МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

кафедра комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

Проєктування та реалізація програмного модуля кур’єрської доставки

Виконав: студент групи 2П-20
Спеціальності
121 – Інженерія програмного забезпечення

Ілля ЧУХАЛО

Керівник:
Станіслав МАРЧЕНКО

Черкаси 2024

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

Кафедра комп'ютерної інженерії та інформаційних технологій
(повна назва випускової кафедри)

Спеціальність 121 "Інженерія програмного забезпечення"
(шифр і назва спеціальності)

Освітня програма Інженерія програмного забезпечення
(назва освітньої програми)

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерної інженерії та
інформаційних технологій
(назва кафедри)
_____ Хотунов В.І.
(підпис) (ПІБ)
«_____»_____2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Чухала Іллі Віталійовича
(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи Проєктування та реалізація програмного модуля кур'єрської доставки
Керівник роботи Марченко Станіслав Віталійович, спеціаліст І категорії
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від " 13 " жовтня 2023 року № 65У.
2. Строк подання студентом кваліфікаційної роботи 03.06.2024
3. Вихідні дані до кваліфікаційної роботи мова програмування JavaScript, технології React.js/Node.js і Express.js, тестові фреймворки: Jest.
4. Зміст кваліфікаційної роботи (перелік питань, які потрібно розробити) Огляд поточного стану предметної області (бізнес-процеси в галузі кур'єрської доставки, огляд програмних аналогів, постановка задачі на розробку). Проєктування та реалізація програмного продукту (аналіз вимог до кур'єрського модуля, архітектурний опис програмного проєкту, програмна реалізація додатку). Тестування програмного продукту (вибір методів тестування, тестовий план проєкту, аналіз отриманих результатів).
5. Дата видачі завдання 15.09.2023р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання з підписами наукового керівника і студента
1	Вступ	20.10.2023	
2	Розділ 1. Огляд предметної області.	22.12.2023	
3	Розділ 2. Проектування та реалізація програмного продукту	15.03.2024	
4	Розділ 3. Тестування програмного продукту	15.05.2024	
5	Висновки	17.05.2024	
6	Оформлення випускної роботи (чистовий варіант)	27.05.2024	
7	Здача випускної роботи на кафедру для рецензування (за 14 днів до захисту)	31.05.2024	
8	Перевірка випускної роботи на наявність ознак плагіату (за 10 днів до захисту)	03.06.2024	
9	Подання випускної роботи на затвердження завідувачу кафедри (за 7 днів до захисту)	06.06.2024	

Студент _____ Чухало І.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Марченко С.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Вебдодаток «Delivery» є рішенням у сфері доставки їжі, що являє собою кур’єрський модуль для служби доставки. Його метою є спрощення та оптимізація процесу доставки страв з різних закладів харчування в межах міста. Додаток обробляє замовлення та надсилає їх кур’єрам. Замовлення включають в себе велику кількості ресторанів, кафе та магазинів, що охоплюють різноманітні кухні світу та цінові категорії, забезпечуючи задоволення будь-яких гастрономічних вподобань та бюджетних обмежень користувачів.

Інтерфейс «Delivery» має чітку та логічну структуру з простою навігацією, що полегшує оформлення замовлень. Використання додатка для замовлення їжі дозволяє зберегти час, який інакше був би витрачений на пошук закладу, вивчення меню та здійснення традиційного замовлення. Користувачі можуть робити замовлення з будь-якого місця, не залишаючи дому чи офісу, що підвищує ефективність використання часу.

Отже, вебдодаток «Delivery» є рішенням, що поєднує економію часу, різноманітні гастрономічні можливості та зручність замовлення їжі, забезпечуючи користувачів якісними та вигідними послугами доставки.

ABSTRACT

The “Delivery” web application is a food delivery solution that includes a courier module for a delivery service. Its purpose is to simplify and optimize the delivery process of dishes from various food establishments within the city. The application processes orders and sends them to couriers. Orders include many restaurants, cafes, and shops, covering a variety of cuisines of the world and price categories, providing satisfaction of any gastronomic preferences and budget restrictions of users.

The “Delivery” interface has a clear and logical structure with simple navigation that facilitates ordering. Using the app to order food allows you to save time that would otherwise be spent on finding an institution, studying the menu and making a traditional order. Users can make orders from anywhere without leaving home or office, which increases time use efficiency.

Thus, the web application “Delivery” is a solution that combines time saving, various gastronomic possibilities and convenience of ordering food, providing users with high-quality and profitable delivery services.

ЗМІСТ

ВСТУП

Актуальність обраної теми. У XXI столітті цифрові технології невпинно змінюють способи ведення бізнесу та обслуговування клієнтів, що сприяє швидкому розвитку галузі електронної комерції. Пандемія COVID-19 стала каталізатором, який різко прискорив темпи розвитку онлайн-торгівлі, змінивши споживчі звички та збільшивши попит на послуги кур'єрської доставки. Ця тенденція створила нові виклики та можливості для служб доставки, які прагнуть задовольнити зростаючі потреби клієнтів та забезпечити ефективність своїх операцій.

В епоху діджиталізації та посиленої конкуренції, впровадження інноваційних технологічних рішень стає необхідністю для компаній, які прагнуть залишатися конкурентоспроможними на ринку. Одним з ключових елементів успішної кур'єрської служби є наявність потужного програмного модуля, здатного планувати оптимальні маршрути доставки з урахуванням численних факторів, таких як затори на дорогах, час доби, розташування кур'єрів та інші змінні умови.

Застосування передових алгоритмів машинного навчання дає змогу динамічно обчислювати найефективніші маршрути в режимі реального часу, мінімізуючи витрати на паливо, скорочуючи час доставки та підвищуючи продуктивність праці кур'єрів. Крім того, збір та аналіз великих обсягів даних про замовлення, маршрути та діяльність кур'єрів відкриває можливості для глибокого розуміння споживчого попиту, прогнозування навантаження на службу доставки та прийняття обґрунтованих бізнес-рішень.

Об'єкт дослідження. Об'єктом дослідження є бізнес-процеси служби доставок, які підлягають автоматизації та програмуванню.

Предмет дослідження. Предметом дослідження є архітектурні рішення та моделі, застосовані для реалізації та впровадження автоматизованої підсистеми роботи кур'єрської служби. Ключовими задачами є вивчення бізнес-процесів доставки та ідентифікація потреб користувачів, виокремлення функціональних і

нефункціональних вимог до автоматизації, аналіз архітектурних підходів для реалізації відповідного програмного модуля.

Мета дослідження. Метою цього проєкту є розробка програмного модуля кур'єрської служби, який автоматизуватиме основні бізнес-процеси у галузі кур'єрських доставок з урахуванням сучасних архітектурних підходів та рішень.

Розроблений програмний модуль матиме на меті оптимізацію маршрутів доставки, автоматизацію завдань та покращення якості обслуговування клієнтів кур'єрської служби. Його реалізація відповідатиме нагальним потребам ринку електронної комерції та дозволить службам доставки залишатися конкурентоспроможними, забезпечуючи високий рівень задоволеності клієнтів та ефективність операцій в умовах зростаючого попиту на їхні послуги. Для демонстрації переваг проєкту буде створено та відтестовано прототип сервісу кур'єрської доставки з подальшим обговоренням його сильних і слабких сторін.

Завдання дослідження. Для досягнення цієї мети поставлено наступні задачі:

1) вивчення бізнес-процесів, задіяних у роботі служб доставки та ідентифікація потреб користувачів;

2) виокремлення функціональних та нефункціональних вимог до автоматизації бізнес-процесів для кур'єрських доставок, аналіз архітектурних підходів до реалізації відповідного програмного модулю;

3) реалізація та тестування демонстраційного прототипу сервісу кур'єрської доставки, обговорення отриманих переваг та недоліків.

РОЗДІЛ 1
ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Бізнес-процеси в галузі кур'єрської доставки

Галузь кур'єрської доставки охоплює низку складних та взаємопов'язаних бізнес-процесів, які забезпечують безперебійну роботу служб доставки та задоволення потреб клієнтів. Ефективне управління цими процесами є запорукою успішної діяльності компаній у цій сфері.

Одним з ключових бізнес-процесів є приймання та обробка замовлень від клієнтів. Цей процес передбачає збір деталізованої інформації про відправлення, такої як адреса одержувача, розміри та вага посилки, спосіб оплати, вимоги щодо терміновості доставки тощо. Точність та повнота цих даних є критично важливою для забезпечення своєчасності та якості доставки. Автоматизація збору та обробки замовлень може значно підвищити ефективність цього процесу [1].

Наступним ключовим етапом є планування та оптимізація маршрутів доставки. Цей процес полягає у розподілі замовлень між наявними кур'єрами та визначенні найбільш ефективних маршрутів з урахуванням численних факторів, таких як відстань, час у дорозі, затори на дорогах, місцезнаходження кур'єрів та їх навантаження. Оптимізація маршрутів дозволяє мінімізувати витрати на паливо, скоротити час доставки та підвищити продуктивність праці кур'єрів.

Процес відстеження та моніторингу відправлень є життєво важливим для забезпечення прозорості та контролю на всіх етапах доставки. Сучасні системи відстеження дозволяють клієнтам у режимі реального часу відслідковувати місцезнаходження їхніх посилок, а компаніям - швидко реагувати на будь-які затримки або проблеми. Цей процес тісно пов'язаний із забезпеченням безпеки та збереження відправлень під час транспортування.

Ефективне управління персоналом, зокрема кур'єрами, є невід'ємною складовою успішних бізнес-процесів у галузі доставки. Цей процес охоплює найм та навчання кваліфікованих кур'єрів, розподіл робочих змін та маршрутів, моніторинг продуктивності праці, забезпечення належних умов роботи та

безпеки співробітників. Належне управління персоналом сприяє підвищенню якості обслуговування клієнтів та загальної ефективності компанії.

Окрім згаданих ключових процесів, важливими також є процеси управління запасами та складських операцій, обробки повернень та скарг клієнтів, а також адміністративні та фінансові процеси, такі як білінг та розрахунки з клієнтами.

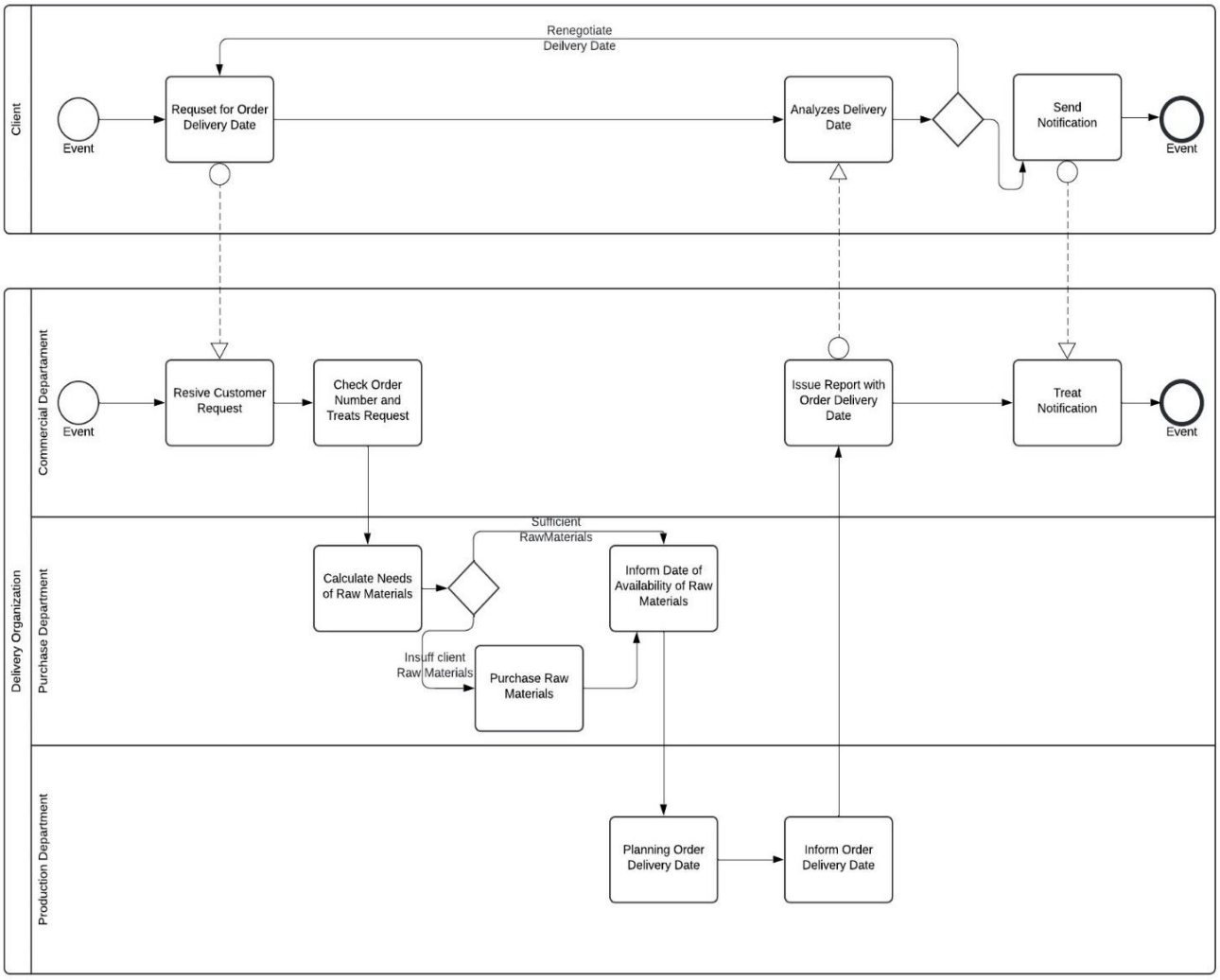

Рисунок 1.1 – BPMN-діаграма бізнес процесів служби доставки

Ця діаграма на рисунку 1.1 є діаграмою бізнес-процесу (BPMN), яка візуалізує процес виконання замовлення та постачання сировини. Давайте розберемо її покроково:

Верхній потік:

– спочатку надходить подія «Запит на дату доставки замовлення»;

– потім виконується завдання «Аналіз дати доставки»;

– після цього, залежно від результату аналізу, надсилається сповіщення.

Середній потік починається з події «Комерційний запит». Далі виконується завдання «Отримати запит від клієнта». Потім відбувається «Перевірка номера замовлення та обробка запиту». Після цього формується «Звіт з датою доставки замовлення». Наприкінці надходить сповіщення «Обробити повідомлення».

Нижній потік починається з завдання «Розрахунок потреб у сировині». Далі проходить через шлюз «Достатньо сировини?». Якщо сировини недостатньо, виконується завдання «Інформувати про наявність сировини» та здійснюється «Закупівля сировини». Потім виконується «Планування дати доставки замовлення». І, нарешті, інформація про дату доставки замовлення передається далі.

Отже, ця BPMN-діаграма описує процес обробки замовлень, включаючи аналіз дати доставки, перевірку наявності сировини, закупівлю недостатньої сировини та планування дати доставки для замовлення.

Автоматизація та оптимізація цих численних бізнес-процесів за допомогою сучасних програмних рішень, що інтегрують технології машинного навчання, аналітики даних та алгоритми оптимізації, є критично важливою для підвищення ефективності, скорочення витрат та забезпечення високої якості обслуговування в умовах зростаючого попиту на послуги кур’єрської доставки.

1.2. Огляд програмних аналогів на ринку

На ринку існує низка програмних рішень, розроблених для автоматизації бізнес-процесів у галузі кур’єрської доставки. Аналіз цих програмних аналогів дозволить визначити їхні сильні та слабкі сторони, а також виявити прогалини та можливості для вдосконалення.

Одним з провідних гравців на ринку є компанія Delivery Solution. Їхній програмний комплекс DeliveryPro являє собою комплексне рішення для автоматизації операцій кур’єрської служби. Він пропонує функції планування

оптимальних маршрутів з урахуванням факторів, таких як відстань, затори та розташування кур’єрів. Система також забезпечує відстеження відправлень у режимі реального часу за допомогою інтеграції з GPS та можливості для клієнтів відслідковувати статус своїх посилок [2].

DeliveryPro пропонує низку переваг, які забезпечують ефективну роботу служби доставки. По-перше, вона надає інтерфейс для відстеження замовлень у режимі реального часу, що дозволяє клієнтам отримувати актуальну інформацію про статус їхніх відправлень. Крім того, програма містить функціональність для оптимізації маршрутів кур’єрів, враховуючи фактори, такі як відстань, час та завантаженість доріг, що сприяє скороченню витрат на пальне та підвищенню продуктивності. DeliveryPro забезпечує надійне зберігання та обробку конфіденційних даних клієнтів відповідно до вимог безпеки та захисту інформації. Але слід визнати, що програма може мати обмежену гнучкість та масштабованість у разі швидкого зростання кількості замовлень або розширення географії доставки.

