

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему «ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ З
МОЖЛИВІСТЮ АВТОМАТИЗАЦІЇ АДМІНІСТРУВАННЯ РОБОТИ СЕРВІСУ
ЗАМОВЛЕННЯ ПІЦЦІ
SOFTWARE IMPLEMENTATION OF A WEB APPLICATION WITH THE
OPTION TO AUTOMATE THE ADMINISTRATION OF A PIZZA ORDERING
SERVICE»

Виконав(ла): студент(ка) 4 курсу, групи КНТ-221

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

БАЛЮРА М.І.

(ПРИЗВИЩЕ та ініціали)

Керівник ЛЕОЩЕНКО С.Д.

(ПРИЗВИЩЕ та ініціали)

Рецензент БАБЕНКО Н.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. СУББОТІН
« » 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

БАЛЮРИ Максима Івановича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація вебзастосунку з можливістю автоматизації адміністрування роботи сервісу замовлення піци. Software Implementation of a Web Application with the Option to Automate the Administration of a Pizza Ordering Service

керівник проєкту (роботи) д-р.філос., доцент, ЛЕОЩЕНКО Сергій Дмитрович

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від «28» квітня 2025 року № 209

2. Строк подання студентом проєкту (роботи) 02 червня 2025 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Обрання засобів та середовища розробки. 3. Структура та програмна реалізація вебзастосунку. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

15 слайдів презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ЛЕОЩЕНКО С.Д., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання «14» квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка структури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділи 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Максим БАЛЮРА
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Сергій ЛЕОЩЕНКО
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
83 с., 11 табл., 35 рис., 3 дод., 27 джерел.

PIZZA, REACT, REDUX, FRONTEND, BACKEND, NODEJS,
TYPESCRIPT, JAVASCRIPT, EXPRESSJS, MONGODB.

Об'єкт дослідження – процес розробки вебзастосунків з можливістю автоматизації адміністрування роботи сервісу замовлення піци.

Предмет дослідження – програмні засоби, зокрема, вебзастосунки для замовлення піци з можливістю автоматизації адміністрування роботи сервісу.

Мета роботи – створення вебзастосунку з можливістю автоматизації адміністрування роботи сервісу замовлення піци.

Матеріали, методи та технічні засоби: редактор коду Visual Studio Code, фреймворк Express.js, база даних MongoDB, мова програмування TypeScript, фреймворк React з використанням бібліотеки для керування стану Redux, препроцесор CSS – SCSS, персональний комп'ютер під керуванням Windows 10 Version 22H2 з процесором Intel I3-8100.

Результати. Розроблено вебзастосунок з можливістю автоматизації адміністрування роботи сервісу замовлення піци.

Висновки. Розроблений вебзастосунок забезпечує ефективний моніторинг роботи сервісу замовлення піци, спрощує контроль за замовленнями та дозволяє оцінювати популярність товарів.

Галузь використання. Вебзастосунок може стати важливим інструментом для підприємств, що працюють в індустрії харчування, особливо в секторі онлайн-замовлення їжі, забезпечуючи автоматизацію, зручність для користувачів та ефективний контроль для адміністраторів.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 83 pages, 11 tables, 35 figures, 3 appendixes, 27 sources.

PIZZA, REACT, REDUX, FRONTEND, BACKEND, NODEJS, TYPESCRIPT, JAVASCRIPT, EXPRESSJS, MONGODB.

The object of research is the process of developing web applications with the option to automate the administration of a pizza ordering service.

The subject of research is software tools, in particular, web applications for ordering pizza with the option to automate the administration of the service.

Purpose – to create a web application with the option to automate the administration of a pizza ordering service.

Materials, methods and equipment: Visual Studio Code code editor, Express.js framework, MongoDB database, TypeScript programming language, React framework using the Redux state management library, CSS preprocessor – SCSS, a personal computer running Windows 10 Version 22H2 with an Intel I3-8100 processor.

Results. A web application has been developed with the option to automate the administration of a pizza ordering service. The application allows you to view orders, a list of users and analyse popular pizza items.

Conclusions. The developed web application provides effective monitoring of the pizza ordering service, simplifies order control and allows you to evaluate the popularity of goods.

Scope of application. The web application can become an important tool for businesses operating in the food industry, especially in the online food ordering sector, providing automation, convenience for users and effective control for administrators.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ.....	9
1 Аналіз предметної області.....	11
1.1 Огляд предметної області.....	11
1.2 Потреба програмних рішень	11
1.3 Огляд аналогів	12
1.3.1 Papa John's.....	12
1.3.2 Domino's Pizza	13
1.3.3 Pizza Hut	14
1.3.4 Порівняння аналогів	16
1.4 Висновки за розділом.....	17
2 Обрання засобів та середовища розробки	18
2.1 Вибір мови програмування	18
2.2 Вибір фреймворку для фронтенду.....	19
2.3 Вибір фреймворку для бекенду	21
2.4 Вибір бази даних	23
2.5 Вибір середовища розробки	25
2.6 Висновки за розділом.....	26
3 Структура та програмна реалізація вебзастосунку.....	27
3.1 Структурна схема розробки	27
3.2 Архітектурний підхід у додатку	28
3.3 Опис серверної частини програми	29
3.3.1 Використані бібліотеки та пакети	29
3.3.2 Структура каталогів серверної частини.....	30
3.3.3 Опис маршрутизації.....	31
3.4 Опис клієнтської частини програми.....	34
3.4.1 Використані бібліотеки та пакети	34
3.4.2 Структура каталогів клієнтської частини.....	35

3.5 Алгоритми взаємодії вебзастосунку	38
3.5.1 Логіка організації сторінок сайту	38
3.5.2 Алгоритми функціоналу програми.....	38
3.6 Висновки за розділом 3.....	41
4 Експлуатація та тестування програми.....	42
4.1 Призначення й умови застосування програми	42
4.2 Виконання програми	43
4.3 Функціональне тестування вебзастосунку	50
4.4 Тестування зручності користування вебзастосунком	54
4.5 Висновки за розділом.....	55
Висновки	56
Перелік джерел посилання	57
Додаток А Технічне завдання	60
Додаток Б Текст програми	64
Додаток В Слайди презентації	75

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

CSS – Cascading Style Sheets;

HTML – Hyper Text Markup Language;

IDE – Integrated Development Environment;

JSX – JavaScript XML;

MERN – MongoDB, Express.js, React, Node.js;

NPM – Node Package Manager;

REST – Representational State Transfer;

SCSS – Sassy CSS;

TS – TypeScript;

БД – база даних.

ВСТУП

Сучасні інформаційні технології відіграють ключову роль у сфері громадського харчування, зокрема у процесах автоматизації замовлень. Вебзастосунки для онлайн-замовлення їжі дають змогу значно спростити взаємодію між клієнтами та закладами харчування, покращуючи якість обслуговування та підвищуючи ефективність бізнесу.

Актуальність теми полягає у зростаючій потребі в цифрових рішеннях для організації та управління замовленнями їжі. Використання вебзастосунків у цій сфері дозволяє зменшити навантаження на персонал, оптимізувати процеси прийому та обробки замовлень, а також покращити зручність для користувачів.

Метою роботи є створення вебзастосунку на клієнт-серверній архітектурі з можливістю автоматизації адміністрування роботи сервісу замовлення піци. Застосунок повинен забезпечувати інтуїтивно зрозумілий інтерфейс для користувачів, а також зручні інструменти для адміністраторів з аналітики.

Особливу увагу приділено використанню технологій React та Express.js, які забезпечують ефективну розробку клієнт-серверних вебзастосунків.

React – це популярна бібліотека, що використовується для розробки динамічних та інтерактивних інтерфейсів користувача. Використовуючи компонентний підхід, вона забезпечує легкість масштабування та підтримки застосунку [1]. Завдяки віртуальному DOM React забезпечує високу продуктивність, а використання TypeScript підвищує надійність коду та спрощує його підтримку. Express.js – це гнучкий фреймворк для Node.js, який використовується для створення серверної частини вебзастосунку. Він забезпечує просту та ефективну обробку HTTP-запитів, що дозволяє швидко реалізовувати API для взаємодії між клієнтською частиною та сервером [2]. React та Express.js разом утворюють потужне рішення для розробки

вебзастосунків, забезпечуючи швидку взаємодію між клієнтом і сервером, масштабованість та зручність у розробці.

Практична значущість роботи полягає у можливості впровадження розробленого вебзастосунку в реальні заклади громадського харчування, що дозволить покращити процес замовлення їжі, підвищити його ефективність і зменшити час обробки замовлень.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд предметної області

Інформаційні технології сьогодення суттєво впливають на сферу автоматизації процесів замовлення їжі. Вебзастосунки для замовлення їжі значно спрощують взаємодію між клієнтами та закладами громадського харчування, підвищуючи зручність користування та оптимізуючи управління замовленнями.

Зростаючий попит на такі рішення обумовлений прагненням ресторанів до автоматизації обробки замовлень, скорочення витрат на обслуговування, покращення взаємодії з клієнтами та можливістю збору аналітичних даних. Сучасні технології, такі як інтеграція з платформами доставки, системи управління доставкою та оптимізація маршрутів, дозволяють значно підвищити ефективність роботи закладів [3].

Даний проєкт спрямований на розробку вебзастосунку з можливістю автоматизації адміністрування роботи сервісу замовлення піци. Головною особливістю системи є інтеграція сучасних технологій для оптимізації взаємодії між клієнтом та закладом. Адміністративна панель дає змогу аналізувати попит та приймати обґрунтовані рішення щодо розвитку сервісу.

Використання сучасних технологій, таких як React для клієнтської частини та Express.js для серверної частини, забезпечує високу продуктивність, масштабованість та зручність у підтримці застосунку.

Таким чином, розробка такого рішення сприятиме вдосконаленню системи управління замовленнями їжі та підвищенню ефективності роботи закладів громадського харчування.

1.2 Потреба програмних рішень

Сучасний ринок закладів громадського харчування активно впроваджує цифрові технології для автоматизації процесів обслуговування клієнтів.

Вебзастосунки для замовлення їжі значно полегшують взаємодію між клієнтом і рестораном, забезпечуючи швидке оформлення замовлення та зменшуючи ризик помилок під час прийому [3].

Основні потреби, що зумовлюють необхідність програмних рішень у сфері онлайн-замовлення піци:

- оптимізація процесу прийому замовлень – традиційні методи є менш ефективними, оскільки потребують участі персоналу та можуть призводити до помилок;

- автоматизація обліку замовлень – вебзастосунок дає змогу зберігати інформацію про замовлення, спрощує аналіз даних та облік продукції;

- зручність для користувачів – клієнти мають можливість переглядати меню, склад страв, ціни та оформляти замовлення без необхідності комунікації з оператором.

Розробка вебзастосунку для замовлення піци сприяє оптимізації роботи закладу та підвищенню рівня сервісу, що поліпшує конкурентоспроможність бізнесу.

1.3 Огляд аналогів

1.3.1 Papa John's

Papa John's – система онлайн-замовлення піци, що надає можливість вибору розміру, основних інгредієнтів та додаткових опцій. Платформа також підтримує програму лояльності та дозволяє оформити замовлення з доставкою або самовивозом.

Приклад інтерфейсу можна побачити на рисунку 1.1 [4].

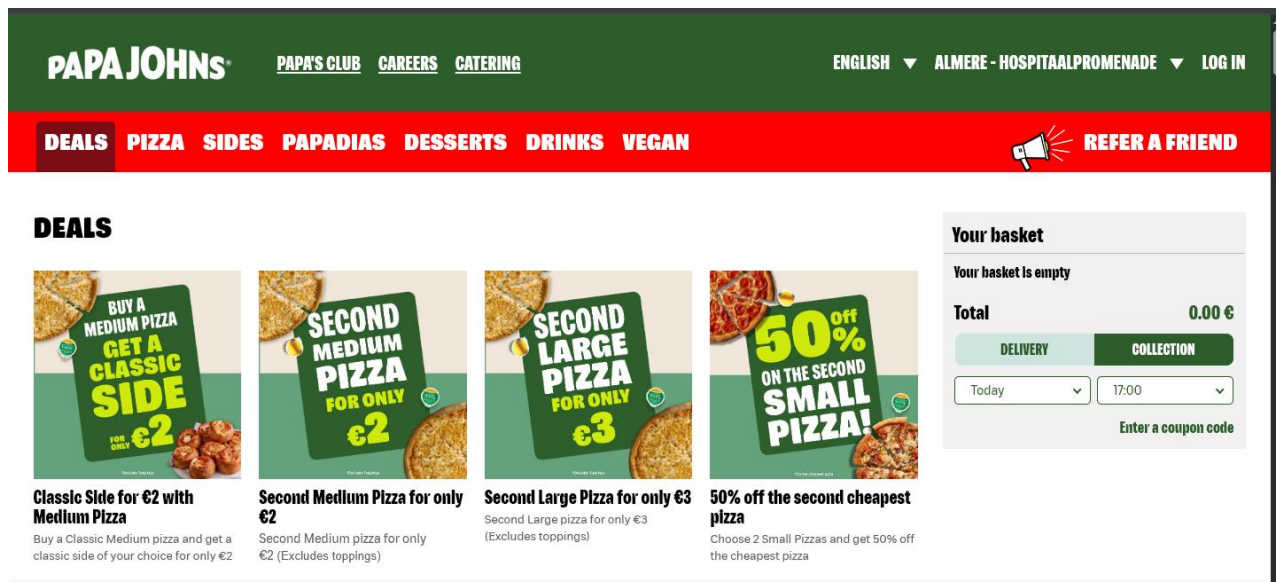


Рисунок 1.1 – Онлайн-система Papa John's [4]

Переваги:

– гнучке налаштування складових піци (вибір розміру, тіста, додаткових інгредієнтів);

– система бонусів та знижок для зареєстрованих користувачів;

– підтримка попереднього замовлення на певний час.

Недоліки:

– ускладнений процес реєстрації та авторизації;

– повільне завантаження сайту в пікові години;

– перевантажений інтерфейс через велику кількість акційних пропозицій.

1.3.2 Domino's Pizza

Domino's Pizza – система онлайн-замовлення піци з можливістю вибору розміру, тіста та додаткових інгредієнтів.

Приклад інтерфейсу можна побачити на рисунку 1.2 [5].

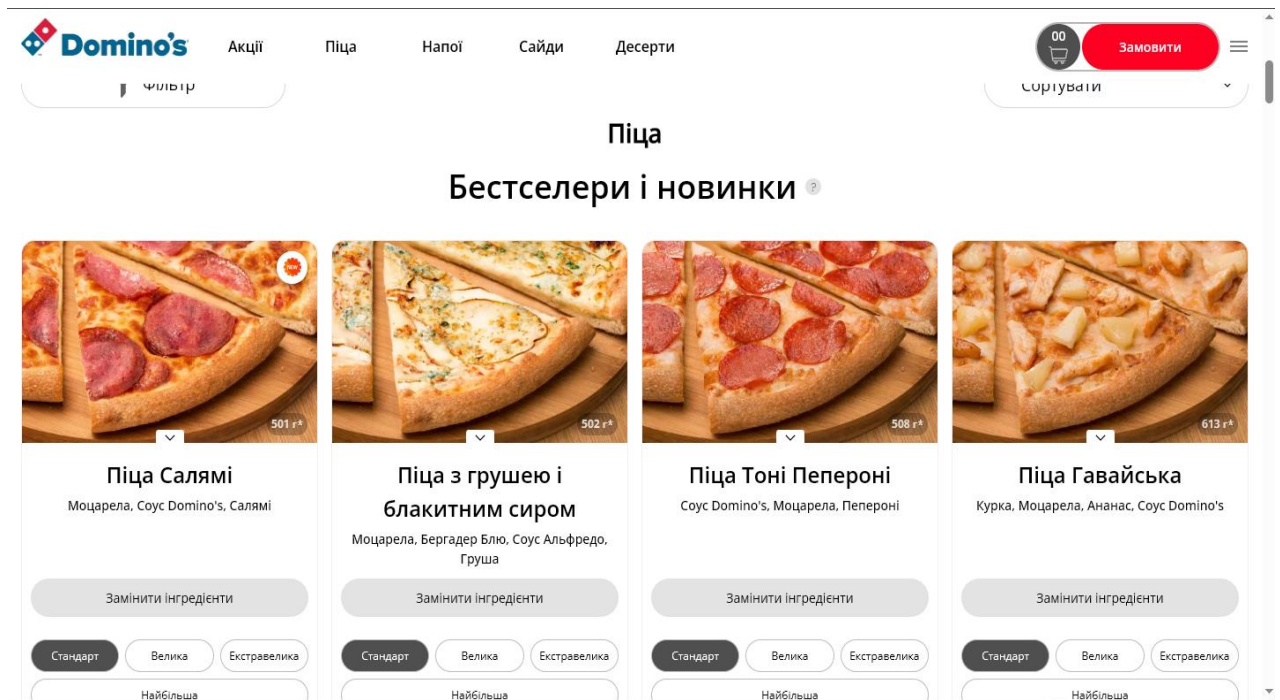


Рисунок 1.2 – Онлайн-система Domino's Pizza [5]

Переваги:

- зручне налаштування піци з великою кількістю можливих модифікацій;
- система бонусів та знижок для зареєстрованих користувачів;
- підтримка попереднього замовлення на певний час.

Недоліки:

- ускладнений процес реєстрації та авторизації;
- перевантажений інтерфейс через велику кількість акційних пропозицій;
- незначна затримка при завантаженні сторінок.

1.3.3 Pizza Hut

Pizza Hut – система онлайн-замовлення піци, де можна налаштувати розмір, тісто та додаткові інгредієнти. Замовлення можна зробити як на доставку, так і на самовивіз.

Приклад інтерфейсу можна побачити на рисунку 1.3 [6].

