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

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
(освітня програма «Комп'ютерна графіка та розробка ігор»)
на тему:

«Веб-додаток для замовлення та доставки їжі»

Виконав студент групи ЗКН-19-1
ЧЕНЧЕВИЙ Денис Сергійович

Керівник роботи:
ГРАФ Марина Сергіївна

Рецензент:
СУГОНЯК Інна Іванівна

Житомир – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук
Марина ГРАФ
«05» січня 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу освітнього ступеня «бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
(освітня програма «Комп'ютерна графіка та розробка ігор»)

Здобувач вищої освіти: **ЧЕНЧЕВИЙ Денис Сергійович**

Керівник роботи: **ГРАФ Марина Сергіївна**

Тема роботи: **«Веб-додаток для замовлення та доставки їжі»**,

затверджена наказом закладу вищої освіти від **«05» січня 2023 р., №11/с**

Термін здачі закінченої роботи: «26» червня 2023 р.

Вихідні дані роботи: об'єктом дослідження є веб-додаток, сервіс для замовлення та доставки їжі. Предметом дослідження є веб-система, базована на сучасних підходах розробки з використанням різних фреймворків та бібліотек.

Консультанти випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Дата	
		Завдання видав	Завдання прийняв
1	Граф М.С.	05.02.2023р.	05.02.2023р.
2	Граф М.С.	20.05.2023р.	20.05.2023р.
3	Граф М.С.	10.06.2023р.	10.06.2023р.

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі. Пошук, огляд та аналіз аналогічних розробок. Формулювання технічного завдання. Опрацювання літературних джерел.	2 лютого 2023 – 1 квітня 2023	Виконано
2	Проектування структури системи	1 квітня 2023 – 16 травня 2023	Виконано
3	Написання програмного коду	12 травня 2023 – 15 червня 2023	Виконано
4	Тестування системи	15 червня 2023 – 20 червня 2023	Виконано
5	Оформлення пояснювальної записки	20 червня 2023 – 25 червня 2023	Виконано

Здобувач вищої освіти **Денис ЧЕНЧЕВИЙ**
Керівник **Марина ГРАФ**

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи на тему «Веб-додаток для замовлення та доставки їжі» складається з переліку умовних скорочень, вступу, трьох розділів, висновків, списку використаної літератури та додатку.

Текстова частина викладена на **61** сторінках друкованого тексту.

Пояснювальна записка має **2** сторінки додатків. Список використаних джерел містить **21** найменування і займає **2** сторінки. В роботі наведено **48** рисунків. Загальний обсяг роботи – **66** сторінок.

В розділах роботи були визначені вимоги та основні критерії до розроблювальної системи, досліджені аналогічні програмні продукти, обрано стек технологій та архітекторний підхід. Наступним чином було спроектовано систему, основні варіанти використання та загальний алгоритм роботи. На основі проектування було розроблено функціонал серверної частини додатку, його було протестовано, в результаті було отримано готовий бекенд. Маючи сервер, було розролено клієнтську частину веб-додатка, були використані та описані методи розробки фронтенду. Було описано хід роботи з веб-додатком, оглянуто різні сценарії його використання.

Висновок містить в собі результати виконаної роботи по розробці веб-додатку для замовлення та доставки їжі.

ABSTRACT

The explanatory note to the qualification work on the topic "Web application for ordering and delivery of food" consists of a list of abbreviations, introduction, three chapters, conclusions, a list of references and an additions.

The text part is set out on 61 pages of printed text.

The explanatory note has 2 pages of appendices. The list of references includes 21 titles and occupies 2 pages. The paper contains 48 pictires. The total volume of the work is 66 pages.

In the chapters of the work, the requirements and basic criteria for the system under development were defined, similar software products were studied, the technology stack and architectural approach were selected. Next, we designed the system, the main options for use and the general algorithm of operation. Based on the design, we developed the functionality of the server side of the application, tested it, and finally got a ready-made backend. With the server in place, the client part of the web application was developed, and frontend development methods were used and described. The process of working with the web application was described, and various scenarios of its use were reviewed.

The conclusion contains the results of the work done to develop a web application for ordering and delivering food.

ЗМІСТ

РЕФЕРАТ 3
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 6
ВСТУП 7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ, ЗАСОБІВ ТА МЕТОДІВ ВИРІШЕННЯ ЗАДАЧІ 9
1.1 Постановка задачі 9
1.2 Аналіз аналогів програмного продукту 10
1.3 Обґрунтування вибору інструментальних засобів розробки 15
Висновки до розділу 1 18
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ФУНКЦІОНАЛУ СИСТЕМИ 19
2.1 Варіанти використання веб-додатку, алгоритми роботи системи 19
2.2 Використання бази даних та її структура 23
2.3 Реалізація функціоналу та тестування серверної частини веб-додатку 27
Висновки до розділу 2 44
РОЗДІЛ 3. ПОБУДОВА ІНТЕРФЕЙСУ, ПОРЯДОК РОБОТИ З ВЕБ-ДОДАТКОМ 45
3.1 Користувацький інтерфейс, методи його побудови 45
3.2 Порядок роботи з веб-додактом для замовлення їжі 51
Висновки до розділу 3 60
ВИСНОВКИ 61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 62
ДОДАТОК 64

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – Application Programming Interface, використовується для взаємодії між клієнтом та сервером, дозволяє отримувати та відправляти дані.

SPA - Single Page Application, веб-додаток, який завантажує лише одну HTML сторінку.

JWT - JSON Web Token, зберігає інформацію про сесію та дозволи, використовується для аутентифікації.

UI – User Interface, користувацький інтерфейс.

UE – User Experience, користувацький досвід.

БД – база даних.

CRUD – Create, Read, Update, Delete, це основні методи створення, отримання, оновлення та видалення даних з БД.

ВСТУП

Світ став швидким і динамічним, люди не мають багато часу, але при цьому їм потрібно швидко і зручно задовольняти свої потреби, включаючи харчування.

Паралельно зі зростанням темпу – збільшується вплив технологій на всі сфери життя. Інтернет став основою для різних потреб, починаючи від спілкування, закінчуючи покупкою різних товарів. До цього можемо додати і сервіси, які дають можливість людям, без втрат часу, замовляти різні речі.

Поєднавши потреби та технології, отримуємо можливіть для створення платформи, яка буде задовольняти потреби людей, а саме потреби в харчуванні. Такі сервіси відомі як «онлайн-доставка їжі» або «онлайн замовлення їжі».

Дані сервіси вже існують, вони активно розвиваються і наразі можна зазначити той факт, що вони сильно змінили сприйняття харчування, відкрили нові можливості для таких бізнесів, як ресторани, кур'єрські служби та самого клієнта, споживача.

Така можливість – замовляти їжу, будучи при цьому вдома або на роботі, дає людям час, який можна використати для будь-чого іншого. Можливість замовляти їжу онлайн перетворила спосіб покупки продуктів у більш швидкий та зручний процес.

Тому актуальність теми веб-додатка для замовлення їжі полягає в створенні можливості для споживача отримати доступ до зручного та швидкого замовлення продуктів харчування на фоні швидкого ритму життя та популярності подібних додатків серед людей, які мають обмежені можливості у часі або банально хочуть скуштувати страву з улюбленого ресторану, при цьому не добираючись до нього в самотужки.

Метою кваліфікаційної роботи є проектування та розробка веб-додатку для замовлення та доставки їжі, яка буде включати в себе такі етапи та завдання:

- аналіз задач, які має виконувати система;
- аналіз подібних аналогів, визначення критерії для розробки;
- аналіз та обґрунтування засобів реалізації веб-сайту;

- проектування системи, її розробка на основі проведених досліджень;

- тестування та налагодження процесів.

Об’єктом дослідження є веб-додаток для замовлення їжі, який надасть зручний, швидкий та надійний спосіб покупки їжі з різних закладів. Рогляд різних аспектів створення додатку з визначеним функціоналом, що включає безпосередньо функціональну частину, користувацький інтерфейс, швидкодію тощо.

Предметом дослідження є використання популярних та надійних технологій для розробки та вдосконалення веб-додатку для замовлення їжі. Дослідження фундаментальних засобів розробки, може включати в себе базу даних, серверну та клієнтську сторони веб-додатку тощо.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ, ЗАСОБІВ ТА МЕТОДІВ ВИРІШЕННЯ ЗАДАЧІ

1.1 Постановка задачі

Основна мета проекту – створення зручного та ефективного інструменту, веб-додатку, що дозволить споживачам легко замовляти їжу від різних закладів, таких як ресторани та кафе.

Додаток має надавати зручний, інтуїтивно зрозумілий та швидкий інтерфейс, що надасть клієнту доступ до перегляду закладів, їх товарів та відповідно можливість замовлення страв.

У випускній роботі необхідно розробити веб-додаток для замовлення їжі, який включає в себе чимало моментів розробки для виконання яких треба виставити певні вимоги до розроблювальної системи, які допоможуть визначати основні критерії для створення додатку відповідно до зазначеної теми:

- розробка системи відповідно до обраної архітектури додатку;

- керування ресторанами та меню, можливість це робити напряму на сайті або з певних додатків розробника, це може бути наприклад Postman;

- збереження даних у БД;

- зручний інтерфейс для клієнта, повинен бути простим та зрозумілим у використанні, забезпечувати простий спосіб навігації по веб-сайту, адаптивність;

- фільтрація і пошук, оскільки додаток може містити велику кількість закладів, які в свою чергу ще більше продуктів, то функціонал фільтрації товарів та пошуку закладів просто в край необхідний;

- кошик замовлень, клієнт має мати можливість додавати обрані страви до кошика, переглядати її зміст, змінювати кількість страв або видаляти їх, оформлювати замовлення;

- масштабованість, система має мати змогу легко додавати функціонал без втрат у продуктивності чи працездатності системи загалом.

1.2 Аналіз аналогів програмного продукту

В попередньому підрозділі були визначені основні вимоги для розроблювальної системи. Маючи їх, можемо розглянути аналогічні програмні продукти, що дозволить на підставі проведеного аналізу визначати найкращі практики та стандарти, які вже існують на ринку. Можна буде виявити певні методики для розробки власного проекту, моменти з рішень у побудові користувацького інтерфейсу. Виявивши переваги та недоліки аналогів, зможемо приймати зважені рішення для вибору засобів розробки для побудови якісної системи.

Розглянемо декілька аналогічних програмних продуктів, акцентуючих більше на можливостях веб-сайтів, ніж загалом за сервіс, адже це трохи інша бізнес частина.

1) Mister.Am – досить масштабний сервіс для замовлення і доставки страв [1]. Співпрацюють з ресторанами та доставляють замовлення безпосередньо до клієнтів.

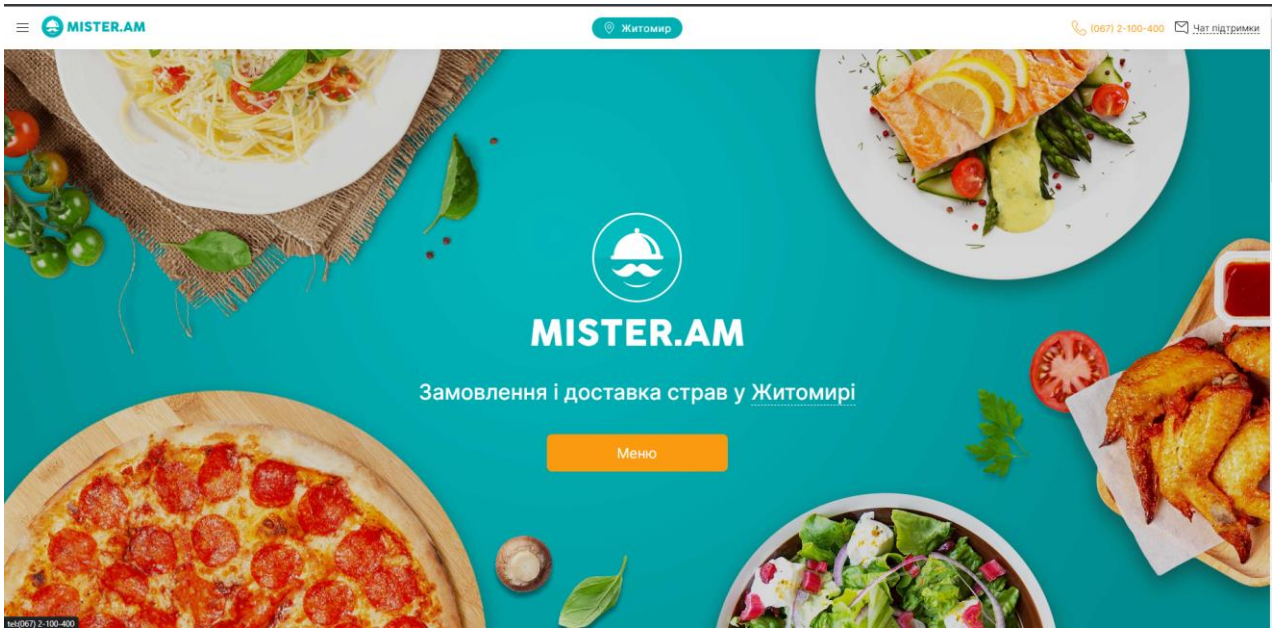

Рис.1.1. Головна сторінка вебсайту

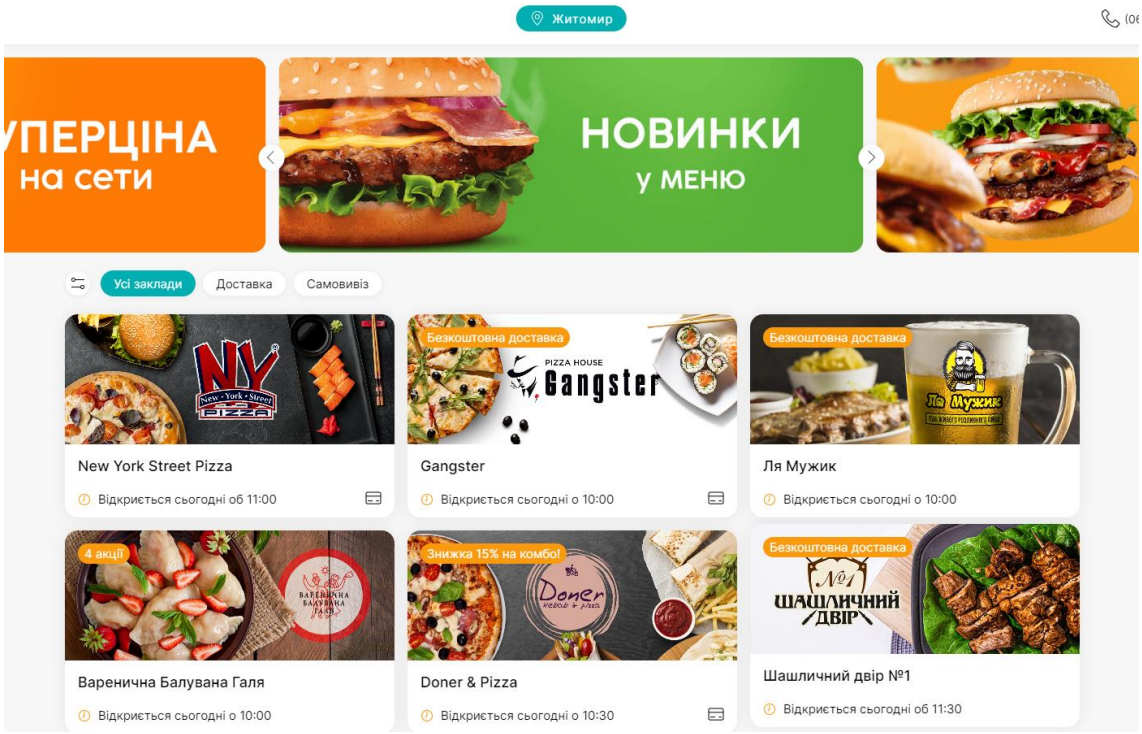

Рис.1.2. Сторінка закладів

Переваги:

- широкий вибір ресторанів, дає різноманітний вибір страв;
- зручний інтерфейс, коректно працює;
- зручний кошик товарів, зберігає дані при перезавантажені сторінки;
- зручна фільтрація.

Недоліки:

- маючи великий асортимент закладів і товарів - відсутній пошук.

2) Час поїсти – онлайн платформа замовлення їжі, спеціалізуються на своєму власному закладі [2].

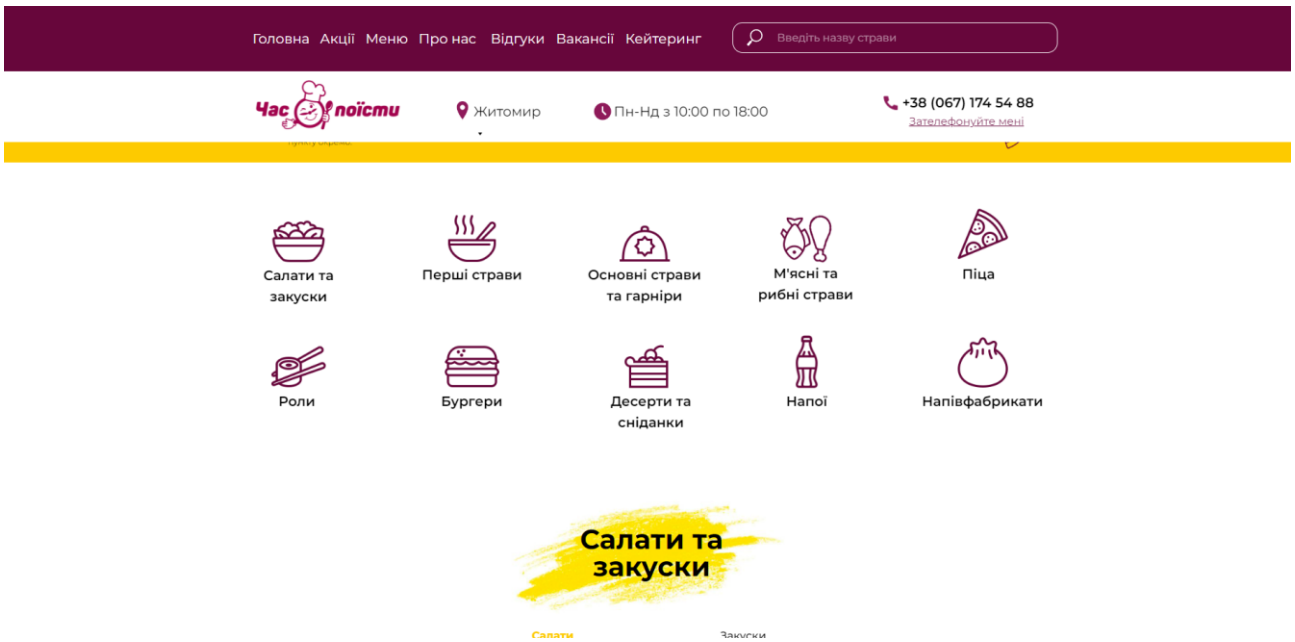

Рис.1.3. Головна сторінка веб-сайту

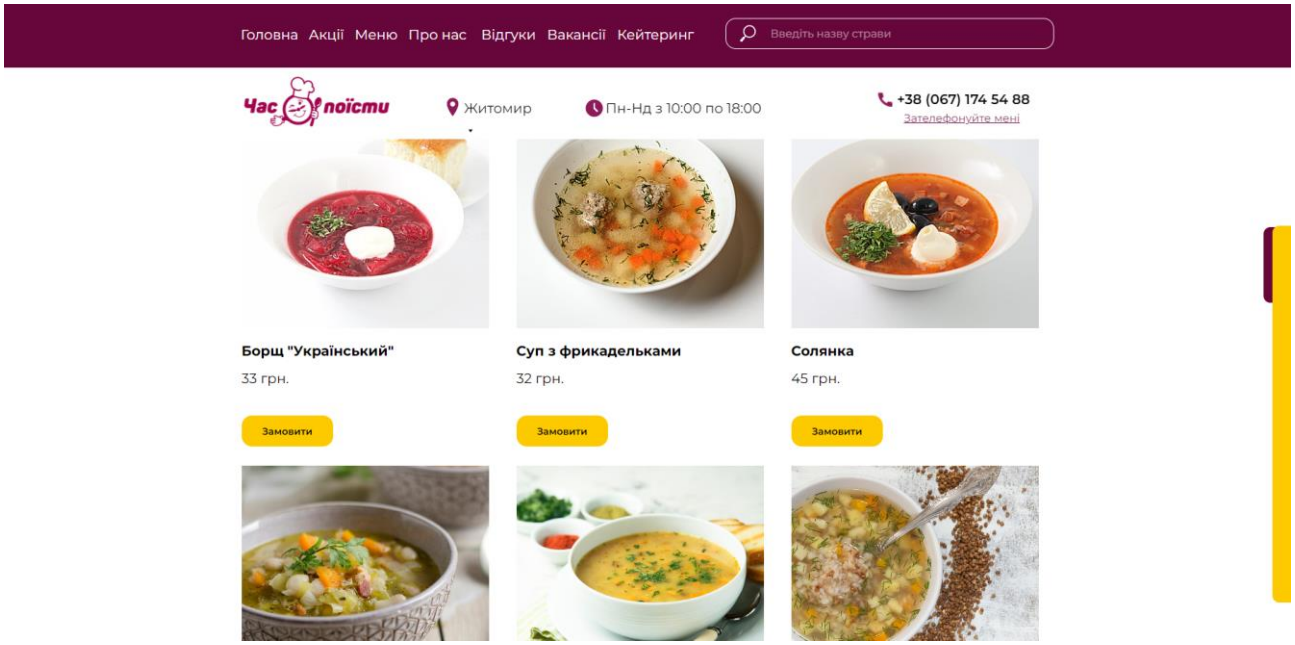

Рис.1.4. Меню страв за категорією

Переваги:

- зручний та швидкий інтерфейс, працює без багів;

- багато інформації про магазин, акції і доставку, якщо є питання у клієнта він зможе знайти відповіді;

- приємний дизайн;

- адаптиність присутня.