Вартість придбання та обслуговування такого програмного забезпечення може бути досить високою для невеликих компаній. Незважаючи на ці недоліки, DeliveryPro є потужним інструментом для оптимізації процесів кур’єрської доставки та підвищення задоволеності клієнтів у цій сфері діяльності.

DeliveryPro має модуль управління персоналом, що дає змогу призначати маршрути кур’єрам, контролювати їхню продуктивність та формувати графіки робочих змін. Крім того, система пропонує автоматизацію процесів білінгу, включаючи генерацію рахунків, облік оплат та інтеграцію з платіжними шлюзами.

Представлений на рис. 1.2 інтерфейс DeliveryPro може бути складним та заплутаним для освоєння, особливо для невеликих компаній. Також система має обмежені можливості для глибокого аналізу даних та виявлення прихованих тенденцій, що може ускладнювати прийняття стратегічних рішень.

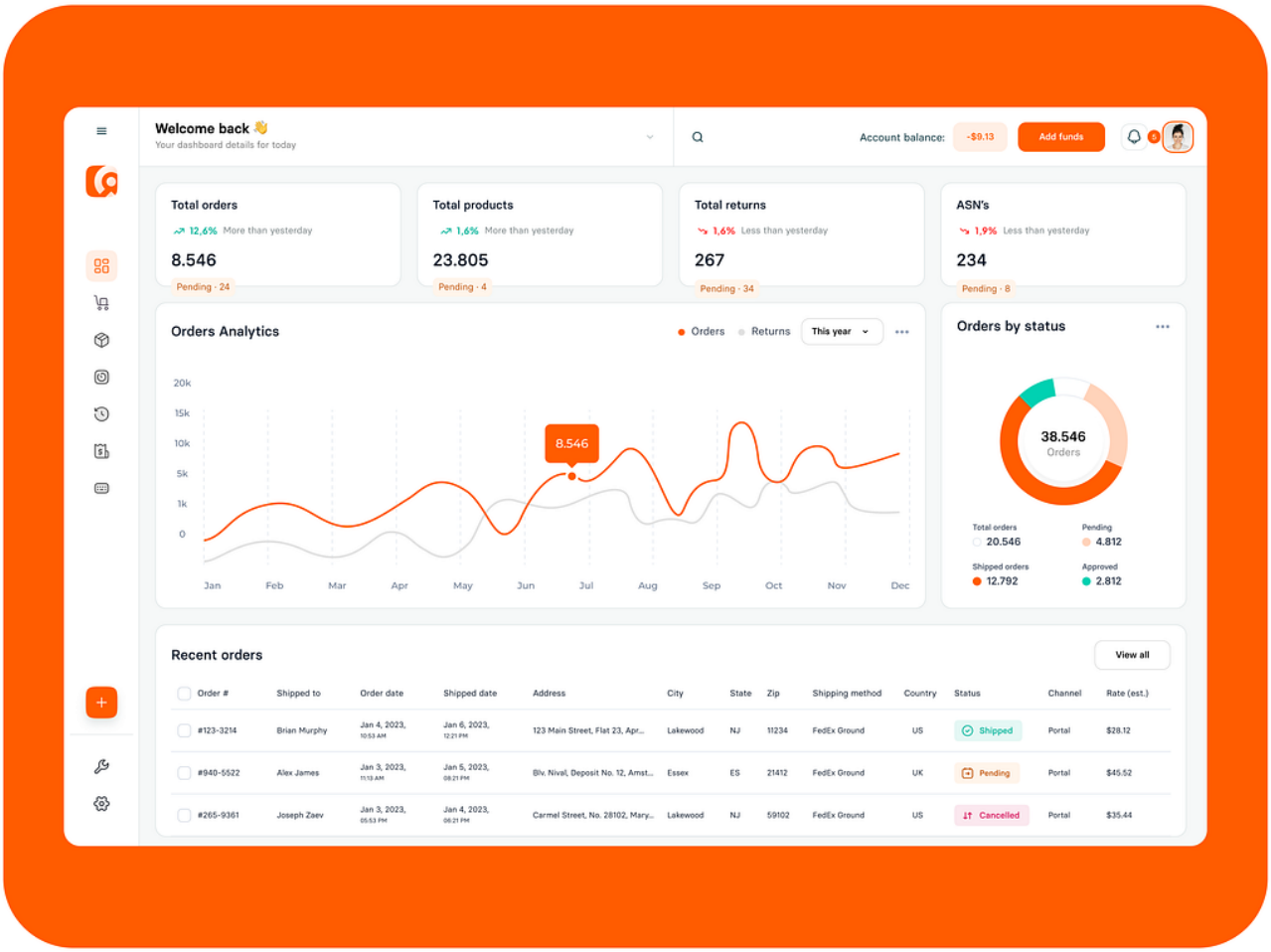

Рисунок 1.2 – Скріншот інтерфейсу застосунку Delivery Pro

Іншим відомим рішенням на ринку є програмний комплекс Courier Wizard від однойменної компанії, інтерфейс програмного комплексу зображений на рис. 1.3. Це рішення спеціалізується на оптимізації маршрутів доставки та рівномірному розподілі навантаження між кур'єрами служби [3].

Courier Wizard має низку переваг, які сприяють ефективному управлінню процесом доставки. Однією з головних переваг є вебінтерфейс, що дозволяє легко відстежувати замовлення в режимі реального часу, забезпечуючи прозорість для клієнтів.

Програма містить функціональність для автоматичного планування оптимальних маршрутів кур'єрів з урахуванням відстані, часу та поточної ситуації на дорогах, що сприяє скороченню витрат на пальне та підвищенню продуктивності. Courier Wizard також забезпечує високий рівень захисту

конфіденційних даних користувачів завдяки використанню сучасних протоколів шифрування та технологій безпеки.

Також важливо зазначити певні недоліки Courier Wizard. Через те, що програма є веборієнтованою, її функціональність може бути обмежена в регіонах з низькою швидкістю Інтернет-з'єднання. Крім того, процес інтеграції з існуючими системами компанії може бути складним та вимагати додаткових ресурсів.

Також важливо, що масштабованість програми на великі обсяги замовлень може бути ускладненою, що може стати перешкодою для подальшого зростання бізнесу. Незважаючи на ці недоліки, Courier Wizard є потужним інструментом для організації ефективної системи кур'єрської доставки, що забезпечує високий рівень задоволеності клієнтів та оптимізацію логістичних процесів.

Система тісно інтегрована з провідними картографічними сервісами, такими як Google Maps, Bing Maps та OpenStreetMap. Це дозволяє точно відстежувати місцезнаходження кур'єрів, обчислювати маршрути з урахуванням поточної дорожньої ситуації та надавати клієнтам актуальну інформацію про статус доставки.

Однак, Courier Wizard наразі не має вбудованих інструментів машинного навчання та можливостей прогнозування попиту на послуги доставки. Це може обмежувати здатність служб кур'єрської доставки приймати вивірені стратегічні рішення на основі аналізу історичних даних та виявлення прихованих тенденцій.

Ще одним важливим гравцем на ринку програмних рішень для кур'єрських служб є представлена на рис. 1.4, компанія DeliveryTech зі своїм хмарним продуктом під назвою CloudDeliver. На відміну від традиційних локальних програм, це рішення повністю розміщується у хмарі та доступне через веб-інтерфейс [4].

Однією з головних переваг CloudDeliver є її доступність з будь-якого пристрою, підключеного до Інтернету. Це забезпечує гнучкість та мобільність для управління операціями кур'єрської служби. Крім того, хмарна архітектура

дозволяє легко масштабувати систему відповідно до зростаючих потреб бізнесу, не вимагаючи додаткових витрат на обладнання.

Рисунок 1.3 – Скріншот інтерфейсу застосунку "Courier Wizard"

Завдяки розміщенню в хмарному середовищі, CloudDeliver забезпечує високу доступність та відмовостійкість, мінімізуючи ризики простоїв та втрати даних. Система автоматично займається резервним копіюванням та оновленнями.

Програма має інтерфейс, який забезпечує прозоре відстеження замовлень у режимі реального часу, що є важливим фактором для забезпечення задоволеності споживачів, сам інтерфейс представлений на рисунку 1.4. Крім того, CloudDeliver оснащена потужною функціональністю для автоматичного планування маршрутів з урахуванням різноманітних факторів, таких як відстань, завантаженість доріг та час доставки, що дає змогу значно оптимізувати витрати на пальне та підвищити ефективність роботи кур’єрської служби.

Однак, хмарний продукт DeliveryTech має обмежені можливості для глибокого налаштування та інтеграції з іншими корпоративними системами чи

додатками. Замовники можуть стикатися з труднощами під час адаптації системи до своїх специфічних вимог або бізнес-процесів.

Окрім комерційних програмних продуктів, на ринку присутні й відкриті програмні проекти для автоматизації служб кур'єрської доставки. Одним з таких проектів є OpenDelivery – система з повністю відкритим вихідним кодом.

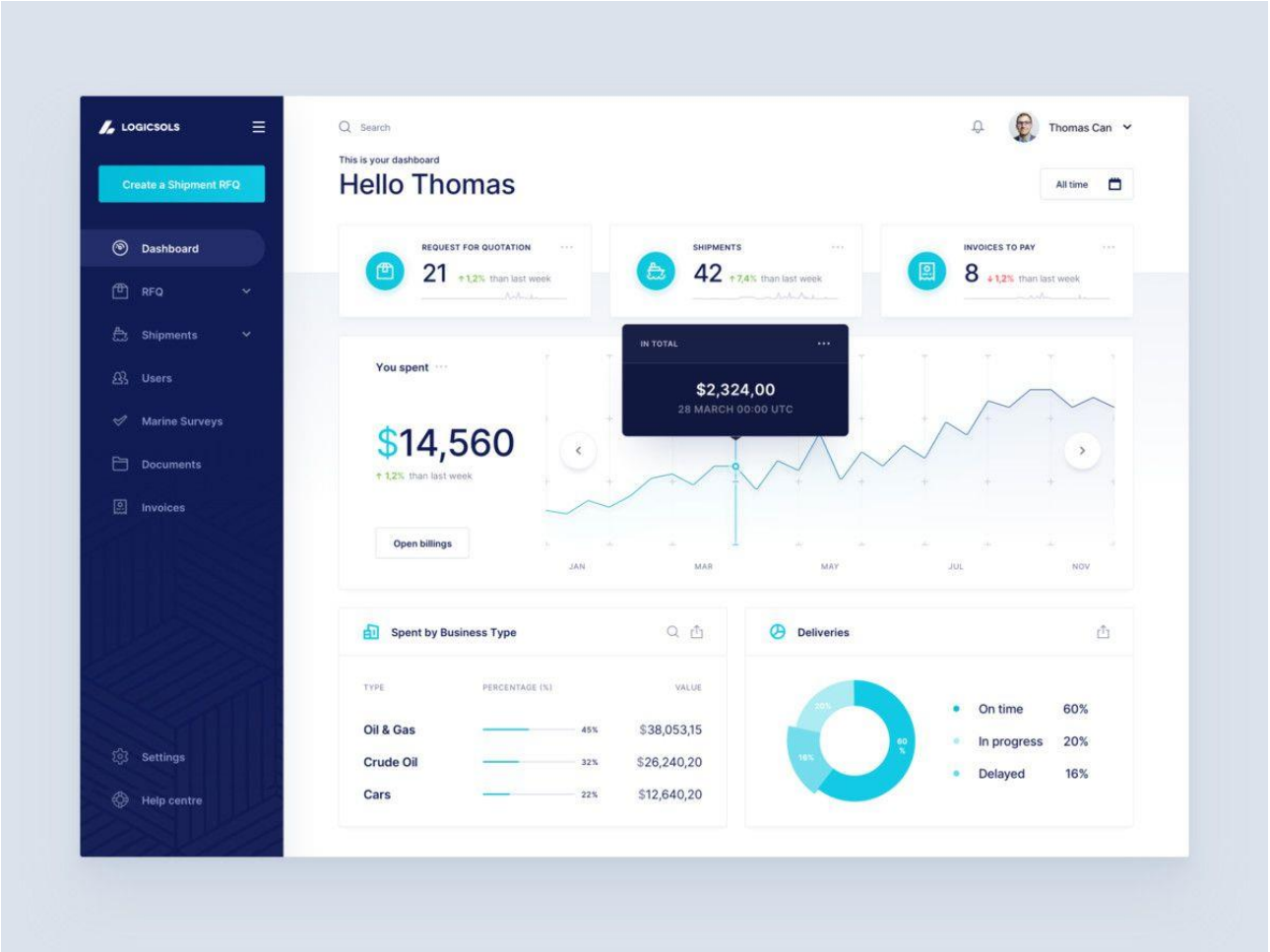

Рисунок 1.4 – Скріншот інтерфейсу застосунку DeliveryTech

Платформа OpenDelivery пропонує унікальну перевагу у вигляді повної гнучкості та можливості адаптації функціональності під специфічні потреби замовників. Маючи доступ до відкритого вихідного коду системи, команди розробників можуть вносити модифікації в базову архітектуру, додавати нові модулі, здійснювати інтеграцію з іншими програмними рішеннями чи сервісами відповідно до вимог бізнесу. Відкритість коду також дозволяє залучати широку спільноту розробників для активної участі у вдосконаленні та виправленні

потенційних недоліків системи, що може забезпечити високу якість та стабільність рішення в довгостроковій перспективі.

Проте впровадження відкритої системи OpenDelivery може виявитися складним завданням, особливо для невеликих компаній з обмеженими IT-ресурсами. Цей процес вимагає залучення досвідчених фахівців, здатних глибоко зрозуміти архітектуру системи, забезпечити її налаштування, інтеграцію та подальшу технічну підтримку. Використання відкритого програмного забезпечення може потенційно створювати ризики для безпеки та виникнення вразливостей, якщо спільнота розробників не приділяє належної уваги цим питанням.

Звідси, OpenDelivery може бути одним з найкращих рішеннь для великих компаній з потужними IT-підрозділами, які здатні витрачати значні ресурси на адаптацію та підтримку системи. Для інших організацій комерційні продукти можуть виявитися більш практичним варіантом, забезпечуючи готову функціональність та технічну підтримку від постачальника.

Аналіз існуючих програмних аналогів на ринку кур’єрських послуг виявляє, що більшість з них зосереджуються лише на базових функціях, таких як планування маршрутів та відстеження відправлень. Однак, вони часто не мають повноцінної інтеграції передових технологій, включаючи машинне навчання, аналітику даних та прогнозування попиту. Використання інноваційних методів забезпечує більш глибоку автоматизацію та оптимізацію бізнес-процесів кур’єрських служб за рахунок використання сучасних аналітичних інструментів та алгоритмів штучного інтелекту.

1.3. Постановка задачі на розробку

Майбутній програмний модуль має забезпечити комплексне рішення, спроможне покращити ефективність операцій служб доставки та підвищити рівень обслуговування клієнтів. Крім того, він має забезпечувати відстеження відправлень, надаючи клієнтам прозору інформацію про статус їхніх посилок та очікувані терміни доставки. Також необхідно реалізувати функції управління

персоналом, зокрема розподілу робочих змін, моніторингу продуктивності кур’єрів та забезпечення їх належної взаємодії із системою.

Програмний модуль повинен інтегрувати передові архітектурні рішення, такі як мікросервісна архітектура, хмарні обчислення та контейнеризація застосунків. Це забезпечить масштабованість, високу доступність, відмовостійкість та полегшить подальший розвиток і технічну підтримку системи.

Окрім функціональних можливостей, важливим аспектом є створення зручного та інтуїтивного користувацького інтерфейсу, що спростить взаємодію з програмним модулем як для операторів кур’єрської служби, так і для клієнтів.

Загалом, цей проєкт спрямований на створення комплексного рішення, яке об’єднає новітні технології, архітектурні підходи та передові практики для вирішення актуальних проблем автоматизації бізнес-процесів кур’єрської доставки, підвищення ефективності операцій та якості обслуговування клієнтів [5].

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

2.1. Проєктування програмного продукту

Проєктування та реалізація ефективного програмного модуля є ключовим етапом у процесі розробки програмного забезпечення. Цей крок передбачає ретельне планування, аналіз вимог, архітектурне проєктування та кодування згідно із встановленими стандартами та найкращими практиками. Належне проєктування модуля забезпечує його зрозумілість, підтримуваність, масштабованість і гнучкість [5].

На початковому етапі проєктування модуля визначаються його функціональні вимоги, а також взаємозв’язки з іншими компонентами системи. Це дає змогу створити чітке уявлення про архітектуру модуля та його роль у загальній системі. Після цього розробляється деталізована специфікація, яка описує інтерфейси, вхідні та вихідні дані, очікувану поведінку та виняткові ситуації, що можуть виникнути під час роботи модуля.