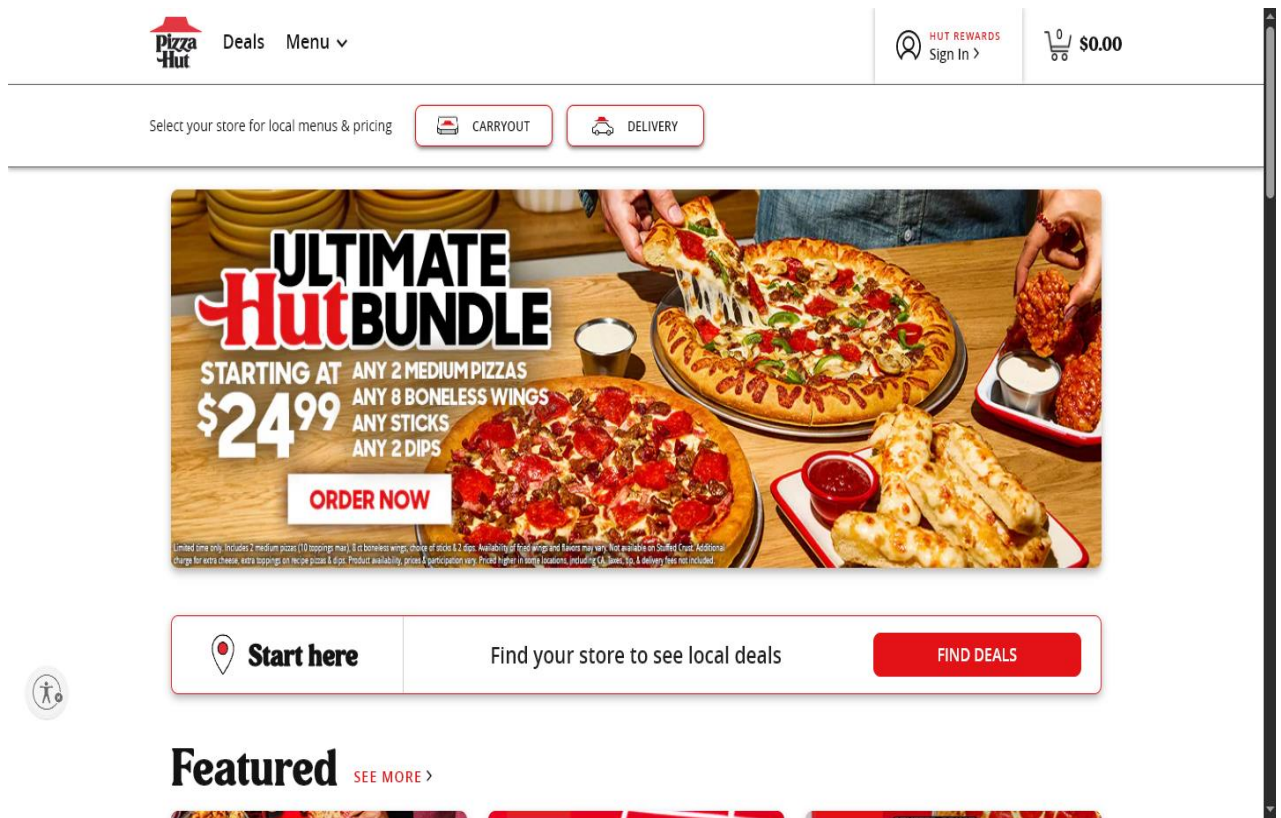


Рисунок 1.3 – Онлайн-система Pizza Hut [6]

Переваги:

- простий та зрозумілий інтерфейс для вибору піци та налаштування її складових;
- наявність програми лояльності для постійних клієнтів;
- адаптивність сайту для різних пристроїв.

Недоліки:

- відсутність широкої аналітики для адміністраторів;
- система більше орієнтована на глобальну мережу, а не на локальні заклади;
- деякі функції доступні лише у мобільному додатку, що може бути незручним для користувачів вебверсії.

1.3.4 Порівняння аналогів

Для порівняння було створено таблицю, в якій будуть основні критерії порівняння платформ для замовлення піци.

Виконаємо порівняння існуючих аналогів за такими критеріями, які наведені у таблиці 1.1.

Таблиця 1.1 – Порівняння аналогів

Критерій	Papa John's	Domino's Pizza	Pizza Hut
Гнучкість налаштування складових піци	Висока	Висока	Середня
Реєстрація та авторизація користувачів	Присутня	Присутня	Присутня
Наявність реклами	Присутня	Присутня	Відсутня
Зручність інтерфейсу	Незручний	Зручний	Зручний
Можливість вибору часу отримання замовлення	Присутня	Присутня	Присутня
Швидкість додавання до кошика	Низька	Низька	Висока

Після проведеного аналізу існуючих рішень видно, що існує досить багато онлайн платформ для замовлення піци. Проте, деякі платформи містять

надмірну рекламну інформацію або мають не зручний інтерфейс, що ускладнює їх використання. Наприклад, на деяких сайтах процес додавання піци до кошика займає більше часу, ніж хотілося б, що може відбити бажання користувача завершити замовлення.

1.4 Висновки за розділом

У розділі було проведено аналіз предметної області. Було розглянуто потребу в програмних рішеннях для оптимізації процесу замовлення. Також здійснено огляд та порівняння існуючих онлайн платформ для замовлення піци, виявлено їх переваги та недоліки. Після чого було сформульовано основні функціональні вимоги до розробки системи.

2 ОБРАННЯ ЗАСОБІВ ТА СЕРЕДОВИЩА РОЗРОБКИ

2.1 Вибір мови програмування

Вибір мови програмування для розробки програмного забезпечення є важливим кроком, що впливає на ефективність, продуктивність та подальшу масштабованість проєкту. Необхідно враховувати не лише відповідність мови до поставленого завдання, але й такі аспекти, як простота масштабування, наявність документації, активність спільноти та доступність бібліотек.

Популярність мови відіграє значну роль, оскільки дозволяє легше знайти розробників, вирішувати технічні питання та використовувати перевірені інструменти.

На сучасному ринку веброзробки значну популярність мають такі мови програмування, як TypeScript, PHP та C#. Вони широко використовуються для створення різних вебдодатків та мають свої сильні та слабкі сторони.

Вибір мови програмування представлено у вигляді таблиці 2.1 виконавши порівняння.

Таблиця 2.1 – Порівняння мов програмування

Критерій	C#	TypeScript	PHP
Гнучкість та масштабованість	Високий рівень	Високий рівень	Середній рівень
Строгість синтаксису	Високий рівень	Високий рівень	Низький рівень
Робота з асинхронними процесами	Високий рівень	Високий рівень	Низький рівень
Активність та підтримка спільноти розробників	Високий рівень	Високий рівень	Високий рівень
Оптимізованість для веброзробки	Середній рівень	Високий рівень	Високий рівень

Для розробки програмного забезпечення обрано мову програмування TypeScript, яка є строго типізованою надмножиною JavaScript і широко застосовується для створення вебзастосунків. TypeScript розширює можливості JavaScript, додаючи статичну типізацію, що дозволяє виявляти потенційні помилки ще на етапі компіляції, підвищуючи надійність та зрозумілість коду. Він компілюється у стандартний JavaScript, що забезпечує сумісність із будь-яким середовищем, яке підтримує JavaScript [7].

Використання TypeScript під час розробки вебзастосунків сприяє суворій типізації, що допомагає виявляти та усувати помилки на ранніх етапах розробки, покращуючи якість програмного забезпечення [7].

Таким чином, застосування TypeScript у проєкті сприятиме створенню більш надійного та масштабованого коду, що є важливим для успішної реалізації сучасних вебзастосунків.

2.2 Вибір фреймворку для фронтенду

Фреймворк – це набір інструментів, бібліотек і правил, які допомагають створювати програми з єдиною структурою, спрощуючи процес розробки та забезпечуючи стандартизовану архітектуру [8]. Використання фреймворку дозволяє зменшити час розробки, спростити підтримку проєкту та уникнути багатьох поширених помилок, оскільки більшість стандартних функцій уже реалізовано.

У розробці вебзастосунків фреймворки відіграють ключову роль, оскільки вони дозволяють зручно працювати з компонентами інтерфейсу, керувати станом застосунку та ефективно оновлювати вміст сторінки без повного перезавантаження. Особливо це важливо для сучасних динамічних вебзастосунків, які повинні швидко реагувати на дії користувача. Серед найпопулярніших фреймворків для створення фронтенду вебзастосунків виділяються React, Vue та Angular. Кожен із них має свої особливості, переваги

та обмеження. Вибір відповідного інструменту залежить від вимог проєкту, необхідної гнучкості, продуктивності, а також легкості навчання.

Для визначення найкращого варіанту було проведено порівняння React, Vue та Angular за ключовими критеріями: продуктивність, гнучкість, складність навчання, популярність, підтримка та складність налаштування.

Результати порівняння представлені у таблиці 2.2.

Таблиця 2.2 – Порівняння фреймворків для фронкенду

Критерій	React	Vue	Angular
Продуктивність	Висока, завдяки Virtual DOM	Висока, оптимізований рендеринг	Висока, проте складніша структура
Гнучкість	Висока, легко інтегрується з іншими бібліотеками	Висока, підтримує модульність	Обмежена, має жорстку структуру
Кривина навчання	Середня, необхідно знати JavaScript і JSX	Низька, простий синтаксис	Висока, використання TypeScript і додаткових концепцій
Популярність	Дуже висока, найбільш використовуваний фреймворк	Висока, але поступається React	Висока, підтримується великими компаніями
Підтримка	Активна спільнота, регулярні оновлення від Meta	Висока, розвивається командою Vue	Висока, підтримується Google

Продовження таблиці 2.2

Критерій	React	Vue	Angular
Складність налаштування	Низька, швидке встановлення через npm	Низька, легка інтеграція	Висока, потребує додаткової конфігурації

Після аналізу фреймворків для розробки інтерфейсу було обрано React. Основними факторами такого вибору є:

- висока продуктивність – завдяки Virtual DOM React швидко оновлює інтерфейс, що важливо для динамічних вебзастосунків;

- гнучкість – він легко інтегрується з іншими бібліотеками та підходить як для малих, так і для великих проєктів. Код на React легкий для розуміння та підтримки, а модульна структура робить його гнучким та сумісним з іншими фреймворками [9];

- популярність і підтримка – велика спільнота, що сприяє швидкому вирішенню проблем, широкий вибір готових рішень та регулярні оновлення роблять React стабільним та перспективним вибором [9];

- зручна робота з станом – можливість ефективного керування даними через Redux або Context API.

2.3 Вибір фреймворку для бекенду

Для реалізації серверної частини проєкту необхідно обрати фреймворк для бекенду, який забезпечить швидку та зручну розробку API, ефективну обробку запитів і стабільну роботу застосунку. Бекенд-фреймворки надають набір інструментів для роботи з маршрутизацією, обробкою запитів, автентифікацією користувачів та взаємодією з базами даних.

Результати порівняння представлені у таблиці 2.3.

Таблиця 2.3 – Порівняння фреймворків для бекенду

Критерій	Express	NestJS	Django
Простота та легкість	Мінімалістичний, легко вивчити	Більш складний, потребує вивчення архітектури	Вимагає знання Django ORM та структури
Гнучкість	Дає повний контроль, легко кастомізується	Має модульну архітектуру	Обмежена структура, менше можливостей для кастомізації
Продуктивність	Висока, завдяки легковісній архітектурі	Добра продуктивність завдяки TypeScript	Менша швидкість через Python та вбудовані механізми
Популярність	Широко використовується у Node.js-розробці	Зростає популярність серед TypeScript-розробників	Велика спільнота Python-розробників
Підтримка та розширюваність	Величезна кількість прм-пакетів	Підтримка модулів та TypeScript	Великий набір вбудованих рішень
Швидкість розробки	Простий у налаштуванні, швидкий старт	Потребує конфігурації та вивчення декораторів	Багато вбудованих механізмів, які потрібно освоїти

Для розробки бекенду проєкту було обрано Express, так як він:

- легкий і мінімалістичний – дозволяє швидко розпочати розробку, зберігаючи повний контроль над логікою застосунку;
- гнучкий – не нав'язує жорсткої структури, що дає змогу інтегрувати різноманітні бібліотеки та бази даних, адаптуючи застосунок під специфічні вимоги проєкту [2];
- продуктивний – працює на Node.js, що робить його чудовим вибором для масштабованих вебдодатків;
- популярний – має активну спільноту та велику кількість готових npm-пакетів для розширення функціональності.

2.4 Вибір бази даних

База даних відіграє ключову роль у будь-якій вебсистемі, оскільки забезпечує збереження та обробку інформації. Вибір відповідного типу бази даних залежить від низки факторів, таких як продуктивність, масштабованість, можливості інтеграції, простота налаштування та модель зберігання даних.

Сучасна веброзробка активно використовує як реляційні (SQL), так і нереляційні (NoSQL) бази даних. Реляційні СУБД працюють із таблицями та чіткими зв'язками між даними, що робить їх ідеальними для систем, де важлива структурованість і цілісність даних [10]. Нереляційні бази даних пропонують більшу гнучкість та краще підходять для обробки великих обсягів неструктурованої інформації.

Для порівняння було обрано три популярні бази даних: MySQL, PostgreSQL (реляційні) та MongoDB (документно-орієнтована NoSQL база).

У таблиці 2.4 наведено порівняння цих середовищ за зазначеними критеріями.

Таблиця 2.4 – Порівняння баз даних

Критерій	MySQL	PostgreSQL	MongoDB
Тип	Реляційна (SQL)	Реляційна (SQL)	Документо-орієнтована (NoSQL)
Продуктивність	Висока, особливо на читанні	Висока, добре підходить для складних запитів	Висока при роботі з неструктурованими даними
Масштабованість	Висока	Висока	Висока
Гнучкість схеми	Фіксована структура таблиць	Фіксована структура таблиць	Гнучка, схему можна змінювати динамічно
Простота налаштування	Легка	Складніша через додаткові можливості	Легка
Підтримка транзакцій	Так	Так	Так
Інтеграція з Node.js	Добра, через mysql	Добра, через pg	Відмінна, через mongoose

Після проведеного аналізу було вирішено використовувати MongoDB. Цей вибір зумовлений високою продуктивністю, гнучкістю у роботі з динамічними структурами даних і простою інтеграцією з Node.js та Express.js за допомогою бібліотеки mongoose. Крім того, підтримка горизонтального масштабування дає змогу ефективно адаптувати систему до зростаючого навантаження, що є критично важливим для сучасних вебзастосунків із великими обсягами даних.

2.5 Вибір середовища розробки

Для ефективної розробки програмного забезпечення важливо обрати зручне середовище розробки (IDE), оскільки воно впливає на продуктивність роботи. Середовище забезпечує автодоповнення коду, підтримку налагодження, інтеграцію з системами контролю версій та зручний інтерфейс для роботи з проєктом.

При виборі середовища розробки враховувалися такі основні критерії: швидкодія, функціональність, інтеграція з технологіями проєкту, зручність використання, популярність, підтримка, вартість. Найпоширенішими рішеннями для веброзробки є Visual Studio Code, WebStorm та Sublime Text.

У таблиці 2.5 наведено порівняння цих середовищ за зазначеними критеріями.

Таблиця 2.5 – Порівняння середовищ

Критерій	Visual Studio Code	WebStorm	Sublime Text
Швидкодія	Висока	Висока	Дуже висока
Функціональність	Висока (через розширення)	Вбудовані інструменти	Середня (основні функції)
Інтеграція з технологіями проєкту	Повна підтримка JavaScript, TypeScript, React, Express.js	Оптимізовано для веброзробки	Обмежена підтримка
Зручність використання	Гнучке налаштування, велика кількість плагінів	Інтуїтивний інтерфейс	Мінімалістичний, але швидкий
Популярність	Дуже висока	Висока	Середня

Продовження таблиці 2.5

Критерій	Visual Studio Code	WebStorm	Sublime Text
Підтримка	Велика спільнота, активна документація	Офіційна підтримка JetBrains	Менша кількість оновлень та розширень
Вартість	Безкоштовне	Платне	Платне

Серед доступних середовищ розробки було обрано Visual Studio Code як основне, оскільки воно безкоштовне, має розширену екосистему плагінів та підтримує всі необхідні технології (React, Express.js, TypeScript). До того ж, його популярність та активна підтримка спільноти роблять його найкращим вибором для веброзробки.

2.6 Висновки за розділом

У цьому розділі було обрано основні технології для розробки вебзастосунку. У якості мови програмування було обрано TypeScript. Для клієнтської частини використано React, для серверної — Express.js. Як базу даних вибрано MongoDB. Також, як середовище розробки було обрано Visual Studio Code. Такий вибір технологій дозволяє створити ефективний, зручний та масштабований вебзастосунок.

3 СТРУКТУРА ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

3.1 Структурна схема розробки

Загальна схема роботи даного вебзастосунку включає три основні компоненти наведених нижче.

Клієнтська частина реалізована на основі React із використанням SCSS, TypeScript та Redux Toolkit для зручного управління станом додатка. Вона забезпечує відображення динамічного контенту, обробку користувацьких дій та взаємодію з сервером через HTTP-запити.

Серверна частина вебзастосунку отримує запити від клієнтської частини та обробляє їх. Вона виконує різні операції, такі як витягування, оновлення, збереження або видалення даних з бази даних. Серверна частина створена з використанням фреймворку Express.js, а для взаємодії з базою даних використана бібліотека Mongoose.

База даних на MongoDB дозволяє ефективно зберігати та обробляти великі обсяги даних завдяки гнучкій схемі BSON-документів. Це забезпечує швидкий доступ до інформації та зручне масштабування, що робить її ідеальним вибором для збереження даних без необхідності жорсткої структури. MongoDB підтримує високу продуктивність та ефективно працює з динамічними даними.

Схематичне зображення схеми роботи вебзастосунку представлено на рисунку 3.1.

