Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка веб-застосунку для автоматизації логістичних процесів

Виконав: студент IV курсу, групи СН-42
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Ляшенко Г.О.
(підпис) (прізвище та ініціали)

Керівник _____ Небесний Р.М.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Козак Р.О.
(підпис) (прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«_27_» _січня_ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Ляшенку Гнату Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка веб-застосунку для автоматизації логістичних процесів

Керівник роботи Небесний Руслан Михайлович, доктор філософії (PhD), доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*07*» *травня* 2025 року № *4/7-444*

2. Термін подання студентом завершеної роботи 25 червня 2025р.

3. Вихідні дані до роботи Наукові та інформаційні інтернет джерела

4. Зміст роботи (перелік питань, які потрібно розробити)

Вступ. 1. Аналіз організації логістичних процесів на українському ринку та постановка завдання. 1.1 Характеристика ринку логістики. 1.2 Аналіз готових рішень для автоматизації логістичних процесів. 1.3 Актуальність та потреби в автоматизації логістичних процесів. 1.4 Постановка завдання та формулювання мети розробки веб-застосунку. 1.5 Висновок до першого розділу. 2. Планування, вибір архітектурного підходу та технологій для створення веб-застосунку для автоматизації логістичних процесів. 2.1 Необхідність планування перед початком активної фази розробки веб-застосунку для автоматизації логістичних процесів. 2.2 Вибір мови програмування. 2.3 Архітектурний підхід для взаємодії серверної та клієнтської частин веб-застосунку. 2.4 Технології для створення серверної частини веб-застосунку. 2.5 База даних та СКБД. 2.6 Архітектурний паттерн серверної частини. 2.7 Технології для створення клієнтської частини веб-застосунку. 2.8 Висновок до другого розділу. 3. Створення веб-застосунку для автоматизації логістичних процесів. 3.1 Підготовка середовища та технологій для розробки. 3.2 Створення серверної частини. 3.3 Створення клієнтської частини. 3.4 Зберігання коду та хостинг веб-застосунку. 3.5 Тестування веб-застосунку. 3.6 Висновок до третього розділу. 4. Безпека життєдіяльності, основи охорони праці. Висновки. Перелік джерел. Додатки.

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

1. Титульний слайд. 2. Актуальність теми. 3. Мета та завдання роботи. 4. Об'єкт і предмет дослідження. 5. Аналіз існуючих рішень. 6. Формулювання вимог до системи. 7. Архітектура системи. 8. Використані технології. 9. Use-case та ER-діаграми. 10. Реалізація серверної частини. 11. Реалізація клієнтської частини. 12. Тестування веб-застосунку. 13. Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Сенчишин В.С., к.т.н., доцент кафедри МТ	02.06.2025	05.06.2025

7. Дата видачі завдання 27 січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	27.01.2025	*Виконано*
2.	Підбір джерел про сферу логістики в Україні	08.05.2025-15.05.2025	*Виконано*
3.	Опрацювання джерел по темі кваліфікаційної роботи	16.05.2025-19.05.2025	*Виконано*
4.	Виконання дослідження щодо існуючих рішень по автоматизації логістичних процесів	20.05.2025-22.05.2025	*Виконано*
5.	Оформлення розділу «Аналіз організації логістичних процесів на українському ринку та постановка завдання»	23.05.2025-25.05.2025	*Виконано*
6.	Оформлення розділу «Планування, вибір архітектурного підходу та технологій для написання веб-застосунку для автоматизації логістичних процесів»	26.05.2025-29.05.2025	*Виконано*
7.	Оформлення розділу «Створення веб-застосунку для автоматизації логістичних процесів»	30.05.2025-01.06.2025	*Виконано*
8.	Виконання завдання до підрозділу «Безпека життєдіяльності»	02.06.2025-03.06.2025	*Виконано*
9.	Виконання завдання до підрозділу «Основи охорони праці»	04.06.2025-05.06.2025	*Виконано*
10.	Оформлення кваліфікаційної роботи	06.06.2025-10.06.2025	*Виконано*
11.	Нормоконтроль	11.06.2025-13.06.2025	*Виконано*
12.	Перевірка на плагіат	16.06.2025	*Виконано*
13.	Попередній захист кваліфікаційної роботи	19.06.2025	*Виконано*
14.	Захист кваліфікаційної роботи	26.06.2025	

Студент _____ Ляшенко Г.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Небесний Р.М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка веб-застосунку для автоматизації логістичних процесів // Кваліфікаційна робота освітнього рівня «Бакалавр» // Ляшенко Гнат Олегович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-42 // Тернопіль, 2025 // С. 61, рис. – 18, табл. – 1, джерел – 29, додат. – 2.

Ключові слова: логістика, веб-застосунок, база даних, автоматизація, вантажні перевезення, проста взаємодія, статус заявки.

Кваліфікаційна робота присвячена розробці веб-застосунку для автоматизації логістичних процесів в Україні. У першому розділі проаналізовано сучасний стан логістичного ринку, досліджено готові IT-рішення, обґрунтовано потребу в автоматизації та сформульовано завдання проєкту.

У другому розділі описано процес планування, обґрунтовано вибір мов програмування, архітектурного підходу та технологій для створення клієнтської й серверної частин.

У третьому розділі детально описано процес створення веб-застосунку, реалізацію бізнес-логіки, налаштування API, інтерфейсу користувача та тестування системи.

У четвертому розділі розглянуто питання безпеки життєдіяльності та охорони праці під час роботи з веб-технологіями.

Об'єкт дослідження: процеси логістики.

Предмет дослідження: засоби автоматизації логістичних процесів за допомогою веб-технологій.

ANNOTATION

Development of a Web Application for Automating Logistics Processes // Qualification work of the educational level "Bachelor" // Liashenko Hnat // Ternopil Ivan Pulyu National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, group SN-42 // Ternopil, 2025 // P. 61, fig. – 18, tabl. – 1, annexes. – 2, references – 29.

Keywords: logistics, web application, database, automation, cargo transportation, simple interaction, order status.

The qualification work is dedicated to the development of a web application for the automation of logistics processes in Ukraine. The first section analyzes the current state of the logistics market, examines ready-made IT solutions, justifies the need for automation, and formulates the project objectives.

The second section describes the planning process and justifies the choice of programming languages, architectural approach, and technologies for creating the client and server parts.

The third section describes in detail the process of creating a web application, implementing business logic, configuring the API, user interface, and testing the system.

The fourth section discusses the issues of life safety and occupational health when working with web technologies.

Object of research: logistics processes.

Subject of research: means of automating logistics processes using web technologies.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (англ. Application Programming Interface) – Інтерфейс прикладного програмування.

CLI (англ. Command-Line Interface) – Інтерфейс командного рядка.

DTO (англ. Data Transfer Object) – Об’єкт передавання даних.

ENV (англ. environment) – Змінна/файл середовища.

HTTP (англ. HyperText Transfer Protocol) – Протокол передавання гіпертексту.

ID (англ. identifier) – Ідентифікатор.

IP (англ. Internet Protocol) – Інтернет-протокол.

JSON (англ. JavaScript Object Notation) – Текстовий формат обміну даними JSON.

KYC (англ. Know Your Customer) – Процедура «знай свого клієнта».

MVP (англ. Minimum Viable Product) – Мінімально життєздатний продукт.

ORM (англ. Object-Relational Mapping) – Об’єктно-реляційне відображення.

REST (англ. Representational State Transfer) – Архітектурний стиль REST.

SQL (англ. Structured Query Language) – Мова структурованих запитів SQL.

SSH (англ. Secure Shell) – Безпечний протокол оболонки SSH.

UI (англ. User Interface) – Користувацький інтерфейс.

БД – База даних.

Бекенд (англ. Backend) – Серверна частина застосунку, яка відповідає за логіку, обробку даних і взаємодію з базою даних.

СКБД – Система керування базами даних.

Фронтенд (англ. Frontend) – Клієнтська частина застосунку, яка відповідає за відображення інтерфейсу та взаємодію з користувачем.

ЗМІСТ

ВСТУП

Актуальність теми. В умовах розвитку ринку логістики в Україні та зростання обсягів вантажоперевезень, важливим є покращення взаємодії між водіями, логістами та клієнтами. Сьогодні існує багато онлайн-платформ, які допомагають організовувати перевезення, проте вони мають певні недоліки, складні інтерфейси або недостатньо враховують потреби користувачів. Тому створення простого та зручного веб-застосунку, який об’єднає кращі бізнес-моделі конкурентів і усуне їхні недоліки, є актуальним напрямком для розвитку логістичних процесів в Україні.

Мета і задачі дослідження. Метою цієї кваліфікаційної роботи є розробка веб-застосунку, який покращить взаємодію між водіями, логістами та клієнтами, що створюють заявки на перевезення. Для досягнення цієї мети потрібно виконати такі завдання:

- проаналізувати існуючі рішення на ринку логістики;
- визначити основні труднощі в користуванні існуючих сервісів;
- впровадити основні бізнес-моделі конкурентів з урахуванням їхніх сильних сторін;
- реалізувати веб-застосунок, який буде простим і зручним для користувачів, з урахуванням поставлених вимог.

Практичне значення одержаних результатів. Результати цієї роботи можуть бути використані для покращення організації логістичних процесів, зменшення часу на пошук і оформлення перевезень, а також підвищення ефективності роботи водіїв і логістів. Створений веб-застосунок допоможе зробити ринок вантажоперевезень більш прозорим і зручним для всіх учасників.

РОЗДІЛ 1. АНАЛІЗ ОРГАНІЗАЦІЇ ЛОГІСТИЧНИХ ПРОЦЕСІВ НА УКРАЇНСЬКОМУ РИНКУ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Характеристика ринку логістики

Сектор вантажних перевезень надзвичайно важливий для України, особливо за нинішніх умов. Через війну доступ до морських портів та аеропортів став обмеженим або взагалі неможливим. Тому основний тягар доставки товарів усередині країни та через її кордони лягав на автомобільний транспорт. Таким чином, логістика та вантажні перевезення залишаються вирішальними для економіки, постачання гуманітарної допомоги, продуктів харчування, палива, будівельних матеріалів та інших важливих ресурсів.

Логістика відіграє в цьому ключову роль. Це допомагає організувати перевезення та з'єднує клієнтів – тих, кому потрібно перевезти вантаж – з перевізниками – водіями та логістичними компаніями, які надають ці послуги. Без якісного управління логістикою такі процеси були б хаотичними, а час і ресурси витрачалися б даремно.

Сьогодні ринок логістичних послуг в Україні перенасичений. Це означає, що водіїв та компаній, які надають транспортні послуги, більше, ніж клієнтів. Через це можна спостерігати зниження цін на транспортні послуги, оскільки конкуренція зростає, і експедитори змушені знижувати вартість своїх послуг, щоб залишатися конкурентоспроможними. Це може призвести до того, що багато водіїв залишаться без замовлень або працюватимуть собі в збиток. При цьому враховувались окремі умови часової невизначеності, оскільки переважна більшість факторів за природою мають довгостроковий вплив [1].

Щоб частково вирішити ці проблеми, існують різні готові рішення, які дозволяють експедиторам та клієнтам знаходити один одного. Деякі сервіси пропонують базу даних доступних замовлень, тоді як інші мають велику базу даних перевізників з обширною інформацією про них. Однак більшість цих

рішень мають суттєві недоліки. Це знижує ефективність використання таких послуг та не задовольняє потреби всіх учасників ринку.

Цифрова трансформація зачіпає всі сектори суспільства, зокрема економіку. Тепер компанії отримують можливість радикально змінити свої бізнес-моделі за допомогою нових цифрових технологій, таких як соціальні мережі, мобільні пристрої, великі дані, Інтернет речей, інші інновації [2].

1.2 Аналіз готових рішень для автоматизації логістичних процесів

На українському ринку є кілька популярних онлайн-сервісів, які допомагають автоматизувати логістичні процеси та організувати перевезення вантажів. У них спільна мета – з’єднати замовників перевезень з виконавцями (водіями або компаніями), але кожен з них реалізує це по-своєму, має свої особливості, переваги та недоліки.

Одним із таких сервісів є della.com.ua [3]. За допомогою цієї платформи клієнти можуть створювати заявки на перевезення, які можуть переглядати водії. Заявки доступні для перегляду навіть неавторизованим користувачам, проте для отримання детальної інформації, такої як контакти або номер телефону, ви повинні зареєструватися та придбати платну підписку. Інтерфейс сайту виглядає застарілим, але досить простим у використанні. Він не підходить для мобільних пристроїв, текст та елементи веб-сайту стають нечитабельні що можна побачити на рисунку 1.1.

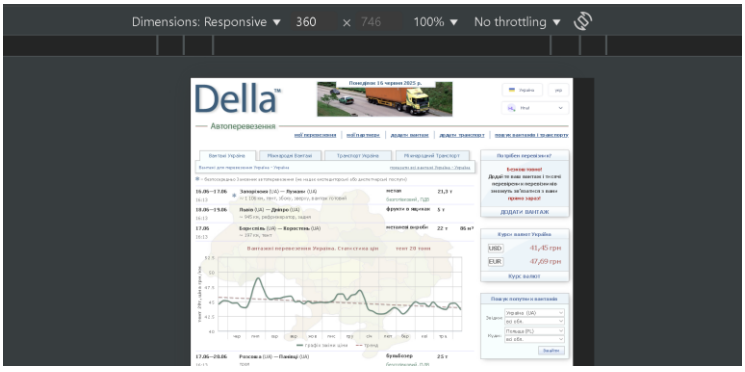

Рисунок 1.1 – Нечитабельна сторінка della.com.ua для мобільних пристроїв

Варто також зазначити, що сервіс підтримує кілька мов і дозволяє працювати не лише на українському ринку, а й з міжнародними перевезеннями.

Ще один відомий сервіс – lardi-trans.ua [4]. Основна увага цієї платформи також приділяється створенню заявок, які з'єднують перевізників з клієнтами. Порівняно з попереднім прикладом, веб-застосунок має сучасніший вигляд та ширший набір функцій. Є мобільна адаптація, яка дозволяє зручно користуватися платформою з телефону. Однак, доступ до додатків можливий лише після реєстрації та проходження процесу KYC (перевірка особи) [5], що може ускладнити перші кроки використання (див. рис. 1.2).