Наступним кроком є вибір відповідних технологій, бібліотек та фреймворків, які будуть використовуватися для реалізації модуля. Цей вибір повинен ґрунтуватися на вимогах до продуктивності, масштабованості, безпеки та сумісності з іншими компонентами системи. Також важливо враховувати досвід розробників та наявні ресурси для забезпечення ефективної роботи над проєктом.

Після цього розпочинається безпосередньо реалізація програмного модуля. Розробник дотримується встановлених стандартів кодування, принципів об’єктно-орієнтованого програмування та найкращих практик розробки програмного забезпечення. Під час реалізації важливо забезпечити належну модульність, підтримуваність та розширюваність коду, щоб полегшити його подальше тестування, інтеграцію та супровід [5-7].

2.1.1. Огляд функціональних вимог до програмного модуля

Розробка програмного забезпечення для кур'єрської доставки вимагає ретельного планування та розгляду, проте найголовнішим аспектом є визначення відповідної функціональності. Незалежно від того, чи є користувач клієнтом, який очікує на доставку відправлення, кур'єром, відповідальним за своєчасну доставку, або адміністратором, який здійснює керівництво операціями, наявність ретельно спроектованого та багатофункціонального додатку для кур'єрської доставки є критично важливим. Доцільно проаналізувати функції, які є необхідними при розробці програмного забезпечення для кур'єрської доставки, щоб забезпечити комфортну взаємодію для всіх учасників процесу [6-7].

Одним з ключових аспектів є забезпечення ефективної взаємодії з агентами доставки, які відіграють вирішальну роль у процесі своєчасної та якісної доставки відправлень. З цією метою необхідно реалізувати низку важливих функцій, які сприятимуть безперебійній роботі агентів та підвищенню загальної продуктивності системи:

– спрощена онлайн-авторизація представлена на рис. 2.1: система має давати змогу агентам вводити індивідуальний логін і пароль для входу в систему.

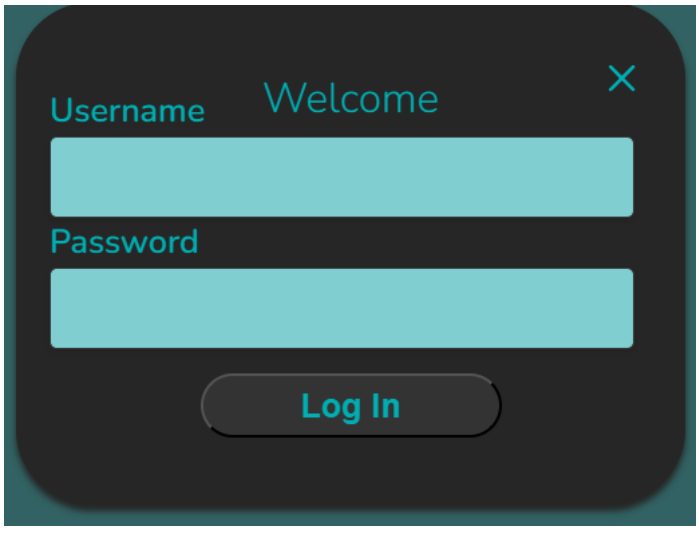

Рисунок 2.1 – Вікно авторизації в модулі кур'єрської доставки

– механізм прийняття замовлень, який представлений на рис.2.2: агенти повинні мати чітке уявлення про доступні замовлення та можливість

приймати або відхиляти їх залежно від своєї доступності та місцезнаходження. Ця функція є критично необхідною для безперебійної роботи додатку.

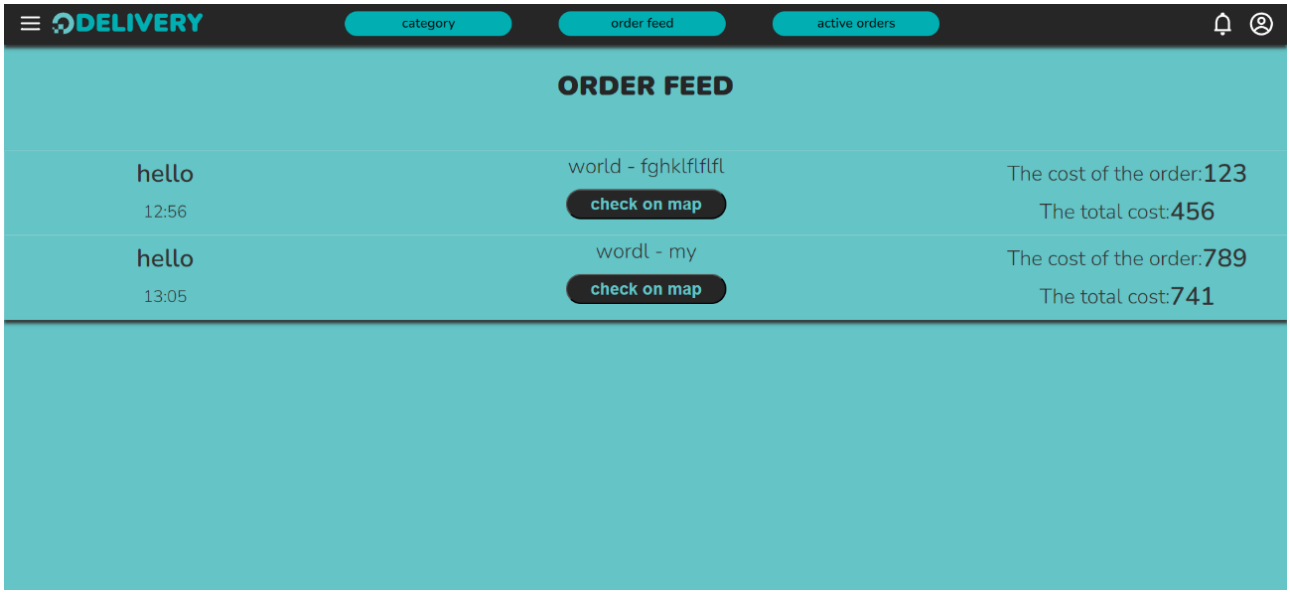

Рисунок 2.2 – Вікно активних замовлень в системі

– інтеграція GPS-відстеження та оптимізації маршрутів: алгоритми мають розраховувати найоптимальніші маршрути для агентів з урахуванням відстані, дорожньої ситуації та наявності декількох доставок. Це надаватиме змогу мінімізувати витрати часу та пального, підвищуючи загальну ефективність доставки.

– категоризація: кур’єри повинні мати можливість швидко і легко вибирати замовлення відповідно до обраної категорії. Це можуть бути такі категорії, як продукти харчування, іграшки, випічка, поштові відправлення та інші. Завдяки чіткій системі категоризації кур’єри зможуть ефективно знаходити потрібні замовлення, що сприятиме швидкому виконанню доставки та підвищенню якості обслуговування клієнтів.

2.1.2. Огляд нефункціональних вимог до програмного модуля

Нефункціональні вимоги до програмного модуля кур’єрської доставки мають важливе значення для забезпечення відповідного рівня продуктивності, надійності, безпеки та зручності використання системи в цілому.

На відміну від функціональних вимог, що визначають конкретні можливості та функції системи, нефункціональні вимоги охоплюють загальні атрибути, характеристики та поведінку програмного забезпечення. Основні нефункціональні вимоги до модуля кур’єрської доставки включають вимоги до продуктивності, доступності, масштабованості, безпеки, зручності використання та сумісності з різними платформами і пристроями.

Забезпечення високої продуктивності є одним із ключових аспектів нефункціональних вимог. Програмний модуль повинен забезпечувати швидку обробку замовлень, оновлення статусів доставки в режимі реального часу та ефективну обробку великих обсягів даних без затримок або збоїв. Додатково система повинна бути доступною для користувачів цілодобово, забезпечуючи мінімальний час простою та високу надійність роботи, що є критично важливим для безперервного обслуговування клієнтів.

Масштабованість системи також є важливим аспектом нефункціональних вимог. Програмний модуль має бути розроблений таким чином, щоб забезпечити можливість легкого розширення та масштабування в майбутньому, дозволяючи обслуговувати зростаючу кількість користувачів та замовлень без зниження продуктивності або виникнення збоїв [8].

Безпека даних та конфіденційність інформації клієнтів є дуже важливими аспектами, що потребують впровадження надійних заходів захисту, таких як шифрування даних, багаторівнева автентифікація користувачів, контроль доступу на основі ролей та інших передових методів забезпечення безпеки. Система повинна гарантувати захист конфіденційної інформації клієнтів від несанкціонованого доступу або витоку.

Зручність використання також є ключовим нефункціональним аспектом, що потребує інтуїтивно зрозумілого та зручного інтерфейсу для користувачів, простоти навігації, чіткої візуалізації даних та ефективної підтримки користувачів у разі виникнення проблем або запитань.

Забезпечення сумісності програмного модуля з різними платформами, пристроями та операційними системами є важливим для безперебійного

функціонування системи в різноманітних середовищах, що використовуються клієнтами та кур’єрами [8].

Загалом, ретельне дотримання нефункціональних вимог є невід’ємною умовою для створення надійної, безпечної, продуктивної та зручної у використанні системи кур’єрської доставки, яка забезпечуватиме високий рівень обслуговування клієнтів, ефективність операцій та конкурентоспроможність на ринку.

2.1.3. Визначення комплексу технологій для розробки програмного забезпечення

Вибір відповідного набору технологій є дуже важливим етапом у процесі розробки програмного забезпечення системи кур’єрської доставки. Правильний підбір технологічного стеку забезпечує ефективність, масштабованість та гнучкість кінцевого продукту. Для реалізації даного проекту було обрано комплекс сучасних та потужних технологій:

React.js – це бібліотека JavaScript з відкритим вихідним кодом, яка використовується для створення користувацьких інтерфейсів. Вона базується на концепції компонентів, які є ізольованими та багаторазово використовуваними частинами коду, що представляють окремі елементи інтерфейсу. React.js забезпечує ефективне оновлення та рендеринг компонентів у браузері за допомогою віртуального DOM, що значно покращує продуктивність порівняно з традиційними підходами [9].

React.js володіє численними перевагами, що роблять її потужним інструментом для створення користувацьких інтерфейсів. Компонентний підхід бібліотеки заохочує розбиття інтерфейсу на окремі, багаторазово використовувані компоненти, полегшуючи розробку, тестування та підтримку коду.

Використання віртуального DOM дозволяє React.js ефективно оновлювати лише ті частини інтерфейсу, які зазнали змін, замість повного перерендеринга, що значно підвищує продуктивність. Односпрямований потік даних робить стан

додатку більш передбачуваним і полегшує відстеження змін та виправлення помилок. React.js має велику та активну спільноту розробників, що забезпечує постійний розвиток бібліотеки, наявність численних корисних бібліотек та інструментів. Крім того, React.js можна використовувати не лише для розробки веб-додатків, але й для створення мобільних додатків за допомогою React Native, забезпечуючи кросплатформність [10].

Однак React.js має певні недоліки. Власна концепція та синтаксис бібліотеки можуть становити виклик для нових розробників, особливо тих, хто не має досвіду роботи з функціональним програмуванням, що може утруднити процес навчання. Оскільки React.js є лише бібліотекою для розробки користувацького інтерфейсу, для створення повноцінного додатку може знадобитися підключення додаткових бібліотек, наприклад, для маршрутизації, управління станом або асинхронних операцій. Крім того, будучи бібліотекою з відкритим вихідним кодом, React.js не має офіційної підтримки розробників, як комерційні рішення, тому розробники покладаються на активну спільноту та ресурси, доступні онлайн.

Для створення користувацького інтерфейсу додатку використовується React.js – бібліотека JavaScript, яка забезпечує компонентний підхід до розробки. Це сприяє модульності, повторному використанню коду та підвищує продуктивність завдяки використанню віртуального DOM. У поєднанні з MobX, бібліотекою для керування станом додатку, що забезпечує реактивну парадигму програмування, React.js дозволяє створити зрозумілий та реактивний інтерфейс, який ефективно оновлюється у відповідь на зміни даних [10].

На стороні сервера використовується Express.js – мінімалістичний веб-фреймворк для Node.js. Express.js забезпечує зручний спосіб маршрутизації запитів, обробки HTTP-запитів та відповідей, а також інтеграцію з різними модулями та бібліотеками. Node.js, як середовище виконання JavaScript на сервері, надає асинхронну модель введення-виведення, що дозволяє ефективно обробляти паралельні запити та забезпечувати високу продуктивність додатку [11, 12].

Express.js є потужним веб-фреймворком, який має низку переваг для розробки серверної частини веб-додатків та API на Node.js. Він відомий своєю високою швидкістю та ефективністю, що дозволяє створювати швидкі та масштабовані додатки. Фреймворк забезпечує гнучкий та модульний підхід до розробки, надаючи можливість легко підключати необхідні модулі та бібліотеки для вирішення конкретних завдань.

Express.js має зручний вбудований маршрутизатор, який забезпечує простий та ефективний спосіб визначення маршрутів для різних HTTP-методів та шляхів. Підтримка middleware-функцій дозволяє розробникам обробляти запити та відповіді, додавати функціональність та забезпечувати модульність коду. Крім того, Express.js має велику та активну спільноту розробників, що забезпечує постійний розвиток фреймворку, наявність численних додаткових бібліотек та ресурсів.

Проте Express.js має певні недоліки. Він не володіє вбудованими інструментами для керування станом додатку, що може ускладнити розробку складних додатків з великою кількістю станів. Незважаючи на те, що Node.js підтримує асинхронне програмування, Express.js не надає вбудованих засобів для обробки асинхронних операцій, що може вимагати використання додаткових бібліотек або власних рішень. Як і багато інших проектів з відкритим вихідним кодом, Express.js не має офіційної підтримки розробників, що покладає більшу відповідальність на спільноту та розробників у вирішенні проблем та підтримці фреймворку [12].

Для зберігання та керування даними було обрано MongoDB – популярну неструктуровану базу даних, яка зберігає дані у вигляді документів JSON. MongoDB забезпечує гнучкість у моделюванні даних, масштабованість та високу продуктивність для обробки великих обсягів даних. У поєднанні з Mongoose, об’єктно-орієнтованим середовищем моделювання даних для MongoDB та Node.js, розробникам надається зручний інтерфейс для визначення схем даних, створення моделей, виконання запитів та роботи з даними в MongoDB [13].

MongoDB – це популярна NoSQL база даних відкритого коду, орієнтована на документи. Вона використовує концепцію «документів» замість традиційних рядків та стовпців у реляційних базах даних. Ця розподілена, масштабована, високопродуктивна база даних зберігає структуровані або напівструктуровані дані у вигляді документів у форматі JSON. Вона належить до класу баз даних NoSQL, що не потребують жорсткої схеми та використовуються для обробки величезних обсягів даних.

У MongoDB відсутня необхідність жорсткої схеми, що надає цій базі даних низку переваг. По-перше, це забезпечує високу продуктивність під час виконання запитів, операцій вставки та оновлення даних.

По-друге, MongoDB демонструє належну масштабованість завдяки можливості реплікації та шардингу даних. Крім того, динамічна схема дозволяє зберігати документи різних структур в одній колекції, чим забезпечується гнучкість системи. Ще однією перевагою є підтримка складних запитів та індексації, а також зберігання даних у форматі JSON, що полегшує їхній обмін між вузлами.

Водночас MongoDB має низку недоліків: система не підтримує класичні транзакції, хоча й дозволяє виконувати невеликі транзакції на рівні документа. Також відсутня можливість об’єднання даних з різних колекцій за допомогою join-ів, що вимагає застосування денормалізації чи програмної реалізації цієї функції. Окрім того, MongoDB не підтримує складні операції, типові для реляційних баз даних. Використання JSON-документів призводить до високої кількісті використовуємої пам’яті для зберігання даних [13].

Для полегшення процесу розробки використовується Nodemon - утиліта, яка автоматично перезавантажує сервер Node.js після внесення змін у вихідний код. Це забезпечує зручність розробки та підвищує продуктивність роботи, оскільки немає потреби вручну перезапускати сервер після кожної зміни.

Nodemon - це інструмент для розробників, який відслідковує будь-які зміни в ваших вихідних файлах Node.js і автоматично перезапускає сервер. Це усуває

необхідність вручну зупиняти і перезапускати сервер після внесення змін під час процесу розробки.

