Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Адаптивний веб-застосунок для замовлення й доставки продуктів харчування

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-4
Даніїл КАСЯНЧУК
(власне ім’я, прізвище)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: ст. викл. Роман ЯРОШЕВИЧ
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"_____" _____ 20___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Касянчук Даніілу Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Адаптивний веб-застосунок для замовлення й доставки продуктів харчування

затверджена наказом по університету від "26" травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи

1) документація мови програмування JavaScript;
2) документація по HTML та CSS;
3) документація до Node.js;
4) редактор коду: Visual Studio Code;
5) перелік використаних програмних засобів: браузери Google Chrome та Mozilla Firefox;
6) операційна система: Windows 11.

4. Перелік питань, що потрібно опрацювати у роботі

1) аналіз проблеми та огляд існуючих рішень;
2) вибір мови програмування для розробки;
3) вибір бібліотек та фреймворків;
4) розробка макету сайту;
5) написання коду;
6) виправлення помилок;
7) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій Слайд-презентація – 17 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	27.05.25-28.05.25	
2	Вибір мови програмування для розробки	29.05.25-30.05.25	
3	Вибір бібліотек та фреймворків	31.05.25-01.06.25	
4	Розробка макету сайту	02.06.25-03.06.25	
5	Написання коду	04.06.25-07.06.25	
6	Виправлення помилок	08.06.25-09.06.25	
7	Оформлення матеріалів кваліфікаційної роботи	10.06.25-11.06.25	
8	Подання кваліфікаційної роботи керівникові та	12.06.25-13.06.25	
	її попередній захист		
9	Подання кваліфікаційної роботи на рецензу-	14.06.25-16.06.25	
	вання		

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ст. викл. Роман ЯРОШЕВИЧ
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 90 с., 25 рис., 2 дод., 7 джерел.

ДОСТАВКА, ВЕБ-ЗАСТОСУНОК, JAVASCRIPT, NODE.JS, EXPRESS, EJS, БАЗИ ДАНИХ, ІНТЕРФЕЙС, РЕСТОРАНИ, КУР'ЄР, КОРИСТУВАЧ.

Метою кваліфікаційної роботи є розробка веб-застосунку для замовлення та доставки продуктів харчування з інтуїтивно зрозумілим та ефективним для користувача та працівника інтерфейсом з використанням сучасних технологій для розробки веб-застосунків.

У ході виконання кваліфікаційної роботи було розроблено функціональний веб-застосунок для замовлення та доставки продуктів харчування з використанням мови програмування JavaScript. Серверна частина сайту була виконана за допомогою Node.js та Express, інтерфейс користувача – за допомогою HTML сторінок, а саме в шаблонізаторі EJS. Для збереження даних використовувалась база даних SQLite. Для різних функцій використовувалися бібліотеки, доступні завдяки використанню платформи Node.js, такі як dotEnv, Express.js, bcrypts та ін. Веб-сайт надає користувачеві такі можливості як: реєстрація та авторизація користувача, перегляд та замовлення продуктів харчування, створення замовлення, відслідковування замовлення, редагування особистої інформації, зміна паролю. Також користувач має роль адмін та кур’єр, які дозволяють редагувати статус замовлення, передавати статус кур’єру та відслідковувати замовлення яке потрібно доставити.

ABSTRACT

Bachelor’s thesis: 90 pages, 25 figures, 2 appendices, 7 sources.

DELIVERY, WEB APPLICATION, JAVASCRIPT, NODE.JS, EXPRESS, EJS, DATABASES, INTERFACE, RESTAURANTS, COURIER, USER.

The major goal of this thesis is to develop a web application for ordering and delivering food products with an intuitive and efficient interface for both users and employees, utilizing modern web development technologies.

During the implementation of the qualification project, a functional web application for food ordering and delivery was developed using the JavaScript programming language. The server-side of the website was built using Node.js and Express, while the user interface was implemented using HTML pages rendered with the EJS templating engine. Data storage was handled using an SQLite database. Various libraries available through the Node.js platform was used for different functionalities, such as dotEnv, Express.js, bcrypt, and others. The website provides users with the following features: user registration and login, browsing and ordering food products, order creation, order tracking, editing personal information, and changing passwords. The user can also have roles such as admin or courier, which allow updating order status, assigning orders to couriers, and tracking deliveries.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ 8
ВСТУП 9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ВИМОГ ДО ВЕБ-ЗАСТОСУНКУ 10
1.1 Актуальність теми 10
1.2 Дослідження функціональних можливостей існуючих рішень ... 11
1.2.1 Аналіз платформи Glovo 12
1.2.2 Аналіз платформи Bolt Food 13
1.3 Формування вимог до розроблюваного веб-застосунку 15
1.4 Постановка задачі кваліфікаційної роботи 18
2 ОПИС АРХІТЕКТУРИ ТА ІНТЕРФЕЙСУ ВЕБ-ЗАСТОСУНКУ 19
2.1 Обґрунтування вибору технологічного стеку 19
2.1.1 Основні відомості про JavaScript 20
2.1.2 Основні відомості про Node.js 22
2.1.3 Основні відомості про Express.js 23
2.1.4 Основні відомості про JSON WebToken 24
2.1.5 Основні відомості про Embedded JavaScript 24
2.1.6 Основні відомості про SQLite 25
2.1.7 Основні відомості про DotEnv 26
2.2 Розробка архітектури веб-застосунку 27
2.2.1 Проєктування клієнт-серверної архітектури 28
2.2.2 Проєктування структури бази даних 29
2.2.3 Організація взаємодії компонентів системи 30
2.3 Проектування інтерфейсу користувача 31
2.3.1 Розробка концепції UX/UI дизайну 31
2.3.2 Аналіз UX/UI дизайну успішних брендів 33
2.3.3 Формування візуальної ідентичності застосунку 35

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ 36
3.1 Організація структури 36
3.2 Реалізація серверної частини застосунку 37
3.2.1 Опис контролерів на базі userController.js 38
3.2.2 Опис раутсів на базі userRoutes.js 41
3.3 Реалізація клієнтської частини застосунку 44
3.3.1 Опис хелпера на базі user.js 44
3.3.2 Опис аутентифікації на базі auth.js 46
3.3.3 Опис сторінок на базі account.ejs 49
4 ТЕСТУВАННЯ ФУНКЦІОНАЛЬНОСТІ ВЕБ-ЗАСТОСУНКУ 51
4.1 Покращення архітектури і структури веб-застосунку 59
ВИСНОВКИ 61
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 63
ДОДАТОК А ГРАФІЧНИЙ МАТЕРІАЛ КВАЛІФІКАЦІЙНОЇ РОБОТИ 64
ДОДАТОК Б 72
ПРОГРАМНИЙ КОД 72
Б.1 Файли реалізації функціоналу 72
Б.2 Файл реалізації сторінки проекту 76

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

СУБД – система управління базами даних

API – програмний інтерфейс прикладного програмування (англ., Application Programming Interface)

DOM – об'єктна модель документа (англ., Document Object Model)

EJS – вбудований JavaScript шаблонізатор (англ., Embedded JavaScript templates)

HR – управління людськими ресурсами (англ., Human Resources)

HTTP – протокол передавання гіпертексту (англ., HyperText Transfer Protocol)

ID – ідентифікатор (англ., Identifier)

MVC – модель–представлення–контролер (англ., Model–View–Controller)

MVP – мінімально життєздатний продукт (англ., Minimum Viable Product)

SMM – маркетинг у соціальних мережах (англ., Social Media Marketing)

SMS – служба коротких повідомлень (англ., Short Message Service)

URL – уніфікований локатор ресурсу (англ., Uniform Resource Locator)

UX/UI – користувацький досвід та інтерфейс (англ., User Experience / User Interface)

ВСТУП

У сучасному світі стрімко зростає популярність служб доставки їжі та продуктів. Швидкий ритм життя, нестача часу на покупки та прагнення до зручності стимулюють розвиток онлайн-сервісів, які дозволяють вирішувати побутові завдання всього за кілька кілків. Особливо це стало помітним упродовж останніх років, коли споживацькі звички радикально змінилися під впливом технологічного прогресу та інших проблем.

В умовах COVID-19 та війна зросли ще більше: часті переїзди, віддалена робота, перебої з інфраструктурою – усе це підвищило попит на зручні та надійні сервіси. Все більше людей користуються сучасними технологіями, що створює сприятливе середовище для розвитку адаптивних веб-платформ.

Навіть наяввність таких корпорацій, як Glovo та Bolt Food, ринок має потребу в більш доступних і зручних рішеннях. Більшість подібних сервісів орієнтовані насамперед на мобільні додатки, тоді як веб-версії часто мають обмежену функціональність і не завжди зручний інтерфейс. При цьому багато користувачів, зокрема вдома або в офісі, вважають за краще оформлювати замовлення саме з комп’ютера або ноутбука.

Окрім того, не всі існуючі сервіси враховують особливості українського ринку – невеликі міста, слабке інтернет-з’єднання, обмежене геопокриття тощо. Тому актуальним завданням є створення універсального, адаптивного й ефективного веб-додатку, який буде зручним у користуванні навіть у складних умовах.

Метою цієї кваліфікаційної роботи стала розробка сучасного веб-додатку для замовлення продуктів харчування. Особлива увага приділялася дизайну: він мав бути зрозумілим, інтуїтивним, з логічною навігацією, адаптивною версткою, зручними кнопками та шрифтами. Основна мета – зробити сервіс максимально доступним і простим у використанні для широкого кола користувачів незалежно від віку чи технічного досвіду.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ВИМОГ ДО ВЕБ-ЗАСТОСУНКУ

1.1 Актуальність теми

У нинішньому контексті цифрової трансформації онлайн-доставка їжі та продуктові послуги стали невід'ємною частиною повсякденного життя. Напружений ритм, постійна робота та бажання заощадити час викликають попит на послуги, які дозволяють людям робити покупки, не виходячи з дому. Онлайн-платформи, що пропонують ці послуги, з кожним роком набирають популярності, особливо серед жителів міст.

Це явище стало особливо помітним в останні роки, коли споживчі звички радикально змінилися під впливом технологічного прогресу та глобальних подій, зокрема пандемії COVID-19. В умовах карантинних обмежень значна частина населення вибрала дистанційну роботу, що збільшило потребу в зручних і безпечних способах доступу до основних товарів.

В Україні актуальність цього питання ще більше підсилюється наслідками масштабної війни. Вимушене переміщення, порушення ланцюгів постачання, нестабільна робота фізичних магазинів у прифронтових районах і, ширше, зміна темпу життя мільйонів громадян сприяли стрімкому зростанню потреби у високоякісних послугах онлайн-доставка. Підвищення цифрової грамотності населення та розширення інтернет-покриття також є факторами, які слід враховувати.

Однак, незважаючи на присутність великих гравців на ринку, таких як Glovo і Bolt Food, більшість з них зосереджуються переважно на мобільних додатках, тоді як веб-версії їхніх сервісів мають обмежену функціональність або не пропонують оптимального досвіду користувача. Це створює нішу для розробки адаптивних, зручних веб-додатків, доступних з будь-якого пристрою

та придатних для потреб широкої аудиторії, від дітей до людей похилого віку.

Тому розробка адаптивного веб-додатку для доставки їжі та продуктів є актуальним та соціально важливим завданням, яке відповідає сучасним викликам та потребам українського ринку.

1.2 Дослідження функціональних можливостей існуючих рішень

Сьогодні на українському ринку доставки їжі та продуктів домінують кілька великих компаній, найвідоміші з яких Glovo та Bolt Food. Ці сервіси активно розвиваються у великих містах, таких як Київ, Львів, Харків, Дніпро, Одеса, і мають значну частку користувачів завдяки масштабним рекламним кампаніям, швидкому обслуговуванню та широкому вибору закладів-партнерів.

Glovo – міжнародна платформа іспанського походження, яка присутня в Україні з 2018 року. Її головна перевага – універсальність: користувачі можуть замовляти не лише ресторанні страви, а й магазинні продукти, ліки та речі для дому. Однак послуга в основному зосереджена на мобільному додатку, а веб-версія пропонує обмежену функціональність, включаючи фільтрацію, збереження історії замовлень і керування профілями.

Bolt Food є дочірньою компанією естонської компанії Bolt, яка також активно представлена на українському ринку. Його сильними сторонами є швидкість доставки та інтеграція з основним додатком Bolt, який поєднує послуги таксі, оренди скутерів і доставки їжі. Bolt Food має привабливий мобільний інтерфейс, але, як і Glovo, веб-платформа є другорядною та не дуже інтуїтивно зрозумілою.

Незважаючи на сильні сторони обох компаній, є кілька слабких сторін, які залишають бажати кращого. Серед них складний або перевантажений інтерфейс, обмежена доступність для малих міст, недостатня адаптація до різного віку користувачів, а також нестабільна робота при слабкому інтернет-з’єднанні. Таким чином, хоча Glovo і Bolt Food є безперечними лідерами

ринку, їхній акцент здебільшого зосереджений на мобільних додатках. Це залишає простір для створення практичних, функціональних і адаптивних веб-додатків, що пропонують більший охоплення, кращу доступність і більшу універсальність використання. Існуючі послуги ще не охоплюють усі платформи (веб- і мобільні), а надають перевагу лише одній із них.

1.2.1 Аналіз платформи Glovo

Glovo – міжнародна служба доставки, що працює на ринку [2]. Він дозволяє замовляти страви з ресторанів, продукти з супермаркетів, ліки з аптек, а також доставляти будь-які інші товари по місту. Сервіс, заснований у 2015 році в Іспанії, стрімко розширювався і наразі представлений у понад 20 країнах, включаючи Україну (рисунок 1.1).