Недоліки:

- якщо споживач не знайомий безпосередньо з продукцією магазину, то він не отримає повний обсяг інформації про страву, бо її недостаньо;

- при додавані страви у кошик, модальне вікно корзини одразу відкривається, воно містить не тільки перелік страв, а й одразу інпути для замовлення, для кращого UE варто було б зробити кошик на окремій сторінці, а на інших зазначати лише кількість товарів у корзині.

3) Hz-food (Хочу з'їсти) – вебсайт для замовлення їжі з мережі магазинів «Хочу з'їсти», їх власний сервіс замовлень та доставки [3].

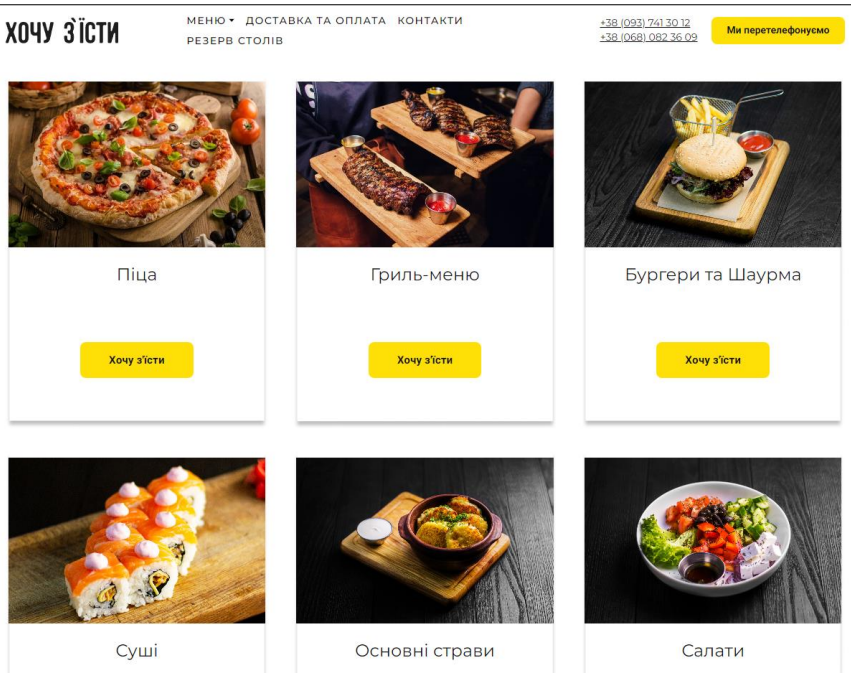

Рис.1.5. Головна сторінка веб-сайту

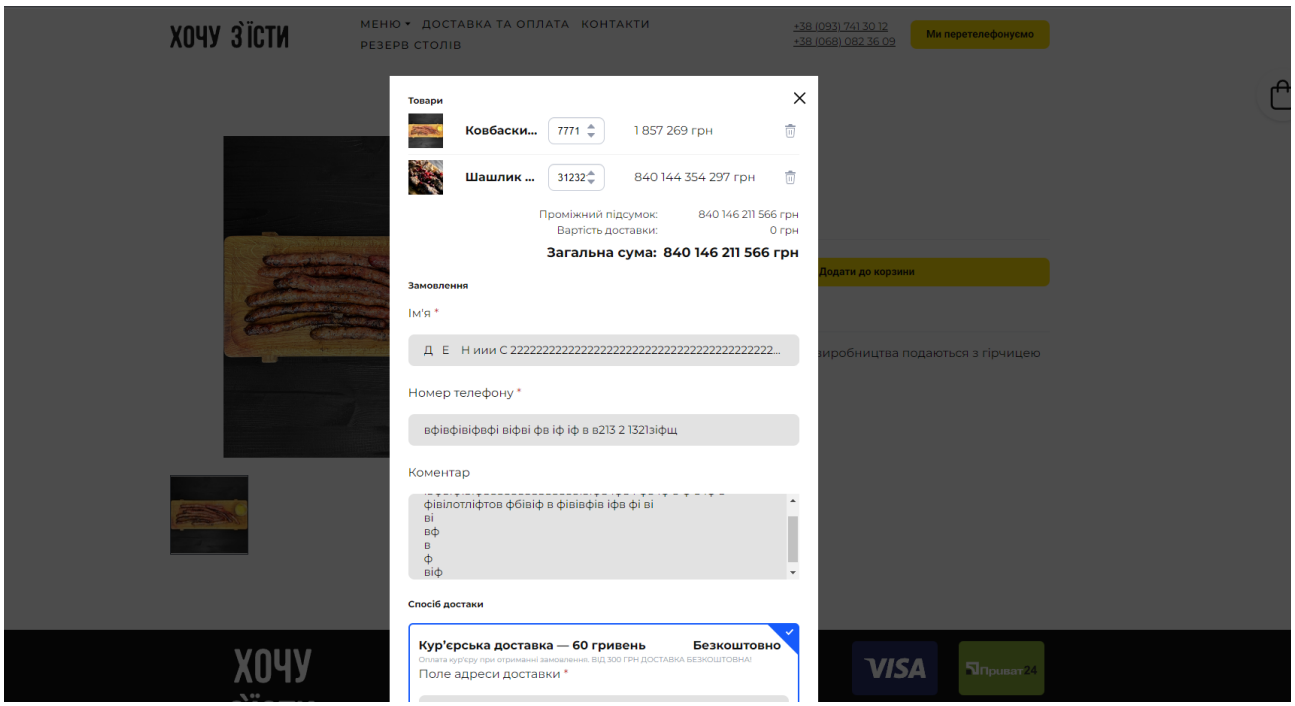

Рис.1.6. Модальне вікно кошика обраних товарів

Переваги:

- простий і приємний дизайн, фірмова стилістика;
- зрозумілий UI, налаштований респонсів;
- зручний кошик, при вибору товара, видно, що товар додався у корзину;
- не нагромаджують інформацією, але вона є, якщо користувач захоче почитати наприклад про гарячі страви – він прогортає вниз і побачить даний блок інформції.

Недоліки:

- не виявлено, для своїх задач – вебсайт має весь потрібний функціонал в поєднанні зі зручністю використання.

Таким чином, було розглянуто три аналогічних програмних продукта, виявлено їх плюси та мінуси, і зроблено висновок, що поставлені вимоги були вірні, сайти мають спільні риси, характерні для роду діяльності, тому можемо взяти найкраще для розробки власного веб-додатку.

1.3 Обґрунтування вибору інструментальних засобів розробки

IT сфера досить динамічна, адже бурхливо та стрімко розвивається. Ми маємо багато іновацій та різних технологій які спрощують та покращують розробку.

Орієнтуючись на вимоги до проекту, вибір мови програмування для веб додатку особливої ролі не грає, але оскільки ми базуємось на сучасні технології, то у сторону PHP не варто дивитись, не зважаючи на його переваги, такі як велика база готових рішень через об'ємну спільноту чи простоту та вбудовану підтримку БД, чи зручні фрейморки. Краще буде обрати JavaScript, адже з допомогою JS ми можемо побудувати додаток з клієнт-серверним архітектруним підходом, використовуючи такі технології, як платформу Node.js та бібліотеку React.

JavaScript широко використовується, зараз це найпопулярніша мова програмування. З її допомогою ми можемо користуватись зручними та сучасними фрейморками та бібліотеками, поєднуючи їх між собою для побудови гнучких веб-додатків. Можемо глянути на статистику використання мов програмування у комерційних проектах [4]:

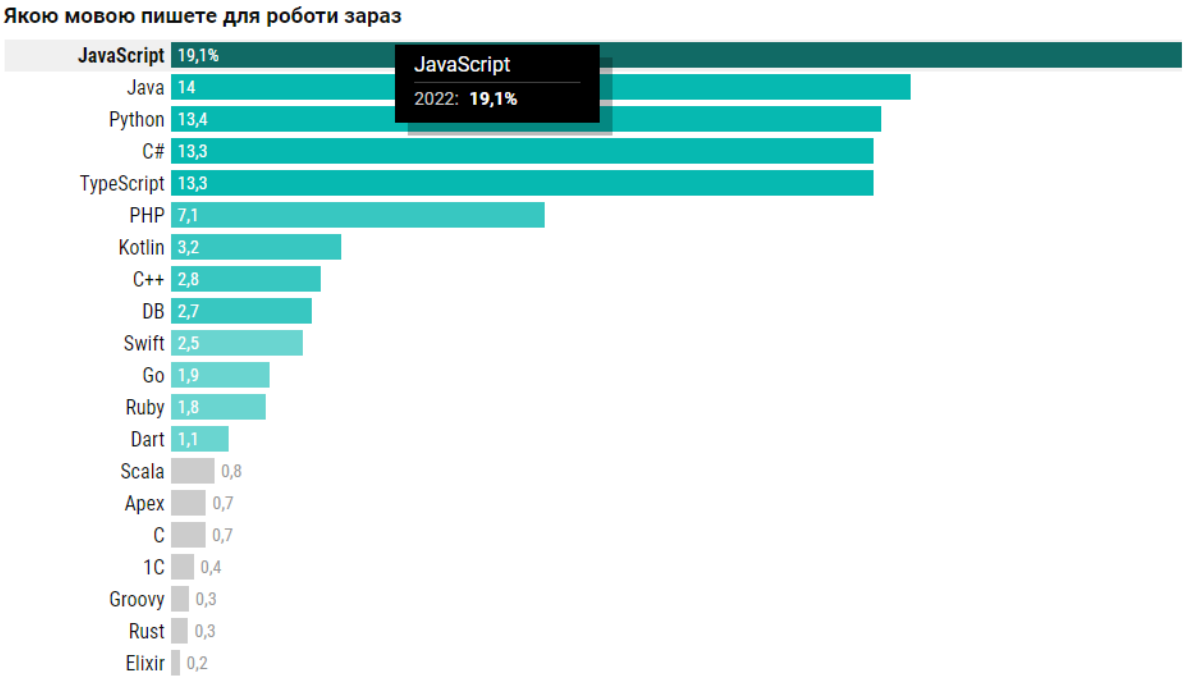

Рис.1.7. Статистика використання мов програмування у комерційних цілях за 2023 рік, взято з DOU

Також за допомогою JS ми можемо використовувати асинхронний підхід до певних моментів розробки, що є корисним для обробки багатоопераційних завдань та взаємодії з сервером [5]. Це дозволить працювати системі більш реактивно та продуктивно. Асинхронний підхід дає можливіть обробляти операції і не блокувати виконання інших частин коду, це також в свою чергу запобігає блокування UI.

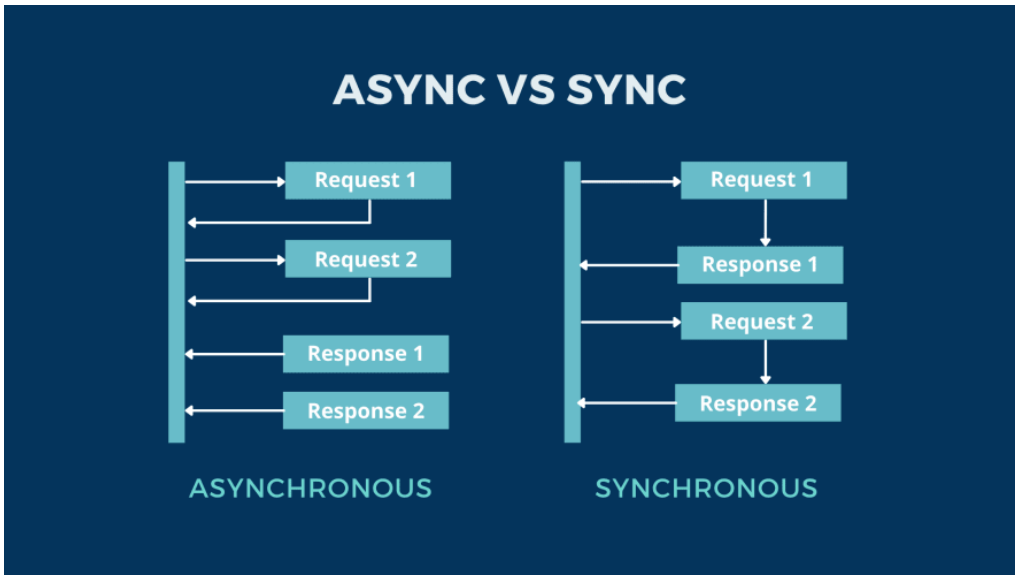

Рис.1.8. Різниця асинхронного і синхронного підходів

Оскільки користуємось JS, можемо використовувати клієнт-сервений архітектурний підхід до розробки, тобто розділимо проект на сервер(бекенд) та клієнт(фронтенд).

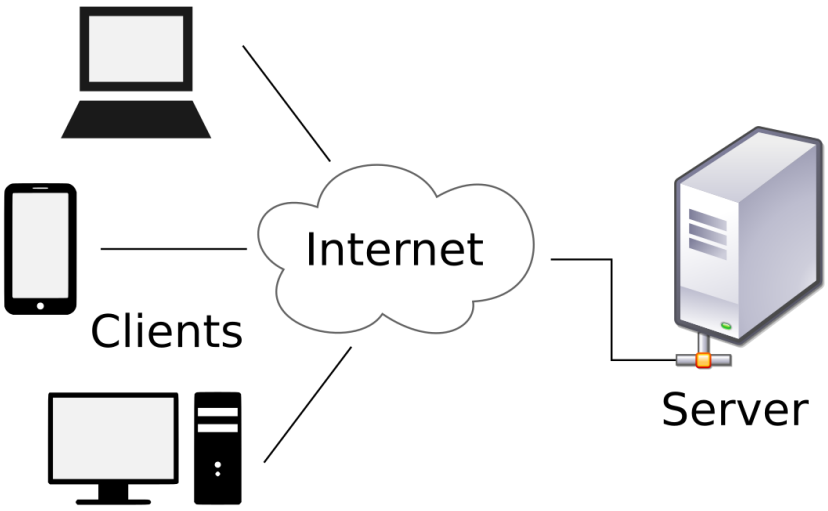

Рис.1.9. Клієнт-серверна модель

Один з самих популярних стеків технологій для реалізації клієнт-серверної моделі є стек MERN. Це комбінація чотирьох технологій для розробки веб-додатків:

- MongoDB, документно-орієнтовна база даних, зберігає дані у вигляді документів у форматі JSON [6]. Це гнучка БД, яка дає високу продуктивність та зручну СУБД MongoDB Atlas – сервіс керування хмарної БД;

- Express.js, простий, легкий та гнучкий фрейморк для Node.js [7]. Надає зручні інструменти для створення серверної частини веб-додатку, одна з головних це маршрутизація, обробка запитів тощо;

- React, сама смачна частина даного стеку, надзручна та універсальна бібліотека для створення UI. Користувацькі інтерфейси, побудовані за допомогою реакту є реально реактивні. Бібліотека надає зручні інструменти для побудови інтерактивних інтерфейсів [8]. Також має чудову екосистему, тобто є безліч пакетів і бібліотек, які можна завантажувати та поєднувати між собою;

- Node.js, середовище виконаня JS на сервері [9]. Забезпечує можливість використання JS на стороні сервера, весь стек буде на цій мові. Дозволяє будувати гнучін та масштабовані додатки.

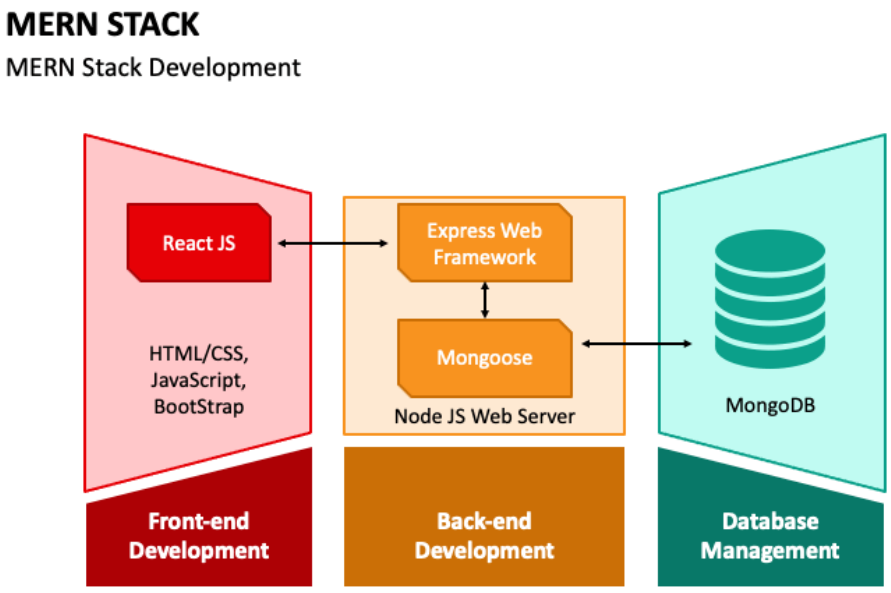

Рис.1.10. Стек MERN

Висновки до розділу 1

Був проведений аналіз предметної області, були визначені вимоги на основі яких було проведено дослідження аналогічних веб-додатків згідно тематики кваліфікаційної роботи. Це дозволило виділити характерні риси основного функціоналу, який має мати сервіс для замовлення їжі.

Маючи основні критерії для розроблювальної системи, було обрано актуальну мову програмування для вирішення поставлених задач, це мова програмування JavaScript. Базуючись на цей вибір, було обрано клієнт-серверний архітектурний підхід, як основу проекта. Було обрано стек технологій, який зручно підходить для даного архітектурного підходу, MERN – MongoDB, в якості бази даних, Express.js, фреймворк для створення сервера, React, фронтенд функціонал та Node.js, середовище для виконання JS коду на сервері.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ФУНКЦІОНАЛУ СИСТЕМИ

2.1 Варіанти використання веб-додатку, алгоритми роботи системи

Опишемо моменти проектування веб-додатку. Визначивши основні можливості функціоналу та графічно побачивши це все – зможемо приступити до якісної та правильної розробки системи.

Розглянемо можливі варіанти використання та проілюструємо їх на відповідних діаграм. Це допоможе нам виявити потреба користувача, визначемо функціонал різних ролей, що дозволить коректно спланувати подальшу розробку.

Діаграма варіантів використання (Use Case Diagram) - це графічне зображення, яке використовується для моделювання функціональності системи, додатку з точки зору її взаємодії з акторами, це можуть бути користувачі або зовнішні системи або фактори, виставлення вимог цих акторів. Діаграма варіантів використання допомагає уявити, як система повинна працювати, які операції та функціональні можливості можуть виконуватися та які актори взаємодіють з системою.

У розроблювальні системі передбачається декілька ролей, щоб оптимізувати та обмежити хід роботи з додатком для різних користувачів. Визначмо такі ролі як користувач(звичайний клієнт або споживач), адміністратор та власник (той самий адміністратор тільки з можливістю керування іншими користувачами з доступом адміна).

Для ролі звичайного користувача можемо виділити такі вимоги як:

1. Перегляд закладів;
2. Пошук закладів;
3. Фільтрація закладів;
4. Перегляд кошику та оформлення замовлення;
5. Фільтрація товарів;
6. Перегляд інформації про заклад;
7. Перегляд товарів відповідного закладу;
8. Додавання товару у кошик.