Nodemon є утилітою, що підвищує продуктивність розробки застосунків на Node.js завдяки автоматичному перезапуску сервера після внесення змін в код, економлячи час і зусилля розробників. Ця утиліта проста у використанні, легко встановлюється та налаштовується за допомогою простих команд запуску. Nodemon підтримує гаряче перезавантаження, що дозволяє зберігати стан додатку між перезапусками і уникати втрати даних та контексту під час розробки. Утиліта здатна відстежувати зміни не лише в файлах JavaScript, але й в інших залежних файлах, наприклад HTML та CSS. Користувачі мають змогу налаштувати поведінку перезапуску, ігноруючи певні файли або директорії [15].

Проте Nodemon додає деяке додаткове навантаження на систему через постійне відстеження змін файлів і в рідкісних випадках може спричинити проблеми з кешуванням чи незбереженими даними.

Ця утиліта не призначена для використання у виробничому середовищі, де рекомендується застосовувати інші інструменти керування Node.js додатками. Також слід враховувати, що Nodemon покладається на файлову систему для відстеження змін, тому може мати певні обмеження в окремих системах або під час використання певних інструментів.

Таким чином, обраний стек технологій забезпечує потужну та гнучку платформу для розробки програмного забезпечення системи кур’єрської доставки, поєднуючи сучасні фреймворки та бібліотеки для створення зручного та реактивного користувацького інтерфейсу, ефективної обробки запитів на стороні сервера, масштабованого та гнучкого зберігання даних, а також зручності розробки.

2.1.4. Вимоги до користувацького інтерфейсу та зручності використання

Вимоги до користувацького інтерфейсу та зручності використання відіграють важливу роль у забезпеченні високого рівня задоволеності

користувачів програмним модулем кур’єрської доставки. Інтерфейс повинен бути зрозумілим, зручним у використанні та естетично привабливим для всіх категорій користувачів, включаючи клієнтів, кур’єрів та адміністраторів.

Дизайн користувацького інтерфейсу має відповідати принципам юзабіліті, забезпечуючи простоту навігації, чітку ієрархію інформації та зручність взаємодії. Розташування елементів керування та навігаційних компонентів повинно бути логічним та послідовним на всіх сторінках додатку. Необхідно використовувати загальноприйняті значки, піктограми та термінологію, знайомі більшості користувачів. Система повинна забезпечувати зручний процес керування замовленнями та маршрутами для кур’єрів [16].

2.2. Проєктування користувацького інтерфейсу кур’єрської доставки

У процесі проєктування користувацького інтерфейсу системи резервування квитків пріоритетом став комфорт і простота для агентів доставки. Основним завданням було створити зрозумілу навігацію, забезпечити швидкий пошук потрібних категорій і зручний спосіб прийому замовлень.

Інтерфейс сконструйовано відповідно до загальновизнаних юзабіліті-стандартів, як-от ефективність використання, запам’ятовуваність та задоволення агентів. Цільовою аудиторією системи є люди різного віку та рівня технічної грамотності, які пробують себе у ролі кур’єра [16].

Для створення макетів і прототипів інтерфейсу, представленого на рис. 2.3. використано інструмент Figma – векторний онлайн-сервіс для розробки інтерфейсів і прототипування з можливістю організації спільної роботи. Figma відрізняється від інших подібних програм тим, що має веб-інтерфейс, який дозволяє легко ділитися проєктами й спільно редагувати їх з іншими користувачами без необхідності встановлювати додаткове програмне забезпечення [17].

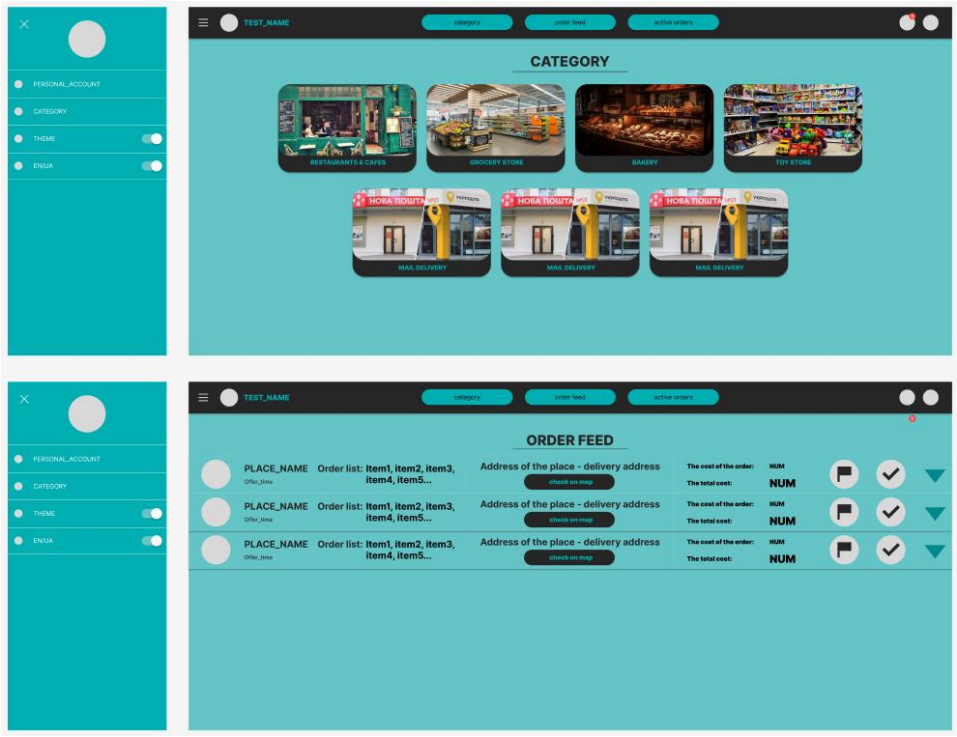

Рисунок 2.3 – Прототип інтерфейсу головної сторінки та замовлення

На головній сторінці закріплена навігаційна панець, яка складається з іконки акаунту, іконки сповіщень від сервісу, логотипу системи та додаткової бічної навігаційної панель, також закріплена панель має навігаційні кнопки: активні замовлення, стрічка замовлень та список категорій замовлень. Після натискання на іконку меню з’являється додаткова бічна навігаційна панель, вона включає: фото акаунту та ім’я, перехід до категорій, стрічки замовлень або стрічки активних замовлень користувача (адміністратору додатково надається доступ до кнопки виклику форми створення замовлення). Головною сторінкою є вибір замовлень за категоріями: ресторани та кафе, магазини, пекарні, магазини іграшок та поштова доставка.

Кінцевий інтерфейс, представлений на рис. 2.4. виконаний у сучасному мінімалістичному дизайні, із ретельно підібраною кольоровою гамою. Основними кольорами, що використовуються в інтерфейсі, є бірюзовий та чорний. Бірюзовий колір, який є відтінком блакитного з зеленуватим відливом, застосовується для навігаційної панелі, заголовків та елементів акцентування. Цей колір асоціюється зі свіжістю, спокоєм та довірою, що є доречним для

сервісу доставки. Чорний колір використовується для фону деяких елементів, таких як категорії та вікно входу, представленим на рис. 2.5, надаючи їм виразності та контрастності.

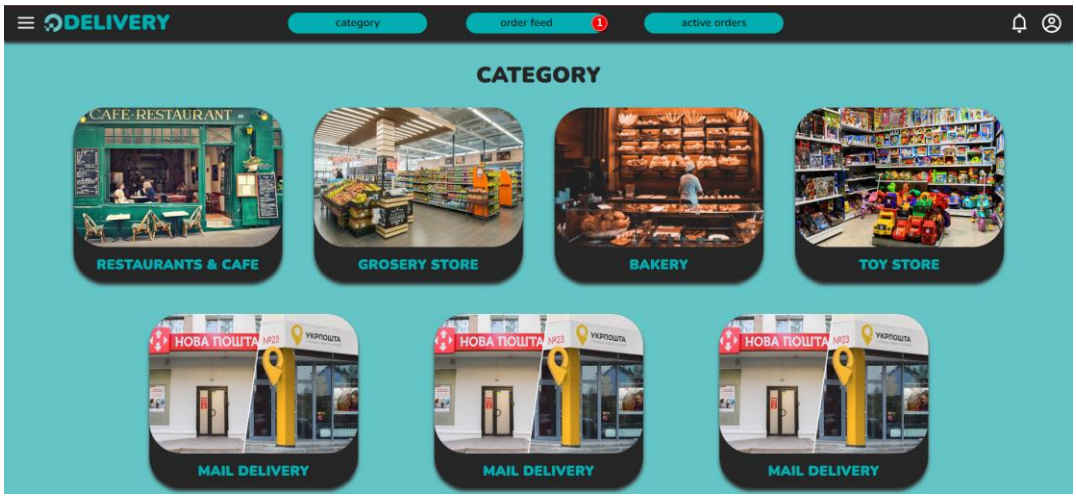

Рисунок 2.4 – Головна сторінка вибору категорій замовлення

На головній сторінці представлено розділ «Category», який дозволяє користувачам фільтрувати стрічку замовлень за обраною категорією товарів. Кожна категорія супроводжується відповідним візуальним зображенням, що допомагає швидко ідентифікувати потрібний тип закладу чи товару. Категорії відображаються у вигляді привабливих плиткових елементів з округленими кутами та підписами, полегшуючи навігацію та вибір бажаної категорії для фільтрації замовлень. На рис. 2.6. представлене спливаюче вікно, де присутнє меню входу для користувачів. Це вікно має класичний дизайн з полями для введення імені користувача та пароля, а також кнопкою входу. Темний колір вікна контрастує з бірюзовим фоном розділу, привертаючи увагу користувача.

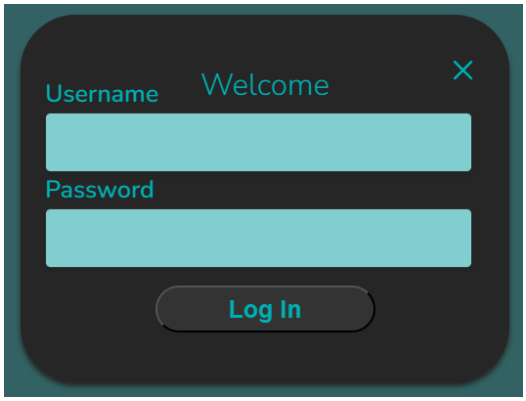

Рисунок 2.5 – Спливаюче вікно входу в обліковий запис користувача

На рисунку 2.6 зображена форма створення замовлення для адміністратора. У верхній частині форми розташоване текстове поле «place name», призначене для введення назви місця, з якого користувач бажає зробити замовлення. Під ним міститься поле «address of the place» для вказання повної адреси цього місця розташування, що забезпечує зручність для доставки. Далі йде поле «delivery addres», де необхідно ввести адресу, на яку буде доставлено замовлення.

Наступним є поле «The cost of the order», де потрібно внести вартість самого замовлення без урахування додаткових зборів. Поруч з ним розміщене поле «The total cost», яке автоматично відображатиме загальну суму замовлення з урахуванням вартості доставки та інших можливих платежів. Таким чином, користувач завжди буде проінформований про кінцеву суму до сплати.

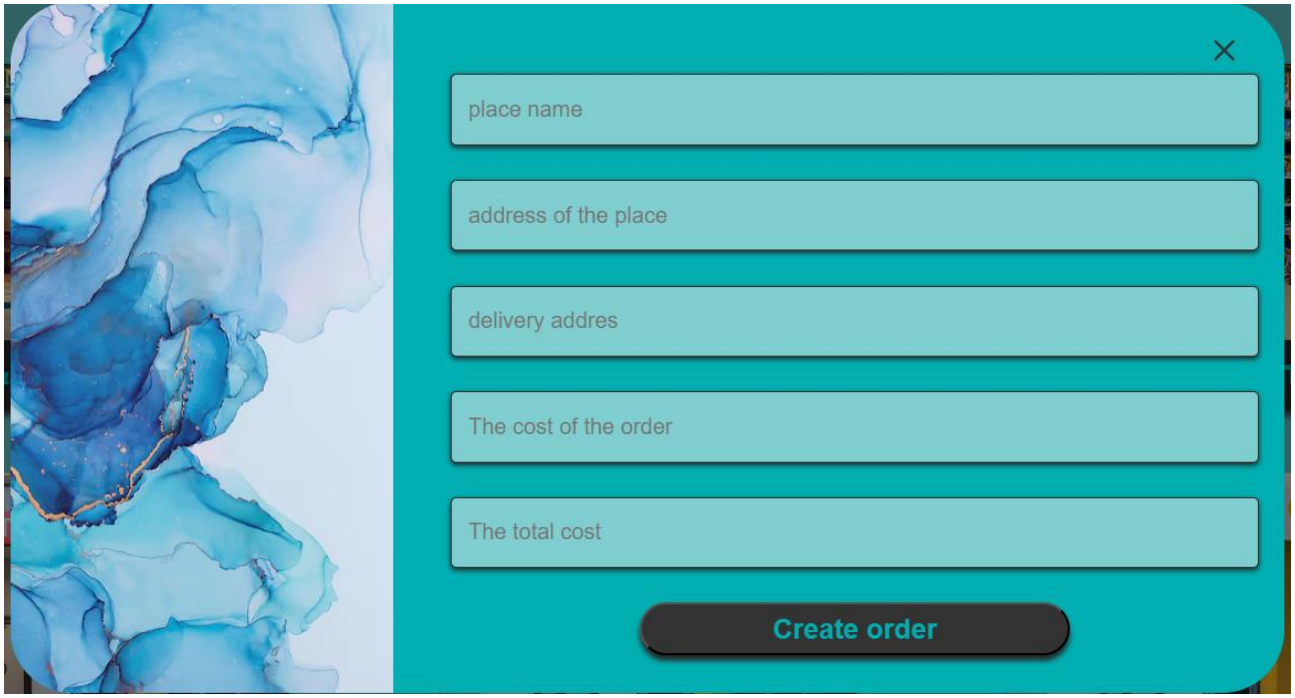

Рисунок 2.6 – Форма створення замовлення для адміністратора

2.3. Розробка користувацького інтерфейсу кур’єрської доставки

У нижній частині форми знаходиться велика кнопка темно-зеленого кольору з написом «Create order». Натиснувши її після заповнення всіх необхідних полів, користувач зможе підтвердити та відправити своє замовлення. Ліворуч від форми розташоване абстрактне зображення синіх розпливчастих

форм, що візуально привертає увагу та допомагає розділу створення замовлень виділятися серед іншого контенту на сторінці.

На рисунку 2.7 представлено користувацький профіль. У верхній частині відображаються дві кнопки, перша відповідає за вихід з профілю користувача, друга в свою чергу відповідає за вихід з облікового запису. Чітко відображається ім’я користувача «admin» великими літерами. Під ним зазначено ADMIN, що явно вказує на роль або статус цього користувача як адміністратора системи.

Нижче розміщено три інформаційні блоки з конкретними даними профілю. Перший блок містить число «16» та підпис «Number of completed orders», який безпосередньо демонструє кількість замовлень, успішно виконаних цим користувачем-адміністратором.

Другий блок відображає цифру «0» поряд з написом «Average rating», що показує середній рейтинг, наданий цьому адміністратору, очевидно, іншими користувачами або клієнтами сервісу.

У третьому блоці міститься текст «some text» та «Some description» під ним. Тут передбачено певний опис або додаткова інформація, пов’язана з профілем адміністратора, хоч її конкретний зміст не уточнюється.

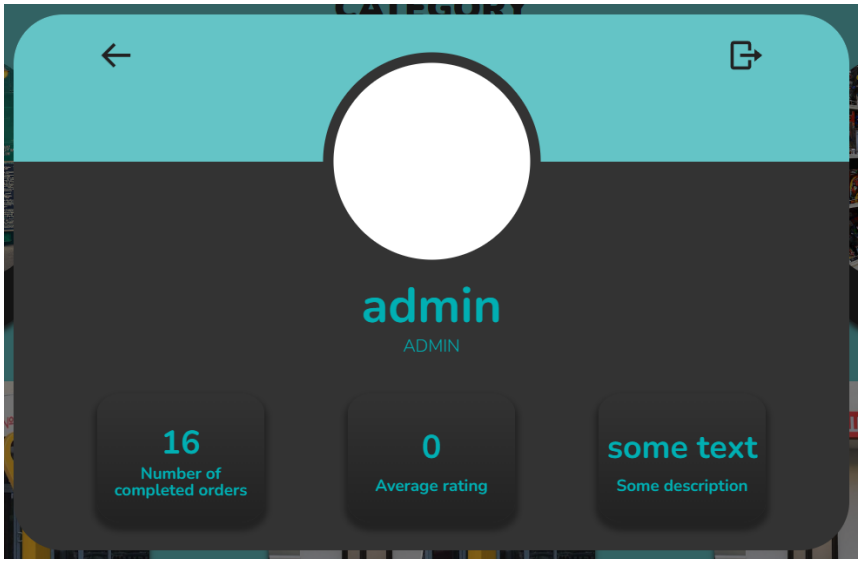

Рисунок 2.7 – Профіль користувача з додатковою інформацією та функціоналом

Загалом, ця сторінка надає повний огляд профілю користувача з іменем «admin», який має адміністративні повноваження, і демонструє ключові

показники його діяльності, зокрема кількість виконаних замовлень, загальний рейтинг та деякі супровідні відомості.

