейс для редагування змісту сторінок сайту на мобільних пристроях;
- Управління товарами: адміністратори повинні мати можливість додавати, редагувати та видаляти товари, вказуючи їхні характеристики та ціни;

|  |  |  |  |  |                    |      |
|--|--|--|--|--|--------------------|------|
|  |  |  |  |  | ІПЗ.КР.Б-121-24-ПЗ | Арк. |
|  |  |  |  |  |                    | 19   |
|  |  |  |  |  |                    |      |

- Перегляд замовлень: адміністратори повинні мати можливість переглядати замовлення та бачити повну інформацію про них, включаючи ціни, які бачив користувач при замовленні. Це забезпечить контроль за виконанням замовлень;
- Реєстрація та авторизація: користувачі повинні мати можливість реєстрації у веб-магазині та авторизації для доступу до особистого кабінету. Це дозволить користувачам додавати товари в кошик та замовляти їх;
- Перегляд товарів: користувачі повинні мати можливість переглядати каталог товарів, фільтрувати їх за різними критеріями та переглядати деталі кожного товару. Це сприятиме зручному пошуку та вибору продуктів;
- Додавання товарів до кошика: користувачі повинні мати можливість додавати товари до кошика для подальшого оформлення замовлення. Це дозволить користувачам зручно формувати свої покупки.

### **Нефункціональні вимоги**

- Час відгуку: час відгуку для звичайних завдань повинен бути не більше 3 секунд, а для складніших - не більше 10 секунд. Це забезпечить швидку реакцію системи та зручність користування;
- Допустима кількість одночасних користувачів: система повинна підтримувати мінімум 200 одночасно працюючих користувачів, які взаємодіють із базою даних магазину. Це забезпечить стабільну роботу при високому навантаженні;
- Масштабування: система має бути масштабованою, щоб була можливість збільшити продуктивність та забезпечити зручну роботу зі збільшенням кількості користувачів.
- Це дозволить адаптувати систему під зростаючі потреби бізнесу.

### **Користувачі**

Основними користувачами системи є адміністратори та кінцеві користувачі (покупці). Адміністратори забезпечують управління та

|  |  |  |  |  |                    |      |
|--|--|--|--|--|--------------------|------|
|  |  |  |  |  | ІПЗ.КР.Б-121-24-ПЗ | Арк. |
|  |  |  |  |  |                    | 20   |
|  |  |  |  |  |                    |      |

наповнення сайту, а користувачі користуються функціями магазину для здійснення покупок. Ці вимоги є основою для подальшої деталізації та розробки вебдодатку, який забезпечить зручний та ефективний інструмент для управління вебмагазинами.

### Варіанти використання:

Система надає вебінтерфейс, через який користувачі можуть створювати онлайн-магазини, управляти продуктами, замовленнями та взаємодіяти з покупцями. Функціональність системи залежить від активності користувачів магазину та адміністраторів. Для наглядної взаємодії між елементами додатку було створено Діаграму варіантів використання (рис 2.1).

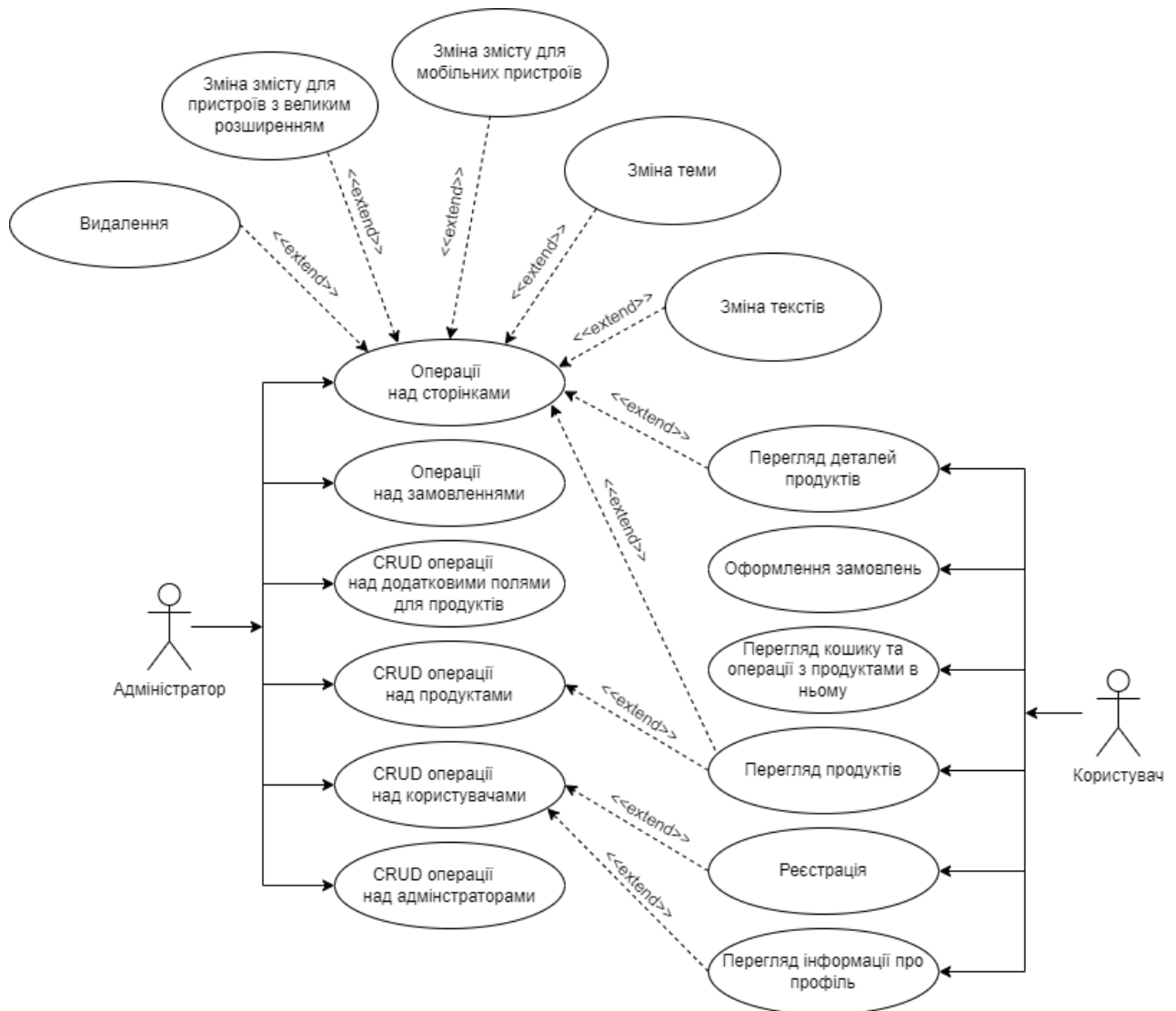


Рис. 2.1 – Діаграма варіантів використання

Опишемо прецеденти вказані в діаграмі варіантів використання в таблицях 2.1 – 2.10.

Таблиця 2.1

Опис прецеденту «Реєстрація»

| Варіант використання | Реєстрація                                                                                                                                                                                                                         |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Основні актори       | Користувач                                                                                                                                                                                                                         |
| Опис                 | Користувачі можуть створювати нові облікові записи, вводючи свої дані, такі як ім'я, електронна адреса та пароль. Для вже зареєстрованих користувачів передбачена можливість входу в систему через введення своїх облікових даних. |

Таблиця 2.2

Опис прецеденту «Додавання продукту»

| Варіант використання | Додавання продукту                                                                                                                                                                                                      |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Основні актори       | Адміністратор                                                                                                                                                                                                           |
| Опис                 | Цей варіант використання включає процес додавання нових продуктів до каталогу магазину. Користувачі можуть вводити детальну інформацію про продукт, таку як назва, опис, ціна, а також завантажити фотографію продукту. |

Таблиця 2.3

Опис прецеденту «Операції над замовленнями»

| Варіант використання | Операції над замовленнями                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Основні актори       | Адміністратор                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Опис                 | У цьому варіанті використання адміністратор має змогу керувати замовленнями, які надходять до магазину. Процес включає перегляд деталей замовлення, таких як перелік замовлених продуктів, контактна інформація покупця, тобто інформація, яку бачив користувач під час замовлення. Адміністратори мають контролювати загальний процес обробки замовлень. Тобто відповідальність на їх обробку, така як відправка листа на пошту, повністю припадає на нього. |

Таблиця 2.4

Опис прецеденту «Перегляд кошику та операції з продуктами в ньому»

| Варіант використання | Перегляд кошику та операції з продуктами в ньому                                                                                                                                                                                                                              |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Основні актори       | Користувач                                                                                                                                                                                                                                                                    |
| Опис                 | Цей варіант використання дозволяє користувачам переглядати всі продукти, додані до їхнього кошика. Користувачі можуть бачити назву, ціну продукту, а також загальну суму замовлення. Вони можуть очистити кошик. Система оновлює загальну суму замовлення з урахуванням змін. |

Таблиця 2.5

Опис прецеденту «Оформлення замовлень»

| Варіант використання | Оформлення замовлень                                                                                                                                                              |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Основні актори       | Користувач                                                                                                                                                                        |
| Опис                 | Цей варіант використання передбачає процес створення замовлення. Після успішного оформлення користувач має отримати лист на електронну пошту як тільки замовлення буде оброблено. |

Таблиця 2.6

Опис прецеденту «CRUD-операції над користувачами»

| Варіант використання | CRUD-операції над користувачами                                                                                                                                                       |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Основні актори       | Адміністратор                                                                                                                                                                         |
| Опис                 | Цей варіант використання включає управління обліковими записами користувачів. Адміністратори можуть переглядати список зареєстрованих користувачів, інформацію про них та редагувати. |

Таблиця 2.7

Опис прецеденту «Зміна текстів»

| Варіант використання | Зміна текстів                                |
|----------------------|----------------------------------------------|
| Основні актори       | Адміністратор                                |
| Опис                 | Передбачає зміну текстів видних користувачу. |

Таблиця 2.8

## Опис прецеденту «Зміна теми»

| Варіант використання | Зміна теми                                                                                                                                                                                                                                                          |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Основні актори       | Адміністратор                                                                                                                                                                                                                                                       |
| Опис                 | Цей варіант використання дозволяє адміністраторам змінювати візуальну тему сайту для покращення користувацького досвіду користувачів. Адміністратори можуть налаштовувати власну тему, змінюючи кольорові схеми, границі, фон, відступи та інші візуальні елементи. |

Таблиця 2.9

## Опис прецеденту «Зміна змісту для пристроїв з великим розширенням»

| Варіант використання | Зміна змісту для пристроїв з великим розширенням                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Основні актори       | Адміністратор                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Опис                 | Цей варіант використання дозволяє адміністраторам забезпечити оптимальне відображення вмісту на пристроях з великим екраном, таких як настільні комп'ютери та великі монітори. Адміністратори можуть адаптувати інтервали між елементами для кращої читабельності та зручності навігації. Це включає використання адаптивного дизайну, який автоматично підлаштовується під розміри екрану, забезпечуючи зручність використання на великих дисплеях. |

Таблиця 2.10

## Опис прецеденту «Зміна змісту для мобільних пристроїв»

| Варіант використання | Зміна змісту для мобільних пристроїв                                                                                                                                                                                                                         |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Основні актори       | Адміністратор                                                                                                                                                                                                                                                |
| Опис                 | Цей варіант використання дозволяє адміністраторам забезпечити оптимальне відображення змісту на мобільних пристроях. Адміністратори можуть налаштовувати макет сторінок та змінювати інтервали між елементами для кращої читабельності на маленьких екранах. |

## Об'єктно-орієнтована структура:

Аналіз функціональних вимог дозволив виділити наступні сутності, що забезпечать програмну реалізацію додатку. На рисунку 2.2 наведено Діаграму класів, яка використовуються для його роботи.