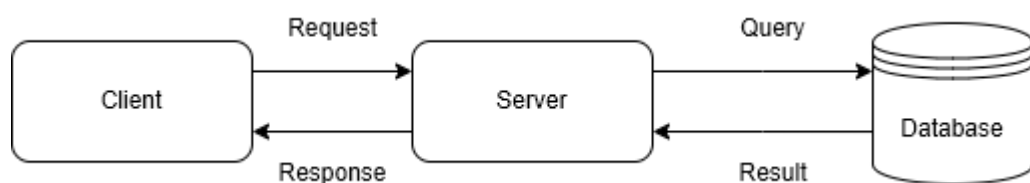


Рисунок 3.1 – Загальна схема роботи вебзастосунку

3.2 Архітектурний підхід у додатку

У процесі розробки вебзастосунку було обрано архітектурний підхід, заснований на шаблоні MVC (Model-View-Controller) у поєднанні з стеком MERN (MongoDB, Express.js, React, Node.js).

Архітектурний підхід MVC – це архітектурний шаблон для розробки програмного забезпечення, який допомагає розділити програму на три основні частини: модель (Model), представлення (View) та контролер (Controller) [11].

Model (Модель) відповідає за роботу з базою даних та бізнес-логікою, збереження та обробку інформації. View (Представлення) відображає дані користувачеві та забезпечує взаємодію з інтерфейсом. Controller (Контролер) забезпечує зв'язок між моделлю та представленням, отримуючи дані від користувача, передаючи їх у модель та оновлюючи інтерфейс відповідно до змін [12].

Однією з важливих особливостей цієї архітектури є її циклічність: користувач взаємодіє з представленням, яке надсилає запити до контролера, а той, у свою чергу, звертається до моделі для роботи з даними. Після цього оновлена інформація повертається у представлення, і процес повторюється [12].

Схематичне зображення схеми роботи архітектурного шаблону MVC представлено на рисунку 3.2.

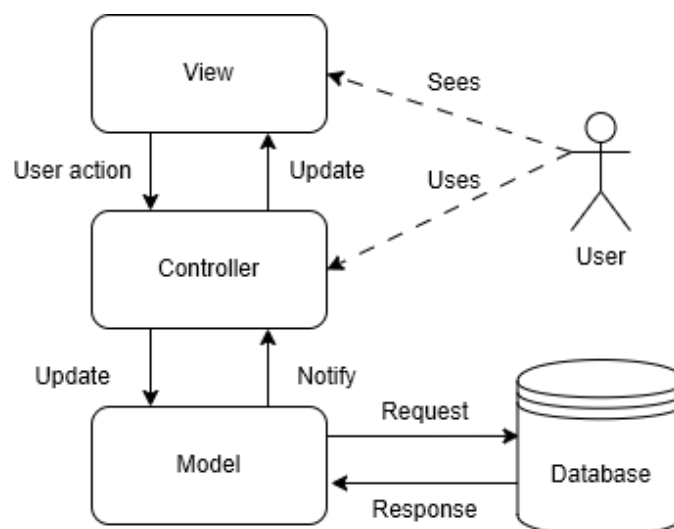


Рисунок 3.2 – Діаграма архітектурного шаблону MVC

Проект, побудований на основі стеку MERN та архітектурного шаблону MVC, включає три основні компоненти.

Модель (Model) реалізована за допомогою MongoDB – бази даних, що визначає структуру даних та надає методи для взаємодії з нею. Взаємодія з нею здійснюється через бібліотеку Mongoose, що спрощує маніпуляції з даними [12].

Представлення (View) будується за допомогою React популярної бібліотеки для створення користувацьких інтерфейсів. Компоненти React визначають елементи інтерфейсу, а JSX (JavaScript XML) використовується для опису їх структури та зовнішнього вигляду. Представлення отримує дані з моделі та відображає їх у зручному для користувача форматі [13].

Контролер (Controller) побудований на основі Express.js, який працює на Node.js. Він приймає HTTP-запити від клієнта, передає їх моделі для обробки та відправляє отриману відповідь назад у представлення [13].

Завдяки використанню MERN-стеку, така архітектура забезпечує високу продуктивність, масштабованість та спрощує розробку багатофункціонального вебзастосунку.

3.3 Опис серверної частини програми

3.3.1 Використані бібліотеки та пакети

У процесі розробки серверної частини вебзастосунку було використано кілька ключових бібліотек та пакетів, встановлених за допомогою менеджера пакетів npm.

npm (Node Package Manager) – це стандартний менеджер пакетів для JavaScript, який автоматично встановлюється разом із Node.js. Він використовується для завантаження та управління бібліотеками та залежностями, необхідними для проєкту [14].

Нижче наведено основні використані залежності:

- jsonwebtoken: бібліотека для створення та верифікації JSON Web Tokens (JWT) [15];
- express: фреймворк для Node.js, який використовується для створення серверних застосунків [2];
- bcryptjs: бібліотека для хешування паролів, використовує алгоритм Blowfish [16];
- mongoose: бібліотека для роботи з MongoDB [17];
- dotenv: бібліотека, яка завантажує змінні середовища з файлу .env;
- nodemon: інструмент, який автоматично перезапускає сервер Node.js при внесенні змін у вихідний код;
- cors: пакет, який дозволяє налаштовувати політику крос-доменного доступу (CORS) у вебзастосунках [18].

3.3.2 Структура каталогів серверної частини

У процесі розробки серверної частини застосунку (рис. 3.3) було організовано файлову структуру, що сприяє зручності підтримки та масштабованості проєкту. Нижче наведено основні каталоги та їхнє призначення:

- controllers: каталог з файлами, які обробляють вхідні запити, взаємодіють із моделями та повертають відповіді клієнту;
- database: каталог з файлами та модулями, пов'язаними з роботою з базою даних;
- middleware: каталог з проміжними обробниками для обробки запитів;
- models: каталог з файлами, які визначають структуру даних і взаємодіють з базою даних;
- node_modules: каталог, де зберігаються всі сторонні пакети та бібліотеки, встановлені за допомогою npm. Цей каталог створюється автоматично при встановленні залежностей проєкту;

– routes: каталог, що містить файли, які визначають маршрутизацію запитів у застосунку. Тут встановлюється, який контролер оброблятиме певний маршрут, що сприяє чіткому розподілу запитів.

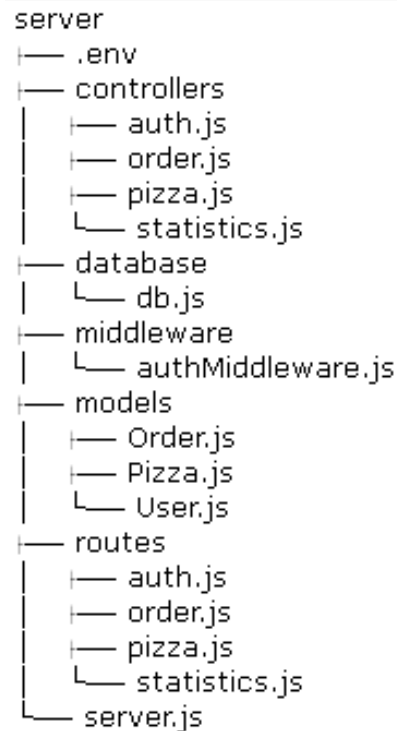


Рисунок 3.3 – Структура каталогів серверної частини проєкту

3.3.3 Опис маршрутизації

Маршрутизація – це процес обробки HTTP-запитів на сервері, який визначає, які ресурси або функції потрібно викликати у відповідь на певний запит. Вона є важливою частиною вебзастосунків, оскільки забезпечує правильний розподіл запитів між різними модулями системи.

Опис основних методів HTTP-запитів:

- GET – призначений для отримання інформації з сервера;
- POST – використовується для надсилання даних та створення нових записів;
- DELETE – застосовується для видалення даних з сервера.

У додатку маршрути згруповані за функціональністю:

- «auth»: для реєстрації та авторизації користувачів;
- «order»: для створення та перегляду замовлень;
- «pizza»: для отримання даних про піци;
- «statistics»: для відображення статистичних даних.

Таблиці 3.1 – 3.3 містять перелік маршрутів для груп auth, order і pizza, опис їх функціонального призначення та відповідні методи запитів:

Таблиця 3.1 – Маршрути групи «auth»

Номер маршруту	Маршрут	Метод	Опис
1	/api/auth/register	POST	Маршрут для реєстрації нового користувача
2	/api/auth/login	POST	Маршрут для входу користувача в систему

Таблиця 3.2 – Маршрути групи «order»

Номер маршруту	Маршрут	Метод	Опис
1	/api/order/	GET	Маршрут для отримання всіх замовлень (тільки адміністратор має доступ)

Продовження таблиці 3.2

Номер маршруту	Маршрут	Метод	Опис
2	/api/order/:userId	GET	Маршрут для отримання замовлень конкретного користувача (тільки адміністратор має доступ)
3	/api/order/add	POST	Маршрут для додавання нового замовлення користувачем

Таблиця 3.3 – Маршрути групи «pizza»

Номер маршруту	Маршрут	Метод	Опис
1	/api/pizza/	GET	Маршрут для отримання списку всіх піц
2	/api/pizza/:id	GET	Маршрут для отримання конкретної піци за її ID

Таблиця 3.4 містить перелік маршрутів для групи statistics, їх функціональне призначення та відповідні методи запитів.

Таблиця 3.4 – Маршрути групи «statistics»

Номер маршруту	Маршрут	Метод	Опис
1	/api/statistics/popular-pizzas	GET	Маршрут для отримання статистики про найпопулярніші піци
2	/api/statistics/users	GET	Маршрут для отримання статистики про замовлення по користувачам

Створення маршрутів у вебдодатку є важливим етапом для організації доступу до різного функціоналу. Такі маршрути дозволяють підтримувати чисту і організовану архітектуру, спрощуючи управління додатком і забезпечуючи зручний доступ до його різних частин.

3.4 Опис клієнтської частини програми

3.4.1 Використані бібліотеки та пакети

Для розробки клієнтської частини програми були використані різноманітні бібліотеки та пакети, встановлені за допомогою менеджера пакетів npm [14]. Вони дозволяють ефективно розв'язувати завдання з обробки даних, побудови інтерфейсу та взаємодії з сервером.

Нижче наведено основні використані залежності:

- react: бібліотека для побудови інтерфейсів користувача, що забезпечує ефективне оновлення віртуального DOM [19];

- react-router-dom: бібліотека для маршрутизації у React, яка дозволяє легко керувати переходами між сторінками, додавати динамічні та вкладені маршрути для створення багатосторінкових додатків [20];
- reduxjs/toolkit: бібліотека спрощує роботу з завданнями та кодом Redux, яка забезпечує зручні інструменти для управління станом додатку [21];
- clsx: бібліотека для умовного об'єднання класів CSS, яка спрощує динамічне додавання класів до елементів [22];
- sass: препроцесор CSS, який додає можливості, такі як змінні, вкладеність та міксини, для більш зручного та структурованого написання стилів [23];
- react-paginate: компонент для реалізації пагінації в React-додатках, який дозволяє зручно розбивати великі списки даних на сторінки для покращення навігації [24];
- react-content-loader: бібліотека для створення скелетного екрану, яка дає змогу відображати тимчасово елементи-заповнювачі під час завантаження контенту [25];
- axios: бібліотека для виконання HTTP-запитів на основі промісів у браузері та Node.js, яка підтримує інтерсептори, таймаути та інші налаштування [26];
- react-toastify: бібліотека, яка дозволяє легко додавати спливаючі повідомлення (тости) до додатку [27].

3.4.2 Структура каталогів клієнтської частини

Для зручності розробки та підтримки клієнтська частина додатку (рис. 3.4) організована у вигляді чіткої структури каталогів. Це забезпечує логічний поділ функціональності, полегшує обслуговування та масштабування проєкту. Нижче наведено основні каталоги та їхнє призначення:

- public – каталог, який містить статичні файли, необхідні для відображення додатку;

- `src` – основний каталог з вихідним кодом програми. У ньому зберігаються всі основні компоненти, стилі, логіка та утиліти;
- `types` – каталог, який використовується для оголошення модулів у TypeScript, що забезпечує коректну типізацію файлів, а також зовнішніх бібліотек;
- `assets` – каталог, що містить зображення, шрифти та інші медіа-файли, що використовуються в інтерфейсі;
- `components` – каталог включає незалежні, багаторазові компоненти, які можна використовувати у різних частинах додатку;
- `layouts` – каталог, що містить компоненти макетів, що визначають загальну структуру сторінок;
- `pages` – каталог з сторінковими компонентами, кожен з яких відповідає окремій сторінці додатку;
- `redux` – каталог, який містить файли управління станом через Redux Toolkit;
- `scss` – каталог включає файли для стилів, написаних мовою SCSS, що забезпечує гнучке налаштування вигляду додатку;
- `utils` – каталог, що містить допоміжні функції, що використовуються у різних частинах програми;
- `node_modules` – каталог з встановленими залежностями Node.js, необхідними для роботи проєкту.

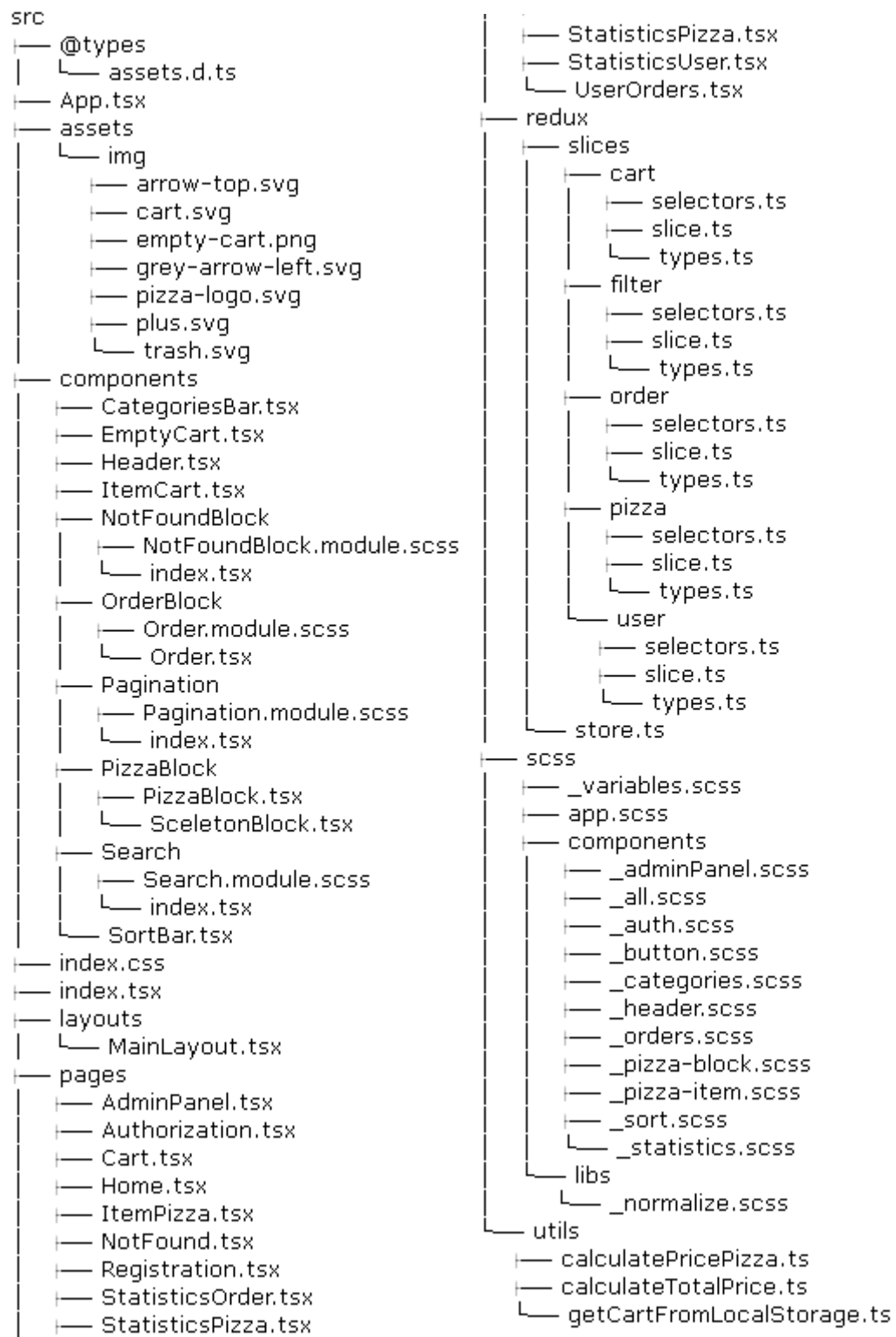


Рисунок 3.4 – Структура каталогів клієнтської частини проекту

3.5 Алгоритми взаємодії вебзастосунку

3.5.1 Логіка організації сторінок сайту

Логіка організації сторінок сайту визначає структуру розділів і взаємодію користувачів із контентом. Вона містить основні елементи навігації, логіку переходів між сторінками та зв'язки між ними.

На рисунку 3.5 представлена схема організації сторінок сайту.