На рисунку 2.8 наведено інтерфейс вкладки «Order Feed». Якщо замовлень немає або користувач не авторизований, під назвою розділу виводиться повідомлення «Not single order yet» показуючи що стрічка замовлень порожня.

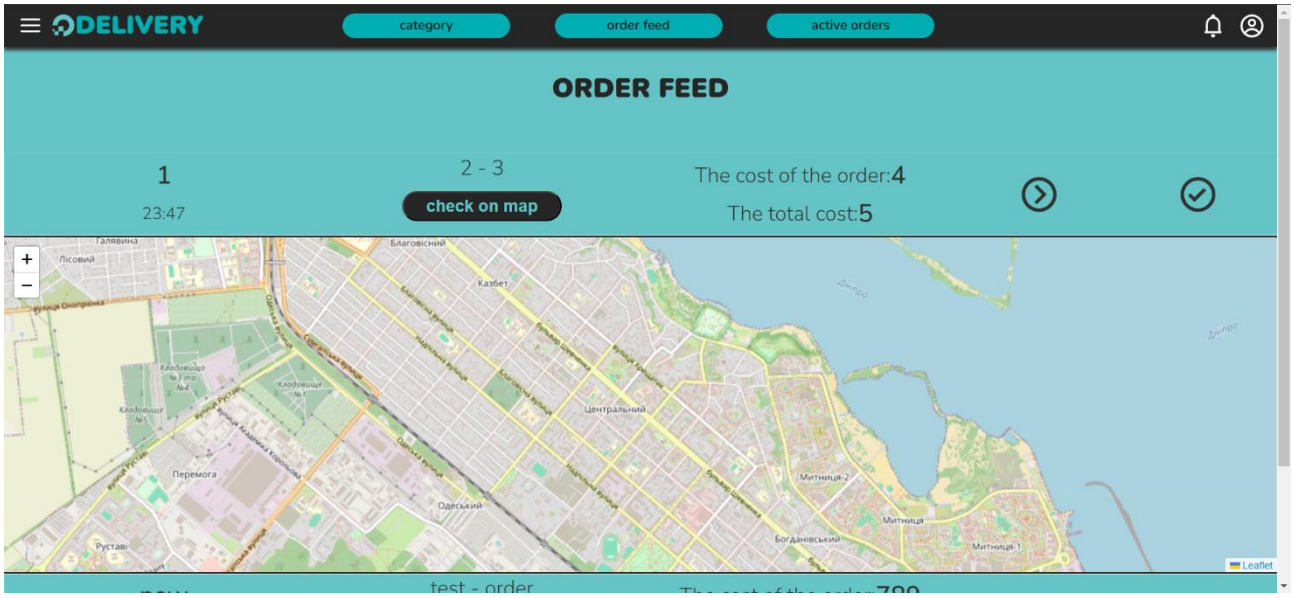

Рисунок 2.8 – Розділ “Order Feed” з замовленнями та мінімапою

Нижче розташовані власне замовлення. Тіло замовлення складається з назви закладу чи іншого місця з якого треба доставити товар, час публікації замовлення, адреса місця замовлення, адреса замовника, а також пункти «The cost of the order: 4» та «The total cost: 5», що відображає вартість замовлення та загальну вартість з урахуванням доставки відповідно. Також у замовленні наявні три кнопки: перша відкриває карту в окремій вкладці з прокладеним маршрутом, друга кнопка розкриває мінімапу під замовленням для ознайомлення приблизного маршруту, третя кнопка – кнопка підтвердження замовлення. При натисканні на цю кнопку кур’єр приймає замовлення і воно переходить до його активних замовлень.

Праворуч від карти наведені подробиці замовлення, під якими присутні іконки у вигляді стрілки та галочки, які можуть вказувати на статус замовлення, наприклад, у процесі доставки чи виконане. У нижній частині інтерфейсу розміщено ще одне замовлення «test – order» із зазначенням його вартості 790.

Загалом, цей інтерфейс дозволяє відстежувати активні замовлення для доставки, їх місцезнаходження, вартість та поточний статус виконання в реальному часі.

На рисунку 2.9 сторінка демонструє розділ «Active Orders» в додатку для доставки «Delivery». Тут відображаються замовлення, які кур'єри вже прийняли до виконання. Наразі показане одне активне замовлення.

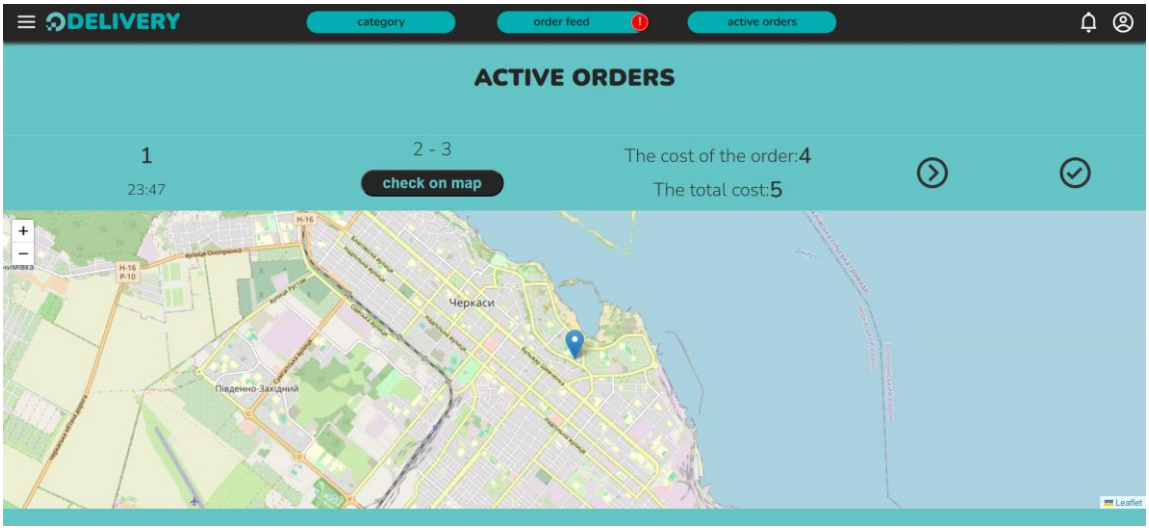

Рисунок 2.9 – Розділ «Active Orders» з активними замовленнями кур'єра

Коли кур'єр приймає нове замовлення, воно з'явиться в цьому розділі з відповідними подробицями, часом та можливістю перегляду на карті. Це допомагає кур'єрам легко відстежувати всі активні замовлення, які вони мають доставити в певний момент часу.

2.4. Реалізація серверної частини

Серверна частина застосунку використовує REST API для взаємодії між клієнтом і сервером. REST (Representational State Transfer) є архітектурним підходом для побудови API, який визначає набір правил та протоколів для обміну даними між клієнтом та сервером.

Для зберігання даних використовується NoSQL база даних MongoDB. MongoDB є документо-орієнтованою базою даних, яка зберігає дані у форматі JSON-подібних документів. Структура побудованої бази наведена на рис. 2.10.

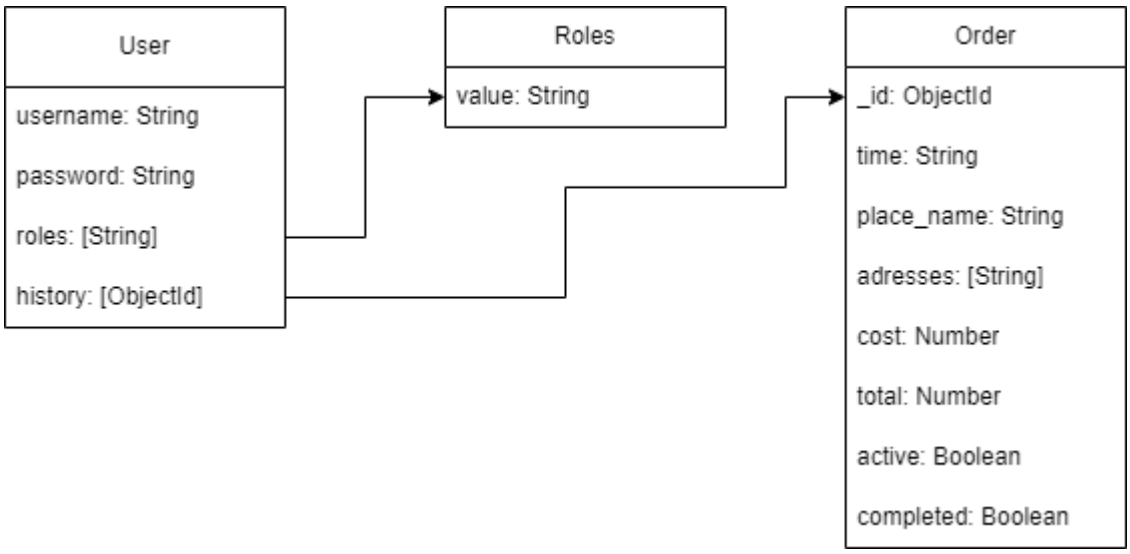

Рисунок 2.10 – Структура бази даних

Лістинг 2.1 демонструє уривок коду схеми User, призначений для представлення користувачів. Схема User визначає структуру документів користувачів. Всередині містяться такі поля як:

– username – ім’я користувача яке не може повторюватись;

– password – пароль користувача;

– roles – масив значень в яких записуються ролі користувача які отримуються з іншої таблиці;

– history – поле в яке передаються унікальні ідентифікатори замовлень, які записуються в історію замовлень користувача.

Лістинг 2.1 – Код схеми User для представлення користувачів

```
const {Schema, model} = require('mongoose')

const User = new Schema({
   username: {type:String, unique:true, required:true},
   password: {type:String, required:true},
   roles:[{type: String, ref:'Role'}],
   history:[{type:Schema.Types.ObjectId , ref :'Order'}]
})
module.exports = model('User', User)
```

Схема Roles визначає тільки одне поле roles, яке є масивом ролей користувача, наприклад «USER» чи «ADMIN». Схема Order визначає структуру замовлення. Поля схеми:

– time – час створення нового замовлення;

– place_name – назва місця/закладу з якого треба доставити товар;

– addresses – масив двох адрес: перша є адресою місця/закладу, друга – адреса замовника;

– cost – вартість замовлення без урахування кур’єрський послуг;

– total – загальна сума з урахуванням всіх інших витрат;

– active – поле, яке приймає значення true або false та відображає статус замовлення (чи воно знаходиться в черзі, чи прийняте кур’єром);

– completed – так само як і поле active є показником статусу замовлення (чи виконується кур’єром чи вже доставлене).

За допомогою Mongoose можна налаштовувати структуру даних у MongoDB, встановлювати правила перевірки на валідацію, визначати зв’язки між різними колекціями та легко взаємодіяти з базою даних через зручний API. Відповідний уривок коду наведено в лістингу 2.2.

Лістинг 2.2 – Код схеми Order для представлення замовлень

```
const {Schema, model} = require('mongoose')

const Order = new Schema({
   time: {type:String, default:new Date},
   place_name: {type:String, },
   adresses:{type:Array},
   cost:{type:Number},
   total:{type:Number},
   category:{type:String},
   active:{type:Boolean},
   completed:{type:Boolean}

})
module.exports = model('Order', Order)
```

Код методу registration() представляє контролер автентифікації, який реалізує функцію реєстрації нового користувача в вебзастосунку. Головні аспекти:

– валідація вхідних даних для запобігання помилкам;

– перевірка наявності користувача з таким самим ім’ям для уникнення дублікатів;

– гешування пароля користувача з використанням bcrypt для підвищення безпеки;

– збереження нового користувача в базі даних MongoDB за допомогою Mongoose;

– повернення відповідного HTTP-відгуку про успішну реєстрацію.

Загалом, код демонструє належну структуру, використовує сучасні бібліотеки (Express, Mongoose, bcrypt) та дотримується кращих практик безпечної обробки даних користувача та взаємодії з базою даних. Уривок коду, що реалізує контролер авторизації користувача, наведено в лістингу 2.3.

Лістинг 2.3 – Код контролера авторизації користувача

```
const User = require('./models/User')
const Role = require('./models/Role')
const bcrypt = require('bcryptjs')
const {validationResult} = require('express-validator')
const jwt = require('jsonwebtoken')
const {secret} = require('./config')

const generateAccessToken=({_id, username, roles, history})=>{
   const payload = {
      _id,
      username,
      roles,
      history
   }
   return jwt.sign(payload, secret, {expiresIn: "24h"})
}
class authController{
   async registration(req, res){
      try {
         const errors = validationResult(req)
         if(!errors.isEmpty()){
            return res.status(400).json({message:"Registration error", errors})
         }
         const {username, password} = req.body
         const candidate = await User.findOne({username})
         if(candidate){
            return res.status(400).json({massage: 'User with this username already exists'})
         }
         const hashPassword = bcrypt.hashSync(password, 7)
         const userRole = await Role.findOne({value:"USER"})
         const user = await User.create({username, password:hashPassword, roles:[userRole.value]})
         return res.status(200).json({message: 'New user sucsessfuly created!'})
      } catch (error) {
         console.log(error)
         res.status(400).json({massage: 'Registration error'})
```

```
    }
}
async login(req, res){
    try {
        const {username, password} = req.body
        const user = await User.findOne({username})
        if(!user){
            return res.status(400).json({message:`User with name: ${username} not found`})
        }
        const validPassword = bcrypt.compareSync(password, user.password)
        if (!validPassword){
            return res.status(400).json({message:"Incorect username or password"})
        }
        // console.log(user.toObject())
        const token = generateAccessToken(user)
        return res.json({token})

    } catch (error) {
        console.log(error)
    res.status(400).json({massage: 'Login error'})
    }
}
async getUsers(req, res){
    try {
        const users = await User.find()
        res.json(users)
    } catch (error) {

    }
}
async check(req, res) {
    try {
        // console.log(req, res)
        const token = generateAccessToken(req.user);
        return res.status(200).json({ token });
    } catch (error) {
        return res.status(500);
    }
}
async toActiveOrder(req,res){
    try {
        if (!req.user) {
            return res.status(401).json({ message: 'Unauthorized access' });
          }
        const userId = req.user._id;
        const user = await User.findById(userId)
        if (!user) {
            return res.status(404).json({ message: 'User not found' });
        }
        user.history.push(req.body._id)
        user.save()
        // console.log(req.body._id , user)
```

```
            const token = generateAccessToken(user)
            return res.json({token})
        } catch (error) {

        }
    }

module.exports = new authController()
```

Метод LogIn() отримує дані користувача (ім’я користувача і пароль) з тіла HTTP-запиту. Далі перевіряється наявність користувача з таким ім’ям у базі даних за допомогою методу findOne() з моделі User. Якщо користувача не знайдено, повертається помилка з кодом 400. Якщо користувач існує, його введений пароль порівнюється з захешованим паролем у базі даних за допомогою функції bcrypt.compareSync(). Якщо паролі не збігаються, повертається помилка з кодом 400.

Якщо ім’я користувача та пароль вірні, генерується JSON Web Token (JWT) за допомогою функції generateAccessToken(), що дає змогу автентифікувати користувача і зберігати інформацію про його сесію. Згенерований токен повертається в HTTP-відгуку в форматі JSON.

У разі виникнення будь-якої помилки під час виконання коду, вона обробляється у блоці catch, і повертається відповідь з кодом 400 та повідомленням про помилку входу. Наведений код демонструє безпечну обробку даних користувача, перевірку автентифікації, використання JSON Web Token для збереження інформації про сесію та належну обробку помилок у веб-застосунку.

Також у коді присутній відносно невеликий метод check(), мета якого перевірити наявність JWT-токена та провести повторну авторизацію при необхідності, наприклад при перезавантаженні сторінки.

Метод toActiveOrder() є асинхронною функцією, яка обробляє запити на активацію замовлення в вебзастосунку. Спочатку перевіряється, чи користувач є автентифікованим. Якщо req.user не існує, повертається відповідь з кодом 401 (Unauthorized access). Інакше отримується ідентифікатор користувача (userId) з req.user.id. Функція знаходить користувача в базі даних за його ідентифікатором

за допомогою методу findById() з моделі User. Якщо користувача не знайдено, повертається відповідь з кодом 400 (User not found). Якщо користувач знайдений, ідентифікатор замовлення (req.body.id) додається до масиву history у документі користувача. Оновлений документ користувача зберігається в базі даних за допомогою методу save(). У разі виникнення будь-якої помилки під час виконання коду, вона обробляється у блоці catch.