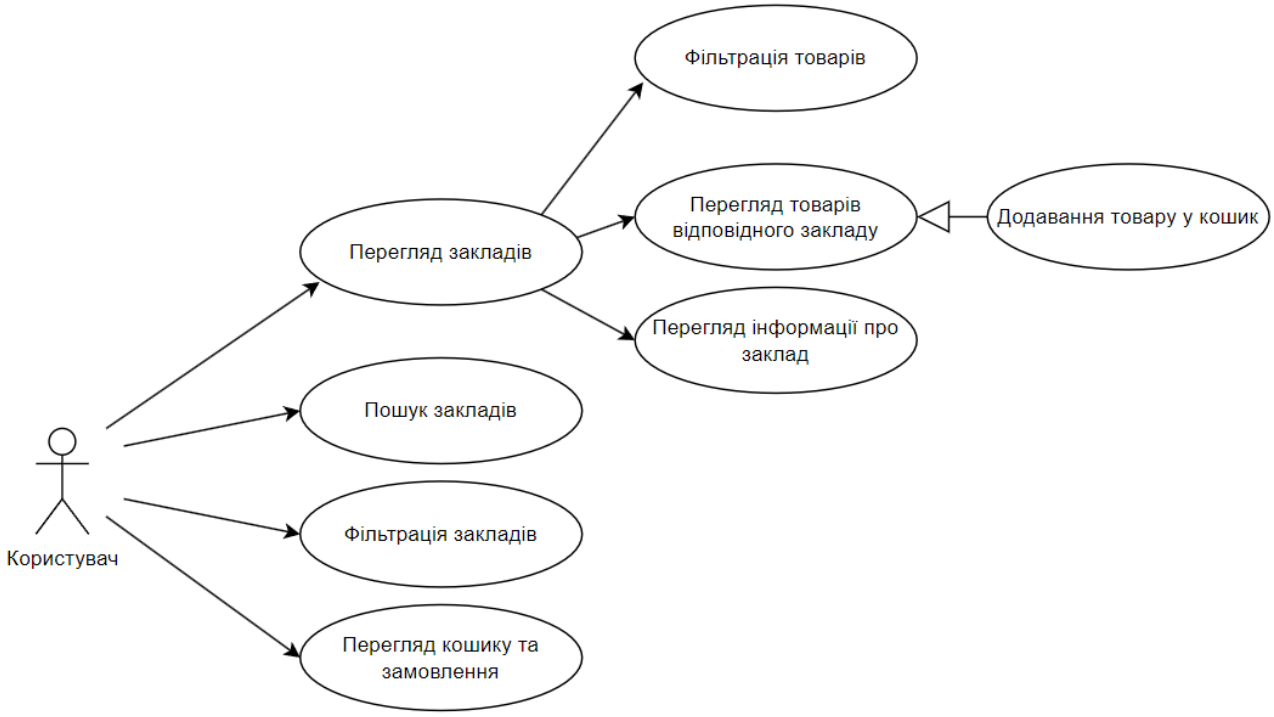

Рис.2.1. Діаграма варіантів використання для ролі користувача

Для ролей адміністратора та власника варінати використання можуть бути як і для звичайного користувача, але розглянемо саме їх унікальні можливості у використанні веб-додатку, який не має клієнт:

1. Дії над закладами;
2. Додавання, редагування та видалення даних про заклади;
3. Дії над продуктами;
4. Додавання, редагування та видалення даних про продукти;
5. (тільки для власника) Дії над адміністраторами;
6. (тільки для власника) Додавання нового адміністратора;
7. (тільки для власника) Змінити роль адміністратора;
8. (тільки для власника) Видалення адміністратора.

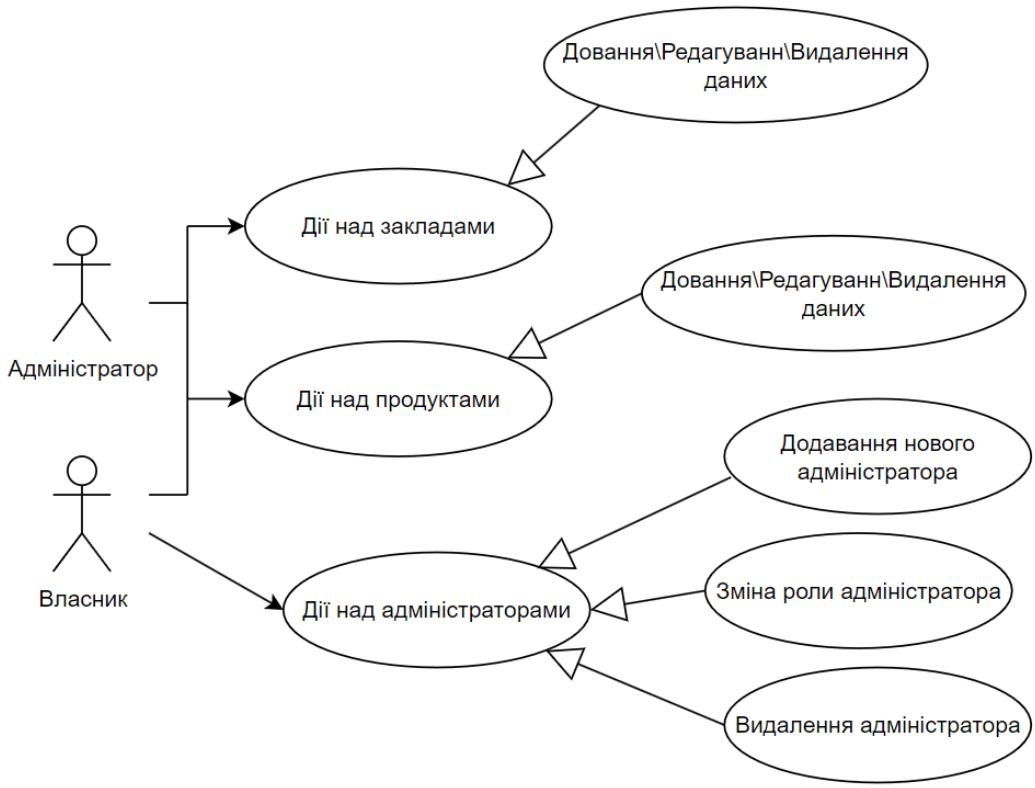

Рис.2.2. Діаграма використання для ролі адміністратора та власника

Також спроектуємо загальний алгоритм роботи веб-додатку. Зробимо це за допомогою діаграми діяльності.

Діаграма діяльності (Activity Diagram) - це графічний інструмент моделювання та проектування, який використовується для представлення, візуалізації послідовності дій, процесів або поведінки системи, компонента чи користувача. Вона показує, які дії виконуються в системі і як вони взаємодіють одна з одною.

Основні елементи діаграми діяльності включають, дії, розгалуження, з'єднувальні стрілки, початкова точка та кінцева точка. Вони допомагають відображати послідовність дій та взаємозв'язки між ними.

Проектування алгоритму роботу системи за допомогою діаграми дільності дасть чітко зрозуміти послідновність дій в системі, одразу виділити недоліки, налагодити процеси, щоб на моменті розробки не виникало критичних труднощів у побудові гнучної системи.

Опишемо створену діаграму діяльності. Початкова точка – відкриття, завантаження головної сторінки сайти, де ми переходимо в меню. Після цих дій відкриється сторінка закладів, де ми можемо роботи певні дії, а саме пошук закладів та здійснювати фільтрацію за категорією. У разі введеня в полі пошуку закладу якого не існує, ми зможемо повернутись до моменту відкриття меню. Переглянувши заклади, можемо обрати один з них, що призведе до переходу сторінки товарів відповідного закладу. Там ми також маємо певні дії, а саме фільтрацію за типом страви, продукту. Користувач може переглянути інформацію про страву та додати певну кількість до кошика. Також з будь якої точки вебсайту – ми можемо перейти до кошика, що по суті і буде сторіною замовлення де клієнт зможе зробити своє остаточне замовлення.

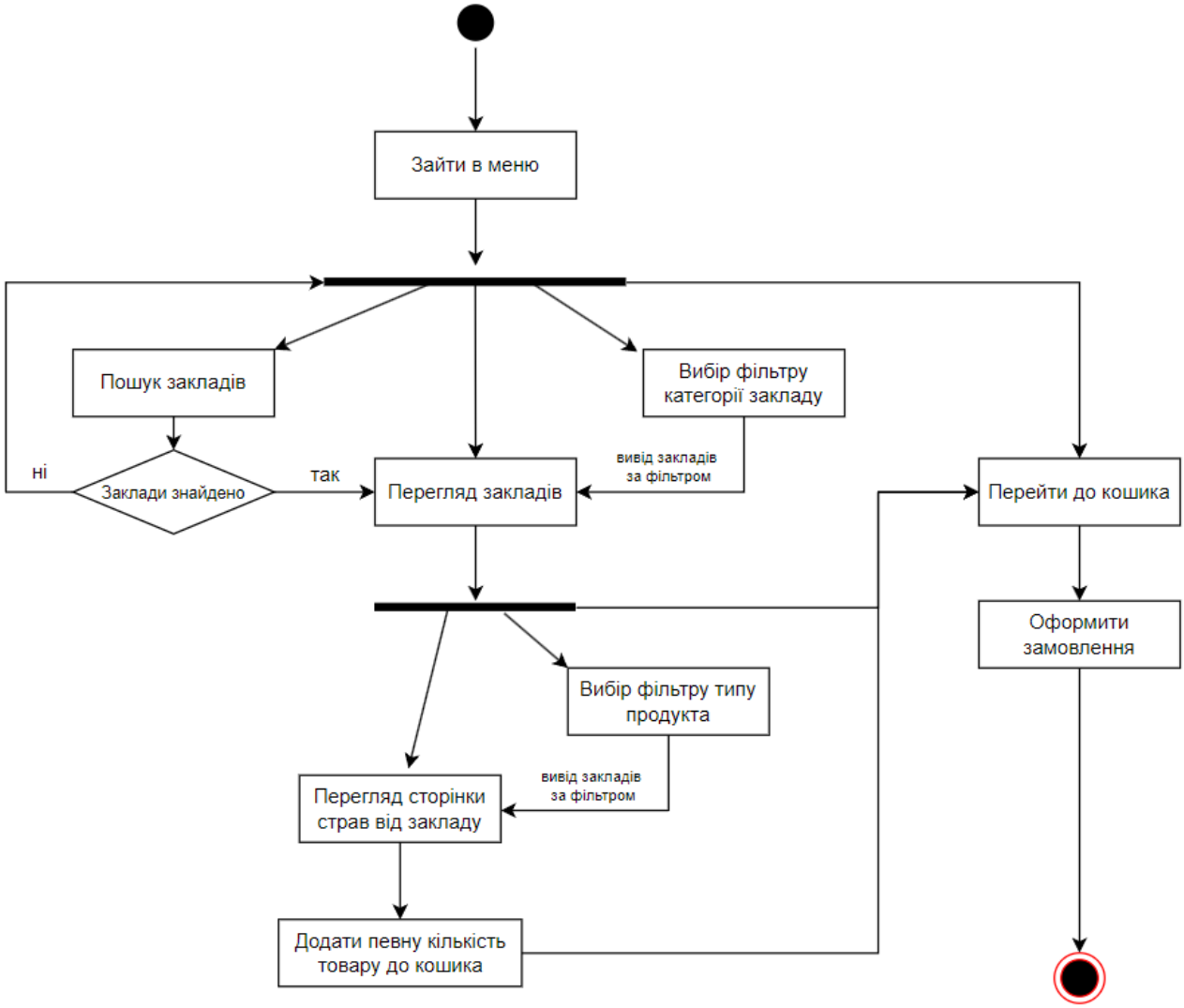

Рис.2.3. Діаграма діяльності, загальний алгоритм роботи веб-додатку

Також хотілося б бачити такий функціонал: якщо в кошику є товари з певного закладу і ми повертаємось в меню ресторанів, то інші заклади або блокуються або буде, наприклад модальне вікно з повідомленням, що товари можна замовляти лише з одного закладу харчування.

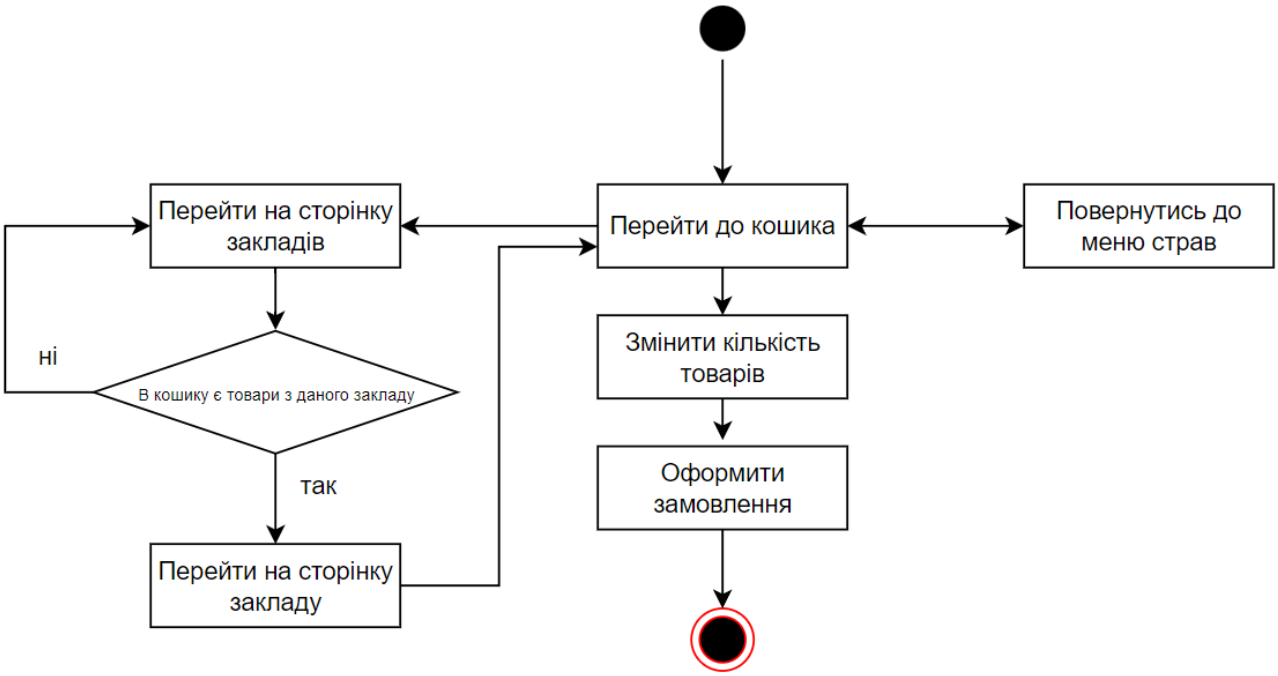

Рис.2.4. Діаграма діяльності, використання кошика

2.2 Використання бази даних та її структура

В розробці кваліфікаційної роботи на тему «Веб-додаток для замовлення та доставки їжі» будемо використовувати базу даних – MongoDB.

Як було зазначено в першому розділі, це документоорієнтовна баа даних, надає гнучкість на швидкість для зберігання та обробки даних. Виділимо головні переваги даної БД:

- гнучкість схем, MongoDB дозволяє зберігати різні дані без конкретних вимог до структури. Можна легко змінювати схему, додавати або видаляти поля в документах, при цьому не треба перероблювати всю колекцію даних;

- простота використання, MongoDB має досить простий та зрозумілий синтаксис запитів, це спрощує розробку, полегшує взаємодію сервера з БД;

- швидкість, база даних швидко працює, конекст відбувається без проблем;

- висока доступність, можна розділяти кластери на різні сервери, це забезпечує гнучкість у разі потреби у розширенні системи.

Для проектування та зображення структури БД для розроблювальної системи, графічно зображимо базу на діаграмі бази даних.

Діаграма бази даних - це графічне зображення структури бази даних, яке використовується для візуалізації зв'язків між таблицями, в нашому випадку колекціями, атрибутами та їх типами даних. Вона надає візуальну репрезентацію схеми бази даних, візуальну уяву про схему БД, допомагає зрозуміти її структуру, взаємозв'язки та залежності.

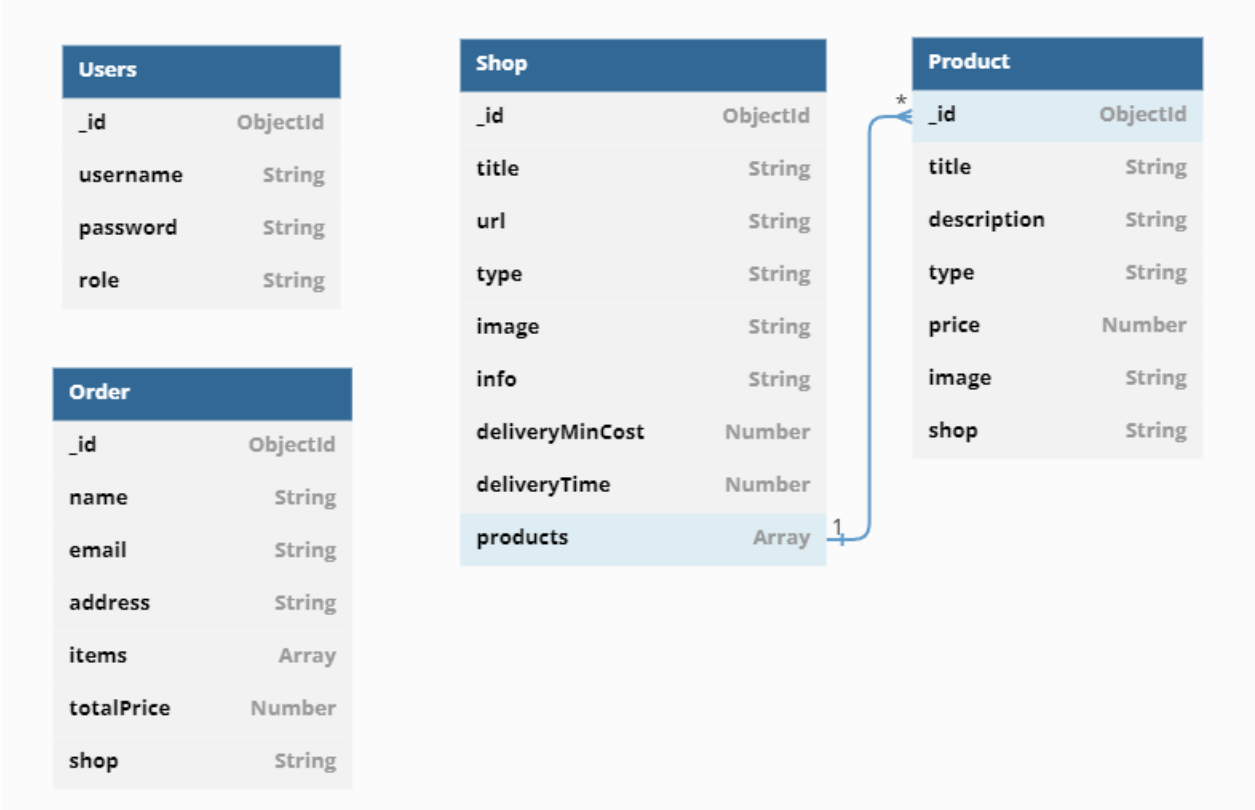

Рис.2.5. Діаграма бази даних

В цілому дана БД повністю задовольняє вимоги до проекту, у разі необхідності, додати якусь колекцію не буде проблемаю, MongoDB легко дасть зробити таку дію.

Розглянемо кожну колекцію, їх поля, типи даних.

Колекція Users – містить дані про акаунт працівника сайту, адміністратора. Передбачається, що звичайний користувач немає свого акаунту, але дана

колекція потрібна, адже треба мати на сайті блок, панель керування, яка буде мати обмежений доступ, тобто тільки адміністратор або власник зможе нею скористатись, тому дана колекція буде мати мінімальний набір полів, які потрібні для авторизації задля отримання доступу до панелі керування. В колекції маємо такі поля:

- _id: ідентифікатор, поле має тип даних *ObjectId* який надає MongoDB;
- username: нікнейм, псевдонім адміністратора, унікальне поле, використовується як логін для авторизації, тип *String*;
- password: пароль до акаунту, потрібен для авторизації, тип *String*;
- role: роль акаунту, оскільки треба мати можливість додавати адміністраторів, то бажано зробити роль власника, який зможе мати цю можливість, тому було вирішено виділити окреме поле для перевірки ролі задля надання доступу. На даний момент передбачається роль адміністратора та власника, тип *String*.

Колекція Shop – містить дані про заклади, які будуть надавати меню страв та продуктів. В колекції маємо такі поля:

- _id: ідентифікатор, поле має тип даних *ObjectId* який надає MongoDB;
- title: назва закладу, тип *String*;
- url: унікальне поле для «URL», за ним можна буде визначати товари закладу та робити інші операції, по суті як _id тільки текстом, яке буде мати назву закладу, використовується як шлях до ресурсу, тип *String* (приклад використання для більшої ясності: *http://sample-name/api/products/**shopUrl**?type=*);
- type: тип, категорія закладу, буде використовуватись для фільтрації отриманих даних, тип *String*;
- image: зображення закладу для відображення на стороні клієнта, використовується в дизайні та побудові інтерфейсу, тип *String*;

- info: містить певну інформацію про заклад, яку зможе побачити клієнт на вебсайті, тип *String*;
- deliveryMinCost: треба зазначати мінімальну вартість доставки, інші критерії по типу відстані або можливості врахування доставки за до дверей замовника будуть однакові, а от мінімальна вартість буде відрізнятись, тому треба винести це в БД, тип *Number*;
- deliveryTime: приблизний час доставки, тип *Number*;
- products: список продуктів, страв які має заклад харчування, це масив, містить ідентифікатори відповідних товарів.

Колекція Product – містить дані про продукти, страви, товари до відповідного закладу харчування.

- _id: ідентифікатор, поле має тип даних *ObjectId* який надає MongoDB;
- title: назва страви, продукту, тип *String*;
- description: опис продукту, передбачається, що клієнт зможе при виборі страви, побачити її опис, тип *String*;
- type: тип, категорія продукту, буде використовуватись для фільтрації отриманих даних, тип *String*;
- image: зображення продукту для відображення на стороні клієнта, використовується в дизайні та побудові інтерфейсу, тип *String*;
- shop: містить унікальну назву закладу, використовується для отримання даних відповідного закладу, як шлях до ресурсу, тип *String* (приклад використання для більшої ясності: *http://sample-name/api/products/**shopUrl**?type=*).

Колекція Order – містить дані про замовлення клієнтів. В колекції маємо такі поля:

- _id: ідентифікатор, поле має тип даних *ObjectId* який надає MongoDB;

- name: ім’я клієнта, треба знати на кого робити доставку, як кур’єру звертатись до клієнта, тип *String*;
- email: імейл клієнта, тип *String*;
- phone: номер клієнта, тип *Number*;
- address: адреса на яку клієнт робить замовлення, тип *String*;
- items: безпосередньо обрані продукти замовення, це масив об’єктів, об’єкт містить дані про продукт, його кількість, назву тощо;
- totalPrice: загальна ціна замовлення, тип *Number*;
- shop: відповідно заклад в якому було зроблено замовлення.