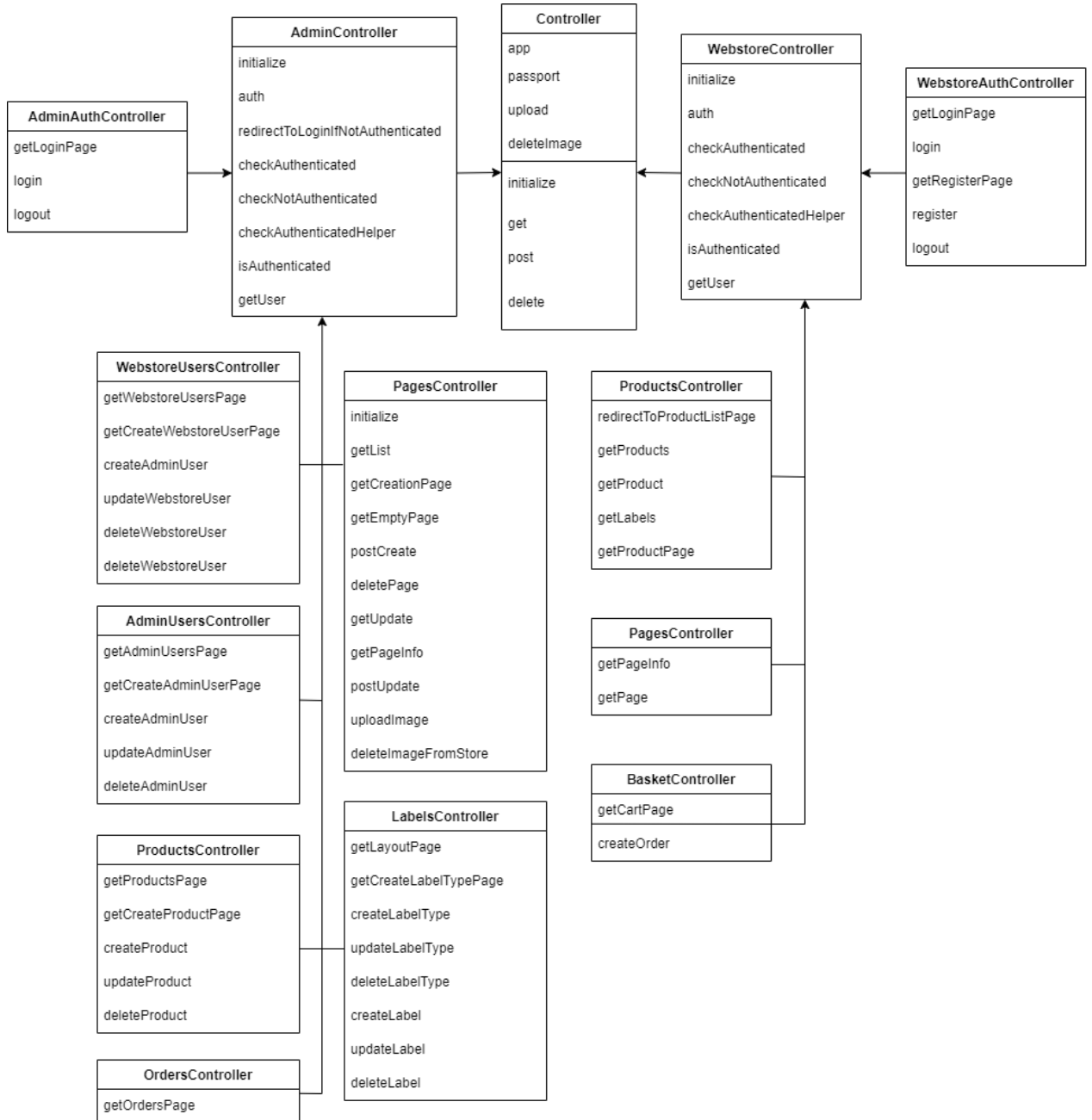


Рис. 2.2 – Діаграма класів

Класи:

### 1. Controller.

1.1. app - зберігає екземпляр додатку Express.

1.2. passport - зберігає екземпляр Passport для аутентифікації.

|  |  |  |  |  |
|--|--|--|--|--|
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

- 1.3. upload - зберігає налаштування для завантаження файлів за допомогою multer.
- 1.4. deleteImage - функція для видалення зображень з файлової системи.
- 1.5. initialize(app, passport, multer, path, \_\_dirname, fs) - ініціалізує додаток з налаштуванням Passport, multer для завантаження файлів і функцією видалення зображень.
- 1.6. get(route: string, ...middleware) - додає маршрут GET до додатку з вказаними middleware.
- 1.7. post(route: string, ...middleware) - додає маршрут POST до додатку з вказаними middleware.
- 1.8. delete(route: string, ...middleware) - додає маршрут DELETE до додатку з вказаними middleware.
2. WebstoreController (наслідує Controller).
  - 2.1. initialize(app) - ініціалізує Passport для користувачів вебмагазину з налаштуванням локальної стратегії.
  - 2.2. auth(req, res, next) - аутентифікує користувача за допомогою локальної стратегії.
  - 2.3. checkAuthenticated(failureRedirect) - перевіряє, чи аутентифікований користувач; перенаправляє на вказану сторінку, якщо ні.
  - 2.4. checkNotAuthenticated(failureRedirect) - перевіряє, чи не аутентифікований користувач; перенаправляє на вказану сторінку, якщо так.
  - 2.5. checkAuthenticatedHelper(isAuthenticated, failureRedirect) - допоміжний метод для перевірки аутентифікації.
  - 2.6. isAuthenticated(req) - перевіряє, чи аутентифікований користувач у поточному запиті.
  - 2.7. getUser(req) - повертає інформацію про поточного користувача без поля password.

3. WebstoreAuthController (наслідує WebstoreController).
  - 3.1. getLoginPage(req, res) - відображає сторінку входу для вебмагазину.
  - 3.2. login(req, res) - обробляє запит на вхід користувача.
  - 3.3. getRegisterPage(req, res) - відображає сторінку реєстрації для вебмагазину.
  - 3.4. register(req, res) - обробляє запит на реєстрацію нового користувача.
  - 3.5. logout(req, res) - обробляє запит на вихід користувача з системи.
4. ProductsController (наслідує WebstoreController).
  - 4.1. redirectToProductListPage(req, res) - перенаправляє на сторінку списку продуктів.
  - 4.2. getProducts(req, res) - отримує список продуктів, враховуючи фільтри та пошукові запити.
  - 4.3. getProduct(req, res) - отримує деталі продукту за його ідентифікатором.
  - 4.4. getLabels(req, res) - отримує список міток продуктів.
  - 4.5. getProductPage(req, res) - відображає сторінку продукту.
5. PagesController (наслідує WebstoreController).
  - 5.1. getPageInfo(req, res) - отримує інформацію про сторінку за її URL.
  - 5.2. getPage(req, res) - відображає сторінку за її URL.
6. BasketController (наслідує WebstoreController).
  - 6.1. getCartPage(req, res) - відображає сторінку кошика покупок.
  - 6.2. createOrder(req, res) - створює нове замовлення на основі продуктів у кошику.
7. WebstoreUsersController (наслідує AdminController).
  - 7.1. getWebstoreUsersPage(req, res) - відображає сторінку з користувачами вебмагазину.
  - 7.2. getCreateWebstoreUserPage(req, res) - відображає сторінку створення нового користувача вебмагазину.

- 7.3. createAdminUser(req, res) - створює нового адміністратора.
- 7.4. updateWebstoreUser(req, res) - оновлює інформацію про користувача вебмагазину.
- 7.5. deleteWebstoreUser(req, res) - видаляє користувача вебмагазину.
8. ProductsController (наслідує AdminController).
  - 8.1. getProductsPage(req, res) - відображає сторінку з продуктами.
  - 8.2. getCreateProductPage(req, res) - відображає сторінку створення нового продукту.
  - 8.3. createProduct(req, res) - Створює новий продукт.
  - 8.4. updateProduct(req, res) - Оновлює інформацію про продукт.
  - 8.5. deleteProduct(req, res) - Видаляє продукт.
9. PagesController (наслідує AdminController).
  - 9.1. initialize() - ініціалізує сторінки, створюючи базові сторінки, якщо вони не існують.
  - 9.2. getList(req, res) - відображає сторінку зі списком сторінок.
  - 9.3. getCreationPage(req, res) - відображає сторінку створення нової сторінки.
  - 9.4. getEmptyPage(req, res) - відображає порожню сторінку.
  - 9.5. postCreate(req, res) - обробляє запит на створення нової сторінки.
  - 9.6. deletePage(req, res) - видаляє сторінку.
  - 9.7. getUpdate(req, res) - відображає сторінку оновлення сторінки.
  - 9.8. getPageInfo(req, res) - отримує інформацію про сторінку за її ідентифікатором.
  - 9.9. postUpdate(req, res) - оновлює сторінку.
  - 9.10. uploadImage(req, res) - обробляє завантаження зображень.
  - 9.11. deleteImageFromStore(req, res) - видаляє зображення з файлової системи.
10. OrdersController (наслідує AdminController).
  - 10.1. getOrdersPage(req, res) - відображає сторінку з замовленнями.
11. LabelsController (наслідує AdminController).

- 11.1. getLayoutPage(req, res) - відображає сторінку з мітками.
- 11.2. getCreateLabelTypePage(req, res) - відображає сторінку створення нового типу мітки.
- 11.3. createLabelType(req, res) - створює новий тип мітки.
- 11.4. updateLabelType(req, res) - оновлює інформацію про тип мітки.
- 11.5. deleteLabelType(req, res) - видаляє тип мітки.
- 11.6. createLabel(req, res) - створює нову мітку.
- 11.7. updateLabel(req, res) - оновлює інформацію про мітку.
- 11.8. deleteLabel(req, res) - видаляє мітку.

12. AdminUsersController (наслідує AdminController).

- 12.1. getAdminUsersPage(req, res) - відображає сторінку з адміністраторами.
- 12.2. getCreateAdminUserPage(req, res) - відображає сторінку створення нового адміністратора.
- 12.3. createAdminUser(req, res) - створює нового адміністратора.
- 12.4. updateAdminUser(req, res) - оновлює інформацію про адміністратора.
- 12.5. deleteAdminUser(req, res) - видаляє адміністратора.

13. AdminController (наслідує Controller).

- 13.1. initialize(app) - ініціалізує Passport для адміністратора.
- 13.2. initializePassport() - налаштовує Passport для адміністраторами.
- 13.3. auth(req, res, next) - аутентифікує адміністратора за допомогою локальної стратегії.
- 13.4. redirectToLoginIfNotAuthenticated - перевіряє, чи аутентифікований користувач; перенаправляє на сторінку входу, якщо ні.
- 13.5. checkAuthenticated(failureRedirect) - перевіряє, чи аутентифікований користувач; перенаправляє на вказану сторінку, якщо ні.

13.6. checkNotAuthenticated(failureRedirect) - перевіряє, чи не аутентифікований користувач; перенаправляє на вказану сторінку, якщо так.

13.7. checkAuthenticatedHelper(isAuthenticated, failureRedirect) - допоміжний метод для перевірки аутентифікації.

13.8. isAuthenticated(req) - перевіряє, чи аутентифікований адміністратор у поточному запиті.

13.9. getUser(req) - повертає інформацію про поточного адміністратора без пароля.

14. AdminAuthController (наслідує AdminController).

14.1. getLoginPage(req, res) - відображає сторінку входу для адміністратора.

14.2. login(req, res) - обробляє запит на вхід адміністратора.

14.3. logout(req, res) - обробляє запит на вихід адміністратора з системи.

## 2.2 Розробка бази даних

Для збереження даних сайту використовується SQL Server, це система управління базами даних, яка базується на реляційній моделі. База буде містити 10 таблиць для тем, користувачів магазину, замовлень, текстів, сторінок, продуктів, міток продуктів, міток, типів міток та адміністраторів. Розглянемо Схему бази даних (рис. 2.3), а також опишемо кожну з таблиць (таблиця 2.11 - 2.20).