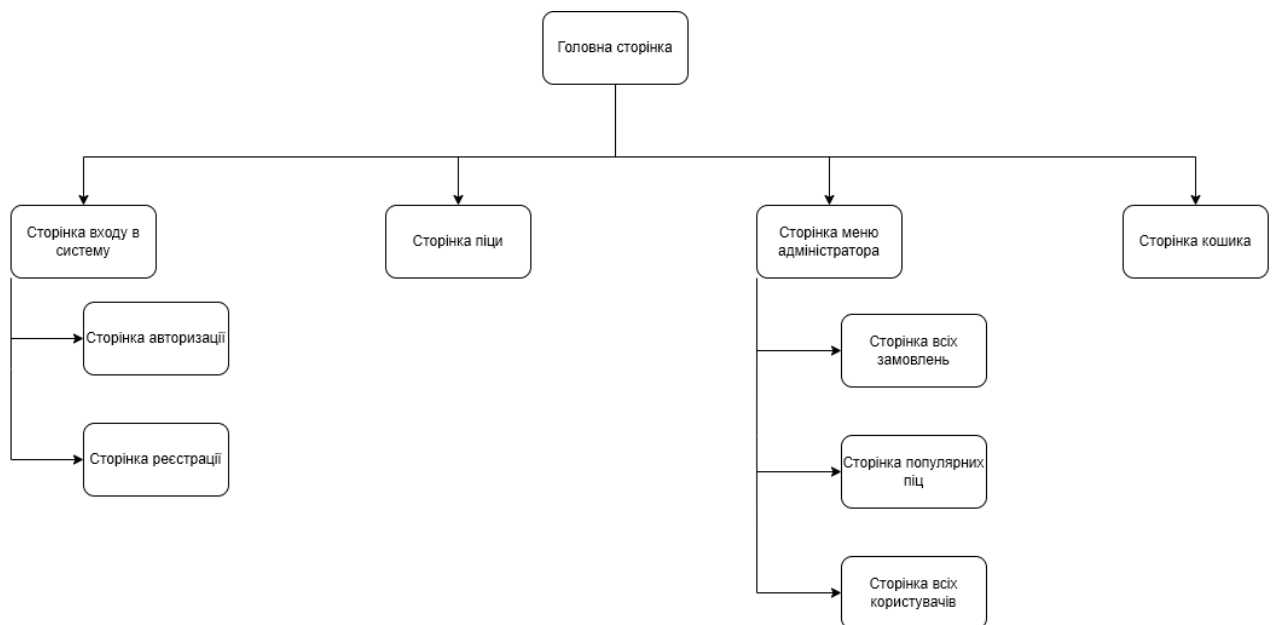


Рисунок 3.5 – Схема організації сторінок сайту

3.5.2 Алгоритми функціоналу програми

Алгоритми функціоналу програми описують ключові процеси взаємодії користувача із системою. Було виокремлено такі функції:

- реєстрація користувача (рис. 3.6);
- авторизація користувача (рис 3.7);
- додавання замовлення (рис. 3.8).

На рисунку 3.6 представлена блок-схема функції реєстрації користувача, яка демонструє основні етапи цього процесу.

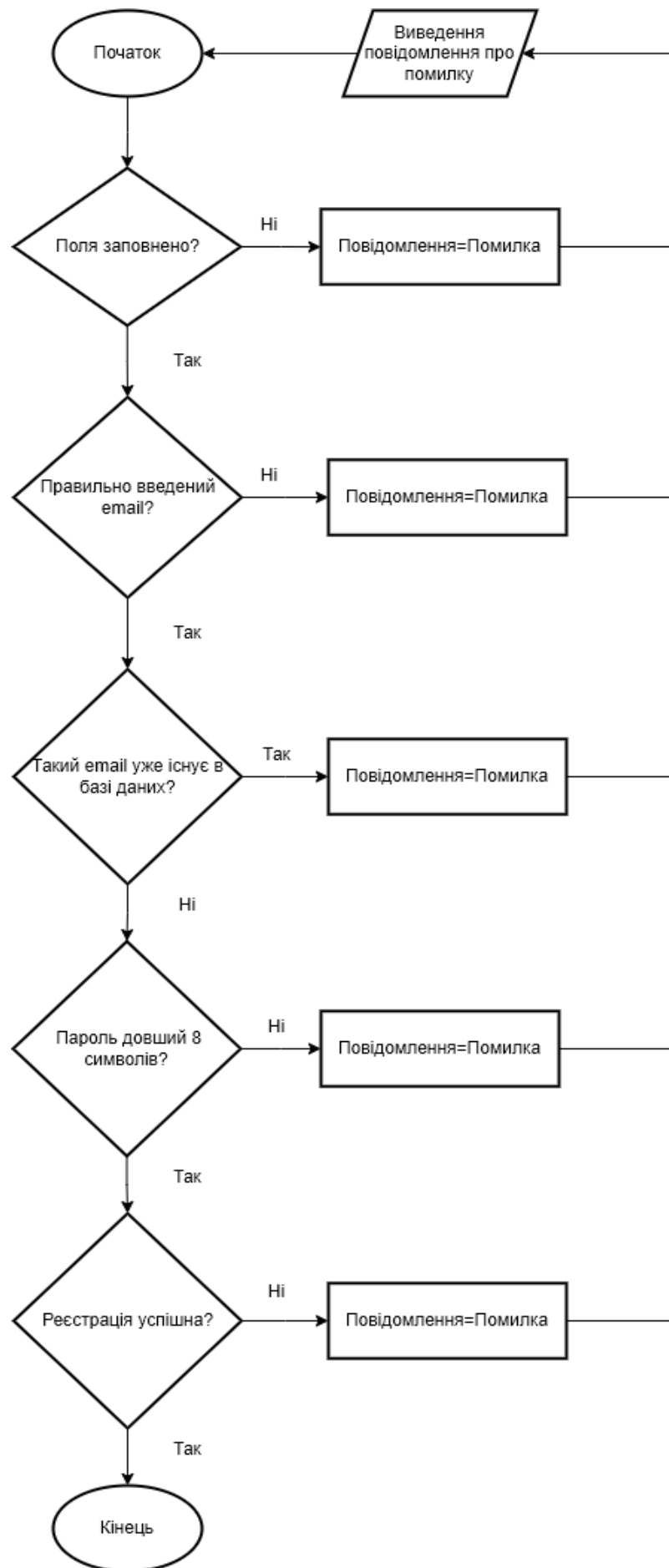


Рисунок 3.6 – Блок схема функції реєстрації користувача

На рисунку 3.7 представлена блок-схема функції авторизації користувача, яка демонструє основні етапи цього процесу.

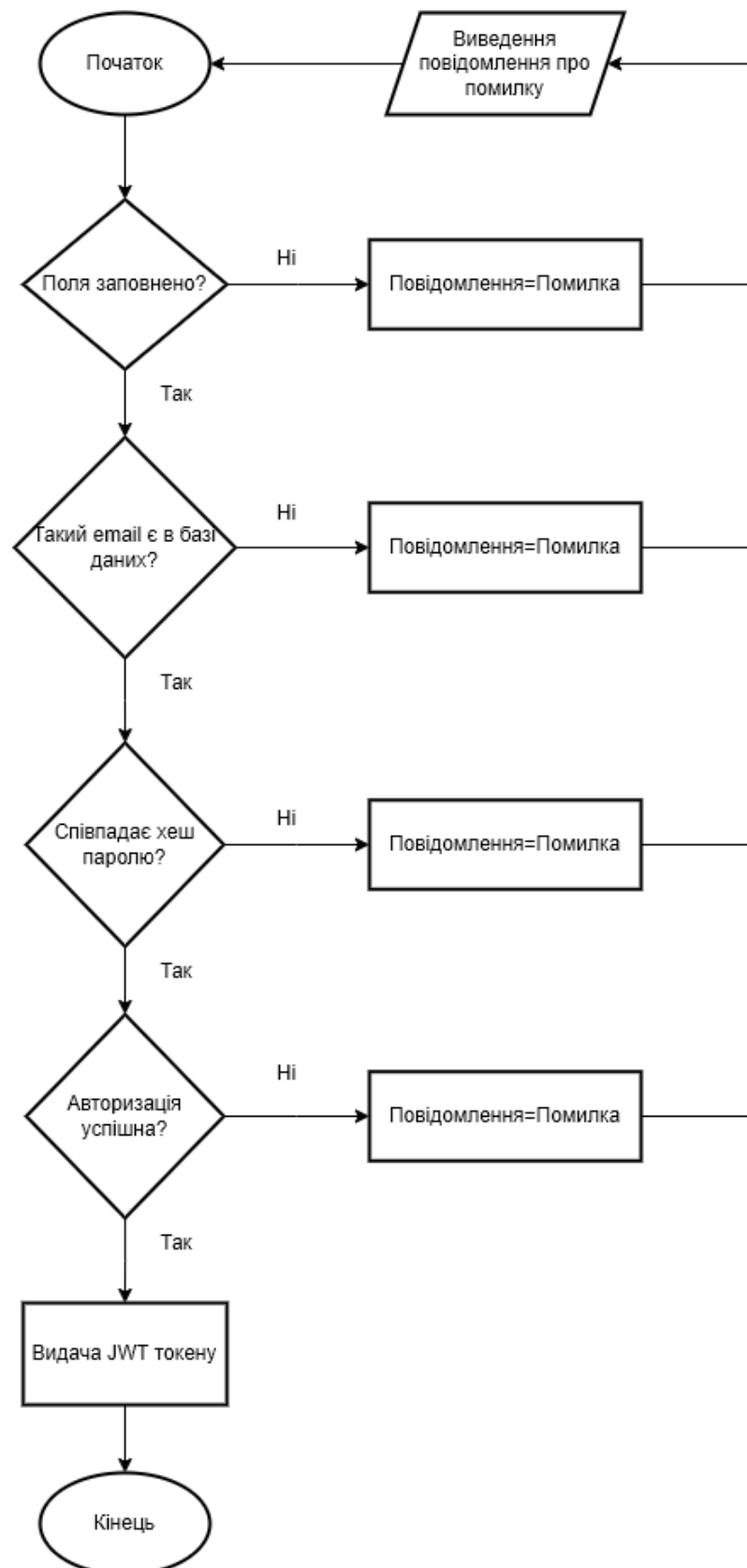


Рисунок 3.7 – Блок схема функції авторизації користувача

На рисунку 3.8 наведена блок-схема функції додавання замовлення користувачем.

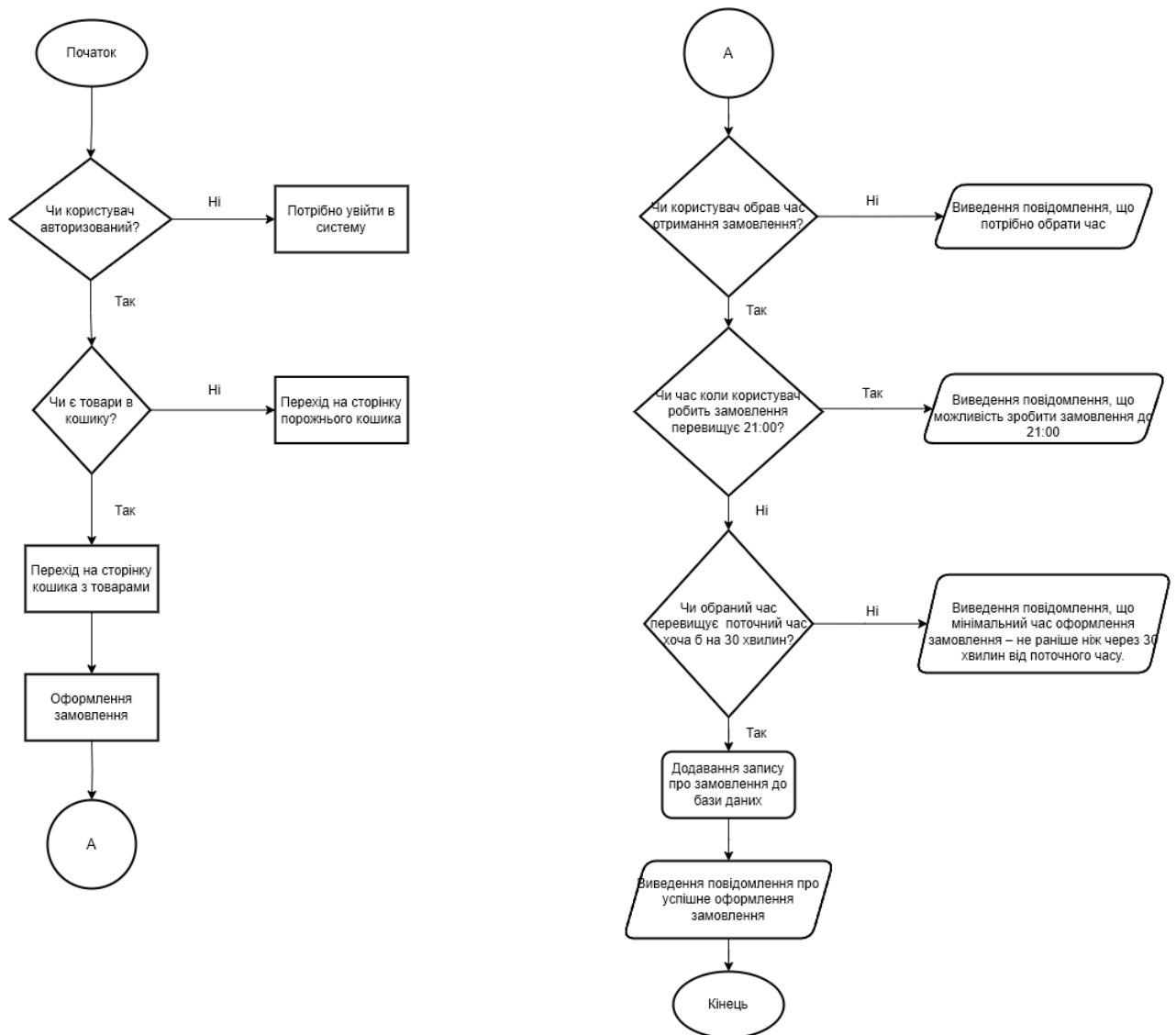


Рисунок 3.8 – Блок схема функції додавання замовлення користувачем

3.6 Висновки за розділом 3

У цьому розділі було розглянуто основні аспекти розробки вебзастосунку. Описано структурну схему розробки, архітектурний підхід. Описані структура каталогів, бібліотек і пакетів серверної та клієнтської частин додатку. А також розглянуто реалізацію структури навігації та сторінок вебдодатку.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програми

Розроблений вебзастосунок реалізує функціонал для автоматизації адміністрування сервісу замовлення піци. Він надає клієнтам можливість переглядати меню, оформлювати замовлення, а адміністраторам – переглядати замовлення, користувачів та відстежувати статистику популярних позицій меню через зручну панель.

Програмне забезпечення має забезпечувати такі функціональні можливості для авторизованого користувача:

- можливість авторизації;
- перегляд меню;
- можливість сортувати меню за категоріями, ціною, алфавітом;
- пошук відповідної позиції з меню по назві;
- додавання піци до кошика;
- оформлення замовлення;
- отримання повідомлень про успішне оформлення замовлення.

Функціональні можливості для неавторизованого користувача:

- можливість реєстрації;
- перегляд меню;
- можливість сортувати меню за категоріями, ціною, алфавітом;
- пошук відповідної позиції з меню по назві.

Адміністратор вебзастосунку повинен мати доступ до наступних функціональних можливостей:

- можливість авторизації;
- перегляд інформації про всі замовлення;
- перегляд інформації про користувачів;
- перегляд статистики за популярними позиціями з меню.

Для роботи серверної частини вебзастосунку необхідно дотримуватись таких мінімальних вимог:

- процесор 2 ядра (Intel Core i3 або еквівалент);
- SSD/ HDD об'ємом від 64 Gb;
- програмне забезпечення: середовище Node.js, фреймворк Express.js, пакетний менеджер npm, база даних MongoDB.

Мінімальні вимоги до апаратного та програмного забезпечення клієнтської частини такі:

- персональний комп'ютер, ноутбук, смартфон або планшет;
- встановлений веббраузер з підтримкою JavaScript;
- оперативна пам'ять 4 ГБ;
- підключення до мережі Internet.

Інтерфейс адаптований для роботи на мобільних пристроях та підтримує браузерні API для швидкої взаємодії з сервером.

4.2 Виконання програми

На головній сторінці користувач має змогу переглядати позиції з меню, здійснювати пошук піци за її назвою, додавати піци до кошика, а також переходити до сторінки авторизації за допомогою відповідної кнопки. Крім цього, реалізована зручна пагінація для перегляду позицій меню, можливість фільтрації за категоріями та сортування за різними параметрами: ціною чи алфавітом.

На рисунку 4.1 інтерфейс головної сторінки, де можна побачити перераховані елементи.

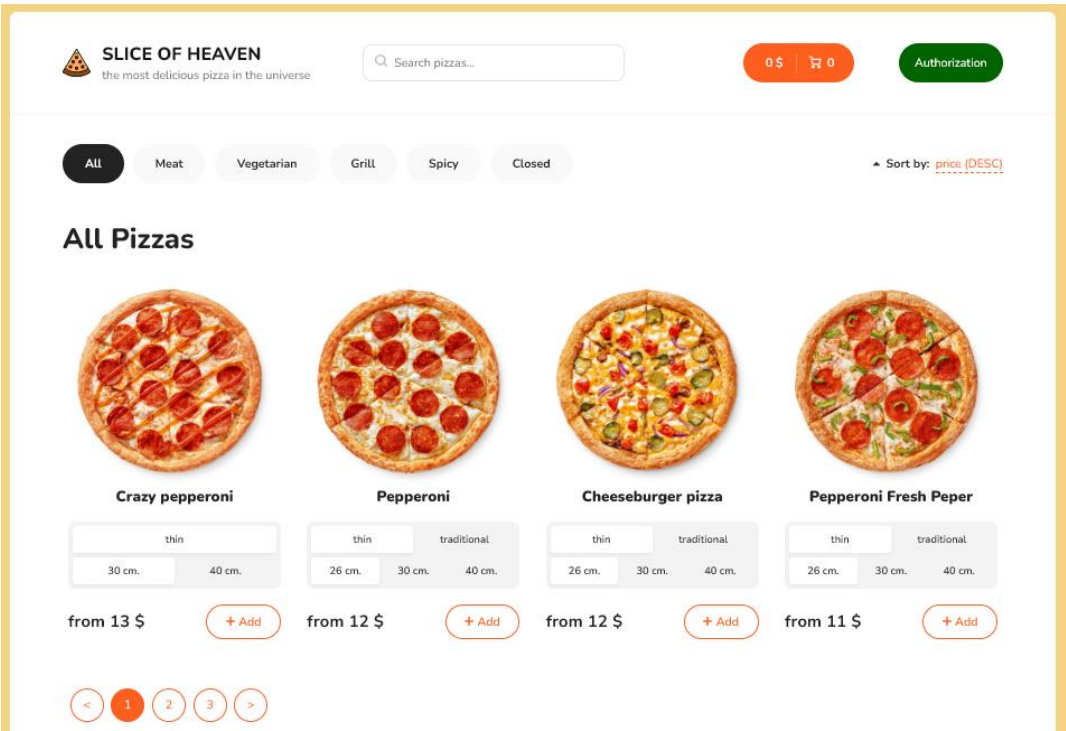


Рисунок 4.1 – Головна сторінка вебзастосунку

Наступним етапом користувач може ознайомитися з детальною інформацією про конкретну піцу, перейшовши на її індивідуальну сторінку. На рисунку 4.2 інтерфейс сторінки детальної інформації про піцу.