2.3 Реалізація функціоналу та тестування серверної частини веб-додатку

В даному підрозділі розглянемо головні моменти при розробці серверної частини веб-додатку для замовлення їжі.

Як було зазначено раніше, ми використовуємо стек MERN. За серверну частину відповідають фреймворк Express.js та Node.js. Принцип роботи даного стеку та основні моменти такі [10]:

- сторона клієнта додатку (фронтенд) розробляється на реакті, яка взаємодіє з сервером (бекендом), відправляючі запити за допомогою HTTP протоколу та методів запиту GET, POST, PUT, PATCH, DELETE;
- серверна частина розробляється на Node.js в поєднанні з Express.js. Створюються серверні маршрути, які обробляють запити від фронтенда та взаємодіють з базою даних MongoDB;
- MongoDB використовується для зберігання та обробки даних.

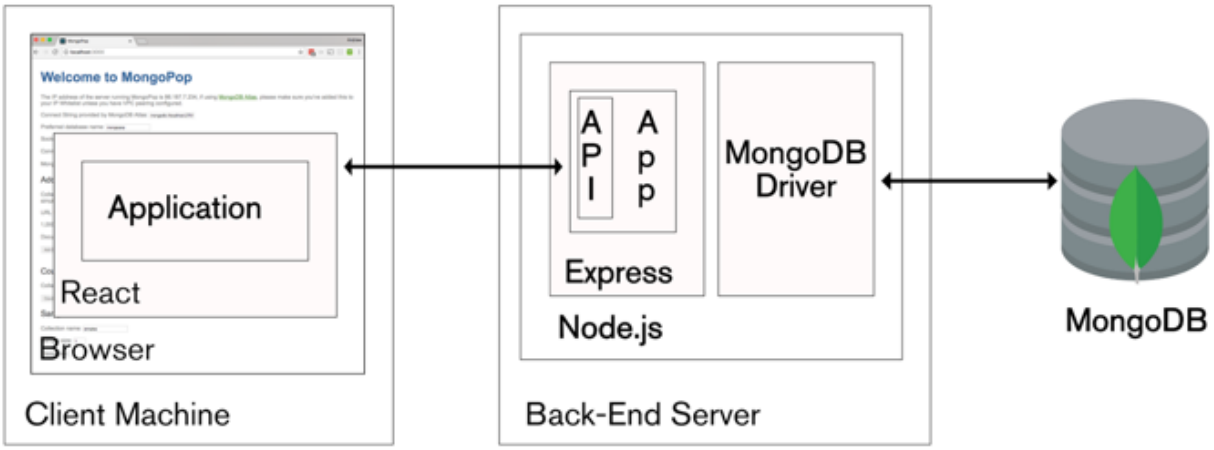

Рис.2.6. Принцип роботи стека MERN

Почнемо з описи структури файлів серверної частини. Кореневий файл це index.js, в ньому ми якраз і створюємо сервер, але писати все в ньому не зручно і не є практичним, тому ми розділяємо код на певні теки, таким чином ми можемо зручно розбивати частини додатку на універсальні шаблони такі як роути, моделі, контролери та middleware (проміжкові обробники коду):

1. Роути (Routes) – визначають URL шляхи, за якими можна здійснювати запити до сервера. Вони встановлюють зв’язки між контролерами, за допомогою яких іде обробка запитів. Методами Exprees.js ми визначаємо роути, встановлюємо відповідні методи;
2. Моделі (Models) – відображають структуру для отримання даних з БД. Використовують для створення, оновлення, видалення та отримання даних, використовують пакет mongoose для роботи з MongoDB;
3. Контролери (Controllers) – обробляють логіку запитів, які надходять до сервера. Взаємодіють з моделями, виконують потрібні операції для обробки запитів. Після виконання операції, повертають певну відповідь клієнту, це може бути як і повідомлення або помилка, так і певні, конкретні дані;
4. Проміжкові обробники коду, надалі просто middleware – функції, які виконуються перед обробкою запиту контролера. Можуть виконувати

певні дії, які потрібні для коректного запиту, по типу це може бути перевірка автентифікації користувача, тобто перевірка доступу. Може бути обробка валідації на сервері, в нашому випадку за допомогою пакета express-validator тощо [11].

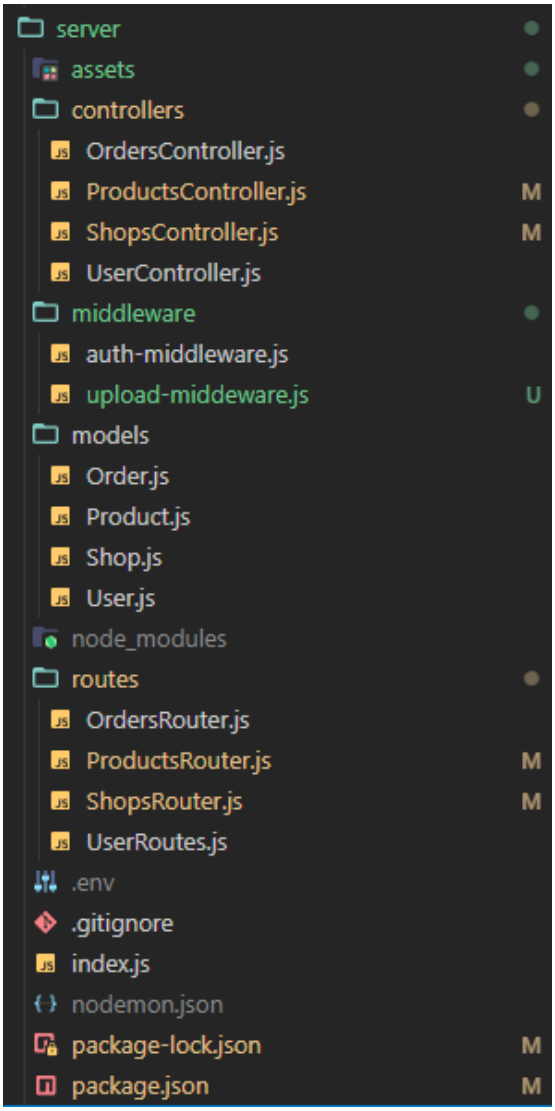

Рис.2.7. Структура файлів серверної частини додатку

Розглянемо основні моменти кореневого файлу index.js. На початку йдуть імпорти потрібних компонентів та пакетів необхідних для роботи:

```
import * as path from "path";
import express from "express";
import bodyParser from "body-parser";
import mongoose from "mongoose";
import cors from "cors";
import ShopsRouter from "./routes/ShopsRouter.js";
import ProductsRouter from "./routes/ProductsRouter.js";
```

```
import OrdersRouter from "./routes/OrdersRouter.js";
import UsersRouter from "./routes/UserRoutes.js";
```

Лістинг.1.1. Імпорт пакетів та необхідних компонентів, а саме роутів

Далі використовуємо ці пакети та роути для створення сервера, оброблюємо вірогідні помилки, встановлюємо статичне положення файлів, які співпадають з потрібним шляхом їх розміщення та підключаємось до БД:

```
const app = express();

app.use(bodyParser.json());

app.use(cors());

app.use("/assets/images", express.static(path.join("assets", "images")));

app.use((error, req, res, next) => {
  res.status(error.code || 500);
  res.json({ message: error.message || "An unknown error occurred!" });
});

app.use("/api/shops", ShopsRouter);
app.use("/api/products", ProductsRouter);
app.use("/api/orders", OrdersRouter);
app.use("/api/users", UsersRouter);

app.use((req, res, next) => {
  res.status(404).json({ message: "Could not find this route" });
});

mongoose
  .connect(
    `mongodb+srv://${process.env.DB_USER}:${process.env.DB_PASSWORD}@cluster0.yxlqh
m6.mongodb.net/${process.env.DB_NAME}?retryWrites=true&w=majority`
  )
  .then(() => {
    app.listen(process.env.PORT || 5000);
  })
  .catch((err) => {
    console.log(err);
  });
```

Лістинг.1.2. Створення сервера та підключенна до бази в кореневому файлі index.js

Розглянемо роут закладу харчування – ShopRouter.js. Імпортуємо пакети та middleware, створюємо роутер та визначаємо основні маршрути та їх методи запитів. В них, за потреби використовуємо middleware. Наприклад для певірки авторизації. Створювати, редагувати та видаляти дані може лише адміністратор або власник. Також присутня валідація вхідних даних. У разі неправильно вводу, буде помилка:

```
import Router from "express";
import ShopsController from "../controllers/ShopsController.js";
import { check } from "express-validator";
import { uploadMiddleware } from "../middleware/upload-middeware.js";
import { authMiddleware } from "../middleware/auth-middleware.js";

const router = new Router();

router.get("/", ShopsController.getShops);

router.get("/types", ShopsController.getShopsTypes);

// router.get("/:shopId", ShopsController.getShopById);
router.get("/:shopUrl", ShopsController.getShopByUrl);

router.post(
  "/",
  authMiddleware,
  uploadMiddleware.single("image"),
  [
    check("title", "Shop name at least 4 characters").isLength({ min: 4 }),
    check("url", "The url of the shope is not entered").notEmpty(),
    check("type", "The name of the shope type is not entered").notEmpty(),
    check("info", "Info is not entered").notEmpty(),
    check("deliveryMinCost", "Delivery minimal cost is not entered")
      .notEmpty()
      .isNumeric(),
    check("deliveryTime", "Delivery time cost is not entered")
      .notEmpty()
      .isNumeric(),
  ],
  ShopsController.createShop
);

router.patch(
  "/:shopId",
  authMiddleware,
  uploadMiddleware.single("image"),
  [
    check("title", "Shop name at least 4 characters").isLength({ min: 4 }),
    check("url", "The url of the shope is not entered").notEmpty(),
```

```
    check("type", "The name of the shope type is not entered").notEmpty(),
    check("info", "Info is not entered").notEmpty(),
    check("deliveryMinCost", "Delivery minimal cost is not entered")
      .notEmpty()
      .isNumeric(),
    check("deliveryTime", "Delivery time cost is not entered")
      .notEmpty()
      .isNumeric(),
  ],
  ShopsController.updateShop
);

router.delete("/:shopId", authMiddleware, ShopsController.deleteShop);

export default router;
```

Лістинг.1.3. Роутер закладів харчування

Опишемо модель закладу харчування. Імпортуємо mongoose для створення схеми, задаємо потрібні поля, вказуємо їх обов'язковість введеня, унікальність (для url) та експортуємо її:

```
import mongoose from "mongoose";

const shopSchema = new mongoose.Schema({
  title: { type: String, required: true },
  url: { type: String, required: true, unique: true },
  type: { type: String, required: true },
  image: { type: String, required: false },
  info: { type: String, required: true },
  deliveryMinCost: { type: Number, required: true },
  deliveryTime: { type: Number, required: true },
  products: [
    {
      type: mongoose.Types.ObjectId,
      required: true,
      default: [],
      ref: "Product",
    },
  ],
});

export default mongoose.model("Shop", shopSchema);
```

Лістинг.1.4. Модель закладу харчування

Розглянемо контролер ProductsController,js, його методи. В CRUD методах ми використовуємо асинхронні функції, суть полягає в тому, що деякі операції, такі як звернення до бази даних, можуть займати певний час для виконання і вимагають очікування результату [15]. У таких випадках асинхронні функції дозволяють виконувати інші операції під час очікування результату виконання асинхронної операції. Ключове слово async перед функцією дозволяє використовувати ключове слово await всередині функції, await вказує JS зупинити виконання функції і чекати на виконання асинхронної операції. Коли результат асинхронної операції готовий, виконання функції продовжується.

Почнемо з методів отримання даних про продукти. Всі так чи інакаше працюють по схожому сценарію, отримуємо дані в реквесті (request), взаємодіємо з ними, звертаємось до БД, повертаємо дані в респонсі (response). Також все огортаємо в блок try – catch, для обробки помилок:

```
async getProducts(req, res) {
    try {
      const products = await Product.find();
      res.status(200).json(products);
    } catch (err) {
      res.status(404).json({ message: err.message });
    }
  }

  async getProductById(req, res) {
    try {
      const productId = req.params.productId;
      const product = await Product.findById(productId);
      res.status(200).json(product);
    } catch (err) {
      res.status(404).json({ message: err.message });
    }
  }

  async getProductsByShopUrl(req, res) {
    try {
      const shopUrl = req.params.shopUrl;
      const productType = req.query.type;

      let query = { shop: shopUrl };

      if (productType) {
        query = { ...query, type: productType };
      }
```

```
      const products = await Product.find(query);
      res.status(200).json(products);
    } catch (err) {
      res.status(404).json({ message: err.message });
    }
  }

  async getProductTypesByShopUrl(req, res) {
    try {
      const shopUrl = req.params.shopUrl;
      const productTypes = await Product.distinct("type", { shop: shopUrl });

      res.status(200).json(productTypes);
    } catch (err) {
      res.status(500).json({ message: err.message });
    }
  }
```

Лістинг.1.5. ProductsController – методи для отримання даних

Метод для створення даних працює таким чином – перевіряємо наявність помилок у вхідних даних. Беремо їх з тіла запита у разі відсутності помилок валідації. Шукаєм чи вже було створено такий продукт. Далі перевіряємо чи вхідні дані про заклад харчування були вірні і такий заклад існує. Наступним чином встановлюємо шлях зображення і нарешті створюємо новий продукт, зберігаємо дані в базі:

```
async createProduct(req, res) {
    try {
      const errors = validationResult(req);
      if (!errors.isEmpty()) {
        return res
          .status(400)
          .json({ message: "Error with creating product", errors });
      }

      const { title, description, type, price, shop } = req.body;

      const existingProduct = await Product.findOne({ title: title });

      if (existingProduct) {
        return res.status(422).json({ message: "This product exists already" });
      }

      const existingShop = await Shop.findOne({ url: shop });
```

```
      if (!existingShop) {
        return res
          .status(404)
          .json({ message: "Could not find shop for provided id" });
      }

      let imagePath;
      if (req.file) {
        imagePath = req.file.path;
      } else {
        imagePath = null;
      }

      const newProduct = new Product({
        title,
        description,
        type,
        price,
        image: imagePath,
        shop,
      });

      await newProduct.save();
      await Shop.findOneAndUpdate(
        { url: shop },
        {
          $push: { products: newProduct.id },
        }
      );

      res.status(201).json(newProduct);
    } catch (err) {
      res.status(500).json({ message: err.message });
    }
  }
```

Лістинг.1.6. ProductsController – метод для створення даних

Метод для оновлення даних працює схожим чином як і для створення. Різниця в тому, що ми вже маємо точний заклад харчування та існуючі дані. Але все ж спочатку треба перевірити по ідентифікатору чи такий продукт існує в БД, далі встановлюємо нові дані. Щодо зображення – якщо воно було в базі, ми старе видаляємо, нове зберігаємо:

```
async updateProduct(req, res) {
    try {
      const errors = validationResult(req);
      if (!errors.isEmpty()) {
```

```
        return res
          .status(400)
          .json({ message: "Error with creating product", errors });
      }

      const { title, description, type, price } = req.body;
      const productId = req.params.productId;

      const product = await Product.findById(productId);

      if (!product) {
        return res.status(404).json({
          message: "Something went wrong, could not find a product",
        });
      }

      product.title = title;
      product.description = description;
      product.type = type;
      product.price = price;

      if (req.file) {
        if (product.image) {
          fs.unlinkSync(product.image);
        }
        product.image = req.file.path;
      }

      await product.save();

      res.status(200).json(product);
    } catch (err) {
      res.status(404).json({ message: err.message });
    }
  }
```

Лістинг.1.7. ProductsController – метод для оновлення даних

Останній типовий метод, який слід розглянути це метод запиту видалення. Отримуємо ідентифікатор, шукаємо наявний продукт. Видаляємо його з БД за ідентифікатором, видаляємо картинку з сервера та оновлюємо в базі документ закладу харчування, а саме його поле продуктів, звідти видаляємо ідентифікатор видаленого продукту:

```
async deleteProduct(req, res) {
    try {
      const productId = req.params.productId;
```

```
    const product = await Product.findById(productId);

    if (!product) {
      return res.status(404).json({
        message: "Something went wrong, could not find a product",
      });
    }

    await Product.findByIdAndDelete(productId);

    if (product.image) {
      fs.unlinkSync(product.image);
    }

    await Shop.findOneAndUpdate(
      { url: product.shop },
      {
        $pull: { products: productId },
      }
    );
    res.status(200).json({ message: "Product deleted successfully" });
  } catch (err) {
    res.status(404).json({ message: err.message });
  }
}
```

Лістинг.1.8. ProductsController – метод для видалення даних

Ще варто розглянути middleware, які використовуються в проекті. Для обмеження доступу до використання певних функцій у контролері – будемо користуватись AuthMiddleware. Виконуємо перевірку автентифікації за допомогою токена JWT [12].

В даному коді ми отримуємо токен з хедера, декодуємо його за допомогою секретного ключа, якщо все добре – токен пройшов верифікацію, викликаємо функцію next(), тим самим даємо доступ до виконання наступного коду:

```
import Jwt from "jsonwebtoken";

export const authMiddleware = (req, res, next) => {
  if (req.method === "OPTIONS") {
    next();
  }

  try {
    const token = req.headers.authorization.split(" ")[1];
    if (!token) {
```

```
      return res.status(403).json({ message: "User is not authorized" });
    }
    const decodedData = Jwt.verify(token, process.env.JWT_KEY);
    req.user = decodedData;
    next();
  } catch (err) {
    return res.status(403).json({ message: "User is not authorized" });
  }
};
```

Лістинг.1.9. AuthMiddleware, перевірка автентифікації

Також в проекті ми можемо завантажувати зображення, це дозволяє нам робити пакет multer, він використовується для обробки завантажений файлів на сервер [13]. Розглянемо UploadMiddleware, встановлюємо об’єкт сховища, визначаємо його параметри – шлях зберігання та унікальне ім’я файлу. Також встановлюємо функцію, яка визначає, які типи можуть файлів можуть бути завантажені, встановлюємо бажані розширення. В кінці експортуємо middleware, яка використовує налаштоване сховище та функцію-фільтр:

```
import multer from "multer";

const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, "assets/images");
  },
  filename: function (req, file, cb) {
    cb(null, Date.now() + "-" + file.originalname);
  },
});

const fileFilter = (req, file, cb) => {
  if (file.mimetype === "image/jpeg" || file.mimetype === "image/png") {
    cb(null, true);
  } else {
    cb(new Error("Дозволені тільки файли з розширенням JPEG або PNG"), false);
  }
};

export const uploadMiddleware = multer({ storage, fileFilter });
```

Лістинг.1.10. UploadMiddleware, обробка завантаження зображень

Тепер перевіримо та протестуємо створений функціонал. В цьому нам допоможе Postman – інструмент для розробки та тестування API [14]. Надає зручний функціонал для надсилання HTTP запитів до сервера.