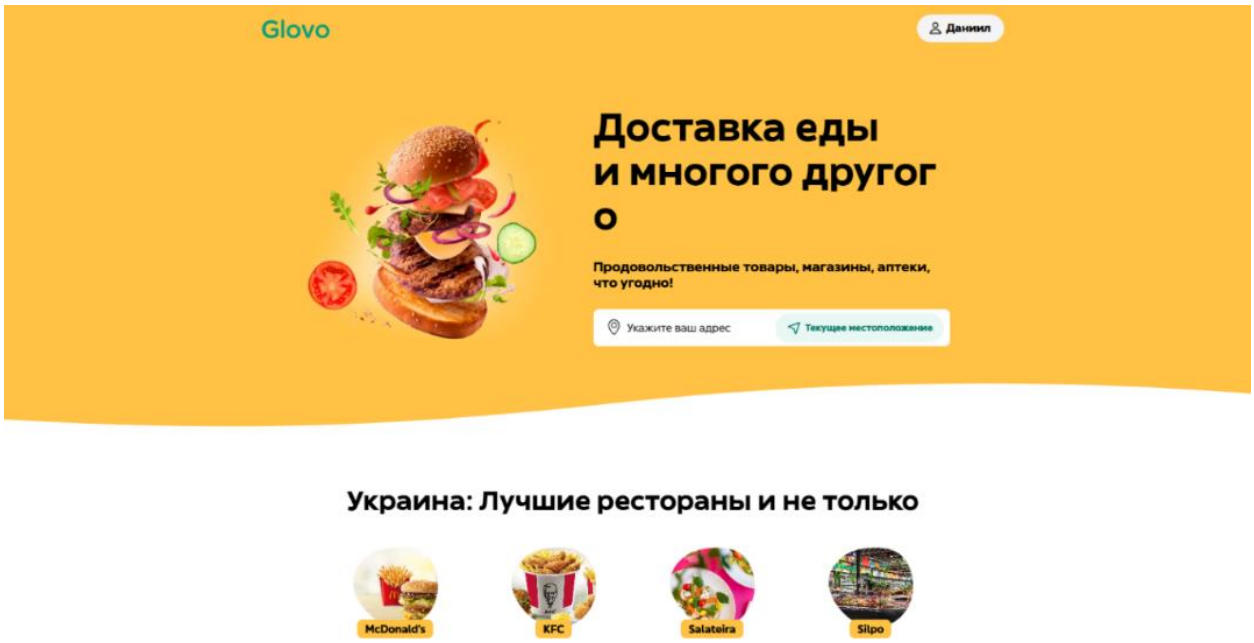

Рисунок 1.1 – Головна сторінка сайту Glovo

Glovo пропонує простий та інтуїтивно зрозумілий інтерфейс як на своєму веб-сайті, так і в мобільному додатку. Для замовлення просто введіть адресу. Система автоматично відобразить ресторани, магазини та інші послуги, доступні поблизу. Користувачам доступні десятки брендів на вибір:

McDonald’s, KFC, Сільпо, Salateira та ін. Також можлива доставка будь-яких видів товарів або передача об’єктів між фізичними особами.

Основними перевагами Glovo є швидка доставка (часто до 30 хвилин), широкий вибір партнерів, відстеження кур’єрської доставки в реальному часі, а також знижки та бонуси для постійних користувачів. Особливо зручною є опція «Кур’єр»: користувач може замовити доставку навіть без прив’язки до магазину чи ресторану.

Однак у цієї послуги є і недоліки. У години пік час доставки може значно збільшитися. Буває і так, що кур’єри затримуються або не можуть виконати замовлення з технічних причин. Комісії ресторанів і магазинів іноді високі, що може вплинути на кінцеву ціну. Деякі користувачі також скаржаться на роботу служби підтримки, якщо виникають проблеми з їх замовленням.

1.2.2 Аналіз платформи Bolt Food

Bolt Food – це служба доставки їжі та частина глобальної екосистеми Bolt [3]. Вона також включає послуги таксі, прокат скутерів і велосипедів, каршеринг, доставку продуктів і бізнес-рішення. Головна сторінка веб-застосунку Bolt візуально чітка, логічна, але вона одразу пропонує мобільне рішення свого застосунку (рисунок 1.2).

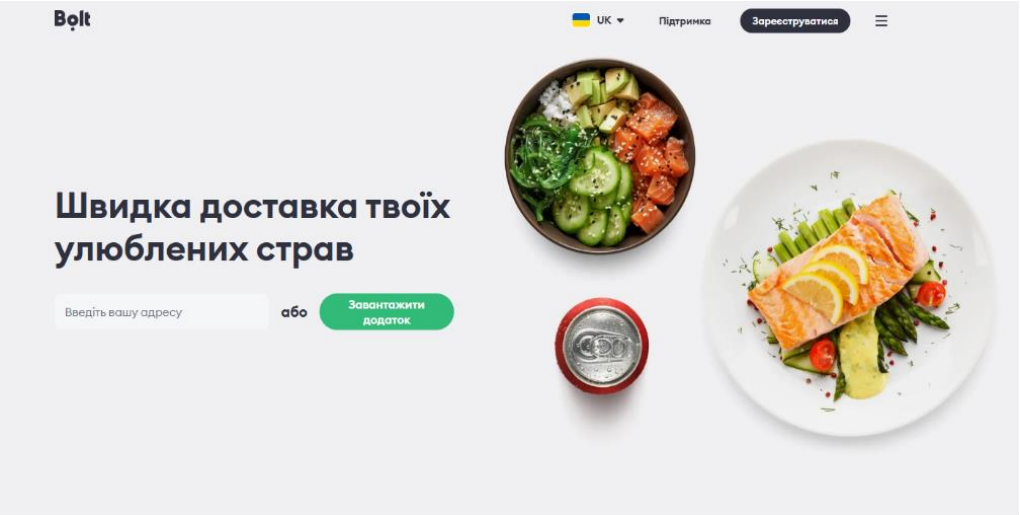

Рисунок 1.2 – Головна сторінка сайту Bolt Food

На українському ринку Bolt Food представлена у великих містах: Києві, Львові, Одесі та Дніпрі. Ця послуга дозволяє замовляти страви у великій кількості ресторанів і кафе, будь то міжнародні мережі або місцеві заклади. Додаток пропонує можливість відстежувати замовлення в режимі реального часу, платити безконтактно, переглядати історію замовлень та інші основні функції.

Серед переваг Bolt – швидка доставка, часті акції та інтеграція з іншими сервісами Bolt (наприклад, можливість замовити таксі відразу після їжі). Кур'єрам також пропонується гнучка система бонусів, яка стимулює їх доставляти замовлення швидше.

Однак веб-сайту Bolt Food бракує зручності та ясності. Він є частиною головного веб-сайту Bolt, який об'єднує таксі, скутери, доставку, бізнес-рішення та інші послуги. Тому навігація складна, особливо для тих, хто просто хоче замовити їжу. Більшість функцій Bolt Food реалізовано через мобільний додаток, тоді як веб-сайт пропонує не всі функції. Деякі сторінки містять загальні блоки, не пов'язані з доставкою їжі, через що інтерфейс може виглядати неповним або захаращеним.

Інші недоліки включають обмежене географічне покриття, вечірні затримки та іноді неефективне обслуговування клієнтів, яке повільно реагує на запити користувачів. Тим не менш, Bolt Food продовжує розвиватися і залишається конкурентоспроможним гравцем на ринку доставки їжі в Україні. Цей багатофункціональний підхід дозволяє користувачам користуватися всіма послугами в одній програмі, але створює певну плутанину, особливо для нових користувачів. Приклад складної навігації через багатофункціональність платформи показано на рисунку 1.3.

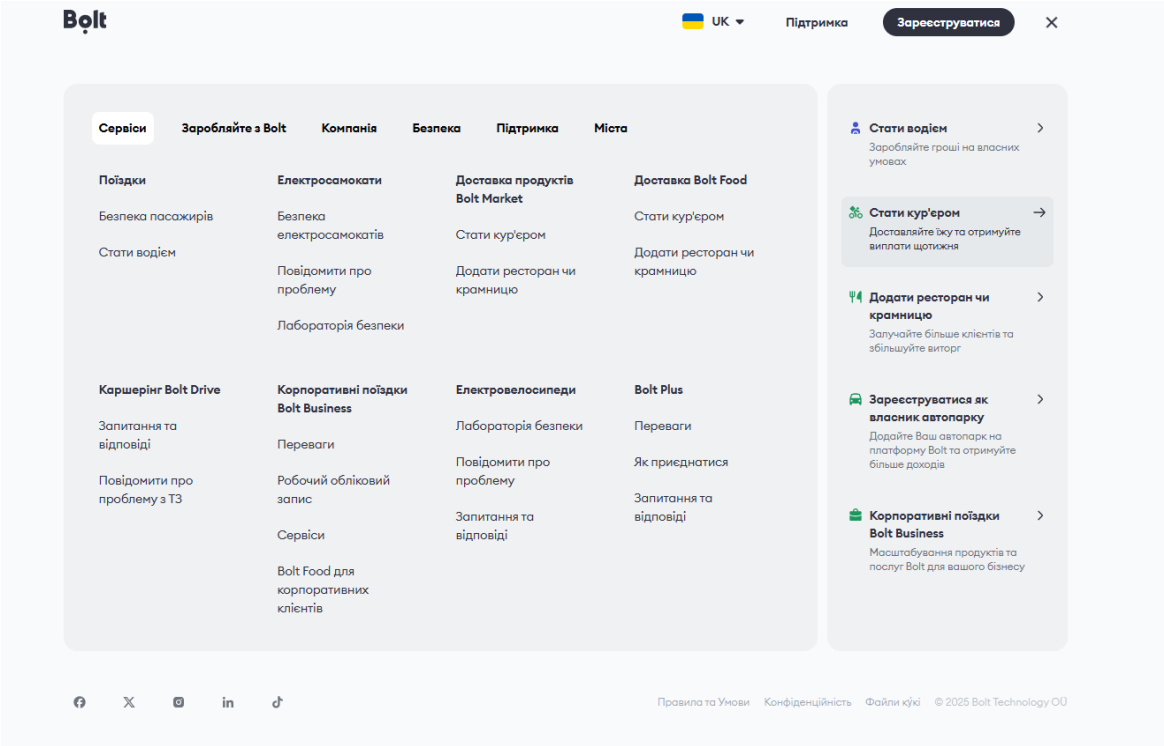

Рисунок 1.3 – Інтерфейс навігаційного меню Bolt із секцією Bolt Food

1.3 Формування вимог до розроблюваного веб-застосунку

У процесі розробки веб-застосунку для онлайн-доставки їжі та продуктів важливо оцінити не лише технічні аспекти реалізації, але й загальну доцільність обраної теми. Зважаючи на актуальні суспільні, економічні та технологічні тенденції, варто проаналізувати основні переваги та можливі недоліки такого проєкту. Такий аналіз дозволяє глибше зрозуміти потенціал розробки, визначити сильні сторони, а також врахувати ризики та виклики, які можуть виникнути під час реалізації та використання продукту.

Провівши аналіз було виявлено наступні переваги:

- актуальність – тема відповідає потребам сучасного суспільства, зокрема в контексті війни, цифровізації та зміни споживчих звичок;
- соціальна важливість – онлайн-доставка їжі та продуктів полегшує життя людям у скрутному становищі: людям похилого віку, внутрішньо переміщеним особам, дистанційним працівникам;
- ринковий попит – незважаючи на наявність великих гравців, існує

попит на нові, адаптивні, практичні та прості веб-рішення;

- широка цільова аудиторія – від дітей, які можуть робити замовлення з аккаунта батьків, до людей похилого віку, які цінують простоту та зрозумілість інтерфейсу;

- володіння сучасними технологіями – реалізація теми дозволяє працювати зі звичайними інструментами (JavaScript, Node.js, CSS/SCSS, SQLite), що підвищує професійну цінність розробника.

Також було виявлено такі недоліки:

- сильна конкуренція – ринок насичений потужними брендами з великими бюджетами та командами розробників;

- логістична залежність – навіть найкраща веб-програма не матиме успіху без якісної служби доставки (що виходить за рамки дипломного проекту);

- технічна складність адаптації – створення справді практичного та адаптивного інтерфейсу вимагає ретельної роботи над інтерфейсом користувача та досвідом користувача;

- обмежені ресурси в освітньому проекті – важко реалізувати всі функції повноцінних комерційних платформ.

Підсумувавши можна виділити, що основною перевагою розробки веб-додатку є його робота безпосередньо у веб-середовищі, що забезпечує максимальну доступність для користувачів без встановлення додаткового програмного забезпечення та перевантаження пам’яті пристрою. Такий підхід дозволяє охопити ширшу аудиторію та спростити доступ до сервісу. Крім того, недоліки поточної реалізації включають обмежену масштабованість, зокрема з точки зору навантаження на сервер, інтеграцію із зовнішніми службами та потенційну підтримку великої кількості одночасних користувачів.

На підставі аналізу існуючих рішень та виявлених недоліків до розроблюваного веб-застосунку висуваються такі основні вимоги:

- система повинна забезпечувати надійну автентифікацію та авторизацію користувачів;

- реєстрація нових користувачів із валідацією введених даних, вхід у систему через JWT-токени та розмежування прав доступу для трьох категорій – клієнтів, адміністраторів та кур’єрів;
- клієнтам необхідно надати зручний інтерфейс для перегляду каталогу продуктів із можливістю сортування, фільтрації, формування кошика та оформлення замовлення;
- система повинна підтримувати відстеження статусу замовлення в реальному часі, керування особистим кабінетом (редагування профілю, зміна паролю та адреси доставки) і надавати доступ до історії попередніх замовлень;
- для адміністраторів передбачено інструменти керування замовленнями (зміна статусів, призначення кур’єрів), перегляду статистики та аналітики, управління обліковими записами користувачів, а також додавання та редагування товарів у каталозі;
- кур’єри мають отримувати перелік призначених їм замовлень із можливістю оновлювати статус доставки та переглядати контактну інформацію клієнтів для оперативної роботи;
- інтерфейс застосунку повинен бути інтуїтивно зрозумілим і відповідати принципам мінімалістичного дизайну, уникати інформаційного перевантаження та мати логічно структуровану навігацію;
- адаптивність реалізації – коректне відображення та працездатність на смартфонах, планшетах і ПК без потреби встановлення окремих додатків, лише в стандартному веб-браузері;
- забезпечення швидкого завантаження сторінок, ефективна обробка одночасних запитів і оптимізована взаємодію з базою даних;
- забезпечення безпеки – захист персональних даних користувачів, захист паролів з використанням бібліотеки bcryptjs та додаткові перевірки доступу через middleware-шари;
- архітектура застосунку має бути модульною для полегшеного розширення функціоналу, підтримувати міграцію на більш потужні СУБД та витримувати значне зростання числа користувачів без втрати продуктивності.

1.4 Постановка задачі кваліфікаційної роботи

Метою кваліфікаційної роботи є розробка адаптивного веб-застосунку для доставки їжі, який дозволить замовляти їжу та відслідковувати клієнтам процес доставки та контролю всього процесу. Для досягнення мети кваліфікаційної роботи та реалізації вимог до продуктивності, масштабованості та зручності використання було визначено такі послідовні етапи:

- провести аналіз існуючих рішень на ринку доставки продуктів харчування та виявити їх переваги і недоліки;
- обґрунтувати вибір технологічного стеку для реалізації веб-застосунку з урахуванням вимог до продуктивності та масштабованості;
- спроектувати архітектуру веб-застосунку з розподілом на клієнтську, серверну частини та базу даних;
- розробити інтуїтивно зрозумілий користувацький інтерфейс з урахуванням принципів UX/UI дизайну та адаптивності;
- реалізувати серверну частину з використанням Node.js та Express.js, включаючи: систему автентифікації та авторизації, RESTful API для взаємодії з клієнтською частиною та бізнес-логіку обробки замовлень;
- створити клієнтську частину з використанням EJS-шаблонів та забезпечити динамічну взаємодію з сервером;
- реалізувати систему відстеження замовлень з візуалізацією етапів доставки;
- провести тестування функціональності веб-застосунку та оцінити відповідність реалізованого рішення поставленим вимогам.

