Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: Розробка веборієнтованої інформаційної системи для автоматизованого пошуку, виклику та моніторингу послуг евакуації транспортних засобів

Виконав: студент 4 курсу
групи ЗПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Великий Максим Валентинович.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Мельник О. В.
(прізвище та ініціали)

Рецензент: к.т.н., доц. каф ОТ Богомолов С.В.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ О.Н. Романюк
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н._____
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФКАЦІЙНУ РОБОТУ СТУДЕНТУ

Великий Максим Валентинович

1. Тема роботи – Розробка веборієнтованої інформаційної системи для автоматизованого пошуку, виклику та моніторингу послуг евакуації транспортних засобів

Керівник роботи: Мельник Олександр Васильович, к. т. н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Термін подання студентом завершеної роботи: до 2 червня 2025 р.

3. Вихідні дані: архітектурні шаблони вебзастосунків, сучасні бібліотеки фронтенду (React), фреймворк Spring Boot, PostgreSQL, засоби REST API, Google Maps / Leaflet API.

4. Зміст пояснювальної записки:

- аналіз предметної області та існуючих аналогів;
- вибір технологій реалізації клієнтської та серверної частин;
- опис структури бази даних;
- розробка модулів системи;

– тестування;
– інструкція користувача.

5. Перелік графічного матеріалу:
– діаграма структури системи;
– схема роботи клієнта та сервера;
– схема бази даних;
– інтерфейс користувача (макети);
– діаграма алгоритму роботи.

6. Консультанти розділів:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Мельник О. В., к.т.н., доц. каф. ПЗ	23.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

Ч.ч.	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз цифрових сервісів виклику евакуатора та формулювання завдань проєкту	25.03.25 – 06.04.25	вик
2	Проєктування архітектури веб-застосунку та розробка інтерфейсу користувача	07.04.25 – 20.04.25	вик
3	Реалізація клієнтської та серверної частин системи з інтеграцією карти	20.04.25 – 07.05.25	вик
4	Тестування функціоналу та розробка адміністративного модуля	07.05.25 – 20.05.25	вик
5	Оформлення пояснювальної записки та підготовка матеріалів до захисту	20.05.25– 30.05.25	вик

Студент _____ М. В. Великий
(підпис) (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ О. В. Мельник
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

УДК 004.75:004.77

Великий М. В. Розробка веборієнтованої інформаційної системи для автоматизованого пошуку, виклику та моніторингу послуг евакуації транспортних засобів : бакалаврська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. – Вінниця: ВНТУ, 2025. – 79 с.

На укр. мові. Бібліогр.: 20 назв ; рис.: 25 ; табл.: 4 ; додатки: 4.

У роботі розроблено вебзастосунок для автоматизованого виклику евакуатора, який дозволяє користувачеві в реальному часі обирати найближчого водія, переглядати тариф, створювати замовлення та відстежувати його статус. Система реалізована на основі клієнт-серверної архітектури з використанням Java (Spring Boot), React і PostgreSQL. Застосунок включає як інтерфейс користувача, так і адміністративну панель для керування замовленнями та водіями.

Проведено аналіз аналогів, проєктування архітектури, реалізацію бази даних, інтерфейсів і API, тестування та впровадження основних функцій. Застосунок орієнтований на підвищення прозорості та оперативності послуг, може масштабуватись для використання в різних регіонах та адаптуватись до мобільних платформ.

Ключові слова: вебзастосунок, евакуатор, інформаційна система, Java, Spring Boot, React, PostgreSQL, автоматизація послуг.

ABSTRACT

UDC 004.75:004.77

Velykyi M. V. Development of a web-based information system for automated search, calling and monitoring of vehicle evacuation services: bachelor's thesis in specialty 121 - Software Engineering, educational program - Software Engineering - Vinnytsia: VNTU, 2025. 79 p.

In Ukrainian. Bibliogr.: 20 titles ; Figures: 25 ; tables: 4 ; appendix: 4.

The paper develops a web application for automated tow truck call, which allows the user to select the nearest driver in real time, view the tariff, create an order and track its status. The system is based on a client-server architecture using Java (Spring Boot), React, and PostgreSQL. The application includes both a user interface and an administrative panel for managing orders and drivers.

We analyzed analogues, designed the architecture, implemented the database, interfaces and APIs, tested and implemented the main functions. The application is focused on increasing the transparency and efficiency of services, can be scaled for use in different regions, and can be adapted to mobile platforms.

Keywords: web application, tow truck, information system, Java, Spring Boot, React, PostgreSQL, service automation.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Зростання кількості транспортних засобів в умовах урбанізації створює нові виклики у сфері надання оперативної дорожньої допомоги. Однією з найбільш поширених потреб є евакуація транспортних засобів у разі поломки, дорожньо-транспортних пригод, порушень правил паркування або блокування проїзду. Існуючі сервіси не завжди забезпечують повний функціонал: обмежений доступ до мап, непрозорість тарифів, відсутність онлайн-контролю за замовленням. Це обумовлює актуальність розробки ефективного цифрового рішення. Бакалаврська кваліфікаційна робота на тему «Розробка веборієнтованої інформаційної системи для автоматизованого пошуку, виклику та моніторингу послуг евакуації транспортних засобів» є відповіддю на зазначені виклики, адже пропонує комплексне рішення, що поєднує інтерактивну карту, гнучке управління замовленнями та прозорий інтерфейс для обох сторін – користувачів і адміністраторів.

Зв’язок роботи з науковими програмами, планами, темами. Роботу виконано в межах науково-дослідної тематики кафедри програмного забезпечення Вінницького національного технічного університету відповідно до навчального плану підготовки бакалаврів за спеціальністю 121 – Інженерія програмного забезпечення.

Мета та завдання дослідження. Метою роботи є створення зручного вебзастосунку, що дозволяє користувачу викликати евакуатор, переглядати доступні варіанти на інтерактивній мапі, а адміністраторам – контролювати замовлення у режимі реального часу. Для реалізації мети необхідно: проаналізувати предметну область та існуючі рішення; розробити архітектуру програмної системи; реалізувати клієнтську та серверну частини з використанням React та Spring Boot; створити базу даних PostgreSQL; забезпечити обмін даними через REST API;

реалізувати рольову авторизацію; інтегрувати картографічну систему Leaflet; провести тестування ключових функцій.

Об'єкт і предмет дослідження. Об'єктом дослідження є процес створення веборієнтованих інформаційних систем у сфері цифрових сервісів дорожньої допомоги. Предметом дослідження виступають методи, інструменти та архітектурні рішення, що застосовуються при розробці таких систем із використанням Java, React, PostgreSQL, REST API та Leaflet.

Методи дослідження. У роботі використовувалися методи об'єктно-орієнтованого проєктування, архітектурного моделювання, створення UI-компонентів, конструювання реляційних структур БД, а також методи ручного функціонального тестування. Технологічний стек: Spring Boot, JPA/Hibernate, React, Leaflet API, PlantUML.

Наукова новизна. Уперше запропоновано комплексне вебрішення для виклику евакуатора, яке забезпечує інтеграцію з мапою в реальному часі, динамічне визначення доступних водіїв з урахуванням місцеположення, обробку статусів замовлень у режимі реального часу, гнучке розмежування прав доступу, а також збереження історії замовлень. Система враховує особливості українського ринку та дозволяє масштабувати її для використання в інших регіонах.

Практичне значення. Розроблена система може бути впроваджена як у приватних службах, так і у муніципальних структурах. Її застосування дозволяє зменшити кількість дзвінків, прискорити реагування, забезпечити зворотний зв'язок та прозору взаємодію між клієнтом і водієм евакуатора.

Особистий внесок здобувача. Усі компоненти вебзастосунку – інтерфейс, логіка взаємодії клієнта із сервером, моделі бази даних, API – реалізовані особисто автором. У межах публікацій автор відповідав за реалізацію бізнес-логіки, побудову архітектури та інтеграцію картографічних сервісів.

Апробація результатів. Результати представлені на двох конференціях:

1. Великий М.В., Мельник О.В. «Інтелектуальний аналіз даних у CRM-системах для обслуговування викликів евакуатора», ОНТУ, 2024.
2. Великий М.В. «Розробка веборієнтованої інформаційної системи для виклику евакуатора», ZCIT, 2024.

Структура та обсяг БКР. Бакалаврська кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел (23 найменування), однієї таблиці, 26 рисунків та чотирьох додатків. Загальний обсяг роботи – 78 сторінок.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ У СФЕРІ ЕВАКУАЦІЇ ТРАНСПОРТНИХ ЗАСОБІВ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз існуючих сервісів евакуації транспортних засобів

У сучасному урбанізованому середовищі важливою складовою забезпечення зручності водіїв є надання якісних та оперативних послуг евакуації транспортних засобів. Традиційна модель надання таких послуг через телефонні дзвінки до диспетчерських служб поступово втрачає актуальність, поступаючись місцем цифровим сервісам. Водночас більшість із цих сервісів мають обмежену функціональність або не відповідають очікуванням користувачів.

На ринку України й інших країн з'явились сервіси, які частково реалізують функціонал автоматизованого виклику евакуатора. Серед них:

Uklon – популярний український сервіс, основною функцією якого є виклик таксі. Однак у деяких містах у додатку доступна можливість виклику евакуатора. Сервіс підтримує геолокацію та дає змогу оформити замовлення через мобільний застосунок. Проте користувач не може вибрати конкретного водія, а функція відстеження руху працює лише частково.

SOS Auto – спеціалізований сервіс для надання допомоги на дорозі, що підтримує виклик евакуатора. Користувач вводить адресу, обирає послугу та чекає на підтвердження. Вебсайт та мобільний застосунок зручні у використанні, однак відсутній функціонал повноцінного онлайн-відстеження евакуатора після оформлення замовлення.

Evacuator PRO – мобільний застосунок українського походження, що дозволяє швидко оформити заявку на евакуацію авто. Має просту структуру, підтримку декількох мов та базовий функціонал для замовлення послуги. Проте не забезпечує інтерактивного відстеження транспорту, автоматичного вибору найближчого водія або прозорого ціноутворення.

Easy Towing – зарубіжний сервіс, орієнтований на ринок США, що дозволяє користувачам обирати водія, бачити орієнтовну вартість та відстежувати рух евакуатора в реальному часі. Є одним із найбільш функціонально повних прикладів, однак сервіс недоступний в Україні.

- Towmates – платформа, орієнтована на компанії, що надають буксирувальні послуги. Застосунок дозволяє керувати завданнями, координувати водіїв, переглядати звіти та статуси. Призначений більше для внутрішнього використання компаніями, ніж для масового користувача.

Отже, аналіз існуючих сервісів показує, що:

- більшість сервісів мають обмеження щодо географічного покриття;
- відсутні або обмежені функції онлайн-відстеження транспорту;
- не реалізовано вибір конкретного водія;
- бракує прозорої та гнучкої системи тарифоутворення;
- деякі сервіси взагалі недоступні на території України.

Таким чином, незважаючи на часткову цифровізацію ринку евакуаційних послуг, більшість існуючих рішень не відповідають очікуванням користувачів. Це свідчить про актуальність розробки нової веборієнтованої інформаційної системи, яка поєднуватиме зручність інтерфейсу, автоматизовану логіку обробки замовлень та інтерактивний моніторинг виконання викликів.

1.2 Порівняння функціональних можливостей аналогічних систем

Розглянемо цифрові сервіси, які частково реалізують функціонал автоматизованого виклику евакуатора. Аналіз дозволить визначити ключові переваги, недоліки та напрями вдосконалення при проєктуванні власної системи.

Uklon – один із найбільш популярних сервісів таксі в Україні, який також надає послуги евакуаторів у великих містах. У застосунку користувач може обрати тип транспорту (включно з евакуатором), вказати місце подачі, побачити орієнтовну вартість та оформити замовлення (рис. 1.1). Застосунок підтримує

геолокацію, однак не надає користувачу можливості самостійно обрати конкретного водія [1].