Створимо новий заклад харчування, оскільки доведеться завантажувати зображення, то скористуємось form-data:

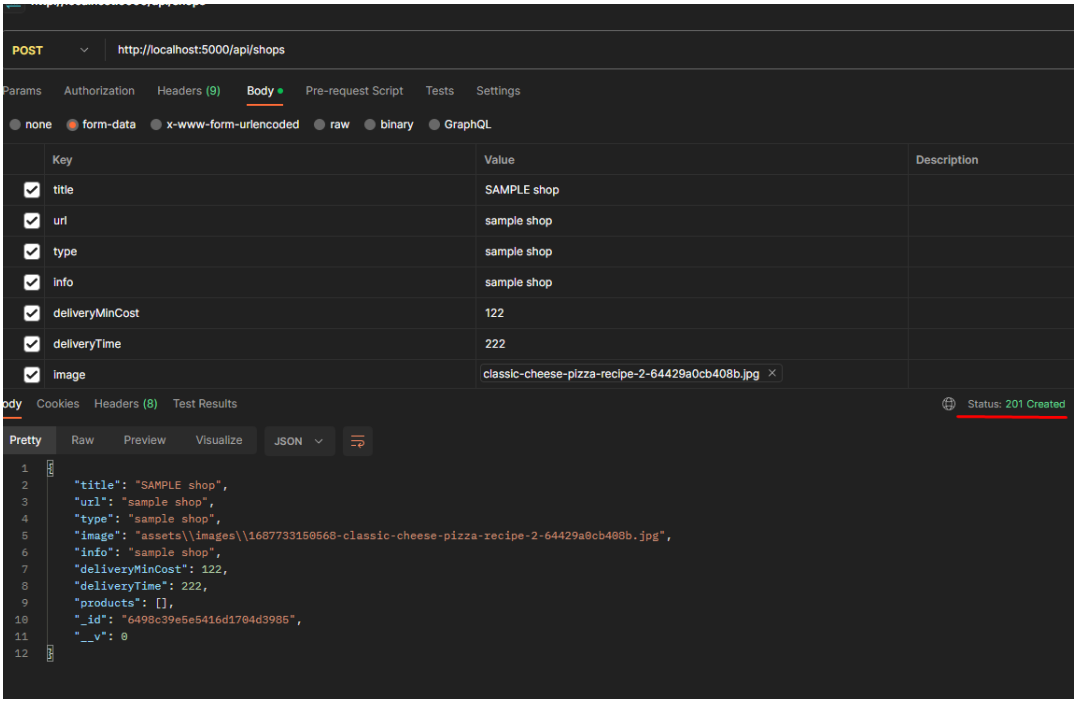

Рис.2.8. Успішне створення нового закладу харчування

До цього ми в хедері вставили токен, тому у нас був доступ до створення:

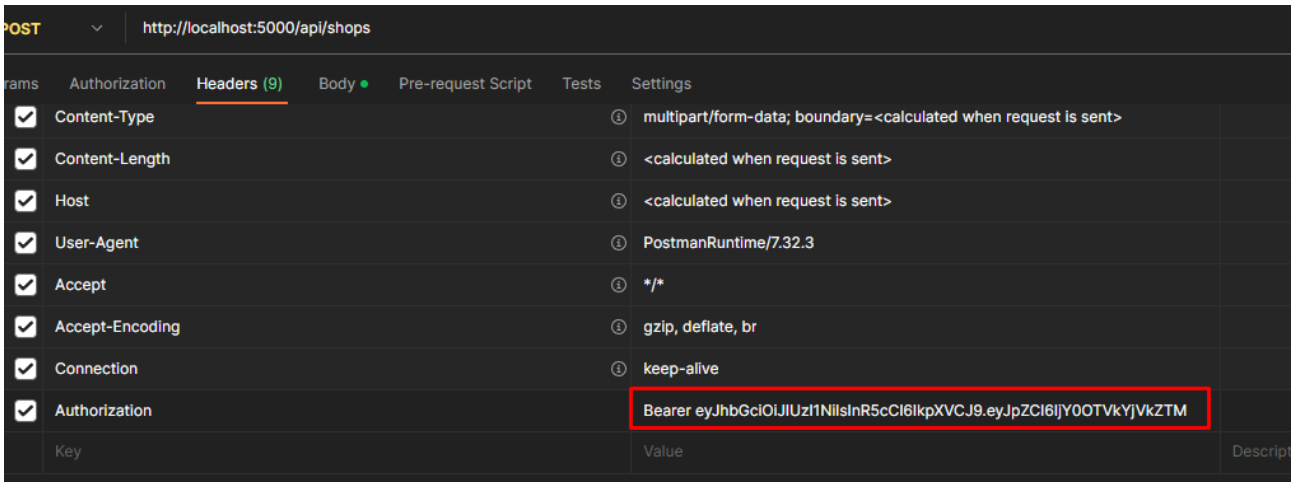

Рис.2.9. Хедер з дійсним токеном автентифікації

Якщо ж його прибрати або вказати не дійсний, то маємо отримати повідомлення, що користувач не авторизований:

Рис.2.10. Повідомлення про відмову в доступі

Також перевіримо валідацію даних, спробуємо не ввести декілька полів:

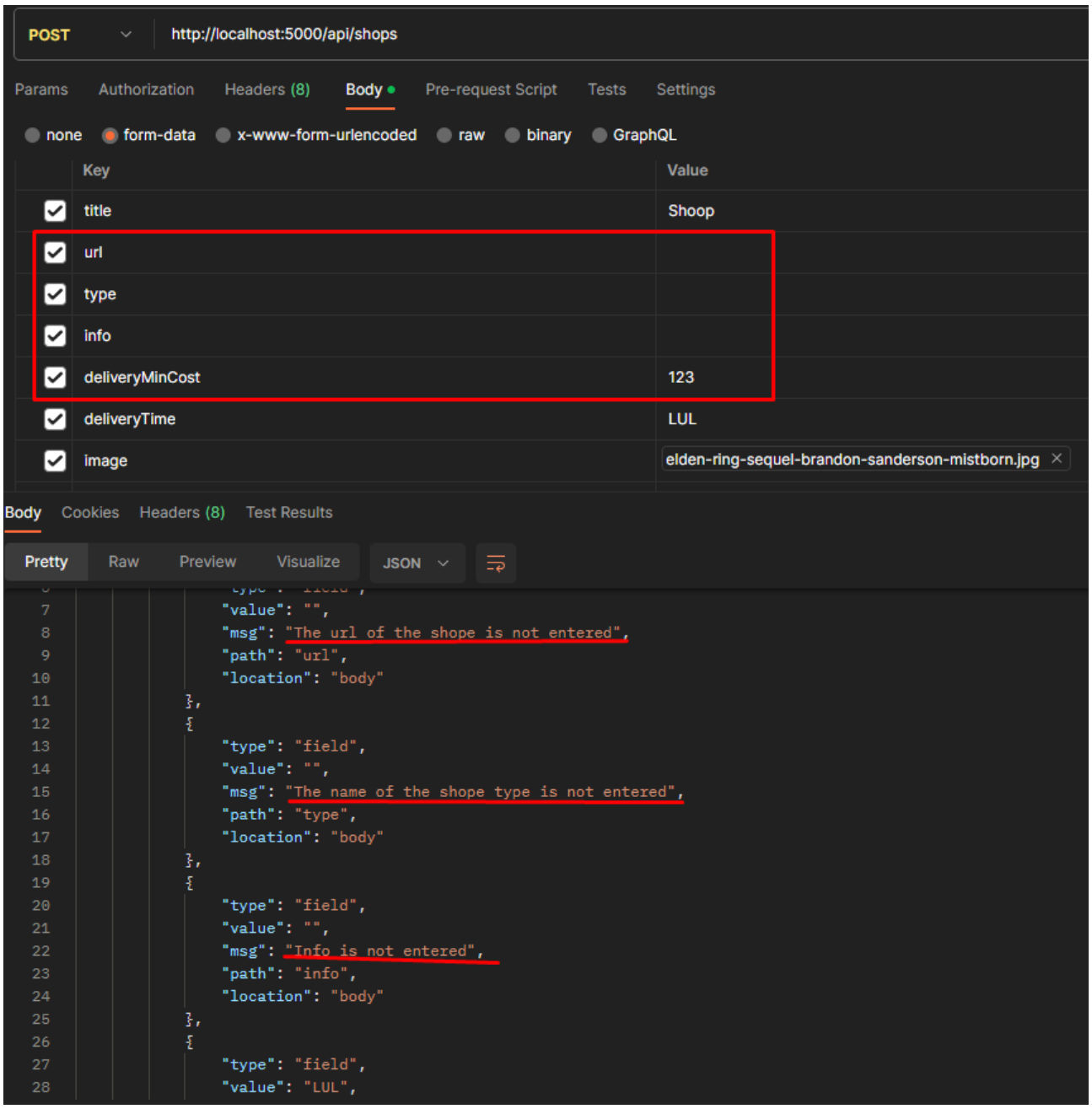

Рис.2.11. Повідомлення про помилку, не валідні вхідні дані

У закладу маємо унікальний url, спробуємо ввести вже існуючий:

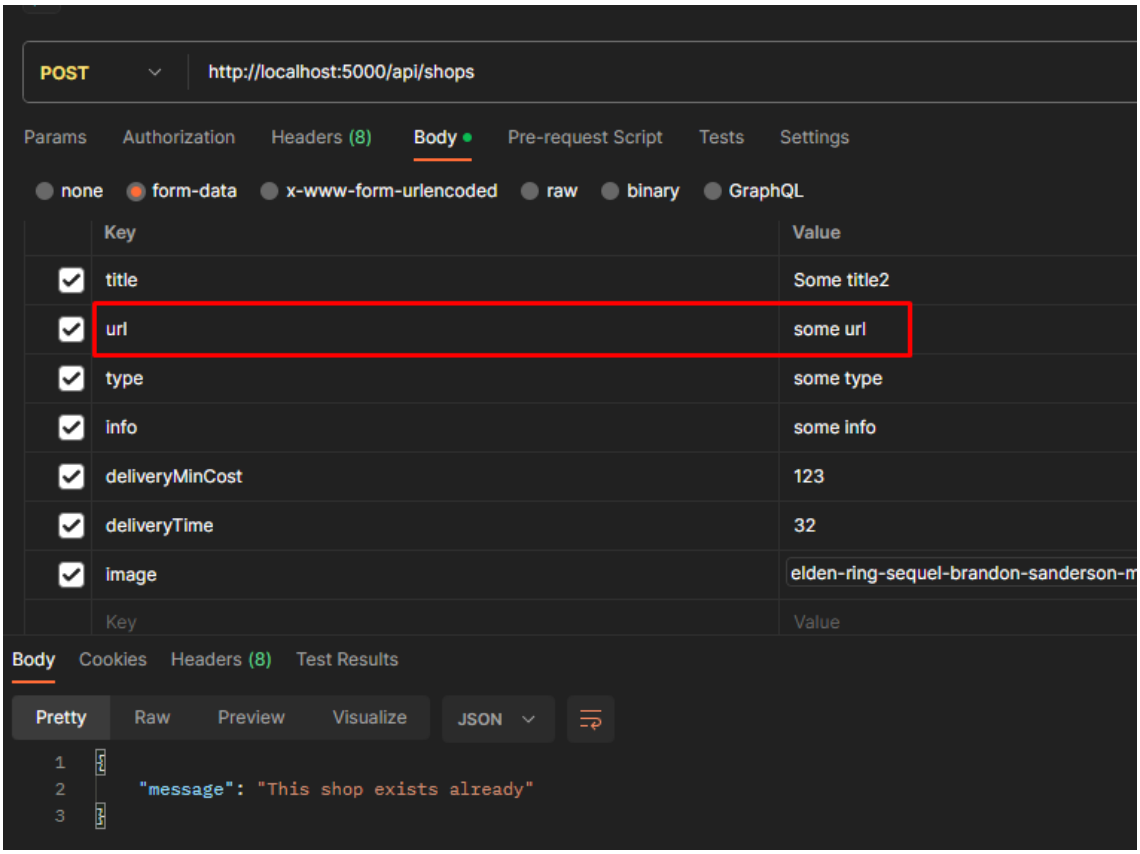

Рис.2.12. Повідомлення про помилку, що заклад з відповідним полем вже існує

Перевіримо метод для отримання даних на прикладі продуктів:

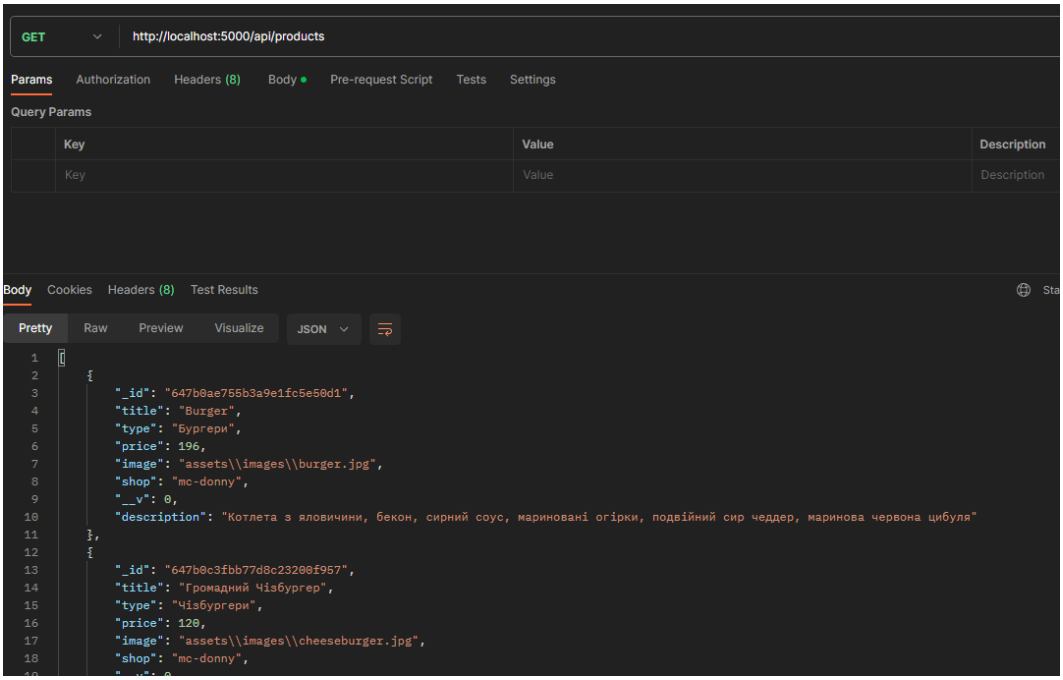

Рис.2.13. Отримання даних про всі продукти

Редагування даних відбувається за допомогою метода PATCH, оновимо дані про продукт:

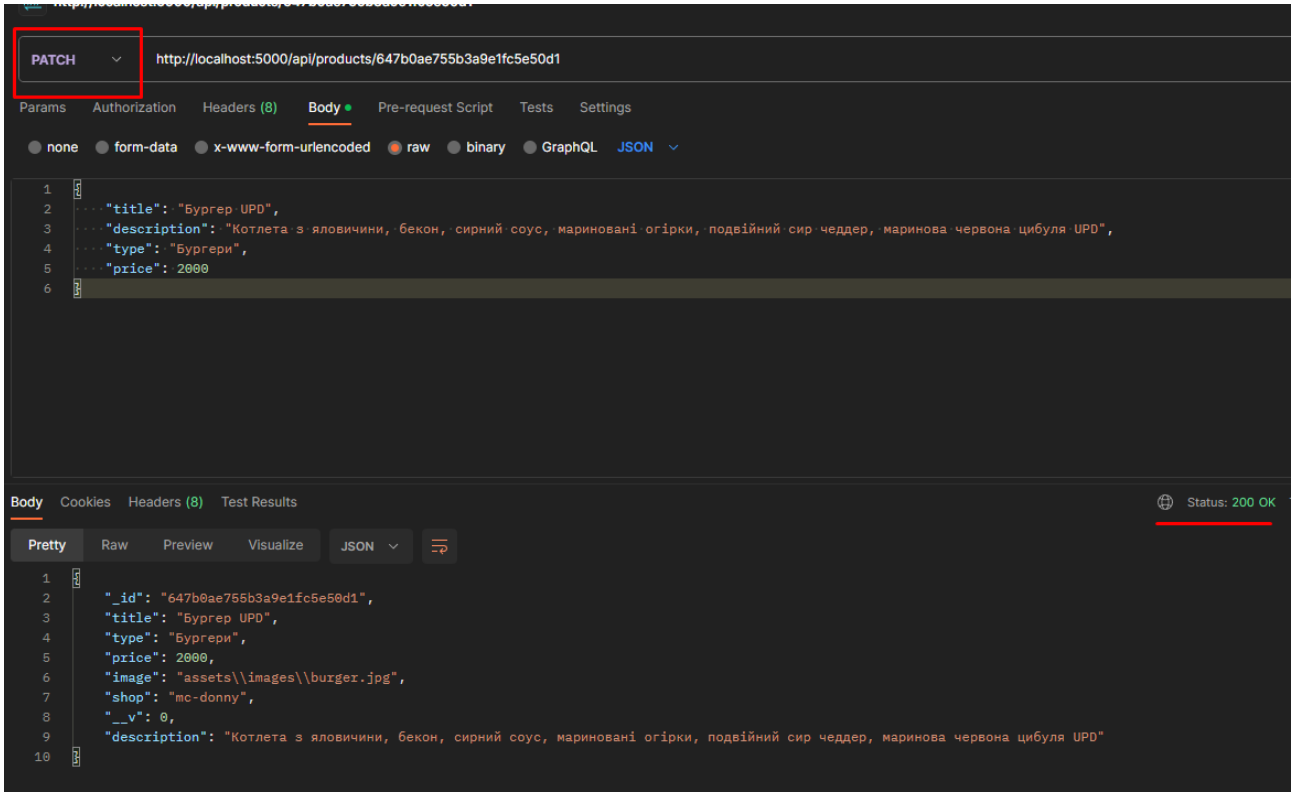

Рис.2.14. Оновлення даних про продукт за ідентифікатором

Видалення даних також відбувається за ідентифікатором, маємо отримати повідомлення про успіше видалення продукту:

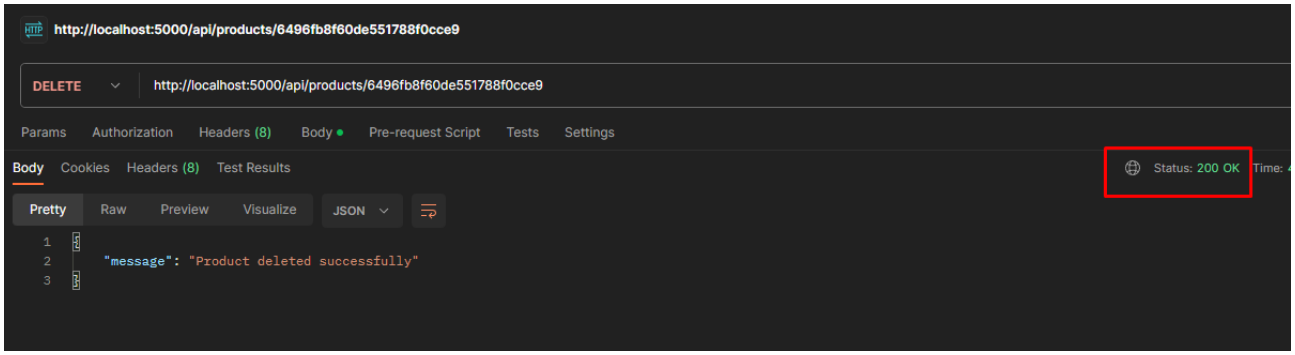

Рис.2.15. Видалення даних про продукт за ідентифікатором

Авторизація відбувається таким чином, вводимо коректні дані, отрмуємо токен та роль:

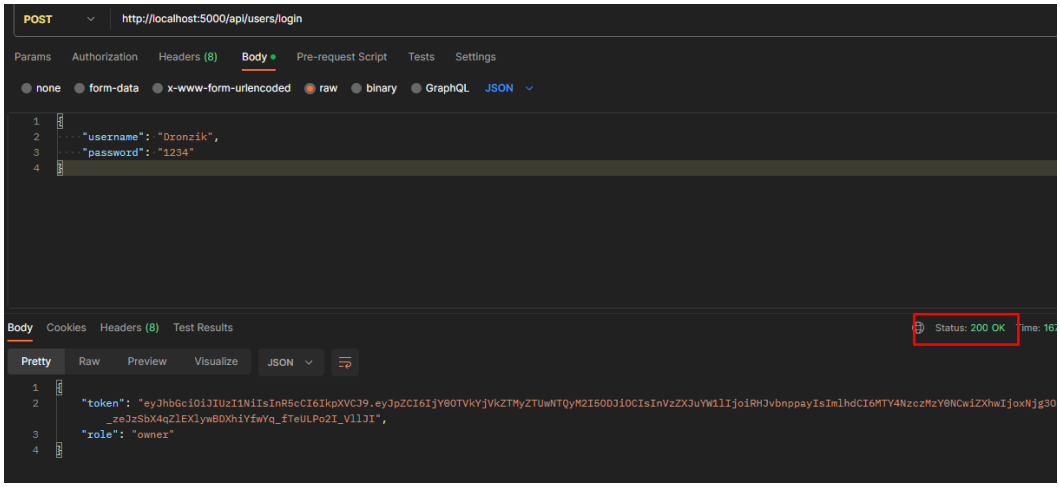

Рис.2.16. Успішна авторизація

Введемо не правильний пароль:

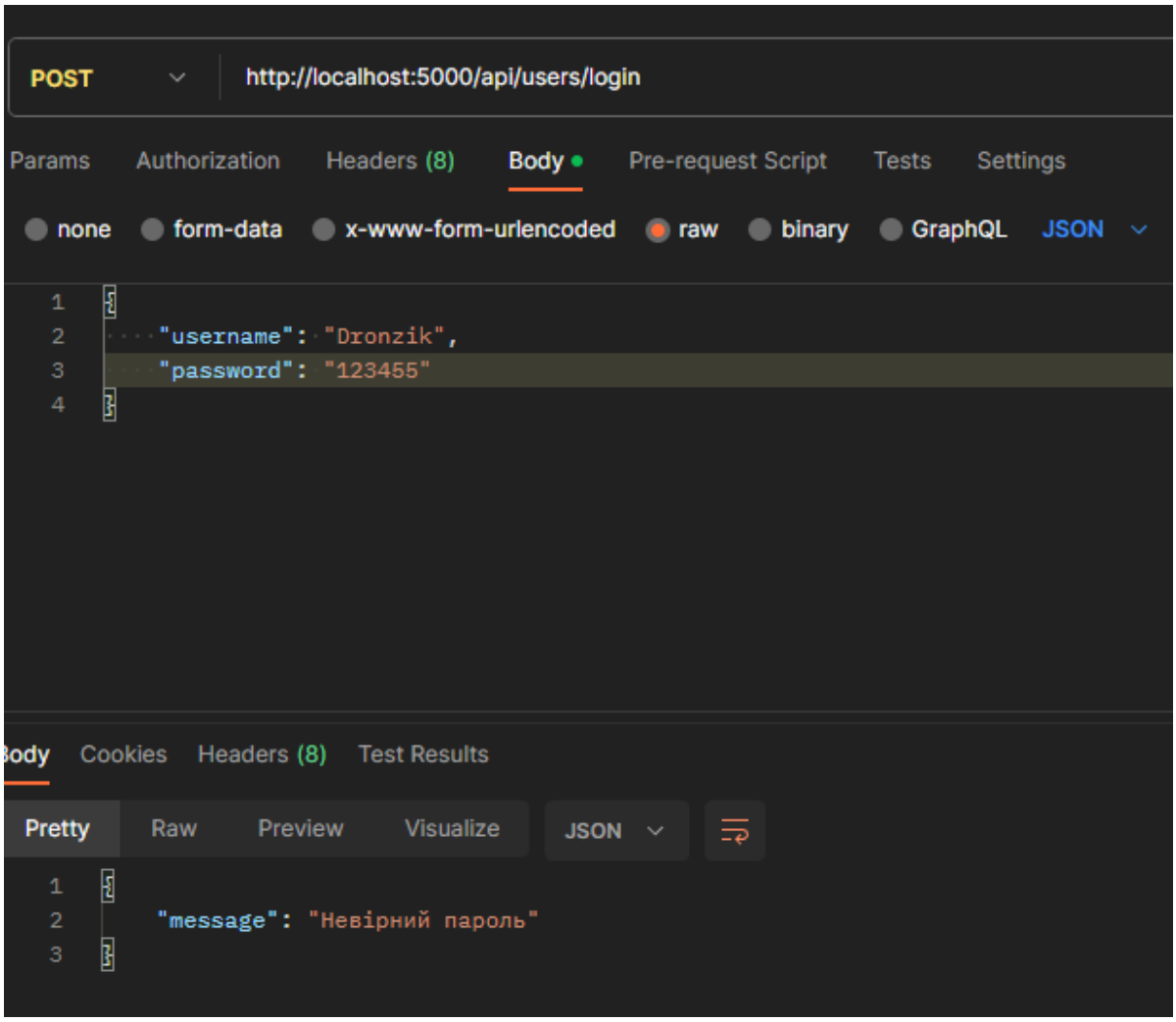

Рис.2.17. Перевірка авторизації, помилка про невірний пароль

Висновки до розділу 2

В другому розділі кваліфікаційної роботи було спроектовано функціонал системи. Були використанні такі UML діаграми як: діаграми варіантів використання, де були показані вимоги для різних ролей, це було графічно проілюстровано, було розроблено загальну діаграму діяльності системи, де було показаний основний алгоритм роботи веб-додатку.