2 ОПИС АРХІТЕКТУРИ ТА ІНТЕРФЕЙСУ ВЕБ-ЗАСТОСУНКУ

Адаптивний веб-застосунок реалізовано з використанням таких засобів реалізації:

- середовище розробки Visual Studio Code;
- мова програмування JavaScript;
- Node.js – для побудови серверної частини, обробки HTTP-запитів та маршрутизації;
- механізм шаблонів EJS (Embedded JavaScript) – для динамічного створення HTML-сторінок;
- мови розмітки та стилізації HTML, CSS та SCSS – для адаптивного дизайну клієнтської частини;
- локальна база даних SQLite – для зберігання інформації про продукти, замовлення та інші елементи веб-сервісу.

2.1 Обґрунтування вибору технологічного стеку

При розробці веб-додатку для доставки їжі та товарів мова програмування JavaScript була свідомо обрана як основна технологія створення клієнтської та серверної частин. Такий вибір пояснюється його універсальністю та широким застосуванням у сучасній веб-розробці. JavaScript дозволяє реалізувати повну логіку як на стороні користувача (frontend), так і на стороні сервера (backend), що значно спрощує керування уніфікованою структурою проекту, полегшує передачу даних між різними частинами програми та мінімізує складність навчання та масштабування [4].

Серверна частина реалізована на основі середовища виконання Node.js, що забезпечує швидку та ефективну обробку запитів завдяки неблокуючій асинхронній моделі. Це особливо важливо для програм, які одночасно використовують велику кількість користувачів, які надсилають запити до бази

даних або розміщують замовлення.

На стороні клієнта JavaScript дозволяє створювати адаптивні та інтерактивні інтерфейси, сумісні з усіма сучасними браузерами без додаткових плагінів. Такий підхід забезпечує зручність і доступність сервісу на різних пристроях, будь то комп'ютер, смартфон або планшет. Крім того, використання цієї мови спрощує реалізацію інтерактивних елементів, динамічне оновлення вмісту сторінки, перевірку форм та інші функції, необхідні для оптимальної взаємодії з користувачем.

JavaScript також має розвинену екосистему бібліотек і модулів, які активно використовуються в проекті: для конфігурації, обробки даних, захисту інформації та інтеграції з іншими сервісами. Це дозволяє гнучко та швидко впроваджувати необхідну логіку та адаптувати проект у майбутньому.

Таким чином, JavaScript у поєднанні з Node.js забезпечує надійну та ефективну основу для створення сучасного, практичного та адаптивного веб-додатку, який відповідає вимогам ринку та очікуванням кінцевих користувачів.

2.1.1 Основні відомості про JavaScript

JavaScript – це інтерпретована мова програмування високого рівня, яка широко використовується для створення інтерактивних інтерфейсів у браузерах. Завдяки механізмам виконання, вбудованим у сучасні веб-браузери (наприклад, V8 у Google Chrome), JavaScript дозволяє обробляти дії користувача, змінювати вміст сторінки без перезавантаження, впроваджувати анімацію, перевіряти форми, працювати з динамічними елементами та взаємодіяти з сервером через API. Це важливо для створення зручного, швидкого та чуйного інтерфейсу, що є важливою вимогою для служби доставки [7].

Ще однією перевагою є можливість використовувати JavaScript на сервері через Node.js, високопродуктивне середовище виконання для роботи з

файлами, мережами та базами даних. Це дозволяє створювати повну серверну логіку, не виходячи за межі однієї мови, спрощуючи архітектуру, полегшуючи налагодження коду та прискорюючи розробку. Такий підхід також зменшує необхідність залучення кількох спеціалістів до різних частин проекту.

JavaScript підтримує кілька парадигм програмування (об'єктно-орієнтовану, функціональну та імперативну), забезпечуючи гнучкість у реалізації бізнес-логіки різної складності. Він використовує динамічну типізацію, об'єктну модель прототипу, а функції є першокласними об'єктами, що відкриває широкі можливості для створення гнучких і модульних архітектур.

Окрім технічних особливостей, важливим фактором у виборі JavaScript є його розвинена екосистема. Велика кількість готових до використання бібліотек і фреймворків (включаючи Express, React, Vue.js, Axios, Sequelize і т. д.) дозволяє значно прискорити розробку і впровадження ключових функцій. Це також спрощує підтримку та масштабування проекту.

Ось чому JavaScript було обрано як універсальну мову, що охоплює як інтерфейс, так і бек-енд, пропонуючи гнучкість, високу продуктивність, доступ до багатьох інструментів і активну спільноту підтримки. У рамках цього проекту це дозволяє створити повну, надійну та просту у використанні веб-програму для замовлення та доставки їжі.

Цей підхід також допомагає зберегти єдність технологічного стеку, що є основним активом для командної роботи та масштабованості додатків. Крім того, використання єдиної мови для всіх систем (розробка повністю на основі JavaScript) зменшує технічний борг і забезпечує кращу читабельність і повторне використання коду між клієнтською та серверною сторонами. Тому витрати на розробку, тестування та підтримку зменшуються.

Завдяки активній спільноті JavaScript постійно розвивається: регулярно з'являються нові інструменти, шаблони та оновлення стандарту ECMAScript, що дозволяє залишатися в авангарді технологій веб-розробки. Розробники можуть швидко знаходити рішення типових проблем, використовувати

сучасні бібліотеки з відкритим кодом і легко інтегрувати зовнішні служби, такі як платіжні системи, аналітичні служби, картографічні API та інші компоненти, необхідні для роботи служби доставки.

Усі ці фактори роблять JavaScript ідеальним вибором для реалізації цього типу веб-додатків, де швидкість, масштабованість, інтерактивність, адаптивність пристрою та здатність швидко розробляти нові функції у відповідь на потреби користувача є важливими.

2.1.2 Основні відомості про Node.js

У цьому проекті для розробки адаптивного веб-додатку для доставки їжі та продуктів Node.js був обраний як основна серверна платформа. Це крос-платформне середовище виконання JavaScript, засноване на потужному двигуні Google V8, який дозволяє виконувати код JavaScript поза браузером. Це забезпечує єдність клієнтської та серверної технологій програми, що спрощує розробку, тестування та підтримку проекту [1].

Node.js підтримує неблокуючу, керовану подіями модель введення-виведення, що робить його особливо ефективним при обробці великої кількості одночасних запитів. Цей аспект важливий для сучасних веб-додатків у сфері доставки, де необхідно забезпечити миттєве реагування на дії користувача, оновлення замовлення в реальному часі та взаємодію з базою даних та API.

Node.js також має розвинену екосистему, що забезпечує доступ до великої кількості готових до використання модулів через NPM (Node Package Manager). Проект використовував бібліотеки для створення REST API, роботи з базами даних (наприклад, SQLite), реалізації авторизації, обробки запитів користувачів і обробки логіки сервера. Це дозволило значно скоротити час розробки та зосередитися на впровадженні бізнес-логіки.

Ще однією важливою перевагою є швидкість: код JavaScript в середовищі Node.js працює дуже швидко, що позитивно впливає на загальну

продуктивність програми. Середовище також отримує переваги від активної підтримки спільноти та постійно оновлюється відповідно до нових стандартів ECMAScript, що гарантує перспективність рішення.

2.1.3 Основні відомості про Express.js

Для свого проекту з розробки адаптивної веб-програми для доставки їжі та продуктів я використовував Express.js, мінімалістичну та гнучку структуру для Node.js, яка значно спрощує створення веб-серверів і API. Його використання дозволило мені організувати чітку архітектуру маршрутизації, розділити логіку на окремі модулі та реалізувати публічні та безпечні маршрути.

Express.js легко інтегрується з функціями проміжного програмного забезпечення, що полегшує додавання автентифікації, обробки помилок та інших необхідних функцій. Наприклад, у моєму додатку були реалізовані окремі маршрути для реєстрації, входу, виходу, сторінок сайту (домашня, про, магазин, стрічка, контакти, відстеження) і створення замовлення. Завдяки підтримці проміжного програмного забезпечення фреймворком було легко підключити модуль авторизації до безпечних маршрутів, таких як /about або /logout, де автентичність користувача перевіряється перед наданням доступу.

Express також дозволив чітко розділити маршрути на логічні частини: наприклад, маршрути аутентифікації, загальні маршрути навігації по сайту та маршрути створення замовлення. Це покращило підтримку коду та дозволило адаптувати функції, не порушуючи загальну структуру програми. Крім того, Express.js тісно інтегрований із механізмами шаблонів, включаючи EJS, який також використовується в проекті для динамічного рендерингу сторінок. Загалом Express став ключовою технологією для побудови серверної частини, уможливлюючи впровадження RESTful API, маршрутів і логіки обробки запитів HTTP, що є основою для служби доставки продуктів у реальному часі.

2.1.4 Основні відомості про JSON WebToken

У рамках цього проекту з розробки веб-додатку для доставки їжі та продуктів бібліотека JSONWebToken (JWT) була використана для впровадження механізму автентифікації користувачів. Цей інструмент дозволяє створювати, підписувати та перевіряти веб-маркери JSON, надійний і поширений спосіб безпечної передачі облікових даних між клієнтом і сервером.

Після того, як користувач зареєструється або ввійде, на сервері генерується маркер. Він містить зашифровану інформацію про нього, включаючи його ім’я користувача. Цей маркер підписується за допомогою секретного ключа, який зберігається у файлі .env і надсилається клієнту, зазвичай у файлі cookie або в заголовку авторизації.

Усі безпечні маршрути (наприклад, /logout, /about) перевіряють наявність цього маркера. Якщо він присутній, він декодується та його дійсність перевіряється за допомогою функції jwt.verify. Якщо перевірка пройшла успішно, інформація про користувача (req.user) додається до об’єкта запиту, і обробка запиту продовжується. Якщо маркер відсутній або недійсний, сервер повертає повідомлення про помилку з відповідним статусом (401 або 400).

Таким чином, використання jsonwebtoken забезпечує надійний захист від несанкціонованого доступу до ресурсомістких або конфіденційних маршрутів. Це дозволило мені реалізувати безпечну авторизацію у моєму веб-додатку без необхідності зберігати сеанс на стороні сервера, покращуючи масштабованість і продуктивність проекту.

2.1.5 Основні відомості про Embedded JavaScript

У рамках розробки веб-додатку для доставки їжі та продуктів цей проект використовував механізм шаблонів EJS (Embedded JavaScript) для динамічної

генерації HTML-сторінок на сервері. Вибір EJS пояснюється його простотою, легкістю інтеграції з Node.js і ефективністю роботи з динамічними даними.

EJS дозволяє вбудовувати JavaScript безпосередньо в HTML-код, роблячи розмітку інтуїтивно зрозумілою та гнучкою. Це особливо корисно для створення сторінок, які вимагають динамічного відображення інформації з бази даних, такої як список продуктів, замовлення, інформація про користувача або сповіщення про стан доставки.

Однією з переваг EJS є його мінімальний поріг входу: розробнику потрібні лише базові знання HTML і JavaScript, щоб почати використовувати цю систему шаблонів. Він не вимагає складного налаштування, легко підключається до Express.js і ідеально підходить для малих і середніх програм, де швидкість розробки є важливим фактором.

У рамках цього проекту EJS дозволив реалізувати адаптивний інтерфейс, який динамічно оновлюється залежно від контексту, наприклад, пропонуючи різний вміст для авторизованих і неавторизованих користувачів, відображаючи замовлення в режимі реального часу або відображаючи сторінки окремих продуктів. Це дозволило досягти високого рівня інтерактивності інтерфейсу без складної інтерфейсної інфраструктури.

2.1.6 Основні відомості про SQLite

При реалізації веб-додатку для доставки їжі та продуктів було вирішено використовувати SQLite як основну систему керування базами даних, оскільки вона ідеально відповідає вимогам, пов’язаним із масштабністю та складністю проекту. SQLite – це вбудована реляційна СУБД, яка зберігає всі дані в одному файлі на сервері. Його виконання не потребує жодних окремих процесів чи послуг, а вимоги до ресурсів низькі. Це забезпечує швидке розгортання та просте обслуговування програми, особливо на етапах розробки, тестування та демонстрації MVP (Minimum Viable Product).

SQLite дозволяє ефективно працювати з табличною структурою даних,

що ідеально підходить для зберігання інформації про користувачів, замовлення, пункти меню, адреси доставки, історію транзакцій та інші сутності проекту. Незважаючи на свою простоту, SQLite підтримує більшість функцій стандарту SQL: транзакції, обмеження цілісності, індекси, зовнішні ключі, а також дозволяє за необхідності ефективно масштабувати структуру бази даних. Крім того, відсутність складної конфігурації або встановлення серверного програмного забезпечення, як у випадку з MySQL або PostgreSQL, є значною перевагою, яка значно прискорює розгортання та зменшує залежності.

Для взаємодії з Node.js SQLite інтегрується за допомогою простих і зручних бібліотек, таких як sqlite3, better-sqlite3 або рішень ORM, таких як Sequelize. Це полегшує реалізацію логіки доступу до даних, виконання операцій CRUD, фільтрування, сортування, обробку транзакцій і створення звітів, усе це необхідно для належного функціонування веб-служби доставки. Підтримка асинхронного виконання запитів у Node.js дає змогу ефективно працювати з SQLite навіть із збільшенням кількості запитів.

Крім того, SQLite є ідеальним рішенням на початковому етапі, оскільки дозволяє зосередитися на логіці програми, не витрачаючи час на налаштування складної інфраструктури. При необхідності проект може легко мігрувати в інші СУБД, такі як PostgreSQL або MySQL, завдяки використанню стандартного синтаксису SQL і зрозумілої логіки моделювання даних. Таким чином, SQLite забезпечує гнучку, масштабовану та практичну основу для реалізації серверної частини цієї програми. Загалом вибір SQLite обумовлений його простотою, ефективністю, хорошою інтеграцією з Node.js, повною достатністю функціоналу для поточних потреб і зручною архітектурою для подальшого розвитку програми доставки їжі.

2.1.7 Основні відомості про DotEnv

Під час реалізації веб-додатку для доставки їжі та продуктів бібліотека

DotEnv використовувалася для безпечного та зручного керування конфіденційними даними та налаштуваннями середовища. DotEnv дозволяє зберігати змінні середовища в спеціальному файлі .env, який не інтегрований у сховище завдяки .gitignore, який забезпечує безпеку конфіденційної інформації, такої як паролі, маркери доступу, шляхи до бази даних або ключі API.