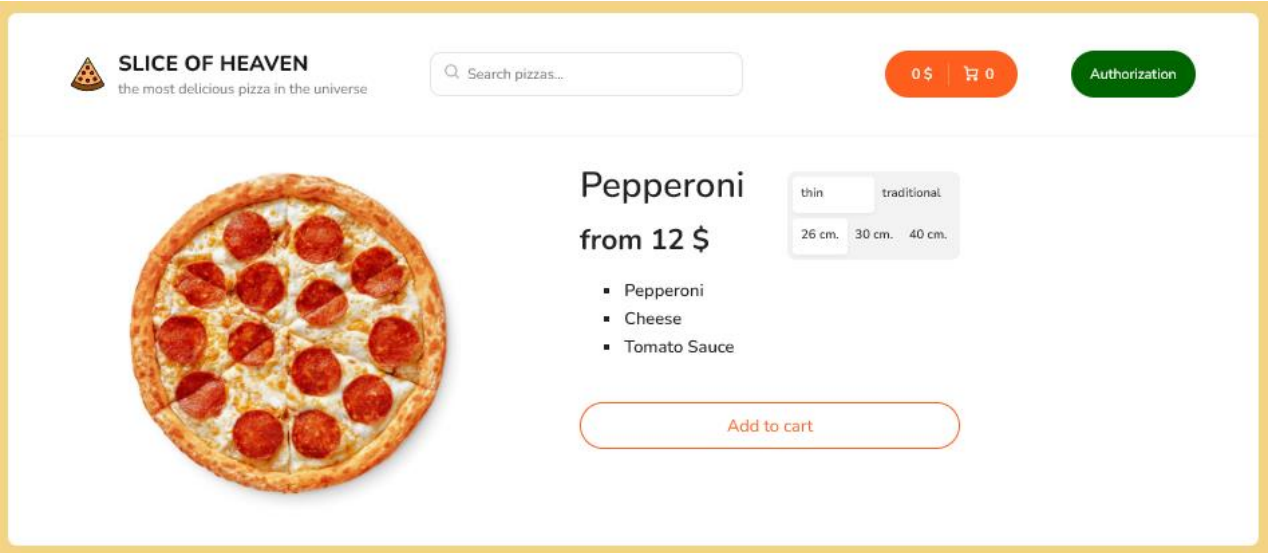


Рисунок 4.2 – Сторінка детальної інформації про піцу

На сторінці піци користувач має можливість обрати бажаний розмір, тип тіста, а також ознайомитися з повним списком інгредієнтів. Після налаштування параметрів піци можна додати її до кошика, натиснувши відповідну кнопку. Такий підхід дозволяє максимально персоналізувати замовлення відповідно до вподобань користувача.

Після додавання піци до кошика користувач може перейти на відповідну сторінку для перегляду замовлення. На рисунку 4.3 зображений інтерфейс сторінки кошика.

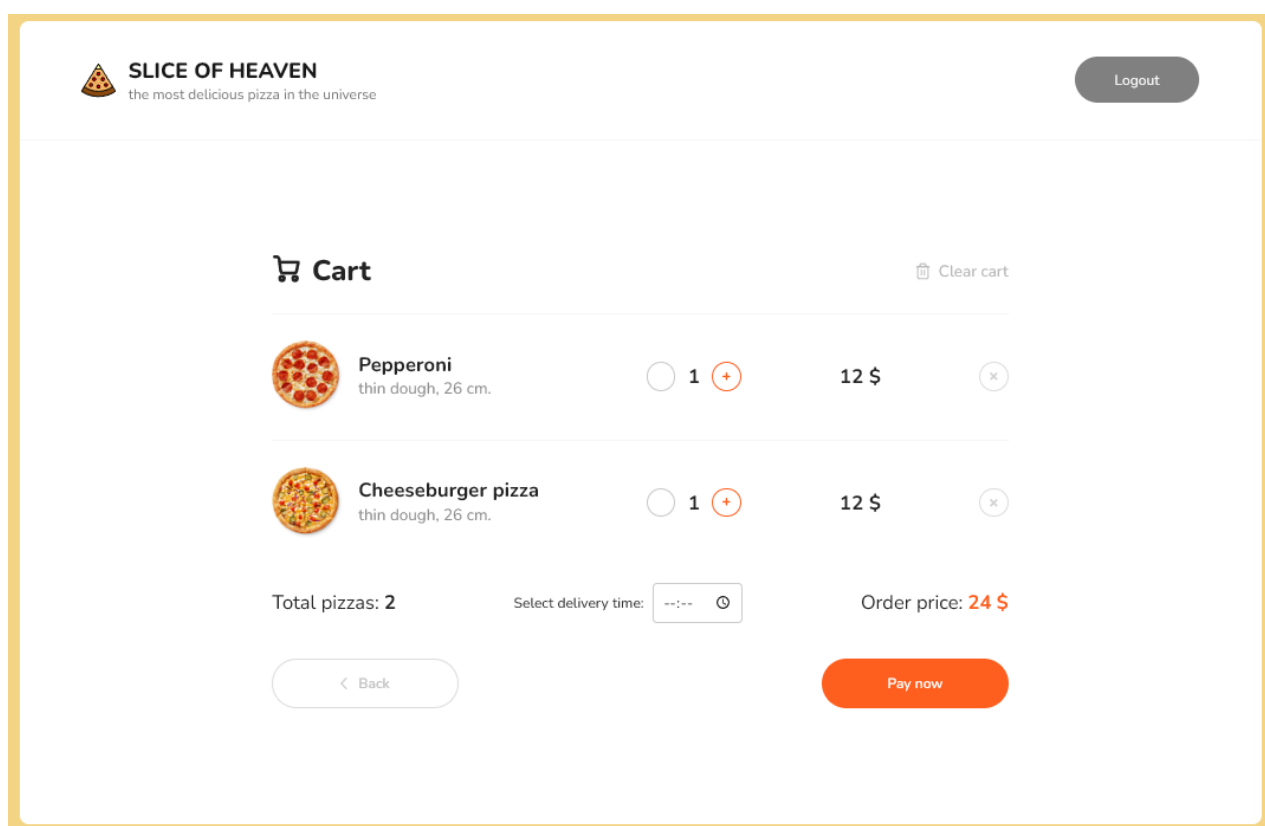


Рисунок 4.3 – Сторінка кошика

На сторінці кошика відображається список усіх вибраних позицій з меню, їх кількість, ціна, а також загальна сума замовлення. Користувач має змогу змінювати кількість обраної піци або повністю видалити позицію зі списку. Окрім цього, є можливість обрати зручний час, коли піца буде готова, щоб самостійно забрати її у зазначений момент. Також доступна кнопка для оформлення замовлення.

У випадку, якщо неавторизований користувач намагається додати піцу до кошика, система автоматично перенаправляє його на сторінку авторизації.

На рисунку 4.4 зображений інтерфейс сторінки для авторизації користувача.

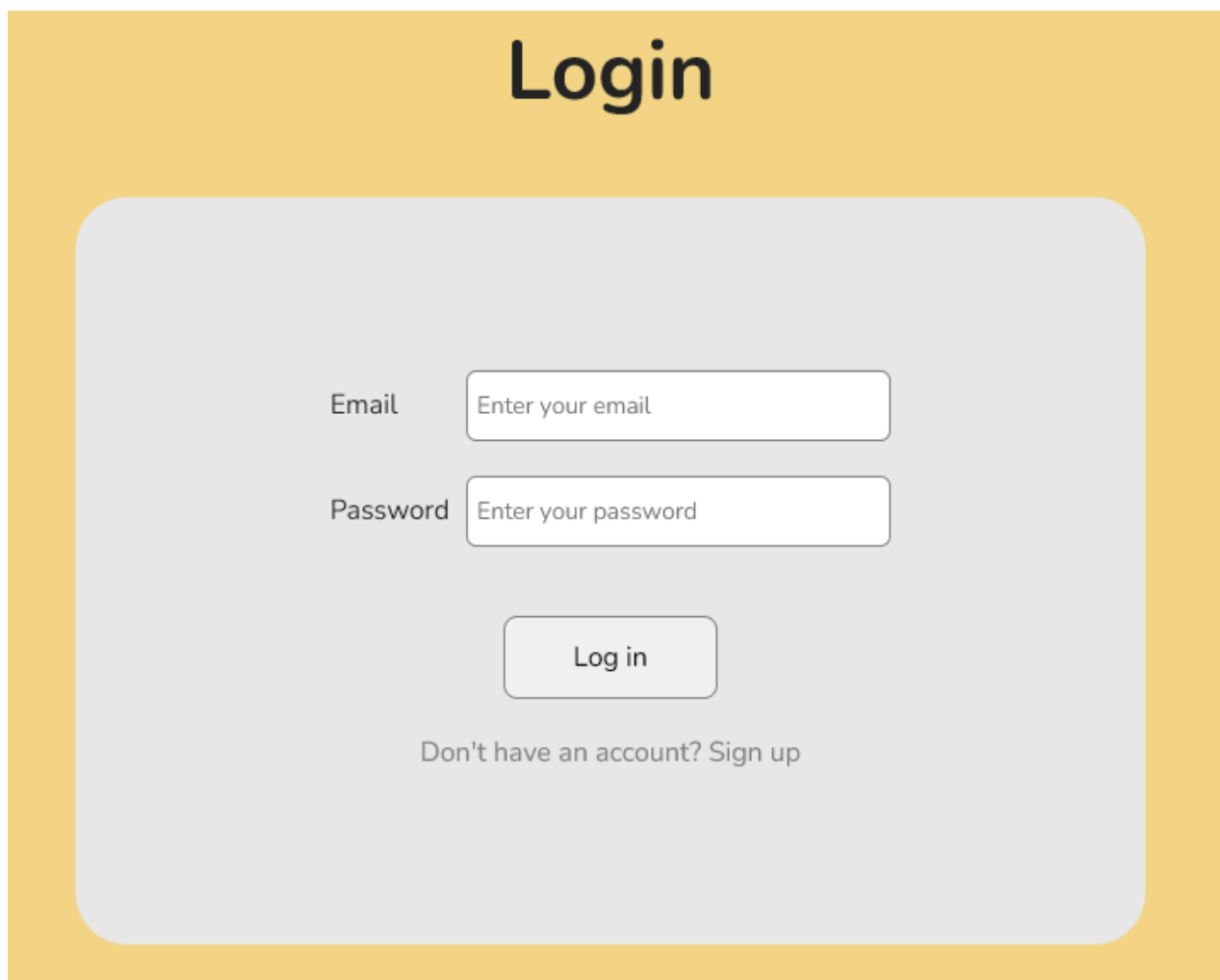
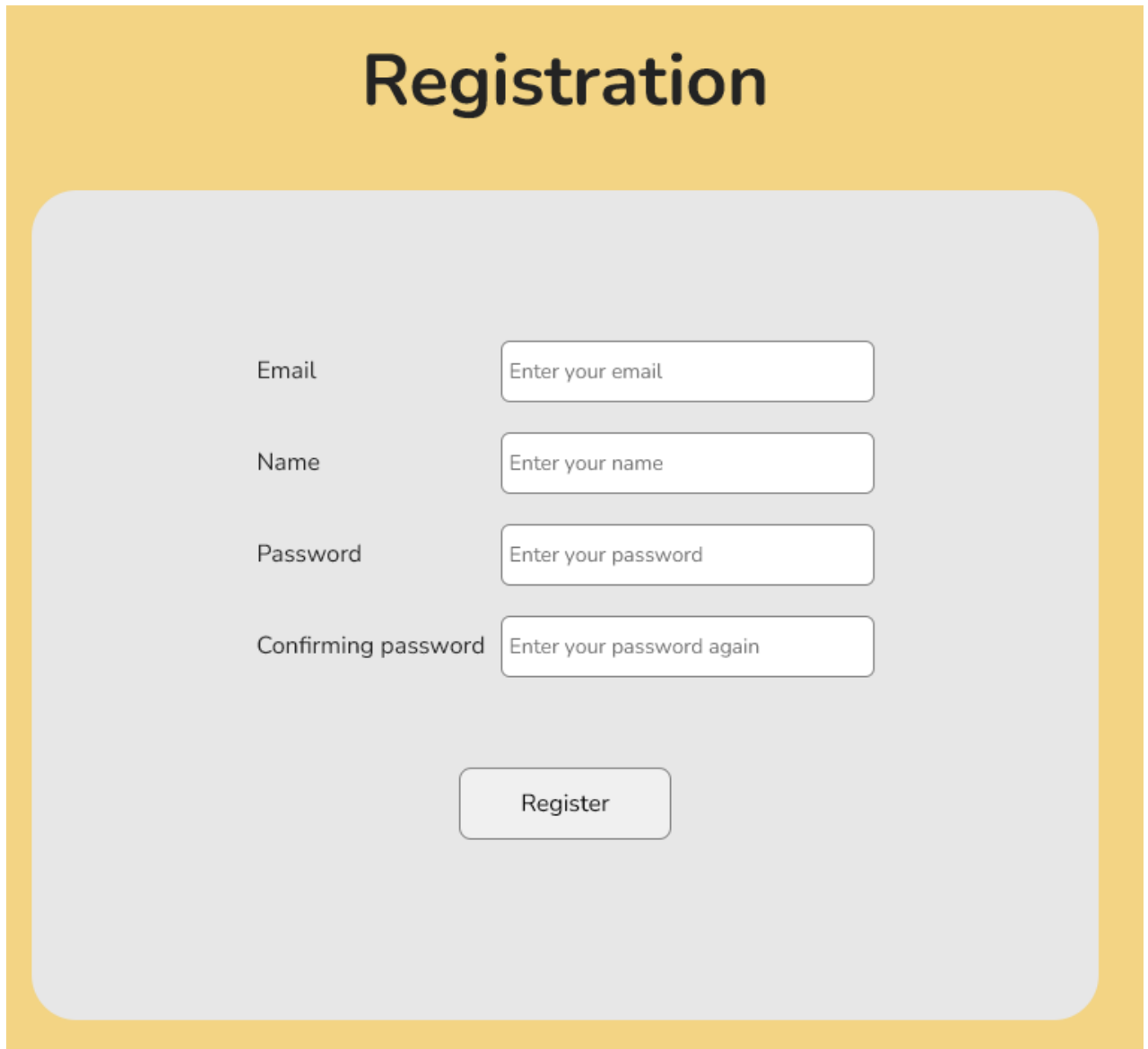
The image shows a login form on a yellow background. At the top, the word "Login" is written in a large, bold, black font. Below it, there is a light gray rounded rectangle containing the form fields. The first field is labeled "Email" and has a text input box with the placeholder "Enter your email". The second field is labeled "Password" and has a text input box with the placeholder "Enter your password". Below these fields is a rounded rectangular button labeled "Log in". At the bottom of the gray area, there is a link that says "Don't have an account? Sign up".

Рисунок 4.4 – Сторінка авторизації

На сторінці авторизації користувач повинен ввести свої облікові дані: адресу електронної пошти та пароль. Після успішної авторизації відкривається можливість додавання товарів до кошика, оформлення замовлень та використання додаткового функціоналу, доступного лише зареєстрованим користувачам.

Користувач, який не має облікового запису, може перейти на сторінку реєстрації, щоб створити власний акаунт.

На рисунку 4.5 можна побачити інтерфейс сторінки для авторизації користувача.



The image shows a registration form titled "Registration" in a large, bold, black font at the top center. The form is set against a light gray background with rounded corners, which is itself centered on a solid yellow background. The form contains four input fields, each with a label to its left and placeholder text inside the field:

- Email**: The input field contains the placeholder text "Enter your email".
- Name**: The input field contains the placeholder text "Enter your name".
- Password**: The input field contains the placeholder text "Enter your password".
- Confirming password**: The input field contains the placeholder text "Enter your password again".

Below these input fields, centered horizontally, is a single button with the text "Register".

Рисунок 4.5 – Сторінка реєстрації

На цій сторінці користувач повинен заповнити поля для введення електронної пошти, імені та пароля. Після успішної реєстрації користувач отримує можливість авторизуватися у вебзастосунку та користуватись усім функціоналом, доступним для зареєстрованих користувачів.

Після успішної авторизації під обліковим записом адміністратора відкривається доступ до адміністративного меню вебзастосунку.

На рисунку 4.6 зображений інтерфейс сторінки адміністративної панелі.