Наведений код демонструє перевірку автентифікації користувача, пошук та оновлення документа користувача в базі даних, використання JSON Web Token для автентифікації користувача та належну обробку помилок у вебзастосунку.

Також у додатку присутній контролер для роботи з власне замовленнями. В orderController наявні методи:

– createOrder() отримує дані замовлення з тіла HTTP-запиту (req.body.order) та створює новий документ замовлення в базі даних за допомогою методу Order.create(order). Якщо замовлення успішно створено, повертає відповідь з кодом 200 і повідомленням «Order created». У разі виникнення помилки повертає відповідь з кодом 500 і повідомленням «Internal server error»;

– getOrders() знаходить всі замовлення в базі даних, де active і completed мають значення false, за допомогою методу Order.find(). Метод повертає масив знайдених замовлень у форматі JSON. Якщо виникає помилка, повертає порожній масив;

– acceptOrder() знаходить замовлення за ідентифікатором req.body._id за допомогою методу Order.findOneAndUpdate(). Він оновлює поле active на true для знайденого замовлення та повертає ідентифікатор оновленого замовлення у форматі JSON;

– completeOrder() є частиною контролера для обробки замовлень у вебзастосунку. Він знаходить замовлення за ідентифікатором req.body._id за допомогою методу Order.findOneAndUpdate(), оновлює поле completed на true для знайденого замовлення та повертає ідентифікатор оновленого замовлення у форматі JSON;

– getActiveOrders() отримує масив ідентифікаторів замовлень з req.query.arr, знаходить замовлення, де _id присутній в отриманому масиві та completed має значення false, за допомогою методу Order.find(). Метод повертає масив знайдених активних замовлень у форматі JSON.

Вище згаданий код використовує модель Order з файлу ./models/Order для взаємодії з базою даних замовлень. Він надає функціональність для створення, отримання, прийняття та завершення замовлень, а також для отримання активних замовлень за масивом ідентифікаторів. Також в додатку використовується технологія проміжного програмного забезпечення (middleware, лістинг 2.4) для відслідковування дій неавторизованих користувачів чи користувачів з меншим рівнем допуску.

Лістинг 2.4 – Код методу authMiddleware для валідації авторизованих дій

```
const jwt = require('jsonwebtoken')
const {secret}= require('../config')
module.exports = function(req,res,next){
   if (req.method ==="OPTIONS"){
      next()
   }
   try {
      if (!req.headers || !req.headers.authorization) {
         return res.status(401).json({ message: 'Unauthorized' }); // Use 401 for missing
authorization
       }
      const token = req.headers.authorization.split(' ')[1]
      if (!token){
         return res.status(400).json({message:"User is not authorized"})
      }
      const decodedData = jwt.verify(token, secret)
      req.user = decodedData

      next()
   } catch (error) {
      console.log(error)
      return res.status(400).json({massage: "User is not authorized"})
   }
}
```

Реалізація клієнтської частини. У проєкті реалізована клієнтська частина модулю кур’єрської доставки, яка забезпечує зручний інтерфейс для взаємодії

користувачів. Центральною частиною клієнтського коду є сховище даних (store), побудоване з використанням бібліотеки MobX.

MobX — це сучасна бібліотека для управління станом додатків в JavaScript, яка спрощує роботу з реактивними даними. Вона ґрунтується на концепції спостережуваних даних (observables) та обчислюваних значень (computed values). MobX автоматично відстежує залежності між даними та оновлює відповідні частини інтерфейсу при змінах цих даних. Це забезпечує ефективну та надійну роботу з даними в додатку, полегшуючи розробку та підтримку коду.

Використання MobX у клієнтській частині модуля кур'єрської доставки дозволяє ефективно керувати станом додатку, спостерігати за змінами даних та реагувати на них відповідним оновленням інтерфейсу користувача. Ця бібліотека спрощує процес синхронізації даних між клієнтом і сервером, забезпечуючи плавний та зручний досвід користування додатком.

Лістинг 2.5 – Код сховища станів MobX store

```
import { makeAutoObservable, reaction, runInAction } from "mobx";

class Store {
   count = "!";
   PanelState = false;
   CategoryState = true;
   OrderFeedState = false;
   ActiveOrdersState = false;
   LogFormState = false;
   OrderFormState = false;
   ProfilePopState = false;
   OrderFormInputValue = ''

   constructor() {
      this._username = '';
      this._roles = [];
      this._isLogin = false;
      this._Order = {};
      this._OrderList = [];
      this._ActiveOrderList = [];
      this._history = [];
      this._assessment = 0
      makeAutoObservable(this);

      reaction(
         () => this._OrderList.filter(object => object.checked === false).length,
         (length) => {
            runInAction(() => {
```

```
                this.count = length;
            });
            // console.log(this.count);
        }
    );

    reaction(
        () => this.OrderFeedState === true,
        (isOrderFeedOpen) => {
            if (isOrderFeedOpen) {
                runInAction(() => {
                    this._OrderList = this._OrderList.map((order) => ({ ...order, checked: true }));
                });
            }
        }
    );
  }

// гетери та сеттери опущено для зручності читання

  PanelOpen() { this.PanelState = true; }
  PanelClose() { this.PanelState = false; }
  CategoryOpen() { this.CategoryState = true; }
  CategoryClose() { this.CategoryState = false; }
  OrderFeedOpen() { this.OrderFeedState = true; }
  ActiveOrdersOpen() { this.ActiveOrdersState = true; }
  ActiveOrdersClose() { this.ActiveOrdersState = false; }
  OrderFeedClose() { this.OrderFeedState = false; }
  OrderFormOpen() { this.OrderFormState = true; }
  OrderFormClose() { this.OrderFormState = false; }
  LogFormOpen() { this.LogFormState = true; }
  LogFormClose() { this.LogFormState = false; }
  ProfilePopOpen() { this.ProfilePopState = true; }
  ProfilePopClose() { this.ProfilePopState = false; }
  pushToOrderList(value) { this._OrderList.push(value); }
}

export const store = new Store();
```

Код представляє сховище (store) на основі MobX для управління станом та даними вебзастосунку. Сховище описується класом Store, який містить такі властивості:

– count використовується для підрахунку нових, непрочитаних замовлень;

– PanelState, CategoryState, OrderFeedState, ActiveOrdersState, LogFormState, OrderFormState, ProfilePopState набувають булевих значень, що

визначають стан різних компонентів інтерфейсу користувача (відкритий/ закритий);

– _username – ім’я користувача;

– _roles – масив ролей користувача;

– _isLogin – булеве значення, яке вказує, чи користувач увійшов у систему;

– _Order – об’єкт, що представляє замовлення;

– _OrderList – масив об’єктів замовлень;

– _ActiveOrderList – масив активних замовлень;

– _history – масив історії;

– _assessment – оцінка.

Також наявні такі методи:

– геттери/сеттери для властивостей сховища;

– PanelOpen(), PanelClose(), CategoryOpen(), CategoryClose(), OrderFeedOpen(), ActiveOrdersOpen(), ActiveOrdersClose(), OrderFeedClose(), OrderFormOpen(), OrderFormClose(), LogFormOpen(), LogFormClose(), ProfilePopOpen(), ProfilePopClose() – методи для зміни стану різних компонентів інтерфейсу користувача;

– pushToOrderList(value) – метод для додавання нового замовлення до _OrderList.

Крім того, у конструкторі класу Store встановлюються початкові значення для деяких властивостей та ініціалізується MobX за допомогою makeAutoObservable.

Також у конструкторі встановлюються дві реакції (reactions) за допомогою MobX. Перша реакція відстежує зміни в _OrderList, фільтруючи об’єкти замовлень, де checked дорівнює false. Кількість таких об’єктів зберігається у властивості count. Друга реакція спрацьовує, коли OrderFeedState стає true. У цьому випадку всі об’єкти замовлень у _OrderList позначаються як checked: true.

Наведений код демонструє використання MobX для управління станом застосунку, включаючи відстеження змін даних та реагування на них, а також надає методи для взаємодії з цими даними та станом інтерфейсу користувача.

У роботі використовується бібліотека Axios, яка є популярним засобом для здійснення HTTP-запитів у JavaScript-додатках. Axios забезпечує зручний спосіб взаємодії з серверною частиною вашого додатку через HTTP-запити. Він використовується для отримання даних з сервера, надсилання даних на сервер, а також для виконання різних операцій, таких як створення, оновлення, видалення та читання даних. Axios використовується для здійснення таких операцій, як отримання списку замовлень, створення нового замовлення, оновлення статусу замовлення (наприклад, позначення замовлення як активного або завершеного) тощо.

Сховище на основі MobX зберігає дані, отримані за допомогою Axios, та керує станом додатку на основі цих даних. Наприклад, коли ви отримуєте список замовлень за допомогою Axios, сховище зберігає цей список у властивості _OrderList. Потім, коли ви відкриваєте компонент для перегляду списку замовлень, він використовує дані з _OrderList для відображення замовлень у інтерфейсі користувача.

Взаємодія між Axios та MobX дозволяє ефективно керувати даними та станом додатку, забезпечуючи плавний та зручний досвід користування. Axios відповідає за отримання та надсилання даних на сервер, тоді як MobX керує станом додатку на основі цих даних та реагує на зміни, автоматично оновлюючи інтерфейс користувача.

За результатами виконання другого розділу кваліфікаційної роботи спроєктовано та реалізовано прототип програмного модуля з функціональністю управління кур’єрськими доставками. Сформоване програмне забезпечення тестуватиметься в наступному розділі.

РОЗДІЛ 3
ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

Тестування є надзвичайно важливим етапом у процесі розробки програмного забезпечення, оскільки воно дозволяє перевірити функціональність, продуктивність та стабільність додатку. Для служби кур'єрської доставки ретельне тестування відіграє критичну роль у забезпеченні безперебійної роботи та своєчасної й коректної доставки товарів.

Цей розділ детально розглядає процес тестування розробленого застосунку. У ньому будуть охоплені різні аспекти тестування, що застосовувалися для перевірки функціональності, продуктивності та безпеки застосунку. На початку розділу буде представлено огляд різних видів тестування, які були використані, таких як модульне тестування, тестування API та тестування користувацького інтерфейсу.

3.1 Вибір методів тестування та формування тест-плану

Програмний засіб Postman є потужним, ключовим інструментом для тестування та налагодження REST API. Його інтерфейс та спосіб формування запитів дають змогу легко взаємодіяти з API, надсилаючи HTTP-запити різних типів та аналізуючи відповіді сервера. Postman забезпечує гнучкість у налаштуванні заголовків, тіла запиту, параметрів та інших деталей, що є критично важливим для всебічного тестування API. Крім того, можливість зберігати та організовувати колекції запитів значно полегшує повторне використання та підтримку тестів протягом усього життєвого циклу розробки.

Однією з ключових переваг Postman є підтримка автоматизації тестування через створення тестових сценаріїв та їхній запуск у спеціальному середовищі Postman Runner. Це дає можливість швидко та ефективно виконувати комплексні тести, перевіряючи різноманітні сценарії використання API. Postman тісно інтегрується з популярними інструментами неперервної інтеграції та неперервного розгортання, такими як Jenkins, Travis CI тощо, що спрощує включення автоматизованого тестування API у загальний процес розробки

програмного забезпечення. Завдяки цим потужним можливостям Postman став незамінним інструментом для розробників REST API в усьому світі.

Postman – це настільна програма, доступна для Windows, macOS та Linux, а також веб-версія для роботи з будь-якого пристрою з підключенням до Інтернету. Ключовою перевагою є можливість створювати та зберігати колекції запитів - організовані групи запитів для зручного повторного використання та обміну між розробниками [18].

Перегляд відповідей від сервера, підтримка перегляду заголовків відповіді та кодів статусу полегшує аналіз даних та пошук помилок. Під час тестування API модулю служби доставки за допомогою Postman було створено колекцію запитів, що охоплювала різні функції, такі як авторизація користувачів, створення та прийняття замовлень, отримання замовлень для авторизованих користувачів, тощо. Ці запити відправлялися до API, а відповіді аналізувалися на відповідність очікуваним результатам.

Таблиця 3.1 – Тестовий набір

Назва тесту	Очікуваний результат	Наявний результат
Успішна реєстрація користувача	Статус-код 200 OK, повідомлення про успішну реєстрацію	Статус-код 200 OK, повідомлення про успішну реєстрацію
Неуспішна реєстрація користувача (невалідні дані)	Статус-код 400 Bad Request, повідомлення про невалідні дані	Статус-код 400 Bad Request, повідомлення про невалідні дані
Неуспішна реєстрація користувача (вже існуючий користувач)	Статус-код 400 Bad Request, повідомлення про існуючого користувача	Статус-код 400 Bad Request, повідомлення про існуючого користувача
Успішна авторизація	Статус-код 200 OK, JWT токен	Статус-код 200 OK, JWT токен
Неуспішна авторизація	Статус-код 400 Bad Request, повідомлення про неправильні дані	Статус-код 400 Bad Request, повідомлення про неправильні дані

Продовження таблиці 3.1

Створення замовлення з недостатніми правами	Статус-код 403 Forbidden, повідомлення про відмову в доступі	Статус-код 403 Forbidden, повідомлення про відмову в доступі
Успішне створення замовлення	Статус-код 200 OK, повідомлення про успішне створення замовлення	Статус-код 200 OK, повідомлення про успішне створення замовлення
Отримання всіх замовлень	Статус-код 200 OK, масив об'єктів-замовлень	Статус-код 200 OK, масив об'єктів-замовлень
Отримання активних замовлень	Статус-код 200 OK, масив об'єктів активних замовлень	Статус-код 200 OK, масив об'єктів активних замовлень

3.2. Результати проведення тестування за сформованим тест-планом

Результати запуску розроблених API-тестів у середовищі Postman наведено нижче. На рис. 3.1 показано тест успішності реєстрації користувача.

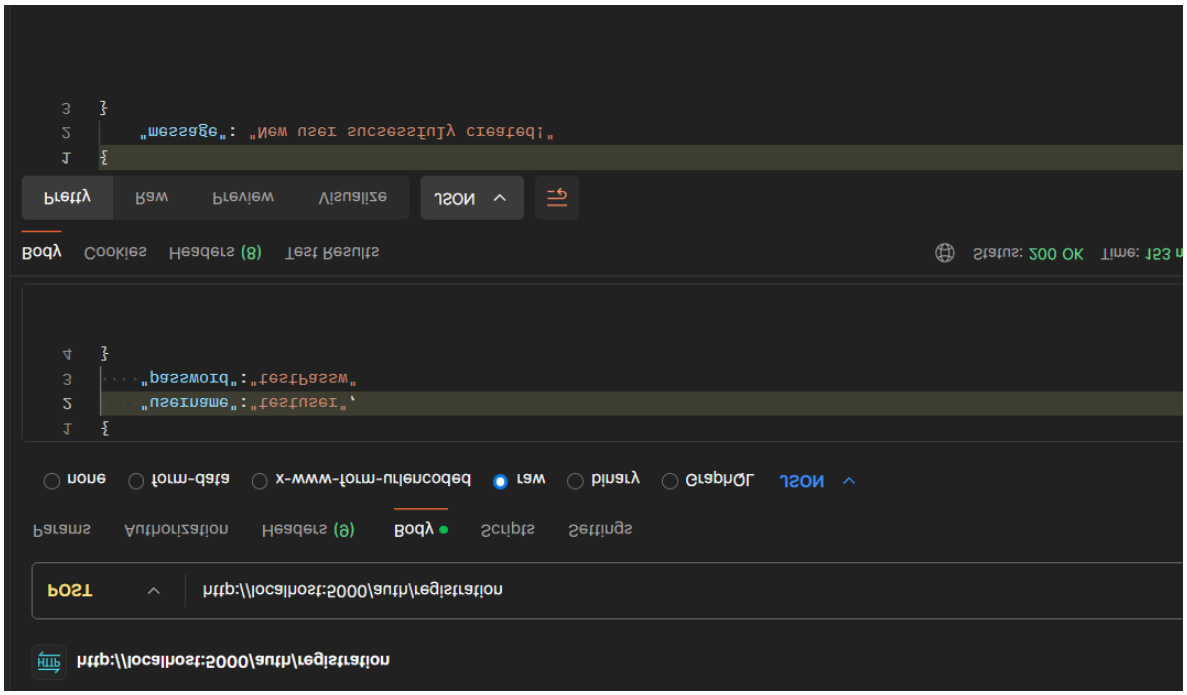

Рисунок 3.1 – Запит Postman для імітації успішної реєстрації користувача

POST-запит на /api/auth/registration імітує процес реєстрації нового користувача в модулі служби доставки. У тілі запиту передаються поля «username» та «password» з відповідними значеннями «testuser» та «testPassw».