Використання DotEnv значно спрощує конфігурацію проекту в різних середовищах (наприклад, розробка, тестування або виробництво) без необхідності змінювати код програми. Все, що стосується конфігурації, розміщується у зовнішньому файлі, що дозволяє легко масштабувати проект, швидко оновлювати налаштування та розгортати програму на різних серверах або хостингах з різними налаштуваннями доступу.

–У моєму проекті DotEnv використовується для зберігання таких даних, як шлях до бази даних SQLite, порт виконання сервера та інші параметри, які можуть відрізнятися залежно від середовища. Це забезпечує розділення коду та конфігурації, що є найкращою практикою в сучасній веб-розробці.

Таким чином, DotEnv служить важливим інструментом для централізованого та безпечного керування конфігурацією, підвищує гнучкість і стабільність програми, а також дозволяє дотримуватися принципів безпечної та чистої архітектури під час розробки.

2.2 Розробка архітектури веб-застосунку

Ключовим аспектом для правильної та функціональної роботи веб-застосунку є його архітектура та структура, вона відповідає за роботу між клієнтською частиною, backend частиною та їх зв’язку з базою даних. Якщо вся архітектура відповідає стандартам, вона правильно обробляє всі запити та працює коректно з усіма технологіями [5].

Структурований підхід для розробки правильної архітектури та функціонал гарантує працездатність фінального продукту для користувача,

кур’єра або ресторану.

2.2.1 Проєктування клієнт-серверної архітектури

Веб-застосунок має чітко розділену архітектуру на клієнтську (Frontend) та серверну (Backend) частини, що забезпечує гнучкість, зручність у розробці та підтримці, а також стабільну взаємодію між усіма компонентами системи.

Клієнтська частина (Frontend) відповідає за інтерфейс користувача та взаємодію з даними. Вона реалізована за допомогою HTML-шаблонізатора EJS, обраного через його простий синтаксис, схожий на HTML, що добре поєднується з Node.js і спрощує взаємодію з серверною частиною. На відміну від сучасних frontend-фреймворків на кшталт React чи Vue, EJS не потребує складного налаштування чи використання Webpack – шаблони рендеряться на сервері та передаються клієнту як готовий HTML.

Це забезпечує швидке завантаження сторінок, позитивно впливає на загальний користувацький досвід і покращує SEO-оптимізацію, оскільки пошукові системи отримують повністю сформовані HTML-сторінки. Важливою перевагою також є можливість повторного використання коду завдяки використанню часткових шаблонів (наприклад, headers, footers, partials), що спрощує структуру, покращує швидкодію та полегшує підтримку. Комбінування EJS із CSS/SCSS та функціональністю JavaScript створює адаптивну, сучасну та зручну клієнтську частину, яка коректно працює на різних пристроях.

Серверна частина (Backend) є ядром всієї бізнес-логіки і реалізована на платформі Node.js. Вона має модульну архітектуру, де кожен компонент виконує окрему функцію. Контролери (authController.js, userController.js, productController.js, orderController.js, merchantController.js) відповідають за обробку відповідних запитів та реалізацію логіки застосунку. Маршрути (authRoutes.js, productRoutes.js, ordersRoutes.js, userRoutes.js, merchantRoutes.js) забезпечують чітку структуру API, спрямовуючи запити до

відповідних контролерів.

Окрему роль відіграють middleware-компоненти, які підвищують безпеку, виконуючи додаткові перевірки, наприклад, автентифікацію чи авторизацію. Також використано сервісні модулі, такі як orderService.js, які централізують обробку складних операцій і сприяють масштабованості та зручності підтримки.

Серверна частина інтегрується з базою даних SQLite, що забезпечує ефективне збереження й обробку даних та стабільну взаємодію з клієнтською частиною. Така структура дозволяє легко розвивати проєкт, масштабувати функціональність і підтримувати високий рівень надійності.

2.2.2 Проєктування структури бази даних

База даних реалізована завдяки локальному файлу SQLite (food-delivery.db), що забезпечує ефективне зберігання і редагування всієї необхідної інформації.

У базі зберігаються усі дані про користувачів, продукти, завмовлення, ресторани, а також історія покупок та відгуки клієнтів. Використання SQLite дозволяє просто зберігати всю інформацію в одному файлі, що полегшує розгортання та тестування застосунку локально, без необхідності налаштування окремого серверу баз даних, що забезпечує зручне та ефективне зберігання всієї необхідної інформації. У базі зберігаються дані про користувачів, продукти, замовлення, ресторани, а також історія покупок і відгуки клієнтів. Використання SQLite дозволяє зберігати всю інформацію в одному файлі, що полегшує розгортання і тестування застосунку без необхідності налаштовувати окремий сервер бази даних.

Структура бази даних розроблена таким чином, щоб забезпечувати швидкий та простий доступ до необхідних даних і зручну роботу з ними завдяки серверній частині. Завдяки цьому проект ефективно обробляє запити користувача, оновлює статусі замовлень і контролює контролює асортимент

продуктів і зберігає відгуки, що покращує загальний користувацький досвід.

Також для початкового наповнення бази даних використовувалися зовнішні джерела тестових даних, що допомагають швидко запустити застосунок і перевірити його функціональність, без ручного введення інформації. Такий підхід прискорює розробку і тестування, забезпечуючи більш гнучке та ефективне управління даними на початкових і кінцевих етапах роботи.

2.2.3 Організація взаємодії компонентів системи

В проєкті клієнтська частина (Frontend), серверна частина (Backend) та база даних SQLite, тісно взаємодіють між собою забезпечуючи коректну роботу застосунку і створюючи ефективний і зручний користувацький досвід. Клієнтська частина відповідає за інтерфейс користувача і відправляє HTTP-запити до серверної частини для отримання або надсилання даних , таких як оформлення замовлення , авторизацію чи перегляд каталогу товарів. Серверна частина в свою чергу приймає ці запити, обробляючи їх через маршрути і контролери, завдяки яким виконується бізнес-логіка застосунку. Middleware забезпечує додаткову перевірку безпеки і валідації даних. Для зберігання усієї інформації в проєкті і отримання даних сервер зберігає до бази даних SQLite, де зберігається уся інформація про користувача, товари, замовлення та інше. Після обробки запиту сервер формує відповідь у вигляді HTML-сторінки або JSON та відправляє сформовану відповідь назад до користувача, який оновлює інтерфейс користувача.

На схемі на рисунку 2.1 – зображено основні компоненти архітектури проекту – клієнтську частину, серверну частину та базу даних, а також їхню взаємодію.

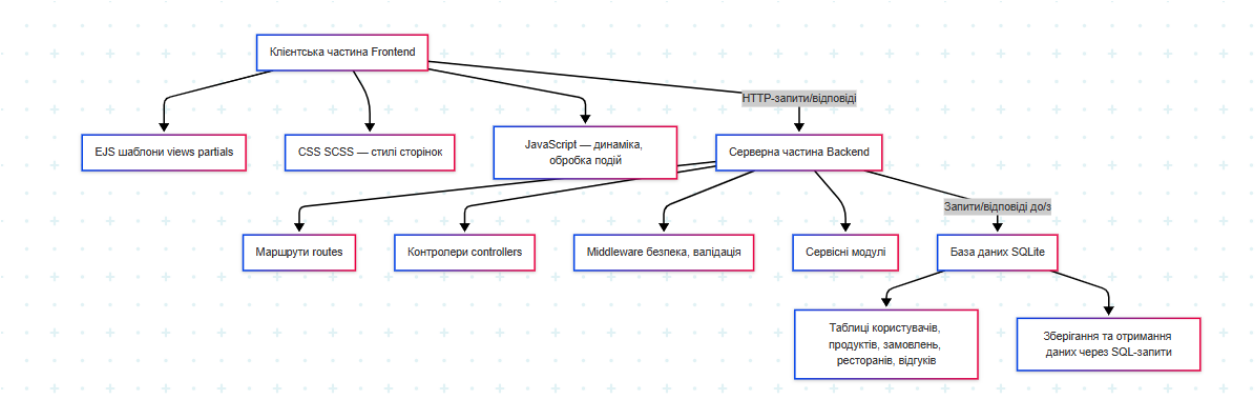

Рис 2.1 – Архітектура веб-застосунку та взаємодія основних компонентів

2.3 Проектування інтерфейсу користувача

Головним пріоритетом у розробці проєкту був дизайн, адже саме він формує перше враження про продукт і виступає своєрідним «квитком» до якісного користувацького досвіду – що було досліджено в тезах [8].

2.3.1 Розробка концепції UX/UI дизайну

З перших етапів роботи було проведено аналіз основних конкурентів у сфері доставки продуктів та їжі. Під час дослідження виявлено, що:

- деякі сайти конкурентів перенасичені функціоналом і складні для розуміння пересічним користувачем;
- інші мають гарно збалансований дизайн, але основна увага у них приділяється мобільному застосунку, а не веб-версії.

У сучасних умовах це відкриває вільну нішу для веб-застосунка зі зрозумілим, мінімалістичним і комфортним дизайном, орієнтованим не на агресивні продажі, а на зручність користувача. Жодних нав'язливих банерів типу «купи», «оплати» чи «доставимо» – тільки логічна, чітка і приємна взаємодія [6].

Під час створення інтерфейсу я орієнтувався на власний користувацький досвід – прагнув створити простір, у якому головний – це користувач. Мінімум інформаційного шуму, відсутність реклами й навантаження, а натомість –

інтуїтивний процес замовлення і очікування.

Для досягнення цієї мети в дизайні були використані кольори: червоний, помаранчевий, жовтий, а також білий і чорний. Вибір кожного з них обґрунтований психологією кольору в сфері обслуговування, доставки та харчування:

- червоний колір – викликає асоціації енергії, збудження, апетиту та терміновості, що у харчовій сфері стимулює апетит та привертає увагу;
- жовтий колір – викликає асоціації радості, тепла, оптимізму та швидкість обслуговування, що у харчовій сфері асоціюється з доступністю та легкістю;
- помаранчевий колір – викликає асоціації ентузіазму, свіжості, смачності та енергія, що у харчовій сфері викликає відчуття приємності та тепла.

Ці кольори належать до теплого спектру, мають природну синергію, посилюють один одного та гармонійно поєднуються. Вони не перевантажують зір, забезпечують легке сприйняття інтерфейсу та створюють приємну, динамічну атмосферу. На кольоровій палітрі яку представлено на рисунку 2.2 можна чітко побачити кольори та їх відтінки які дуже добре поєднуються між собою та доповнюють один одного.

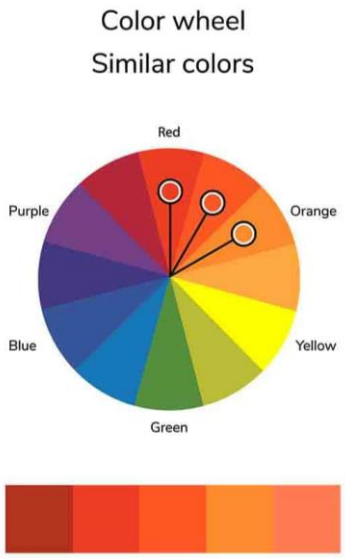

Рисунок 2.2 – Демонстрація суміжності кольорів на палітрі

Слід зазначити, що надмірне використання жовтого та червоного може викликати асоціації з рекламною агресією (як у супермаркетах), тому ці кольори виконують допоміжну роль. Основним кольором фону був обраний помаранчевий із блюром або градієнтом. Такий підхід дозволяє зберігати візуальний баланс – акценти залишаються на елементах інтерфейсу (кнопки, блоки), які чітко виділяються, не перевантажуючи око.

2.3.2 Аналіз UX/UI дизайну успішних брендів

Під час роботи з дизайном натхнення було на основі успішних брендів в сфері харчування, брав до уваги такі бренди McDonald’s, Burger King, KFC, Fanta, Lay’s. Ці усі бренди також обрали червоний, жовтий і помаранчевий не випадково. Вони усі розуміли, що ці кольори:

- підвищують апетитту;
- викликають емоційну реакцію;
- формують потяг до продукту;
- підсилюють інтерес до їжі.

Їхній дизайн створено таким чином, щоб миттєво взаємодіяти з емоціями користувача, викликаючи асоціації з задоволенням, швидкістю та доступністю.

В своїй роботі прагнув досягти наступного:

- передати настрій смаку та задоволення – через теплі відтінки;
- використати динамічну палітру, яка не просто красиво виглядає, а також спонукає до дії: купити, спробувати, замовити, насититись;
- побудувати влучний стиль, який не тільки влучно потрапляє в бажання користувача, а також запам’ятовується і залишає гарне враження в підсвідомості.

Дизайн логотипів на рисунку 2.3, які включать в себе ці кольори.

Рисунок 2.3 – Дизайн логотипів цих брендів

Найбільш запам'ятовуючим дизайном пакування для споживачів – є пакування McDonald’s представлене на рисунку 2.4. Кожне пакування має свій неповторний стиль який впізнаваний по всьому світу.

Рисунок 2.4 – Дизайн продукції та упаковок від McDonald’s

Продукція Fanta також має гарний збалансований дизайн в котрому є почуття свіжості та насолоди, рисунок 2.5 добре це демонструє

Рис 2.5 – Дизайн продукції та логотипу Fanta

Дизайн цих брендів – не просто вдалий, він набув вже культового значення та статусу. Багато з їх рішень залишаються незмінними протягом десятиліть. Завдяки постійній присутності на ринку та великій кількості їх продукції, клієнти не лише звикли до дизайну та образів, що вже важко уявити альтернативний вигляду цієї продукції та логотипів.

Я приділив дизайну особливу увагу, адже він не лише створює естетику, а й підсилює впізнаваність, довіру та продажі. Мій дизайн втілює цілі, яких я прагнув досягти в рамках цієї роботи

2.3.3 Формування візуальної ідентичності застосунку

Основними перевагами мого дизайну ефективна взаємодія користувача та працівника з інтерфейсом . усі дизайнерські рішення спрямовані на зручність та швидкодію та інтуїтивне створення замовлення, а також його контроль і доставка.

Однією з найважливіших складових є дизайн трекінгу заказів, він відображає не лише кроки доставки а також і легкість та швидкість доставки завдяки своєму дизайну, має витончене значення для кінцевого користувача.

Окрему увагу приділено спливаючи вікнам реєстрації та гарні форми звернення мають такий дизайн, що не лише органічно вписуються до загального дизайну а й доповнюють його, створюючи приємне враження від використання застосунку. Враження від користування має вирішальне значення для користування різними сервісами серед користувачів.