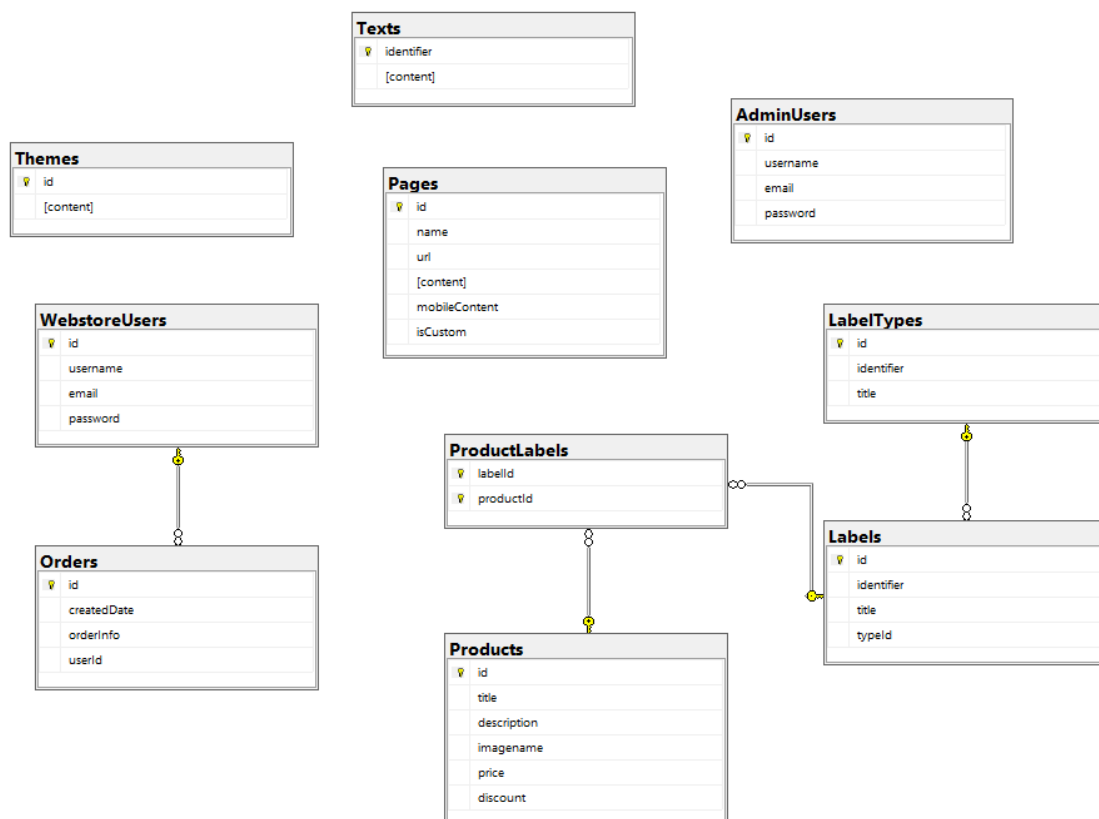


Рис. 2.3 – Схема бази даних

Таблиця 2.11

Опис таблиці Texts

| Поле    | Тип           | Призначення        |
|---------|---------------|--------------------|
| id      | nvarchar(255) | Ключ групи текстів |
| content | nvarchar(max) | Дані текстів       |

Таблиця 2.12

Опис таблиці Pages

| Поле          | Тип           | Призначення                    |
|---------------|---------------|--------------------------------|
| id            | int           | Ідентифікатор                  |
| name          | nvarchar(255) | Назва                          |
| url           | nvarchar(255) | URL-ідентифікатор              |
| content       | nvarchar(max) | Зміст сторінки                 |
| mobileContent | nvarchar(max) | Зміст для мобільних девайсів   |
| isCustom      | bit           | Визначає чи є сторінка базовою |

|  |  |  |  |  |
|--|--|--|--|--|
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Таблиця 2.13

Опис таблиці Themes

| Поле    | Тип           | Призначення   |
|---------|---------------|---------------|
| id      | int           | Ідентифікатор |
| content | nvarchar(max) | Дані теми     |

Таблиця 2.14

Опис таблиці WebstoreUsers

| Поле     | Тип           | Призначення      |
|----------|---------------|------------------|
| id       | int           | Ідентифікатор    |
| username | nvarchar(255) | Ім'я користувача |
| email    | nvarchar(255) | Пошта            |
| password | nvarchar(255) | Пароль           |

Таблиця 2.15

Опис таблиці Orders

| Поле        | Тип           | Призначення               |
|-------------|---------------|---------------------------|
| id          | int           | Ідентифікатор             |
| createdDate | datetime2(7)  | Дата створення            |
| orderInfo   | nvarchar(max) | Дані замовлення           |
| userId      | int           | Ідентифікатор користувача |

Таблиця 2.16

Опис таблиці ProductLabels

| Поле      | Тип | Призначення            |
|-----------|-----|------------------------|
| labelId   | int | Ідентифікатор мітки    |
| productId | int | Ідентифікатор продукту |

Таблиця 2.17

Опис таблиці AdminUsers

| Поле     | Тип           | Призначення   |
|----------|---------------|---------------|
| id       | int           | Ідентифікатор |
| username | nvarchar(255) | Ім'я          |
| email    | nvarchar(255) | Пошта         |
| password | nvarchar(255) | Пароль        |

Таблиця 2.18

Опис таблиці Products

| Поле        | Тип           | Призначення     |
|-------------|---------------|-----------------|
| id          | int           | Ідентифікатор   |
| title       | nvarchar(255) | Дані текстів    |
| description | nvarchar(max) | Опис            |
| imagename   | nvarchar(255) | Назва картинки  |
| price       | float         | Ціна            |
| discount    | tinyint       | Відсоток знижки |

Таблиця 2.19

Опис таблиці LabelTypes

| Поле       | Тип           | Призначення       |
|------------|---------------|-------------------|
| id         | int           | Ідентифікатор     |
| identifier | nvarchar(255) | URL-ідентифікатор |
| title      | nvarchar(255) | Назва             |

Таблиця 2.20

Опис таблиці Labels

| Поле       | Тип           | Призначення              |
|------------|---------------|--------------------------|
| id         | int           | Ідентифікатор            |
| identifier | nvarchar(255) | URL-ідентифікатор        |
| title      | nvarchar(255) | Назва                    |
| typeId     | int           | Ідентифікатор типу мітки |

Ця структура бази даних дозволяє ефективно керувати даними про користувачів, продукти, замовлення та мітки продуктів у веб-магазині. Кожна таблиця має чітко визначені поля та зв'язки між ними, що забезпечує цілісність даних.

### 2.3 Проектування та реалізація алгоритмів роботи системи

Алгоритм роботи з сайтом вебмагазину включає кілька основних етапів, одним з яких є процес реєстрації та входу користувача, і важливо якомога детальніше описати алгоритм, адже це найскладніший алгоритм з яким взаємодіє користувач магазину. Саме після входу в обліковий запис

користувач отримає доступ до додавання товарів в кошик та оформлення замовлень. Нижче описано деталі цього процесу, а також обробку можливих помилок.

Процес реєстрації та входу:

1. Реєстрація користувача.

1.1. Користувач вводить необхідні дані для реєстрації (ім'я користувача, електронна пошта, пароль);

1.2. Система перевіряє валідність введених даних.

1.3. Якщо дані невалідні, система виводить повідомлення про помилку та причину невалідності.

1.4. Якщо дані валідні, система створює новий обліковий запис для користувача.

2. Вхід до системи.

2.1. Користувач вводить ім'я користувача та пароль.

2.2. Система перевіряє валідність введених даних.

2.3. Якщо дані невалідні, система виводить повідомлення про помилку та причину невалідності.

2.4. Якщо дані валідні, користувач успішно входить до системи.

Для опису алгоритму роботи наведемо Діаграму активності (рис. 2.4), на ній зображено основний цикл, який проходить користувач, у випадку якщо йому потрібно здійснити вхід у обліковий запис.

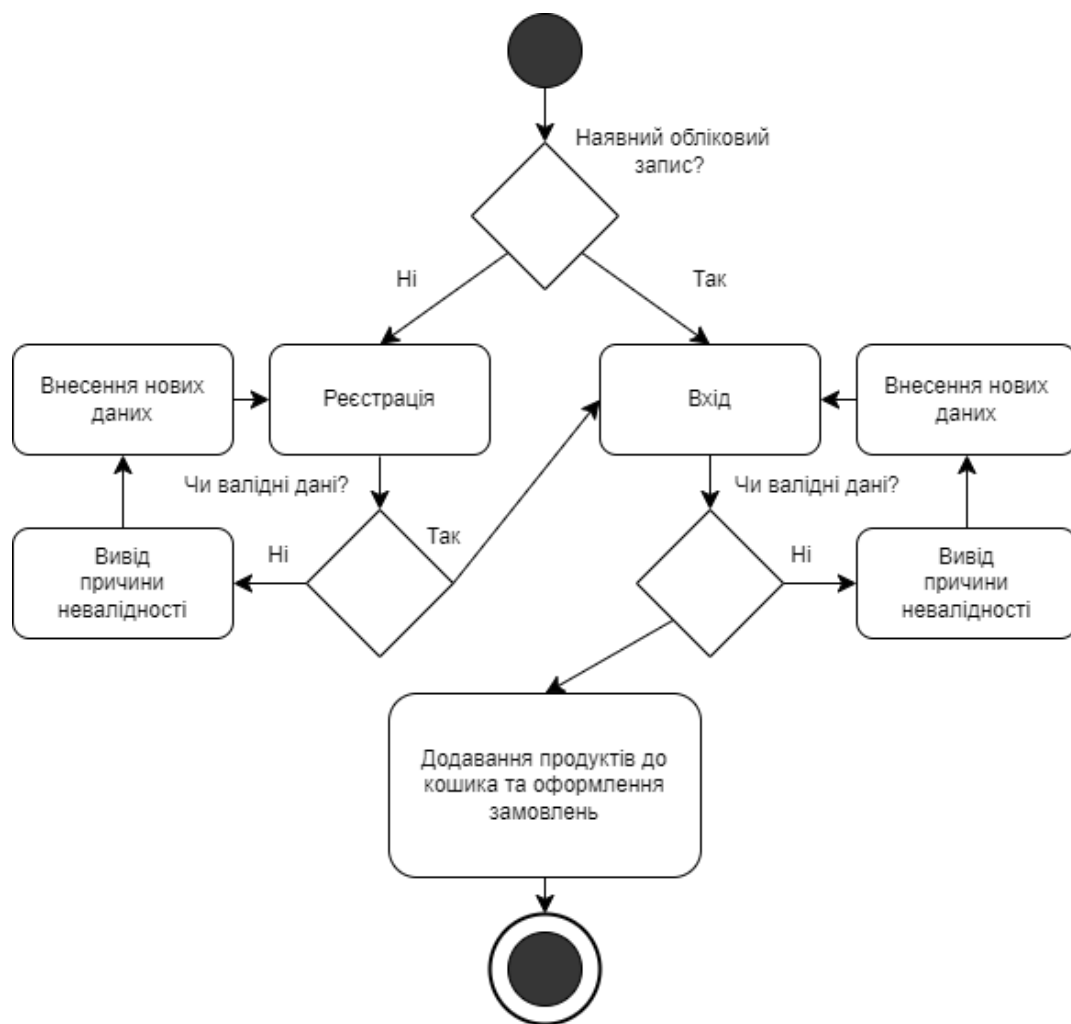


Рис. 2.4 – Діаграма активності

Цей алгоритм забезпечує користувачу інтуїтивно зрозумілий процес взаємодії з вебмагазином, враховуючи можливі помилки та способи їх виправлення.

## 2.4 Реалізація функціоналу програми для створення вебмагазинів

Оскільки основна частина адмін-панелі буде написана на шаблонізаторі pug, який надає зручну роботу з формами та гарну структурування html – розмітки, але не надає потрібних в проєкті інструментів для оперування динамічним контентом, потрібно написати набір функцій, що буде вирішувати цю проблему.