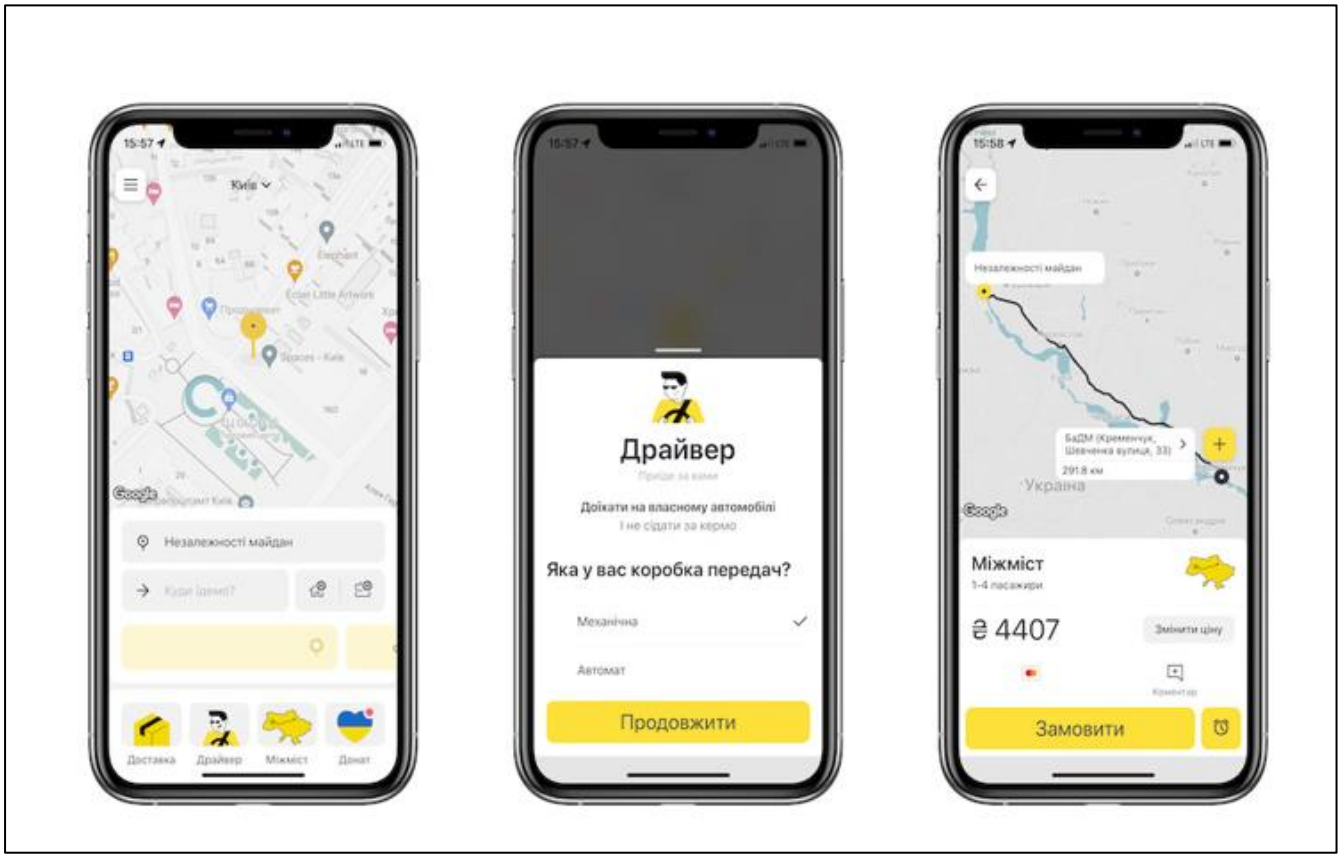

Рисунок 1.1 – Приклад роботи застосунку Uklon

SOS Auto – український сервіс допомоги на дорозі, що має мобільний застосунок і сайт. Окрім виклику евакуатора, сервіс надає допомогу з акумулятором, шиномонтажем тощо. Користувач вказує місцезнаходження, обирає послугу, і система надсилає запит найближчому водієві. Відстеження евакуатора у реальному часі відсутнє. Вартість послуги виводиться після підтвердження замовлення (рис.1.2).

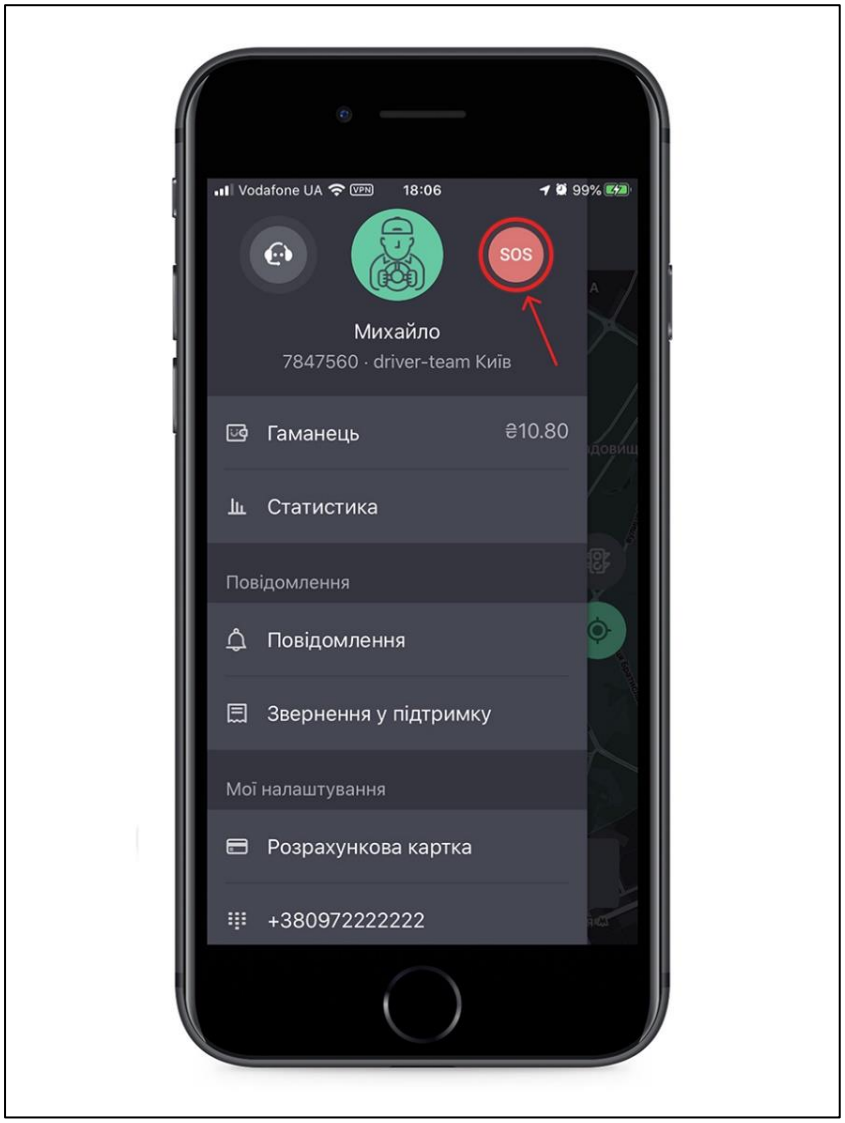

Рисунок 1.2 – Приклад роботи застосунку SOS Auto

Easy Towing – американський сервіс, що дозволяє користувачам викликати евакуатор через мобільний додаток [2]. Користувач вводить адресу, тип авто, отримує пропозиції від водіїв з цінами та часом прибуття. Є можливість відстеження евакуатора в реальному часі та прямого зв'язку з водієм. Сервіс недоступний в Україні (рис. 1.3).

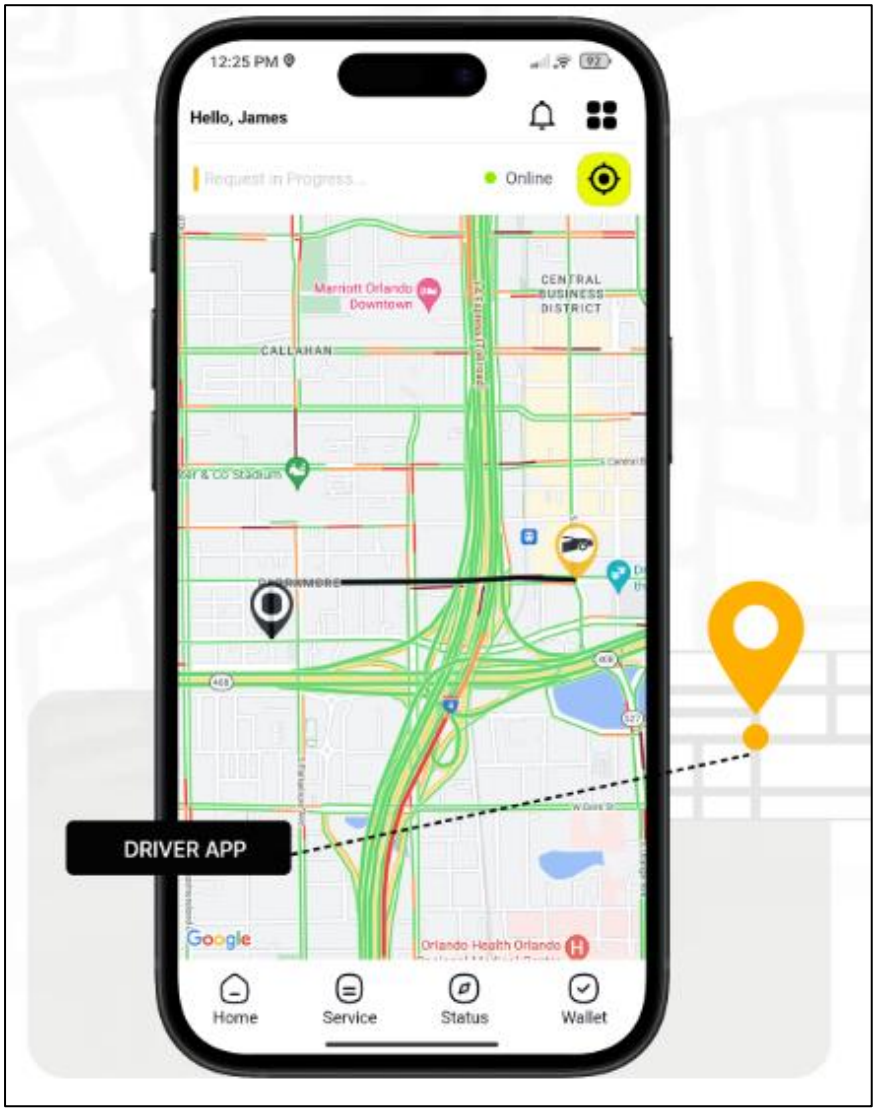

Рисунок 1.3 – Приклад роботи сайту Easy Towing

Evacuator PRO – інноваційний підхід до сервісу замовлення евакуатора. Користувач може швидко та просто отримати кваліфіковану допомогу для евакуації свого автомобіля [3]. Застосунок підтримує кілька мов, включаючи українську, і має компактний розмір, що дозволяє завантажити його навіть при повільному інтернет-з'єднанні (рис. 1.4).

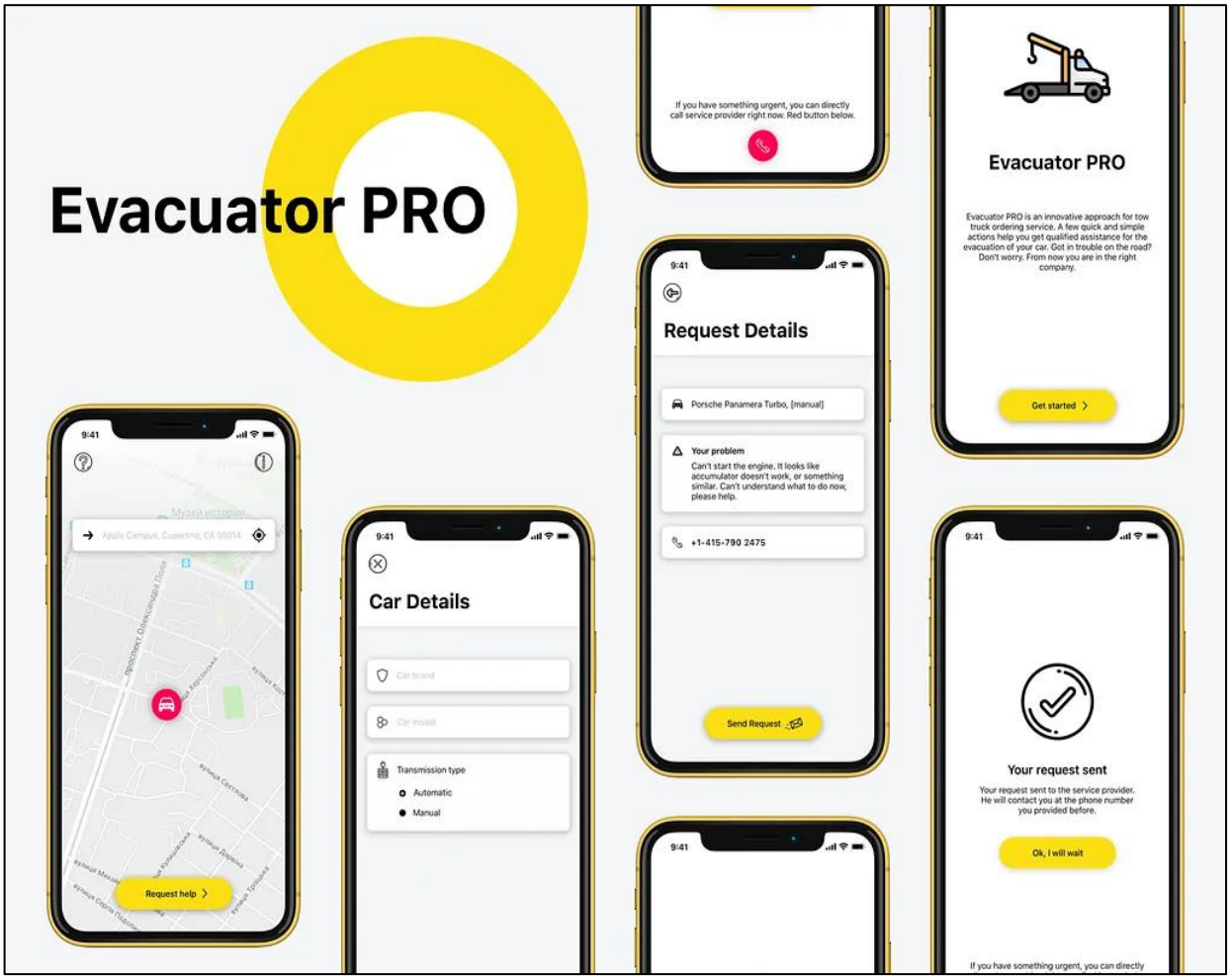

Рисунок 1.4 – Приклад роботи застосунку Evacuator PRO

Towmates – інноваційний застосунок для буксирувальних компаній та водіїв евакуаторів (рис. 1.5). Дозволяє компаніям створювати облікові записи, призначати завдання та відстежувати свій автопарк у режимі реального часу. Водії можуть приймати або відхиляти призначені завдання, використовувати інтегровану карту для навігації та документувати свою роботу шляхом завантаження фотографій [4].