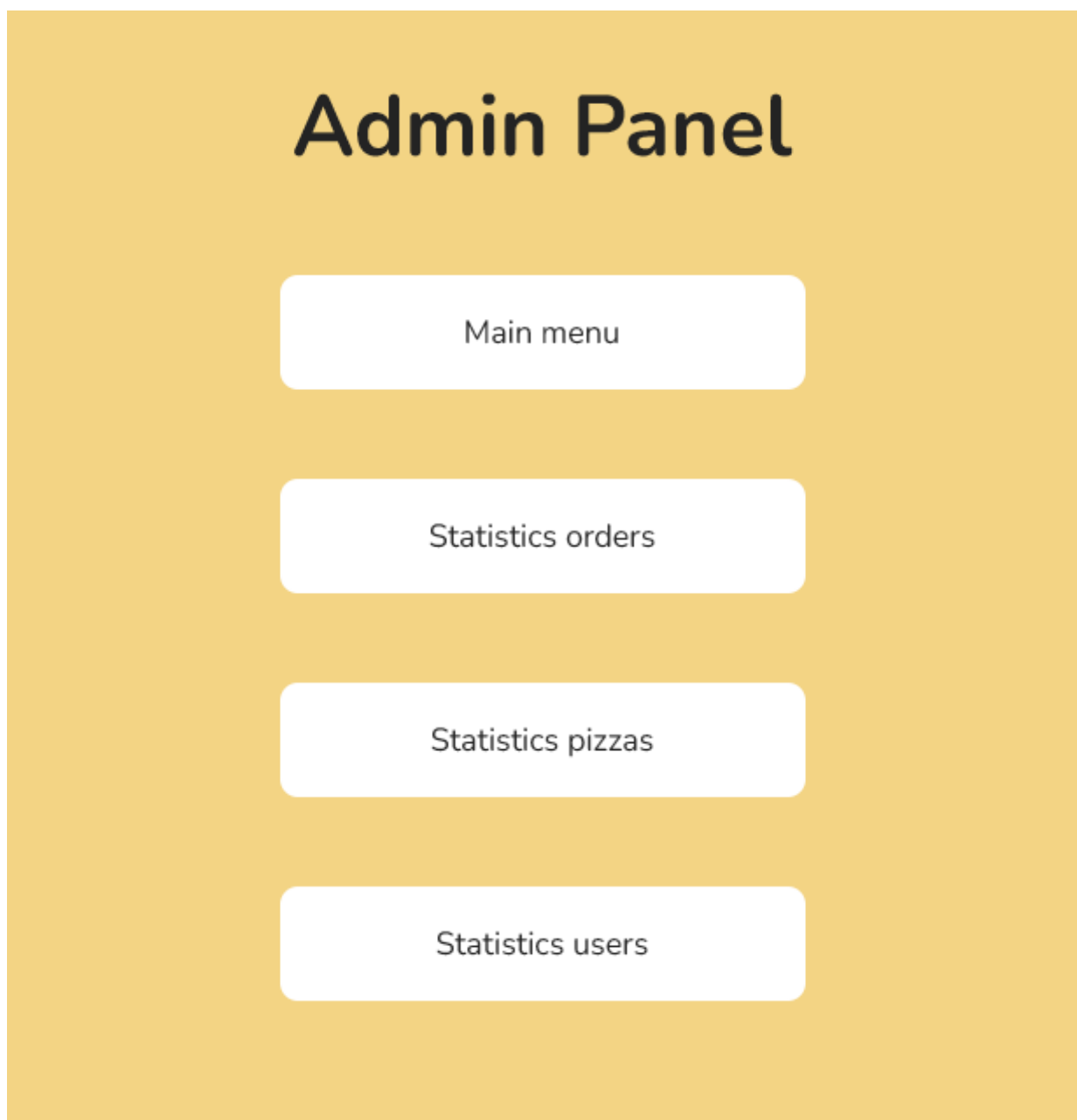


Рисунок 4.6 – Сторінка адміністративної панелі

На цій сторінці адміністратор може перейти до інших розділів для перегляду детальної статистики. Інтерфейс адміністративного меню побудований таким чином, щоб забезпечити зручну навігацію до сторінок зі списком користувачів, замовлень, а також популярних позицій меню.

На рисунку 4.7 можна побачити інтерфейс сторінки всіх замовлень.

Back

Order Statistics

Statistics pizzas

Order ID	User ID	Pizzas	Total Price	Date	Delivery date and time
8a4315f7-473e-45aa-b436-5f9896f89b2f	66c676bd-6392-4855-84ee-eb9ce9a50dde	Pepperoni - 1 pcs - \$12	\$12	12.05.2025, 17:10:37	12.05.2025, 18:10:00
ac60d00e-6fe8-4637-9028-1221c1edb5bc	66c676bd-6392-4855-84ee-eb9ce9a50dde	Crazy pepperoni - 1 pcs - \$13	\$13	12.05.2025, 17:10:41	12.05.2025, 18:10:00
3c819798-f192-46a1-8aae-20e501a39083	66c676bd-6392-4855-84ee-eb9ce9a50dde	Crazy pepperoni - 2 pcs - \$13 Cheeseburger pizza - 1 pcs - \$12	\$38	12.05.2025, 17:10:47	12.05.2025, 20:10:00
f6e9f3b1-8e47-43ff-bbcf-505d90ec7d81	66c676bd-6392-4855-84ee-eb9ce9a50dde	Pepperoni Fresh Peper - 1 pcs - \$11 Pepperoni - 1 pcs - \$12 Cheeseburger pizza - 1 pcs - \$12	\$35	12.05.2025, 17:10:55	12.05.2025, 19:10:00
5baca371-3c3e-40ad-a2de-9c1f3e025d12	6b69d626-07d8-43e9-b7ae-c6b100471849	Pepperoni - 3 pcs - \$12 Cheeseburger pizza - 1 pcs - \$12	\$48	12.05.2025, 17:11:19	12.05.2025, 17:50:00
f53d9e60-b3c3-4203-8a63-736c94fc00e7	6b69d626-07d8-43e9-b7ae-c6b100471849	Cheeseburger pizza - 1 pcs - \$12 Crazy pepperoni - 1 pcs - \$13	\$25	12.05.2025, 17:11:31	12.05.2025, 20:40:00

Рисунок 4.7 – Сторінка всіх замовлень

На цій сторінці відображаються всі замовлення, зроблені користувачами: зокрема, дата та час, ім'я користувача, обрана піца, кількість, загальна сума дата замовлення.

На рисунку 4.8 інтерфейс сторінки популярних позицій з меню.

Back

Pizza Statistics

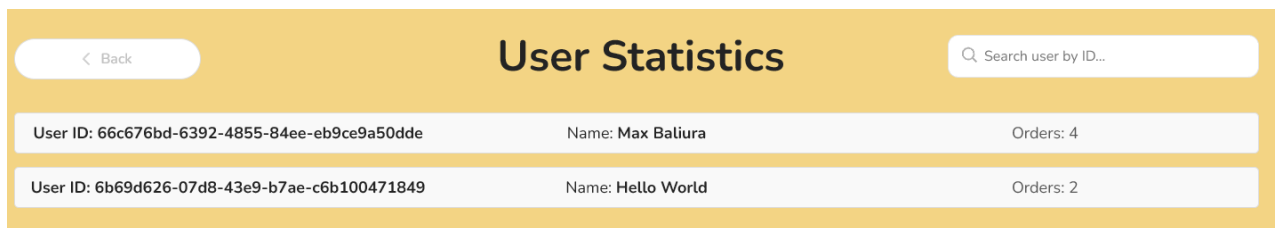
Users

Pepperoni thin 26		Ordered: 5 times
Crazy pepperoni thin 30		Ordered: 4 times
Cheeseburger pizza thin 26		Ordered: 4 times
Pepperoni Fresh Peper thin 26		Ordered: 1 times

Рисунок 4.8 – Сторінка популярних позицій з меню

На сторінці відображається перелік піц, які замовляли найчастіше. Кожна позиція містить інформацію про назву піци, кількість замовлень та інші аналітичні дані. Це дає можливість адміністратору аналізувати популярність страв та приймати обґрунтовані рішення щодо оновлення меню.

Ще однією важливою частиною адміністративного меню є сторінка (рис. 4.9) зі статистикою користувачів, яка дозволяє адміністратору отримувати інформацію про всіх зареєстрованих клієнтів.



< Back			User Statistics			Search user by ID...		
User ID: 66c676bd-6392-4855-84ee-eb9ce9a50dde			Name: Max Baliura			Orders: 4		
User ID: 6b69d626-07d8-43e9-b7ae-c6b100471849			Name: Hello World			Orders: 2		

Рисунок 4.9 – Сторінка всіх користувачів

На цій сторінці відображається інформація про користувачів, зокрема їхнє ім'я, унікальний ідентифікатор (ID) та кількість оформлених замовлень. Це дає змогу адміністратору аналізувати активність.

4.3 Функціональне тестування вебзастосунку

Для перевірки коректності роботи вебзастосунку було проведено функціональне тестування основних компонентів системи. Метою тестування є виявлення можливих помилок у роботі користувацького інтерфейсу, перевірка доступу до функціональних можливостей відповідно до ролі користувача, а також перевірка працездатності адміністративної частини застосунку.

Результати тестування наведено у таблиці 4.1, де вказано назву кожного тесту, його короткий опис, очікуваний результат, фактичний результат після тестування. Усі тести були проведені вручну на етапі завершення розробки.

Таблиця 4.1 – Результати функціонального тестування вебзастосунку

№	Функціональна можливість	Опис тесту	Очікуваний результат	Фактичний результат
1	2	3	4	5
1	Авторизація	Перевірка процесу авторизації користувача з правильними даними	Користувач успішно авторизується після введення правильних даних	Авторизація успішна
2	Реєстрація	Перевірка процесу реєстрації нового користувача	Користувач успішно реєструється	Реєстрація пройшла успішно
3	Пошук піци по назві	Перевірка функціональності пошуку по назві піци	Пошук знаходить правильну піцу за введеною назвою	Пошук працює коректно
4	Сортування меню	Перевірка можливості сортування меню за алфавітом, ціною	Меню сортується за вибраним критерієм	Сортування працює коректно
5	Зміна категорії меню	Перевірка функціональності зміни категорії меню	Користувач може переглядати піци, що належать до вибраної категорії	Зміна категорії працює

Продовження таблиці 4.1

1	2	3	4	5
6	Пагінація меню	Перевірка можливості пагінації для перегляду меню	Користувач може листати меню, переглядаючи кілька сторінок меню	Пагінація працює коректно
7	Сторінка інформації про піцу	Перевірка відображення інформації про піцу	На сторінці відображаються всі параметри піци	Інформація відображається правильно
8	Додавання до кошика	Перевірка можливості додати піцу до кошика без авторизації	Користувач не може додати піцу до кошика без авторизації	Додавання неможливе без авторизації
9	Додавання до кошика	Перевірка можливості додати піцу до кошика після авторизації	Користувач успішно додає піцу до кошика після авторизації	Додавання успішне
10	Кошик замовлення	Перевірка відображення товарів у кошику	У кошику відображаються вибрані піци, а також правильна кількість та ціна	Кошик працює коректно

Продовження таблиці 4.1

1	2	3	4	5
11	Оформлення замовлення	Перевірка можливості оформлення замовлення	Користувач може вибрати час для отримання піци та оформити замовлення	Замовлення оформлено успішно
12	Панель адміністратора	Перевірка доступу адміністратора до статистики	Адміністратор має можливість переглядати інформацію про замовлення, користувачів і популярні позиції	Панель адміністратора працює коректно
13	Статистика замовлень	Перевірка перегляду всіх замовлень адміністратором	Адміністратор може побачити всі замовлення, їх дату, час, користувача який зробив замовлення	Статистика замовлень відображається коректно
14	Статистика популярних позицій	Перевірка перегляду популярних позицій у меню	Адміністратор може побачити статистику за популярними позиціями в меню	Статистика відображається коректно

Кінець таблиці 4.1

1	2	3	4	5
15	Статистика користувачів	Перевірка перегляду інформації про користувачів	Адміністратор бачить ім'я, айді та кількість замовлень для кожного користувача	Інформація про користувачів відображається коректно

4.4 Тестування зручності користування вебзастосунком

Окрім функціонального тестування, було проведено тестування зручності користування розробленим вебзастосунком. Основна мета полягала у визначенні, наскільки легко та інтуїтивно користувачі можуть взаємодіяти з інтерфейсом сайту, швидко знаходити потрібну інформацію та виконувати основні дії.

На головній сторінці користувачі мають змогу переглядати список піц, шукати конкретні позиції за назвою, сортувати їх за категоріями, ціною або алфавітом, що значно спрощує процес навігації. Кнопка авторизації помітна й розміщена у зручному для доступу місці. Елементи сторінки чітко виділені, структура логічна, що сприяє швидкому освоєнню навіть новими користувачами.

Форма реєстрації та авторизації проста й не вимагає надмірної кількості даних, що позитивно впливає на зручність використання. Процес додавання товарів до кошика є зрозумілим і супроводжується візуальним підтвердженням, а оформлення замовлення займає мінімум часу.

Інтерфейс адміністратора має зручну навігацію, де всі функції згруповані за відповідними категоріями: перегляд замовлень, статистика по

користувачах, популярні позиції меню. Це дозволяє адміністратору швидко знаходити потрібну інформацію.

У результаті тестування було підтверджено, що вебзастосунок є зручним, інтуїтивно зрозумілим та адаптованим до потреб як звичайного користувача, так і адміністратора. Дизайн, структура сторінок та логіка взаємодії відповідають сучасним вимогам до вебзастосунків, що забезпечує позитивний досвід користування.

4.5 Висновки за розділом

У цьому розділі було представлено результати реалізації вебзастосунку для автоматизації адміністрування сервісу замовлення піци. Описано функціональні можливості для різних типів користувачів, наведено приклади інтерфейсу та основні дії. Визначено мінімальні вимоги до клієнтської та серверної частин, що забезпечують коректну роботу вебдодатку. Проведено тестування функціоналу та перевірку зручності взаємодії з вебзастосунком, що підтвердило його ефективність та доступність для користувачів.

ВИСНОВКИ

У результаті виконання дипломної роботи було розроблено вебзастосунок для автоматизації адміністрування сервісу замовлення піци. Для реалізації проекту використано сучасний технологічний стек: TypeScript як мова програмування, React – для створення клієнтської частини, Express – для реалізації серверної частини, та MongoDB – як гнучку та масштабовану базу даних. Середовищем розробки обрано Visual Studio Code.

Клієнтська частина дозволяє користувачам переглядати каталог піц, додавати їх до кошика, оформлювати замовлення, а також здійснювати авторизацію та реєстрацію. Серверна частина обробляє запити, взаємодіє з базою даних, виконує обробку даних користувачів та замовлень. Інтерфейс програми створено з урахуванням зручності, швидкості взаємодії та адаптивності до різних розмірів екранів.

Практична цінність полягає у створенні цілісного прикладного рішення, яке може бути використано як база для впровадження онлайн-системи прийому та обробки замовлень у закладах громадського харчування. Реалізований вебзастосунок забезпечує зручність адміністрування, автоматизує ключові операції та дозволяє скоротити час обслуговування клієнтів.

Подальший розвиток системи може передбачати розширення функціональності, наприклад, додавання модуля оформлення доставки замовлень із вибором адреси та часу доставки. Також можливе впровадження відстеження статусу замовлення, онлайн-оплати та особистих кабінетів для користувачів з історією замовлень.

Окрім реалізації функціоналу, було проведено функціональне тестування основних модулів вебзастосунку. Також виконано тестування зручності використання інтерфейсу, що підтвердило ефективність взаємодії користувача з системою.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. React [Electronic resource]. – Access mode: <https://react.dev/>.
2. Express (framework for Node.js) [Electronic resource]. – Access mode: <https://expressjs.com/>.
3. Технології в ресторанному бізнесі [Електрон. ресурс]. – Режим доступу: <https://wezom.com.ua/ua/blog/tehnologiyi-v-restorannomu-biznesi>.
4. Papa John's [Electronic resource]. – Access mode: <https://almere.papajohns.co.nl/menu>.
5. Domino's Pizza [Електрон. ресурс]. – Режим доступу: <https://dominos.ua/uk/kyiv/>.
6. Pizza Hut [Electronic resource]. – Access mode: <https://www.pizzahut.com/>.
7. Аспекти типізації за допомогою TypeScript під час розробки вебзастосунку [Електрон. ресурс]. – Режим доступу: https://www.researchgate.net/publication/357189779_Some_Aspects_of_Strict_Typing_in_TypeScript_While_Web_Application_Developing.
8. What is this framework? [Electronic resource] – Access mode: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries/Introduction.
9. Чому і коли варто обирати React? [Електрон. ресурс] – Режим доступу: <https://it-rating.ua/news-3658>.
10. Як організовані реляційні бази даних? [Електрон. ресурс] – Режим доступу: <https://data-b-i.com/uk/article/yak-organyzoveny-relyatsiyni-bazy-danykh.html>.
11. Побудова динамічних вебзастосунків за допомогою MVC [Електрон. ресурс]. – Режим доступу: <https://blog.ithillel.ua/articles/building-dynamic-web-applications-using-mvc>.
12. Leveraging MVC Architecture for Efficient Web Development [Electronic resource]. – Access mode:

<https://www.blumandarin.com/blog/leveraging-mvc-architecture-for-efficient-web-development> .

13. Understanding MVC Architecture in the MERN Stack [Electronic resource]. – Access mode: <https://medium.com/@ansari028amaan/understanding-mvc-architecture-in-the-mern-stack-5cc083828298>.

14. Documentation CLI npm [Electronic resource]. – Access mode: <https://docs.npmjs.com/cli/v9/commands/npm?v=true>.

15. JSON Web Token [Electronic resource]. – Access mode: <https://www.npmjs.com/package/jsonwebtoken>.

16. bcrypt — npm [Electronic resource]. – Access mode: <https://www.npmjs.com/package/bcrypt>.

17. Mongoose version 8.13.1 [Electronic resource]. – Access mode: <https://mongoosejs.com/>.

18. Cross-Origin Resource Sharing (CORS) [Electronic resource]. – Access mode: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS>.