Коли потрібно мати справу з динамічним перемальовуванням сторінки, очевидним методом рішення є використання мови JavaScript, адже в ній є можливості для створення HTML-елементів та їх інтеграцію в уже існуючу сторінку. Для цього спочатку буде створена функція, яка прийматиме назву

тега елемента, який потрібно створити. Після виконання цієї функції має повернутися інша функція, яка вже виконуватиме роботу тега, тобто прийматиме атрибути та властивості елемента. В свою чергу, після визначення атрибутів та властивостей отримаємо вже наступну функцію і при її виклику повернеться готовий HTML-елемент. При цьому в неї буде надано можливість передавання дочірніх елементів, щоб створити повноцінне HTML-дерево (рис. 2.5).

```
const createElement = (tagName, { ...initProps } = {}) => {
  return ({ attributes, properties, ...props } = {}) => {
    return (...children) => {
      let elem = document.createElement(tagName);

      [initProps, props].forEach(props =>
        Object.entries(props)
          .forEach(([key, value]) => elem[key] = value)
      );

      attributes && Object.entries(attributes)
        .forEach(([key, value]) => elem.setAttribute(key, value));

      properties && Object.entries(properties)
        .forEach(([key, value]) => elem.style.setProperty(key, value));

      if (children)
        children.forEach(child => elem.append(child));

      return elem;
    }
  }
}
```

Рис. 2.5 – Створення елемента

Після створення функції для створення елемента необхідно викликати її для всіх основних тегів, які будуть використовуватися в редакторі сторінок, і для простих тегів це буде виконано без передачі параметрів для ініціалізації, а для тегів, які пов'язані з введенням даних користувачем, наприклад, в кнопках, поля для введення файлу та текстових полях, буде викликатися з передачею типу (рис. 2.6).

```

export const button = createElement('button');
export const a = createElement('a');
export const div = createElement('div');
export const p = createElement('p');
export const header = createElement('header');
export const footer = createElement('footer');
export const nav = createElement('nav');
export const aside = createElement('aside');
export const section = createElement('section');
export const main = createElement('main');
export const span = createElement('span');
export const script = createElement('script');
export const label = createElement('label');
export const table = createElement('table');
export const tr = createElement('tr');
export const td = createElement('td');
export const select = createElement('select');
export const option = createElement('option');
export const img = createElement('img');
export const input = createElement('input');
export const radio = createElement('input', { type: 'radio' });
export const checkbox = createElement('input', { type: 'checkbox' });
export const textInput = createElement('input', { type: 'text' });
export const buttonInput = createElement('input', { type: 'button' });
export const numberInput = createElement('input', { type: 'number' });
export const passwordInput = createElement('input', { type: 'password' });
export const range = createElement('input', { type: 'range' });
export const textarea = createElement('textarea');
export const h1 = createElement('h1');
export const h2 = createElement('h2');
export const h3 = createElement('h3');
export const h4 = createElement('h4');
export const h5 = createElement('h5');
export const h6 = createElement('h6');
export const ul = createElement('ul');
export const li = createElement('li');
export const small = createElement('small');
export const strong = createElement('strong');
export const form = createElement('form');
export const row = (props) => (...children) =>
  |   table()(tr(props)(...children));

```

Рис. 2.6 – Створення тегів

Далі потрібно буде створити функцію, яка здійснюватиме створення компонента [18-30]. Основною задачею компонента буде операції зі станом, від якого залежить вигляд сайту, і він буде частиною головного стану (рис. 2.7).

```
function createGeneralState() {
  const generalState = {};

  return {
    getState: id => {
      if (!generalState[id])
        generalState[id] = {};
      return generalState[id];
    },
  };
};
```

Рис. 2.7 – Створення головного стану

Тобто цей компонент має оновлювати вигляд сторінки коли його стан змінився. Для цього за допомогою замикання буде створено набір функції, які потім можна буде використати для контролю над динамічним перемальовуванням розмітки.

Перші дві з цих функцій будуть відповідати за побочні ефекти перемальовування. За допомогою них можна буде зареєструвати функції зворотнього виклику, які виконуються до перемалювання або після (рис. 2.8).

```
const [useAfterRender, useBeforeRender] =
  ['afterRender', 'beforeRender'].map(effect => {
    state[effect] = [];
    return (func) => state[effect].push(func);
  });
```

Рис. 2.8 – Функції для побочних ефектів

Також є необхідність у створення залежності віддаленого дерева від стану цього компоненту. Як приклад, вигляд бокової панелі має залежати від того, що відбувається в редакторі сторінки, незважаючи на те, що вони не являються батьками один по відношенню до одного (рис. 2.9).

```

state.portals = [];

const openPortal = (element, contentFunc) => {
  element.innerHTML = '';
  element.append(contentFunc())
  state.portals.push({ element, contentFunc });
};

```

Рис. 2.9 – Функція для створення залежності віддаленого дерева

Неможливо буде уникнути написання функції для отримання стану компоненту та його оновлення, адже саме через цю причину він і створюється. Всередині компонента її можна буде викликати безліч разів, а для цього потрібно буде створити унікальний ідентифікатор стану, до якого відноситься виклик (рис. 2.10).

```

const useState = (initValue) => {
  const key = newStateIndex++;
  state[key] = initValue;

  return [
    () => state[key],
    (value) => {
      state[key] = value;
      state.render?.();
    }
  ];
};

```

Рис. 2.10 – Функція для керування станом

І наостанок залишається функція, яка буде викликатися на оновлення стану та перемальовувати залежну розмітку. У ній також викликатимуться функції-ефекти, будуть перемальовуватися залежні віддалені дерева. І створиться вона у функції, яка ініціалізує основне залежне дерево (рис. 2.11).

```

const render = (contentFunc) => {
  const wrapper = contentFunc();
  wrapper.innerHTML = '';

  state.render = () => {
    state.beforeRender.forEach(f => f());

    state.portals.forEach(({ element, contentFunc }) => {
      element.innerHTML = '';
      element.append(contentFunc())
    });

    wrapper.innerHTML = '';
    wrapper.append(...contentFunc().childNodes);

    state.afterRender.forEach(f => f());
  };

  state.beforeRender.forEach(f => f());
  wrapper.append(...contentFunc().childNodes);
  state.afterRender.forEach(f => f());

  return wrapper;
}

```

Рис. 2.11 – Функція для перемальовування

## Висновки до другого розділу

Під час розробки програми для конструювання вебмагазинів було враховано всі ключові аспекти, які забезпечують його стабільну та ефективну роботу. Проект включає реалізацію функціональних вимог, таких як реєстрація та авторизація користувачів, управління продуктами та замовленнями, зміна змісту, а також адаптацію змісту для різних пристроїв.

Архітектура системи була ретельно спланована для забезпечення максимальної гнучкості та масштабованості. Завдяки використанню сучасних технологій та інструментів адміністратори можуть легко створювати нові сторінки, редагувати вміст, змінювати теми та керувати товарами. Користувачі, в свою чергу, отримують зручний інтерфейс для перегляду товарів, додавання їх до кошика та оформлення замовлень.

Усі етапи розробки були орієнтовані на створення інтуїтивно зрозумілого та функціонально насиченого додатку, що відповідає потребам як адміністратора, так і користувача магазину. Результатом є стабільна та

ефективна платформа, готова до масштабування та подальшого розвитку, яка забезпечує зручність у використанні та задовольняє вимоги сучасного вебмагазину.

## РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПРОГРАМОЮ

### 3.1 Порядок встановлення та налаштування параметрів платформи

В системі наявний один вузол розгортання на Node.js, який розгортається на операційній системі Windows 10.

Для опису архітектури платформи було створено діаграму розгортання (рис. 3.1).

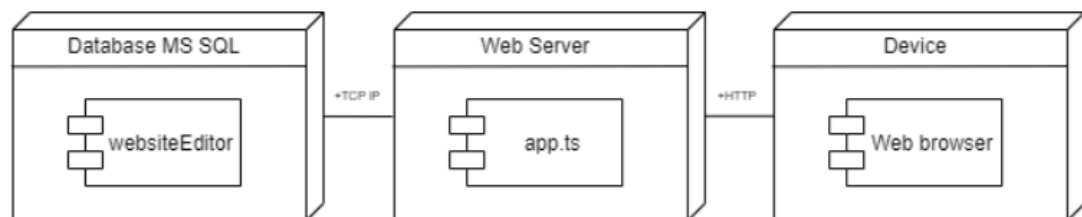


Рис. 3.1 – Діаграма розгортання

Налаштування веб-сервера (Web Server):

1. Встановлення веб-сервера: веб-сервер може бути налаштований на будь-якій платформі (наприклад, Linux, Windows) з використанням відповідного програмного забезпечення, такого як Node.js.
2. Розгортання додатку: на веб-сервері розгортається додаток, написаний на TypeScript (файл app.ts). Цей додаток є центральним компонентом, який обробляє запити від клієнтів та взаємодіє з базою даних.
3. Налаштування середовища виконання: встановлюються всі необхідні залежності та бібліотеки для додатку через пакетний менеджер (наприклад, npm для Node.js). Конфігуруються змінні середовища, такі як налаштування підключення до бази даних.
4. Запуск додатку: запускається додаток на веб-сервері, забезпечуючи доступність через HTTP протокол.

Для більш точного опису архітектури платформи було створено діаграму компонентів (рис. 3.2).

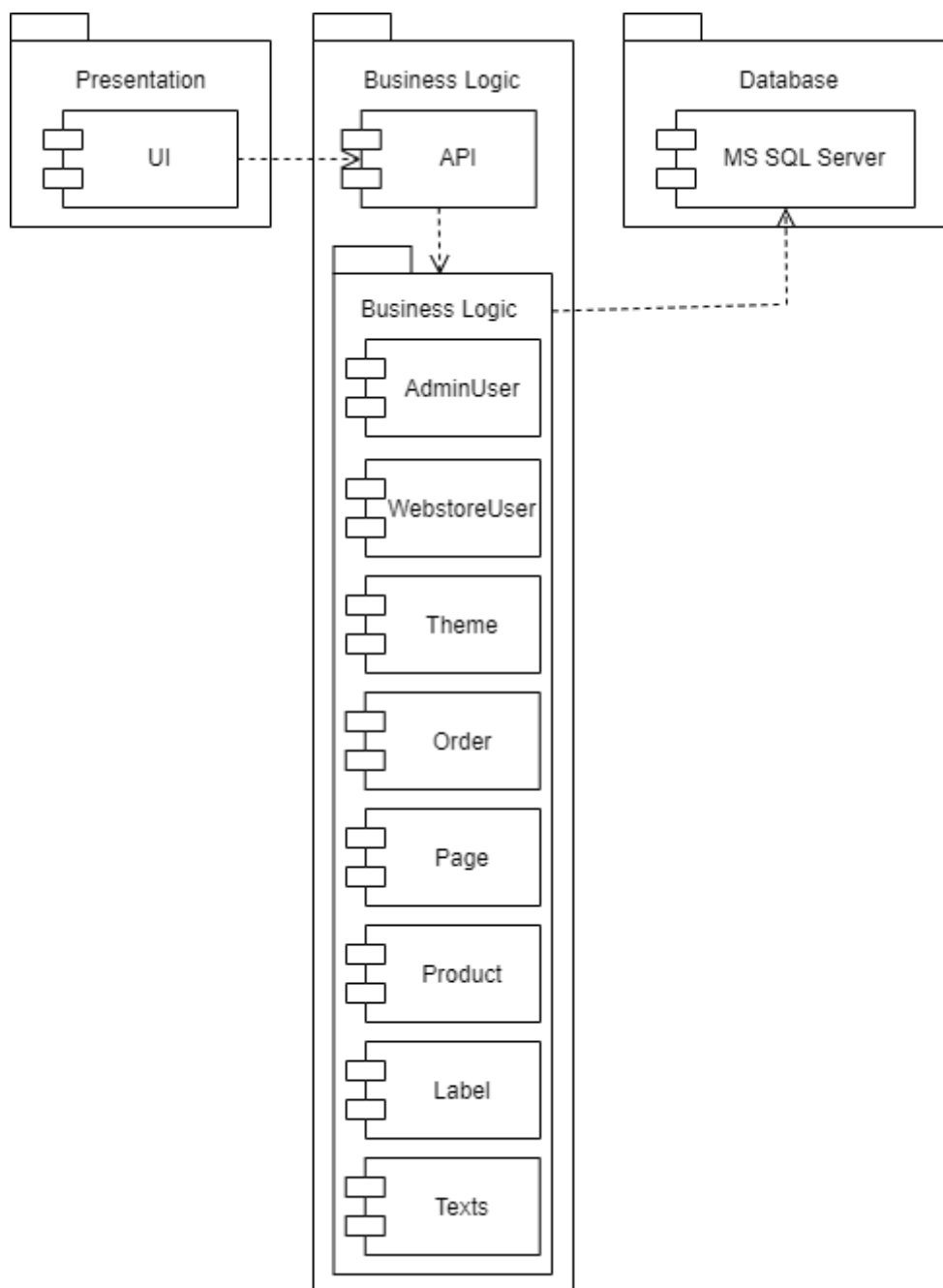


Рис. 3.2 – Діаграма компонентів

Компонент “Business Logic” включає кілька модулів, кожен з яких відповідає за конкретну частину функціональності додатку:

- AdminUser: модуль для управління адміністраторами. Включає функції створення, редагування та видалення облікових записів адміністраторів;
- WebstoreUser: модуль для управління користувачами магазину. Забезпечує функції реєстрації, авторизації та управління обліковими записами користувачів;

- Theme: модуль для управління темами сайту. Включає функції зміни візуального оформлення сайту;
- Order: модуль для управління замовленнями. Обробляє створення, перегляд та зміну замовлень;
- Page: модуль для управління сторінками сайту. Забезпечує створення, редагування та видалення сторінок;
- Product: модуль для управління продуктами. Включає функції додавання, редагування, видалення та перегляду продуктів;
- Label: модуль для управління мітками. Включає функції створення та управління мітками, які можуть бути прикріплені до продуктів;
- Texts: модуль для управління текстовим вмістом сайту. Забезпечує редагування текстів, які відображаються користувачам.