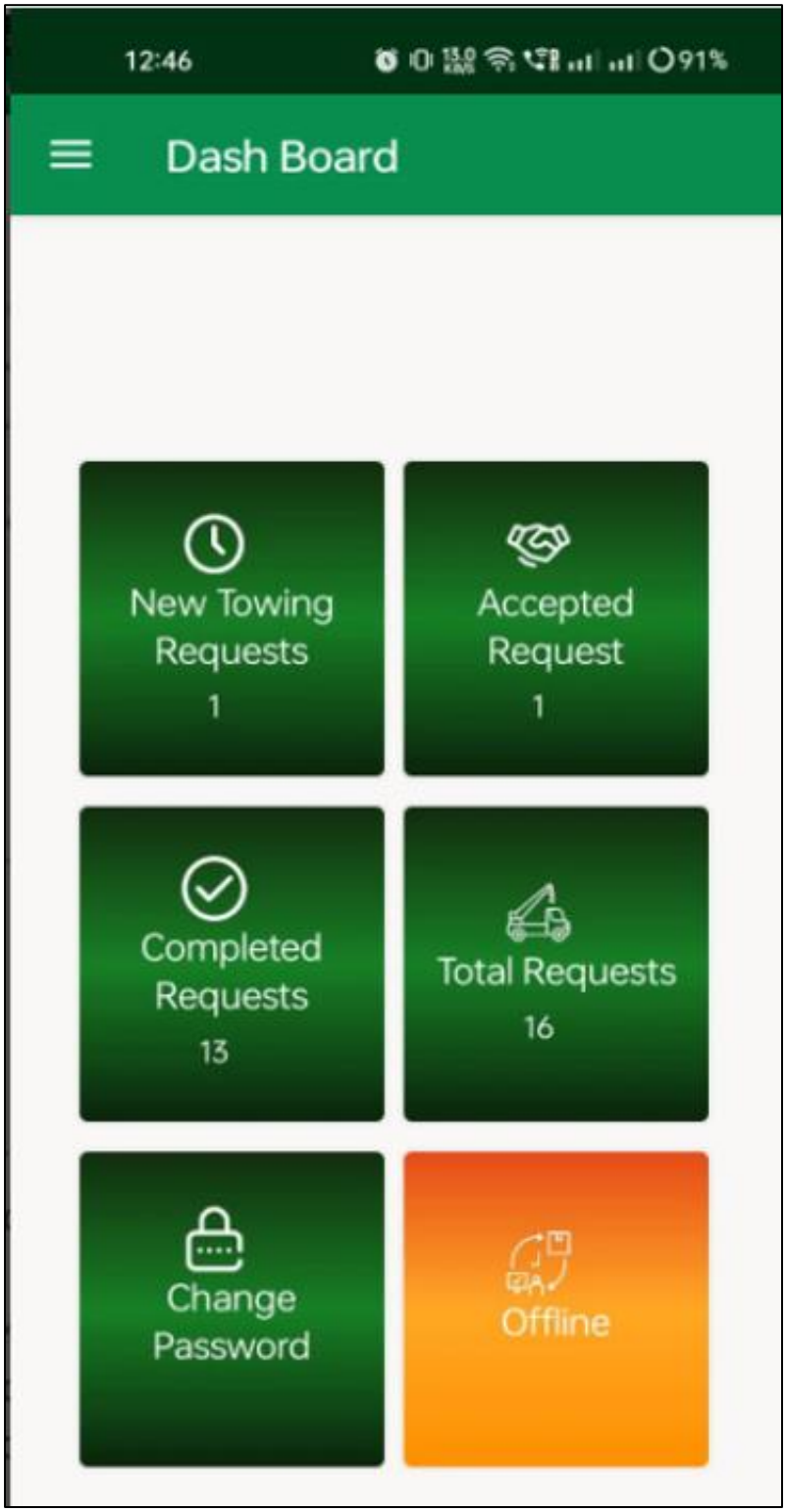

Рисунок 1.5 – Приклад використання Towmates

Результати порівняння аналогів зведено в табл. 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерії	Uklon	SOS Auto	Evacuator PRO	Easy Towing	Towmates	Власна система
Геолокація та точка виклику	+	+	+	+	+	+
Онлайн-відстеження евакуатора	+	-	-	+	+	+
Вибір найближчого водія	-	-	-	+	+	+
Прозоре тарифоутворення	+	+	-	+	+	+
Мобільний застосунок	+	+	+	+	+	+
Вебверсія системи	+	+	-	-	-	+
Підтримка українського ринку	+	+	+	-	-	+
Загальна оцінка	75%	70%	60%	80%	85%	100%

Як видно з таблиці 1.1, жоден із розглянутих сервісів не забезпечує повного набору функціональних можливостей, які необхідні для якісної організації процесу

виклику евакуатора. Частина систем не підтримує вебверсії, інші не мають онлайн-відстеження або функції вибору найближчого виконавця. В окремих випадках відсутня прозора система тарифоутворення, що знижує довіру користувачів.

Крім того, деякі сервіси не мають підтримки українського ринку або орієнтовані переважно на внутрішні потреби компаній. У зв'язку з цим доцільною є розробка власної веборієнтованої інформаційної системи для автоматизованого виклику евакуатора, яка об'єднає в собі ключові переваги існуючих рішень та усуне їхні недоліки.

1.3 Обґрунтування створення вебзастосунку для евакуації авто

Ринок цифрових сервісів для виклику евакуаторів демонструє стрімкий розвиток, проте більшість існуючих рішень є обмеженими за функціональністю, географією охоплення або зручністю користування. Як показано в попередньому підрозділі, жоден з наявних сервісів не забезпечує повного циклу взаємодії користувача з евакуаційною службою – від пошуку найближчого водія до онлайн-відстеження та прозорого формування тарифу.

Сучасний користувач очікує не лише функціональності, а й інтерактивності, гнучкості, адаптивності інтерфейсу, можливості вибору та миттєвого зворотного зв'язку. Водночас компанії-постачальники послуг зацікавлені в централізованому управлінні замовленнями, контролі водіїв, зниженні впливу людського чинника, зборі аналітики та скороченні витрат на обробку запитів.

Існуючі аналоги часто обмежуються мобільними платформами, що унеможливлює доступ із настільних пристроїв, або орієнтовані на внутрішні потреби компаній без широкого охоплення кінцевого користувача. Бракує гнучкої тарифної системи, модулів GPS-моніторингу, засобів інтеграції з іншими службами або муніципальними базами.

Зважаючи на ці недоліки, розробка нової веборієнтованої інформаційної системи набуває особливої актуальності. Пропоноване рішення дозволить:

- реалізувати повноцінну вебплатформу, доступну з будь-якого пристрою, незалежно від ОС;
- здійснювати пошук найближчого евакуатора на основі геолокації користувача;
- визначати тариф у реальному часі з урахуванням відстані, типу послуги та попередніх замовлень;
- забезпечити візуалізацію транспорту на карті, включно з онлайн-відстеженням руху;
- надати адміністративну панель для компаній з керуванням автопарком і замовленнями;
- масштабувати систему на декілька міст, із можливістю подальшої інтеграції з мобільним застосунком.

Крім того, система може використовуватися страховими компаніями, муніципальними службами та логістичними операторами, які потребують автоматизованих інструментів виклику, аналітики та моніторингу. Її впровадження сприятиме підвищенню якості послуг, зменшенню навантаження на диспетчерські служби та підвищенню прозорості роботи ринку евакуаційних послуг.

Таким чином, створення нового вебзастосунку є не лише доцільним, а й необхідним кроком у напрямку цифрової трансформації сервісів дорожньої допомоги в Україні.

1.4 Постановка задачі бакалаврської кваліфікаційної роботи

Метою бакалаврської кваліфікаційної роботи є розробка вебзастосунку для пошуку, виклику та моніторингу евакуаторів, що забезпечує ефективну взаємодію між користувачами та операторами евакуаційних служб у режимі реального часу. Система має бути доступною з будь-якого пристрою, мати зручний і адаптивний інтерфейс, підтримувати онлайн-відстеження транспорту, прозоре формування тарифів та адміністрування з боку сервісу.

На основі сформульованої мети визначено такі основні задачі дослідження:

- проаналізувати існуючі сервіси евакуації транспортних засобів та виявити їх переваги і недоліки;
- обґрунтувати необхідність створення нового вебзастосунку з розширеною функціональністю;
- сформулювати вимоги до розроблюваної системи (функціональні та нефункціональні);
- спроєктувати архітектуру застосунку із чітким розділенням на клієнтську, серверну та базову частини;
- реалізувати основні функціональні модулі:
    - авторизації/реєстрації;
    - визначення геолокації користувача;
    - пошуку найближчих евакуаторів;
    - створення замовлення;
    - інтерактивного моніторингу на мапі;
    - адміністративного керування викликами;
- розробити інтерфейс користувача з використанням React;
- побудувати систему зберігання даних на базі PostgreSQL;
- виконати тестування реалізованих функцій;
- підготувати супровідну технічну документацію, включно з інструкцією користувача.

Повний текст технічного завдання наведено у Додатку А.

Результатом виконання роботи має стати функціональний вебзастосунок, придатний до впровадження в сервісну діяльність приватних або муніципальних структур, які надають послуги евакуації транспортних засобів.

1.5 Висновки

У першому розділі бакалаврської кваліфікаційної роботи проведено комплексний аналіз сучасних інформаційних систем для виклику евакуаторів, включаючи популярні сервіси Uklon, SOS Auto, Evacuator PRO, Easy Towing та Towmates. На основі порівняльного аналізу (табл. 1.1) встановлено, що жоден із представлених сервісів не надає повного комплексу функціональних можливостей, необхідних для зручного та ефективного користування послугами евакуації транспортних засобів.

Обґрунтовано доцільність розробки нового вебзастосунку, який усуває недоліки існуючих рішень, підтримує вебдоступ, забезпечує онлайн-відстеження евакуатора, дозволяє обирати найближчого водія та містить адміністративну панель для керування замовленнями. Така система є актуальною для впровадження як у приватному, так і в комунальному секторі послуг.

У цьому розділі також визначено мету бакалаврської кваліфікаційної роботи та сформульовано основні задачі, що охоплюють етапи аналізу, проєктування, розробки, тестування та документування вебзастосунку. Детальні вимоги до функціоналу системи наведено у технічному завданні (див. Додаток А).

2 ПРОЄКТУВАННЯ ІНТЕРФЕЙСУ ТА СТРУКТУРИ ВЕБ-ОРІЄНТОВАНОЇ СИСТЕМИ ЕВАКУАЦІЇ

2.1 Визначення основних даних і процесів у системі

Вебзастосунок для пошуку, виклику та моніторингу евакуатора функціонує за принципами клієнт-серверної архітектури, де всі дії користувача з графічним інтерфейсом на фронтенді пов'язані з обробкою та передачею даних на сервер. Для ефективної реалізації функціональності системи необхідно чітко окреслити набір вхідних та вихідних даних, які використовуються в усіх ключових процесах: реєстрація, виклик евакуатора, відображення на карті, адміністрування.

На початковому етапі взаємодії з вебзастосунком користувач або водій проходить процес реєстрації чи авторизації. Ці дії формують перший блок вхідних даних – логін, пароль, ім'я, контактний телефон. Після успішної автентифікації система призначає сеанс, і користувач отримує доступ до основних функцій.

Наступним кроком є визначення місця розташування користувача. Застосунок використовує вбудовані засоби геолокації браузера або дає можливість ввести адресу вручну. Ці дані також надсилаються на сервер, де система виконує пошук доступних водіїв евакуаторів поблизу. На основі координат та статусу кожного водія формується вихідний список доступних опцій для замовлення.

Коли користувач підтверджує виклик евакуатора, у систему надходить новий запит з параметрами: ID користувача, координати виклику, обраний водій або автоматичний вибір найближчого, тариф, додаткові побажання. У відповідь система змінює статус замовлення на «очікує підтвердження» або «виконується» – ці дані вже є вихідними та відображаються в інтерфейсі як для замовника, так і для виконавця.

Ще одним джерелом вхідних даних є взаємодія водія з системою: підтвердження чи відхилення замовлення, оновлення статусу прибуття, прибуття на місце, завершення виклику. Кожна така дія фіксується та передається назад

клієнтській частині. Для забезпечення прозорості, у клієнтському інтерфейсі користувач бачить зміну статусу, а також може спостерігати рух виконавця на мапі, синхронізованій через API (наприклад, Google Maps або Leaflet).

Окремо варто відзначити взаємодію із формами зворотного зв'язку та оцінювання. Після завершення замовлення користувач має можливість залишити відгук або оцінку якості послуг. Ці текстові дані також передаються на сервер і зберігаються в базі для подальшого аналізу або модерації. Вони є вхідними для модуля відгуків і можуть впливати на рейтинг водія, що є вихідною інформацією як для адміністратора, так і для інших користувачів.

У системі також передбачено блок адміністративного керування. Вхідними даними для адміністратора є команди додавання, редагування або видалення облікових записів водіїв, зміна тарифів, перегляд історії замовлень, модерація відгуків. Усі зміни, які виконує адміністратор, передаються на сервер і оновлюють відповідні частини бази даних. Вихідною інформацією для адміністратора є статистика, поточні активні виклики, журнали подій.