Відповідь від сервера має статус-код 200 Ok, що означає успішне створення ресурсу . У відповіді міститься повідомлення про успішну реєстрацію нового користувача.

POST-запит на /api/auth/registration, але із невалідними даними ("username": "testUser", "password": "testPassword"). Сервер повертає статус-код 400 Bad Request та відповідь з повідомленням про те, що введені дані користувача невалідні, а саме поле password має бути від 4 до 10 символів, як видно на рис. 3.2. Це імітує ситуацію, коли користувач намагається зареєструватися з даними, які не проходять валідацію в системі.

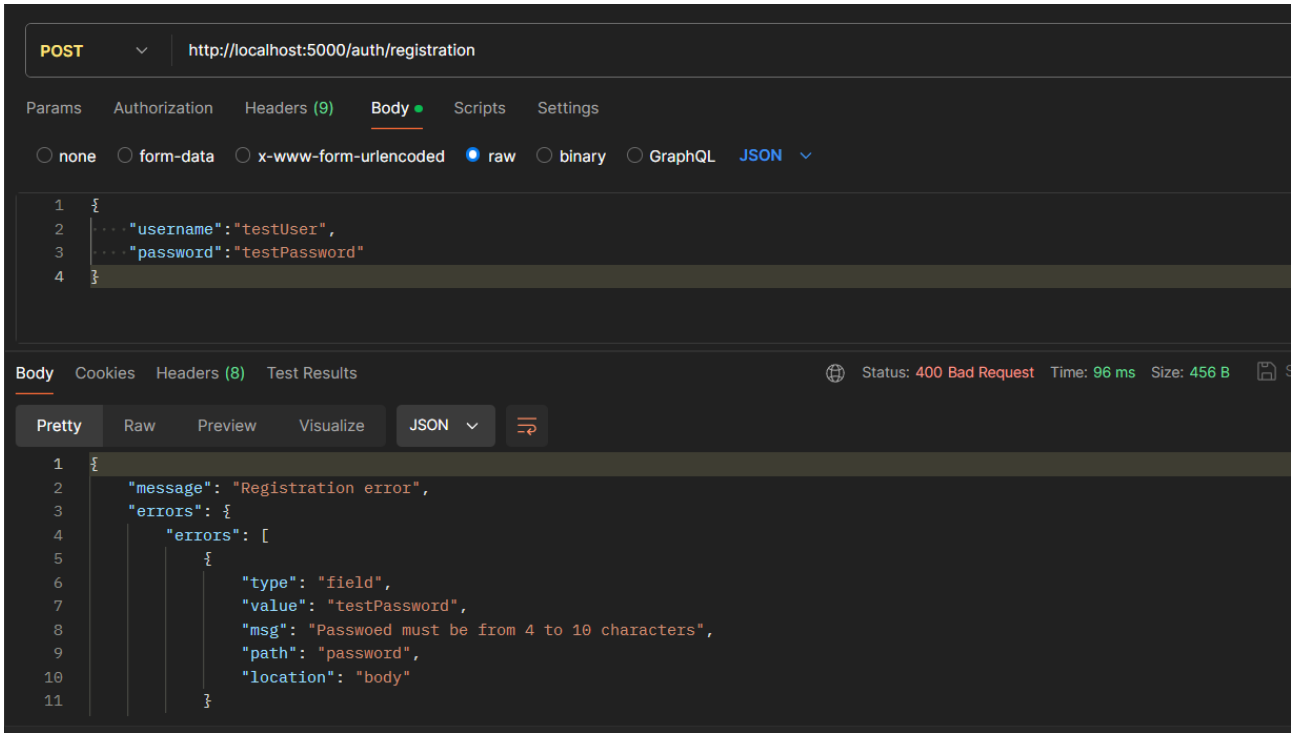

Рисунок 3.2 – Запит Postman для імітації неуспішної реєстрації користувача

POST-запит на /api/ auth/registration, але із даними існуючого користувача ("username": "testUser", "password": "testPasw"). Сервер повертає статус-код 400 Bad Request та відповідь з повідомленням про те, що користувач з таким ім'ям вже існує. Це імітує ситуацію, коли користувач намагається зареєструватися з даними, які вже використовуються в системі. На рис. 3.3 показано запуск відповідного тесту.

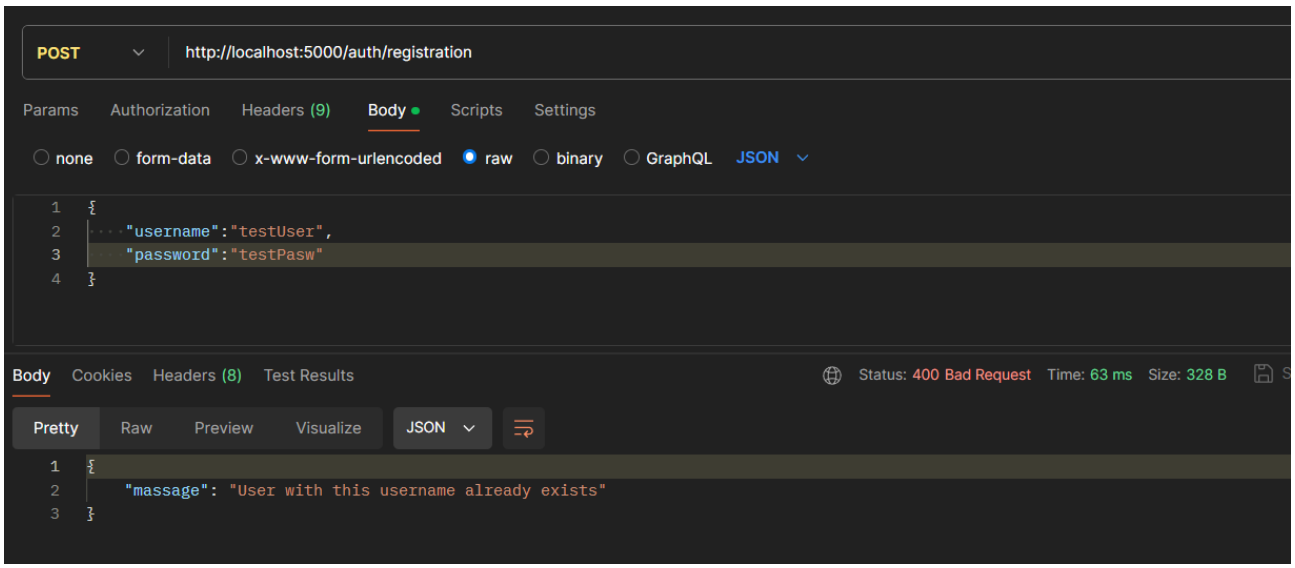

Рисунок 3.3 – Запит Postman для імітації неуспішної реєстрації користувача

POST-запит на /api/auth/login, з даними існуючого користувача ("username": "testUser", "password": "testPasw"). Сервер повертає статус-код 200 OK та відповідь з повідомленням в якому міститься згенерований JWT token кристувача. Це імітує ситуацію, коли користувач намагається авторизуватись, як показано на рис. 3.4.

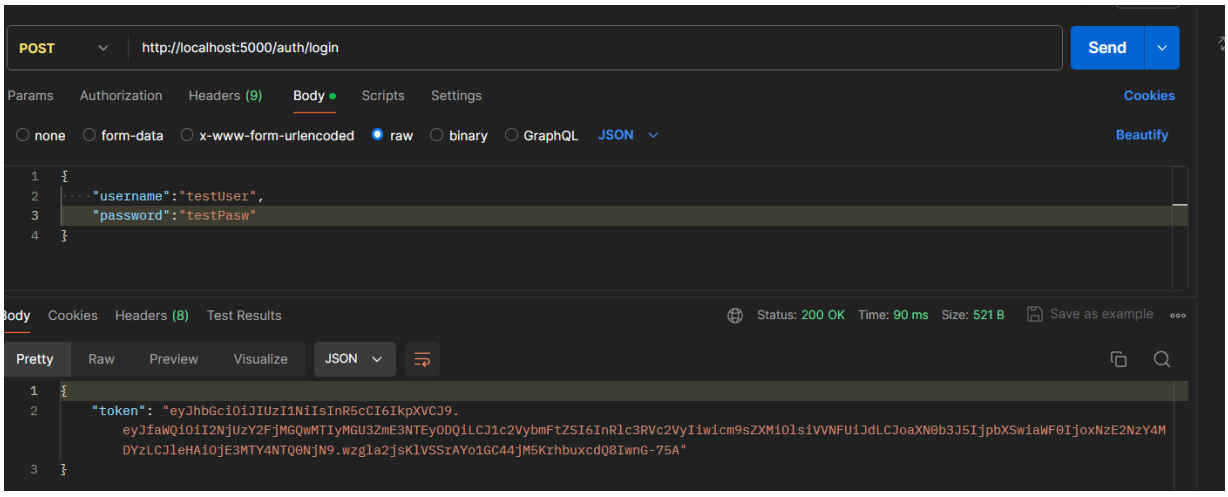

Рисунок 3.4 – Запит Postman для імітації успішної авторизації

POST-запит на /api/auth/login, з даними користувача ("username": "testUser", "password": "testPas"). Сервер повертає статус-код 400 Bad Request та відповідь з повідомленням про неправильність вводу паролю або логіну користувача. Це

імітує ситуацію, коли користувач намагається авторизуватись, але вводить неправильні дані. Рис. 3.5 демонструє цей тест.

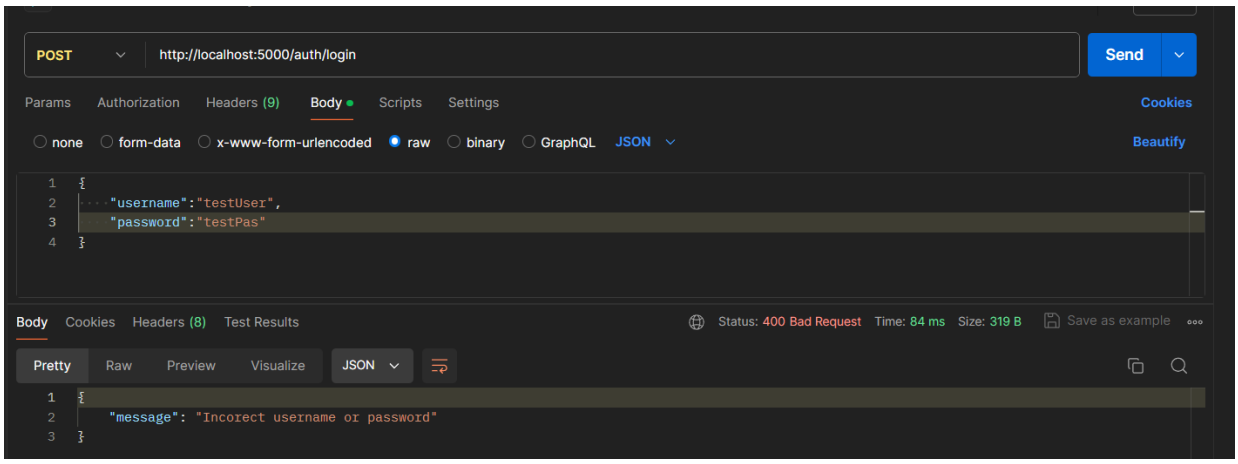

Рисунок 3.5 – Запит Postman для імітації не успішної авторизації

POST-запит на /api/order/createOrder, з header'ом Authorization який включає в себе згенерований в попередньому тесті JWT token з даними користувача ("username": "testUser", "roles":["USER"]). Сервер повертає статус-код 403 Frobidden та відповідь з повідомленням про відмову в доступі. Це імітує ситуацію, коли користувач намагається створити нове замовлення, але не має достатньо прав доступу для цього (рис. 3.6).

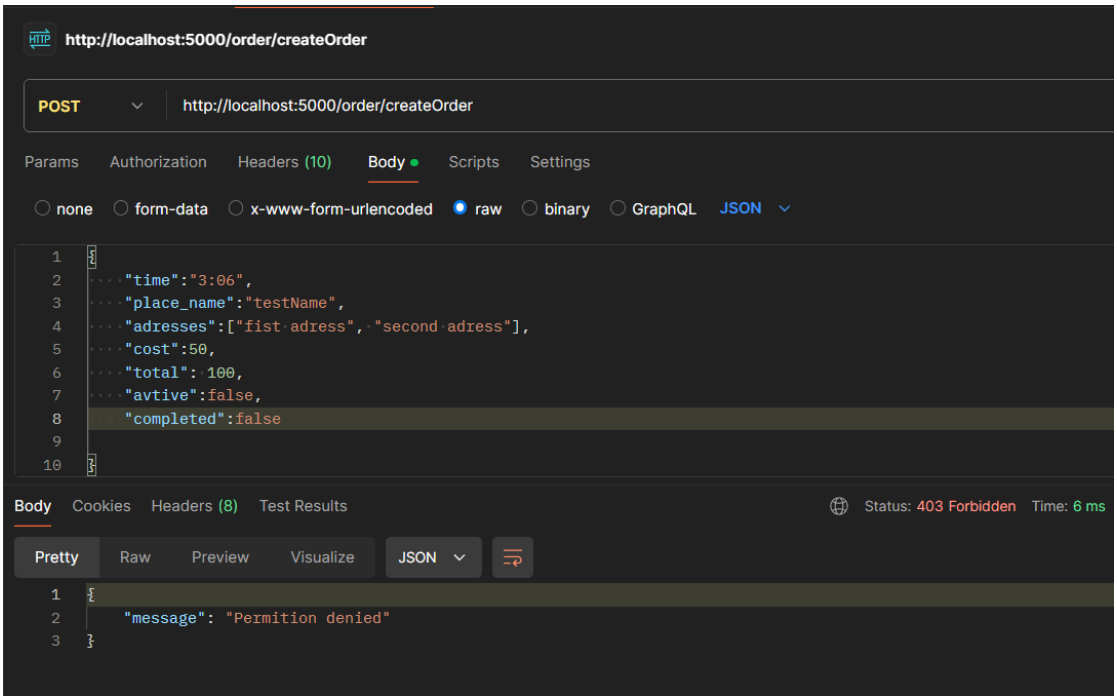

Рисунок 3.6 – Запит Postman для імітації створення нового замовлення з недостатніми правами

POST-запит на /api/order/createOrder, з header’ом Authorization який включає в себе згенерований JWT token з даними користувача ("username": "admin", “roles”:[“ADMIN”]). Сервер повертає статус-код 200 Ok та відповідь з повідомленням про успішне створення нового замовлення. Це імітує ситуацію, коли адміністатор намагається створити нове замовлення.

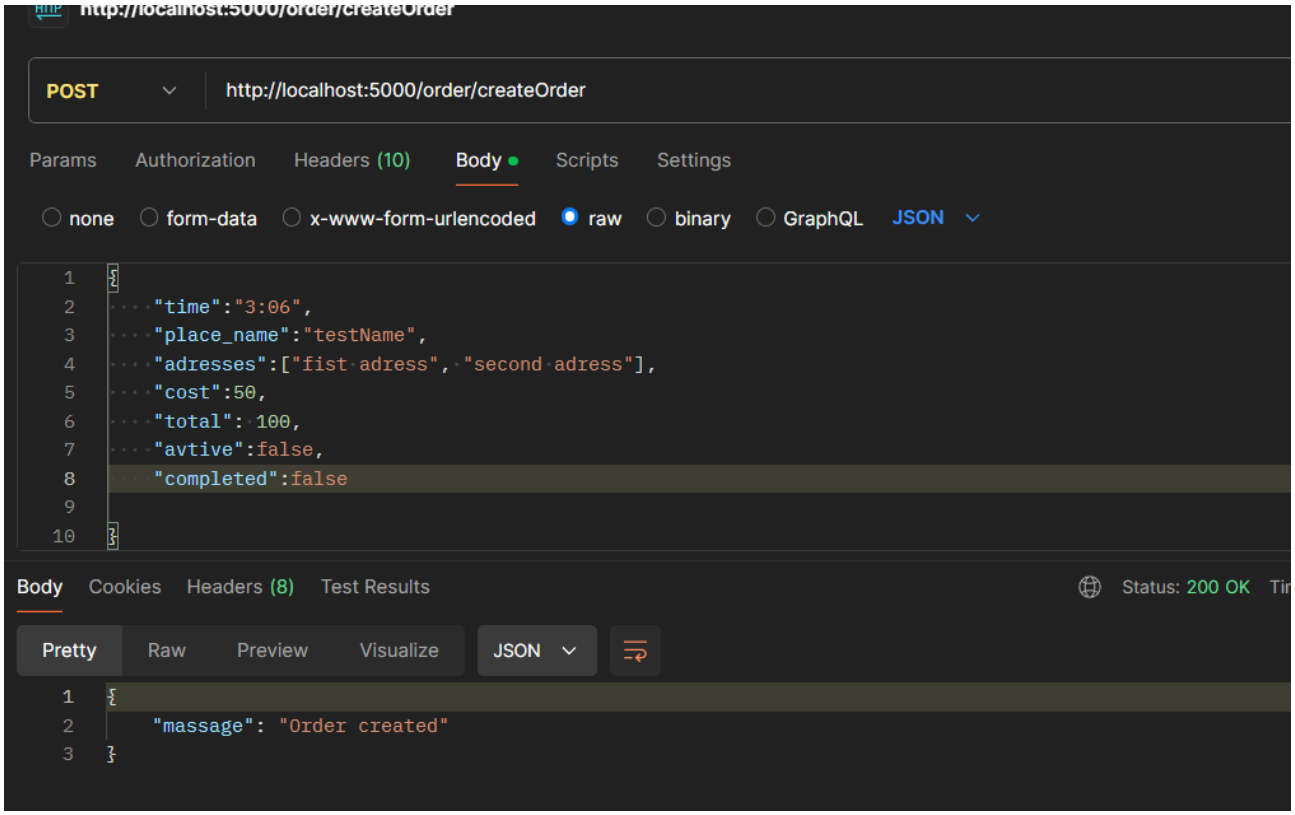

Рисунок 3.7 – Запит Postman для імітації створення нового замовлення

GET-запит на /api/order/getOrders, з header-ом Authorization який включає в себе згенерований JWT token з даними користувача ("username": "admin", “roles”:[“ADMIN”]). Сервер повертає статус-код 200 Ok та відповідь з масивом об’єктів-замовлень. Це імітує ситуацію, коли авторизований користувач намагається отримати всі замовлення. Відповідний тест зображено на рис. 3.8.