### 3.2 Структура інтерфейсу сайту для створення вебмагазинів.

Процес роботи системи починається з входу адміністратора (рис. 3.3).

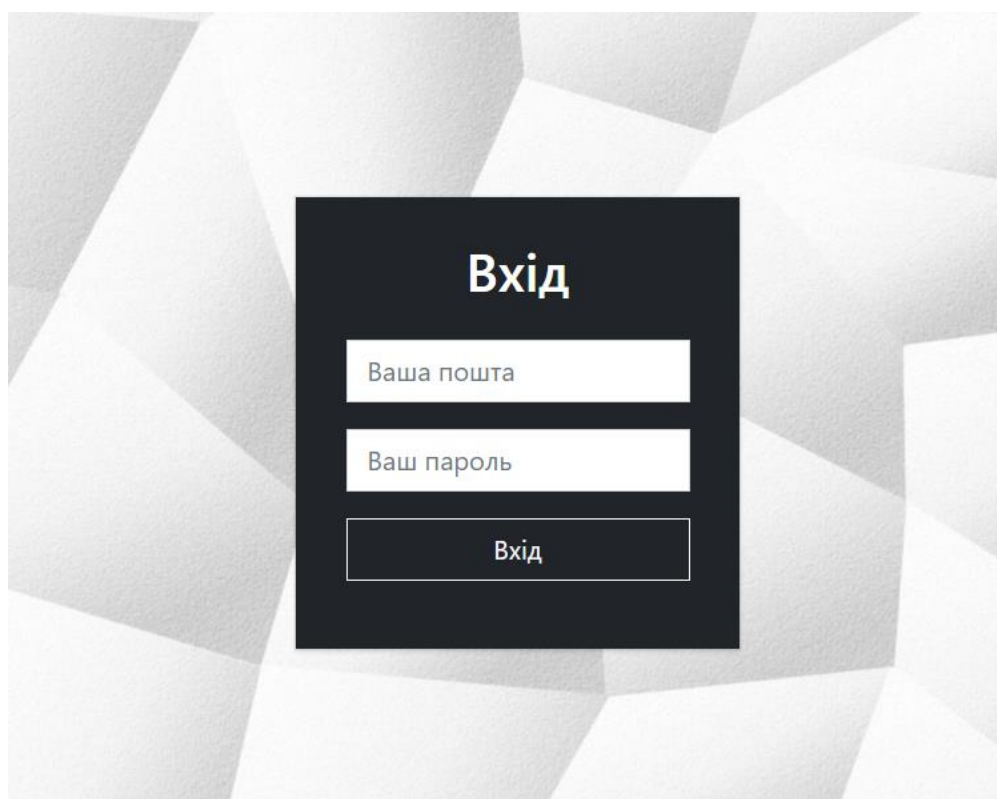


Рис. 3.3 – Вхід

Якщо процес входу пройшов успішно, адміністратор потрапляє на сторінку продуктів (рис. 3.4).

|  |  |  |  |  |                    |      |
|--|--|--|--|--|--------------------|------|
|  |  |  |  |  | ІПЗ.КР.Б-121-24-ПЗ | Арк. |
|  |  |  |  |  |                    | 44   |
|  |  |  |  |  |                    |      |

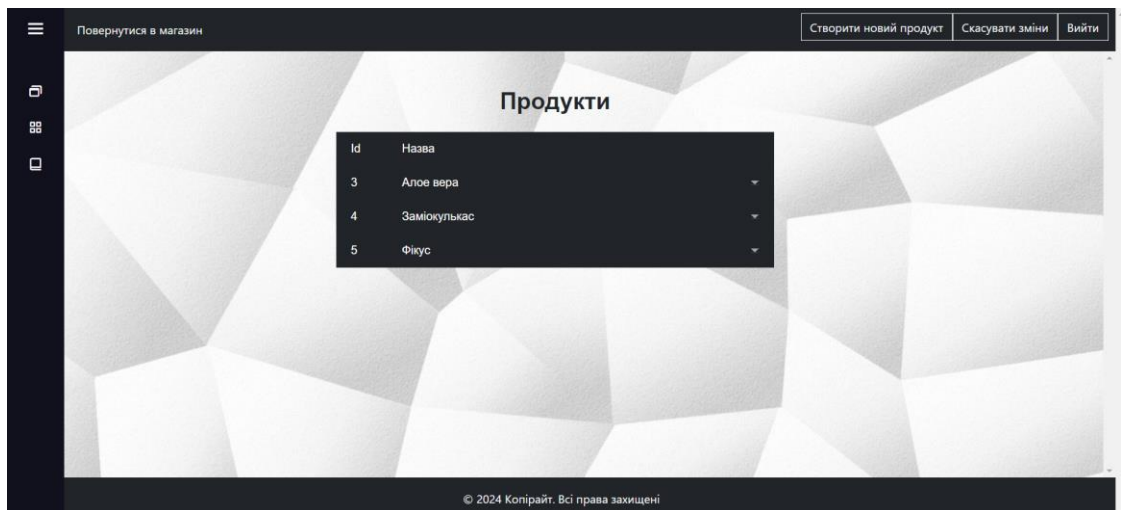


Рис. 3.4 – Сторінка продуктів

В адмін-панелі користувач може побачити навігацію зліва, яку можна розгорнути та подивитися всі сторінки адмін-частини (рис. 3.5).

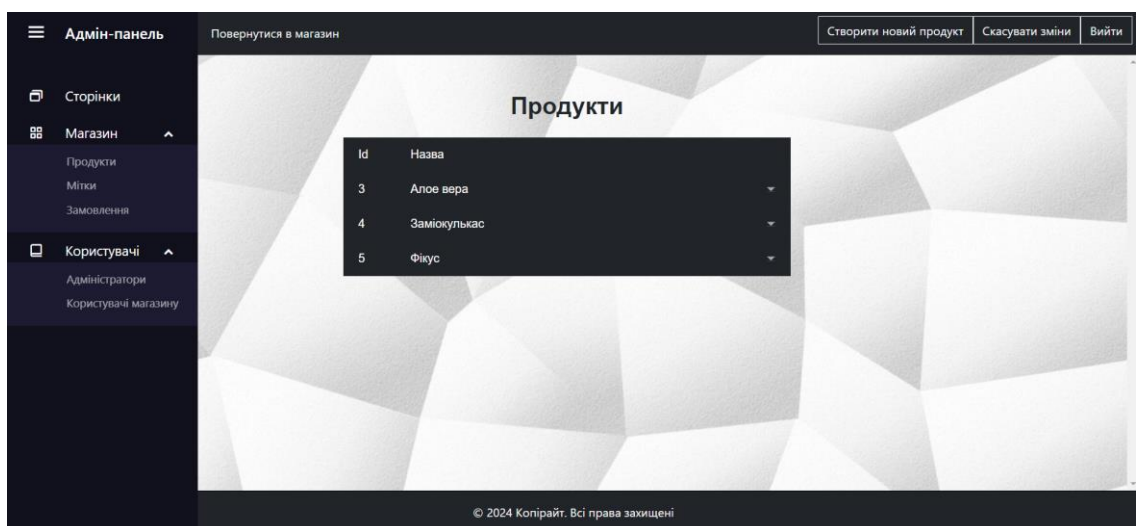
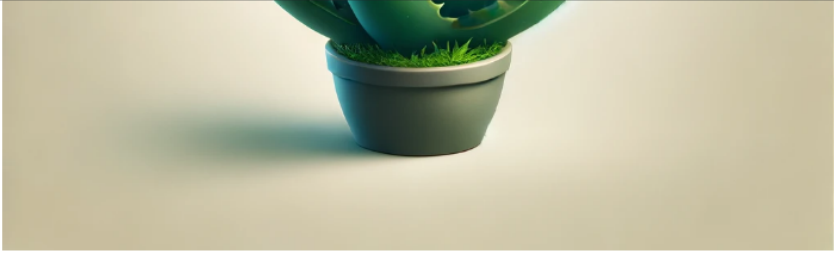


Рис. 3.5 – Бокова панель

При розгортанні продукта можна побачити повну інформацію про нього, а також редагувати її. У відкритій секції можна продивитися ідентифікатор продукта, назву, ціну, знижку, всі мітки, картинку, опис. Крім того є кнопка для видалення. Зміна текстових полів відбувається по натисканню на їх значення та послідовне використання кнопки оновлення (рис. 3.6).

Ст



Id:

3

Назва:

Алое вера

Ціна:

40

Знижка:

10

Мітка:

Розмір

Малий

Вікова група

-

Змінити картинку:

Choose File

Опис:

Рослина для дому

Оновити

Видалити

Рис. 3.6 – Деталі продукта

Можна перейти на вкладку з мітками, в яких можна налаштувати типи полів, які будуть у продукта, а також значення, яких вони можуть набувати. Це зроблено, по-перше, для того, щоб додати додаткову інформацію про продукт, а по-друге – для системи фільтрація на списку продуктів (рис. 3.7).

Мітки

| Id | Назва  | Ідентифікатор |                  |
|----|--------|---------------|------------------|
| 1  | Розмір | size          | Оновити Видалити |

| Id | Назва    | Ідентифікатор |                  |
|----|----------|---------------|------------------|
| 1  | Малий    | small         | Оновити Видалити |
| 2  | Середній | average       | Оновити Видалити |
| 8  | Великий  | big           | Оновити Видалити |

Назва

Ідентифікатор

Створити мітку

Рис. 3.7 – Мітки

|  |  |  |  |  |
|--|--|--|--|--|
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Знову відкривши бокову панель, та при переході на вкладку з замовленнями адміністратору надається можливість переглядати замовлення користувачів. Ці замовлення відображаються тей стан продуктів, який користувач бачив при оформленні замовлення. Після його отримання на адміністратора покладається відповідальність його обробити та прокомунікувати з покупцем (рис. 3.8).

| Замовлення |                   |  |
|------------|-------------------|--|
| Id         | Пошта             |  |
| 17         | admin@gmail.com ▼ |  |
| 18         | admin@gmail.com ▼ |  |
| 19         | admin@gmail.com ▼ |  |
| 20         | admin@gmail.com ▼ |  |
| 21         | admin@gmail.com ▲ |  |

| Id      | Назва     | Ціна  |
|---------|-----------|-------|
| 3       | Алое вера | 36.00 |
| Загалом |           | 36.00 |

Рис. 3.8 – Замовлення

При натисканні на вкладку з адміністраторами, користувач побачить їх повний список. На ньому можна буде як і редагувати поля, видаляти користувачів, так і створювати нових (рис. 3.9).

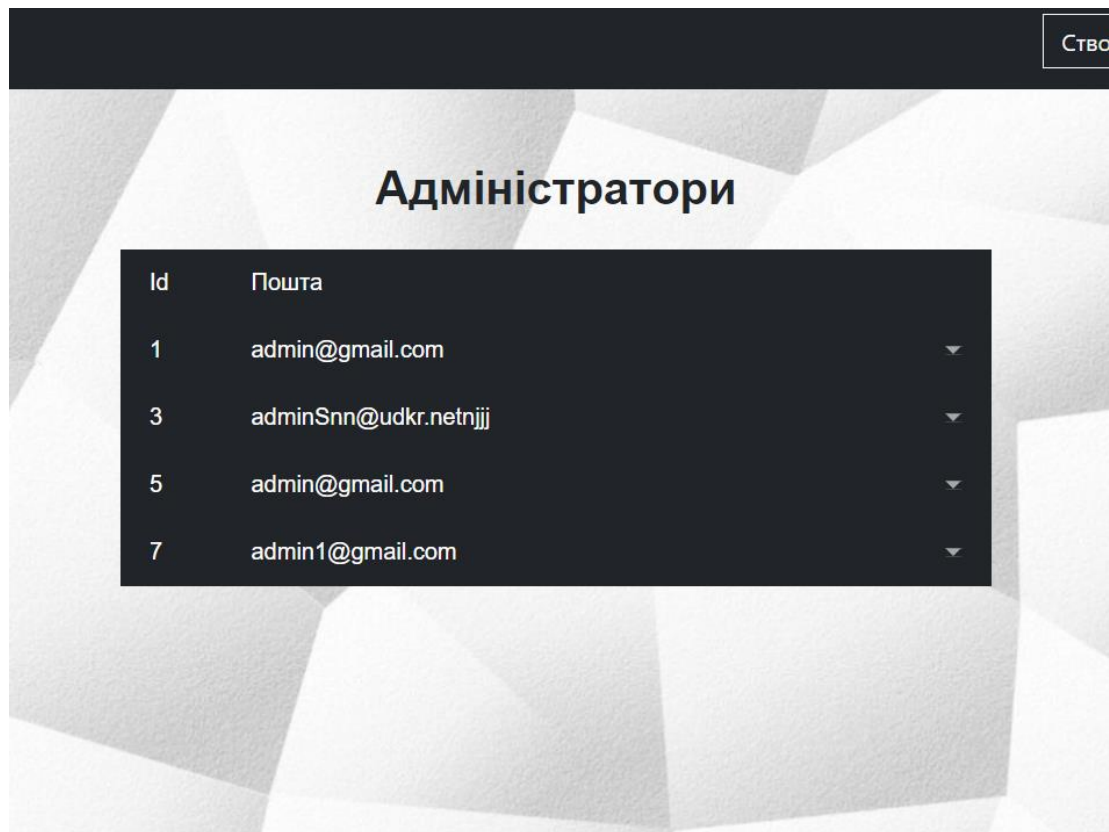


Рис. 3.9 – Адміністратори

При переході на вкладку з користувачами магазину, буде можливість переглянути їх повний список та редагувати (рис. 3.10).