19. React — Вікіпедія [Електрон. ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/React>.

20. Роутинг у React [Електрон. ресурс]. – Режим доступу: <https://blog.ithillel.ua/articles/routing-in-react>.

21. Redux Toolkit як засіб ефективної розробки [Електрон. ресурс]. – Режим доступу: <https://markup-ua.com/redux-toolkit-yak-zasib-efektivnoï-rozrobki/>.

22. clsx — npm [Electronic resource]. – Access mode: <https://www.npmjs.com/package/clsx>.

23. Sass [Electronic resource]. – Access mode: <https://sass-lang.com/>.

24. react-paginate — npm [Electronic resource]. – Access mode: <https://www.npmjs.com/package/react-paginate>.

25. Skeleton React [Electronic resource]. – Access mode: <https://skeletonreact.com/>.

26. Axios [Electronic resource]. – Access mode: <https://axios-http.com/>.

27. Using React-Toastify to style toast messages [Electronic resource]. –
Access mode: <https://blog.logrocket.com/using-react-toastify-style-toast-messages/>.

ДОДАТОК А
Технічне завдання

Вступ

Сучасні сервіси доставки їжі набувають дедалі більшої популярності, особливо в умовах активного розвитку онлайн-комерції. Піцерії та заклади швидкого харчування прагнуть автоматизувати процеси прийому замовлень і взаємодії з клієнтами, щоб зменшити навантаження на персонал, уникнути помилок та пришвидшити обслуговування. Вебзастосунки, які забезпечують такі функції, є перспективним інструментом для малого і середнього бізнесу, адже дозволяють не лише приймати замовлення онлайн, а й керувати асортиментом, переглядати статистику продажів та адмініструвати платформу в зручному інтерфейсі.

Розробка подібних систем потребує впровадження сучасних технологій та створення інтуїтивно зрозумілого користувацького середовища. Це дозволяє значно підвищити якість обслуговування, забезпечити прозорість процесу замовлення і створити комфортні умови як для клієнтів, так і для адміністраторів сервісу.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Програмна реалізація вебзастосунку з можливістю автоматизації адміністрування роботи сервісу замовлення піци», затверджене наказом Національного університету «Запорізька політехніка» № 209 від 28 квітня 2025 р.

A.2 Призначення розробки

Розробка вебзастосунку спрямована на створення зручного сервісу для замовлення піци, який дозволить автоматизувати процес обробки замовлень та

взаємодії з клієнтами через сучасний інтерфейс, що полегшує адміністрування та підвищує ефективність обслуговування.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Програмне забезпечення вебзастосунку з можливістю автоматизації адміністрування роботи сервісу замовлення піци забезпечує виконання таких функцій:

- можливість реєстрації та авторизації користувачів;
- відображення списку доступних піц із сортуванням та фільтрацією за параметрами;
- можливість пошуку піци за назвою;
- додавання піц до кошика;
- оформлення замовлення;
- перегляд списку замовлень, користувачів, популярних позицій з меню адміністратором;
- збереження даних користувачів, товарів та замовлень у базі даних.

A.3.2 Вимоги до інтерфейсу програми

Програма повинна мати:

- інтерфейс інтуїтивно зрозумілим, адаптивним до різних розмірів екранів;
- наявність повідомлень про успішні дії;
- зручну навігацію;
- використання сучасного дизайну з чіткою структурою розділів.

А.3.3 Умови експлуатації

Розроблений вебзастосунок призначений для використання в середовищі сучасних браузерів Google Chrome, Mozilla Firefox, Opera на персональних комп'ютерах, ноутбуках, планшетах і смартфонах. Для коректної роботи необхідне стабільне підключення до Інтернету та підтримка JavaScript у браузері. Інтерфейс адаптовано для комфортної роботи на екранах різного розміру.

А.3.4 Вимоги до складу та параметрів технічних засобів

Для роботи клієнтської частини застосунку з боку користувача необхідний пристрій з такими характеристиками::

- процесор від 1 ГГц з 2 або більше ядрами;
- оперативна пам'ять 4 ГБ або більше;
- SSD або HDD від 64 ГБ;
- браузер – Google Chrome, Mozilla Firefox або Opera;
- підключення до мережі Інтернет.

Для розробки та експлуатації вебзастосунку на стороні сервера необхідні такі технічні засоби:

- процесор – не менше 1.5 ГГц, з підтримкою щонайменше двох ядер;
- оперативна пам'ять 4 ГБ або більше;
- SSD або HDD від 20 ГБ;
- встановлене програмне середовище – Node.js, база даних MongoDB.

ДОДАТОК Б
Текст програми

Б.1 Текст програми з файлу auth.js

```

const { body } = require('express-validator');
const User = require('../models/User');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const { v4: uuidv4 } = require('uuid');
require('dotenv').config();

const SECRET_KEY = process.env.SECRET_KEY;

const { validationResult } = require('express-validator');

exports.register = async (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ message: errors.array() });
  }

  const { username, email, password } = req.body;

  try {
    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res.status(400).json({ message: 'This email is already taken' });
    }

    const hashedPassword = await bcrypt.hash(password, 10);
    const id = uuidv4();

    const newUser = new User({ id, username, email, password: hashedPassword,
role: 'user' });
    await newUser.save();

    res.status(201).json({ id, username, email });
  } catch (error) {
    console.error('Error registering user:', error);
    res.status(500).json({ message: 'Error registering user', error });
  }
};

```

```

exports.login = async (req, res) => {
  const { email, password } = req.body;

  try {
    const user = await User.findOne({ email });
    if (!user) return res.status(400).json({ message: 'Invalid email or password'
});

    const isPasswordValid = await bcrypt.compare(password, user.password);
    if (!isPasswordValid) return res.status(400).json({ message: 'Invalid email
or password' });

    const token = jwt.sign(
      { id: user.id, username: user.username, email: user.email, role: user.role
},
      SECRET_KEY,
    );

    res.status(200).json({ token, role: user.role, id: user.id });
  } catch (error) {
    console.error('Error logging in:', error);
    res.status(500).json({ message: 'Error logging in', error });
  }
};

```

Б.2 Текст програми з файлу order.js

```

const OrderModel = require('../models/Order');

const getAllOrders = async (req, res) => {
  if (req.user.role !== 'admin') {
    return res.sendStatus(403);
  }

  try {
    const orders = await OrderModel.find({});
    res.status(200).json(orders);
  } catch (error) {
    console.error('Error retrieving orders:', error);
    res.status(500).json({ message: 'Error retrieving orders', error });
  }
};

const getOrdersByUser = async (req, res) => {
  if (req.user.role !== 'admin') {
    return res.sendStatus(403);
  }
};

```

```

const { userId } = req.params;

try {
  const orders = await OrderModel.find({ userId });

  if (orders.length === 0) {
    return res.status(404).json({ message: 'Orders not found' });
  }

  res.status(200).json(orders);
} catch (error) {
  console.error('Error retrieving orders:', error);
  res.status(500).json({ message: 'Error retrieving orders', error });
}
};

const addOrder = async (req, res) => {
  const { id, userId, pizzas, totalPrice, date, deliveryDateTime } = req.body;

  try {
    const formattedPizzas = pizzas.map((pizza) => ({
      id: pizza.id,
      title: pizza.title,
      price: pizza.price,
      imageUrl: pizza.imageUrl,
      count: pizza.count,
      types: pizza.types,
      sizes: pizza.sizes,
    }));

    const newOrder = new OrderModel({
      id,
      userId,
      pizzas: formattedPizzas,
      totalPrice,
      date: date || new Date().toISOString(),
      deliveryDateTime,
    });

    await newOrder.save();
    res.status(201).json({ message: 'Order created', order: newOrder });
  } catch (error) {
    console.error('Error creating order:', error);
    res.status(500).json({ message: 'Error creating order', error });
  }
};

module.exports = {
  getAllOrders,
  getOrdersByUser,
  addOrder,
};

```

Б.3 Текст програми з файлу pizza.js

```

const PizzaModel = require('../models/Pizza');

const getPizzas = async (req, res) => {
  try {
    const { category, sortBy, order, search, page = 1, limit = 4 } = req.query;

```

```

const filters = {};
if (category) filters.category = category;
if (search) filters.title = new RegExp(search, 'i');

const sortOptions = {};
if (sortBy) sortOptions[sortBy] = order === 'desc' ? -1 : 1;

const pizzas = await PizzaModel.find(filters)
  .sort(sortOptions)
  .skip((page - 1) * limit)
  .limit(Number(limit));

res.json(pizzas);
} catch (error) {
  console.error('Error fetching pizzas:', error);
  res.status(500).json({ message: 'Error fetching pizzas', error });
}
};

const getPizzaById = async (req, res) => {
  const { id } = req.params;

  try {
    const pizza = await PizzaModel.findOne({ id });

    if (!pizza) {
      return res.status(404).json({ message: 'Pizza not found' });
    }

    res.status(200).json(pizza);
  } catch (error) {
    console.error('Error fetching pizza by id:', error);
    res.status(500).json({ message: 'Error fetching pizza', error });
  }
};

module.exports = {
  getPizzas,
  getPizzaById,
};

```

Б.4 Текст програми з файлу statistics.js

```

const OrderModel = require('../models/Order');
const UserModel = require('../models/User');

const getPopularPizzas = async (req, res) => {
  if (req.user.role !== 'admin') {
    return res.sendStatus(403);
  }

  try {
    const orders = await OrderModel.find({});
    const pizzaPopularity = {};

    orders.forEach((order) => {
      order.pizzas.forEach((pizza) => {
        if (!pizzaPopularity[pizza.id]) {
          pizzaPopularity[pizza.id] = {

```

```

        title: pizza.title,
        imageUrl: pizza.imageUrl,
        count: 0,
        types: pizza.types,
        sizes: pizza.sizes,
    });
}
pizzaPopularity[pizza.id].count += pizza.count;
});
});

const sortedPizzas = Object.values(pizzaPopularity).sort((a, b) => b.count
- a.count);
res.status(200).json(sortedPizzas);
} catch (error) {
    console.error('Error fetching pizza statistics:', error);
    res.status(500).json({ message: 'Error fetching statistics', error });
}
};

const getUserStatistics = async (req, res) => {
    if (req.user.role !== 'admin') {
        return res.sendStatus(403);
    }

    try {
        const orders = await OrderModel.find({});
        const userStatistics = {};
        orders.forEach((order) => {
            if (!userStatistics[order.userId]) {
                userStatistics[order.userId] = 0;
            }
            userStatistics[order.userId] += 1;
        });
        const userIds = Object.keys(userStatistics);
        const users = await UserModel.find({ id: { $in: userIds } });
        const formattedStatistics = users.map((user) => ({
            userId: user.id,
            username: user.username,
            orderCount: userStatistics[user.id] || 0,
        }));

        res.status(200).json(formattedStatistics);
    } catch (error) {
        console.error('Error fetching user statistics:', error);
        res.status(500).json({ message: 'Error fetching user statistics', error });
    }
};

module.exports = { getPopularPizzas, getUserStatistics };

```

Б.5 Текст програми з файлу authMiddleware.js

```

const jwt = require('jsonwebtoken');

const authenticateJWT = (req, res, next) => {
    const authHeader = req.headers.authorization;

    if (authHeader) {
        const token = authHeader.split(' ')[1];

```

```

    jwt.verify(token, process.env.SECRET_KEY, (err, user) => {
      if (err) {
        return res.sendStatus(403);
      }

      req.user = user;
      next();
    });
  } else {
    res.sendStatus(401);
  }
};

module.exports = { authenticateJWT };

```

Б.6 Текст програми з файлу authRoute.js

```

const express = require('express');
const router = express.Router();
const auth = require('../controllers/auth');
const { body } = require('express-validator');

router.post(
  '/register',
  [
    body('email').isEmail().withMessage('Invalid email'),
    body('email').isLength({ min: 6 }).withMessage('Email must be longer than 5 characters'),
    body('password').isLength({ min: 8 }).withMessage('Password must be longer than 7 characters'),
  ],
  auth.register,
);

router.post('/login', auth.login);

module.exports = router;

```

Б.7 Текст програми з файлу orderRoute.js

```

const express = require('express');
const { authenticateJWT } = require('../middleware/authMiddleware');
const { getAllOrders, getOrdersByUser, addOrder } = require('../controllers/order');

const router = express.Router();

router.get('/', authenticateJWT, getAllOrders);

router.get('/:userId', authenticateJWT, getOrdersByUser);

router.post('/add', addOrder);

module.exports = router;

```

Б.8 Текст програми з файлу pizzaRoute.js

```
const express = require('express');
const { getPizzas, getPizzaById } = require('../controllers/pizza');

const router = express.Router();

router.get('/', getPizzas);

router.get('/:id', getPizzaById);

module.exports = router;
```

Б.9 Текст програми з файлу statisticsRoute.js

```
const express = require('express');
const { getPopularPizzas, getUserStatistics } = require('../controllers/statistics');
const { authenticateJWT } = require('../middleware/authMiddleware');

const router = express.Router();

router.get('/popular-pizzas', authenticateJWT, getPopularPizzas);

router.get('/users', authenticateJWT, getUserStatistics);

module.exports = router;
```

Б.10 Текст програми з файлу calculatePricePizza.ts

```
export const calculatePricePizza = (activeSize: number, activeType: number,
price: number) => {
  const sizePriceIncrement = activeSize === 1 ? 2 : activeSize === 2 ? 4 : 0;
  const typePriceIncrement = activeType === 1 ? 1 : 0;
  return price + sizePriceIncrement + typePriceIncrement;
};
```

Б.11 Текст програми з файлу calculateTotalPrice.ts

```
import { TCartPizza } from '../redux/slices/cart/types';

export const calculateTotalPrice = (pizzas: TCartPizza[]) => {
  return pizzas.reduce((sum: number, pizza: TCartPizza) => {
    return sum + pizza.price * pizza.count;
  }, 0);
};
```

Б.12 Текст програми з файлу getCartFromLocalStorage.ts

```
import { TCartPizza } from '../redux/slices/cart/types';
import { calculateTotalPrice } from '../calculateTotalPrice';

export const getCartFromLocalStorage = () => {
  const data = localStorage.getItem('cart');

  const pizzas = data ? JSON.parse(data) : [];
  const totalPrice = calculateTotalPrice(pizzas);

  return {
    pizzas: pizzas as TCartPizza[],
    totalPrice,
  };
};
```

Б.13 Текст програми з файлу Order.tsx

```
import React from 'react';
import { TOrder } from '../redux/slices/order/types';
import styles from './Order.module.scss';

const Order: React.FC<TOrder> = ({ id, userId, pizzas, totalPrice, date,
deliveryDateTime }) => {
  return (
    <div className={styles.order}>
      <div className={styles.order__id}><id></div>
      <div className={styles.order__userId}><userId></div>
      <div>
        <ul className={styles.order__item_list}>
          {pizzas.map((pizza) => (
            <li className={styles.order__item_detail} key={pizza.id}>
              {pizza.title} - {pizza.count} pcs - ${pizza.price}
            </li>
          ))}
        </ul>
      </div>
      <div className={styles.order__total_price}>${totalPrice}</div>
      <div
        className={styles.order__date}><new
Date(date).toLocaleString()></div>
      <div className={styles.order__delivery_date}>
        {new Date(deliveryDateTime).toLocaleString()}
      </div>
    </div>
  );
};

export default Order;
```

Б.14 Текст програми з файлу StatisticsPizza.tsx

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';
import { Link, useNavigate } from 'react-router-dom';

type TStatisticsPizza = {
```

```

    title: string;
    count: number;
    imageUrl: string;
    types: string;
    sizes: number;
  };

const StatisticsPizza: React.FC = () => {
  const [popularPizzas, setPopularPizzas] = useState<TStatisticsPizza[]>([]);
  const [loading, setLoading] = useState<boolean>(true);
  const [error, setError] = useState<string | null>(null);

  const currentUser = JSON.parse(localStorage.getItem('currentUser') || '{}');
  const navigate = useNavigate();

  useEffect(() => {
    if (!currentUser || (currentUser && currentUser.role !== 'admin')) {
      navigate('/');
    }
  }, [currentUser, navigate]);

  useEffect(() => {
    const fetchStatistics = async () => {
      try {
        const token = localStorage.getItem('token');
        const response = await
        axios.get('http://localhost:5000/statistics/popular-pizzas', {
          headers: {
            Authorization: `Bearer ${token}`,
          },
        });

        setPopularPizzas(response.data);
        setLoading(false);
      } catch (error) {
        setError('Error fetching pizza statistics');
        setLoading(false);
      }
    };

    fetchStatistics();
  }, []);

  if (loading) {
    return <div>Loading...</div>;
  }

  if (error) {
    return <div>{error}</div>;
  }

  return (
    <div className="statistics">
      <div className="statistics__header">
        <div className="cart__bottom-buttons">
          <Link to="/adminPanel" className="button button__outline button__add
go-back-btn">
            <svg
              width="8"
              height="14"
              viewBox="0 0 8 14"
              fill="none"
              xmlns="http://www.w3.org/2000/svg">
              <path

```

```

        d="M7 13L1 6.93015L6.86175 1"
        stroke="#D3D3D3"
        strokeWidth="1.5"
        strokeLinecap="round"
        strokeLinejoin="round"></path>
    </svg>