Головними фішками я вважаю:

- підсвітка функціональних кнопок;
- інші кольори для кнопок, які переходять до інших вкладок та меню;
- трекінг сторінка, яка має незабутній дизайн;
- гарно підлаштовані форми та впливаючі вікна, які відповідають загальному дизайну.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ

На основі розробленої архітектури та дизайну вебзастосунку для доставки їжі здійснено його програмну реалізацію. Вебзастосунок створено з використанням таких основних технологій:

- Frontend – через шаблони EJS, інтегровані з Node.js, CSS, JavaScript;
- Backend – Node.js з фреймворком Express;
- база даних – SQLite;
- контроль доступу – middleware-файли;
- обробка форм та API – REST-запити.

3.1 Організація структури

Організація структури веб-застосунку є ключовим етапом для забезпечення стабільної, масштабованої та зручної в підтримці системи. Чітке розмежування відповідальностей між окремими компонентами покращує керованість кодом, пришвидшує розробку та спрощує інтеграцію нових функцій. Загальну схему структури веб-застосунку зображено на рисунку 3.1.

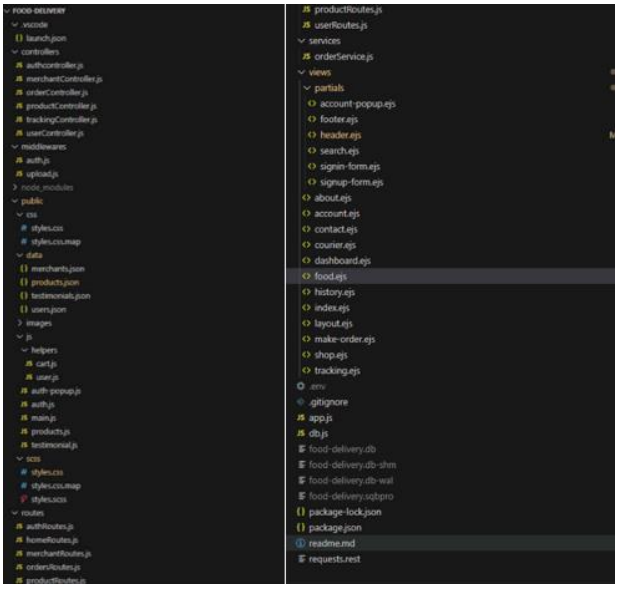

Рисунок 3.1 – Організація структури веб-застосунку

Структура проєкту поділена на логічні модулі, що забезпечують розподіл відповідальностей і полегшують підтримку коду:

- controllers/ – містить контролери, які обробляють запити користувачів та реалізують основну бізнес-логіку;
- middlewares/ – проміжне програмне забезпечення, яке виконує перевірку доступу та обробку запитів;
- routes/ – маршрутизація HTTP-запитів до відповідних контролерів;
- views/ – шаблони для відображення інтерфейсу, реалізовані з використанням EJS. Включають в себе усі функціональні сторінки;
- services/ – додаткові сервіси, що інкапсулюють складні бізнес-логіки, наприклад orderService.js для обробки замовлень;
- helpers/ – допоміжні скрипти для фронтенду, які відповідають за динамічну поведінку сторінок та інтерактивність;
- data/ – локальні JSON-файли з тестовими даними, які використовувалися на етапі розробки;
- scss/ та css/ – стилі, написані з використанням SCSS, що компілюються у CSS-файли для застосування в інтерфейсі;
- public/images/ – директорія зі статичними зображеннями товарів, фонами та іншими ресурсами.

3.2 Реалізація серверної частини застосунку

Для детального опису було обрано повну реалізацію однієї з ключових функцій вебзастосунку – роботу з обліковим записом користувача. До цієї функціональності залучено п’ять основних компонентів: серверна логіка, маршрутизація, перевірка автентифікації, клієнтський скрипт та шаблон відображення сторінки. Назви цих файлів в проекті userController.js, user.js, userRoutes.js, account.ejs, auth.js.

3.2.1 Опис контролерів на базі userController.js

У веб-розробці контролери є ключовим компонентом серверної сторони програми. Вони реалізують логіку обробки HTTP-запитів, взаємодії з базою даних, перевірки прав доступу та генерації відповідей у відповідному форматі, як правило, у форматі JSON або за допомогою шаблонів. Контролери реалізовані відповідно до шаблону MVC (Model–View–Controller), де контролер діє як посередник між моделлю (даними) і представленням (інтерфейс користувача).

Кожен контролер відповідає за певну логіку, пов’язану з відповідним ресурсом, наприклад: користувачі, замовлення, продукти тощо. У цьому проекті призначено окремий контролер керування користувачами: userController.js.

Опис файлу userController.js:

Цей файл є частиною каталогу контролерів і реалізує повний набір функцій для адміністрування користувачів і отримання історії їхніх команд. Усі операції виконуються через підключену до проекту базу даних SQLite, а захист паролем забезпечує бібліотека bcryptjs.

Функція лістингу 3.1, дозволяє отримати як повний перелік усіх зареєстрованих користувачів, так і дані про конкретного користувача за його унікальним ідентифікатором. Для відображення додаткової інформації, зокрема ролі користувача (наприклад, клієнт, адміністратор тощо), до таблиці users приєднується таблиця roles. Отримати повний список користувачів може лише адміністратор. Якщо ж запитується користувач за ID, але такий не знайдений – повертається помилка 404.

Лістинг 3.1 – Вибірка користувачів і перевірка на існування

```
// Отримання всіх користувачів із приєднанням ролей
const users = db.prepare(`
  SELECT u.id, u.name, u.email, u.phone, r.name AS role,
u.created_at
  FROM users u
```

```
  JOIN roles r ON u.role_id = r.id
`).all();

// Отримання одного користувача за ID
const user = db.prepare(`
  SELECT u.id, u.name, u.email, u.phone, r.name AS role,
u.created_at
  FROM users u
  JOIN roles r ON u.role_id = r.id
  WHERE u.id = ?
`).get(req.params.id);

if (!user) {
  return res.status(404).json({ error: 'User not found' });
}
```

Функція updateUser ,яка представлена в лістингку 3.2 – реалізує можливість редагування інформації про користувача, включаючи ім’я, email, номер телефону, роль та пароль. Вона побудована з урахуванням гнучкої логіки: оновлюються лише ті поля, які були передані у тілі запиту (req.body). Перед оновленням пароля виконується його хешування через bcrypt для забезпечення безпечного зберігання.У випадку, якщо роль користувача передана у вигляді назви, вона перевіряється в базі — якщо такої ролі не існує, повертається помилка. Якщо жодне з полів не передане, сервер також повертає помилку. Після цього формується динамічний SQL-запит із оновленням лише зазначених даних.

Лістинг 3.2 – Оновлення користувача з перевіркою ролі та хешуванням пароля

```
updateUser: async (req, res) => {
    const { name, email, phone, role, password } = req.body;
    const userId = req.params.id;

    const updates = [];
    const params = [];

    if (name) {
        updates.push('name = ?');
        params.push(name);
    }

    if (email) {
        updates.push('email = ?');
        params.push(email);
```

```
    }

    if (phone) {
        updates.push('phone = ?');
        params.push(phone);
    }

    if (role) {
        const roleRow = db.prepare('SELECT id FROM roles WHERE
name = ?').get(role);
        if (!roleRow) return res.status(400).json({ error:
'Invalid role' });

        updates.push('role_id = ?');
        params.push(roleRow.id);
    }

    if (password) {
        const salt = await bcrypt.genSalt(10);
        const passwordHash = await bcrypt.hash(password, salt);

        updates.push('password_hash = ?');
        params.push(passwordHash);
    }

    if (updates.length === 0) {
        return res.status(400).json({ error: 'No updates
provided' });
    }

    params.push(userId);

    const stmt = db.prepare(`
        UPDATE users
        SET ${updates.join(', ')}
        WHERE id = ?
    `);

    stmt.run(...params);
    res.json({ message: 'User updated successfully' });
}
```

deleteUser – функція дозволяє видалити користувача з бази за його ID. У разі, якщо користувача не знайдено, повертається помилка з відповідним повідомленням.

getOrderHistory (лістинг 3.3) – повертає повну історію замовлень для поточного авторизованого користувача. Інформація включає: ідентифікатор

замовлення, статус, адресу доставки, дату створення, а також вкладений список товарів у межах кожного замовлення. До кожного елемента додається також інформація про продавця (merchant).

Лістинг 3.3 – Запит історії замовлень з групуванням товарів

```
SELECT o.id, o.status, ...,
json_group_array(json_object(
  'id', oi.id,
  'product_id', oi.product_id,
  ...
)) AS items
FROM orders o
LEFT JOIN ...
GROUP BY o.id
```

Функція представлена у виді парсингу файлу orders.forEach(order =>{ order.items = JSON.parse(order.items);}) – перетворю текстове JSON-представлення списку товарів у кожному замовленні (order.items) на об’єкт JavaScript. Це необхідно для подальшої роботи з даними як з масивом.

Контролер користувача забезпечує ефективне керування обліковими записами адміністратора та користувачів. Завдяки централізованій логічній обробці ці функції легко протестувати, оновити та розгорнути пізніше.

3.2.2 Опис раутсів на базі userRoutes.js

Раутси (Routes) у вебзастосунку — це маршрути, які визначають, як сервер реагує на HTTP-запити за певними адресами (URL). Вони є невіддільною частиною архітектури будь-якого вебзастосунку, особливо при використанні таких фреймворків, як Express.js у Node.js.

Раутси відповідають за зв’язок між клієнтськими запитами (з браузера чи API) та серверною логікою, яка реалізована в контролерах. У межах архітектури MVC маршрутизація — це „дороги“, які ведуть до контролерів.

Імпорт залежностей – у файли імпортуються додаткові модулі до роботи маршрутизатора: Express – до створення маршрутизатора; проміжні

обробники (middleware) auth, roleCheck та requireAuth для контролю доступу та аутентифікації користувачів; контролери userController та authcontroller, що відповідають функціям роботи з користувачами та авторизацією; і об'єкт db для взаємодії з базою даних SQLite (лістинг 3.4).

Лістинг 3.4 – Імпорт залежностей

```
const express = require('express');
const { auth, roleCheck, requireAuth } =
require('../middlewares/auth');
const userController = require('../controllers/userController');
        const { getCurrentUser, updateCurrentUser } =
require('../controllers/authcontroller');
const db = require('../db');
```

Ініціалізація маршрутизатора створюється за допомогою такої функції – const router = express.Router();router.use(auth). Де новий екземпляр маршрутизатора Express, проходить на всьому маршрутизаторі встановлює автентифікація проміжного програмного забезпечення, що означає, що всі маршрути, зазначені у цьому файлі, будуть доступні лише для автентифікованих користувачів.

Маршрутизація для адміністратора – маршрути лістинг 3.5, що починаються з /api/users, призначені для адміністратора. Вони захищені проміжним програмним забезпеченням roleCheck, який перевіряє, що роль користувача є 'admin'. За допомогою цих маршрутів адміністратор може отримати список усіх користувачів, інформацію про конкретного користувача за ID, оновити дані користувача (включаючи ім'ям, емейлом, телефоном, роликом і паролем — пароль захищений у контролері), а також видалити користувача з бази даних.

Лістинг 3.5 – Маршрутизація для адміністратора

```
router.get('/api/users', roleCheck('admin'),
userController.getAllUsers);
router.get('/api/users/:id', roleCheck('admin'),
userController.getUserById);
router.put('/api/users/:id', roleCheck('admin'),
userController.updateUser);
```

```
router.delete('/api/users/:id',roleCheck('admin'),userController
.deleteUser);
```

Маршрутизація для поточного авторизованого користувача – для звичайних користувачів передбачені маршрути, які переглядають свій профіль на сторінці HTML (/my/account) — тут виконується SQL-запит, який отримує персональні дані користувача разом з його роликом і передає їх у шаблонизатор для відображення.Також користувач може оновити особисті дані через PATCH-запит на /api/my/account, що обробляє функцію updateCurrentUser з authcontroller. Для перегляду історії замовлень користувачі можуть отримати JSON-дані через API-запит /api/my/orders-history, який виконує контролер userController. Візуальне відображення історії замовлень доступне на запит /my/orders-history. Тут також виконується SQL-запит для отримання даних, а потім сторінка рендерується через шаблонизатор (лістинг 3.6).

Лістинг 3.6 – Маршрутизація для поточного авторизованого користувача

```
router.get('/my/account', requireAuth, (req, res) => {
    const user = db.prepare(`
        SELECT u.id, u.name, u.email, u.phone, r.name AS role,
u.created_at
        FROM users u
        JOIN roles r ON u.role_id = r.id
        WHERE u.id = ?
    `).get(req.user.id);

    res.render('account', { user: user });
});

router.patch('/api/my/account', requireAuth, updateCurrentUser);

router.get('/api/my/orders-history',requireAuth,
userController.getOrderHistory);

router.get('/my/orders-history', requireAuth, (req, res) => {
    const user = db.prepare(`
        SELECT u.id, u.name, u.email, u.phone, r.name AS role,
u.created_at
        FROM users u
        JOIN roles r ON u.role_id = r.id
        WHERE u.id = ?
    `).get(req.user.id);
```

```
    res.render('history', { user: user });
});
```

Експорт маршрутизатора module.exports = router – ця функція експортує об'єкт router, щоб його можна було підключити в основному файлі сервера. Це дозволяє розділяти маршрути по модулях і підтримувати чисту структуру проєкту.

Після опису всіх маршрутів маршрутизатор експортується для підключення до головного файлу сервера, що дозволяє інтегрувати всі певні маршрути до основного ресурсу.

3.3 Реалізація клієнтської частини застосунку

Клієнтська частина веб-застосунку створена з акцентом на зручність користувача, швидкий інтерфейс і логічну структуру. Для її побудови використано шаблонізатор EJS, який дозволяє формувати динамічні сторінки з урахуванням даних, отриманих із сервера. Уся логіка взаємодії з інтерфейсом реалізована на JavaScript, що забезпечує гнучкість та інтерактивність без перезавантаження сторінки. Стилізація виконана з використанням CSS/SCSS, що дозволяє створювати адаптивний і візуально привабливий інтерфейс.

У цій частині розглянуто ключові модулі клієнтської логіки, включно з хелперами, механізмами автентифікації, обробкою форм та структурою особистого кабінету користувача.

3.3.1 Опис хелпера на базі user.js

Хелпер (helper) – це допоміжна функція або набір функцій, які спрощують виконання певних повторюваних завдань у коді.

Об'єкт User - Цей модуль містить об’єкт User, який реалізує функції для роботи з даними користувача в localStorage браузера. Це зручний спосіб зберігати інформацію про поточного користувача на стороні клієнта між

сесіями.

1. getToken – ця функція отримує маркер автентифікації користувача. Він намагається витягти об’єкт користувача з localStorage, проаналізувати його з формату JSON і повернути властивість маркера. Якщо немає даних або маркер відсутній, функція повертає значення null. Це дозволяє швидко перевірити, чи авторизований користувач.