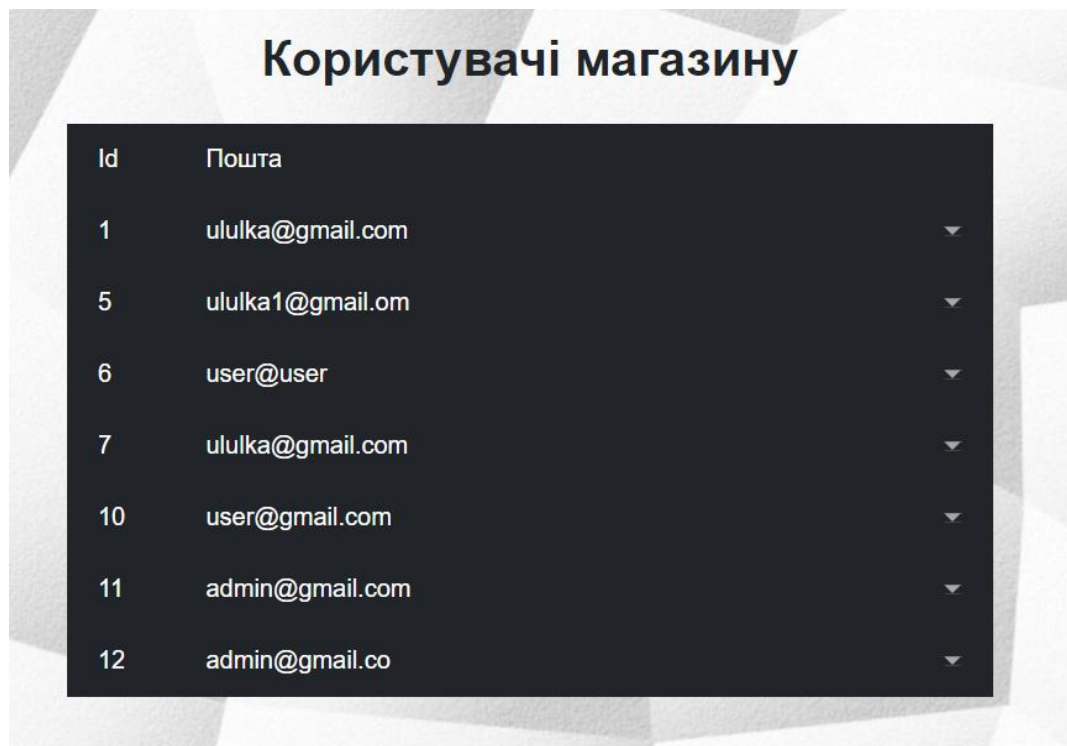


Рис. 3.10 – Користувачі магазину

|  |  |  |  |  |
|--|--|--|--|--|
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Найголовнішою функцією цього сайту є редагування змісту сторінок. Для цього спочатку потрібно буде перейти на список сторінок. На ньому можна буде побачити всі базові сторінки, включаючи список продуктів, кошик, вхід, реєстрація, деталі продукту та сторінку, яка буде показуватися у випадку, якщо введеного URL не існує на сайті. Базові сторінки заборонено видаляти, а також вони мають фіксований URL, який не можна змінити (рис. 3.11).

| Назва            | URL          |                     |
|------------------|--------------|---------------------|
| Список продуктів | products     | Редагувати          |
| Деталі продукту  | product      | Редагувати          |
| ShoppingCart     | shoppingCart | Редагувати          |
| Login            | login        | Редагувати          |
| Register         | register     | Редагувати          |
| Не знайдено      | 404          | Редагувати          |
| Про нас          | about-us     | Редагувати Видалити |

Рис. 3.11 – Сторінки магазину

При переході на список продуктів, відкривається вкладка з вікном, на якому видно вигляд сайту. При наведенні на це вікно стають явними кнопки редагування текстів, кнопки видалення, та кнопки додавання рядків, колонок та елементів. Справа на боковій панелі надається можливість переглянути фіксований URL сторінки для базових сторінок, але для створених власноруч можливість зміни надається (рис. 3.12).

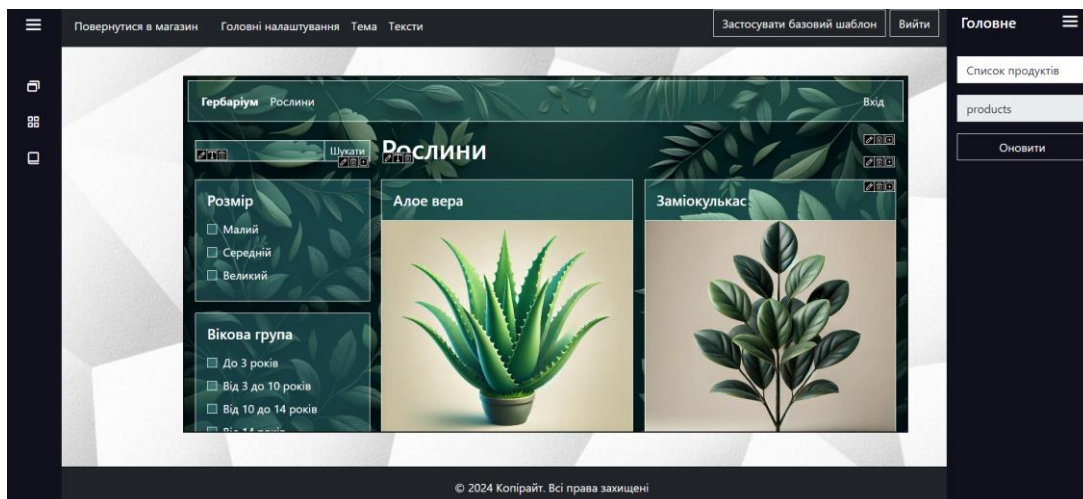


Рис. 3.12 – Сторінка редагування сторінки

При натисканні на вкладку з темою, відкривається панель з налаштуваннями, в яких можна змінити колір хедера та футера, а також налаштувати фон сайту (рис. 3.13).

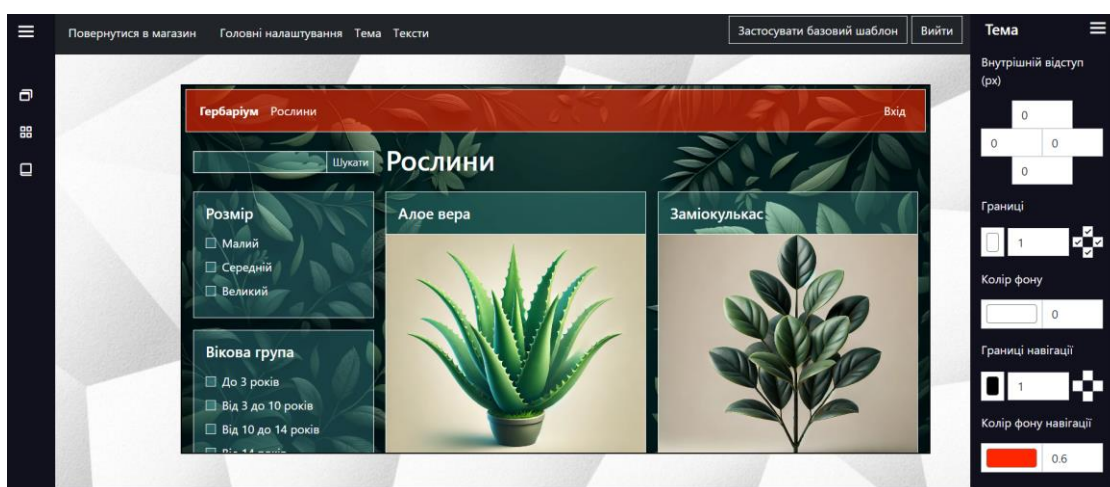


Рис. 3.13 – Редагування теми

При натисканні вікна кнопки додавання в колонці є можливість переглянути всі блоки, які можна додати на цю сторінку, визначається це за посиланням на сторінку. Наприклад, для списку продуктів, цей перелік буде складатися з пошуку, фільтрів, власне списку продуктів, а також загальних елементів, таких як посилання та текст (рис. 3.14).

|  |  |  |  |  |
|--|--|--|--|--|
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

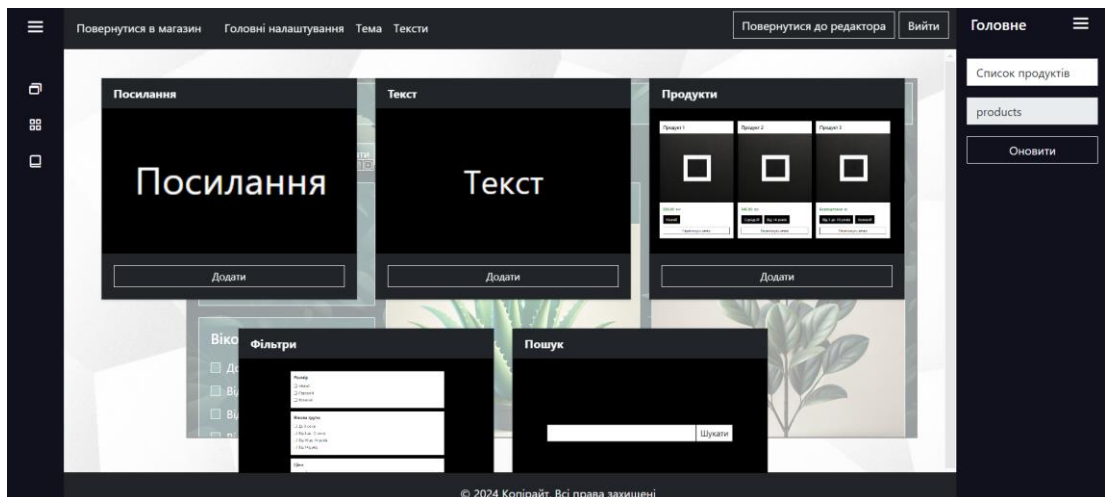


Рис. 3.14 – Редагування теми

### 3.3 Тестування сайту для конструювання вебмагазинів

Тестування є невід'ємною частиною забезпечення якості сайтів. Воно дозволяє детально перевірити функціональність, зручність використання та інтерактивність інтерфейсу з точки зору кінцевого користувача. Нижче наведено опис процесу ручного тестування та методи, які використовувалися.

Етапи ручного тестування:

#### 1. Планування тестування

1.1.Складання тест-плану: Визначення обсягу тестування, ключових функціональних областей та сценаріїв, які необхідно перевірити.

1.2.Підготовка тестових випадків: Розробка докладних тест-кейсів, які включають кроки для виконання, очікувані результати та критерії прийняття.

#### 2. Підготовка тестового середовища

2.1.Налаштування тестових облікових записів: Створення тестових облікових записів для адміністратора та звичайного користувача для перевірки різних рівнів доступу.

2.2.Встановлення браузерів: Використання різних веб-браузерів (Google Chrome, Mozilla Firefox, Safari, Microsoft Edge) для перевірки крос-браузерної сумісності.

#### 3. Виконання тестування

3.1. Функціональне тестування: Перевірка основних функціональних можливостей сайту, таких як реєстрація, авторизація, створення та редагування сторінок, управління продуктами, оформлення замовлень і перегляд кошика. Кожен тест-кейс виконувався вручну, фіксувалися результати та будь-які відхилення від очікуваної поведінки.