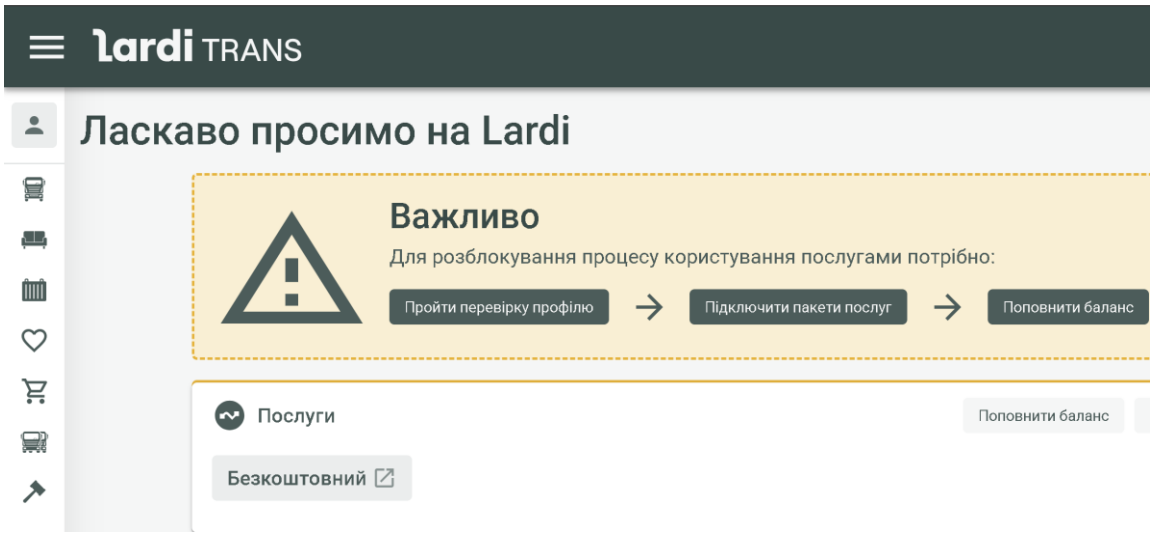

Рисунок 1.2 – Обмеження доступу до веб-сайту lardi-trans.ua

Крім того, сервіс пропонує платну підписку, яка розблоковує додаткові функції. Хоча інтерфейс користувача сучасний, він може бути складним для новачків – на перший погляд важко розібратися, де і як знайти потрібну інформацію.

Іншим прикладом є сервіс kabanchik.ua [6], який має окрему категорію "Вантажні перевезення". Цей ресурс більше схожий на базу даних перевізників, де кожен водій може створити свій власний профіль, додати інформацію про себе, транспортний засіб, досвід та види послуг, які пропонує (див. рис. 1.3).

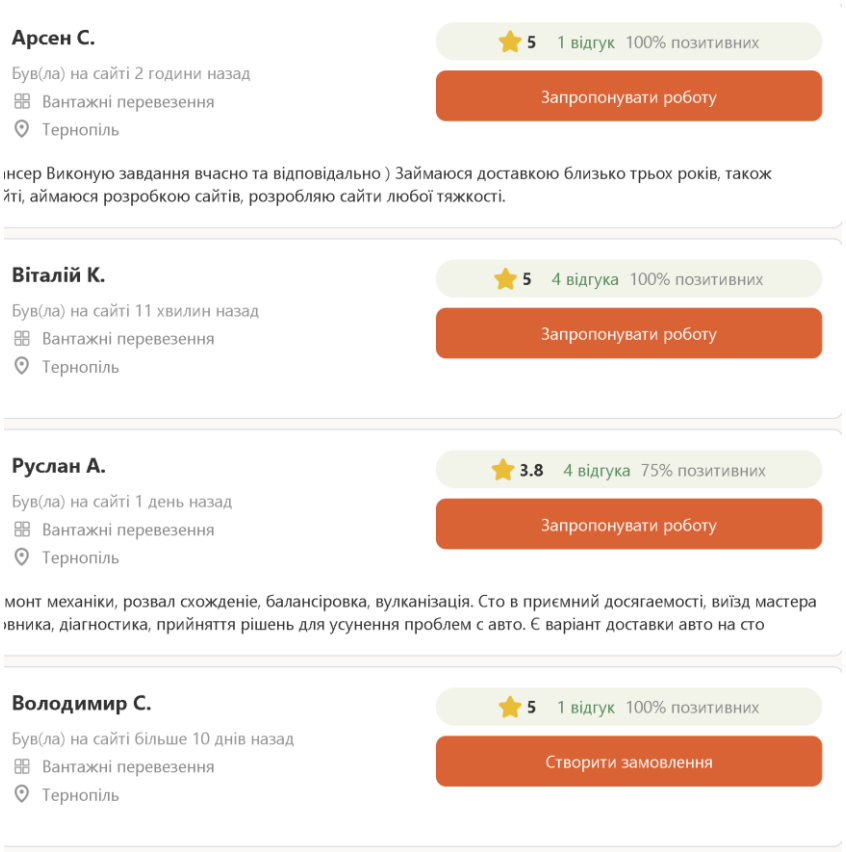

Рисунок 1.3 – База даних перевізників на веб-сайті kabanchik.ua

Клієнти, у свою чергу, повинні самостійно шукати серед усіх профілів потрібного підрядника. Такий підхід менш активний – водії не конкурують за роботу, а просто чекають, поки клієнт їх знайде. У сучасних умовах, коли на ринку більше перевізників, ніж клієнтів, цей метод взаємодії може бути не дуже ефективним.

1.3 Актуальність та потреби в автоматизації логістичних процесів

Ринок логістичних послуг в Україні наразі використовує модель, за якою клієнти створюють заявки на перевезення, а водії (або логісти) самі обирають для них найбільш підходящі. Такий підхід є ефективним, оскільки дозволяє обом сторонам гнучко адаптуватися до власних потреб – водії шукають вигідні маршрути, а клієнти можуть швидше знайти підрядника. Однак, для того, щоб такий процес проходив безперебійно, важливо, щоб платформи, що надають ці

послуги, були зрозумілими, швидкими, доступними та зручними для користувачів.

Одна з головних проблем існуючих сервісів – це незручний або перевантажений інтерфейс користувача. Наприклад, деякі платформи мають застарілий зовнішній вигляд, який не адаптований для мобільних пристроїв. У теперішній час, коли більшість користувачів заходять на веб-сайт через свій телефон, це спричиняє труднощі – кнопки можуть бути дрібними, деяка інформація може не відображатися, а навігація стає складною. З іншого боку, навіть сучасні платформи з адаптивним дизайном часто перевантажені функціями, які відлякують користувача, а не допомагають йому. Інтерфейс користувача може бути заплутаним, і людині без досвіду може бути важко зрозуміти, як знайти або опублікувати вакансію.

Ще однією важливою проблемою є обмежений доступ до інформації. Наприклад, на деяких платформах запити на перевезення неможливо переглянути без реєстрації, а повні контактні дані видно лише після оплати підписки. Це створює перешкоду для нових користувачів, які хочуть ознайомитися з сервісом, перш ніж приймати рішення про співпрацю.

Також варто відзначити відсутність профілів перевізників, або обмежений функціонал таких профілів. Деякі сервіси пропонують можливість створити анкету, в якій ви описуєте свій досвід та транспорт. Однак, це скоріше виняток, ніж правило. У наш час для клієнта дуже важливо бачити, з ким він має справу – який досвід має водій, який вид транспорту він пропонує та чи є відгуки від інших клієнтів. Існування таких профілів може значно підвищити рівень довіри між сторонами.

Ще одна проблема – складна реєстрація. Деякі сервіси вимагають повної верифікації за допомогою паспортних даних, перш ніж користувач зможе отримати доступ до функціональності. Це може відлякати потенційного клієнта, який просто хоче спочатку протестувати сервіс без зайвих кроків.

Крім того, на ринку є сервіси, які більше схожі на просту базу даних профілів, де не відбувається активного пошуку роботи чи подання заявки. У

таких випадках пошук підрядника повністю перекладається на замовника. Наразі між перевізниками існує сильна конкуренція, тому важливо, щоб саме водії активно відгукувались на заявки, а не навпаки.

Також можна назвати деякі інші поширені недоліки, що зустрічаються на подібних платформах: повільна робота сайту, відсутність підтримки, технічні помилки або недостатній захист персональних даних. Усі ці фактори підривають довіру користувачів до таких послуг та ускладнюють взаємодію між логістами, перевізниками та замовниками.

1.4 Постановка завдання та формулювання мети розробки веб-застосунку

Проаналізувавши існуючі сервіси для автоматизації логістичних процесів, можна зробити висновок, що кожен з них має як сильні, так і слабкі сторони. Більшість платформ пропонують ефективну модель комунікації між клієнтом і перевізником, публікуючи заявки з усією інформацією. Це зручно, оскільки водії та логісти можуть вибрати саме те, що їм найкраще підходить. Водночас, деякі сервіси дозволяють створювати профілі з інформацією про себе, що підвищує довіру потенційних клієнтів.

Однак, усі розглянуті готові рішення мають свої проблеми. Застарілий або надмірно складний інтерфейс користувача, відсутність адаптації для мобільних пристроїв, обмежений доступ без підписки, складна система реєстрації. Все це ускладнює роботу з сервісами, особливо для нових користувачів.

З огляду на це, постає завдання у створенні сучасного веб-застосунку, який поєднує найкращі ідеї конкурентів, одночасно усуваючи їхні недоліки. Основна увага цього веб-застосунку приділяється простій моделі взаємодії через список заявок, де перевізники та замовники можуть швидко знаходити один одного. Додатково буде можливість створювати та заповнювати власні профілі, щоб краще презентувати себе, свій транспортний засіб та свій досвід.

У майбутньому застосунок також може бути розширений, включивши функцію чату для полегшення спілкування.

Особлива увага буде приділена практичному, зрозумілому та адаптивному інтерфейсу, який працює на різних пристроях – як комп’ютерах, так і телефонах.

1.5 Висновок до першого розділу

У першому розділі наведено детальний аналіз сучасного стану логістичних процесів в Україні. Він показує, що ринок вантажних перевезень надзвичайно важливий для економіки, особливо у воєнний час, коли традиційні транспортні шляхи обмежені. Автомобільний транспорт несе основний тягар, що підвищує потребу в ефективній організації логістики.

Було розглянуто існуючі сервіси автоматизації логістичних процесів, що вже працюють на українському ринку. Усі вони мають спільну мету – зв’язок клієнтів із перевізниками, але реалізують її по-різному. Аналіз показав, що хоча ці сервіси виконують свої основні функції, більшість із них страждають від суттєвих недоліків. Незручні або застарілі інтерфейси користувача, обмежений доступ до функцій без підписки, складна реєстрація та відсутність повних профілів перевізників. Це створює бар’єри для користувачів та знижує ефективність взаємодії.

На основі цього аналізу було визначено головну мету – створити зручний, простий та адаптивний веб-застосунок, який враховує проблеми конкурентів та пропонує покращене рішення. Запропонована модель взаємодії дозволяє клієнтам, логістам та водіям швидко знаходити одне одного через систему запитів, доповнену персональними профілями. Такий підхід робить логістичні послуги доступнішими, зрозумілішими та ефективнішими для всіх учасників ринку.

РОЗДІЛ 2. ПЛАНУВАННЯ, ВИБІР АРХІТЕКТУРНОГО ПІДХОДУ ТА ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ АВТОМАТИЗАЦІЇ ЛОГІСТИЧНИХ ПРОЦЕСІВ

2.1 Необхідність планування перед початком активної фази розробки веб-застосунку для автоматизації логістичних процесів

Важливим кроком у розробці веб-застосунків є планування. На цьому етапі потрібно визначити основні завдання, продумати порядок їх виконання, а також врахувати можливі труднощі, які можуть виникнути в процесі розробки. Планування допомагає уникнути хаотичної роботи, зменшити ризики, розподілити обов’язки та чітко визначити кінцеву мету. Після успішного розгляду та реалізації ідеї важливо чітко визначити подальший розвиток проєкту.

Без плану легко втратити фокус і витратити більше часу та ресурсів, ніж потрібно. Планування дозволяє ефективно використовувати наявні ресурси, встановлювати терміни для кожного етапу та отримувати уявлення про загальну картину розвитку. Це особливо важливо для складних проєктів, де є багато функцій та взаємодій між різними частинами системи.

Визначається важливість та пріоритетність кожного завдання. Розглядається терміни виконання, залежності між завданнями та ризики. Встановлюються пріоритети, щоб забезпечити вчасне та успішне виконання критичних завдань [1].

Крім того, перш ніж почати писати код, слід подумати про те, які технології використовувати, як виглядатиме архітектура застосунку та які інструменти зроблять процес розробки зручнішим. Такий підхід допоможе уникнути багатьох помилок у майбутньому та досягти більш якісного результату.

Таким чином, ретельне планування є основою успішної реалізації веб-застосунку і ним не слід нехтувати. Це створює чітке бачення проєкту, допомагає краще організувати роботу та наблизитися до мети.

2.1.1 Діаграма варіантів використання

Діаграма варіантів використання (англ. Use-case diagram) – це спосіб представлення того, як користувачі взаємодіють із системою. Він показує, які дії користувач може виконувати у веб-застосунку та як система реагує на ці дії [7]. Використовуючи таку діаграму, можна зрозуміти основні особливості програми ще до початку розробки.

Головна мета такої діаграми – показати, хто саме використовуватиме систему (наприклад, водій, логіст або замовник) і що кожен з них може робити: створювати заявки, переглядати маршрути, спілкуватися в чаті тощо. Це допомагає не лише розробникам, а й цілій команді краще уявити, як система працюватиме в реальності.

На рисунку 2.1 наведено діаграму, що показує основні варіанти використання мого веб-застосунку та те, як користувачі взаємодіють із системою.