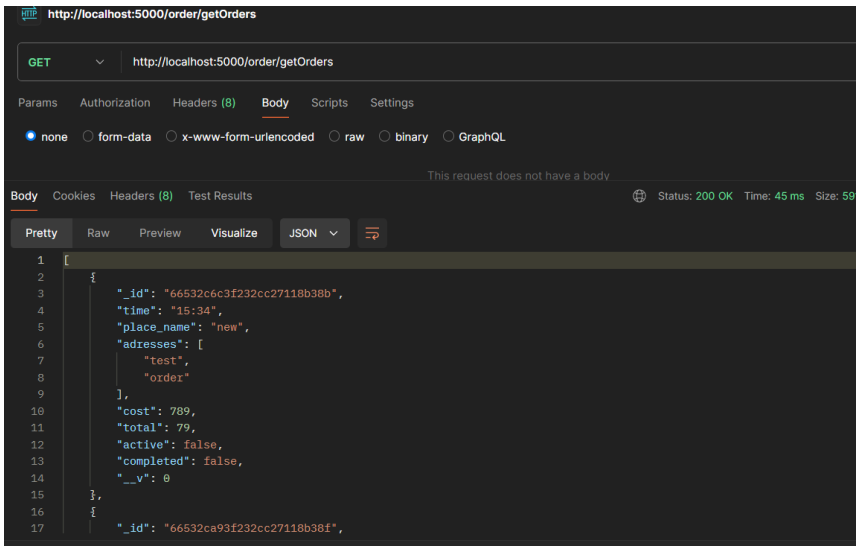

Рисунок 3.8 – Запит Postman для імітації отримання всіх замовлень з бази даних

GET-запит на /api/order/getActiveOrders, з header-ом Authorization, який включає в себе згенерований JWT token з даними користувача ("username": "admin", “roles”:[“ADMIN”]), та параметрами запиту в якому вказані унікальні індитифікатори шуканих замовлень. Сервер повертає статус-код 200 Ok та відповідь з масивом об’єктів-замовлень. Це імітує ситуацію, коли авторизований користувач намагається отримати всі свої активні замовлення. Вказаний тест продемонстровано на рис. 3.9.

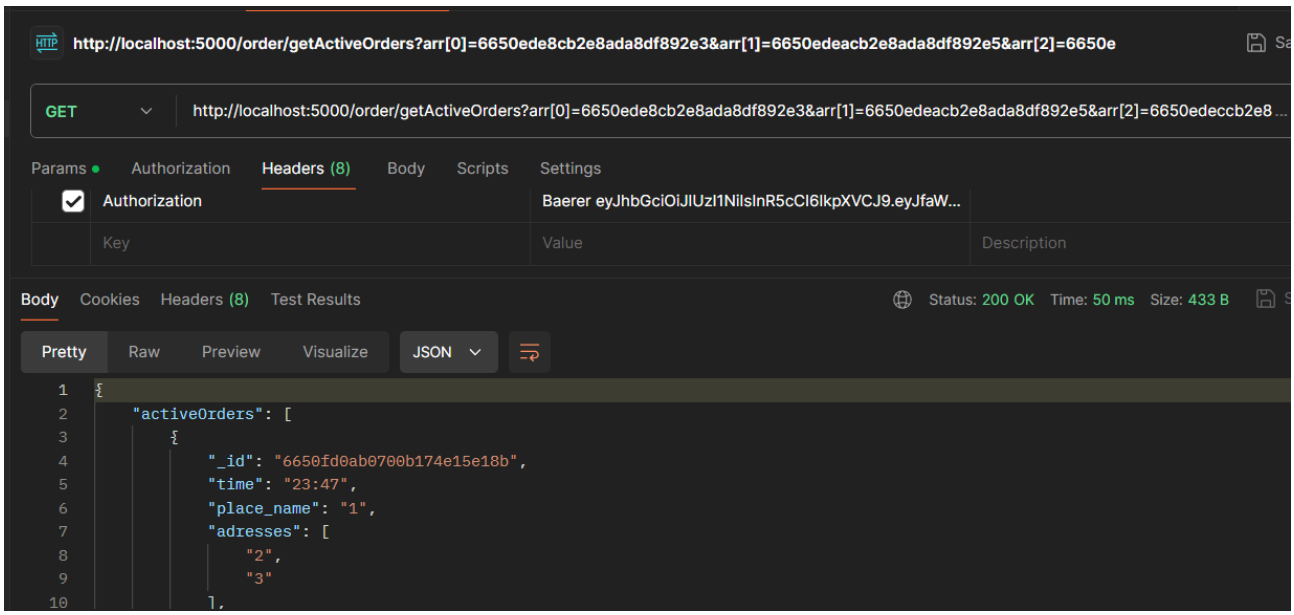

Рисунок 3.9 – Запит Postman для імітації отримання всіх активних замовлень з бази даних

Ці запити є частиною всебічного тестування різноманітних функцій застосунку за допомогою Postman. Вони імітують реалістичні сценарії використання, такі як реєстрація користувачів, створення чатів та завантаження зображень, що дає змогу ретельно перевірити коректність роботи API та відповідність очікуваним результатам.

Використання Postman значно спростило й оптимізувало процес тестування API. Його зручний інтерфейс, підтримка широкого спектру функцій та можливість автоматизації допомогли ефективно виявляти та усувати недоліки на етапі розробки. Завдяки Postman вдалося забезпечити високу якість та стабільність застосунку [18].

3.3. Результати проведення тестування зручності використання

Юзабіліті відіграє ключову роль у забезпеченні позитивного досвіду користувачів та ефективності продукту. Для комплексної оцінки юзабіліті було створено гугл форму для опитування потенційних користувачів, що охопив різні аспекти процесу тестування. Це дозволило краще зрозуміти потреби та очікування користувачів щодо продукту.

Серед застосованих методів були опитування та інтерв'ю для збору відгуків, тестування на основі реалістичних сценаріїв використання.

Для кількісної оцінки успішності юзабіліті були визначені відповідні метрики. Ці метрики дозволили отримати об'єктивні дані для аналізу та вдосконалення продукту.

Результати юзабіліті тестування були ретельно проаналізовані, що дозволило виявити недоліки та можливості для вдосконалення продукту. Звіти про результати тестування представлені нижче. За результатами опитування 86,4% користувачів вважають екран вибору категорій зручним і зрозумілим. Результати представленні на рисунку 3.10.

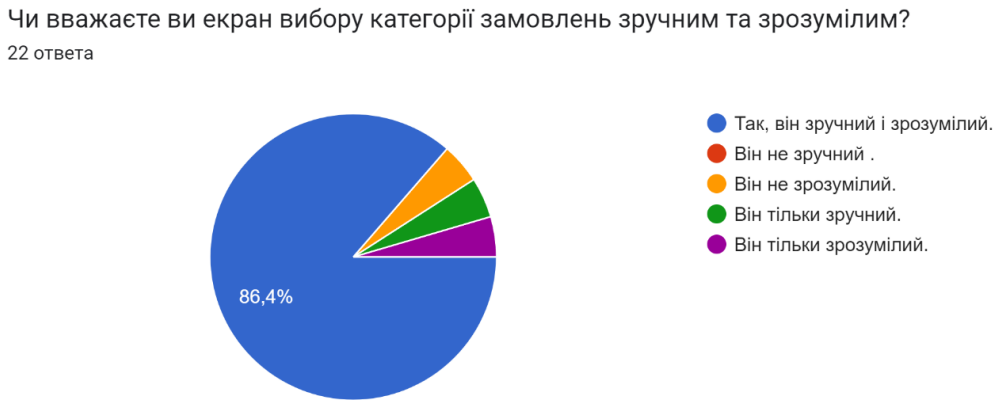

Рисунок 3.10 – Діаграма результатів першого пункту опитування.

У третьому розділі опитування результати менші, а саме 77,3% користувачів вважають екран активних замовлень кур'єра зручним та зрозумілим одночасно, 13.6% вважають його тільки зручним, інші – тільки зрозумілим. На інші запитання в опитуванні користувачі давали схожі відповіді з відповідним середнім відсотком позитивних відповідей не менше 70%.

Рисунок 3.11 – Діаграма результатів третього пункту опитування.

Результати тестування зручності використання продемонстрували загалом позитивне сприйняття продукту користувачами. Більшість респондентів вважали основні екрани та функції зручними і зрозумілими для використання. Загалом, юзабіліті-тестування продемонструвало, що продукт має високий рівень зручності використання та зрозумілості для користувачів. Водночас, були

виявлені певні можливості для вдосконалення окремих компонентів та функцій, що може ще більше покращити загальний досвід користувачів у майбутньому.

Виконане тестування розробленого програмного продукту забезпечує задовільний рівень його якості. Було перевірено як функціональну, так і нефункціональну складові, проте з урахуванням обмеження на те, що розроблявся лише прототип окремого програмного модуля. Більш ретельна оцінка якості може задіювати модульне тестування, інші види інтеграційних тестів, зокрема, UI-тестування, тестування безпеки, продуктивності тощо.

ВИСНОВКИ

У процесі розробки модулю кур’єрської служби доставки було виконано ретельний аналіз ринку, що дало змогу визначити потреби та вимоги користувачів до подібних додатків. На основі отриманих даних був спроектований детальний прототип, який став відправною точкою для подальшої роботи. Наступним важливим кроком стало проєктування бази даних, яка була ретельно розроблена з урахуванням структури прототипу та необхідних функціональних можливостей застосунку. Це забезпечило ефективне зберігання та обробку даних, а також полегшило подальшу інтеграцію з іншими компонентами системи.

Продовженням проєктування стала архітектура системи, що базувалася на прототипі та враховувала вимоги до масштабованості, безпеки та продуктивності. Після завершення розробки на основі запропонованої архітектури було проведено ретельне тестування бекенду з використанням інструменту Postman. Ця фаза була критично важливою для виявлення та усунення будь-яких помилок або недоліків у роботі застосунку на ранній стадії, забезпечуючи високу якість та стабільність системи.

Для реалізації модуля кур’єрської доставки були обрані передові технології, такі як MongoDB – високопродуктивна та масштабована система керування базами даних, Node.js/Express.js – потужний стек для розробки серверної частини, React.js – популярна бібліотека для створення користувацьких інтерфейсів, та Axios – надійна бібліотека для здійснення HTTP-запитів.

У результаті ретельної роботи та застосування передових практик і технологій був створений якісний і функціональний модуль кур’єрської доставки, який задовольняє потреби користувачів та забезпечує зручний і ефективний спосіб доставки товарів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Яценко А. Як відкрити доставку їжі - все про організацію бізнесу служби доставки їжі. URL: https://lookfort.com/uk/yak-vidkryty-dostavku-yizhi/ (дата звернення: 24.12.2023).

2. Delivery Solution. URL: https://deliverysolution.site/ (дата звернення: 24.12.2023).

3. Courier Wizard LLC. Courier Wizard LLC. URL: https://www.courierwizardllc.com/ (дата звернення: 26.12.2023).

4. DeliveryTech. DeliveryTech. URL: https://www.deliverytech.net/ (дата звернення: 12.01.2024).

5. Проектування програмного забезпечення – Wezom. IT-компанія повного циклу розробки програмних продуктів WEZOM - Київ, Україна. URL: https://wezom.com.ua/ua/blog/proektirovanie-programmnogo-obespecheniya (дата звернення: 15.03.2024).

6. Why Software Design Is Important. IEEE Computer Society. URL: https://www.computer.org/resources/importance-of-software-design-is-important?Campaign_ID=263&gad_source=1&gclid=CjwKCAjwgdayBhBQEiwAXhMxtrNPt4p6IeCIsBCh4_CAiJqu9v0xn3bhkibFUVpNt82lnmc39tVmohoC644QAvD_BwE (дата звернення: 16.03.2024).

7. Обов’язкові функції онлайн-додатку для кур’єрської доставки 2023 року | Блог на Stfalcon. Custom Software Development Company | Stfalcon.com. URL: https://stfalcon.com/uk/blog/post/must-have-features-of-online-courier-delivery-app (дата звернення: 16.03.2024).

8. Автоматизація керування доставкою: кому необхідна та які завдання вирішує автоматизація доставки – Wezom. IT-компанія повного циклу розробки програмних продуктів WEZOM - Київ, Україна. URL: https://wezom.com.ua/ua/blog/avtomatizaciya-upravleniya-dostavkoj (дата звернення: 18.03.2024).

9. React. React. URL: https://react.dev/ (дата звернення: 20.03.2024).

10. ReactDOM – React. URL: https://legacy.reactjs.org/docs/react-dom.html (дата звернення: 20.03.2024).

11. Reactjs vs React Native - The all-in-one platform for effective SEO. URL: https://www.ranktracker.com/uk/blog/reactjs-vs-react-native-which-is-the-best-option-to-consider-for-mobile-apps/ (дата звернення: 20.03.2024).

12. Rud A. Що таке Node.js та для чого він потрібен? | Блог HyperHost.UA. URL: https://hyperhost.ua/info/uk/shho-take-nodejs-ta-dlya-cogo-vin-potriben?gad_source=1&gclid=CjwKCAjwgdayBhBQEiwAXhMxttFnRDwb1sDgrKVOEM8BQa6OHyjHRm4bmF7inzXv2kiCNEZDcnSieBoCn9QQAvD_BwE (дата звернення: 23.03.2024).

13. Express - це фреймворк для веб-застосунків, побудованих на Node.js. URL: https://expressjs.com/uk/ (дата звернення: 23.03.2024).

14. Основи MongoDB. DevZone. URL: https://devzone.org.ua/post/osnovy-mongodb (дата звернення: 24.03.2024).

15. Nodemon що це, як використовувати і які його переваги. FoxmindEd. URL: https://foxminded.ua/nodemon-shcho-tse/#:~:text=Сьогодні%20ми%20розберемося,%20nodemon%20що,його%20в%20разі%20зміни%20файлів (дата звернення: 24.03.2024).

16. User Friendly Interface ⊳ 17 способів зробити інтуїтивно зрозумілий інтерфейс | Блог HOSTiQ.ua. URL: https://hostiq.ua/blog/ukr/user-friendly-interface/ (дата звернення: 30.03.2024).

17. Омельчук Є. Що таке Figma: функції, інструменти та переваги - академія Wezom. Академия Wezom - Навчаємо IT технологіям з нуля. URL: https://wezom.academy/ua/chto-takoe-figma-funktsii-instrumenty-ipreimuschestva/ (дата звернення: 30.03.2024).

18. Demir D. What is Postman (A Tutorial for Beginners). Apidog Blog. URL: https://apidog.com/blog/what-is-postman/?utm_source=google_dsa&utm_medium=g&utm_campaign=20556541359&utm_content=154844519700&utm_term=&gad_source=1&gclid=CjwKCAjwx-CyBhAqEiwAeOcTdZi542x5wO3CbGbKEdx54dhGVOpCS3OpOImGRXz1fzms352pUf9VJRoCRG0QAvD_BwE (дата звернення: 25.05.2024).