Уся обробка даних відбувається через REST API, що дозволяє забезпечити чітку логіку маршрутизації, модульність та масштабованість. Обмін інформацією відбувається у форматі JSON, що є стандартом для сучасних вебсервісів. Усі запити проходять етап валідації, автентифікації та авторизації залежно від ролі користувача.

Ключові сценарії, які використовуються в системі, були описані у вигляді діаграми діяльності. На ній відображено послідовність дій користувача при виклику евакуатора – від моменту входу до системи і до завершення замовлення. Ця діаграма дає змогу візуально простежити основні блоки обробки вхідних та вихідних даних, які є фундаментом архітектури вебзастосунку (рис/ 2.1).

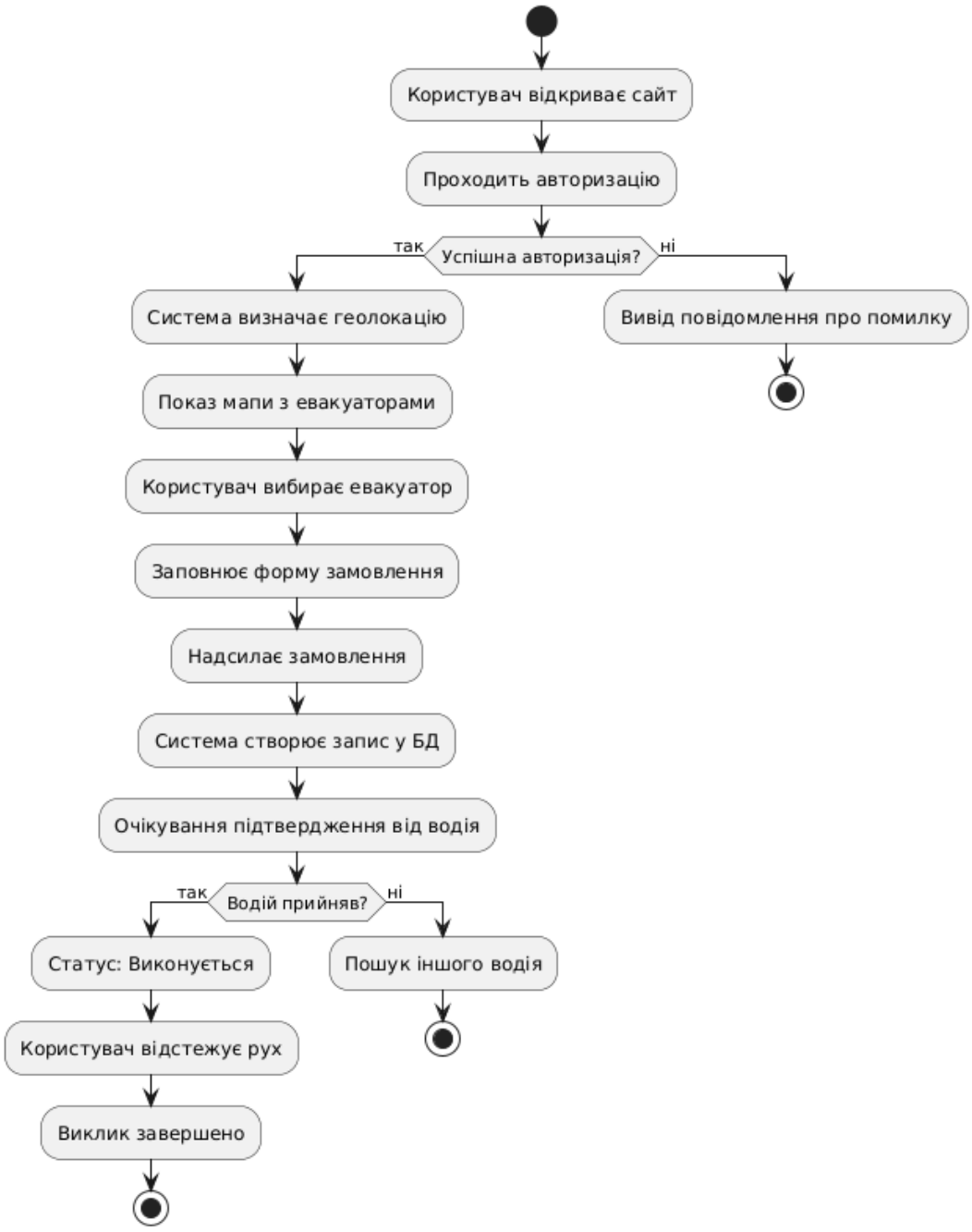

Рисунок 2.1 – Діаграма діяльності користувача вебзастосунку

З технічного боку, система передбачає чітке розмежування даних між клієнтом і сервером. Серверна частина (на базі Spring Boot) відповідає за бізнес-логіку, зберігання даних у PostgreSQL, а також за захист запитів. Клієнтська частина (React) формує динамічні інтерфейси та відображає результат взаємодії в реальному часі.

Таким чином, правильно структурований аналіз вхідних і вихідних даних дозволяє спроєктувати ефективну інформаційну модель системи, закласти основу для побудови модульної архітектури, та забезпечити коректну взаємодію користувачів із застосунком на всіх етапах роботи [5].

2.2 Розробка інтерфейсу користувача вебзастосунку

Інтерфейс користувача є ключовим компонентом інформаційної системи, адже саме він забезпечує доступ до основної функціональності застосунку та визначає якість взаємодії з користувачем. У межах бакалаврської кваліфікаційної роботи розроблено веб-інтерфейс для сервісу виклику евакуатора з урахуванням актуальних принципів UI/UX-дизайну, а також адаптивності до мобільних пристроїв.

Для побудови інтерфейсу використано бібліотеку React, що дозволяє створювати багаторазові компоненти та динамічні сторінки без перезавантаження. Застосунок реалізовано у вигляді односторінкового вебзастосунку (SPA), який містить набір окремих візуальних блоків для різних функцій системи.

До основних екранів вебзастосунку належать: головна сторінка, що містить логотип, короткий опис сервісу та кнопки входу або реєстрації. Дизайн витриманий у синьо-білій кольоровій гамі, з великими елементами керування, оптимізованими під мобільні пристрої.

Також реалізовано зручну навігацію між розділами: форма входу, екран із картою, перелік евакуаторів, меню користувача та сторінки профілю. Інтерфейс забезпечує логічну послідовність дій, швидку реакцію на введення даних, візуальний зворотний зв’язок, а також відповідає принципам доступності для широкого кола користувачів, включно з тими, хто користується мобільним інтернетом або пристроями з невеликим екраном (рис. 2.2).

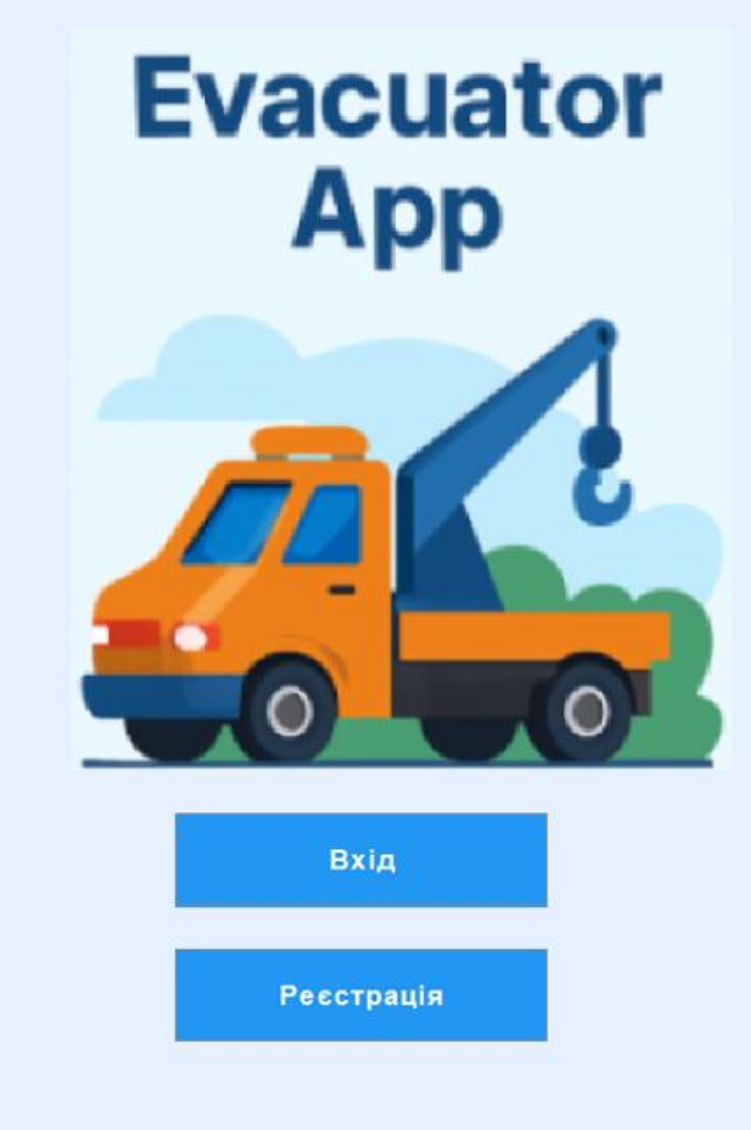

Рисунок 2.2 – Головна сторінка застосунку

Форма входу/реєстрації реалізована з валідацією обов’язкових полів, повідомленнями про помилки, а також розділенням між формою входу та реєстрації. Реєстраційна форма містить поля для ПІБ, пошти, логіна та пароля. Приклад продемонстровано на рис. 2.3.

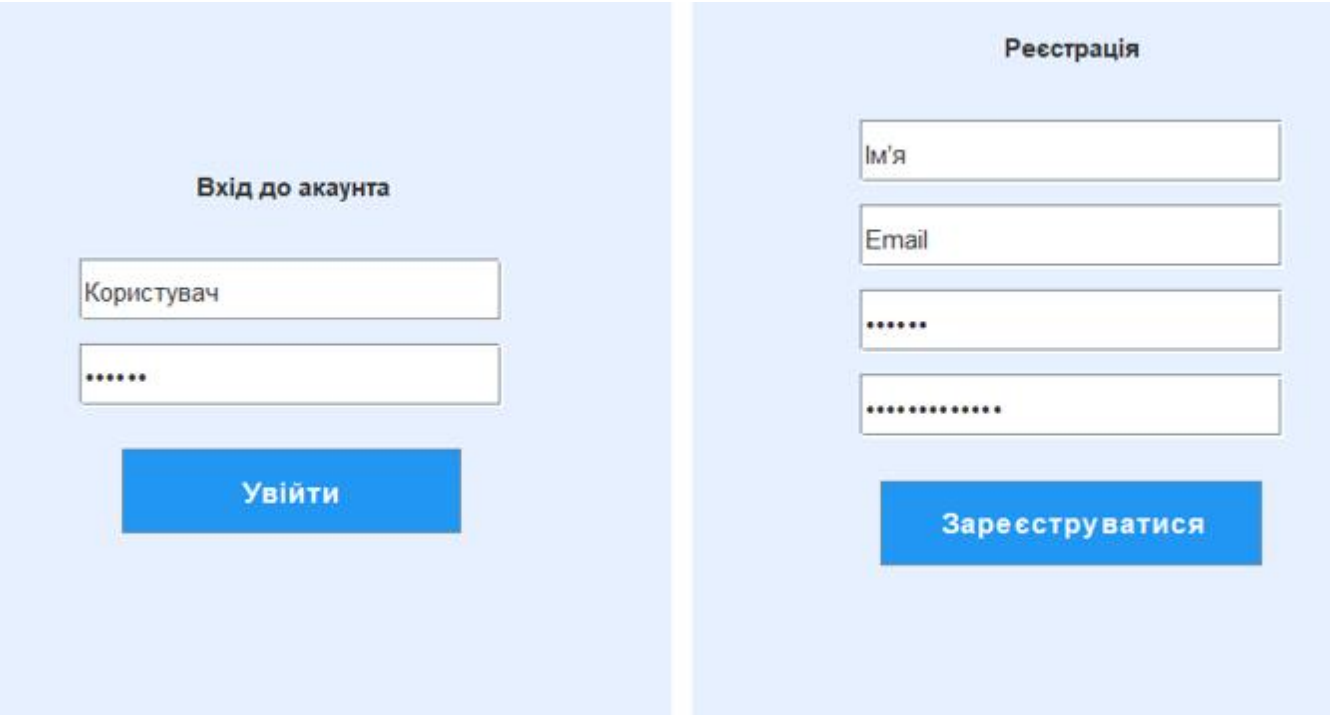

Рисунок 2.3 – Сторінка авторизації користувача

Карта з евакуаторами – це центральний екран, який відображає всі активні евакуатори поблизу у вигляді маркерів на мапі. Для реалізації інтерактивної карти використано бібліотеку Leaflet із можливістю зумування, навігації та оновлення розташування в реальному часі [6].

Маркер кожного евакуатора динамічно оновлюється на основі координат, що надходять із серверної частини через REST API. При натисканні на маркер користувач отримує коротку інформацію про водія: ім’я, рейтинг, орієнтовну ціну та кнопку для виклику. Зображення карти адаптується до розміру вікна та підтримує зміну масштабу залежно від щільності доступних евакуаторів. Завдяки використанню Leaflet також реалізовано можливість автоматичного центрування карти на поточному місцезнаходженні користувача (рис. 2.4).