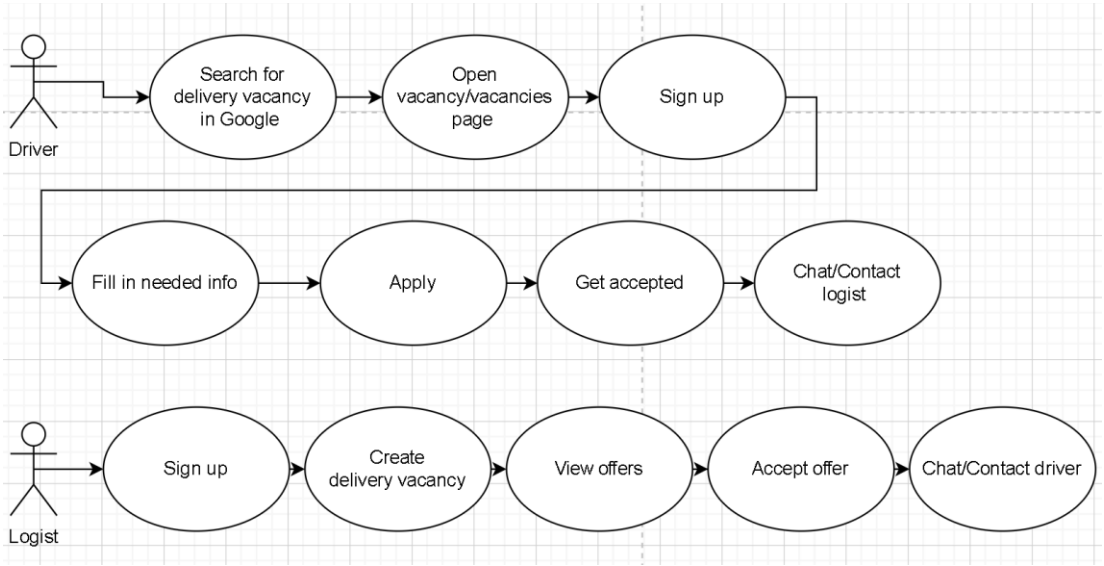

Рисунок 2.1 – Діаграма варіантів використання

Створення такої діаграми є важливим кроком на етапі планування, оскільки вона забезпечує чітке розуміння системних вимог і допомагає уникнути плутанини в майбутньому. Це також служить основою для розробки архітектури та вибору технологій.

2.1.2 Діаграма відношень сутностей

Під час розробки веб-застосунку важливо точно розуміти, які дані використовуються в системі та як ці дані пов’язані між собою. Для цього створюється діаграма прецедентів (англ. Entity-Relationship Diagram) [8].

Сутності – це основні об’єкти, з якими працює система. У нашому випадку це користувачі, профілі, заявки на перевезення, транспортні засоби та інші. Кожна сутність зазвичай реалізується як окрема таблиця в БД. Діаграму можна використовувати для візуального представлення того, які сутності знаходяться в системі, які атрибути вони мають та як вони пов’язані одна з одною.

Зв’язки між сутностями можуть бути різних типів: один до одного, один до багатьох або багато до багатьох (див. рис. 2.2).

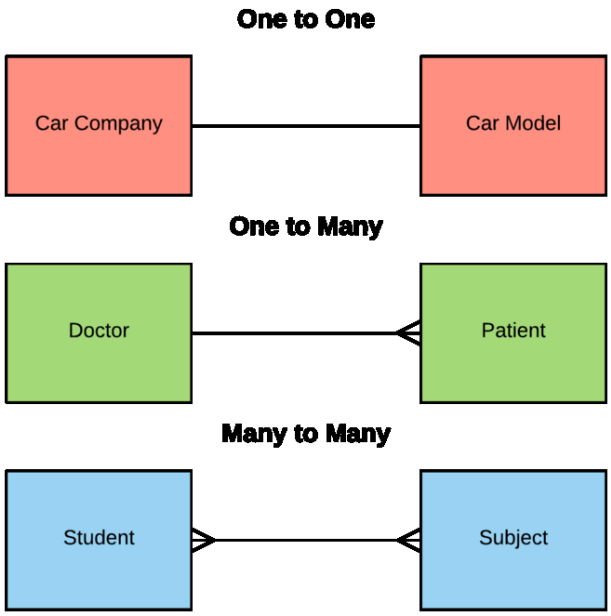

Рисунок 2.2 – Типи зв’язків між сутностями [9]

Наприклад, клієнт може мати багато запитів, але кожен запит належить лише одному клієнту – це зв’язок один до багатьох.

На рисунку 2.3 наведено діаграму, яка показує основні сутності веб-застосунку та зв’язки між ними.

Рисунок 2.3 – Діаграма відношень сутностей

Така діаграма дуже корисна для розробників, оскільки дозволяє побачити загальну структуру бази даних та взаємодію між сутностями ще до початку впровадження, уникнути помилок та краще організувати зберігання та взаємодію з даними у проєкті.

2.2 Вибір мови програмування

Одним з найважливіших рішень під час розробки веб-застосунку є вибір мови програмування. Від цього залежить легкість розробки, швидкість системи, масштабованість та його підтримка в майбутньому.

У наш час існує багато мов програмування, які використовуються для створення веб-застосунків. Найпопулярнішими серед них є JavaScript, Python, PHP, Java та C#. Кожна з них має свої переваги та недоліки, і вибір залежить від конкретних завдань та їх складності.

JavaScript є однією з найбільш широко використовуваних мов для веб-розробки, особливо для створення користувацьких інтерфейсів. Його можна використовувати як на стороні клієнта, так і на стороні сервера (наприклад, за допомогою Node.js) [10]. Перевага полягає у великій спільноті, багатьох готових рішеннях та швидкості розробки. Однак, іноді код JavaScript може бути складнішим для підтримки через динамічну типізацію.

Python добре підходить для серверної частини застосунків. Його синтаксис простий та зрозумілий. Існує багато корисних фреймворків, таких як Django або Flask [11]. Python ідеально підходить для проєктів, де важливі швидка розробка та читабельність коду. Однак, за великих навантажень, він може бути менш ефективним, ніж інші мови програмування.

PHP використовується у веб-розробці вже багато років, особливо для створення динамічних веб-сайтів. Він легко інтегрується з базами даних, а також добре підходить для роботи з контентом [12]. Хоча новіші версії мови стали набагато кращими, деякі розробники все ще вважають PHP застарілим або менш зручним.

Java та C# частіше використовуються у великих корпоративних додатках [13]. Вони надійні та добре підходять для проєктів з високими вимогами до безпеки та стабільності [14]. Однак для малих або середніх проєктів ці мови можуть бути дещо "важкими" та вимагати більше часу на налаштування та розгортання.

Вибираючи мову програмування, враховувались такі фактори, як швидкість розробки, наявність документації, активність спільноти та особистий досвід. Це дозволило обрати найкращий варіант для реалізації веб-застосунку.

2.2.1 Мова програмування JavaScript

JavaScript виконується безпосередньо у веб-браузері, тобто не потрібно додатково встановлювати жодні програми для того, щоб код працював. Усі сучасні браузери (Google Chrome, Mozilla Firefox, Safari, Edge та інші) підтримують цю мову "з коробки", що робить її дуже доступною для використання.

Важливою особливістю JavaScript є велика кількість готових інструментів бібліотек та фреймворків. Найпопулярніші з них React, Vue.js та Angular. Вони допомагають значно спростити розробку складних інтерфейсів, економлять час та дозволяють створювати зручні, швидкі й сучасні веб-додатки. Наприклад, React активно використовується такими компаніями як Facebook та Instagram.

Ще однією сильною стороною JavaScript є його універсальність. Окрім розробки інтерфейсів (фронтенду), JavaScript можна використовувати і для створення серверної частини (бекенду) з допомогою середовища Node.js. Це означає, що вся веб-розробка може вестися лише з використанням однієї мови, що зручно і зменшує поріг входу для новачків.

JavaScript має й активну спільноту. Щодня з'являються нові статті, відеоуроки, плагіни, приклади коду. Також регулярно випускаються оновлення мови, які додають нові можливості та покращують її стабільність і безпеку. Це означає, що JavaScript постійно розвивається і йде в ногу з вимогами часу.

Проте, незважаючи на велику кількість переваг, JavaScript має й деякі недоліки. Основна проблема – це динамічна типізація. Іншими словами, під час написання коду не потрібно вказувати, якого типу дані використовуються (наприклад, число чи текст). Це може зекономити час на початковому етапі, але водночас збільшує ризик появи помилок, які стають помітними лише під час

виконання програми. Особливо це може ускладнити роботу над великими проєктами, де багато взаємозв'язаних компонентів.

Ще один виклик – структура коду. Якщо не дотримуватись чітких правил і стандартів під час написання JavaScript-коду, проєкт може стати складним у підтримці. Саме тому досвідчені розробники часто використовують додаткові інструменти, такі як TypeScript, який додає статичну типізацію та допомагає уникнути частини помилок ще до запуску коду.

2.2.2 Мова програмування TypeScript

TypeScript – це мова програмування, яка була створена як розширення для JavaScript. Основна її особливість полягає в тому, що вона додає статичну типізацію. Це означає, що під час написання коду розробник може вказати, який саме тип даних (наприклад, число, текст або список) має бути у змінної, функції чи параметра [15]. Завдяки цьому помилки можна виявити ще до запуску програми, тобто ще на етапі розробки.

Такий підхід особливо корисний у великих проєктах, де працює багато розробників і код має складну структуру. Типізація дозволяє краще контролювати, як саме використовуються змінні, функції та об’єкти, що значно зменшує ризик логічних помилок та непередбачуваної поведінки програми.

TypeScript був створений компанією Microsoft і швидко здобув популярність серед професійних розробників. Він повністю сумісний із JavaScript, тобто будь-який існуючий код можна поступово адаптувати під TypeScript. Це дуже зручно для тих команд, які хочуть поступово перейти на новий стандарт без необхідності повністю переписувати старі частини проєкту.

Одна з великих переваг TypeScript це підтримка з боку сучасних середовищ розробки (IDE). Наприклад, у Visual Studio Code TypeScript забезпечує автоматичне доповнення коду, підказки, перевірку типів у реальному часі та навіть рефакторинг – автоматичну зміну структури коду без

його поломки. Завдяки цьому процес розробки стає набагато комфортнішим і швидшим.

Ще один важливий плюс це зрозумілість коду. Коли в коді чітко вказано, який тип має кожна змінна, іншим розробникам легше зрозуміти, як працює той чи інший блок. Це дуже допомагає, коли над проєктом працює команда або коли потрібно повертатися до старого коду після тривалого часу.

2.2.3 Порівняльна характеристика JavaScript і TypeScript

JavaScript та TypeScript дуже схожі, але мають фундаментальні відмінності. JavaScript – це мова без суворих правил типізації, а TypeScript – розширена версія з чіткою структурою та перевіркою помилок. У таблиці 2.1 наведено порівняння цих мов програмування за їхніми основними критеріями.

Таблиця 2.1 – Порівняння критеріїв JavaScript і TypeScript

Критерій	**JavaScript**	**TypeScript**
Типізація	Динамічна	Статична
Перевірка помилок	Під час виконання	Під час написання коду
Складність вивчення	Простий для початківців	Складніший, потребує більше часу
Підказки в редакторі	Менше підказок	Більше підказок і автозаповнення
Підтримка великих проєктів	Складніше без додаткових інструментів	Краще підходить для великих команд і проєктів
Необхідність компіляції	Не потребує	Потребує компіляції у JavaScript

Після ознайомлення з обома мовами, було прийнято рішення використовувати TypeScript для розробки веб-застосунку. Основна причина – зручність роботи над проєктом такої складності. Статична типізація дозволяє

швидше знаходити помилки, має кращу структуру коду, а також спрощує подальше обслуговування програми.

2.3 Архітектурний підхід для взаємодії серверної та клієнтської частин веб-застосунку

Під час розробки веб-застосунку обрано архітектуру, яка включає створення два окремі проєкти – клієнтську частину (фронтенд) та серверну частину (бекенд).

Серверна частина відповідає за основну бізнес-логіку, обробку запитів, а також зберігання та отримання даних з бази даних. Саме сервер обробляє всі внутрішні процеси, перевіряє точність даних і вирішує, що і як має працювати. Для розробки сервера я використовую сучасну мову програмування з підтримкою API для легкої передачі даних між клієнтом і сервером.

Клієнтська сторона – це те, що користувач бачить у браузері. Він відображає інформацію, надіслану сервером, і дозволяє користувачеві взаємодіяти із системою: надсилати форми, переглядати списки, писати повідомлення тощо.

Для реалізації цієї ідеї використано клієнт-серверну архітектуру (див. рис. 2.4).

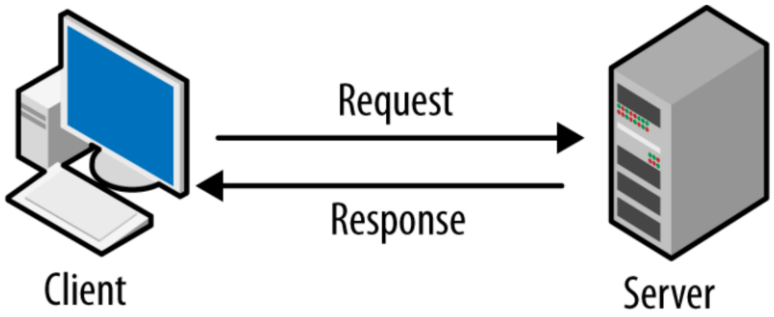

Рисунок 2.4 – Клієнт-серверна архітектура [16]

Суть цієї архітектури полягає в чіткому розподілі обов’язків: клієнт відповідає за інтерфейс, а сервер – за логіку та зберігання даних. Клієнт

надсилає запити на сервер (наприклад, "показати всі активні заявки"), а сервер повертає заздалегідь опрацьовану відповідь, яку клієнт потім відображає у зручному для користувача вигляді [16].

Переваги такого підходу полягають у тому, що можна легко оновлювати клієнт або сервер окремо, не змінюючи всю систему. Наприклад, можна покращити дизайн інтерфейсу, не втручаючись у логіку сервера. Такий підхід також дозволяє масштабувати систему. У майбутньому можна додати мобільний додаток або інші сервіси, що підключаються до того ж сервера.

Такий підхід є сучасним стандартом у розробці веб-застосунків та пропонує зручність як у розробці, так і в подальшій підтримці проєкту.

2.4 Технології для створення серверної частини веб-застосунку

Для розробки серверної частини веб-застосунку використано Node.js разом з фреймфорком NestJS. Цей вибір не був випадковим – опираючись на наявний досвід роботи з цими технологіями, можна зробити висновок, що вони добре підходять для створення сучасних, зручних для користувача та масштабованих веб-сервісів.

Node.js – це середовище, яке дозволяє запускати JavaScript не лише в браузері, але й на сервері. Це зручно, оскільки можна використовувати одну й ту саму мову програмування як для клієнта, так і для сервера. Крім того, Node.js швидкий, має багато попередньо створених бібліотек та велику спільноту [17]. Це дозволяє легко знаходити приклади, поради та готові рішення, коли виникають труднощі.

NestJS – це фреймворк, який працює на Node.js і дозволяє створювати логіку на стороні сервера чітким та структурованим чином [18]. Це схоже на фреймворки, що використовуються в інших мовах програмування, таких як Java або .NET. Це робить код зрозумілішим, легшим для читання та простішим у підтримці. NestJS також добре працює з TypeScript, що робить розробку зручнішою, надаючи підказки безпосередньо в редакторі коду.

Ці технології популярні та активно використовуються в багатьох компаніях та проєктах. Вони постійно оновлюються, мають велику базу знань та інструменти, тому я впевнений, що з їхньою допомогою зможу створити надійний та практичний сервер для мого веб-застосунку.