2. getInfo – функція повертає повний об’єкт користувача, що зберігається в localStorage. Якщо інформація відсутня, повертається значення null. Він використовується для отримання збережених даних про користувача, таких як ім’я, електронна адреса, роль тощо.

3. setInfo – ця функція приймає об’єкт userData і зберігає його в localStorage як рядок JSON під ключем «user». Він використовується для оновлення або налаштування інформації про користувача після входу в систему або зміни даних.

4. clear – ця функція видаляє інформацію користувача з localStorage, фактично «виходячи» з системи на стороні клієнта.

Лістинг 3.7 дозволяє централізовано керувати локальним зберіганням даних користувача, що спрощує автентифікацію та керування профілями на клієнті.

Лістинг 3.7 – файлу user.js

```
const User = {
    getToken() {
        const userData =
JSON.parse(localStorage.getItem('user'));
        return userData?.token || null;
    },
    getInfo() {
        return JSON.parse(localStorage.getItem('user')) || null;
    },

    setInfo(userData) {
        localStorage.setItem('user', JSON.stringify(userData));
    },

    clear() {
        localStorage.removeItem('user');
    }
```

```
};
```

3.3.2 Опис аутентифікації на базі auth.js

Імпорт та ініціалізація елементів за допомгою лістингу 3.8, де із DOM отримуються посилання на кнопки та елементи для входу і реєстрації: кнопка входу (.btn.signin), попапи для входу (#signin-popup) та реєстрації (#signup-popup), кнопки закриття (.close-popup), а також кнопки для перемикання між формами (#show-signup і #show-signin).

Лістинг 3.8 – Імпорт та ініціалізація елементів

```
const signInBtn = document.querySelector('.btn.signin');
const signInPopup = document.getElementById('signin-popup');
const signUpPopup = document.getElementById('signup-popup');
const closeBtns = document.querySelectorAll('.close-popup');
const showSignUp = document.getElementById('show-signup');
const showSignIn = document.getElementById('show-signin');
```

Лістинг 3.9 – відповідає за відкриття та закриття модальних вікон (попапів) входу користувача. При натисканні на кнопку входу (якщо вона існує на сторінці) відкривається відповідний попап, який відображається зі стилем flex, а прокрутка сторінки блокується, щоб зосередити увагу користувача на вікні входу.Для зручності всі кнопки закриття попапів мають обробники, які приховують усі активні попапи та відновлюють прокрутку сторінки. Такий підхід забезпечує зручну та передбачувану взаємодію з інтерфейсом авторизації.

Лістинг 3.9 – Відкриття та закриття попапів авторизації

```
// Відкриття попапу входу
if (signInBtn && signInPopup) {
    signInBtn.addEventListener('click', (e) => {
        e.preventDefault();
        signInPopup.style.display = 'flex';
        document.body.style.overflow = 'hidden';
    });
}

// Закриття всіх попапів
```

```
closeBtns.forEach(btn => {
    btn.addEventListener('click', () => {
        document.querySelectorAll('.auth-popup').forEach(popup
=> {
            popup.style.display = 'none';
        });
        document.body.style.overflow = '';
    });
});
```

Перемикання між формами входу та реєстрації – якщо кнопки для перемикання існують, додаються обробники, які ховають один попап і показують інший (лістинг 3.10).

Лістинг 3.10 – Перемикання між формами входу та реєстрації

```
if (showSignUp && showSignIn) {
    showSignUp.addEventListener('click', (e) => {
        e.preventDefault();
        signInPopup.style.display = 'none';
        signUpPopup.style.display = 'flex';
    });

    showSignIn.addEventListener('click', (e) => {
        e.preventDefault();
        signUpPopup.style.display = 'none';
        signInPopup.style.display = 'flex';
    });
}
```

Закриття попапу кліком поза контентом – додається обробник кліку по документу, який закриває попап, якщо клік було зроблено на фоні попапу, і знову дозволяє прокрутку (лістинг 3.11).

Лістинг 3.11 – Закриття попапу кліком поза контентом

```
document.addEventListener('click', (e) => {
    if (e.target.classList.contains('auth-popup')) {
        e.target.style.display = 'none';
        document.body.style.overflow = '';
    }
});
```

Обробка відправки форм – функція handleFormSubmit приймає ID форми та адресу ендпойнта API, додає обробник submit до форми.Збирає дані

форми.Якщо це реєстрація, перевіряє співпадіння паролів.Відправляє POST-запит із JSON-тілом.При успіху перенаправляє користувача.При помилці показує повідомлення(лістинг 3.12).

Лістинг 3.12 – Обробка відправки форм

```
const handleFormSubmit = async (formId, endpoint) => {
    const form = document.getElementById(formId);
    if (!form) return;

    form.addEventListener('submit', async (e) => {
        e.preventDefault();

        const formData = new FormData(form);
        const data = Object.fromEntries(formData.entries());

        if (formId === 'signup-form-element') {
            const confirmPassword =
document.getElementById('signup-confirm-password').value;
            if (data.password !== confirmPassword) {
                alert('Паролі не співпадають!');
                return;
            }
        }

        try {
            const response = await fetch(endpoint, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(data)
            });

            const result = await response.json();

            if (response.ok) {
                window.location.href = '/';
            } else {
                alert(result.error || 'Сталася помилка');
            }
        } catch (error) {
            console.error('Error:', error);
            alert('Сталася помилка. Спробуйте ще раз.');
        }
    });
};
```

Ініціалізація обробників форм за допомгою коду

handleFormSubmit('signinformelement','http://localhost:3000/api/auth/login');handleFormSubmit('signup-form-element', 'http://localhost:3000/api/auth/register') – запускаються функції обробки для форм входу і реєстрації з відповідними URL API.

3.3.3 Опис сторінок на базі account.ejs

Лістинг 3.13 – це фрагмент EJS описує структуру особистого кабінету користувача на веб-сайті. Інтерфейс складається з трьох основних вкладок: "Профіль", "Безпека" та "Адреси". У вкладці "Профіль" користувач може переглядати та редагувати особисті дані, зокрема ім’я та номер телефону. У розділі "Безпека" реалізовано зміну пароля. Вкладка "Адреси" дозволяє додавати, переглядати та зберігати кілька адрес доставки. Вся структура розроблена з використанням шаблонів EJS, що дозволяє динамічно відображати дані користувача.

Лістинг 3.13 – Описує структури особистого кабінету

```
<div class="account-container">
    <div class="account-header">
        <h1><i class="bx bx-user-circle"></i> Особистий
кабінет</h1>
        <div class="account-tabs">
            <button class="tab active" data-
tab="profile">Профіль</button>
            <button class="tab" data-
tab="security">Безпека</button>
            <button class="tab" data-
tab="addresses">Адреси</button>
        </div>
    </div>
```

Скрипт на лістингу 3.14 забезпечує динамічну взаємодію у вкладках особистого кабінету користувача: при кліку на одну з вкладок (наприклад, «Профіль», «Безпека» чи «Адреси») автоматично приховуються всі інші розділи, а обраний стає активним і відображається з відповідним стилем. Це дозволяє користувачеві зручно перемикатися між секціями без

перезавантаження сторінки. Скрипт реалізовано за допомогою стандартних методів JavaScript, таких як querySelector, forEach і classList, що забезпечує легку та швидку роботу інтерфейсу. Такий підхід сприяє покращенню UX та робить взаємодію з особистим кабінетом інтуїтивно зрозумілою. Крім того, він легко масштабується під додавання нових вкладок або зміну структури сторінки.

Лістинг 3.14 – Опис скрипту роботи вкладок

```
    document.addEventListener('DOMContentLoaded', () => {
        document.querySelectorAll('.tab').forEach(tab => {
            tab.addEventListener('click', function () {
                document.querySelectorAll('.tab').forEach(t =>
t.classList.remove('active'));
                document.querySelectorAll('.tab-
content').forEach(c => c.classList.remove('active'));

                this.classList.add('active');
                document.getElementById(`${this.dataset.tab}-
tab`).classList.add('active');
            });
        });
```

4 ТЕСТУВАННЯ ФУНКЦІОНАЛЬНОСТІ ВЕБ-ЗАСТОСУНКУ

Для початку роботи з веб-застосунком потрібно потрібно завантажити на комп’ютер програмне забезпечення Visual Studio Code та Node.js. Відкрити папку з усіма файлами та почати роботу за допомогою натискання клавіші F5, в будь якому файлі js. Далі користувач відкриває будь-який браузер та прописує в пошуковій системі це посилання http://localhost:3000. Після цього користувач потрапляє до головний сторінки веб-застосунку (рисунок 4.1).

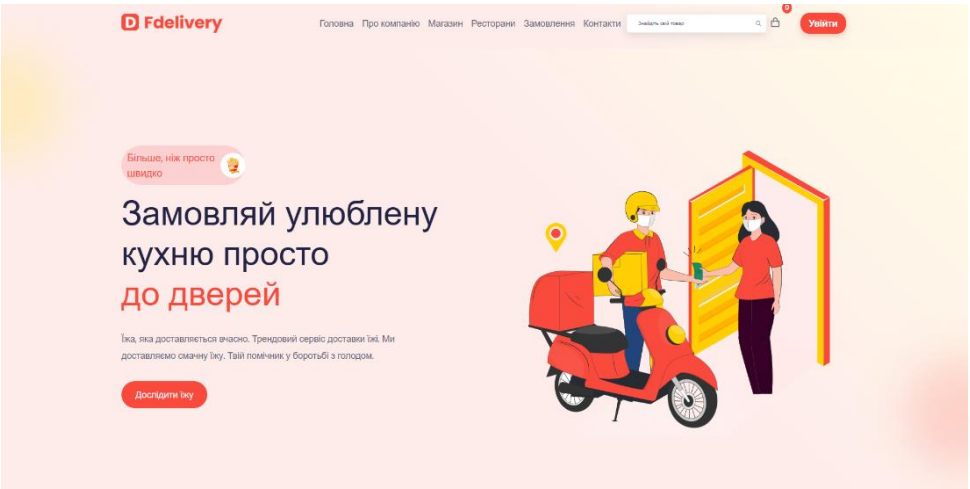

Рисунок 4.1 – Початкова сторінка проекту

Сторінка має невеликий лендінг, показаний нижче у вигляді скріншотів програми на рисунку 4.2.

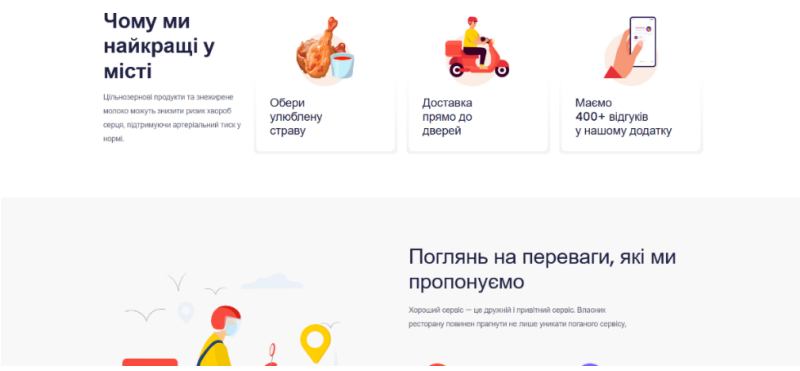

Рисунок 4.2 – Лендінг початкової сторінки

Інформаційна частина головної сторінки застосунку показана на рисунку 4.3.

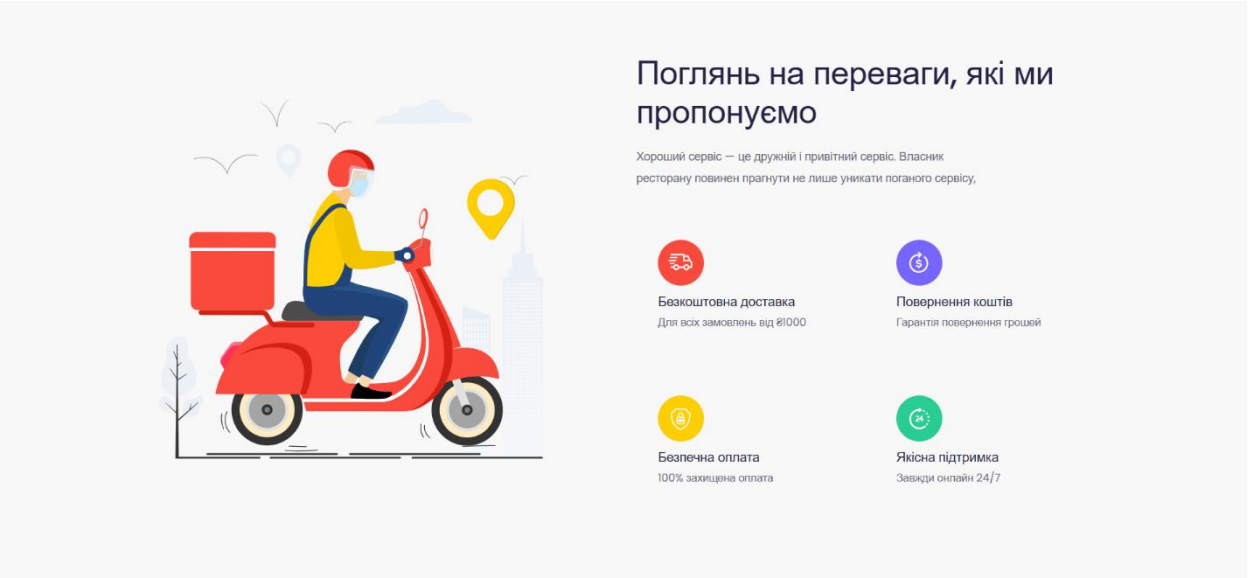

Рисунок 4.3 – Лендінг початкової сторінки

Лендінг сторінки який представлений до цього моменту має лише візуальне значення окрім кнопки Дослідити їжу, яка нас переносить до сторінки ресторани, також інформація зі сторінки ресторани , є в лендінгу першої сторінки, який представлений на рисунку 4.4. Ця частина сторінка має сортування їжі та відповідні кнопки для сортування, також користувач має можливість свайпу, при перетягуванні екрану.