Була розглянута база даних, створена діаграми БД, описано її поля, для чого вони використовуються.

Після проектування була розроблена серверна частина веб-додатку з використанням фреймоворку Express.js та платформою Node.js. Розроблений функціонал був протестований за допомогою інструмента Postman. У результаті отримали працездатний бекенд.

РОЗДІЛ 3. ПОБУДОВА ІНТЕРФЕЙСУ, ПОРЯДОК РОБОТИ З ВЕБ-ДОДАТКОМ

3.1 Користувацький інтерфейс, методи його побудови

Важливо розробити швидкий та зручний для користувача інтерфейс системи, який буде правильно працювати з даними, отримувати їх, додавати та видаляти. Для клієнта основі вимоги були виставлені, серверна частина готова, тож залишилось зробити фронтенд. Ми використовуємо бібліотеку React, де функціональні компоненти це основний принцип побудови застосунку.

Можемо виділити переваги даного підходу:

- простий та зрозумілий код, використовуємо просту функцію JS замість класа для створення функціонального компонента;

- повторне використання, завжди добре коли код не дублюється;

- продуктивність, функціональні компоненти мають хорошу та ефективну продуктивність роботи;

- багато npm-пакетів використовують цей підхід, можемо з легкістю інтегрувати їх.

React додаток побудований з використанням принципу SPA (Single Page Application) - це підхід, де вся сторінка завантажується одного разу при запуску додатка, а подальша навігація та взаємодія з користувачем відбуваються без повного перезавантаження сторінки. Замість цього, відбувається динамічна зміна контенту на сторінці за допомогою JavaScript, перерендеринг.

Щоб мати навігацію на сайті та дотриматись принципу SPA будемо використовувати бібліотеку React Router, яка дозволяє використовувати та будувати маршрутизацію, навігацію між сторінками на фронтенді [16].

Також слід розглянути важливу тему пропс дрілінгу (props drilling) та методи вирішення цієї проблеми. Будуючи реакт застосунок ми стикаємось з великою кількістю компонентів, більшість з яких потребує параметри для виконання – пропси. Таким чином складається дерево компонентів, зазвичай пропси можуть передавитись з верхнього рівня до нижнього, проходячи купу

інших компонентів, при цьому ніяк з ними не взаємодіячи. Це свідчить про погану організацію компонентної структури проекту. Пропс дрілінг може призвести до збільшення складності коду, погіршення читабельності та зайвих дій при зміні структури компонентів, які можна було б уникнути.

Для вирішення цієї проблеми, можемо виділити два методи, які можуть вдало використовуватись у веб-додатку для замовлення їжі, це:

1. Контекст (Context), вбудована функція у реакті. Це механізм, що дозволяє передавати дані через дерево компонентів, без необхідності тягнути пропси через всі проміжні етапи (компоненти). Створюється контекст, який може містити стейти та функції. Далі треба огорнути кореневий компонент, зазвичай це App.jsx в Context.Provider, це дасть доступ до контексту всім дочірнім компонентам;

2. Стейт менеджери, це бібліотеки які можуть встановити глобальний стейт у додатку, допомагає їм управляти, змінювати дані. Завдяки стейт менеджеру, ми може отримувати дані в будь якому компоненті безпосередньо з централізованого стейту, уникнувши передачу пропсів через проміжні компоненти. Механізм схожий на контекст, різниця в тому, що зазвичай надбудови над реактом працюють швидше і дають більш потужні інструменти для роботи з глобальним стейтом.

Отже, підемо шляхом стейт менеджерів, це більш швидко та дані технології зазвичай і використовують при побудові якісних реакт застосунків. Наразі існує чимало подібних надбудов, але розглянемо дві самі актуальні на теперішній момент, які зараз на слуху, це Redux та Zustand.

Redux – потужна бібліотека для керування станом, дуже популярна [17]. Можемо пересвідчитись в цьому, глянувши кількість завантажень [18]. Використовує концепцію однозначного потоку, потребує налаштувань таких елементів, як екшненів (дій, actions), редьюсерів (reducers) та стор (store). Початкове налаштування редаксу досить складне, але дозволяє отримувати більшу гнучкість для контролю глобального стейту.

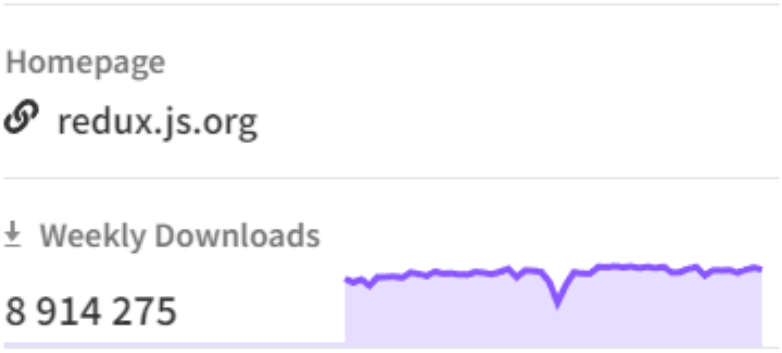

Рис.3.1. Кількість завантажень бібліотеки Redux

Zustand – чудова альтернатива редаксу, незважачи, що це відносно нова бібліотека, вона дуже швидко та стрімко набирає свою популярність, це також можемо побачити на графіку завантажень [19]. Zustand має досить простий синтаксис, немає складних екшенів та редьюсерів, змінювати стейт можна за допомогою сетерів (setters). Синтаксис цієї бібліотеки декларативний, можемо оголошувати стан безпосередньо в компонентах.

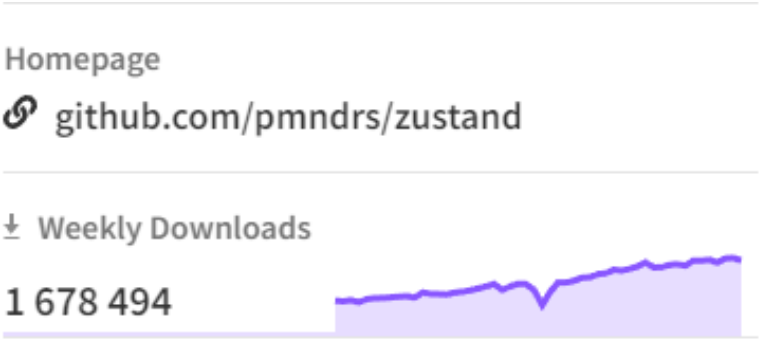

Рис.3.2. Кількість завантажень бібліотеки Zustand

Зважаючи на потреби проекту, вибір бібліотеки Zustand буде більш доцільним. Бібліотека має простий синтаксис, а оскільки проект не є досить об’ємним, то швидкодія буде на найвищому рівні та ми будемо мати простий спосіб керування глабальним стейтом.

Розглянемо використання даної бібліотеки на прикладі кошика товарів. В Store.js – оголосимо стейт продуктів кошика та методи для додавання до кошика товарів, зменшення кількості та збільшення кількості продуктів, очищення кошика товарів:

```
import { create } from "zustand";

export const useCartStore = create((set) => ({
  cartProducts: [],
  addToCart: (
    productId,
    productTitle,
    productPrice,
    productImage,
    productQuantity,
    productShop
  ) =>
    set((state) => {
      const existingProduct = state.cartProducts.find(
        (product) => product.id === productId
      );

      if (existingProduct) {
        return {
          cartProducts: state.cartProducts.map((product) => {
            if (product.id === productId) {
              return {
                ...product,
                quantity: product.quantity + productQuantity,
              };
            }
            return product;
          }),
        };
      } else {
        return {
          cartProducts: [
            ...state.cartProducts,
            {
              id: productId,
              title: productTitle,
              price: productPrice,
              image: productImage,
              quantity: productQuantity || 1,
              shop: productShop || null,
            },
          ],
        };
      }
    }),
  addToProductCounter: (productId) =>
    set((state) => ({
      cartProducts: state.cartProducts.map((item) => {
        if (item.id === productId) {
          return { ...item, quantity: item.quantity + 1 };
        }
```

```
        return item;
      }),
    })),
  subtractFromProductCounter: (productId) =>
    set((state) => ({
      cartProducts: state.cartProducts
        .map((item) => {
          if (item.id === productId) {
            if (item.quantity - 1 <= 0) {
              return null;
            }
            return { ...item, quantity: item.quantity - 1 };
          }
          return item;
        })
        .filter(Boolean),
    })),
  clearCart: () => set(() => ({ cartProducts: [] })),
}));
```

Лістинг.2.1. Store.js, функціонал кошика товарів, методи для роботи зі стейтом

Використання методів стора в компонентах може виглядає наступним чином:

```
const productsInCart = useCartStore((state) => state.cartProducts);
const addToCartProduct = useCartStore((state) => state.addToCart);
const { clearCart } = useCartStore();
const subtractProduct = useCartStore(
  (state) => state.subtractFromProductCounter
);
const addProduct = useCartStore((state) => state.addToProductCounter);
```

Лістинг.2.2. Використання глобального стейту в компонентах

Щодо алгоритму роботи кошика, що було зображено на діаграмі діяльності (Рис.2.4). Це можна зробити також за допомогою глобального стейту. Оголосимо стейт поточного закладу харчування, при додаванні заносимо в нього ідентифікатор магазина:

```
export const useActiveShop = create((set) => ({
  activeShop: null,
  setActiveShop: (shopId) => set((state) => ({ activeShop: shopId })),
}));
```

Лістинг.2.3. Глобальний стейт поточного закладу харчування

Ще з методів для побудови реакт застосунку, слід зазначити інструменти для валідації форм. Хорошим вибором буде поєднанати дві популярні бібліотеки Formik та Yup.

Formik - це бібліотека, яка надає потужні засоби для управління станом форм, обробки введення користувача, відправки даних тощо [20]. Вона спрощує роботу з формами в React, забезпечуючи зручний спосіб зберігання стану форми, обробки подій, валідації та обробки помилок.

Yup - це бібліотека валідації, яка дозволяє визначити правила, схему валідації для полів форми [21]. Вона надає простий спосіб описати правила перевірки вхідних даних, такі як обов'язковість поля, мінімальна або максимальна довжина, формат даних тощо.

Наприклад використання бібліотеки yup, оголошуємо поля форми та правила валідації:

```
const customerSchema = yup.object().shape({
  name: yup.string().required("Потрібно ввести ім'я"),
  email: yup.string().email().required("Введіть валідний імейл"),
  phone: yup.number().required("Введіть номер телефону"),
  address: yup.string().required("Введіть адресу"),
});
```

Лістинг.2.4. Схема правил валідації

Далі схему можемо використовувати в хуці useFormik, який приймає об’єкт налаштувань з параметрами форми:

```
const formik = useFormik({
    initialValues: {
      name: "",
      email: "",
      phone: "",
      address: "",
    },
    validationSchema: customerSchema,
    onSubmit: (values) => {
      return values;
    },
  });

  const { values, errors, touched, handleChange, handleBlur, handleSubmit } =
    formik;
```

Лістинг.2.5. Використання бібліотеки Formik

Для збереження деяких даних після перезавантаження сторінки, можна використовувати локальне сховище бразуера (Locale Storage).

Локальне сховище - це механізм зберігання даних веб-браузера, який дозволяє зберігати дані на боці клієнта. Він забезпечує простий спосіб зберігання та отримання даних без необхідності взаємодії з сервером. Можемо його використовувати для зберігання продуктів кошика або для збереження токена авторизації адміністратора.

Отже, таким чином, використовуючі описані методи та принципи розробки і буде створюватись клієнтська частина веб-додатку.

3.2 Порядок роботи з веб-додактом для замовлення їжі

У даному підрозділі оглянемо розроблений проект та протестуємо різні сценарії порядку роботи з веб-сайтом.

Головна сторінка веб-додатку виглядає наступним чином:

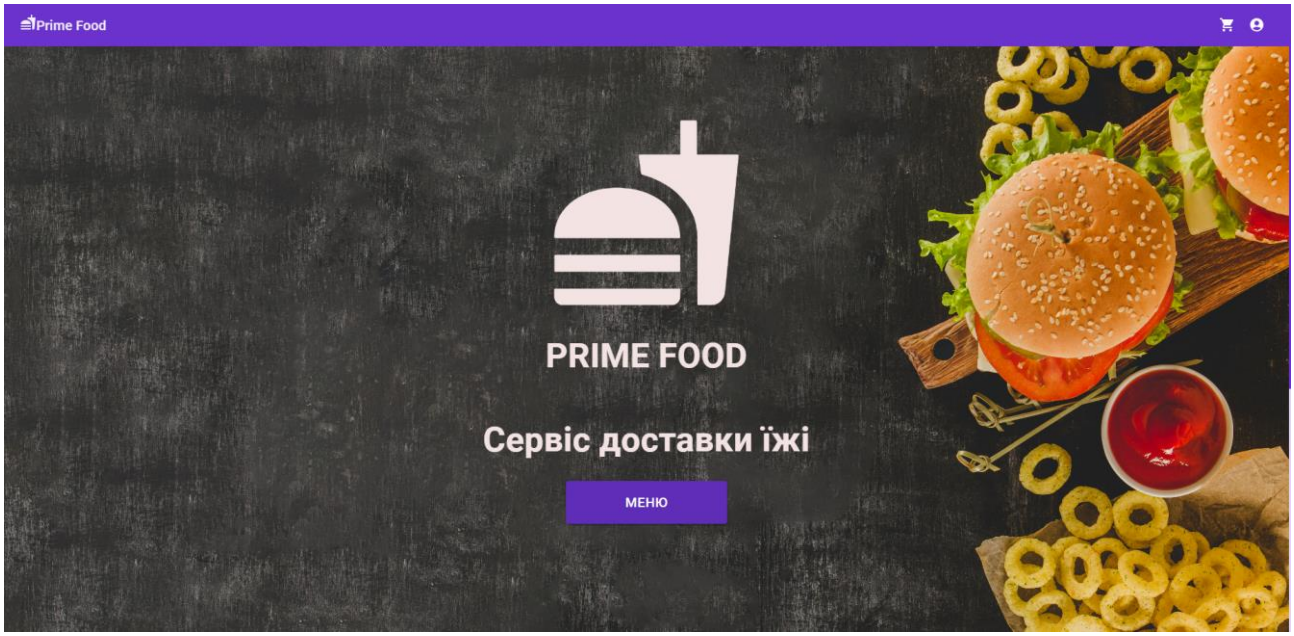

Рис.3.3. Головна сторінка веб-додатку

Бачимо навігаційну панель, кошик, вхід до панелі адміністратора. Одразу видно призначення сайту, перехід в головне меню закладів харчування.

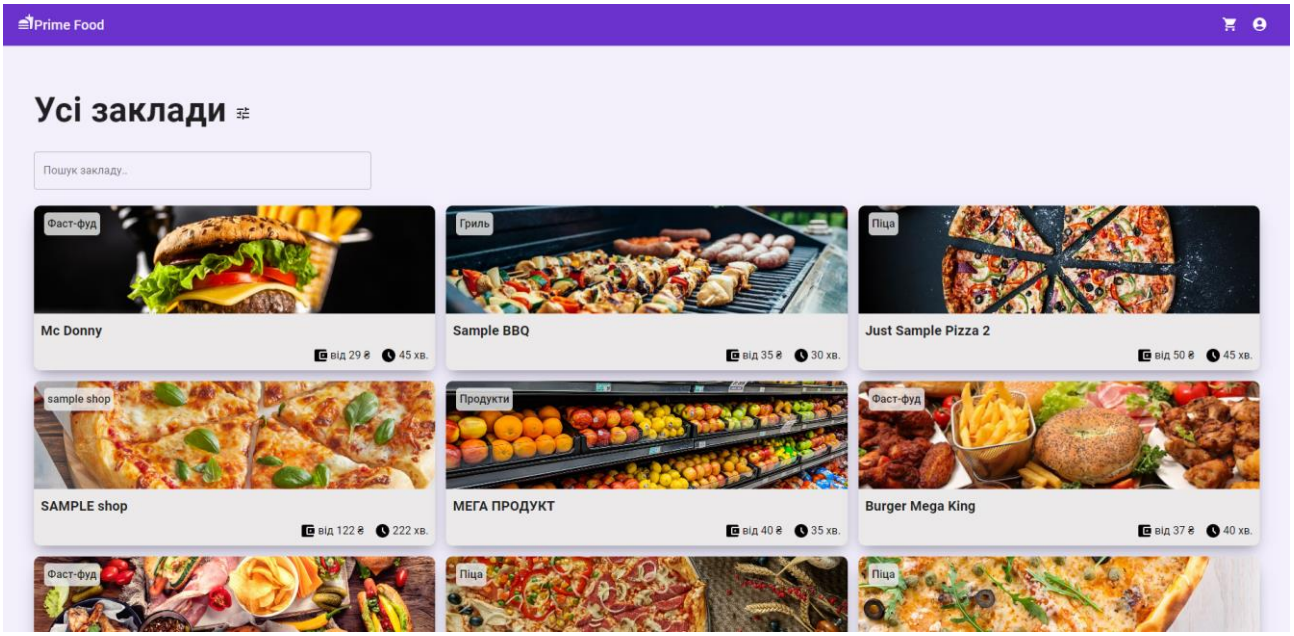

Рис.3.4. Заклади харчування

Присутня адаптивність:

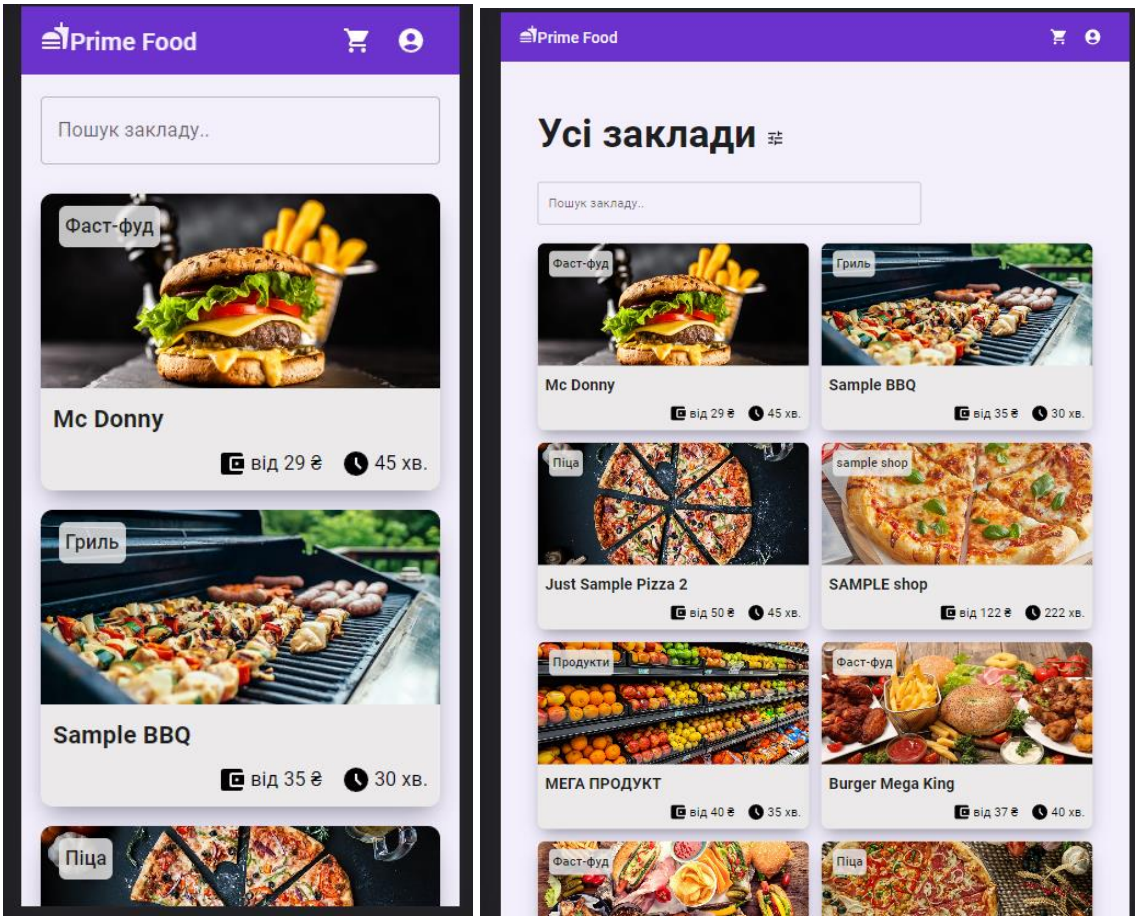

Рис.3.5. Перегляд меню закладів з різною роздільною здатністю

Скористаємось фільтрацією, де відкривається модальне вікно категорій:

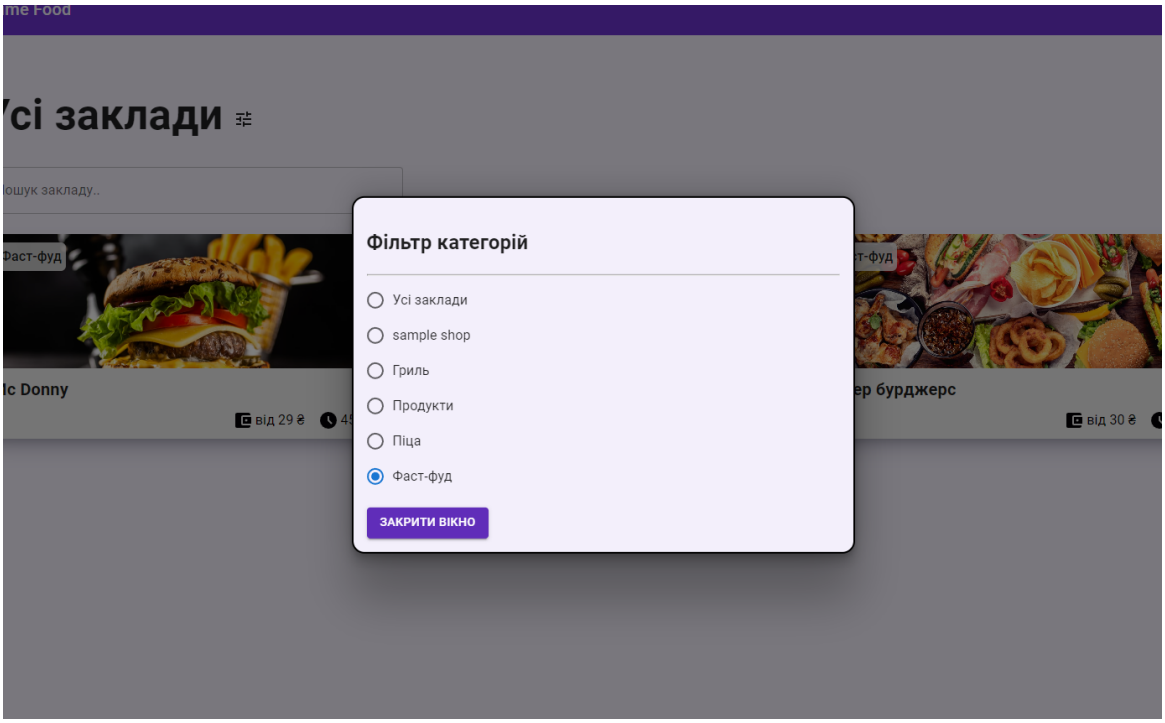

Рис.3.6. Модальне вікно фільтрації

Пошук закладів виглядає наступним чином:

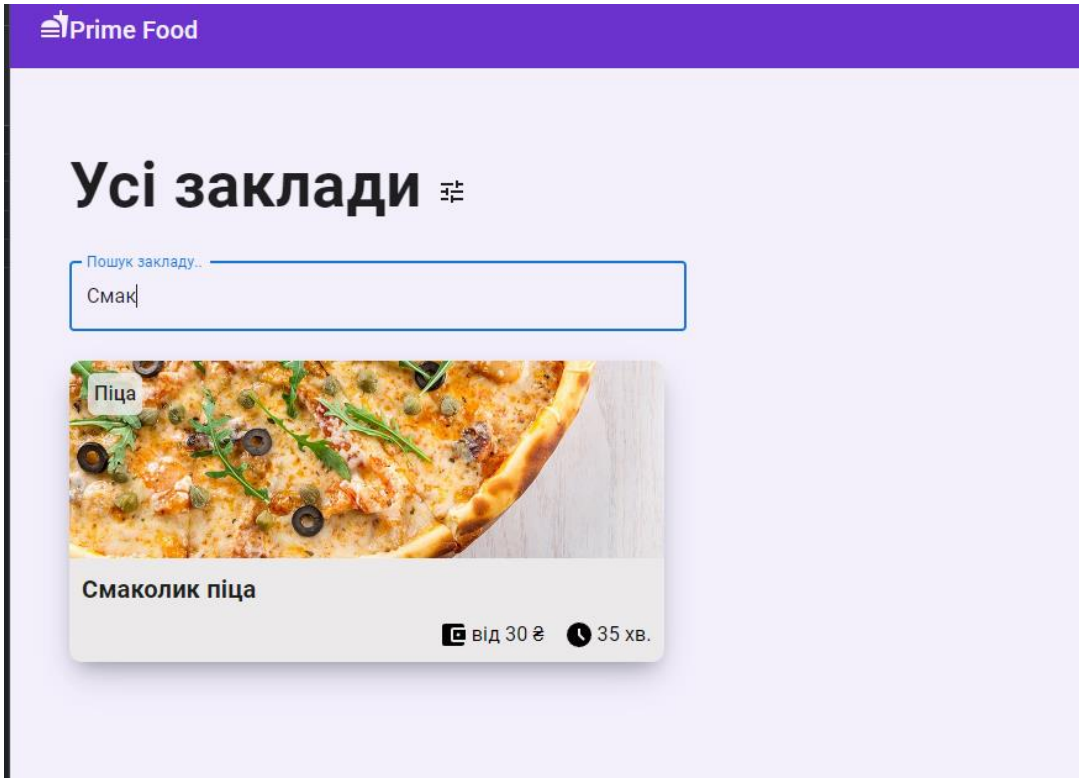

Рис.3.7. Пошук закладів харчування

Перейдемо до певного закладу харчування. Тут бачимо блок інформації про ресторан. Присутня фільтрація за категоріями й сітка товарів:

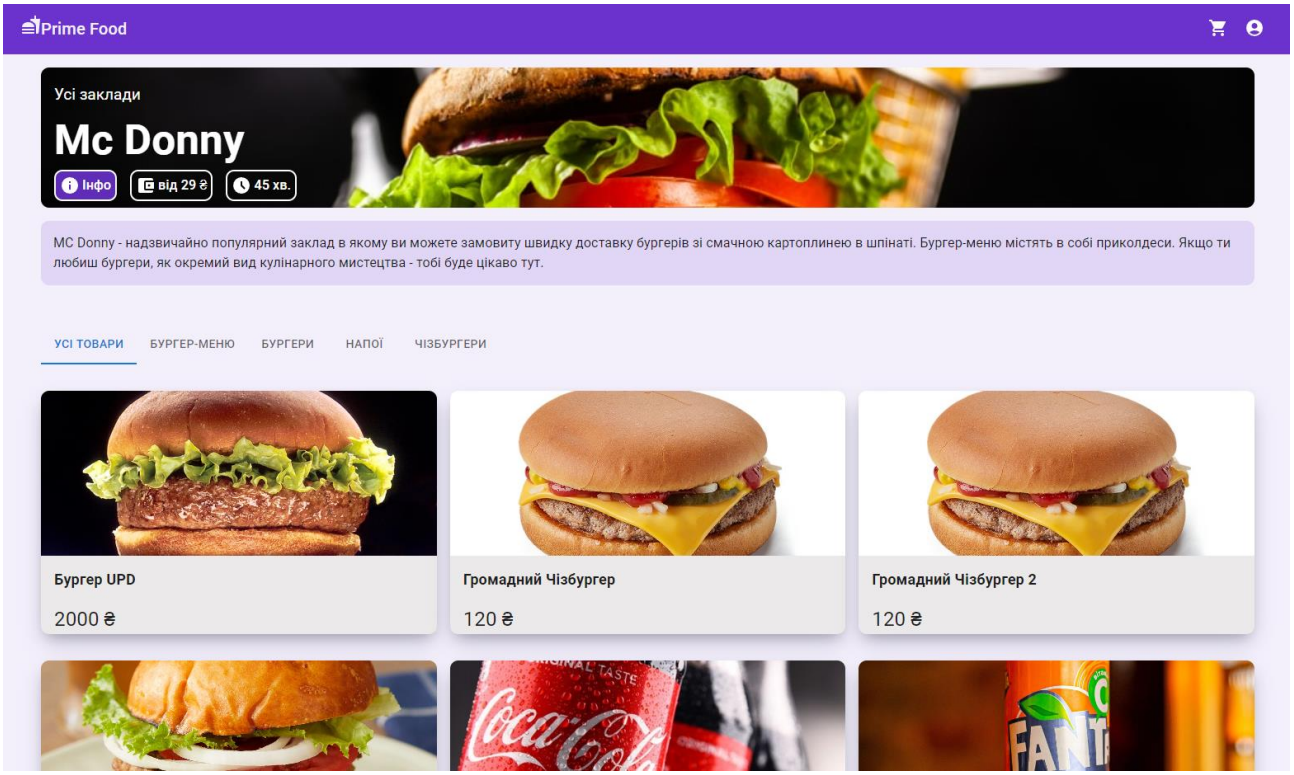

Рис.3.8. Сторінка товарів

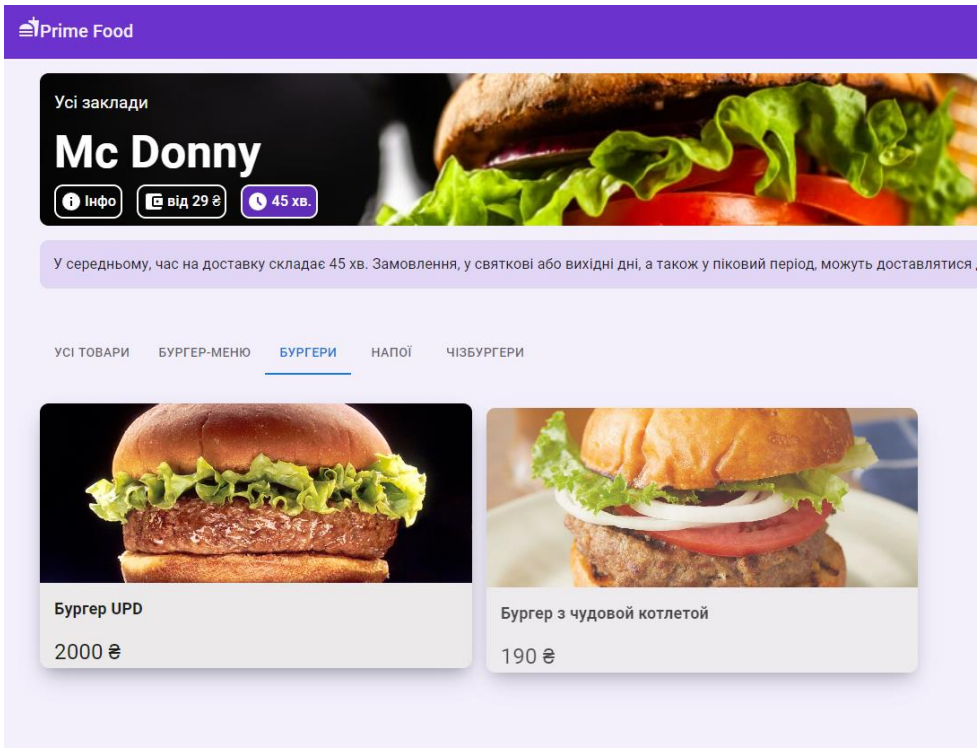

Рис.3.9. Сторінка товарів, використання фільтрації

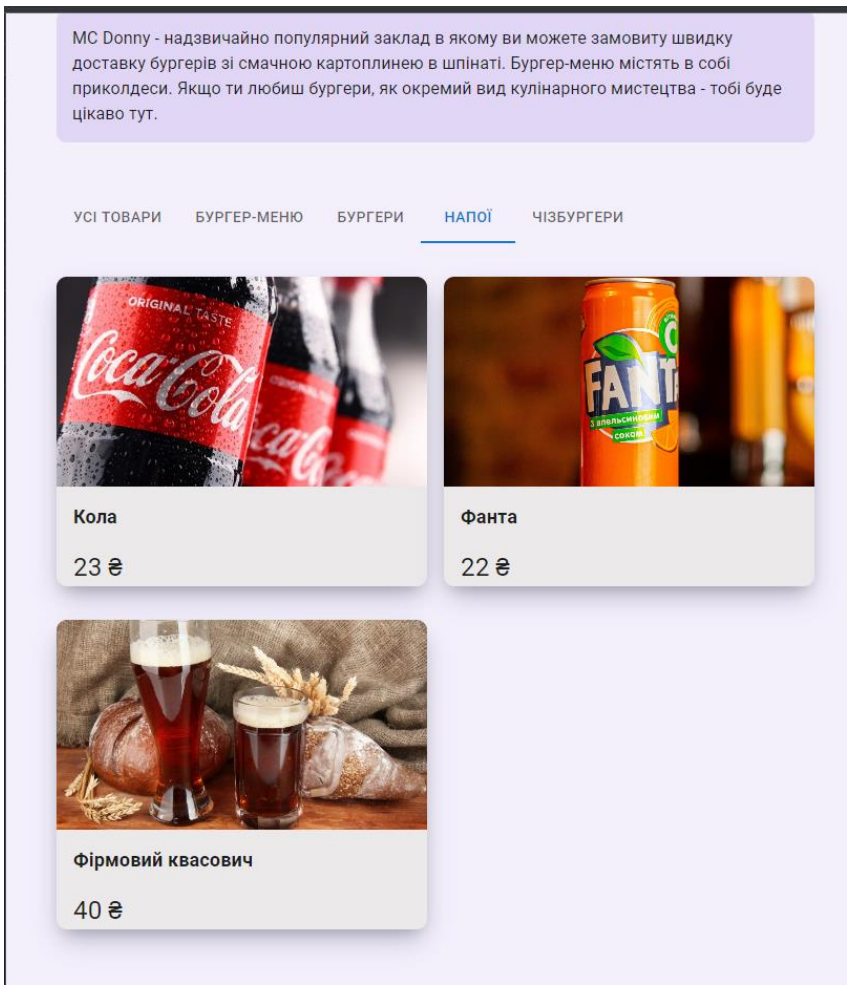

Рис.3.10. Сторінка товарів, використання фільтрації, вигляд з іншого розміру екрану

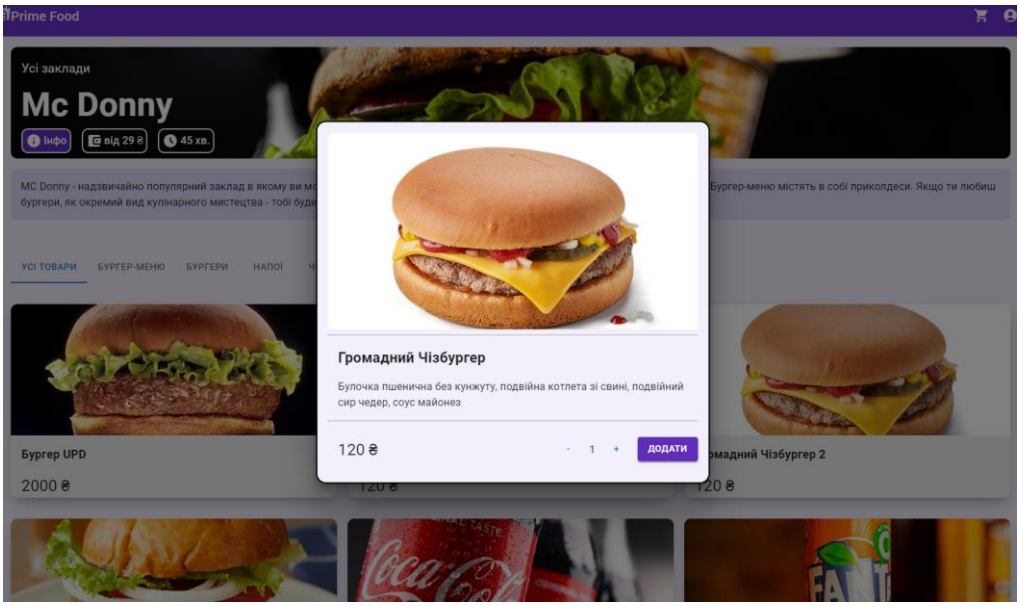

Рис.3.11. Модальне вікно продукту, його опис, можливість додавання до кошику

Додамо до кошика декілька товарів та перейдемо на сторінку замовлення, заповнимо усі поля коректно, у блоці переліку товарів, можемо змінювати кількість продуктів у замовленні або видаляти їх.:

Рис.3.12. Сторінка замовлення

Оформимо замовлення, отримуємо відповідне підтвердження:

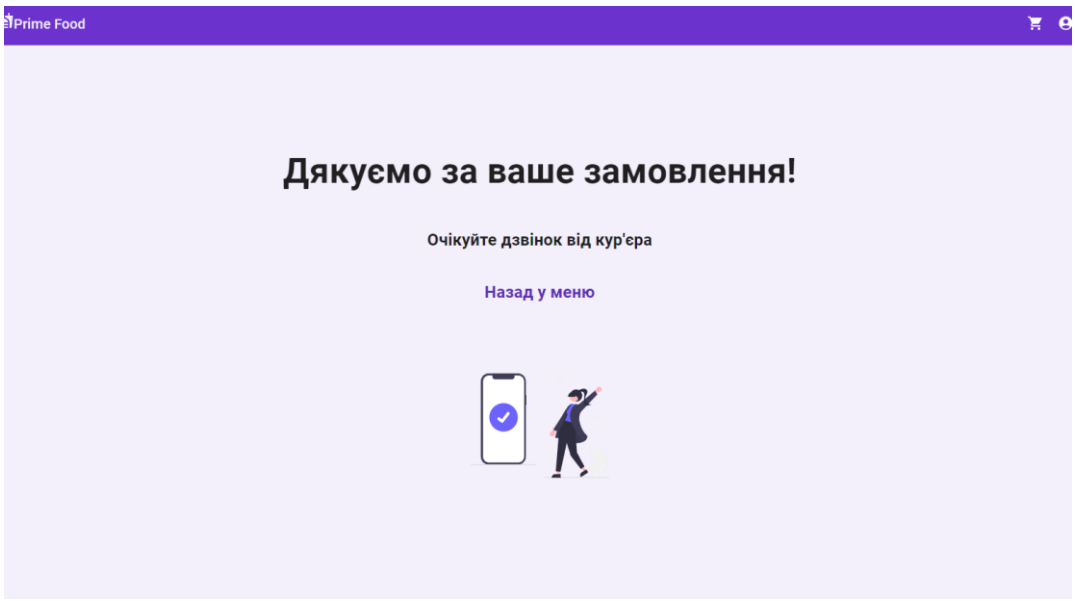

Рис.3.13. Успішне замовлення

Якщо перейти у порожній кошик, то отримаємо:

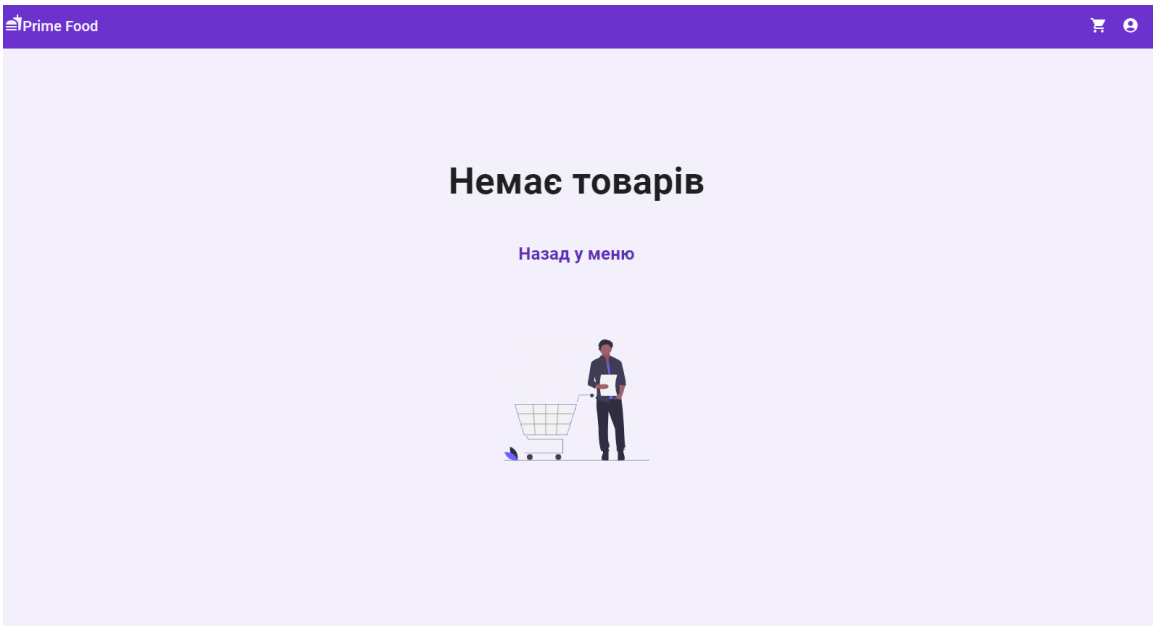

Рис.3.14. Порожній кошик

Якщо ж у кошику будуть товари і перейдемо у меню закадів, то побачимо відповідне повідомлення:

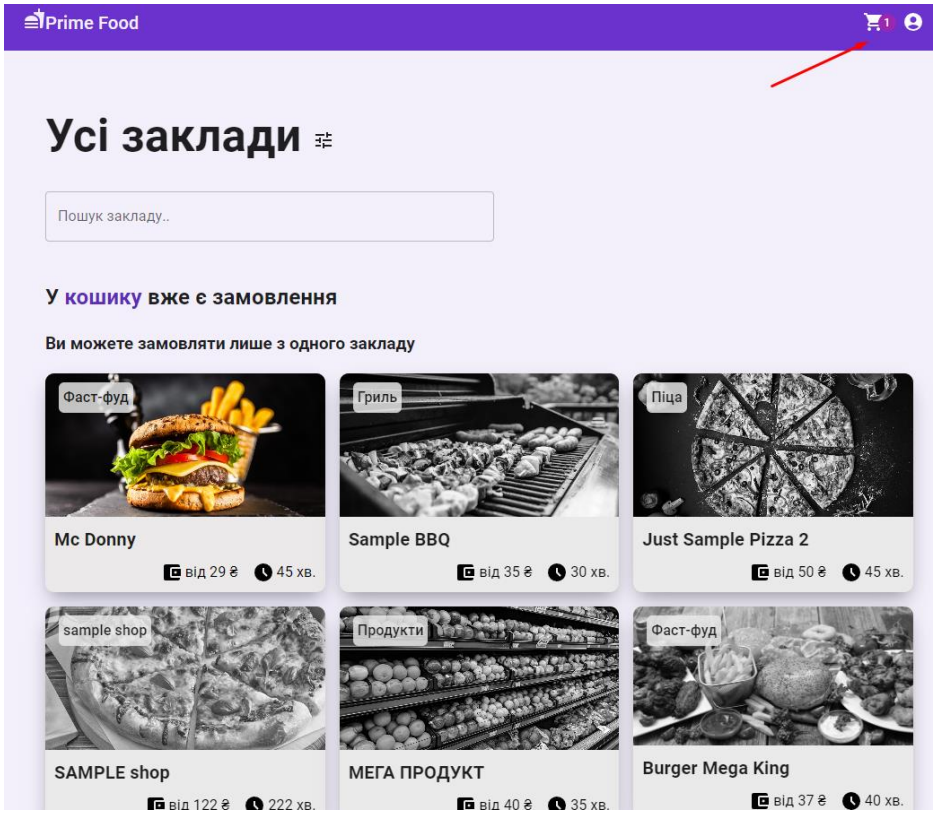

Рис.3.15. Меню закладів, коли в кошику вже присутні товари

Перейдемо до панелі адміністратора, треба залогінитись. Перевіримо декілька сценаріїв:

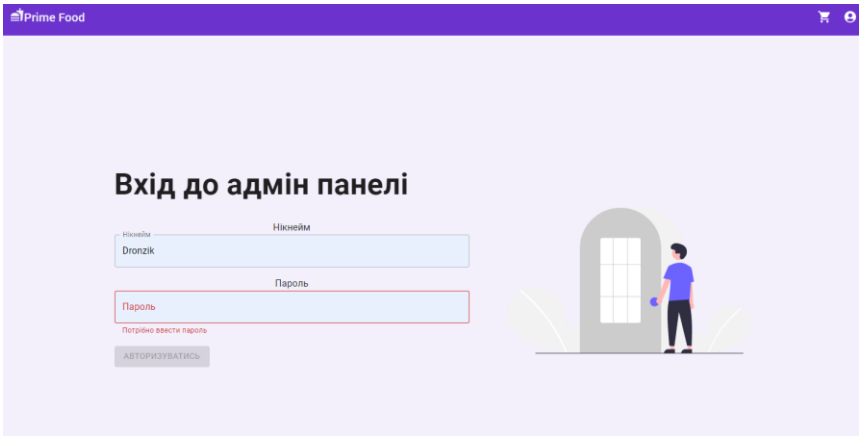

Рис.3.16. Спроба авторизації, невірний введений пароль на клієнті

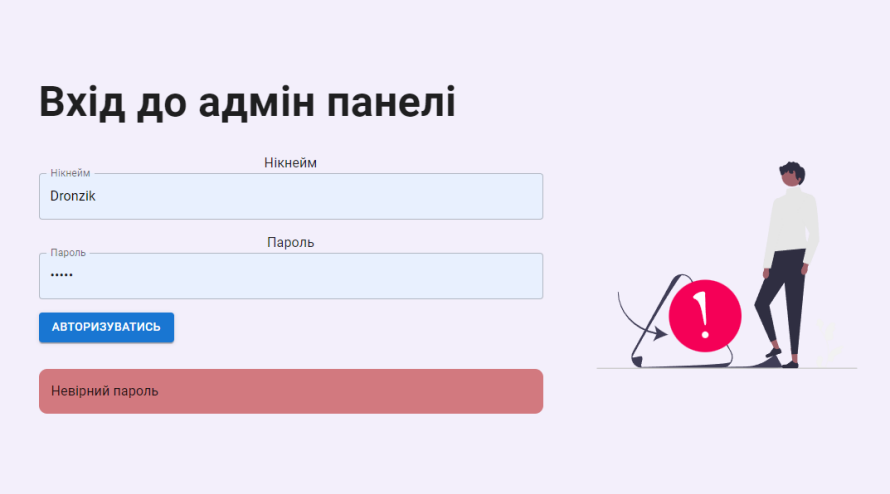

Рис.3.17. Спроба авторизації, невірний введений пароль, помилка з сервера

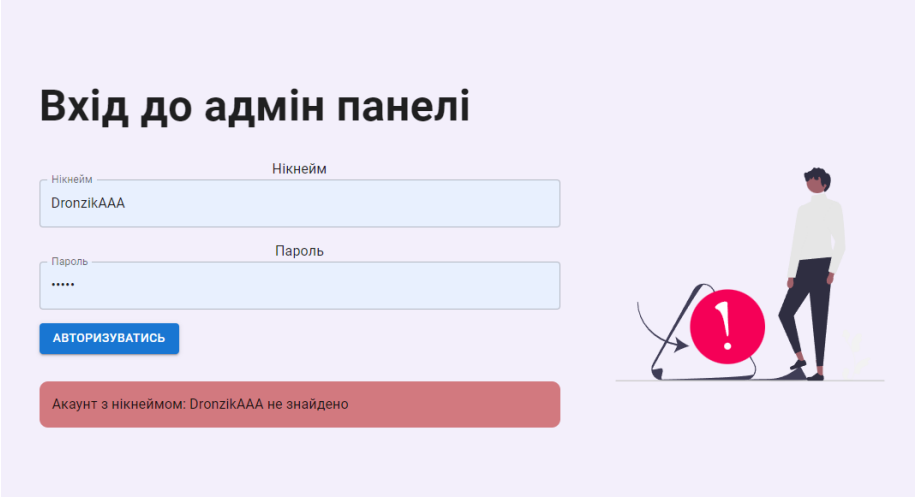

Рис.3.18. Спроба авторизації, не знайдено акаунт

Якщо ж введемо коректно дані, то отримаємо доступ до панелі адміністратора:

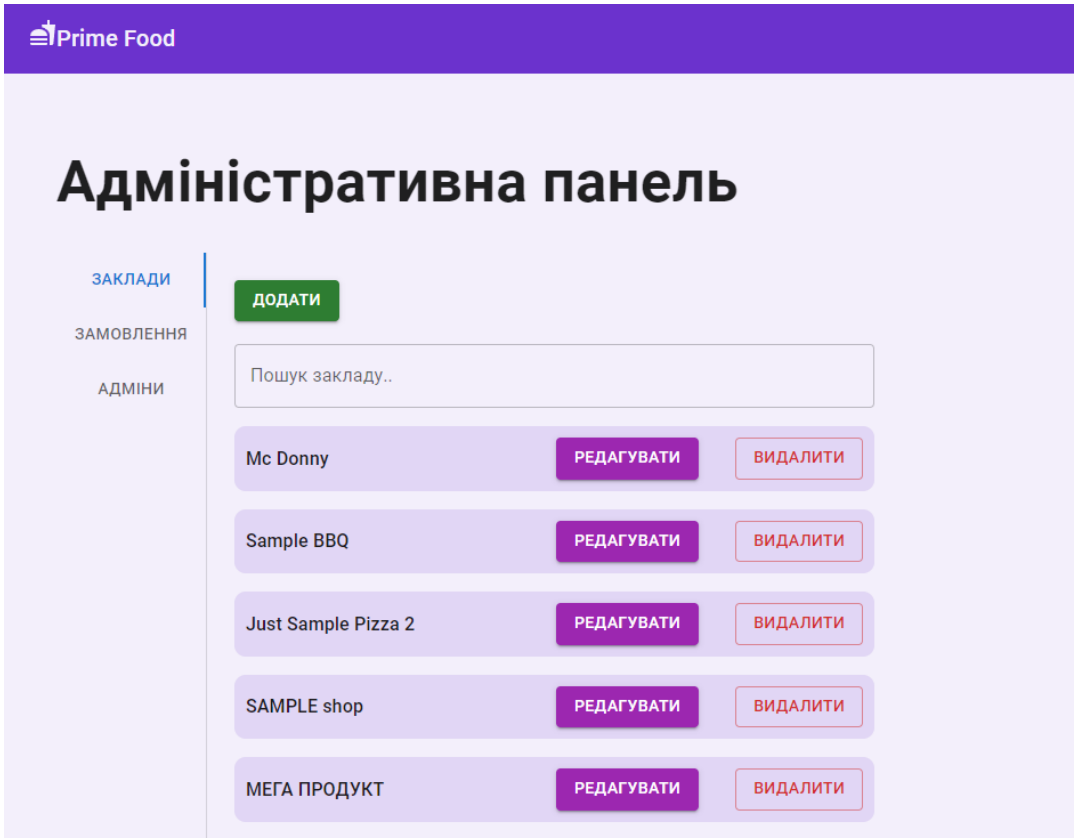

Рис.3.19. Адміністративна панель, заклади харчування

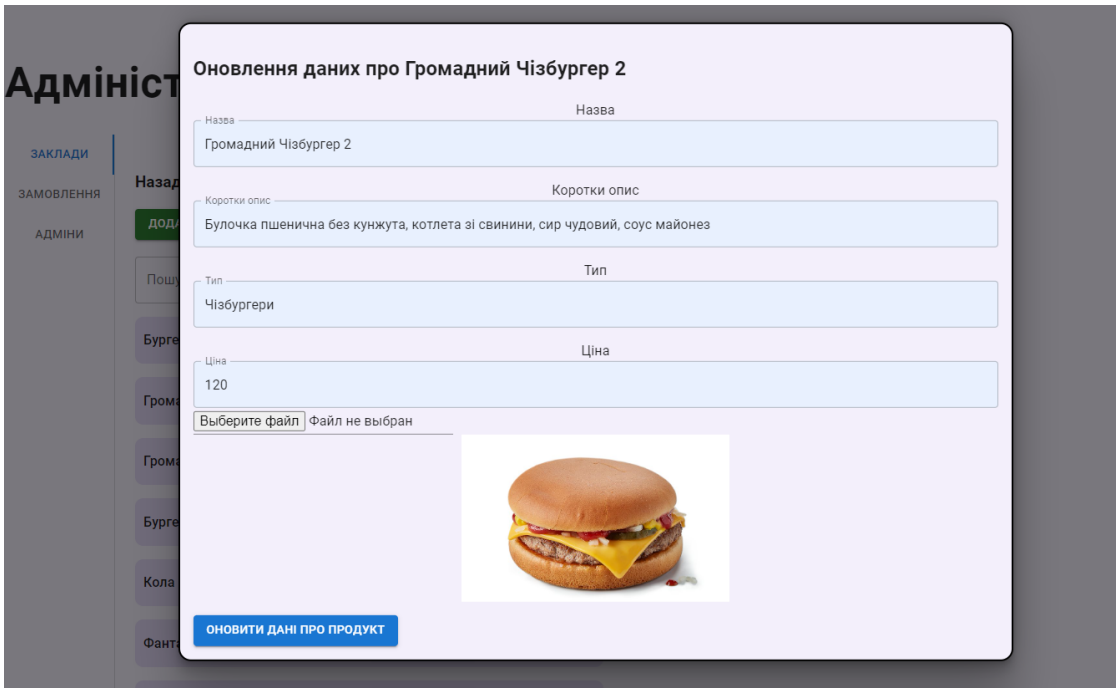

Рис.3.20. Адміністративна панель, редагування продукта

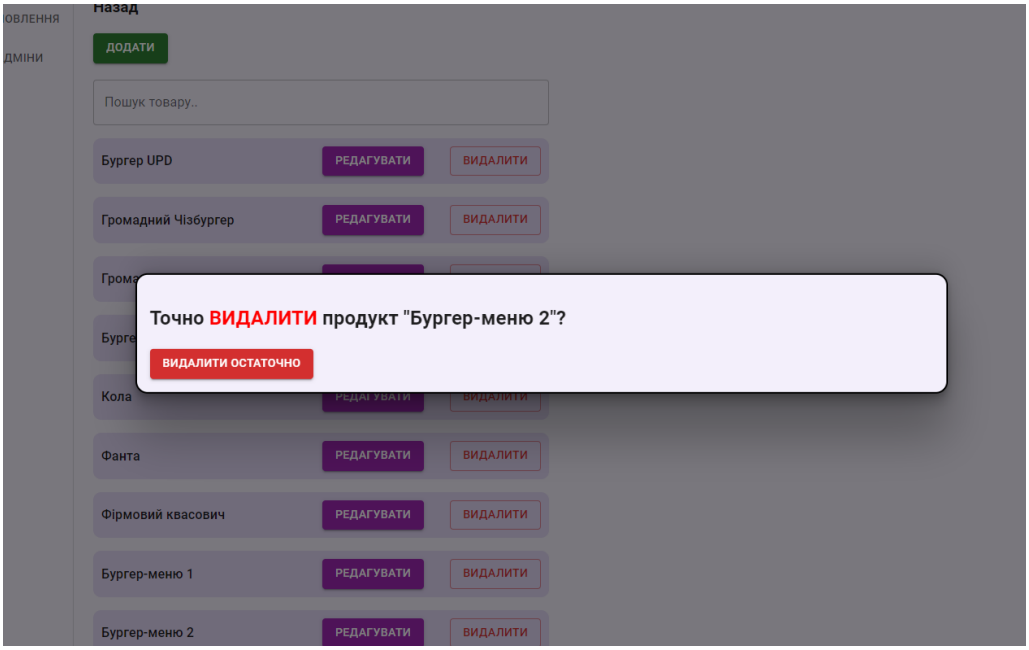

Рис.3.21. Адміністративна панель, попередження перед видаленням продукту

Висновки до розділу 3

В третьому розділі кваліфікаційної роботи було розглянуто основні методи для побудови реакт застосунку. Ми обрали потрбні фундаментальні бібліотеки для роботи фронтенду, такі як: React Router для навігації, Zustand для стейт менеджменту та Formik в поєднанні з Yup для перевірки валідації на формах.

Таким чином було розроблено якісний користувацький інтерфейс, який добре взаємодіє з сервером, має приємний та зручний дизайн. Було огляно роботу веб-додатка, розглянуто різні сценарії використання, що дало нам висновко, що все працює коректно та відповідає очікуваним результатам.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи на тему «Веб-додаток для замовлення та доставки їжі» було реалізована веб-сайт, використовуючі сучасні підходи та методи розробки веб-орієнтовних систем.

Для того, щоб розробити систему, були визначені конкретні вимоги до програмного продукту. Було проведено аналіз аналогів, виявлено їх переваги та недоліки на основі яких були визначені характерні риси сервісів для замовлення їжі. Після цього було досліджено якими технологіями краще розробляти проект, який архітектурний підхід слід обрати. Тож було визначено, щоб веб додаток буде розроблятись за допомогою стека MERN, з архітектурним підходом REST API.

У другому розділі кваліфікаційної роботи було спроектовано веб-систему, визначено її варіанти використання, основні алгоритми роботи. Це дозволило нам наочно побачити, як може працювати додаток. Далі було розроблено серверну частину веб-додатка, де ми використовували фреймворк Express.js в поєднанні з платформою Node.js. Маючи готовий бекенд, ми провели тестування основного функціонала за допомогою інструмента Postman.

У третьому розділі були визначені основні методи та інструменти для побудови фронтенду на базі React. Ми провели дослідження який краще використовувати підхід до використання глобального стейту, яку бібліотеку використовувати для навігації на клієнті, як перевіряти валідацію.

Отримавши завершений проект, ми оглянули хід роботи з ним, провели певні сценарії використання у ході яких було зазначено, що веб-додаток для замовлення та доставки їжі працює повністю правильно та корекно та відповідає поставленим вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Mister.Am [Електронний ресурс] – Режим доступу до ресурсу: https://misteram.com.ua/
2. Час поїсти [Електронний ресурс] – Режим доступу до ресурсу: https://chaspoisty.com/
3. Хочу з'їсти [Електронний ресурс] – Режим доступу до ресурсу: https://hz-food.com.ua/
4. DOU, спільнота розробників в Україні [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/
5. DEV, соціальна мережа для розробників, стаття про те, як працює асинхронний підхід в JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://dev.to/vinaykishore/how-does-asynchronous-javascript-work-behind-the-scenes-4bjl
6. Електронна документація до бази даних MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/docs/
7. Документація фреймворка Exprees.js [Електронний ресурс] – Режим доступу до ресурсу: https://expressjs.com/
8. Документація бібліотеки React [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/
9. Документація до платформи Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/
10. Стаття на ресурсі Mongo DB про стек MERN та подібні до нього [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/blog/post/the-modern-application-stack-part-1-introducing-the-mean-stack
11. Документація пакету express-validator [Електронний ресурс] – Режим доступу до ресурсу: https://express-validator.github.io/docs/
12. JWT Tokens [Електронний ресурс] – Режим доступу до ресурсу: https://jwt.io/

13. Пакет multer для обробки та зберігання файлів [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/multer
14. Postman, інструмент для розробки та тестування [Електронний ресурс] – Режим доступу до ресурсу: https://www.postman.com/
15. MDN Web Docs документація до мови JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript
16. Електронна документація React Router [Електронний ресурс] – Режим доступу до ресурсу: https://reactrouter.com/en/main
17. Документація до Redux [Електронний ресурс] – Режим доступу до ресурсу: https://redux.js.org/
18. Стейд менеджер Redux [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/redux
19. Стейд менеджер Zustand [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/zustand
20. Документація бібліотеки Formik [Електронний ресурс] – Режим доступу до ресурсу: https://formik.org/
21. Пакет Yup [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/yup

ДОДАТОК

Додаток А

Фрагмент лістингу UserController.js, містить методи для обробки такий операцій як: отримання даних про всіх адміністраторів, створення даних та метод для авторизації. Також маємо функцію, яка генерує JWT токен:

```
import User from "../models/User.js";
import Jwt from "jsonwebtoken";
import bcrypt from "bcrypt";

const generateAccessToken = (id, username) => {
  const payload = { id, username };
  return Jwt.sign(payload, process.env.JWT_KEY, { expiresIn: "24h" });
};

class UsersController {
  async getUsers(req, res) {
    try {
      const users = await User.find();
      res.status(200).json(users);
    } catch (err) {
      res.status(404).json({ message: err.message });
    }
  }

  async createUser(req, res) {
    try {
      const { username, password } = req.body;

      const existingUser = await User.findOne({ username: username });

      if (existingUser) {
        return res.status(422).json({ message: "This user exists already" });
      }

      const hashedPassword = await bcrypt.hash(password, 8);

      const newUser = new User({
        username,
        password: hashedPassword,
        role: "admin",
      });

      await newUser.save();

      res.status(201).json(newUser);
    } catch (err) {
      res.status(409).json({ message: err.message });
    }
  }
```

```
async login(req, res) {
  try {
    const { username, password } = req.body;
    const user = await User.findOne({ username });
    if (!user) {
      return res
        .status(400)
        .json({ message: `Акаунт з нікнеймом: ${username} не знайдено` });
    }

    const validPassword = bcrypt.compareSync(password, user.password);

    if (!validPassword) {
      return res.status(400).json({ message: `Невірний пароль` });
    }

    const token = generateAccessToken(user._id, user.username);

    return res.json({ token: token, role: user.role });
  } catch (err) {
    console.log(err);
    res.status(400).json({ message: "Login error" });
  }
}
```