2.5 База даних та СКБД

У процесі розробки та проектування технічних і програмних систем такими критеріями виступають показники якості СКБД, зокрема, функціональність експлуатація, зручність у використанні та ін. [19].

Для зберігання даних у веб-застосунку використовується база даних PostgreSQL у поєднанні з pgAdmin – зручним інструментом для управління базою. Такий вибір забезпечує надійність, зручність у роботі та хорошу масштабованість системи.

PostgreSQL – це одна з найпотужніших та найпопулярніших систем керування реляційними базами даних з відкритим вихідним кодом. Вона активно використовується як у невеликих, так і у великих комерційних проєктах. Основна причина її популярності – висока стабільність, підтримка складних запитів, гнучкість та точне дотримання стандартів SQL [20].

Однією з ключових переваг PostgreSQL є підтримка зв’язків між таблицями – наприклад, один до одного, один до багатьох або багато до багатьох. Це дозволяє ефективно організувати структуру даних і забезпечити їхню логічну узгодженість. Крім того, система має вбудовані механізми перевірки цілісності даних, що допомагає уникати помилок при збереженні або оновленні інформації.

Ще одна важлива перевага – швидкість роботи. PostgreSQL добре оптимізований і здатен швидко обробляти великі обсяги даних, що особливо важливо для веб-застосунків, які працюють із тисячами або мільйонами записів. Система підтримує індекси, транзакції, тригери, розширення та багато інших функцій, які допомагають будувати надійні, продуктивні рішення.

Для зручності роботи з PostgreSQL використовується pgAdmin – графічний інтерфейс, який значно полегшує взаємодію з базою даних. За його допомогою можна: переглядати й редагувати таблиці, додавати або видаляти записи, створювати зв’язки між таблицями, писати SQL-запити та бачити результати, швидко перевіряти структуру бази [21].

Вибір PostgreSQL і pgAdmin є вдалим рішенням для даного веб-застосунку, оскільки ці інструменти добре відомі у спільноті, активно використовуються в реальних проєктах, мають хорошу документацію та численні приклади. Вони дозволяють побудувати стабільну, зручну й масштабовану систему зберігання даних, що легко підтримується та розширюється у майбутньому.

2.6 Архітектурний паттерн серверної частини

У бекенді веб-застосунку використано модульну архітектуру, яка добре підходить для масштабованих проєктів. Основна ідея цього підходу полягає в розділенні логіки програми на окремі, незалежні модулі. Кожен модуль відповідає за певну частину функціональності та містить усі необхідні компоненти для повноцінної роботи.

У цьому випадку, прикладом таких модулів є user, vehicle, driver-details та інші. Кожна з цих папок-модулів містить такі елементи:

- контроллер – відповідає за прийом запитів з клієнта та передачу їх до сервісу;
- сервіс – містить бізнес-логіку;
- репозиторій – працює з базою даних;
- DTO – визначає структуру даних, які приймаються або повертаються [22];
- сутність – описує таблицю в базі даних;
- інтерфейси – задають структури для типів даних.

Ось так виглядає файлова структура одного модуля user (див. рис. 2.5).

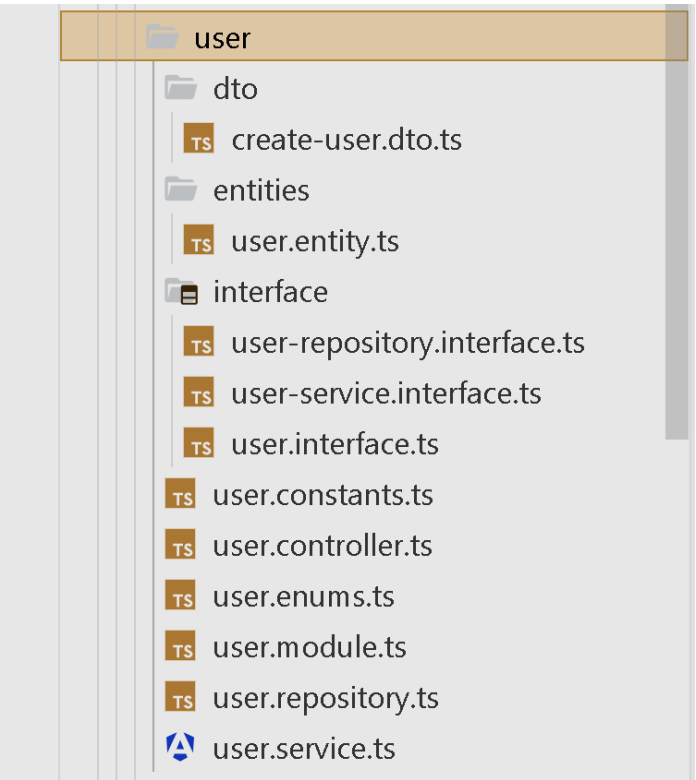

Рисунок 2.5 – Файлова структура модуля user

Такий підхід дозволяє зручно впорядкувати код, спростити навігацію по проєкту та тестування, а також полегшити додавання нових функцій у майбутньому. Розробники можуть працювати над різними модулями паралельно, не заважаючи один одному.

2.7 Технологій для створення клієнтської частини веб-застосунку

Для створення клієнтської частини даного веб-застосунку обрано React, Next.js та Material UI. Ці технології добре підходять для сучасної розробки, а також значно спрощують та прискорюють процес створення зручного інтерфейсу.

React – це бібліотека JavaScript, яка дозволяє створювати інтерфейси з окремих компонентів [23]. Це означає, що кожну частину сторінки (наприклад, кнопку, форму чи таблицю) можна написати окремо, а потім легко поєднати з

іншими. Такий підхід зручний і дозволяє використовувати один і той самий код кілька разів.

Next.js – це фреймворк на основі React. Він додає багато корисних речей: спрощену навігацію між сторінками, швидке завантаження сайту, Підтримка серверного рендерингу (тобто сторінка вже може бути завантажена з необхідними даними) [24]. Все це робить сайт зручнішим та швидшим для користувачів. Оптимізація продуктивності та SEO є надзвичайно важливими аспектами сучасної розробки веб-додатків, особливо з огляду на зростаючу конкуренцію в онлайн-просторі та зростання вимог користувачів до якості веб-ресурсів. [25]

Material UI – це набір готових компонентів (кнопок, форм, карток тощо), які виглядають стильно та сучасно [26]. Це означає, що не доведеться витрачати багато часу на дизайн, адже можна просто використовувати готові елементи, які вже добре виглядають і добре працюють на різних пристроях.

Разом ці технології дозволяють швидше писати код, легше його підтримувати та створювати інтерфейс, простий у використанні.

2.8 Висновок до другого розділу

Правильне планування та вибір архітектурного підходу і технологій перед початком активної розробки веб-застосунку є дуже важливими етапами, що суттєво покращують ефективність подальшої роботи. Якщо всі нюанси взаємодії між компонентами, а також вибір необхідних технологій, продумані заздалегідь, процес розробки значно прискорюється.

Якщо архітектура та технології обрані правильно, можна уникнути багатьох помилок, які можуть виникнути під час розробки, та зберегти час для створення самого застосунку. Вибір таких технологій, як Node.js, React, PostgreSQL та інші, дозволяє ефективно працювати над різними частинами проєкту, адже кожна з них добре інтегрується з іншими. Це робить код більш зрозумілим, а процес розробки – швидким і без зайвих ускладнень.

Вибравши правильну архітектуру та технології, можна уникнути багатьох помилок, які можуть виникнути під час розробки, та заощадити час під час створення самого застосунку. Обираючи такі технології, як Node.js, React, PostgreSQL та інші, можна ефективно працювати над різними частинами проєкту, оскільки кожна з них добре інтегрується з іншими. Це робить код більш зрозумілим, а процес розробки швидшим та без зайвих ускладнень.

Загалом, ретельне планування з самого початку дозволить створити надійний та зручний веб-застосунок, який функціонуватиме стабільно та безперебійно.

РОЗДІЛ 3. СТВОРЕННЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ АВТОМАТИЗАЦІЇ ЛОГІСТИЧНИХ ПРОЦЕСІВ

3.1 Підготовка середовища та технологій для розробки

Важливо почати з підготовки комфортного середовища для розробки. Середовище розробки – це все, що допомагає програмісту писати, тестувати та перевіряти код. Якщо все налаштувати правильно, робота стане набагато легшою та швидшою. У приємній обстановці можна уникнути відволікання на технічні деталі та зосередитися на самій програмі.

Використано редактор Visual Studio Code для написання коду. Це практична та легка програма з багатьма корисними розширеннями, підказками щодо написання коду та підтримкою багатьох мов програмування. Він простий у використанні та добре підходить як для розробки на стороні клієнта, так і на стороні сервера.

Ще одним важливим інструментом є система контролю версій Git. Необхідно вести історію змін коду. Це дозволяє точно бачити, що змінилося, повертатися до попередніх версій, а також легко співпрацювати в команді, коли над проєктом працює кілька людей.

Таке розділення та належна підготовка середовища допоможе уникнути хаосу в проєкті та дозволить зосередитися на найважливішому – створенні якісного веб-застосунку.

3.2 Створення серверної частини

Для створення бекенду веб-застосунку використано технологію Node.js у поєднанні з фреймворком NestJS.

Перш ніж розпочати розробку, потрібно встановити базу даних PostgreSQL, де буде зберігатися вся інформація.

Після встановлення всіх необхідних залежностей можна перейти до створення основних модулів, налаштування бази даних та реалізації API, який використовуватиме клієнтська частина.

Після підготовки середовища можна переходити до створення самого проєкту. Спочатку потрібно встановити Nest CLI. Це корисно, оскільки дозволяє швидко створити базову структуру проєкту.

Щоб створити новий проєкт, спочатку потрібно встановити CLI, а потім виконати команду ініціалізації:

```
$ npm i -g @nestjs/cli & nest new server
```

Після цього створюється папка з основними файлами, і тепер можна запустити сервер у режимі розробки.

Наступний крок – підключення бази даних. Для цього використано бібліотеку TypeORM. Це дозволяє комфортно працювати з об’єктно-орієнтованими базами даних без необхідності вручну писати велику кількість SQL-запитів [27].

Щоб встановити TypeORM та драйвер PostgreSQL, потрібно запустити команду:

```
$ npm install @nestjs/typeorm typeorm pg
```

Після цього створено окремий файл для зчитування налаштувань бази даних (див. лістинг 3.1).

Лістинг 3.1 – Код підключення до бази даних

```
import { registerAs } from '@nestjs/config';
export default registerAs('db', () => ({
  type: process.env.DATABASE_TYPE || 'postgres',
  host: process.env.DATABASE_HOST,
  port: parseInt(process.env.DATABASE_PORT, 10) || 3306,
  username: process.env.DATABASE_USERNAME,
  database: process.env.DATABASE_NAME,
}));
```

Для зручного керування конфігурацією створено файл .env, який зберігає всі змінні середовища. Це дозволяє швидко змінювати налаштування без необхідності редагування коду. Приклад .env файлу наведено у лістингу 3.2.

Лістинг 3.2 – Змінні середовища проєкту

```
PORT=7777

DATABASE_TYPE=postgres
DATABASE_HOST=localhost
DATABASE_PORT=5432
DATABASE_USERNAME=postgres
DATABASE_PASSWORD=12345678
DATABASE_NAME=2024_12_27_diploma
```

Точкою входу в програму є файл main.ts. Тут запускається сервер, вносяться глобальні налаштування та активується валідація. Приклад коду показано в лістингу 3.3.

Лістинг 3.3 – Код файлу main.ts

```
import { NestFactory } from '@nestjs/core';
import { ValidationPipe } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';

import { AppModule } from './app.module';

async function bootstrap() {
  const app = await NestFactory.create(AppModule, { cors: true });

  app.setGlobalPrefix('api/v1');
  app.useGlobalPipes(new ValidationPipe());

  const configService = app.get(ConfigService);

  await app.listen(configService.get('app.port'));
}

bootstrap();
```

Все це дозволяє швидко запустити сервер і продовжити роботу над логікою програми.

3.2.1 Створення сутностей

Для роботи з базою даних потрібно описати які таблиці вона міститиме та що саме в ній буде зберігатися. У NestJS це називається створенням сутності. Кожна сутність є таблицею в базі даних, яка зберігає певну інформацію. Наприклад, сутність користувача відповідатиме таблиці users.

На рівні сервера кожна сутність також має свій власний інтерфейс, тобто опис того, які поля вона має, які типи даних там зберігаються та як з ними працюється.

Для сутності user створено окремий інтерфейс – простий опис даних, доступних користувачеві (див. лістинг 3.4).

Лістинг 3.4 – Інтерфейс UserInterface

```
import { UserRole } from '../user.enums';

export interface UserInterface {
  id: string;
  createdAt: Date;
  updatedAt: Date;
  username: string;
  password: string;
  role: UserRole;
}
```

Далі створено саму сутність, яка буде зберігатися в базі даних (див. лістинг 3.5).

Лістинг 3.5 – Сутність User

```
import {
  Column,
  CreateDateColumn,
  Entity,
  PrimaryGeneratedColumn,
  UpdateDateColumn,
} from 'typeorm';

import { UserInterface } from '../interface/user.interface';
import { UserRole } from '../user.enums';
```

```
@Entity('users')
export class User implements UserInterface {
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @CreateDateColumn({ type: 'timestamp' })
  createdAt: Date;

  @UpdateDateColumn({ type: 'timestamp' })
  updatedAt: Date;

  @Column({ unique: true })
  username: string;

  @Column()
  password: string;

  @Column({ type: 'enum', enum: UserRole })
  role: UserRole;
}
```

У цьому прикладі описано, що сутність User має такі поля: id (унікальний ідентифікатор), createdAt (дата створення сутності), updatedAt (дата оновлення сутності), username (ім’я користувача), password (пароль каристувача) та role (роль користувача). Ця структура автоматично створить відповідну таблицю в базі даних, з якою в подальшому необхідно працювати.

3.2.2 Взаємодія з базою даних

Щоб мати змогу комфортно працювати з базою даних, створено та використано окремий шар під назвою репозиторій. Це своєрідний посередник між самою базою даних та бізнес-логікою. Репозиторій дозволяє зручно читати, створювати та оновлювати дані без необхідності щоразу вручну писати запити до бази даних.

Для кожної сутності потрібен окремий інтерфейс репозиторію, щоб одразу було зрозуміло, які методи знаходяться в цьому класі. Приклад інтерфейсу показано в лістингу 3.6.