    <span>Back</span>
  </Link>
</div>

<h1 className="statistics__title">Pizza Statistics</h1>

<div className="cart__bottom-buttons">
  <Link to="/statisticsUser" className="button button__outline
button__add go-back-btn">
    <span>Users</span>
  </Link>
</div>
</div>
<div className="statistics__list">
  {popularPizzas.map((pizza, index) => (
    <div className="statistics__item" key={index}>
      <div>
        <span className="statistics__item-title">{pizza.title}</span>
        <p className="statistics__item-description">
          {pizza.types} {pizza.sizes}
        </p>
      </div>
      <img className="statistics__item-image" src={pizza.imageUrl}
alt={pizza.title} />
      <span className="statistics__item-count">Ordered: {pizza.count}
times</span>
    </div>
  ))}
</div>
</div>
);
};

export default StatisticsPizza;

```

ДОДАТОК В
Слайди презентації

Програмна реалізація вебзастосунку з можливістю автоматизації адміністрування роботи сервісу замовлення піци

Виконав
ст.гр.КНТ-221
Керівник
д-р.філос., доцент

Максим БАЛЮРА
Сергій ЛЕОЩЕНКО

Рисунок В.1 – Слайд 1

2

Об'єкт, предмет та мета роботи

Об'єкт дослідження – процес розробки вебзастосунків з можливістю автоматизації адміністрування роботи сервісу замовлення піци.

Предмет дослідження – програмні засоби, зокрема, вебзастосунки для замовлення піци з можливістю автоматизації адміністрування роботи сервісу.

Метою роботи – створення вебзастосунку з можливістю автоматизації адміністрування роботи сервісу замовлення піци.

Рисунок В.2 – Слайд 2

3

Постановка завдання

Метою дипломної кваліфікаційної роботи бакалавра є розробка онлайн-системи для замовлення піци з інтеграцією сучасних веб-технологій, що забезпечує зручний інтерфейс, швидке оформлення замовлень та ефективне управління процесами продажу.

Завдання:

- виконати аналіз предметної області та програмних рішень для онлайн-замовлення піци;
- здійснити проектування архітектури програмного забезпечення для онлайн-системи замовлення піци;
- розробити програмне забезпечення, що включає клієнтську, серверну частини та базу даних;
- виконати тестування розробленої системи для оцінки функціональності та зручності.

Рисунок В.3 – Слайд 3

4

Огляд аналогів

Критерії	Papa John's	Domino's Pizza	Pizza Hut
Гнучкість налаштування складових піци	Висока	Висока	Середня
Реєстрація та авторизація користувачів	Присутня	Присутня	Присутня
Наявність реклами	Присутня	Присутня	Відсутня
Зручність інтерфейсу	Незручний	Зручний	Зручний
Можливість вибору часу отримання замовлення	Присутня	Присутня	Присутня
Швидкість додавання до кошика	Низька	Низька	Висока

Рисунок В.4 – Слайд 4

5

Порівняння фронтенд фреймворків

Критерії	React	Vue	Angular
Продуктивність	Висока, завдяки Virtual DOM	Висока, оптимізований рендеринг	Висока, проте складніша структура
Гнучкість	Висока, легко інтегрується з іншими бібліотеками	Висока, підтримує модульність	Обмежена, має жорстку структуру
Кривина навчання	Середня, необхідно знати JavaScript і JSX	Низька, простий синтаксис	Висока, використання TypeScript і додаткових концепцій
Популярність	Дуже висока, найбільш використовуваний фреймворк	Висока, але поступається React	Висока, підтримується великими компаніями
Підтримка	Активна спільнота, регулярні оновлення від Meta	Висока, розвивається командою Vue	Висока, підтримується Google
Складність налаштування	Низька, швидке встановлення через npm	Низька, легка інтеграція	Висока, потребує додаткової конфігурації

Рисунок В.5 – Слайд 5

6

Порівняння бекенд фреймворків

Критерії	Express	NestJS	Django
Простота та легкість	Мінімалістичний, легко вивчити	Більш складний, потребує вивчення архітектури	Вимагає знання Django ORM та структури
Гнучкість	Дає повний контроль, легко кастомізується	Має модульну архітектуру	Обмежена структура, менше можливостей для кастомізації
Продуктивність	Висока, завдяки легковісній архітектурі	Добра продуктивність завдяки TypeScript	Менша швидкість через Python та вбудовані механізми
Популярність	Широко використовується у Node.js-розробці	Зростає популярність серед TypeScript-розробників	Велика спільнота Python-розробників
Підтримка та розширюваність	Величезна кількість npm-пакетів	Підтримка модулів та TypeScript	Великий набір вбудованих рішень
Швидкість розробки	Простий у налаштуванні, швидкий старт	Потребує конфігурації та вивчення декораторів	Багато вбудованих механізмів, які потрібно освоїти

Рисунок В.6 – Слайд 6

Порівняння баз даних

Критерії	MySQL	PostgreSQL	MongoDB
Тип	Реляційна (SQL)	Реляційна (SQL)	Документо-орієнтована (NoSQL)
Продуктивність	Висока, особливо на читанні	Висока, добре підходить для складних запитів	Висока при роботі з неструктурованими даними
Масштабованість	Висока	Висока	Висока
Гнучкість схеми	Фіксована структура таблиць	Фіксована структура таблиць	Гнучка, схему можна змінювати динамічно
Простота налаштування	Легка	Складніша через додаткові можливості	Легка
Підтримка транзакцій	Так	Так	Так
Інтеграція з Node.js	Добра, через mysql	Добра, через pg	Відмінна, через mongoose

Рисунок В.7 – Слайд 7

Схема роботи вебзастосунку

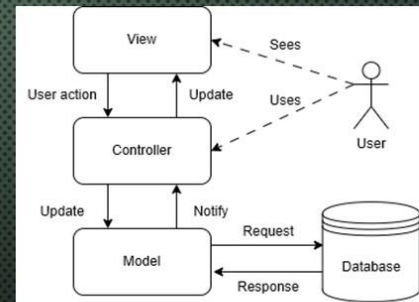
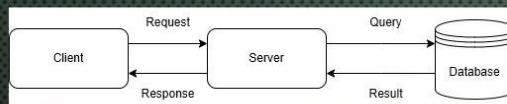


Рисунок В.8 – Слайд 8

Файлова структура вебзастосунку

```

client/
├── Конфігураційні файли (.gitignore, tsconfig.json тощо)
├── @types/ (Оголошення модулів: svg, png, scss, debounce)
├── assets/ (Графічні ресурси: svg-іконки, images)
├── components/ (Загальні компоненти: header, footer, pizza-block, categories-bar тощо)
├── layouts/ (Макети)
├── pages/ (Сторінки: admin-panel, cart, home тощо)
├── redux/ (Стан: слайси, редюсери, типи, дії, store)
├── scss/ (Глобальні стилі та ресурси: normalize.scss, button.scss, pizza.scss)
├── Вхідні файли (index.tsx, App.tsx)
└── utils/ (Утиліти: функції, хелпери)
  
```

```

server/
├── Конфігураційні файли (.env, .gitignore, package.json)
├── controllers (Логіка обробки запитів: створення, оновлення даних тощо)
├── database (Конфігурації підключення до бази даних)
├── middleware (Проміжні функції для обробки запитів)
├── models (Схеми бази даних, моделі для збереження і взаємодії з даними)
├── routes (Маршрути для обробки HTTP-запитів)
└── server.js (Основний файл сервера, підключення middleware, запуск сервера)
  
```

Рисунок В.9 – Слайд 9

Логіка організації сторінок сайту

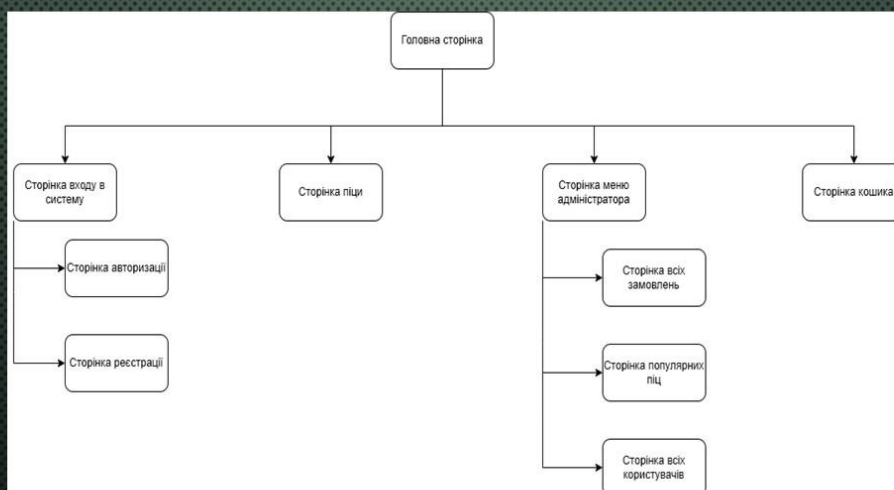


Рисунок В.10 – Слайд 10

Основні алгоритми

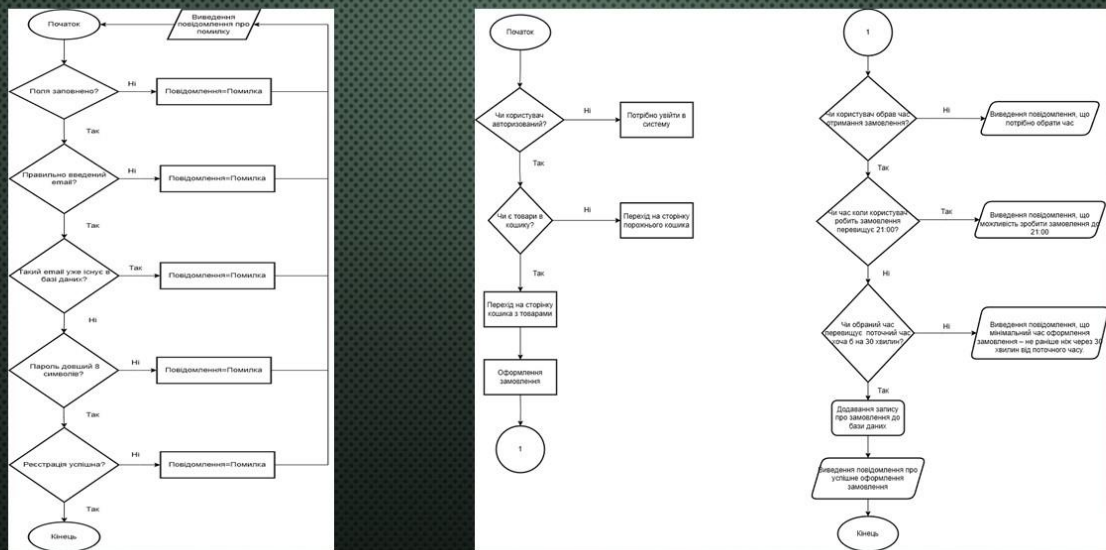


Рисунок В.11 – Слайд 11

Функціональна тестування

№	Функціональна можливість	Опис тесту	Очікуваний результат	Фактичний результат
1	Авторизація	Перевірка процесу авторизації користувача з правильними даними	Користувач успішно авторизується після введення правильних даних	Авторизація успішна
2	Реєстрація	Перевірка процесу реєстрації нового користувача	Користувач успішно реєструється	Реєстрація пройшла успішно
3	Пошук піци по назві	Перевірка функціональності пошуку по назві піци	Пошук знаходить правильну піцу за введеною назвою	Пошук працює коректно
4	Сортування меню	Перевірка можливості сортування меню за алфавітом, ціною	Меню сортується за вибраним критерієм	Сортування працює коректно

Рисунок В.12 – Слайд 12

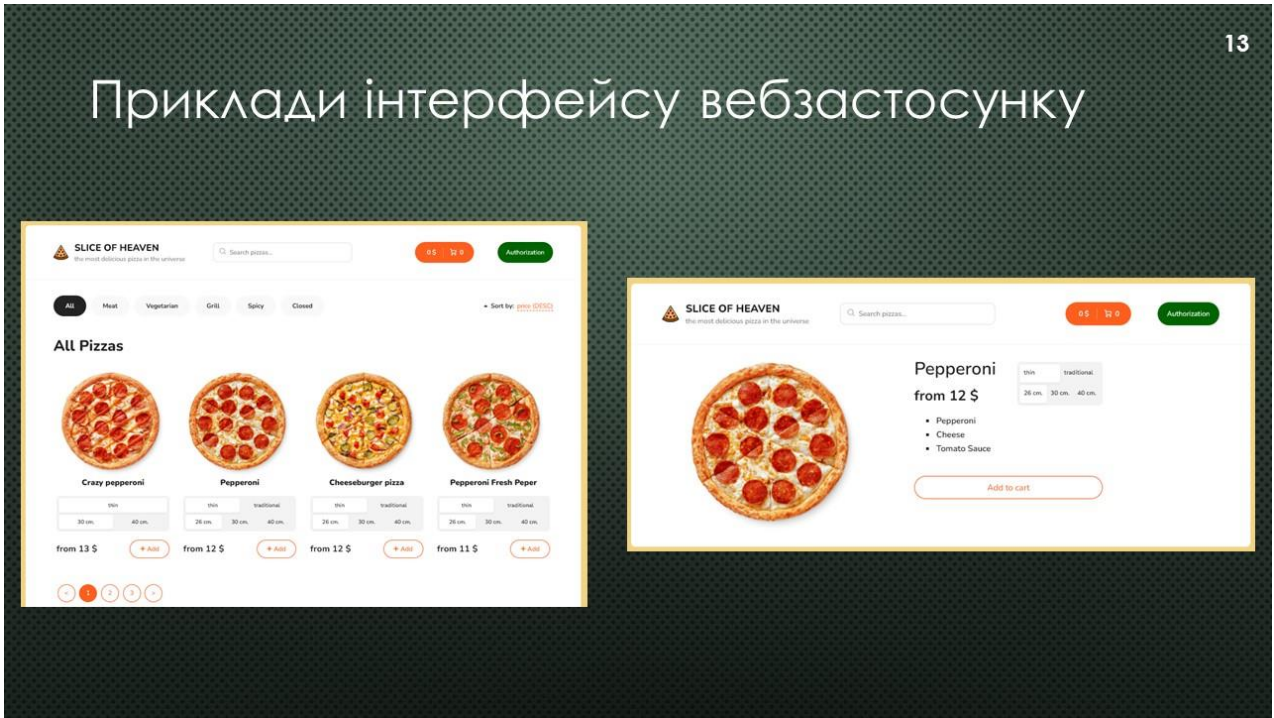


Рисунок В.13 – Слайд 13

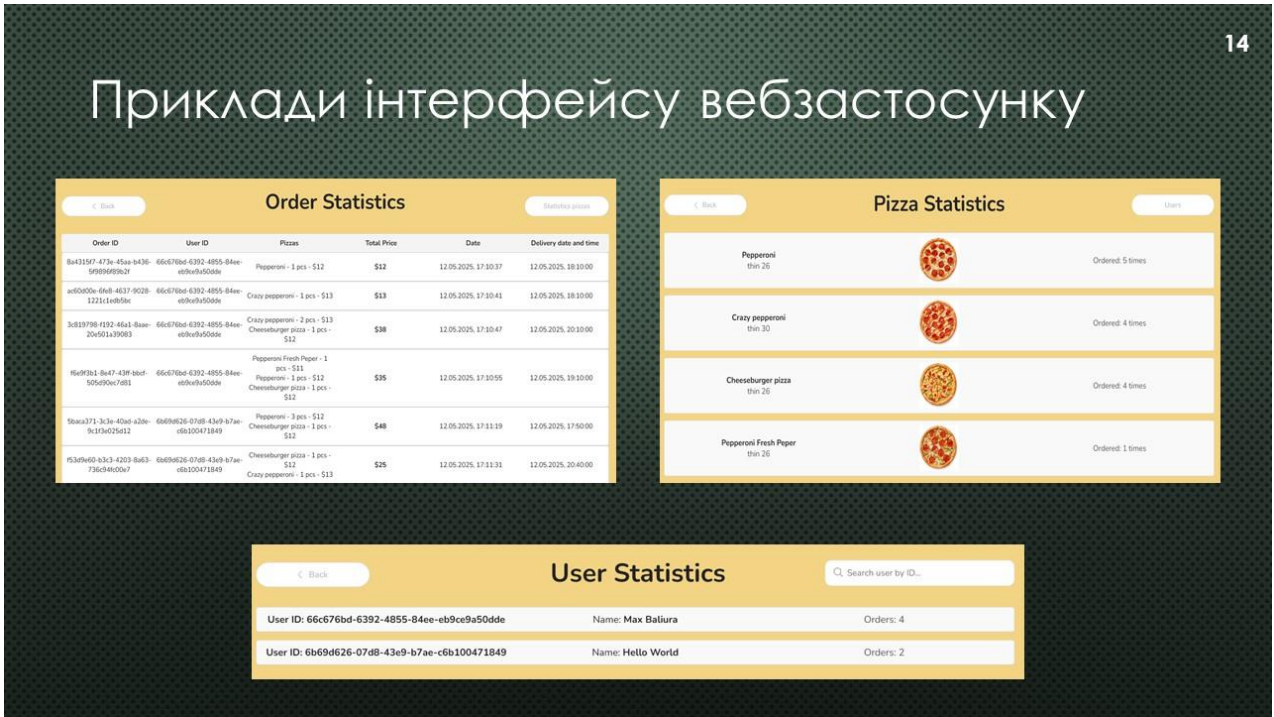


Рисунок В.14 – Слайд 14

Висновки

Розроблений вебзастосунок успішно реалізовує поставлені цілі, включаючи основні функції:

- можливість реєстрації та авторизації користувачів
- відображення списку доступних піц із сортуванням та фільтрацією за параметрами
- можливість пошуку піци за назвою
- додавання піц до кошика
- оформлення замовлення
- перегляд списку замовлень, користувачів, популярних позицій з меню адміністратором.

Використання React, Express.js та MongoDB забезпечило високу продуктивність і легке масштабування. Додаток успішно протестовано, функціонал працює стабільно, а вибрані технології підтвердили свою надійність і ефективність.

Подальший розвиток системи може передбачати розширення функціональності, наприклад, додавання модуля оформлення доставки замовлень із вибором адреси та часу доставки. Також можливе впровадження відстеження статусу замовлення, онлайн-оплати та особистих кабінетів для користувачів з історією замовлень.

Рисунок В.15 – Слайд 15