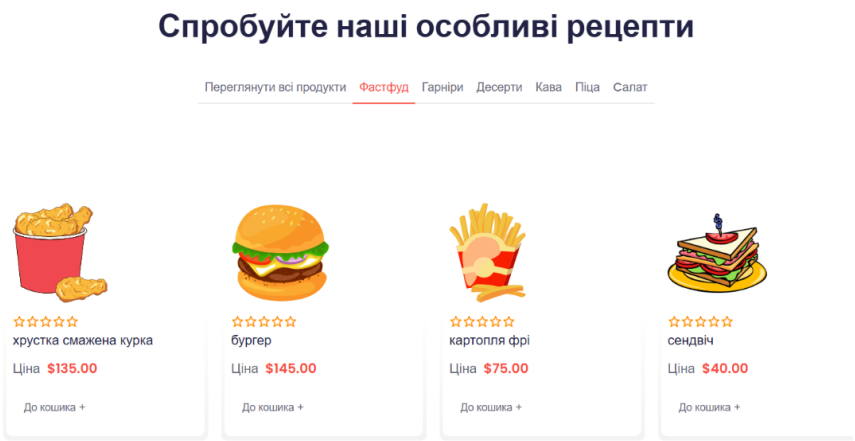

Рисунок 4.4 – Лендінг початкової сторінки

Далі представлена частина лендінгу першої сторінки на рисунку 4.5, де можна представити розробників, їх цитати, для заохочення користувачів доєднатися до команди, її функціонування, в натискані на окремого розробника и слайдиться його цитата та інформаці про нього

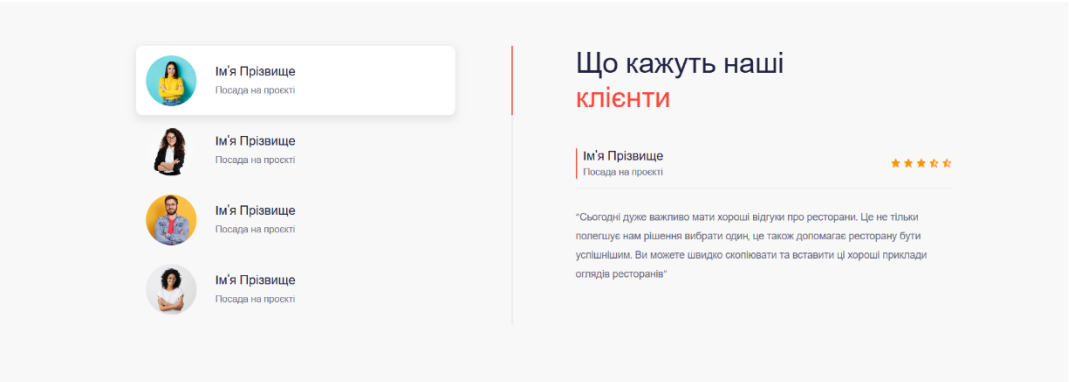

Рисунок 4.5 – Лендінг початкової сторінки

Далі користувач бачить хедер сторінки на рисунку 4.6, де є кілька функціональних кнопок, які посилають до окремих сторінок та є місце для розміщення посилань через іконки будь-яких соціальних мереж, або месенджерів.

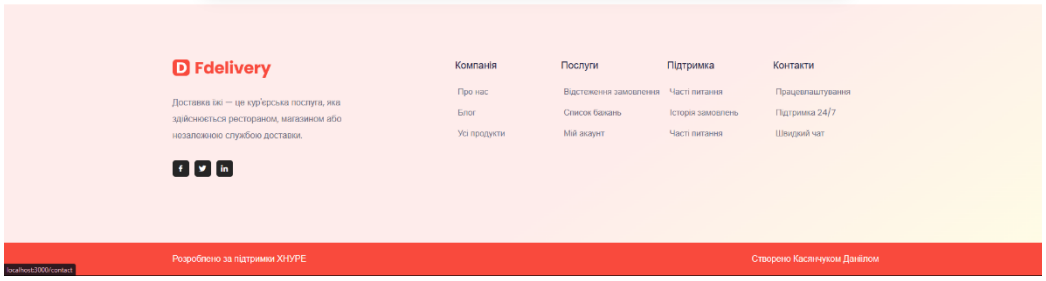

Рисунок 4.6 – Кінець сторінки, дизайн хедеру

Користувач має можливість увійти та створити свій аккаунт, завдяки кнопці Увійти, яка розташована в хедері сторінки і показана одразу при вході на сайт. Користувачу варто зареєструватися, для отримання повного функціоналу веб-застосунку. Нижче на рисунку 4.7, продемонстровано форми для входу та реєстрації.

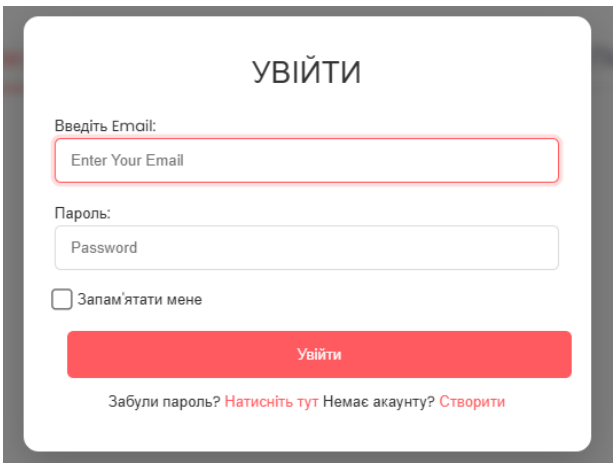

Рисунок 4.7 – Форма входу в аккаунт

Для переходу на форму реєстрації користувач, повинен натиснути на кнопку створити, він перейде до форми реєстрації представлену на рисунку 4.8, де зможе заповнити свої дані , і далі увійти до свого аккаунту

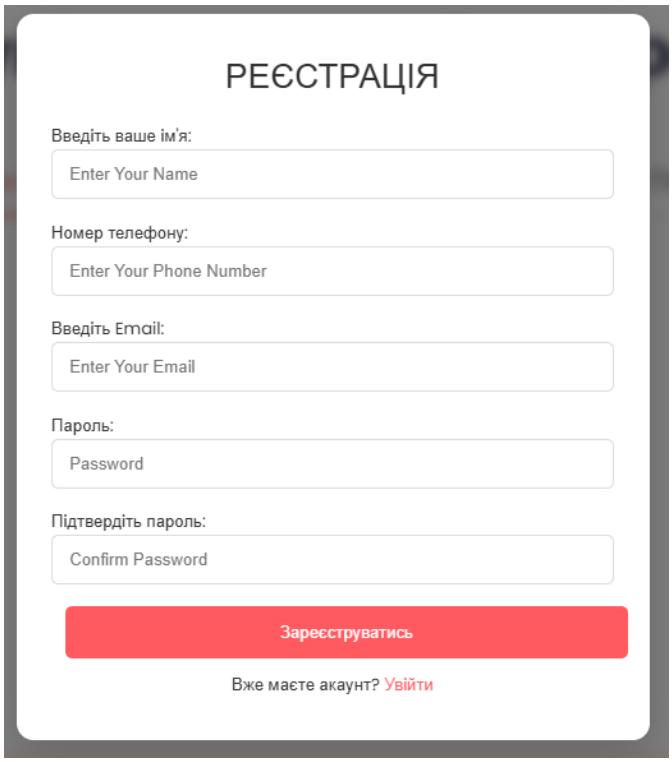

Рисунок 4.8 – Форма реєстрації в аккаунт

Після входу до аккаунту, користувач має дві додаткових сторінки. На першій сторінці особистий кабінет, який уявляє собою інформацію про вас, яку ви можете змінювати та редагувати, також є дві додаткові сторінки

«Безпека» – де є можливість змінити свій пароль та «Адреси» – де можна змінити свою вказану адресу (рисунок 4.9).

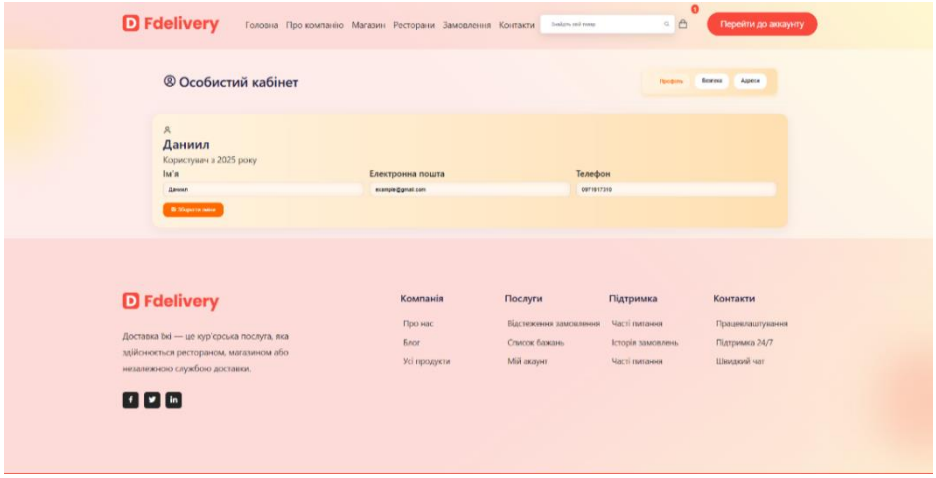

Рисунок 4.9 – Сторінка особистого кабінету

На другій сторінці користувач потрапляє до історії своїх замовлень, де він може відслідкувати свої замовленням де можете сортувати свої замовлення за статусом, періодом та часом його створенням (рисунок 4.10).

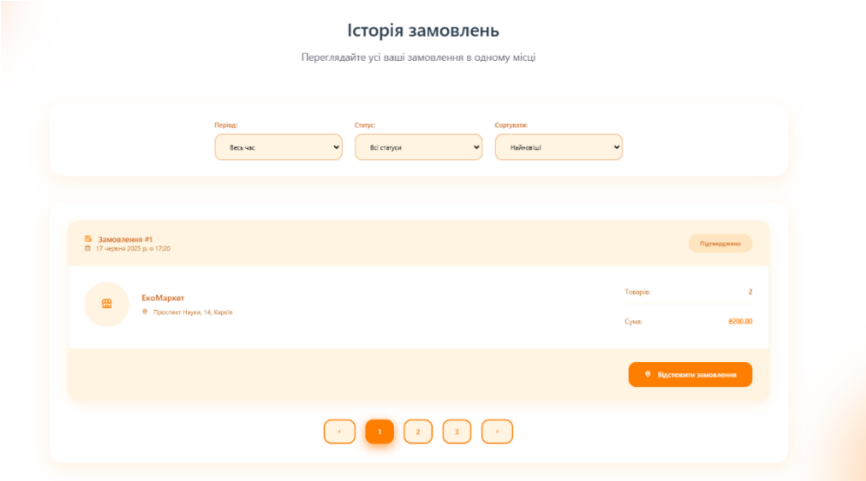

Рисунок 4.10 – Сторінка історії замовлень

На рисунку 4.11 можна побачити сторінку Корзини, там ви можете видаляти частини замовлення, додавати коментарі та регулювати коли ви хочете побачити замовлення та візуальну можливість зміну оплати. Після підтвердження замовлення користувач попаде до наступної сторінки

відслідковування замовлення

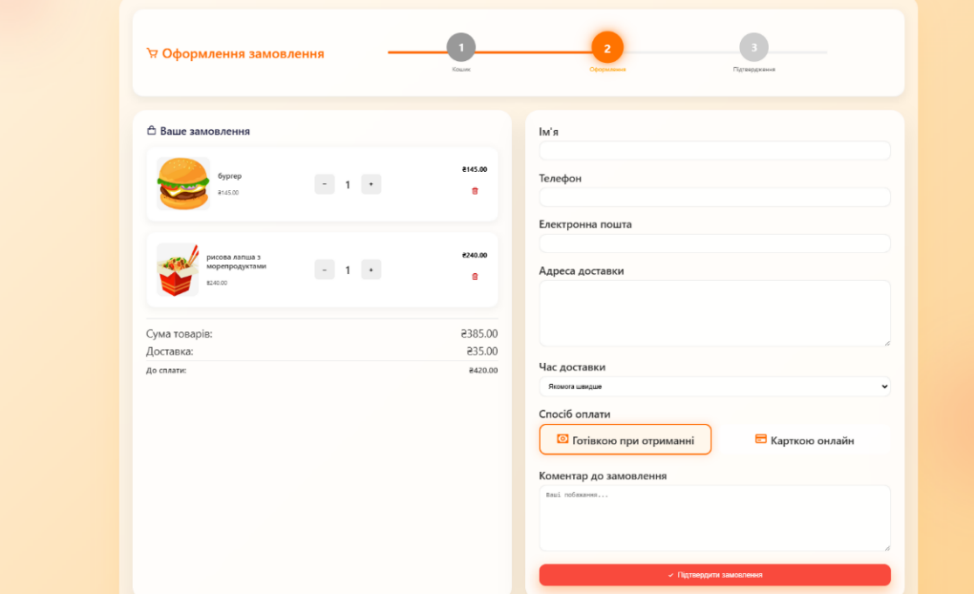

Рисунок 4.11 – Дизайн сторінки створення замовлення

Такий вигляд має відстеження замовлення, вона має лише інформаційний характер, де користувач бачить інформацію про свою доставки та бачить візувально красиві анімації з переходу кожного жтапу доставки і тим самим може відслідковувати етапи доставки (рисунок 4.12).

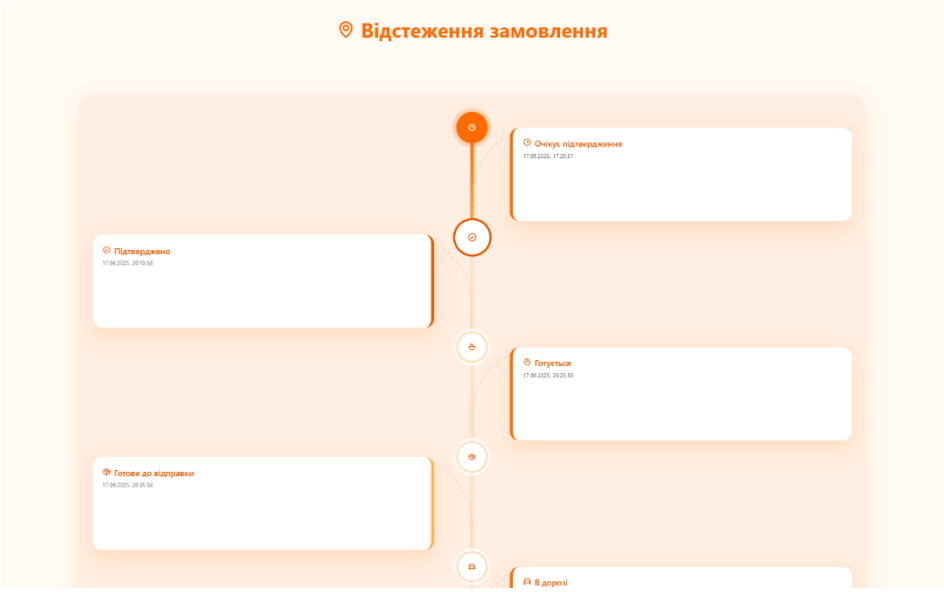

Рисунок 4.12 – Дизайн сторінки відстеження замовлення

Ця сторінка має лендіг для більш наглядної інформації про етапи доставк, яку можна побачити на рисунку 4.13.