Лістинг 3.6 – Інтерфейс UserRepositoryInterface

```
import { UserInterface } from './user.interface';

export interface UserRepositoryInterface {
  create(entity: UserInterface): Promise<UserInterface>;
  getAll(): Promise<UserInterface[]>;
  getById(id: string): Promise<UserInterface>;
  getByUsername(username: string): Promise<UserInterface>;
}
```

А ось приклад того, як сам клас репозиторію реалізує ці методи. Він використовує TypeORM для роботи з базою даних. Приклад класу показано в лістингу 3.7.

Лістинг 3.7 – Клас UserRepository

```
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';

import { UserRepositoryInterface } from './interface/user-
repository.interface';
import { User } from './entities/user.entity';
import { UserInterface } from './interface/user.interface';

export class UserRepository implements UserRepositoryInterface {
  constructor(
    @InjectRepository(User) private readonly userRepository:
Repository<User>,
  ) {}

  create(entity: UserInterface): Promise<UserInterface> {
    return this.userRepository.save(entity);
  }

  getAll(): Promise<UserInterface[]> {
    return this.userRepository.find();
  }

  getById(id: string): Promise<UserInterface> {
    return this.userRepository.findOne({ where: { id } });
  }

  getByUsername(username: string): Promise<UserInterface> {
    return this.userRepository.findOne({ where: { username } });
  }
}
```

Це означає, що всі запити до бази даних зібрані в одному місці, що значно спрощує підтримку та розширення проєкту. Репозиторії для інших модулів написані по аналогії.

3.2.3 Організація бізнес-логіки

Бізнес-логіка веб-застосунку зосереджена в сервісах. Це окремий шар, що відповідає за виконання основних дій із сутностями. Він не працює з базою даних напряму, а використовує вже написані раніше репозиторії. Таким чином, код стає чистішим та зрозумілішим.

Прикладом є клас UserService, який відповідає за створення, пошук та отримання користувачів. Нижче наведено приклад одного з методів класу – create, код якого наведено в лістингу 3.8.

Лістинг 3.8 – Метод create

```
async create(dto: CreateUserDTO): Promise<UserInterface> {
  const { username, password, role } = dto;

  if (await this.userRepository.getByUsername(username)) {
    throw new BadRequestException('This username is already
taken');
  }

  const user = new User();

  user.username = username;
  user.password = await hash(password, this.hashSalt);
  user.role = role;

  return this.userRepository.create(user);
}
```

Цей метод перевіряє, чи вже існує користувач з таким логіном, хешує пароль та створює нового користувача через репозиторій. Окрім методу create, сервіс має інші методи, описані в окремому інтерфейсі UserServiceInterface (див. лістинг 3.9).

Лістинг 3.9 – Інтерфейс UserServiceInterface

```
import { CreateUserDTO } from '../dto/create-user.dto';
import { UserInterface } from './user.interface';

export interface UserServiceInterface {
  create(dto: CreateUserDTO): Promise<UserInterface>;
  getAll(): Promise<UserInterface[]>;
  getById(id: string): Promise<UserInterface>;
  getByUsername(username: string): Promise<UserInterface>;
}
```

Усі ці методи реалізовані в класі UserService та дозволяють зручно співпрацювати з користувачами без дублювання коду. Бізнес-логіка інших модулів написана у відповідних сервісах по аналогії.

Програмний код та інші файли модуля user наведено в додатку Б.

3.2.4 Реалізація REST API на рівні контроллерів

Для того, щоб клієнтська сторона (фронтенд) могла отримувати або надсилати дані на сервер, потрібен певний тип взаємодії. Найзручнішим і найпоширенішим підходом є REST API. Це набір чітких правил, які дозволяють обмін даними між клієнтом і сервером через звичайні HTTP-запити (GET, POST, PUT, DELETE) [28].

У NestJS ця логіка реалізована за допомогою контролерів. Контролер – це місце, де ми отримуємо запити, викликаємо необхідні сервіси та повертаємо відповідь. У лістингу 3.10 наведено приклад реалізації класу UserController.

Лістинг 3.10 – Клас UserController

```
import {
  Body,
  Controller,
  Get,
  Inject,
  Param,
  Post,
  UseGuards,
} from '@nestjs/common';
```

```
import { CreateUserDTO } from './dto/create-user.dto';
import { USER_SERVICE } from './user.constants';
import { UserServiceInterface } from './interface/user-service.interface';
import { UserInterface } from './interface/user.interface';
import { AuthGuard } from '../auth/auth.guard';

@Controller('users')
export class UserController {
  constructor(
    @Inject(USER_SERVICE) private readonly userService: UserServiceInterface,
  ) {}

  @UseGuards(AuthGuard)
  @Post()
  create(@Body() dto: CreateUserDTO): Promise<UserInterface> {
    return this.userService.create(dto);
  }

  @Get()
  getAll(): Promise<UserInterface[]> {
    return this.userService.getAll();
  }

  @Get('/:id')
  getById(@Param('id') id: string): Promise<UserInterface> {
    return this.userService.getById(id);
  }
}
```

Тепер сервер готовий приймати та обробляти HTTP-запити. Усі контроллери в інших модулях написані по аналогії.

3.2.5 Тестування REST API

Після впровадження REST API потрібно переконатися, що все працює належним чином. Для цього використано популярний сервіс Postman.

За допомогою Postman можна легко надсилати HTTP-запити на сервер, переглядати відповіді та тестувати всі доступні запити. Це дозволяє перевірити, як працює створення користувача, отримання всіх користувачів або, наприклад, отримання одного користувача за унікальним ідентифікатором [29].

Окрім тестування, Postman також можна зручно використовувати як місце для зберігання документації. У ньому можна створювати колекції запитів, описувати кожен запит, додавати приклади запитів та відповідей – все це допомагає краще зрозуміти структуру API та полегшує подальшу роботу з проєктом.

На рисунку 3.1 зображено приклад тестування одного з запитів.

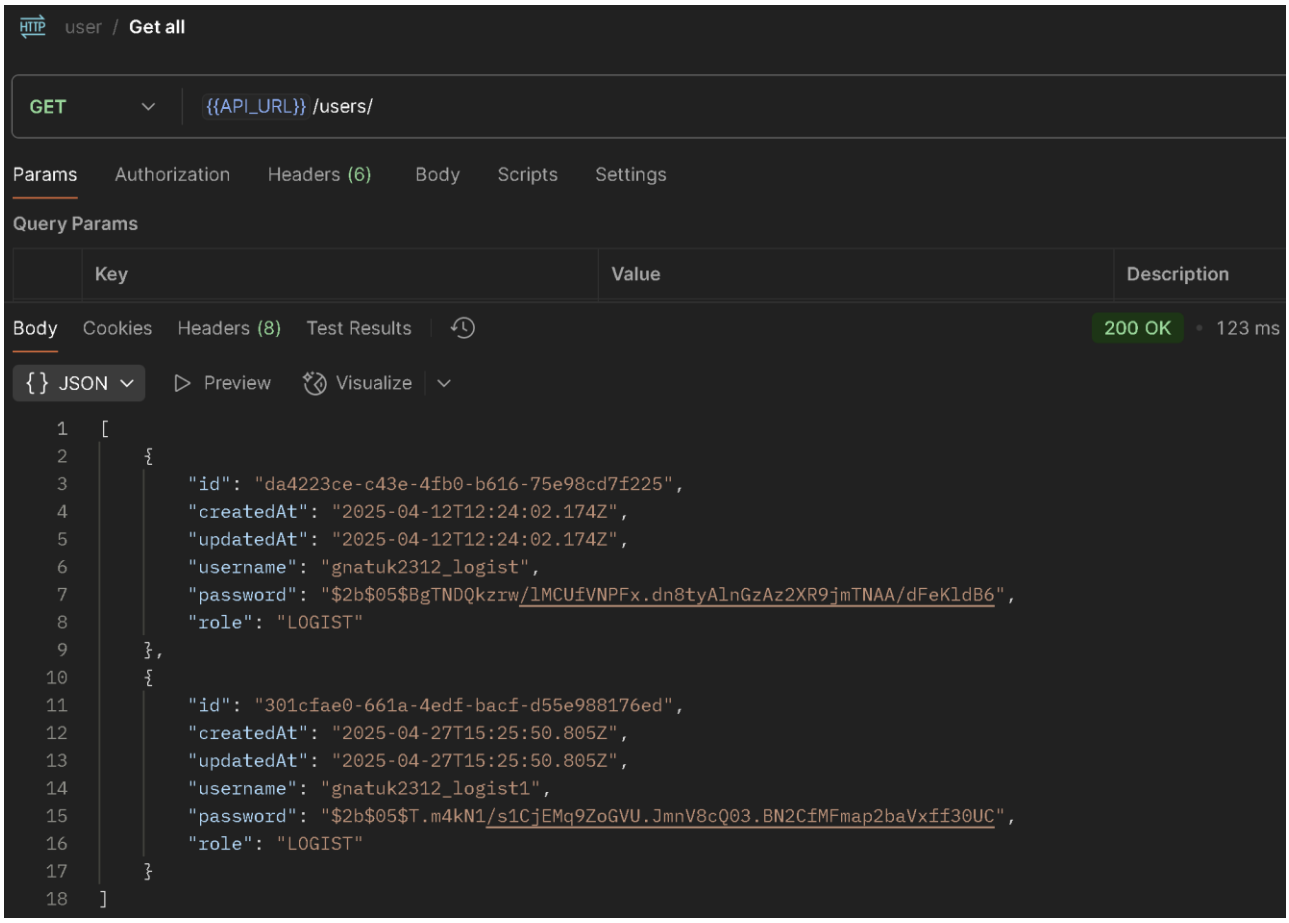

Рисунок 3.1 – Тестування запиту getAllUsers

Аналогічно задокументовано та протестовано інші модулі серверної частини веб-застосунку.

3.3 Створення клієнтської частини

Клієнтська частина – це те, що бачить користувач і з чим взаємодіє. Вона відповідає за відображення даних, отриманих від сервера, у привабливому вигляді та надання користувачеві можливості комфортно з ними працювати.

Тут потрібно заповнювати форми, списки, натискати кнопки – все, що допомагає людині виконувати дії, не замислюючись про те, що відбувається "під капотом". Наприклад, коли користувач створює новий обліковий запис або відкриває список доступних замовлень, клієнтська сторона відповідає за те, щоб ці дії виглядали простими та зрозумілими.

Головне завдання цього розділу – реалізувати інтерфейс, який працює швидко та зручно, а також має гарний вигляд.

3.3.1 Створення та налаштування проєкту

Роботу над клієнтською частиною розпочато зі створення нового проєкту на базі Next.js з використанням TypeScript.

Щоб створити новий проєкт, потрібно виконати команду:

```
$ npx create-next-app@latest . –typescript
```

Після цього проєкт готовий до налаштування.

Щоб створити гарний користувацький інтерфейс, використано бібліотеку компонентів Material UI. Вона пропонує готові кнопки, поля введення, картки та інші елементи, якими легко користуватися. Щоб додати цю бібліотеку, потрібно виконати таку команду:

```
$ npm install @mui/material @emotion/react @emotion/styled
```

Також створено власну тему для проєкту, в якій визначено основні кольори та шрифти. Код показано в лістингу 3.11.

Лістинг 3.11 – Конфігурація теми проєкту

```
"use client";
import { createTheme } from "@mui/material/styles";
import { green, grey, red, yellow } from "@mui/material/colors";
const theme = createTheme({
  cssVariables: true,
  palette: {
    primary: green,
    secondary: grey,
    error: red,
    warning: yellow,
  },
  typography: {
    fontFamily: "var(--font-roboto)",
  },
});
export default theme;
```

Для взаємодії з бекендом (наприклад, для отримання або надсилання даних), використано бібліотеку Axios. Вона проста у використанні та дозволяє комфортно працювати з HTTP-запитами.

Щоб встановити Axios, потрібно виконати команду:

```
$ npm install axios
```

Після встановлення створено власну конфігурацію, яка включає базовий шлях до сервера, час очікування відповіді та перехоплювачі для обробки помилок і відповідей. Код наведено в лістингу 3.12.

Лістинг 3.12 – Конфігурація бібліотеки axios

```
import axios from "axios";
import {
  transformErrorInterceptor,
  transformResponseInterceptor,
} from "./interceptors";
const baseAxiosInstance = axios.create({
  baseURL: process.env.NEXT_PUBLIC_API_URL,
```

```
  timeout: 10000,
  headers: { "Content-Type": "application/json" },
});
baseAxiosInstance.interceptors.response.use(
  transformResponseInterceptor,
  transformErrorInterceptor
);
export default baseAxiosInstance;
```

Завдяки такому налаштуванню клієнтська частина зручно та швидко взаємодіє із сервером і має охайний та унікальний вигляд.

3.3.2 Планування та реалізація сторінок

На етапі планування, створено схему сторінок, для кращого розуміння, як виглядатиме структура веб-застосунку.

Ця діаграма допомагає зрозуміти, як організована навігація та взаємодія між сторінками (див. рис. 3.2).

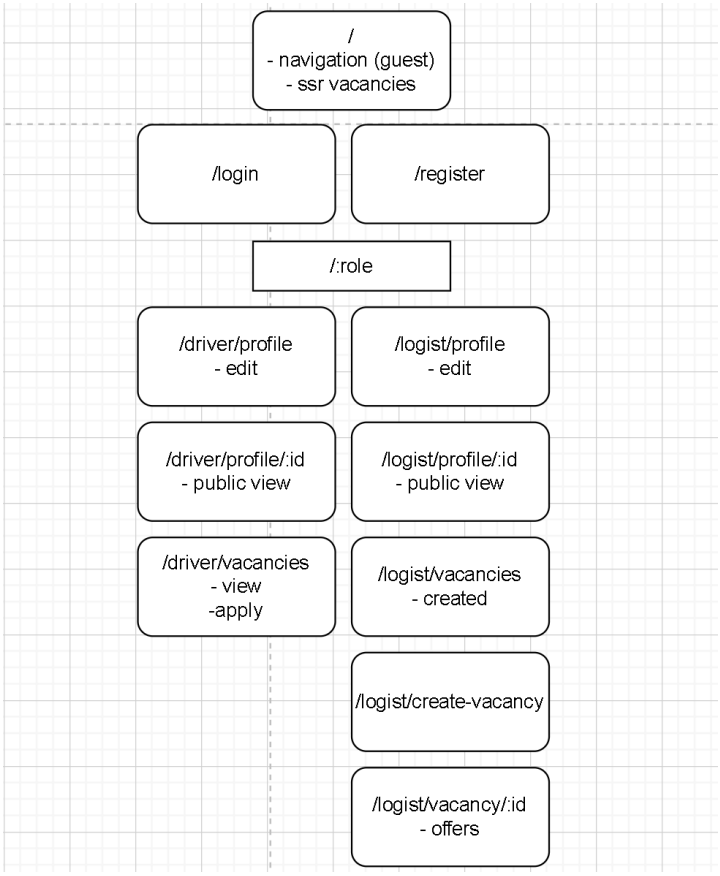

Рисунок 3.2 – Схема сторінок веб-застосунку

У Next.js маршрутизація легко створюється за допомогою файлової структури в папці app/. Кожна папка в цій директорії відповідає окремій сторінці або групі сторінок. Наприклад, файл page.tsx у папці app/ буде головною сторінкою, а файли у вкладених папках (наприклад, app/logist/page.tsx) створюють сторінки для відповідних маршрутів. Це спрощує додавання нових сторінок без необхідності налаштовувати маршрути вручну.