Рисунок 2.4 – Відображення евакуаторів на карті

Форма створення замовлення відкривається після вибору евакуатора і дозволяє вказати адресу подачі, додаткові побажання, контактний номер. Після оформлення з’являється сторінка із підтвердженням замовлення та статусом, яка зображена на рис. 2.5.

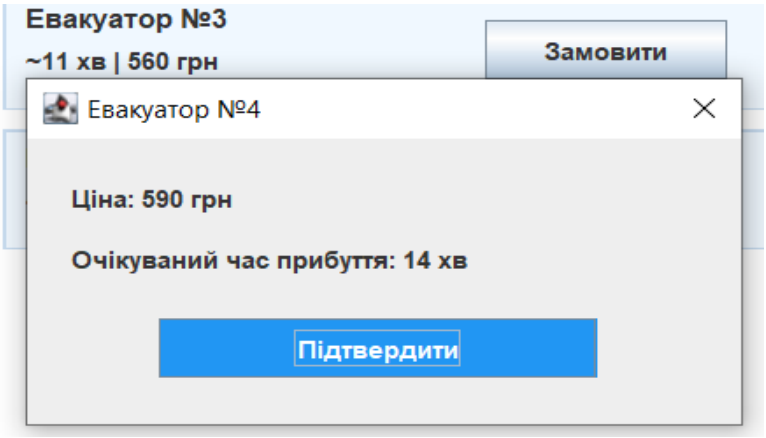

Рисунок 2.5 – Форма створення замовлення

Адміністративна панель – окремий інтерфейс, доступний лише адміністраторам. Вона містить список активних замовлень, водіїв, журнал подій та

можливість змінювати статуси або редагувати облікові записи. Панель реалізована у вигляді вкладок із таблицями та кнопками керування (рис. 2.6).

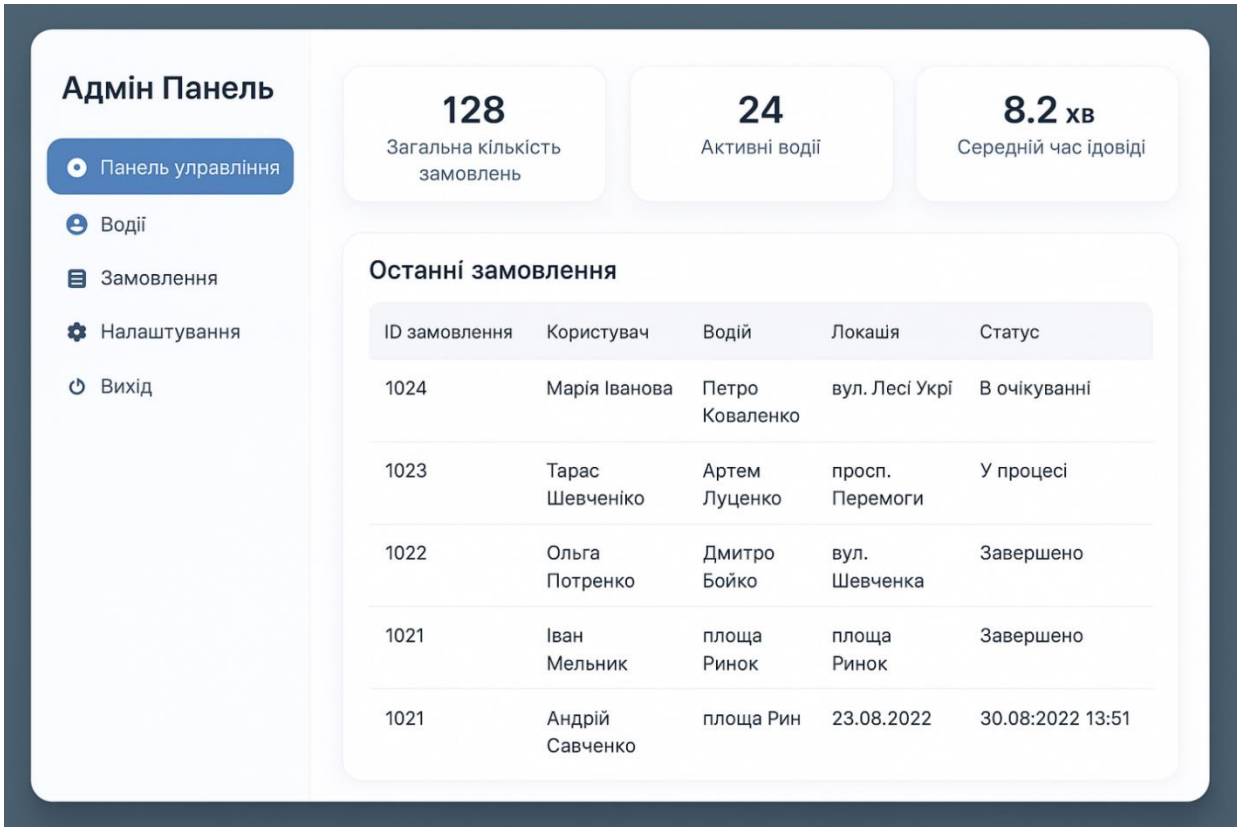

Рисунок 2.6 – Інтерфейс адміністратора

Усі компоненти розроблені з урахуванням адаптивності – дизайн коректно відображається на екранах смартфонів, планшетів і ПК. Особливу увагу приділено зручності: великі кнопки, плавні переходи між сторінками, автоматичне оновлення статусів. Для взаємодії з картою, формами та меню використано стандартні компоненти React, а також кастомізовані стилі на основі CSS-модулів.

Інтерфейс побудовано відповідно до принципів зручності та швидкого доступу до функцій. Усі навігаційні дії виконуються без перезавантаження сторінки, а елементи інтерфейсу максимально оптимізовані під користувача, який потребує термінової допомоги.

Таким чином, розроблений вебінтерфейс є важливою частиною системи, що забезпечує не лише функціональність, але й комфорт користувача при взаємодії із сервісом [7].

2.3 Розробка архітектури програмної системи

Архітектура вебзастосунку відіграє ключову роль у забезпеченні його стабільності, масштабованості, безпеки та ефективної взаємодії між користувачами, водіями та адміністраторами системи. У межах бакалаврської кваліфікаційної роботи реалізовано багаторівневу клієнт-серверну архітектуру, що дозволяє чітко розділити відповідальності між компонентами, спростити підтримку та забезпечити можливість подальшого розширення функціоналу.

Система побудована на основі таких технологій:

- Frontend (клієнтська частина): JavaScript, бібліотека React;
- Backend (серверна частина): Java, фреймворк Spring Boot;
- База даних: PostgreSQL;
- API: RESTful інтерфейси для взаємодії між frontend і backend;
- Інтеграція мапи: Leaflet із відкритими картографічними сервісами.

Програмна система логічно поділена на три основні рівні:

Клієнтський рівень (presentation layer): Вебінтерфейс користувача, реалізований за допомогою React, забезпечує взаємодію з усіма функціями застосунку – від реєстрації до виклику евакуатора та перегляду карти. React дозволяє реалізовувати односторінкову архітектуру з динамічно оновлюваними компонентами, що знижує навантаження на сервер і покращує зручність користування.

Серверний рівень (application layer): Серверна логіка реалізована за допомогою Spring Boot. Тут відбувається обробка всіх вхідних запитів від клієнта, валідація, авторизація, маршрутизація, управління сесіями, логіка формування

замовлень, обробка статусів викликів, передача координат тощо. Результати обробки запитів передаються у форматі JSON через REST API.

Рівень зберігання (data layer): Усі дані системи (користувачі, водії, замовлення, історія, рейтинги) зберігаються у реляційній базі даних PostgreSQL. Взаємодія між сервером і БД відбувається за допомогою ORM-технологій (Hibernate, JPA). Структура бази побудована у вигляді пов'язаних між собою таблиць із первинними і зовнішніми ключами [8].

Загальна схема взаємодії між компонентами системи подана на рис. 2.7.

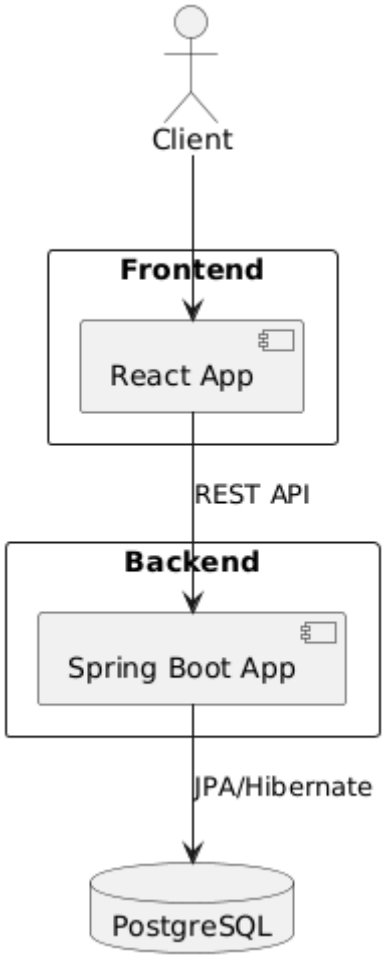

Рисунок 2.7 – Діаграма архітектури програмної системи вебзастосунку

На діаграмі видно, як клієнтська частина через HTTP-запити взаємодіє з backend-сервером. Кожен запит обробляється відповідним контролером на стороні

Spring Boot, який викликає відповідну сервісну логіку та звертається до бази даних у разі потреби. Відповідь формується у структурованому форматі та надсилається назад клієнту.

Крім базової трирівневої структури, система також враховує безпекові аспекти:

- доступ до API обмежено через JWT-токени (аутентифікація);
- розмежовано доступ до функцій за ролями (користувач, водій, адміністратор);
- дані користувача захищено при зберіганні (хешування паролів, обмеження доступу до полів).

Перевагами обраної архітектури є:

- чітке розділення відповідальностей між компонентами;
- можливість масштабування кожного рівня окремо;
- зручність у тестуванні та налагодженні;
- гнучкість при оновленні окремих частин системи (наприклад, заміна карти або бази даних без впливу на логіку клієнта).

Таким чином, розроблена архітектура забезпечує стабільну та ефективну роботу системи для автоматизованого виклику евакуатора, дозволяє забезпечити розширюваність і простоту подальшого розвитку програмного продукту [9].

2.4 Розробка алгоритмів функціонування сервісу евакуації

Для забезпечення коректної роботи вебзастосунку, всі основні процеси системи реалізовано у вигляді взаємопов’язаних алгоритмів. Розробка алгоритмів дозволяє формалізувати поведінку користувача, логіку обробки запитів, порядок оновлення статусів замовлень та інші ключові дії.

Алгоритми описують сценарії взаємодії користувача з вебзастосунком, починаючи від реєстрації й завершуючи отриманням послуги евакуації. Основна ідея – представити роботу кожного модуля через послідовність логічних кроків, де

кожен етап перевіряє певні умови, взаємодіє з сервером або базою даних та формує відповідь на клієнтську частину.

На рис. 2.8 представлено загальний алгоритм роботи вебзастосунку для користувача, який викликає евакуатор.

Рисунок 2.8 – Блок-схема алгоритму функціонування системи

Алгоритм функціонування системи наступний:

1. Користувач заходить на сайт і проходить авторизацію. Якщо облікового запису немає – реєструється.
2. Після входу система визначає місцеположення користувача автоматично або дозволяє ввести адресу вручну.
3. Здійснюється запит до сервера для отримання списку вільних евакуаторів поблизу.
4. Користувач переглядає мапу, обирає одного з доступних виконавців або дозволяє системі автоматично підібрати найближчого.
5. Відкривається форма створення замовлення. Після підтвердження запит надсилається на сервер.
6. Замовлення зберігається в базі даних зі статусом «Очікує підтвердження».
7. Водій отримує запит і приймає або відхиляє його.
8. У разі прийняття – статус змінюється на «Виконується», а користувач бачить рух авто на карті.
9. Після завершення замовлення статус оновлюється на «Завершено», і користувач може залишити відгук.

Окремо розроблено алгоритм автоматичного руху евакуатора в системі, який оновлює координати транспортного засобу в режимі реального часу та контролює стан на основі зовнішніх подій. Цей алгоритм подано на рис. 2.9.

Додатково система забезпечує збереження історії замовлень для кожного користувача, що дає змогу переглядати попередні виклики, їх статуси та залишені відгуки. Це підвищує прозорість надання послуг та сприяє формуванню довіри до сервісу. У разі виникнення спірних ситуацій, адміністратор має змогу отримати повну інформацію про замовлення через внутрішню панель управління.

Рисунок 2.9 – Блок-схема алгоритму автоматичного руху евакуатора

Алгоритм автоматичного руху евакуатора наступний:

- Після отримання замовлення водій рухається до користувача згідно з прокладеним маршрутом.
- Якщо система виявляє, що водій наближається до місця виклику – користувачу надсилається повідомлення.
- У разі зупинки (світлофор, перешкоди) система призупиняє відображення руху або змінює статус на «Очікування».

- При успішному прибутті статус змінюється на «На місці», а після завершення – на «Завершено».
- Координати водія оновлюються з певною частотою через WebSocket або REST-запити.

Для реалізації алгоритмів застосовуються як клієнтські механізми (React hooks, обробка подій), так і серверна логіка (обробка статусів, розрахунок відстаней, оновлення бази даних).