3.2. Тестування зручності використання (Usability Testing): Оцінка інтуїтивності та простоти навігації сайту. Використовувались сценарії, які імітують реальні дії користувачів, наприклад, пошук продуктів, зміна налаштувань профілю, додавання товарів до кошика та оформлення замовлення.

3.3. Тестування валідності форм: Перевірка обробки введених даних у формах (наприклад, реєстрація, вхід, створення продуктів). Тестувалися різні типи введених даних, включаючи коректні та некоректні значення, для перевірки роботи валідації та обробки помилок.

3.4. Перевірка мобільної версії: Виконання тест-кейсів на мобільних пристроях з різними розмірами екрану для перевірки адаптивного дизайну та зручності використання мобільної версії сайту.

3.5. Тестування наявних повідомлень та сповіщень: Оцінка правильності та зрозумілості повідомлень системи (наприклад, повідомлень про помилки, підтверджень дій).

Тестування відіграло важливу роль у забезпеченні якості сайту для конструювання вебмагазинів, допомагаючи ідентифікувати та виправити дефекти, які були не передбачені під час етапу написання коду, а також які були створені під час процесу рефакторингу старого коду. Це забезпечило надійну та зручну платформу для кінцевих користувачів.

### **Висновки до третього розділу**

У цьому розділі було детально описано розроблений сайт для створення вебмагазинів, починаючи від користувацького інтерфейсу і закінчуючи

|  |  |  |  |  |                    |      |
|--|--|--|--|--|--------------------|------|
|  |  |  |  |  | ІПЗ.КР.Б-121-24-ПЗ | Арк. |
|  |  |  |  |  |                    | 52   |
|  |  |  |  |  |                    |      |

процесом тестування. Інтерфейс користувача, зосереджений на веб-інтерфейсі, надає всі необхідні параметри для взаємодії з системою, що дозволяє користувачам легко керувати процесом створення та редагування вебмагазинів. Було продемонстровано роботу системи, яка включає управління сторінками, продуктами, замовленнями та іншими елементами сайту. Такий підхід сприяє кращому розумінню взаємодії компонентів системи та підкреслює її модульність.

Налаштування системи було спрощено до вибору параметрів через веб-інтерфейс, що робить її використання легким і мінімізує можливість помилок, пов'язаних з конфігурацією. Ручне тестування системи, яке було описано, включає ряд сценаріїв, що перевіряють усі аспекти системи, від створення облікових записів до управління замовленнями та товарами. Це підтверджує коректність роботи системи та забезпечує високий рівень довіри до її результатів.

Завдяки детальному аналізу, можна зрозуміти структуру системи, її ключові можливості та взаємодію між компонентами. Такий підхід не тільки висвітлює функціональність системи для кінцевих користувачів, але й виокремлює аспекти, які можуть бути покращені в майбутніх версіях.

## ВИСНОВКИ

У межах виконання даної кваліфікаційної роботи було розроблено програму для створення вебмагазинів, який включає сучасні функції та забезпечує високу гнучкість і адаптивність до потреб користувачів. Основна мета проекту полягала у створенні надійного та ефективного інструменту, який дозволяє користувачам без спеціальних технічних знань створювати та керувати своїм вебмагазином. Протягом роботи було здійснено аналіз існуючих рішень, визначення ключових вимог до програми, проектування архітектури, вибір відповідних технологій та методів розробки, а також реалізація і тестування створеного продукту.

Результатом роботи стала програма, здатна ефективно підтримувати весь цикл створення вебмагазину - від реєстрації користувачів до управління продуктами та замовленнями. Значну увагу було приділено інтуїтивно зрозумілому інтерфейсу, що забезпечує легкість використання додатку та ефективність роботи з даними. Важливим аспектом було впровадження адаптивного дизайну, який гарантує зручність користування додатком на різних пристроях, включаючи мобільні телефони та планшети.

Особливу увагу було приділено механізму редагування сторінок, що дозволяють користувачам легко кастомізувати сайт. Крім того, сайт підтримує ряд спеціальних функцій, таких як налаштування теми, зміна текстів та управління товарами, що робить його більш гнучким та адаптивним до сучасних вимог електронної комерції.

В процесі розробки було враховано кращі практики сучасної веброзробки та рекомендації з безпеки, що дозволило створити надійну платформу для вебмагазинів. Аналіз сучасних систем управління контентом, таких як WordPress та Wix, дозволив виявити ключові тренди та напрямки для покращення, що в свою чергу надало можливість інтегрувати в додаток передові технології управління та оптимізації контенту.

Завершення цього проекту не лише дозволило створити ефективний інструмент для електронної комерції, але й дало можливість провести

|  |  |  |  |  |                    |      |
|--|--|--|--|--|--------------------|------|
|  |  |  |  |  | ІПЗ.КР.Б-121-24-ПЗ | Арк. |
|  |  |  |  |  |                    |      |
|  |  |  |  |  |                    | 54   |

детальний аналіз і глибше зрозуміти роботу сучасних систем управління контентом. Результати роботи мають значний потенціал для подальших досліджень у галузі вебтехнологій та електронної комерції.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. WordPress, Your Way [Електронний ресурс] // WordPress.com. – Режим доступу: <https://wordpress.com/>
2. Website builder – create a website in minutes – squarespace [Електронний ресурс] // Squarespace. – Режим доступу: <https://www.squarespace.com/>
3. Конструктор сайтів | створіть сайт безкоштовно вже сьогодні | wix.com [Електронний ресурс] // wix.com. – Режим доступу: <https://uk.wix.com/>
4. Documentation | NestJS - A progressive Node.js framework [Electronic resource] // Documentation | NestJS - A progressive Node.js framework. – Режим доступу: <https://docs.nestjs.com/techniques/mvc>
5. Express 5.x - API reference [Електронний ресурс] // Express - Node.js web application framework. – Режим доступу: <https://expressjs.com/en/5x/api.html#app.get>
6. Node.js – Run JavaScript Everywhere [Електронний ресурс] // Node.js – Run JavaScript Everywhere. – Режим доступу: <https://nodejs.org/en>
7. TypeORM - amazing ORM for typescript and javascript. [Електронний ресурс] // TypeORM - Amazing ORM for TypeScript and JavaScript. – Режим доступу: <https://typeorm.io/>
8. Documentation - everyday types [Електронний ресурс] // TypeScript: JavaScript With Syntax For Types. – Режим доступу: <https://www.typescriptlang.org/docs/handbook/2/everyday-types.html>
9. Databases - SQL server [Електронний ресурс] // Microsoft Learn: Build skills that open doors in your career. – Режим доступу: <https://learn.microsoft.com/en-us/sql/relational-databases/databases/databases?view=sql-server-ver16>
10. W3Schools.com [Електронний ресурс] // W3Schools Online Web Tutorials. – Режим доступу: <https://www.w3schools.com/sql/>

11. Bcrypt [Электронный ресурс] // npm. – Режим доступа: <https://www.npmjs.com/package/bcrypt>
12. Passport.js [Электронный ресурс] // Passport.js. – Режим доступа: <http://www.passportjs.org/>
13. Simpson K. You Don't Know JS Yet: Get Started / Kyle Simpson., 2020. – 143 с.
14. Davis C. Cloud Native Patterns: Designing change-tolerant software / Cornelia Davis., 2019. – 400 с.
15. Casciaro M. Node.js Design Patterns - Third edition: Design and implement production-grade Node.js applications using proven patterns and techniques / M. Casciaro, L. Mammino., 2020. – 664 с.
16. Brown E. Web Development with Node and Express: Leveraging the JavaScript Stack / Ethan Brown., 2020. – 332 с.
17. Ludin S. Learning HTTP/2: A Practical Guide for Beginners / S. Ludin, J. Garza., 2017. – 154 с.
18. Accomazzo A. Fullstack React: The Complete Guide to ReactJS and Friends / A. Accomazzo, N. Murray, A. Lerner., 2018. – 836 с.
19. Fowler M. Refactoring: Improving the Design of Existing Code / Martin Fowler., 2018. – 431 с.
20. React [Электронный ресурс] // React. – Режим доступа: <https://reactjs.org/>
21. Redux - A JS library for predictable and maintainable global state management | Redux [Электронный ресурс] // Redux - A JS library for predictable and maintainable global state management | Redux. – Режим доступа: <https://redux.js.org/>
22. TanStack query [Electronic resource] // TanStack | High Quality Open-Source Software for Web Developers. – Mode of access: <https://tanstack.com/query/v3>
23. Ant Design - The world's second most popular React UI framework [Электронный ресурс] // Ant Design - The world's second most popular React UI framework. – Режим доступа: <https://ant.design/>

24. Bootstrap [Электронный ресурс] // Bootstrap · The most popular HTML, CSS, and JS library in the world. – Режим доступа: <https://getbootstrap.com/>
25. React bootstrap | react bootstrap [Электронный ресурс] // React Bootstrap | React Bootstrap. – Режим доступа: <https://react-bootstrap.netlify.app/>
26. Formik [Электронный ресурс] // Formik: Build forms in React, without the tears. – Режим доступа: <https://formik.org/>
27. React Hook Form - performant, flexible and extensible form library [Электронный ресурс] // React Hook Form - performant, flexible and extensible form library. – Режим доступа: <https://react-hook-form.com/>
28. Home v6.23.1 | react router [Электронный ресурс] // Home v6.23.1 | React Router. – Режим доступа: <https://reactrouter.com/en/main>
29. Material UI: React components that implement Material Design [Электронный ресурс] // MUI: The React component library you always wanted. – Режим доступа: <https://mui.com/material-ui/>
30. Styled-components [Электронный ресурс] // styled-components. – Режим доступа: <https://styled-components.com/>

ДОДАТКИ

Реалізація бібліотеки для перемальовування змісту.

Лістинг коду:

```
const createElement = (tagName, { ...initProps } = {}) => {
  return ({ attributes, properties, ...props } = {}) => {
    return (...children) => {
      let elem = document.createElement(tagName);

      [initProps, props].forEach(props =>
        Object.entries(props)
          .forEach(([key, value]) => elem[key] = value)
      );

      attributes && Object.entries(attributes)
        .forEach(([key, value]) => elem.setAttribute(key, value));

      properties && Object.entries(properties)
        .forEach(([key, value]) => elem.style.setProperty(key, value));

      if (children)
        children.forEach(child => elem.append(child));

      return elem;
    }
  }
}

export const button = createElement('button');
export const a = createElement('a');
export const div = createElement('div');
export const p = createElement('p');
export const header = createElement('header');
export const footer = createElement('footer');
export const nav = createElement('nav');
export const aside = createElement('aside');
export const section = createElement('section');
export const main = createElement('main');
export const span = createElement('span');
export const script = createElement('script');
export const label = createElement('label');
export const table = createElement('table');
export const tr = createElement('tr');
export const td = createElement('td');
export const select = createElement('select');
export const option = createElement('option');
export const img = createElement('img');
export const input = createElement('input');
export const radio = createElement('input', { type: 'radio' });
```

```

export const checkbox = createElement('input', { type: 'checkbox' });
export const textInput = createElement('input', { type: 'text' });
export const buttonInput = createElement('input', { type: 'button' });
export const numberInput = createElement('input', { type: 'number' });
export const passwordInput = createElement('input', { type: 'password' });
export const range = createElement('input', { type: 'range' });
export const textarea = createElement('textarea');
export const h1 = createElement('h1');
export const h2 = createElement('h2');
export const h3 = createElement('h3');
export const h4 = createElement('h4');
export const h5 = createElement('h5');
export const h6 = createElement('h6');
export const ul = createElement('ul');
export const li = createElement('li');
export const small = createElement('small');
export const strong = createElement('strong');
export const form = createElement('form');
export const row = (props) => (...children) =>
    table()(tr(props)(...children));

function createGeneralState() {
    const generalState = {};

    return {
        getState: id => {
            if (!generalState[id])
                generalState[id] = {};

            return generalState[id];
        },
    }
};

const generalState = createGeneralState();

export function createComponent(helpersProvider, stateId = getUUID()) {
    return (function () {
        const state = generalState.getState(stateId);
        let newStateIndex = 0;

        const [useAfterRender, useBeforeRender] =
            ['afterRender', 'beforeRender'].map(effect => {
                state[effect] = [];
                return (func) => state[effect].push(func);
            });

        state.portals = [];

        const openPortal = (element, contentFunc) => {