На рисунку 3.3 навдено файлову структуру папки app/, для кращого розуміння реалізації роутингу.

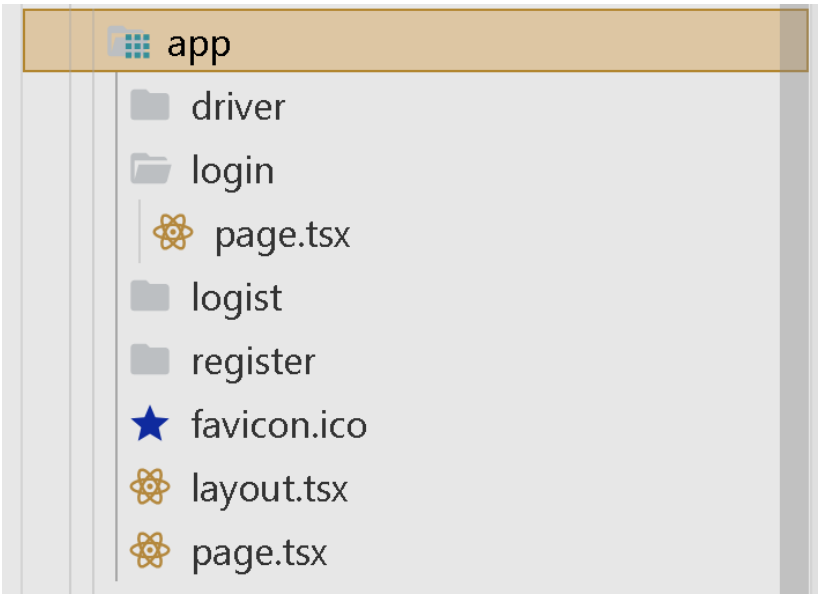

Рисунок 3.3 – Файлова структура у папці app/

Приклад програмного коду головної сторінки наведено в додатку А.

3.3.3 Авторизація

У цьому веб-застосунку є два типи користувачів: водії та логісти. Ці ролі вибираються на сторінці реєстрації. Користувачі обирають свою роль під час реєстрації, заповнюючи форму з полями для імені, пароля та ролі.

Приклад реалізації методу обробки даних форми реєстрації показано в лістингу 3.13.

Лістинг 3.13 – Інтерфейс полів та метод форми реєстрації onSubmit

```
interface RegisterFormInterface {
  username: string;
  password: string;
  role: UserRole;
}

const onSubmit = async (data: RegisterFormInterface) => {
    try {
      const auth = await signUpRequest({ body: data });

      if (auth) {
        setAuth(auth);
        showSnackbar("Account created successfully", "success");
        push("/");
      }
    } catch (error) {
      const { message } = error as AxiosErrorDataInterface;

      showSnackbar(message, "info");
    }
};
```

Уся логіка створення користувача реалізована на сервері, клієнт просто викликає функцію signUpRequest(), реалізація якої показана в лістингу 3.14.

Лістинг 3.14 – Функція signUpRequest та її інтерфейси

```
export interface SignUpArgumentsInterface {
  body: {
    username: string;
    password: string;
    role: UserRole;
  };
}

export interface AuthInterface {
  accessToken: string;
  refreshToken: string;
  user: UserInterface;
}

export const signUpRequest = (
  args: SignUpArgumentsInterface
): Promise<AuthInterface> => {
  const { body } = args;

  return axios.post("/auth/sign-up", body);
};
```

Тепер, коли користувач має свою роль, це значення використане для правильного відображення потрібних сторінок. Залежно від того, чи є користувач, наприклад, водієм чи логістом, можна надавати/обмежувати доступ до різних функцій або інтерфейсів на веб-сайті.

3.4 Зберігання коду та хостинг веб-застосунку

Для зберігання коду створено два окремих репозиторії на GitHub – один для клієнтської частини проєкту, а інший для серверної. Це дозволяє зручно організувати код та розділити логіку проєкту на дві частини, що спрощує подальшу роботу, оновлення та підтримку. Такий підхід дозволяє уникнути плутанини, а також спрощує роботу над проєктом.

Посилання на репозиторії:

– клієнтська частина: https://github.com/gnatuk2312/diploma-client;

– серверна частина: https://github.com/gnatuk2312/diploma-server.

Хостинг веб-застосунку – це останній крок перед його запуском на ринок. Це дозволяє зробити проєкт доступним для користувачів з усього світу. Після налаштування та тестування веб-застосунку на локальному сервері, його розгорнуто на платформі Netlify, що забезпечує автоматичне розгортання та швидку роботу з оновленнями в подальшому. Посилання на веб-застосунок: https://ukraine-logistics.netlify.app/.

Netlify має чудову інтеграцію з GitHub, що дозволяє автоматично розгортати нові версії. Після того, як додаються нові зміни до репозиторію, Netlify автоматично виявляє та розгортає їх на хостингу, що значно спрощує процес розгортання та оновлення проєкту.

На рисунку 3.4 показано інтерфейс Netlify та розгорнутого на ньому веб-застосунку.

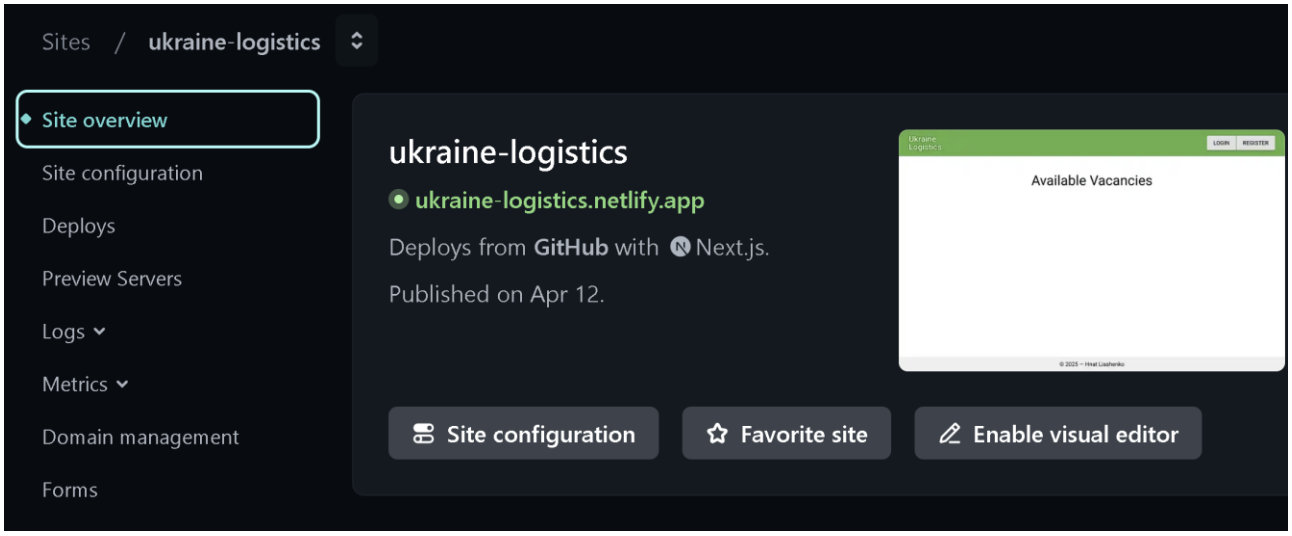

Рисунок 3.4 – Інтерфейс головної сторінки Netlify

Базове використання цієї платформи є безкоштовним і тому підходить для невеликих проєктів або створення прототипів. Всі налаштування здійснюються через простий інтерфейс, що дозволяє ефективно керувати проєктом без зайвих клопотів.

Для хостингу бекенду та бази даних використано сервіс DigitalOcean. Придбано окремий віртуальний сервер під назвою Droplet. На рисунку 3.5 показано інтерфейс цього сервісу.

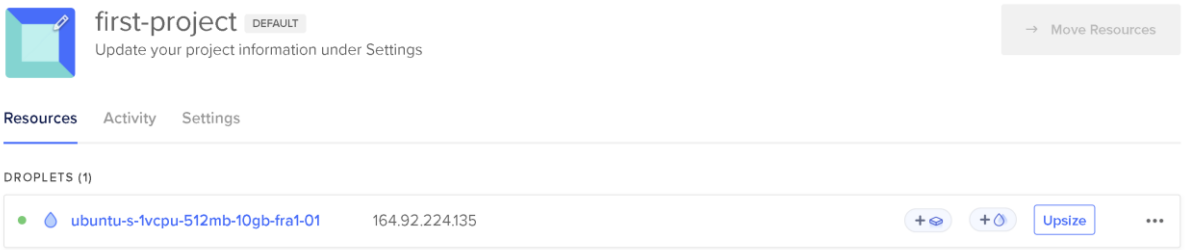

Рисунок 3.5 – Інтерфейс головної сторінки DigitalOcean

Після створення Droplet до нього потрібно підключитись через SSH. Це спосіб віддаленого доступу до сервера через термінал. Там потрібно встановити та налаштувати базу даних та розгорнути бекенд застосунку. Тепер, щоб зв’язатися із сервером, використовуємо IP-адресу цього Droplet 164.92.224.135 (див. рисунок 3.6).

```
root@DESKTOP-CU7IIBB:~# ssh root@164.92.224.135
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-113-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Sat May  3 12:27:50 UTC 2025

  System load:  0.07              Processes:             107
  Usage of /:   99.8% of 9.51GB   Users logged in:       0
  Memory usage: 59%               IPv4 address for eth0: 164.92.224.135
  Swap usage:   4%                IPv4 address for eth0: 10.19.0.5

  => / is using 99.8% of 9.51GB

Expanded Security Maintenance for Applications is not enabled.

160 updates can be applied immediately.
89 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Sat Apr 12 12:29:06 2025 from 5.58.181.30
root@ubuntu-s-1vcpu-512mb-10gb-fra1-01:~#
```

Рисунок 3.6 – Підключення до віртуальної машини через SSH

Все працює стабільно та швидко, а інтерфейс DigitalOcean зручний та зрозумілий навіть для тих, хто не має великого досвіду в роботі з віртуальними машинами.

3.5 Тестування веб-застосунку

Після завершення розробки веб-застосунку було проведено його тестування з метою перевірки як працюють основні функції, чи правильно відображається інтерфейс, і чи може користувач зручно взаємодіяти із системою. Застосунок містить декілька основних сторінок, які доступні залежно від ролі користувача (логіст або водій), а також є загальнодоступна головна сторінка для всіх відвідувачів (див. рис. 3.7).

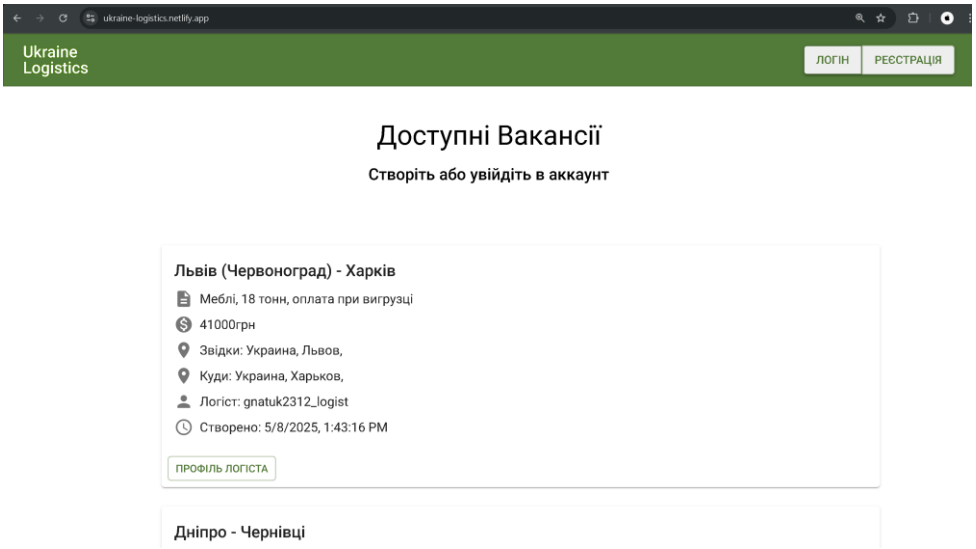

Рисунок 3.7 – Головна сторінка

Починається робота з головної сторінки, яка доступна для неавторизованих користувачів. На ній коротко пояснюється призначення системи та доступні варіанти переходу – реєстрація нового користувача або вхід у систему. Це дозволяє новим користувачам створити обліковий запис, а зареєстрованим – увійти за допомогою свого логіна та пароля (див. рис. 3.8).

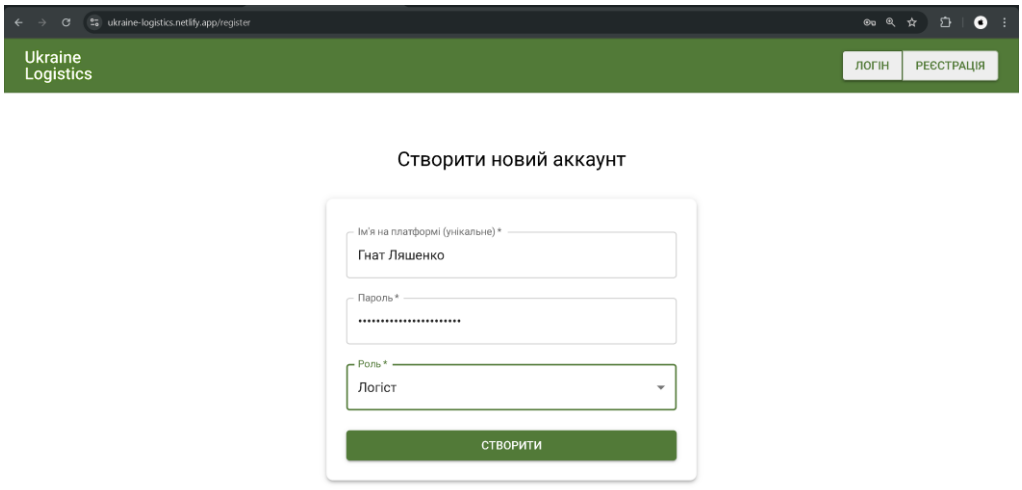

Рисунок 3.8 – Сторінка авторизації

У ході тестування спочатку перевіримо функціональність системи, увійшовши як логіст. Після авторизації відкривається панель логіста, звідки можна перейти до сторінки створення нової заявки на вантажне перевезення.