Система побудована з урахуванням можливості масштабування алгоритмів, наприклад:

- додавання стану «Очікує оплати»,
- автоматичного повторного запиту водієві в разі відмови,
- push-повідомлень про зміну статусів.

Таким чином, розроблені алгоритми забезпечують стабільну та передбачувану поведінку застосунку, знижують ризик помилок, покращують UX, і можуть бути легко доповнені новими сценаріями в майбутньому [10].

2.5 Висновки

У другому розділі бакалаврської кваліфікаційної роботи проаналізовано структуру та архітектуру вебзастосунку для виклику евакуатора. Розглянуто ключові типи даних, які забезпечують взаємодію між клієнтом, водієм і адміністративною частиною. Розроблено адаптивний інтерфейс користувача на основі React з фокусом на зручність і швидку навігацію. Архітектура поділена на клієнтську, серверну та базову частини, реалізовано REST API. Також описано алгоритми функціонування системи та побудовано блок-схеми для візуалізації логіки обробки замовлень і зміни статусів.

3 РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ ДЛЯ ВИКЛИКУ ЕВАКУАТОРА

3.1 Обґрунтування вибору засобів реалізації програмного забезпечення

На етапі розробки інформаційної системи для автоматизованого виклику евакуатора особливу увагу приділено вибору мови програмування та відповідного середовища розробки. Від коректності цього вибору залежить не лише зручність написання коду, але й ефективність подальшого тестування, розширюваність функціоналу, підтримка безпеки та стабільність роботи готового вебзастосунку. Враховувалися також вимоги до масштабованості проєкту, актуальність технології, підтримка сучасних фреймворків та активність спільноти розробників. Особливо важливо обрати такі інструменти, які не лише відповідають технічним потребам, але й забезпечують перспективи подальшого розвитку системи. З огляду на це, був здійснений всебічний аналіз технічних характеристик та практичної доцільності кожної технології. Це дало змогу сформувати обґрунтовану технічну базу для подальшої реалізації програмної частини.

У процесі дослідження проаналізовано чотири популярні мови програмування, які використовуються для розробки сучасних вебзастосунків: Java, Python, JavaScript та PHP. Кожна з них має свої переваги й недоліки, а також специфіку у сфері застосування. Порівняння виконувалося за низкою ключових критеріїв: стабільність, підтримка багатопоточності, наявність розвиненого фреймворку для веброзробки, кросплатформність, підтримка безпеки, а також наявність великої спільноти й документації. Оцінювання здійснювалося за допомогою бінарної системи: «1» означає відповідність мови певному критерію, а «0» – її відсутність. Це дозволило провести об’єктивне порівняння без суб’єктивного впливу.

Результати порівняння мов програмування наведено в таблиці 3.1.

Таблиця 3.1 – Порівняння мов програмування за ключовими характеристиками

Характеристика	**Java**	**Python**	**JavaScript**	**PHP**
Стабільність	1	0	0	0
Кросплатформність	1	1	1	1
Підтримка багатопоточності	1	0	0	0
Наявність фреймворків для веброзробки	1	1	1	1
Безпека при роботі з даними	1	0	0	0
Масштабованість	1	0	0	0
Документованість і підтримка	1	1	1	1
Підтримка REST API	1	1	1	1
Ідеальність для серверної логіки	1	0	0	0
Ідеальність для клієнтської логіки	0	0	1	0
Загальний коефіцієнт	9	4	5	4

Як видно з таблиці 3.1, за сумарною кількістю позитивних характеристик безумовне лідерство належить мові Java. Вона демонструє високу стабільність, підтримує багатопоточну обробку запитів, має потужну спільноту та забезпечує належний рівень безпеки при роботі з даними. Саме тому для реалізації серверної частини вебзастосунку обрано Java у поєднанні з фреймворком Spring Boot, що дозволило впровадити логіку REST API, безпечну авторизацію користувачів, взаємодію з базою даних PostgreSQL та масштабовану архітектуру. Для створення клієнтської частини – інтерфейсу користувача, форми виклику евакуатора, відображення карти – використано JavaScript у поєднанні з React, оскільки ця

комбінація забезпечує швидке оновлення контенту, інтерактивність та легку інтеграцію з картою й серверною логікою [11]. Після вибору мов програмування наступним кроком стало визначення середовищ розробки, які використовувалися для реалізації обох частин системи. Серед найпоширеніших варіантів розглядалися IntelliJ IDEA, Visual Studio Code, Eclipse та PyCharm. Критерії оцінювання включали зручність інтерфейсу, підтримку налагодження, інтеграцію з фреймворками, продуктивність, сумісність із системами контролю версій, підсвічування коду, кросплатформність і загальну стабільність роботи (табл. 3.2).

Таблиця 3.2 – Порівняння середовищ розробки за характеристиками

Характеристика	**VS Code**	**IntelliJ IDEA**	**Eclipse**	**PyCharm**
Підтримка Java	1	1	1	0
Підтримка JavaScript	1	1	0	0
Інтеграція зі Spring Boot	0	1	1	0
Підтримка React	1	1	0	0
Інтеграція з Git	1	1	1	1
Підсвічування коду та автодоповнення	1	1	1	1
Інтеграція з Postgres / ORM	0	1	1	0
Зручність відлагодження	1	1	0	1
Підтримка багатопроєктності	1	1	1	0
Кросплатформність	1	1	1	1
Загальний коефіцієнт	8	10	7	4

Використання зазначених інструментів забезпечило зручну структурування проєкту, стабільну розробку функціоналу, а також можливість швидкої перевірки працездатності за допомогою вбудованих засобів налагодження. Крім того,

інтеграція з GitHub дозволила ефективно керувати змінами, зберігати резервні копії коду та координувати версії клієнтської й серверної частин.

Таким чином, вибір мови програмування Java та відповідних середовищ розробки повністю задовольнив вимоги до бакалаврської кваліфікаційної роботи, забезпечивши технічну надійність, ефективність у реалізації функціоналу та високий рівень підтримки з боку сучасних інструментів і спільнот [12].

3.2 Реалізація клієнтської частини застосунку

Клієнтська частина вебзастосунку є основною точкою взаємодії між користувачем і функціональними можливостями системи. Її реалізація потребувала створення зручного, інтуїтивно зрозумілого та адаптивного інтерфейсу, який би дозволяв швидко виконати ключові дії: реєстрацію, авторизацію, перегляд мапи, виклик евакуатора та відстеження виконання замовлення. Для реалізації frontend використано JavaScript-бібліотеку React, яка дозволяє створювати односторінкові вебзастосунки (SPA) з можливістю оновлення окремих компонентів без повного перезавантаження сторінки.

Основною перевагою React є можливість створювати інтерфейс на основі незалежних логічних блоків – компонентів. У межах бакалаврської кваліфікаційної роботи кожна функціональна частина інтерфейсу реалізовувалася як окремий компонент, що дозволяє забезпечити структурованість коду, полегшує налагодження та подальше оновлення системи. Компоненти були побудовані із застосуванням сучасного підходу на основі хуків (React Hooks), що забезпечує зручну роботу зі станом, ефектами та подіями користувача.

Головна сторінка застосунку містить логотип, короткий опис сервісу та дві основні кнопки: «Увійти» та «Зареєструватися». Інтерфейс побудований за принципом mobile-first, що гарантує коректне відображення як на десктопних, так і на мобільних пристроях. Усі елементи оформлені відповідно до синьо-білої кольорової гами, що асоціюється з дорожньою службою та забезпечує візуальний

комфорт (рис. 2.2).

Форма реєстрації реалізована у вигляді окремого компонента з валідацією всіх полів. Користувач повинен вказати ПІБ, контактний номер, електронну пошту, логін і пароль. При введенні некоректних або неповних даних на екран виводиться відповідне повідомлення про помилку. Дані з форми передаються на сервер методом POST через HTTP-запит за допомогою бібліотеки axios (рис. 3.2).

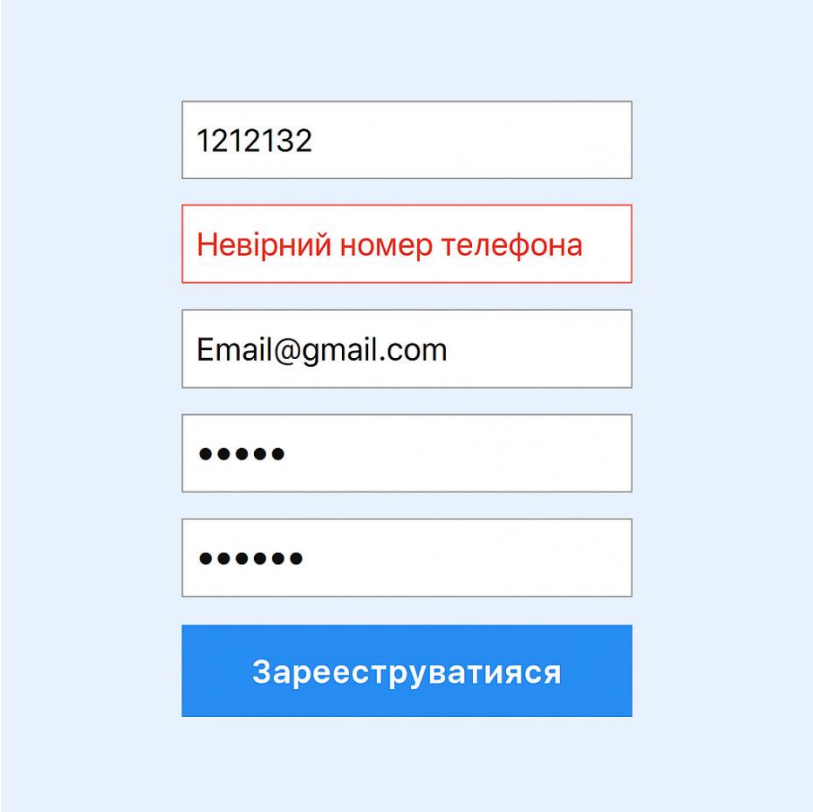

Рисунок 3.2 – Сторінка реєстрації користувача

Після авторизації або реєстрації користувач потрапляє на основну сторінку із відображенням мапи. Мапа реалізована з використанням бібліотеки Leaflet.js, яка дозволяє легко інтегрувати відкриті картографічні сервіси на основі OpenStreetMap. На мапі відображаються всі активні евакуатори, координати яких надходять із серверної частини через REST API. Кожен евакуатор позначений маркером, що містить коротку інформацію: тип транспорту, тариф, рейтинг. Користувач може

натиснути на маркер, переглянути деталі та викликати обраного водія. Приклад продемонстровано на рис. 3.3.

Рисунок 3.3 – Відображення евакуаторів на інтерактивній карті

Окремим компонентом є форма створення замовлення, яка відкривається після вибору евакуатора. У ній зазначається місце подачі, кінцева адреса, контактна інформація, тип послуги (евакуація після ДТП, порушення ПДР, технічна несправність тощо). Після надсилання форми система надсилає запит до сервера, де створюється новий запис у базі даних, а користувач отримує повідомлення про статус виклику (див. зображення 3.4).

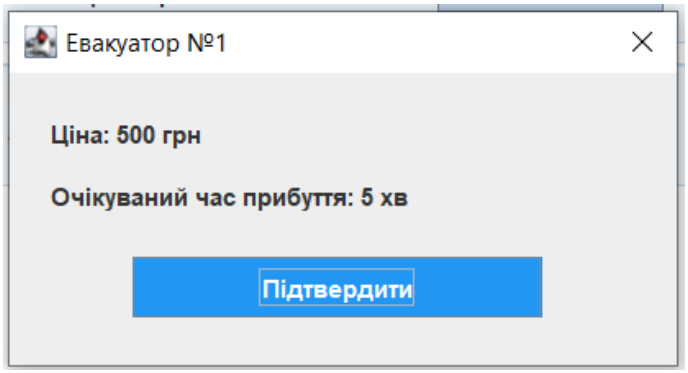

Рисунок 3.4 – Компонент оформлення замовлення евакуатора

У застосунку також реалізовано адаптивне бокове меню, яке відкривається за допомогою кнопки «гамбургер» (три вертикальні лінії у верхньому куті інтерфейсу). Це меню є зручним способом навігації, особливо для мобільних користувачів, оскільки автоматично приховується після вибору пункту та не займає постійно простір екрану. Через нього користувач має змогу швидко переглянути історію замовлень, перейти до розділу «Профіль», змінити пароль, налаштувати параметри безпеки або вийти з облікового запису.

Меню автоматично адаптується до ролі користувача. У разі входу адміністратора інтерфейс розширюється новими функціональними пунктами: перегляд замовлень усіх користувачів, перегляд журналів дій, керування списком водіїв, редагування тарифів, модерація відгуків та доступ до аналітичного модуля. Ці функції дають змогу адміністрації ефективно контролювати процес обслуговування, оперативно реагувати на проблеми та оптимізувати роботу сервісу. Наявність ролевої моделі доступу дозволяє гнучко розмежувати функціональність для звичайних користувачів і адміністраторів, що забезпечує безпеку та простоту інтерфейсу (рис. 3.5).