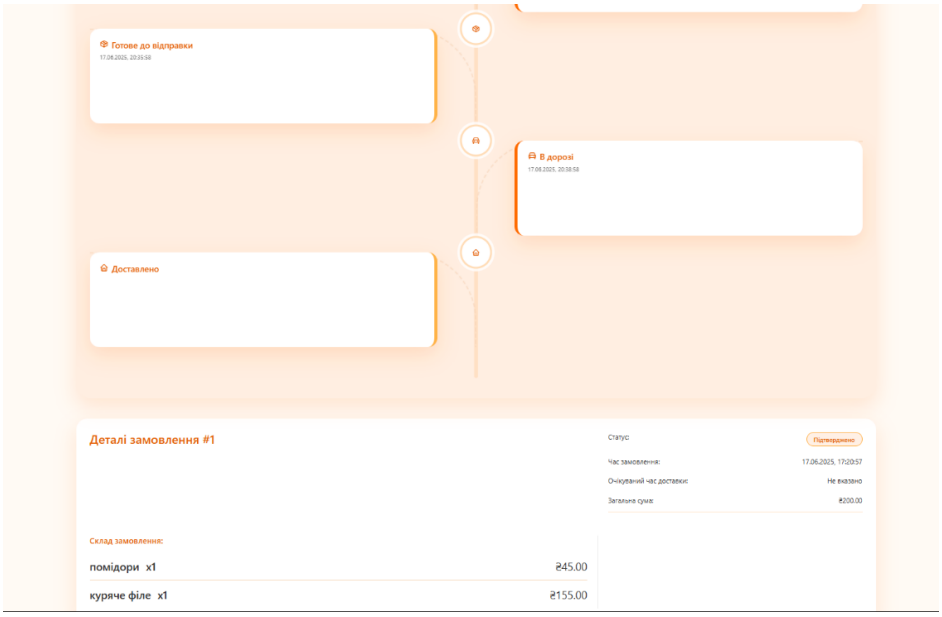

Рисунок 4.13 – Дизайн сторінки відстеження замовлення

Також у користувача є можливість зв’язатися з командою застосунку та написати будь-яке звернення, на рисунку 4.14 представлений дизайн сторінки.

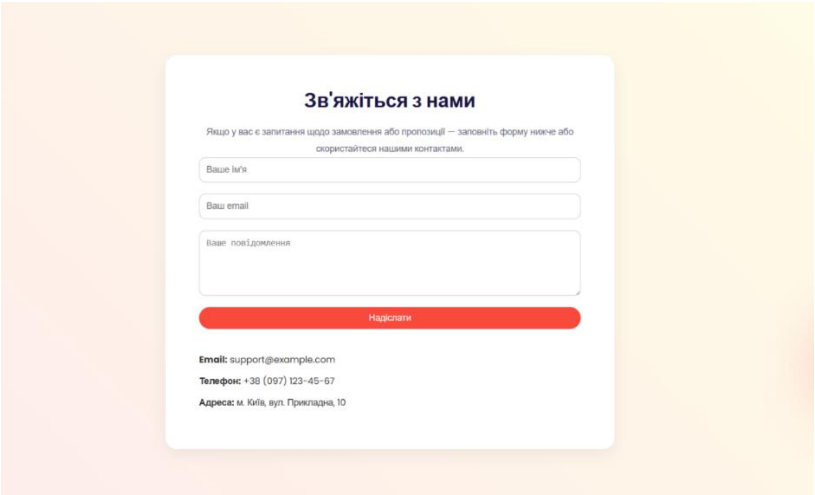

Рисунок 4.14 – Дизайн контактної сторінки

Користувачі матимуть три ролі адмін, кур’єр та звичайний користувач.

В адмін ролі додаться нова сторінка «Адмін панель». Адміль панель дозволяє передавати до кур’єрів замовлення та контролювати етапи замовлення та шукати замовлення за номером. Адміністратор бачить усю інформацю даних про кур’єрів та користувачів і відлідковує статистуку веб-застосунку (рисунок 4.15).

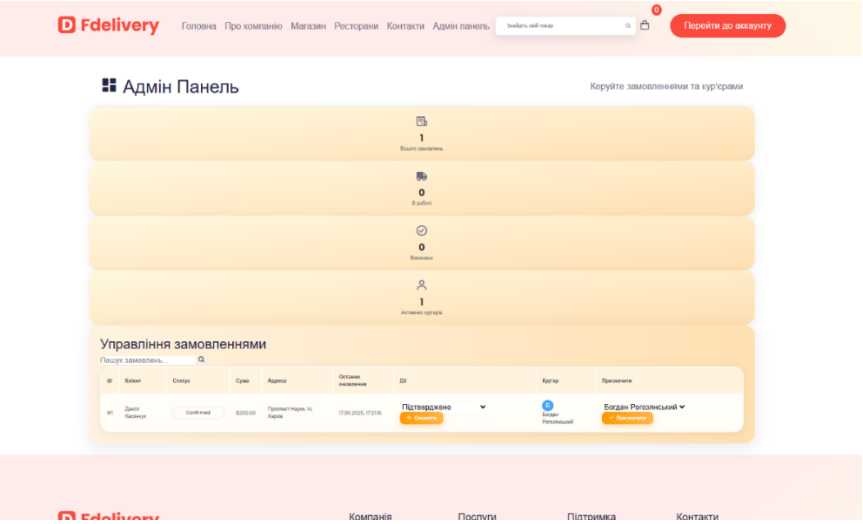

Рисунок 4.15 – Дизайн адмін панелі

Роль кур’єра також має оновлений хедер, у користувача з’являється «Кур’єрська панель», де кур’єр бачить статистику замовлень та відлідковує замовлення які належать йому (рисунок 4.16).

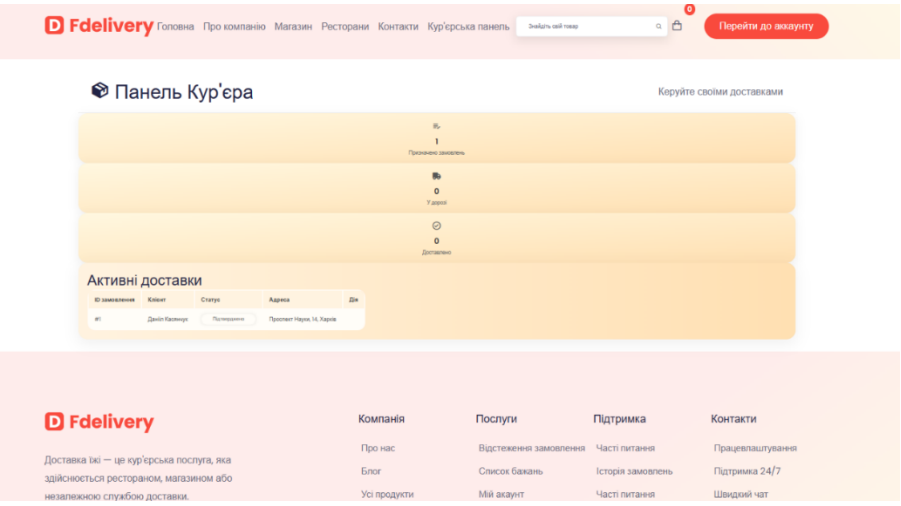

Рисунок 4.16 – Дизайн кур’єр панелі

4.1 Покращення архітектури і структури веб-застосунку

Проєкт має низку потенційних удосконалень, які необхідно реалізувати для забезпечення повноцінного функціонування з боку кінцевого користувача. На стадії розробки кваліфікаційної роботи було використано легкий стек технологій Node.js + SQLite, який добре демонструє ключові можливості, проте в реальному середовищі такі рішення потребують масштабування, адаптації та глибшої інтеграції із сучасними технологіями.

У роботі було використано SQLite – легка вбудована база даних, яка підходить лише для демонстрації. Однак для повноцінного веб-застосунку доцільно перейти на PostgreSQL, яка підтримує складні транзакції, паралельну обробку запитів, більшу стабільність при масштабуванні. Особливо важливим є використання PostGIS – модуля для роботи з геоданими. Він дозволяє точно розрахувати маршрути, зони доставки, координати клієнтів та кур’єрів. Це дає суттєву перевагу в плануванні логістики та оптимізації витрат часу.

Для зручності користувача і відповідності сучасним стандартам та вимогам, робота має підтримувати онлайн-оплати замовлень. Підключення платіжної системи (наприклад: Fondy, LiqPay, PayPal) дозволить користувачам оплачувати замовлення швидко, безпечно і безпосередньо через інтерфейс застосунку. Це впливає на покращення користувацького досвіду, підвищить рівень довіри до сервісу та відкриє можливості для реалізації передоплати.

Комунікація з клієнтом – один з ключових елементів зручного та гарного сервісу. Важливо реалізувати систему розсилок, яка інформуватиме користувача підтвердження замовлення та очікування доставки, а також маркетингові повідомлення. Це можна реалізувати через email (Nodemailer) або SMS API (Twilio, TurboSMS). Своєчасна комунікація підвищує рівень задоволеності користувачів та створює відчуття турботи з боку сервісу.

Щоб не робити навантаження на основний сервер довгими і повторюваними операціями (як-от надсилання листів, збереження аналітики, оновлення статусів), важливо додати асинхронну обробку подій. Зокрема, це

можливо реалізувати за допомогою черг задач (наприклад: Bull.js у зв’язці з Redis). Такий підхід зробить систему стабільнішою при великій кількості одночасних запитів та дасть змогу легко масштабуватися в майбутньому.

Сторінка трекінгу зараз реалізована базово. Для кращого дизайну потрібно впровадити повноцінний трекер доставки з візуалізацією усіх кроків доставки та приготування. Також бажано реалізувати реальний час оновлення статусу – через WebSocket або регулярний polling. У майбутньому можлива інтеграція з картою для відображення доставки в реальному часі.

Для стабільної роботи сервісу необхідно забезпечити постійну URL-адресу і розгортання проєкту на хостингу. Варто обрати сучасні платформи, як Render, яка добре підходить як для фронтенду, так і для бекенду. Постійний хостинг дозволяє тестувати застосунок на різних пристроях та відкриває можливості для оптимізації, також це підвищує загальний рівень професійності проєкту.

ВИСНОВКИ

В результаті виконаної роботи повністю завершено розробку адаптивного веб-додатку для доставки їжі та продуктів. Проект реалізований з урахуванням сучасних вимог до продуктивності, зручності, масштабованості та безпеки, що дозволяє говорити про його високу готовність до впровадження в реальних ринкових умовах. Рішення побудовано на базі сучасного стеку JavaScript з використанням Node.js, Express.js та EJS, що забезпечило як ефективну реалізацію логіки, так і зручний механізм створення динамічного контенту.

Однією з ключових особливостей цього проекту є архітектурна гнучкість, яка дозволяє легко масштабувати програму, додавати новий функціонал або інтегрувати сторонні сервіси без істотних змін базового коду. Такий підхід забезпечує довгострокову актуальність розробки та мінімізує витрати на підтримку.

Особлива увага приділена дизайну інтерфейсу користувача. Програма створена з урахуванням принципів адаптивності, що гарантує зручне використання на різних пристроях – від смартфонів до великих екранів. Інтерфейс відрізняється мінімалістичним стилем, логічною структурою та інтуїтивно зрозумілою навігацією, що суттєво підвищує якість взаємодії користувача із системою. Дизайн не лише підтримує естетику сучасних цифрових сервісів, а й допомагає формувати довіру у клієнтів.

Розробка має низку конкурентних переваг у порівнянні з існуючими гравцями ринку, такими як Glovo або Bolt Food. Зокрема, основний наголос зроблено на швидкість, простоту використання та адаптацію до потреб малого та середнього бізнесу. Це дозволяє нам пропонувати гнучкіше рішення, яке легко налаштовується під локальні умови, індивідуальні бізнес-процеси та специфіку цільової аудиторії.

Інтегровані функції дозволяють користувачам зручно оформляти

замовлення, переглядати історію, відстежувати статус доставки в режимі реального часу та максимально просто та ефективно взаємодіяти з платформою. Все це створює позитивний досвід та підвищує лояльність.

У перспективі веб-додаток має всі необхідні передумови для подальшого розвитку: можлива реалізація мобільної версії, рекомендаційної системи, багатомовного інтерфейсу, платіжних інтеграцій, покращених аналітичних інструментів для бізнесу та вихід на нові географічні ринки. Завдяки гнучкій архітектурі проект легко адаптується до змінних вимог, що відкриває широкі можливості для масштабування та монетизації.

Отже, виконана робота демонструє не лише технічну готовність, а й стратегічну продуманість, високий рівень орієнтації та потенціал для комерційного успіху. Розроблений додаток може стати потужним інструментом для підприємців, які прагнуть вийти на ринок доставки із сучасним, гнучким та ефективним рішенням.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Node.js Foundation, «Introduction to Node.js», Node.js, 2024. [Електронний ресурс]. Режим доступу: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs. Дата звернення: 21 травня 2025.

2. Glovo, «Доставка їжі та багато іншого», Glovo, 2024. [Електронний ресурс]. Режим доступу: https://glovoapp.com/ua/ru/. Дата звернення: 21 травня 2025.

3. Bolt, «Bolt Food — доставка їжі та продуктів», Bolt, 2024. [Електронний ресурс]. Режим доступу: https://bolt.eu/uk-ua/food/. Дата звернення: 21 травня 2025.

4. Doyle, B., & Lopes, C. V. (2008). "Survey of Technologies for Web Application Development" — огляд технологій для розробки веб-застосунків

5. Stack Overflow – платформа для пошуку типових проблеи під час розробки вебзастосунку, Stack Overflow, 2024, [Електронний ресурс]. Режим доступу: https://stackoverflow.co/ Дата звернення: 17 червня 2025.

6. Uxplanet – платформа для веб-дизайнерів, для пошуку нестандартних рішень та поліпшення наглядності, Uxplanet, 2024, [Електронний ресурс]. Режим доступу: https://uxplanet.org/ Дата звернення: 17 червня 2025.

7. MDN Web Docs (JavaScript) – документація до мови програмування JavaScript, MDN Web Docs, 2024, [Електронний ресурс]. Режим доступу: https://developer.mozilla.org/ru/docs/Web/JavaScript Дата звернення: 17 червня 2025.

8. Касянчук Д.І., Ярошевич Р.О. Вплив UX/UI-дизайну на ефективність адаптивних вебзастосунків у сфері замовлення та доставки продуктів харчування // Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління: тези доповідей п’ятнадцятої міжнародної науково-технічної конференції 24 – 25 квітня 2025 р., Баку – Харків – Жиліна https://doi.org/10.32620/ICT.25.t3.