На цій сторінці користувач заповнює потрібні поля – наприклад, пункт відправлення, пункт призначення, опис вантажу та інші деталі. Після натискання кнопки «Створити Вакансію» система зберігає нову заявку в базу даних і відображає її у загальному списку (див. рис. 3.9).

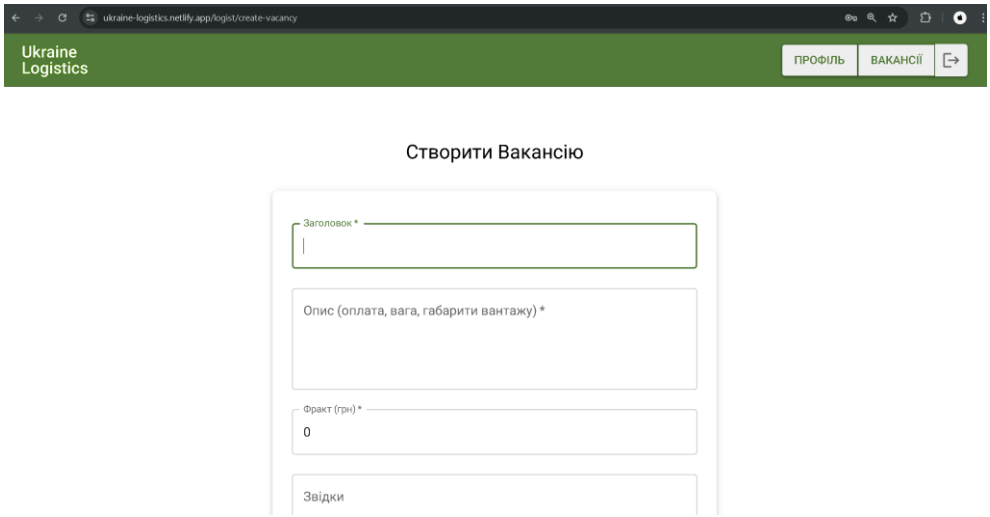

Рисунок 3.9 – Сторінка логіста для створення вакансії

Наступним кроком є перевірка роботи застосунку з роллю водія. Після авторизації водій потрапляє на сторінку перегляду доступних заявок. Тут відображаються всі заявки, які створили логісти. Як бачимо, новостворена заявка одразу з’явилася у списку – це означає, що база даних працює правильно, а передача інформації між логістами та водіями реалізована успішно (див. рис. 3.10).

Рисунок 3.10 – Сторінка водія для перегляду вакансії

Таким чином, у результаті тестування було підтверджено, що функції входу, реєстрації та виходу з профілю працюють стабільно, система правильно розмежовує доступ залежно від ролі користувача, логіст може успішно створювати заявки, водій бачить нові заявки без потреби оновлення сторінки, усі дії користувачів зберігаються в базі даних та відображаються в інтерфейсі.

3.6 Висновок до третього розділу

У цьому розділі детально описано процес створення веб-застосунку – від налаштування бекенду, організації бізнес-логіки та створення REST API до реалізації клієнтської частини та хостингу проєкту. Кожен етап був продуманий таким чином, щоб забезпечити зручність як для кінцевого користувача, так і для розробника. Структура коду організована: сервер і клієнт розділені на окремі репозиторії, використовуються сучасні технології, як-от Next.js, TypeScript, Material UI, NestJS та інші. Особливу увагу було приділено авторизації та ролям користувачів, щоб кожен мав доступ лише до відповідної інформації. Зрештою, було розгорнуто як клієнтську, так і серверну частини, що зробило проєкт повністю функціональним.

Цей веб-застосунок наразі є MVP – мінімально життєздатним продуктом, створеним для тестування на реальних користувачах. Головна мета – з’ясувати, чи справді це відповідає їхнім очікуванням та вирішує проблеми конкурентів. Таким чином можна збирати відгуки, робити корисні висновки та покращувати програму в майбутніх версіях.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Правове забезпечення та організаційно-функціональна структура захисту населення

Забезпечення безпеки життєдіяльності населення є важливою складовою національної безпеки будь-якої держави. В Україні цю сферу регулює низка законодавчих актів, які визначають обов'язки державних органів, підприємств, організацій і громадян у сфері захисту життя і здоров'я. Основою правового забезпечення є Конституція України, де вказано, що людина, її життя і здоров'я визнаються найвищою соціальною цінністю. У межах цієї конституційної норми сформована ціла система нормативно-правових актів, що охоплюють захист населення у разі надзвичайних ситуацій, охорону праці, санітарно-епідеміологічне благополуччя, екологічну безпеку та інші важливі напрями.

Одним із ключових законів є Закон України «Про цивільний захист», який визначає єдину державну систему цивільного захисту. Ця система покликана запобігати надзвичайним ситуаціям, ліквідовувати їх наслідки, забезпечувати оповіщення населення, евакуацію та надання допомоги постраждалим. Організаційно вона включає центральні органи виконавчої влади, місцеві адміністрації, органи місцевого самоврядування, підприємства, установи й організації, а також громадян, які зобов'язані виконувати визначені заходи безпеки. Центральним органом у цій системі є Державна служба України з надзвичайних ситуацій (ДСНС), яка координує дії з ліквідації загроз та надзвичайних ситуацій, проводить навчання, організовує рятувальні заходи та інформує населення.

Забезпечення безпеки життєдіяльності є важливим напрямом державної політики України. Основоположним документом у цій сфері є Конституція України, стаття 3 якої визначає, що життя і здоров'я людини є найвищою соціальною цінністю. Це закріплено також у Цивільному кодексі України (ст. 281), де гарантується право кожної особи на життя і безпеку.

Серед ключових нормативно-правових актів, які регулюють безпеку громадян:

- Закон України «Про охорону праці» № 2694-XII від 14.10.1992 – встановлює державну політику в галузі охорони праці, права працівників на безпечні умови.
- Закон України «Про цивільний захист» № 1706-IV від 24.06.2004 – регламентує єдину державну систему цивільного захисту.
- Кодекс законів про працю України (КЗпП) – містить норми щодо гарантій безпеки праці (розділ XI).

Кримінальний кодекс України, зокрема:

- ст. 271 – «Порушення вимог законодавства про охорону праці», що передбачає відповідальність за недотримання правил безпеки;
- ст. 272 – «Порушення правил безпеки під час виконання робіт з підвищеною небезпекою».

Організаційно-функціональна структура захисту населення включає:

- Державну службу України з надзвичайних ситуацій (ДСНС) – головний координатор дій під час надзвичайних ситуацій;
- органи виконавчої влади, місцевого самоврядування, підприємства, які зобов’язані забезпечити евакуацію, оповіщення та захист населення;
- медичні, освітні, транспортні та критичні інфраструктурні установи, кожна з яких розробляє та реалізує власні плани дій у разі загроз.

Функціонування системи захисту здійснюється на основі централізованого управління з елементами територіального та галузевого підходу, що забезпечує оперативність реагування та адаптацію до специфіки регіону.

У межах підприємств, установ та організацій безпека життєдіяльності забезпечується шляхом впровадження систем управління охороною праці, інструктажів з безпеки, оцінки ризиків та планування дій у надзвичайних ситуаціях. Керівники зобов’язані організовувати навчання працівників з питань безпеки, а також створювати безпечні умови праці. На рівні територіальних

громад реалізація заходів захисту покладається на виконавчі органи місцевої влади, які відповідають за евакуацію, укриття населення, оповіщення та надання першої допомоги.

Окрему роль у структурі захисту населення відіграють медичні установи, заклади освіти, транспортні підприємства й об'єкти критичної інфраструктури. Кожен із них має спеціальні плани дій у разі загрози або виникнення надзвичайних ситуацій. Загалом організаційно-функціональна структура захисту населення в Україні функціонує за принципом централізованого управління з елементами територіального та галузевого підходу.

4.2 Особливості безпеки під час вантажно-розвантажувальних робіт

Вантажно-розвантажувальні роботи пов’язані з підвищеною небезпекою як для робітників, так і для оточуючих, тому питання безпеки у цій сфері мають важливе значення. При виконанні таких робіт задіяні вантажопідйомні механізми, транспортні засоби, різні види тари та обладнання, а також працівники, які безпосередньо взаємодіють з цими об’єктами. Усі ці фактори створюють ризик отримання травм, аварій, пошкодження майна або навіть загрози життю.

Основними умовами безпечної організації вантажно-розвантажувальних робіт є правильне планування процесу, дотримання інструкцій, підготовка персоналу та контроль за станом обладнання. Роботи мають проводитися лише навченим персоналом, який пройшов відповідний інструктаж з охорони праці. Перед початком змін необхідно перевірити технічний стан вантажопідйомної техніки, стропів, піддонів, механізмів та іншого обладнання. Заборонено використовувати несправну техніку або застарілі засоби кріплення вантажів, оскільки це може призвести до обвалення або падіння вантажу.

Важливою умовою є правильна організація робочого простору. Площадки для розвантаження повинні бути рівними, чистими, без сторонніх предметів, а освітлення – достатнім для безпечного виконання операцій. Особливу увагу

слід приділяти зберіганню та штабелюванню вантажів. Вони мають бути стійко укладені, не перевищувати встановлену висоту, а також бути захищеними від можливого зміщення.

Згідно з Правилами охорони праці під час вантажно-розвантажувальних робіт, затвердженими наказом Мінсоцполітики № 1140 від 27.11.2018:

Максимальна вага вантажу, що переноситься вручну:

- для чоловіків – не більше 50 кг при разовому підйомі;
- для жінок – не більше 10 кг постійно, 15 кг – періодично;
- для осіб віком 16–18 років – не більше 16,4 кг (хлопці), 7 кг (дівчата).

Штабелювання вантажів:

- висота не повинна перевищувати 1,7 м при ручному укладанні;
- при використанні механізмів – не більше 3,5 м, залежно від виду вантажу і методу кріплення.

Довжина вантажу, який переміщується вручну, не повинна перевищувати 2,5 м, якщо його маса понад 30 кг.

Для уникнення травматизму всі працівники повинні використовувати засоби індивідуального захисту: каски, рукавиці, спецвзуття, жилети з елементами світловідбиття. У разі виконання робіт з небезпечними речовинами або пиловими матеріалами – обов’язковим є використання респіраторів або захисних масок. Також не можна нехтувати вимогами сигналізації та маркування небезпечних зон.

Додаткову увагу слід приділяти комунікації між працівниками. Використання звукових або світлових сигналів під час переміщення вантажів допомагає уникнути випадкових зіткнень або несанкціонованих дій. Бригадир або відповідальна особа має постійно контролювати хід робіт, оперативно реагувати на порушення або потенційні загрози.

Крім того, важливою є організація безпечного руху транспортних засобів у зоні розвантаження. Має бути встановлено обмеження швидкості, визначено

маршрути руху, а місця навантаження – відгороджені від проходів пішоходів. Це дозволяє мінімізувати ризик наїзду або зіткнення.

Безпека вантажно-розвантажувальних робіт залежить не лише від технічного стану обладнання, але й від рівня культури виробництва. Там, де працівники дотримуються правил, проходять регулярні навчання, а керівництво приділяє увагу умовам праці, рівень травматизму істотно нижчий.

4.3 Висновок до четвертого розділу

Безпека життєдіяльності та охорона праці – це ті сфери, які потребують системного підходу, постійного контролю та участі кожного. Українське законодавство забезпечує необхідну правову базу для захисту населення в умовах надзвичайних ситуацій, а також на робочих місцях. Організаційно-функціональна структура побудована таким чином, щоб забезпечити оперативне реагування, запобігання загрозам і надання допомоги у кризових ситуаціях.

Особливої уваги потребують роботи, пов’язані з підвищеною небезпекою, такі як вантажно-розвантажувальні операції. У таких випадках дотримання правил безпеки, технічних норм, інструкцій та використання засобів захисту є запорукою збереження життя і здоров’я працівників. Усі учасники виробничого процесу мають розуміти важливість своєї ролі у збереженні безпечного середовища праці, а керівництво повинно створювати всі умови для цього.

Таким чином, лише спільними зусиллями держави, роботодавців і працівників можливо досягти високого рівня безпеки як у повсякденному житті, так і в професійній діяльності.

ВИСНОВКИ

Створено веб-застосунок для покращення логістичних процесів в Україні, який поєднав найкращі бізнес-моделі конкурентів, спростив інтерфейс для зручнішої взаємодії користувачів та усунув основні недоліки існуючих рішень на ринку.

В першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр» було проведено аналіз проблем логістичної сфери в Україні та предметної області загалом. Також було детально вивчено конкурентів, їх сильні і слабкі сторони, що дало змогу визначити ключові напрямки для вдосконалення.

В другому розділі описано розробку архітектури веб-застосунку, вибір технологій, а також створення use-case та ER-діаграм, які допомогли структурувати роботу системи та побудувати логіку взаємодії між користувачами.

У третьому розділі наведено опис процесу реалізації застосунку: розробку бекенду і фронтенду, налаштування хостингу бази даних, підключення домену та організацію зберігання коду проекту, що забезпечило створення повноцінного функціонального продукту.

У розділі «Безпека життєдіяльності, основи охорони праці» висвітлено основи правового забезпечення захисту населення та функціонування системи цивільного захисту в Україні, а також розглянуто ключові аспекти безпеки праці під час вантажно-розвантажувальних робіт. Детально проаналізовано вимоги до організації безпечного робочого середовища, засобів захисту працівників і технічних заходів попередження виробничого травматизму. Розуміння та дотримання принципів безпеки на всіх рівнях дозволяє не лише мінімізувати ризики, а й забезпечити ефективну та стабільну роботу створеного веб-застосунку в реальних умовах логістичної галузі.