```

```

        element.innerHTML = '';
        element.append(contentFunc())
        state.portals.push({ element, contentFunc });
    };

    const render = (contentFunc) => {
        const wrapper = contentFunc();
        wrapper.innerHTML = '';

        state.render = () => {
            state.beforeRender.forEach(f => f());

            state.portals.forEach(({ element, contentFunc }) => {
                element.innerHTML = '';
                element.append(contentFunc())
            });

            wrapper.innerHTML = '';
            wrapper.append(...contentFunc().childNodes);

            state.afterRender.forEach(f => f());
        };

        state.beforeRender.forEach(f => f());
        wrapper.append(...contentFunc().childNodes);
        state.afterRender.forEach(f => f());

        return wrapper;
    }

    const useState = (initValue) => {
        const key = newStateIndex++;
        state[key] = initValue;

        return [
            () => state[key],
            (value) => {
                state[key] = value;
                state.render?.();
            }
        ];
    }

    return helpersProvider({
        useState, render, useAfterRender, useBeforeRender, openPortal,
    });
})();

export const getUUID = () => _UUID.generate();

```

## Реалізація класу PagesController.

## ЛІСТИНГ КОДУ:

```

import Controller from '../Controller.js';
import AdminController from './AdminController.js';
import Page from '../../models/Page.js';
import typeorm from 'typeorm';
import Theme from '../../models/Theme.js';
import Text from '../../models/Text.js';
const { getRepository } = typeorm;

export default class PagesController extends AdminController {
  public static async initialize() {
    const repository = getRepository(Page);
    const pages = await repository.find();

    const basePages = [];

    for (const key of Object.keys(BasePageUrl)) {
      const page = pages.find(page => page.url === BasePageUrl[key]);
      if (page)
        continue;

      basePages.push({
        name: key,
        url: BasePageUrl[key],
        content: getBaseTemplate(BasePageUrl[key]),
        mobileContent: getMobileTemplate(BasePageUrl[key]),
        isCustom: 0,
      });
    }

    if (basePages.length > 0)
      await repository.save(basePages);
  }

  @Controller.get('/admin/pages',
AdminController.redirectToLoginIfNotAuthenticated)
  private static async getList(req, res) {
    res.render('admin/pages', {
      pages: await getRepository(Page).find(),
    });
  }

  @Controller.get('/admin/pages/create',
AdminController.redirectToLoginIfNotAuthenticated)
  private static getCreationPage(req, res) {
    res.render('admin/pageCreation');
  }
}

```

```

    }

    @Controller.get('/admin/pages/emptyPage',
AdminController.redirectToLoginIfNotAuthenticated)
    private static getEmptyPage(req, res) {
        res.render('webstore/emptyPage');
    }

    @Controller.post('/admin/pages/create',
AdminController.redirectToLoginIfNotAuthenticated)
    private static async postCreate(req, res) {
        try {
            await getRepository(Page).save({
                name: req.body.name,
                url: req.body.url,
                content: req.body.content,
                mobileContent: req.body.mobileContent,
                isCustom: 1,
            });

            const themeRepository = getRepository(Theme);
            const theme = await themeRepository.findOne();
            await themeRepository.save({
                ...(theme || {}),
                content: req.body.theme,
            });

            const textRepository = getRepository(Text);
            const texts = JSON.parse(req.body.texts);
            await textRepository.save(
                Object.keys(texts).map(key => ({
                    identifier: key,
                    content: JSON.stringify(texts[key]),
                })))
        );
    } catch (error) {
        res.redirect('/admin/pages/create');
        return;
    }

    res.redirect('/admin/pages');
}

@Controller.post('/admin/pages/delete',
AdminController.redirectToLoginIfNotAuthenticated)
    private static async deletePage(req, res) {
        try {
            const id = req.body.id;
            const repository = getRepository(Page);
            const page = await repository.findOne({ id });

```

```

        if (page && page.isCustom)
            await repository.remove(page);
    } catch (error) {
        res.redirect('/admin/pages');
        return;
    }

    res.redirect('/admin/pages');
}

@Controller.get('/admin/pages/update/:id',
AdminController.redirectToLoginIfNotAuthenticated)
private static async getUpdate(req, res) {
    res.render('admin/pageUpdate');
}

@Controller.get('/admin/pages/getData/:id?')
private static async getPageInfo(req, res) {
    const page = req.params.id !== undefined && await
getRepository(Page).findOne({ id: req.params.id });
    const theme = await getRepository(Theme).findOne();
    const texts = await getRepository(Text).find();

    res.send({ ...(page || {}), theme: theme?.content, texts });
}

@Controller.post('/admin/pages/update',
AdminController.redirectToLoginIfNotAuthenticated)
private static async postUpdate(req, res) {
    try {
        const repository = getRepository(Page);
        const page = await repository.findOne({ id: req.body.id });
        await repository.save({
            id: page.id,
            name: req.body.name,
            url: page.isCustom ? req.body.url : page.url,
            content: req.body.content,
            mobileContent: req.body.mobileContent,
            isCustom: page.isCustom,
        });

        const themeRepository = getRepository(Theme);
        const theme = await themeRepository.findOne();
        await themeRepository.save({
            ...(theme || {}),
            content: req.body.theme,
        });

        const textRepository = getRepository(Text);
        const texts = JSON.parse(req.body.texts);

```

```

        await textRepository.save(
            Object.keys(texts).map(key => ({
                identifier: key,
                content: JSON.stringify(texts[key]),
            }))
        );
    } catch (error) {
        console.log(error);
        res.redirect('/admin/pages');
        return;
    }

    res.redirect('/admin/pages');
}

@Controller.post('/upload',
AdminController.redirectToLoginIfNotAuthenticated)
private static uploadImage(req, res) {
    Controller.upload(req, res, err => {
        console.log({ err });
        if (err) {
            res.send(err);
        } else {
            if ((req as any).file == undefined) {
                res.send('No file selected!');
            } else {
                res.send(`File uploaded successfully!`);
            }
        }
    });
}

@Controller.delete('/delete/:filename',
AdminController.redirectToLoginIfNotAuthenticated)
private static async deleteImageFromStore(req, res) {
    try {
        Controller.deleteImage(req.params.filename, (err) => {
            if (err) {
                console.error(`Error deleting file: ${err}`);
                return res.status(500).send('Error deleting file.');
```

```

enum BasePageUrl {
    ProductList = 'products',
    ProductDetails = 'product',
    ShoppingCart = 'shoppingCart',
    Login = 'login',
    Register = 'register',
    NotFound = '404'
}

function getBaseTemplate(key: string): string {
    switch (key) {
        case BasePageUrl.ProductList:
            return '[{"id":"ae21b029-1001-43a9-b729-c35cd01a659d","cols":[{"id":"499693e4-9f19-4db6-83c0-9370902b8f0e","elements":[{"elementKey":"search","id":"7bba2d64-349b-4efe-9bce-4f94f4178790","data":{"margin":{"top":"10","bottom":"10"}}}], "width":"25","style":"","id":"3b703f60-01cc-426e-a864-f45a687cff00","width":"75","elements":[{"elementKey":"text","id":"b246405d-32b9-4d54-9b85-03a4cf430117","data":{"content":"Список товаров","size":"h1","padding":{"left":"0"}}}], "style":"","padding":{"left":"15"} } ] ], "style":"","margin":{"top":"15","left":"15","right":"15","bottom":"" } }, {"id":"8a0a99f6-d3a7-4208-ab0a-458b15679109","cols":[{"id":"bb76d3dc-848b-4c17-9791-5700f57377ec","elements":[{"elementKey":"productsFilters","id":"b6584cd3-03e2-47e4-a953-2ea2b43bd79c"}], "width":"25","style":"","id":"be437d33-4d32-45f8-ac18-89bde1d45835","width":"75","elements":[{"elementKey":"productList","id":"3e3a8321-6116-458c-9f3b-0d42be073f11","data":{"padding":{"left":"" } } ], "style":"","padding":{"left":"15"} } ] ], "style":"","margin":{"top":"15","left":"15","bottom":"15","right":"15"} } ] ]';
        case BasePageUrl.ProductDetails:
            return '[{"id":"d1eedaff-3c10-4133-9455-b6268c0deed3","cols":[{"id":"bd5cb2ab-0657-49fb-b429-b5eaaf949646","elements":[{"elementKey":"productDetails","id":"7cdeac0e-df31-46d9-b04b-356fcd31ff50"}], "width":100,"style":"" } ], "style":"","margin":{"top":"15","bottom":"15"} } ]';
        case BasePageUrl.ShoppingCart:
            return '[{"id":"a9934c10-a46e-4b8d-8c84-48bbbd95cbe9","cols":[{"id":"9ebb109d-e87e-4e00-89c7-6879d04cbf11","elements":[{"elementKey":"shoppingCart","id":"7c715342-4d6f-46fe-a649-084f408ca691"}], "width":100,"style":"" } ], "style":"","margin":{"top":"100","bottom":"100"} } ]';
        case BasePageUrl.Login:
            return '[{"id":"d8537d0f-1bc6-45be-9ad5-1c3c401ebfa2","cols":[{"id":"593b3932-f22c-49e9-9ce1-8c755abbc731","elements":[{"elementKey":"login","id":"36e7cadf-2f47-4f76-b7d7-791258cfc2c8"}], "width":100,"style":"" } ], "style":"","margin":{"top":"125","bottom":"125"} } ]';
        case BasePageUrl.Register:
    
```



```

        case BasePageUrl.Register:
            return '[{"id":"de7600e4-5add-43b8-adcf-
fecbfe1bfd8c","cols":[{"id":"ff710447-b433-4d11-9e83-
5df98605bc2b","elements":[{"elementKey":"register","id":"58ef727e-cc95-49b3-90b2-
aba514b0d21c"}],"width":100,"style":""}], "style":"","margin":{"top":"15","bottom"
:"15","left":"5","right":"5"}}]';
        case BasePageUrl.NotFound:
            return '[{"id":"2121dae4-b2ad-4df5-be5d-
6cee4e02533c","cols":[{"id":"be186e8d-82c3-4f70-bec2-
46e27f4ab2d2","elements":[{"elementKey":"text","id":"7880b90c-bc38-4a80-bc13-
901218aa7c9e","data":{"size":"h1","textAlign":"center","content":"404"}}],"width"
:100,"style":""}], "style":"","margin":{"top":"40","left":"20","right":"20","botto
m":"40"}}]';
        default:
            return '';
    }
}

```

## Реалізація класу WebstoreController.

## ЛІСТИНГ КОДУ:

```

import Controller from '../Controller.js';
import bcrypt from 'bcrypt';
import { Strategy as LocalStrategy } from 'passport-local';
import WebstoreUser from '../../models/WebstoreUser.js';
import typeorm from 'typeorm';
const { getRepository } = typeorm;

export default class WebstoreController extends Controller {
  public static initialize(app) {
    this.passport.use('webstoreUserLocal', new LocalStrategy(
      {
        usernameField: 'email',
        passwordField: 'password',
        passReqToCallback: true,
      },
      async (req, email, password, done) => {
        try {
          const webstoreUser = await
getRepository(WebstoreUser).findOne({ email });
          if (webstoreUser && (await bcrypt.compare(password,
webstoreUser.password)))
            return done(null, { webstoreUser, adminUser:
req.user?.adminUser });

          return done(null, req.user, { messages: { error: 'Введені
дані не є вірними.' }});
        } catch (err) {
          return done(err);
        }
      }
    ));
  }

  protected static auth(req, res, next) {
    Controller.passport.authenticate('webstoreUserLocal', function (err,
user, info) {
      if (err) {
        return next(err);
      }

      if (!user.webstoreUser) {
        req.session.context = { ...req.session.context, ...info };
        return res.redirect('/login');
      }
    }
  }
}

```

```

        req.logIn(user, function (err) {
            if (err) {
                return next(err);
            }

            return res.redirect('/');
        });
    })(req, res, next);
}

protected static checkAuthenticated(failureRedirect) {
    return this.checkAuthenticatedHelper(true, failureRedirect);
}

protected static checkNotAuthenticated(failureRedirect) {
    return this.checkAuthenticatedHelper(false, failureRedirect);
}

private static checkAuthenticatedHelper(isAuthenticated, failureRedirect) {
    return async (req, res, next) => {
        const user = await getRepository(WebstoreUser).findOne({
            email: req.user?.webstoreUser?.email
        });

        if (Boolean(user) === isAuthenticated)
            return next();

        res.redirect(failureRedirect);
    }
}

protected static async isAuthenticated(req) {
    return Boolean(await WebstoreController.getUser(req));
}

protected static async getUser(req) {
    const user = await getRepository(WebstoreUser).findOne({
        email: req.user?.webstoreUser?.email
    });

    return user ? (({ password, ...rest }) => rest)(user) : null;
}
}

```