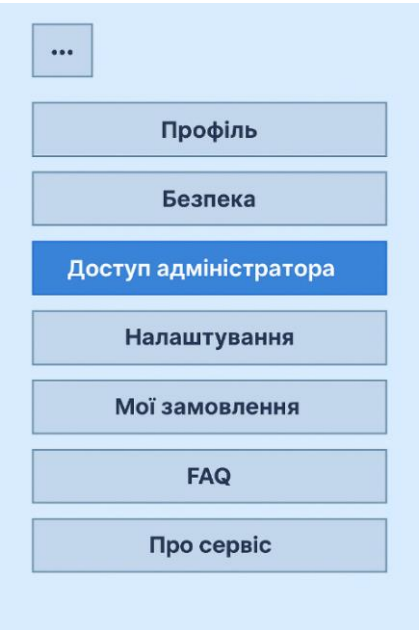

Рисунок 3.5 – Адаптивне меню з розділами профілю та керування

Вся логіка взаємодії з сервером реалізована через HTTP-запити з використанням бібліотеки axios. Статус авторизації та токен зберігаються у браузері (localStorage), що дозволяє підтримувати сесію користувача при оновленні сторінки. Крім того, на клієнтському рівні обробляються повідомлення про помилки сервера, таймаути та перевірка доступності API.

Таким чином, клієнтська частина вебзастосунку повністю виконує функції взаємодії користувача з основними можливостями системи, забезпечує швидкий відгук, зручну навігацію, сучасний інтерфейс та високу стабільність роботи при різних сценаріях використання [13].

3.3 Реалізація серверної частини застосунку

Серверна частина вебзастосунку є критично важливою складовою системи, що відповідає за обробку запитів від клієнта, взаємодію з базою даних, логіку формування відповідей, а також за забезпечення безпеки, авторизації та збереження сесій. Реалізація серверної частини була виконана на мові програмування Java 17 із використанням фреймворку Spring Boot [14], що дозволяє швидко та ефективно будувати масштабовані REST-сервіси зі структурованою логікою.

Spring Boot обрано через його широкі можливості у сфері побудови корпоративних застосунків, зручну інтеграцію з ORM-рішеннями (Hibernate, JPA), автоматичну конфігурацію компонентів, а також високий рівень підтримки REST API. Архітектура backend-частини побудована за принципом розділення відповідальностей (separation of concerns) та включає чітко визначені рівні: контролери (controller), сервіси (service), репозиторії (repository) та DTO-об’єкти для передачі даних між шарами.

Кожен модуль системи має власний контролер, який відповідає за обробку HTTP-запитів. Наприклад, UserController обробляє реєстрацію, авторизацію, запити на профіль, тоді як OrderController відповідає за створення, зміну та перегляд

замовлень на евакуацію. Усі контролери отримують дані з клієнта у вигляді DTO-об’єктів, які проходять валідацію з використанням анотацій (рис. 3.6).

Для підвищення надійності та безпеки всі запити захищено через механізм JWT (JSON Web Token), який дозволяє перевіряти роль користувача та контролювати доступ до певних ендпоінтів. Крім того, обробка виключень реалізована централізовано з використанням @ControllerAdvice, що спрощує відлагодження та забезпечує повернення стандартизованих помилок. Такий підхід дозволяє створити гнучку, зрозумілу та масштабовану серверну частину, яка легко підтримується та розширюється при додаванні нових функціональних модулів.

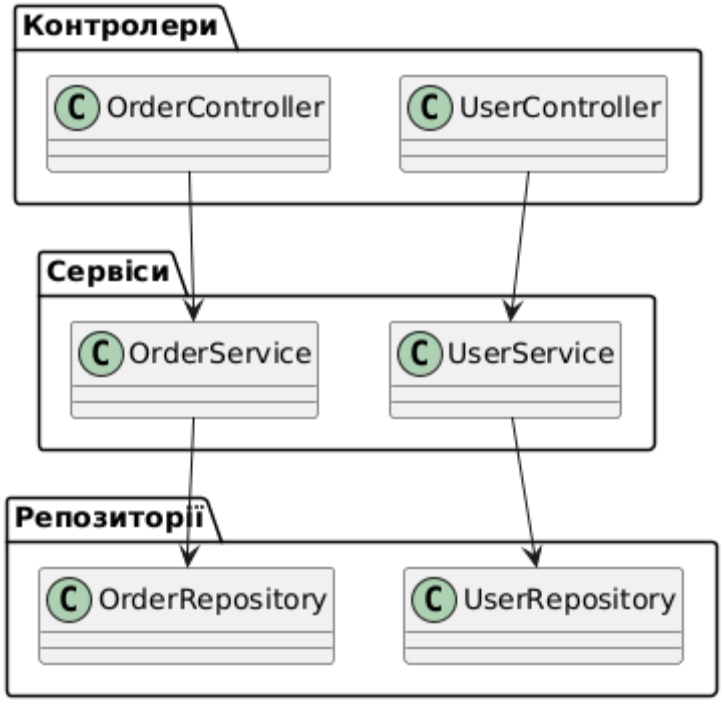

Рисунок 3.6 – Логічна структура серверної частини застосунку

Бізнес-логіка винесена до сервісного шару. Тут реалізовано обробку сценаріїв на кшталт: «створити замовлення», «знайти вільних водіїв», «розрахувати вартість евакуації», «оновити статус», «відобразити історію замовлень користувача». Наприклад, під час створення нового замовлення система перевіряє, чи є доступні

евакуатори, автоматично визначає найближчого водія (за координатами), створює відповідний запис у БД і надсилає відповідь клієнту зі статусом.

Взаємодія з базою даних реалізована за допомогою Spring Data JPA, що дозволяє працювати з сутностями як з об'єктами, не створюючи вручну SQL-запити. Основні сутності, такі як User, Order, Driver, Vehicle, Review, мають зв'язки типу «один до багатьох» та «багато до одного», які задаються через анотації @OneToMany, @ManyToOne тощо. Усі таблиці автоматично створюються на основі цих сутностей при запуску застосунку. Для зручності управління транзакціями використовується @Transactional.

Окрему увагу приділено безпеці застосунку. Реалізовано JWT-аутентифікацію (JSON Web Token), яка дозволяє авторизованим користувачам отримувати токен при вході в систему, зберігати його на клієнтській стороні та передавати з кожним запитом у заголовках (Authorization: Bearer ...). На основі токена сервер автоматично ідентифікує користувача, його роль (user, admin) і рівень доступу. Усі критичні маршрути захищені анотаціями @PreAuthorize або @Secured. Приклад продемонстровано на рис. 2.7.


```
{
  "token"eyJhbGciOIJIIzIINiIsInT5
  cCI6IkpXVCJ9.eyJztwbiOiIZIiwiAF
  OIioxNjkzODky70022os.I_gjdYnd6v
  1j6eKOALOURJ2YJS7vkO"
  ,
  "id":  35,
  "userId":  3,
  "driverId": null
  "status": "PENDING",
  "pickupLocation":"50.4425, 30.520",
  "destination":"50.4507, 30.5140",
  "fare": 500
}
```


Рисунок 3.7 – Приклад відповіді REST API при створенні замовлення

Крім цього, реалізовано глобальну обробку помилок (@ControllerAdvice), логування ключових подій (успішний вхід, створення замовлення, зміна статусу), а також конфігурацію CORS для дозволу запитів з різних доменів — що є важливим при роботі із frontend, який може розміщуватись на окремому хостингу.

Таким чином, серверна частина вебзастосунку реалізована як гнучкий, безпечний та ефективний багаторівневий модуль із чіткою логікою обробки запитів, захистом даних користувача та можливістю подальшого масштабування. Обрані інструменти — Java, Spring Boot, PostgreSQL, JWT — дозволили створити надійну архітектуру, яка відповідає вимогам до інформаційних систем [15].

3.4 Реалізація бази даних і структури зберігання даних

Реалізація бази даних є однією з ключових складових створення інформаційної системи, оскільки вона забезпечує збереження всіх критичних даних про користувачів, водіїв, замовлення, транспортні засоби, маршрути та історію взаємодії із сервісом. У межах бакалаврської кваліфікаційної роботи для реалізації системи зберігання даних обрано реляційну систему керування базами даних PostgreSQL, яка є надійною, продуктивною та відкритою за ліцензією. Вона забезпечує повноцінну підтримку транзакцій, типів зв’язків між таблицями, зовнішніх ключів, а також дозволяє реалізувати складні SQL-запити та індексування.

Для взаємодії з PostgreSQL використовується ORM-рівень JPA (Java Persistence API) із реалізацією через Hibernate. Це дозволяє описати сутності бази даних як Java-класи, а зв’язки між таблицями – як анотації (@OneToMany, @ManyToOne, @JoinColumn тощо). При запуску програми база даних може створюватися автоматично на основі моделі сутностей або оновлюватися відповідно до змін у коді. Такий підхід значно спрощує підтримку структури бази, пришвидшує розробку і забезпечує узгодженість між логікою застосунку та його

даними. Крім того, ORM-рівень мінімізує ризики людських помилок при написанні запитів і підвищує зручність у тестуванні.

Основна структура бази даних побудована на п’яти основних сутностях:

• User – зберігає дані про клієнтів: логін, пароль, ПІБ, контактний номер, роль у системі (user/admin);

• Driver – містить інформацію про водіїв евакуаторів: ПІБ, номер телефону, наявність прав, координати поточного розташування;

• Vehicle – описує транспортний засіб водія: модель, номерний знак, тип евакуатора;

• Order – містить усю інформацію про замовлення: координати точки виклику, призначення, статус (нове, підтверджено, виконується, завершено), обраний тариф, пов’язаний користувач і водій;

• Review – зберігає відгуки користувачів після завершення евакуації, рейтинг і текст повідомлення.

Між таблицями реалізовано наступні зв’язки:

• Один користувач може мати багато замовлень (User → Order);

• Один водій також може виконувати багато замовлень (Driver → Order);

• Один водій керує лише одним транспортним засобом (Driver → Vehicle);

• Кожне замовлення після завершення може містити лише один відгук (Order → Review).

У подальшому структура БД може бути легко розширена додатковими сутностями, наприклад, для реалізації модулів тарифікації, геоаналітики, історії сесій, системи сповіщень тощо. Така гнучкість дозволяє масштабувати систему без серйозного рефакторингу основної логіки. Враховуючи реляційну модель, забезпечується цілісність даних, а використання зовнішніх ключів унеможливлює появу "висячих" записів. Усе це створює надійну основу для функціонування сервісу та гарантує стабільну роботу при збільшенні обсягів даних.

Крім основних зв’язків між сутностями, важливим елементом є реалізація обмежень та перевірок на рівні бази даних, зокрема унікальності логінів, існування відповідних записів при створенні зовнішніх ключів, а також каскадного видалення або оновлення. Це дозволяє не лише захистити цілісність інформації, але й делегувати частину логіки безпосередньо на СКБД, що розвантажує серверну частину та прискорює обробку запитів. Завдяки PostgreSQL система зберігання стала надійною, гнучкою та готовою до розширення в межах як локального використання, так і повноцінного розгортання у хмарному середовищі.

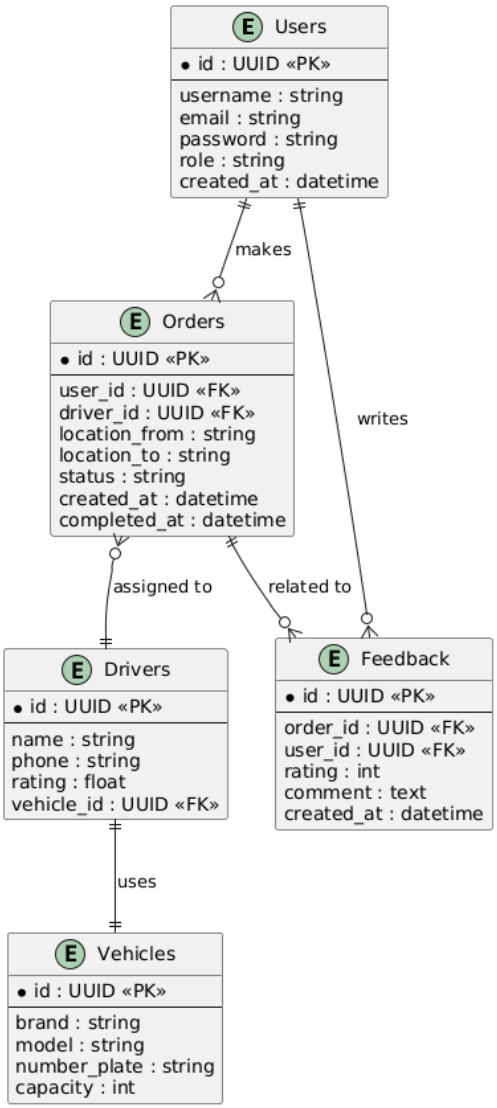

Рисунок 3.8 – Структурна схема бази даних (ER-діаграма)

ER-діаграма (рис. 3.8) наочно демонструє логіку побудови бази даних. Усі ключі, що визначають зв’язки між таблицями, реалізовані як зовнішні ключі (FOREIGN KEY), що забезпечує цілісність даних та захист від помилок під час виконання запитів. Також реалізовано первинні ключі (id) у вигляді автоінкрементованих чисел (тип serial або bigserial), що дозволяє зручно ідентифікувати кожен запис.

У рамках проєкту додатково створено SQL-скрипт, що дозволяє автоматично створити структуру бази при необхідності розгортання проєкту на новому сервері. Усі поля таблиць мають обмеження щодо обов’язковості (NOT NULL), а також валідацію з боку бекенду перед записом у базу.

Завдяки обраній структурі бази даних забезпечено:

- швидкий пошук доступних евакуаторів за координатами та статусом;
- збереження історії замовлень для кожного користувача;
- надання адміністраторам доступу до всіх записів для моніторингу;
- ефективну роботу з відгуками та рейтингами водіїв;
- забезпечення зв’язності всієї логіки системи через єдину схему.