ПЕРЕЛІК ДЖЕРЕЛ

1 Nebesnyi, R., Kunanets, N., Vaskiv, R., & Veretennikova, N. (2021, September). Formation of an IT Project Team in the Context of PMBOK Requirements. In *2021 IEEE 16th International Conference on Computer Sciences and Information Technologies (CSIT)* (Vol. 2, pp. 431-436). IEEE. (дата звернення 20.05.2025)

2 Strutynska, I., Kozbur, G., Dmytrotsa, L., Sorokivska, O., & Melnyk, L. (2019, June). Influence of digital technology on roadmap development for digital business transformation. In *2019 9th International Conference on Advanced Computer Information Technologies (ACIT)* (pp. 333-337). IEEE. (дата звернення 20.05.2025)

3 DELLA™ Вантажні перевезення. [Електронний ресурс] // DELLA™. – 2025. – Режим доступу до ресурсу: https://della.com.ua/. (дата звернення 20.05.2025)

4 Lardi-Trans: Платформа для професіоналів у сфері вантажоперевезень. [Електронний ресурс] // Lardi-Trans. – 2025. – Режим доступу до ресурсу: https://lardi-trans.ua/. (дата звернення 20.05.2025)

5 Знай свого клієнта. [Електронний ресурс] / Дописувачі Вікіпедії // Wikipedia. – 2024. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Знай_свого_клієнта. (дата звернення 20.05.2025)

6 Вантажні перевезення в Тернополі. [Електронний ресурс] // Kabanchik. – 2025. – Режим доступу до ресурсу: https://kabanchik.ua/ua/ternopil/category/vantazhni-perevezennia. (дата звернення 20.05.2025)

7 Діаграма прецедентів. [Електронний ресурс] / Дописувачі Вікіпедії // Wikipedia. – 2023. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Діаграма_прецедентів. (дата звернення 24.05.2025)

8 Entity–relationship model. [Електронний ресурс] / Wikipedia Contributors // Wikipedia. – 20023. – Режим доступу до ресурсу:

https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model. (дата звернення 24.05.2025)

9 Difference Between One-to-One, One-to-Many, Many-to-One, Many-to-Many in Neural Network. [Електронний ресурс] // i2tutorials. – 2025. – Режим доступу до ресурсу: https://www.i2tutorials.com/what-is-the-difference-between-one-to-one-one-to-many-many-to-one-many-to-many-in-neural-network/. (дата звернення 25.05.2025)

10 JavaScript. [Електронний ресурс] / Дописувачі Вікіпедії // Wikipedia. – 2025. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JavaScript. (дата звернення 27.05.2025)

11 Python. [Електронний ресурс] / Дописувачі Вікіпедії // Wikipedia. – 2025. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Python. (дата звернення 27.05.2025)

12 PHP. [Електронний ресурс] / Дописувачі Вікіпедії // Wikipedia. – 2025. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/PHP. (дата звернення 27.05.2025)

13 Java. [Електронний ресурс] / Дописувачі Вікіпедії // Wikipedia. – 2025. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Java. (дата звернення 27.05.2025)

14 C#. [Електронний ресурс] / Дописувачі Вікіпедії // Wikipedia. – 2025. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/C_Sharp. (дата звернення 27.05.2025)

15 TypeScript. [Електронний ресурс] / Дописувачі Вікіпедії // Wikipedia. – 2025. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/TypeScript. (дата звернення 27.05.2025)

16 Client-Server Model. [Електронний ресурс] / Darvish Darab // cs421_f20. – 2023. – Режим доступу до ресурсу: https://darvishdarab.github.io/cs421_f20/docs/readings/client_server/. (дата звернення 27.05.2025)

17 Node.js. [Електронний ресурс] / Дописувачі Вікіпедії // Wikipedia. – 2025. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Node.js. (дата звернення 27.05.2025)

18 NestJS. [Електронний ресурс] / Дописувачі Вікіпедії // Wikipedia. – 2025. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/NestJS. (дата звернення 27.05.2025)

19 Nebesnyi, R., Pasichnyk, V., Kunanets, N., Veretennikova, N., & Kunanets, O. (2020, September). Formation of IT Project Implementation Team. In *2020 IEEE 15th International Conference on Computer Sciences and Information Technologies (CSIT)* (Vol. 2, pp. 203-206). IEEE.

20 PostgreSQL. [Електронний ресурс] / Дописувачі Вікіпедії // Wikipedia. – 2025. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/PostgreSQL. (дата звернення 28.05.2025)

21 pgAdmin. [Електронний ресурс] / pgAdmin Project // pgAdmin. – 2025. – Режим доступу до ресурсу: https://www.pgadmin.org/. (дата звернення 28.05.2025)

22 DTO. [Електронний ресурс] / Дописувачі Вікіпедії // Wikipedia. – 2025. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/DTO. (дата звернення 28.05.2025)

23 React. [Електронний ресурс] / Дописувачі Вікіпедії // Wikipedia. – 2025. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/React. (дата звернення 28.05.2025)

24 Next.js. [Електронний ресурс] / Дописувачі Вікіпедії // Wikipedia. – 2025. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Next.js. (дата звернення 28.05.2025)

25 Strutynska, I., Dmytrotsa, L., Kozbur, H., Ermolayev, V., Mallet, F., Yakovyna, V., & Spivakovsky, A. (2019). The Main Barriers and Drivers of the Digital Transformation of Ukraine Business Structures. *ICTERI*, *1*, 50-64.

26 Material UI. [Електронний ресурс] / MUI // MUI. – 2025. – Режим доступу до ресурсу: https://mui.com/. (дата звернення 28.05.2025)

27 TypeORM. [Електронний ресурс] / TypeORM // typeorm.io. – 2025. – Режим доступу до ресурсу: https://typeorm.io/. (дата звернення 28.05.2025)

28 REST. [Електронний ресурс] / Дописувачі Вікіпедії // Wikipedia. – 2025. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/REST. (дата звернення 28.05.2025)

29 Postman (software). [Електронний ресурс] / Wikipedia Contributors // Wikipedia. – 2025. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Postman_(software). (дата звернення 28.05.2025)

ДОДАТКИ

Додаток А

Лістинг А.1 – Код головної сторінки веб-застосунку.

```
"use client";

import {
  Box,
  Typography,
  Card,
  CardContent,
  CardActions,
  Button,
  Grid,
  Dialog,
  DialogTitle,
  DialogContent,
  TextField,
  DialogActions,
  Stack,
} from "@mui/material";
import { FC, useEffect, useState } from "react";
import { useForm, Controller } from "react-hook-form";
import RoomIcon from "@mui/icons-material/Room";
import MonetizationOnIcon from "@mui/icons-
material/MonetizationOn";
import PersonIcon from "@mui/icons-material/Person";
import DescriptionIcon from "@mui/icons-material/Description";
import AccessTimeIcon from "@mui/icons-material/AccessTime";
import { VacancyInterface } from "@/interfaces/models.interface";
import { useAuth } from "@/config/auth";
import { getStatusNewVacanciesRequest } from
"@/services/api/vacancy/vacancy.api";
import { createOfferRequest } from
"@/services/api/offer/offer.api";
import { useSnackbar } from "@/config/mui/snackbar";

const VacanciesPage: FC = () => {
  const [vacancies, setVacancies] =
useState<VacancyInterface[]>([]);
  const [selectedVacancy, setSelectedVacancy] =
    useState<VacancyInterface | null>(null);
  const [open, setOpen] = useState(false);
  const { user } = useAuth();
  const { showSnackbar } = useSnackbar();

  const {
    control,
```

```
    handleSubmit,
    reset,
    formState: { isSubmitting },
  } = useForm<{ comment: string }>();

  const fetchVacancies = async () => {
    try {
      const data = await getStatusNewVacanciesRequest();
      setVacancies(data);
    } catch (e) {
      console.error("Failed to load vacancies", e);
    }
  };
  const onSubmit = async (data: { comment: string }) => {
    if (!user || !selectedVacancy) return;
    try {
      await createOfferRequest({
        body: {
          vacancyId: selectedVacancy.id,
          creatorId: user.id,
          comment: data.comment,
        },
      });
      reset();
      setOpen(false);
      showSnackbar("Ви успішно відгукнулись на вакансію",
"success");
    } catch (e) {
      console.error("Failed to create offer", e);
    }
  };

  useEffect(() => {
    fetchVacancies();
  }, []);

  return (
    <Box maxWidth="md" mx="auto" mt={5}>
      <Typography variant="h4" gutterBottom textAlign="center">
        Доступні Вакансії
      </Typography>
      <Typography variant="h6" gutterBottom textAlign="center">
        Створіть або увійдіть в аккаунт
      </Typography>

      <Grid container spacing={3} sx={{ my: 6, mx: 2 }}>
        {vacancies.map((vacancy) => (
          <Grid item xs={12} key={vacancy.id}>
            <Card>
              <CardContent>
                <Typography variant="h6" gutterBottom>
                  {vacancy.title}
                </Typography>
```

```
                <Stack direction="row" alignItems="center"
spacing={1} mb={1}>
                  <DescriptionIcon color="action" />
                  <Typography>
                    {vacancy.description || "No description
provided"}
                  </Typography>
                </Stack>

                <Stack direction="row" alignItems="center"
spacing={1} mb={1}>
                  <MonetizationOnIcon color="action" />
                  <Typography>{vacancy.unitPrice /
100}грн</Typography>
                </Stack>

                <Stack direction="row" alignItems="center"
spacing={1} mb={1}>
                  <RoomIcon color="action" />
                  <Typography>
                    Звідки: {vacancy?.from?.country},
{vacancy?.from?.city},{" "}
                    {vacancy?.from?.street}
                  </Typography>
                </Stack>

                <Stack direction="row" alignItems="center"
spacing={1} mb={1}>
                  <RoomIcon color="action" />
                  <Typography>
                    Куди: {vacancy?.to?.country},
{vacancy?.to?.city},{" "}
                    {vacancy?.to?.street}
                  </Typography>
                </Stack>

                <Stack direction="row" alignItems="center"
spacing={1} mb={1}>
                  <PersonIcon color="action" />
                  <Typography>Логіст:
{vacancy?.creator?.username}</Typography>
                </Stack>

                <Stack direction="row" alignItems="center"
spacing={1}>
                  <AccessTimeIcon color="action" />
                  <Typography>
                    Створено: {new
Date(vacancy.createdAt).toLocaleString()}
                  </Typography>
                </Stack>
              </CardContent>
              <CardActions>
                <Button
```

```
                  href={`/logist/profile/${vacancy?.creator?.id}`}
                  variant="outlined"
                  size="small"
                >
                  Профіль логіста
                </Button>

                {user && (
                  <Button
                    variant="contained"
                    size="small"
                    onClick={() => {
                      setSelectedVacancy(vacancy);
                      setOpen(true);
                    }}
                  >
                    Відгукнутись на вакансію
                  </Button>
                )}
              </CardActions>
            </Card>
          </Grid>
        ))}
      </Grid>

      {/* Modal */}
      <Dialog
        open={open}
        onClose={() => setOpen(false)}
        fullWidth
        maxWidth="sm"
      >
        <DialogTitle>Відгукнутись на вакансію</DialogTitle>
        <form onSubmit={handleSubmit(onSubmit)}>
          <DialogContent>
            <Controller
              name="comment"
              control={control}
              defaultValue=""
              render={({ field }) => (
                <TextField
                  {...field}
                  fullWidth
                  label="Коментар"
                  multiline
                  rows={4}
                  variant="outlined"
                />
              )}
            />
          </DialogContent>
          <DialogActions>
            <Button onClick={() =>
setOpen(false)}>Закрити</Button>
```

```
            <Button type="submit" variant="contained"
disabled={isSubmitting}>
              Відгукнутись
            </Button>
          </DialogActions>
        </form>
      </Dialog>
    </Box>
  );
};

export default VacanciesPage;
```

Додаток Б

Літсинг Б.1 – Код модуля user (файл сервісу з бізнес-логікою веб-застосунку).

```
import {
  BadRequestException,
  Inject,
  Injectable,
  NotFoundException,
} from '@nestjs/common';
import { hash } from 'bcrypt';
import { UserServiceInterface } from './interface/user-
service.interface';
import { CreateUserDTO } from './dto/create-user.dto';
import { UserInterface } from './interface/user.interface';
import { User } from './entities/user.entity';
import { USER_REPOSITORY } from './user.constants';
import { UserRepositoryInterface } from './interface/user-
repository.interface';

@Injectable()
export class UserService implements UserServiceInterface {
  private readonly hashSalt = 5;

  constructor(
    @Inject(USER_REPOSITORY)
    private readonly userRepository: UserRepositoryInterface,
  ) {}

  async create(dto: CreateUserDTO): Promise<UserInterface> {
    const { username, password, role } = dto;

    if (await this.userRepository.getByUsername(username)) {
      throw new BadRequestException('This username is already
taken');
    }

    const user = new User();

    user.username = username;
    user.password = await hash(password, this.hashSalt);
    user.role = role;

    return this.userRepository.create(user);
  }

  getAll(): Promise<UserInterface[]> {
    return this.userRepository.getAll();
  }
```

```
  async getById(id: string): Promise<UserInterface> {
    const user = await this.userRepository.getById(id);

    if (!user) throw new NotFoundException();
    return user;
  }
  getByUsername(username: string): Promise<UserInterface> {
    return this.userRepository.getByUsername(username);
  }
}
```

Літсинг Б.2 – Код модуля user (файл модуля з підключенням усіх залежностей)

```
import { Module, forwardRef } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { User } from './entities/user.entity';
import { AuthModule } from '../auth/auth.module';
import { USER_REPOSITORY, USER_SERVICE } from './user.constants';
import { UserRepository } from './user.repository';
import { UserService } from './user.service';
import { UserController } from './user.controller';

@Module({
  imports: [TypeOrmModule.forFeature([User]), forwardRef(() =>
AuthModule)],
  providers: [
    {
      provide: USER_REPOSITORY,
      useClass: UserRepository,
    },
    {
      provide: USER_SERVICE,
      useClass: UserService,
    },
  ],
  controllers: [UserController],
  exports: [USER_SERVICE],
})
export class UserModule {}
```

Лістинг Б.3 – Константи у модулі user

```
export const USER_REPOSITORY = 'USER_REPOSITORY';
export const USER_SERVICE = 'USER_SERVICE';
```

Лістинг Б.4 – Ролі у модулі user

```
export enum UserRole {
  DRIVER = 'DRIVER',
  LOGIST = 'LOGIST',
}
```

Лістинг Б.5 – DTO клас для створення нового користувача

```
import { IsEnum, IsString } from 'class-validator';

import { UserRole } from '../user.enums';

export class CreateUserDTO {
  @IsString()
  username: string;

  @IsString()
  password: string;

  @IsEnum(UserRole)
  role: UserRole;
}
```