Таким чином, реалізована база даних відповідає вимогам до вебзастосунків реального часу, підтримує багатокористувацьку роботу, забезпечує логічну зв’язність усіх даних і може масштабуватися відповідно до потреб системи [16].

3.5 Висновки

У третьому розділі безпосередньо реалізовано програмне забезпечення вебзастосунку для автоматизованого виклику евакуатора. Проведено обґрунтований вибір мови програмування та середовища розробки, у результаті чого для серверної частини використано Java та Spring Boot, а для клієнтської – JavaScript із бібліотекою React. Такий вибір забезпечив високу стабільність, модульність і масштабованість системи, а також ефективну інтеграцію між frontend та backend частинами.

Було створено повнофункціональний користувацький інтерфейс з адаптивною структурою, підтримкою інтерактивної карти та механізмом виклику евакуатора в декілька кліків. На клієнтському рівні реалізовано всі необхідні сценарії — від реєстрації до перегляду історії замовлень.

Серверна частина системи реалізована як набір REST API-сервісів, що обробляють запити користувача, керують логікою замовлень, водіїв та адміністративних функцій. Додатково впроваджено авторизацію через JWT, валідацію вхідних даних, логування та захист від несанкціонованого доступу.

Базу даних спроєктовано відповідно до принципів нормалізації з використанням реляційної СУБД PostgreSQL. Реалізовано всі необхідні зв’язки між основними сутностями, забезпечено підтримку транзакцій та цілісності даних.

Таким чином, у межах цього розділу створено повноцінну програмну реалізацію проєкту, яка відповідає всім функціональним і нефункціональним вимогам системи, закладеним у попередніх етапах бакалаврської кваліфікаційної роботи

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Методика тестування

Для перевірки працездатності вебзастосунку застосовувалося ручне функціональне тестування, яке охоплювало перевірку ключових дій користувача, а також модульне тестування основних логічних компонентів серверної частини. Усі перевірки проводилися в інтерактивному середовищі браузера з використанням реальних сценаріїв роботи системи, що максимально наближено до умов реального використання застосунку.

Особливу увагу приділено тестуванню взаємодії з формами (реєстрація, вхід, створення замовлення), відображенню мапи, роботі API-запитів, зміні статусів замовлення, доступу до панелі адміністратора. На етапі тестування оцінювалися як функціональні аспекти (правильність відповіді сервера, обробка помилок, валідація даних), так і візуальні – адаптивність інтерфейсу, розташування елементів, зручність навігації.

Модульне тестування backend-частини проводилось на рівні окремих методів сервісів: обробка реєстрації користувача, створення нового замовлення, пошук вільного водія, формування відповіді про статус. Для цього використовувалися мок-дані, а результати перевірялися порівнянням очікуваних та фактичних відповідей API [17].

Крім позитивних сценаріїв, проводилася перевірка некоректної взаємодії користувача з формами – наприклад, введення некоректного формату електронної пошти, порожні поля, надсилання запиту без авторизації. У таких випадках система коректно відображала повідомлення про помилки або блокувала дію, що підтверджує наявність ефективної валідації на стороні клієнта та сервера. Результати перевірених функціональних сценаріїв і стан виконання відображено у рис. 4.1.

№	Функція	Результат
1	Реєстрація користувача	Успішно
2	Авторизація користувача	Успішно
3	Отримання геолокації	Успішно
4	Виклик евакуатора	Успішно
5	Прийняття замовлення водієм	Успішно
6	Відображення евакуатора на карті	Успішно
7	Завершення замовлення	Успішно

Рисунок 4.1 – Результати функціонального тестування вебзастосунку

Таким чином, тестування продемонструвало стабільну роботу всіх основних модулів застосунку, включаючи як користувацький, так і адміністративний інтерфейс. Система реагує на дії користувача відповідно до закладеної логіки, коректно обробляє помилки та забезпечує зручний доступ до основних функцій. Усі виявлені на початковому етапі недоліки були виправлені до фінального етапу тестування.

4.2 Тестування розробленого програмного забезпечення

У процесі реалізації програмного забезпечення вебзастосунку для автоматизованого виклику евакуатора особливу увагу приділено його тестуванню. Тестування є невід’ємною складовою розробки, що дозволяє виявити та усунути недоліки в логіці роботи, перевірити відповідність програмного забезпечення функціональним вимогам, оцінити стабільність і коректність взаємодії між модулями системи.

Тестування проводилось у локальному середовищі за допомогою настільного комп’ютера з доступом до Інтернету. Усі модулі вебзастосунку попередньо були розгорнуті у середовищі IntelliJ IDEA, що дозволяло швидко вносити зміни та контролювати результат їх впливу на роботу системи. Клієнтська частина

тестувалася у браузері Google Chrome, а серверна — через інтеграцію з Postman, логами в IDE та тестовими викликами API.

Виконано функціональне тестування основних сценаріїв користувача. Перевірка здійснювалася за принципом «чорного ящика», що дозволяє оцінити поведінку системи без заглиблення у внутрішню реалізацію. У результаті тестування підтверджено стабільність роботи, правильність обробки запитів, а також коректність відображення елементів інтерфейсу. Особливу увагу приділено обробці помилкових даних: система правильно реагувала на спроби введення порожніх полів, некоректних номерів телефону або повторного використання зареєстрованої пошти.

У рамках функціонального тестування перевірялися такі дії:

• реєстрація нового користувача з перевіркою унікальності логіна;

• авторизація із захистом від неправильних облікових даних;

• відображення інтерактивної карти з активними евакуаторами;

• створення нового замовлення з автоматичним вибором найближчого водія;

• зміна статусу замовлення (очікується, виконується, завершено);

• перегляд історії замовлень;

• можливість залишити відгук про водія після завершення замовлення;

• доступ адміністратора до перегляду усіх замовлень.

Крім того, була проведена перевірка доступності функціоналу залежно від ролі користувача – звичайні користувачі не мали доступу до адміністративних інструментів, а всі спроби несанкціонованих запитів повертали статус помилки 403. Це підтвердило правильність реалізації ролей у системі авторизації. Також проводились перевірки сумісності застосунку з різними роздільними здатностями екранів, і виявлено, що інтерфейс коректно адаптується під мобільні пристрої, зберігаючи функціональність і читабельність усіх компонентів.

Після запуску застосунку користувач переходить на головний екран, який містить логотип, короткий опис сервісу та кнопки «Вхід» і «Реєстрація». Всі дії

супроводжуються візуальним зворотним зв’язком, що дозволяє користувачу легко орієнтуватися в інтерфейсі та розуміти статус операцій у режимі реального часу.

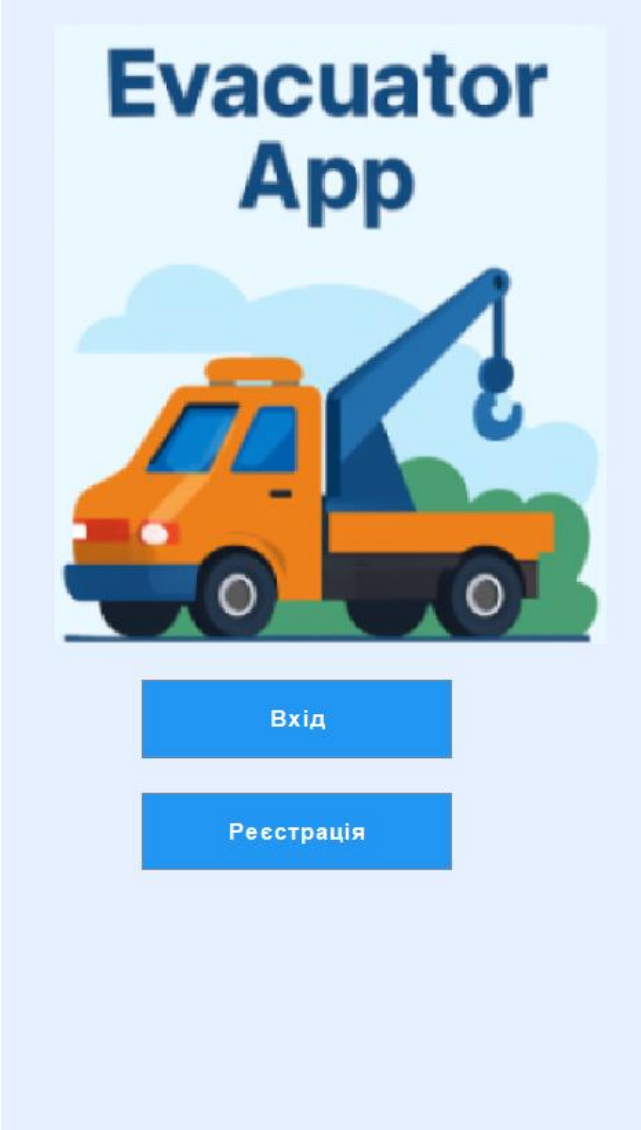

Рисунок 4.2 – Головна сторінка застосунку

При натисканні на кнопку «Реєстрація» відкривається відповідна форма, яка містить поля для введення імені, телефону, email, логіна та пароля. Система перевіряє правильність введених даних та повідомляє про помилки, якщо такі є.

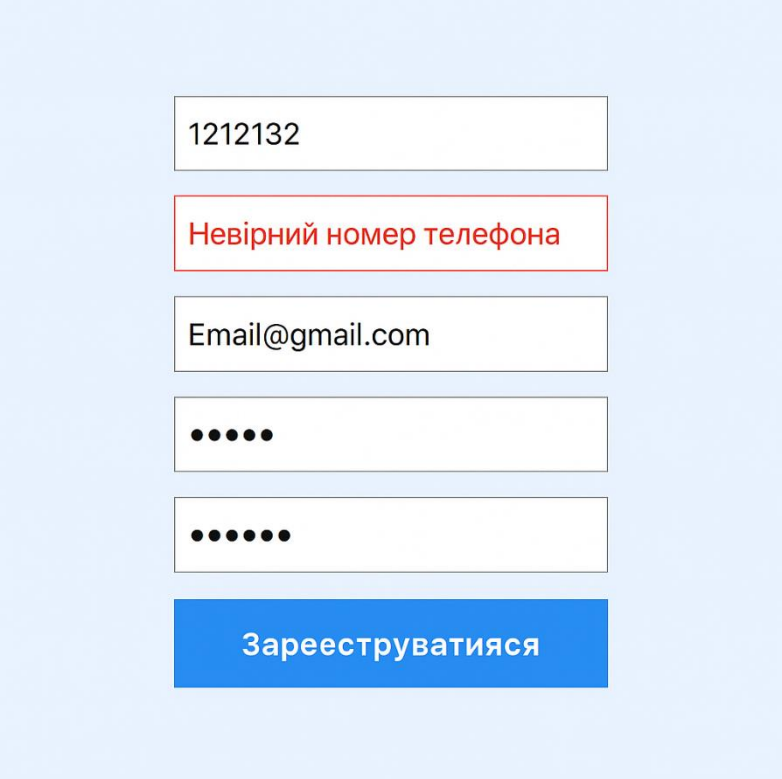

Рисунок 4.3 – Форма реєстрації користувача

Після реєстрації користувач потрапляє на головний екран із картою. Карта реалізована з використанням бібліотеки Leaflet та демонструє активних водіїв у радіусі доступності. Дані про координати евакуаторів отримуються через REST API з серверної частини.

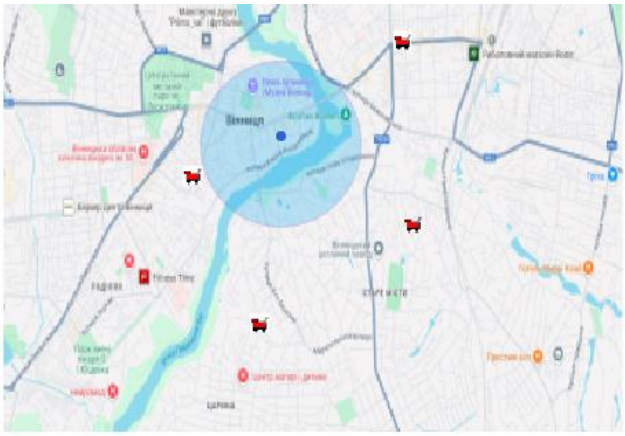

Рисунок 4.4 – Відображення евакуаторів на карті

Користувач має можливість вибрати точку подачі та замовити евакуатор. Після створення замовлення система автоматично змінює його статус та відображає результат у вигляді повідомлення.

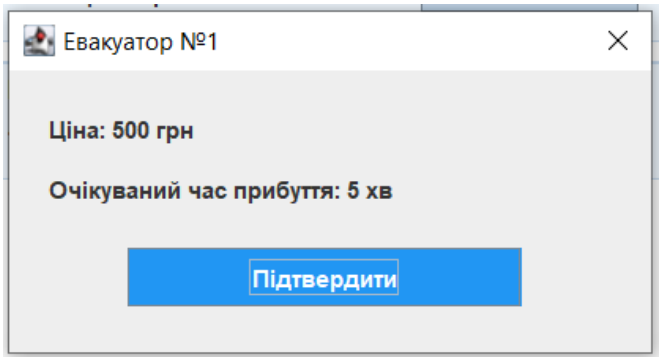

Рисунок 4.5 – Створення нового замовлення

Адміністратор має доступ до повного списку замовлень, можливість переглядати профілі водіїв, змінювати статус замовлення та контролювати систему в цілому.

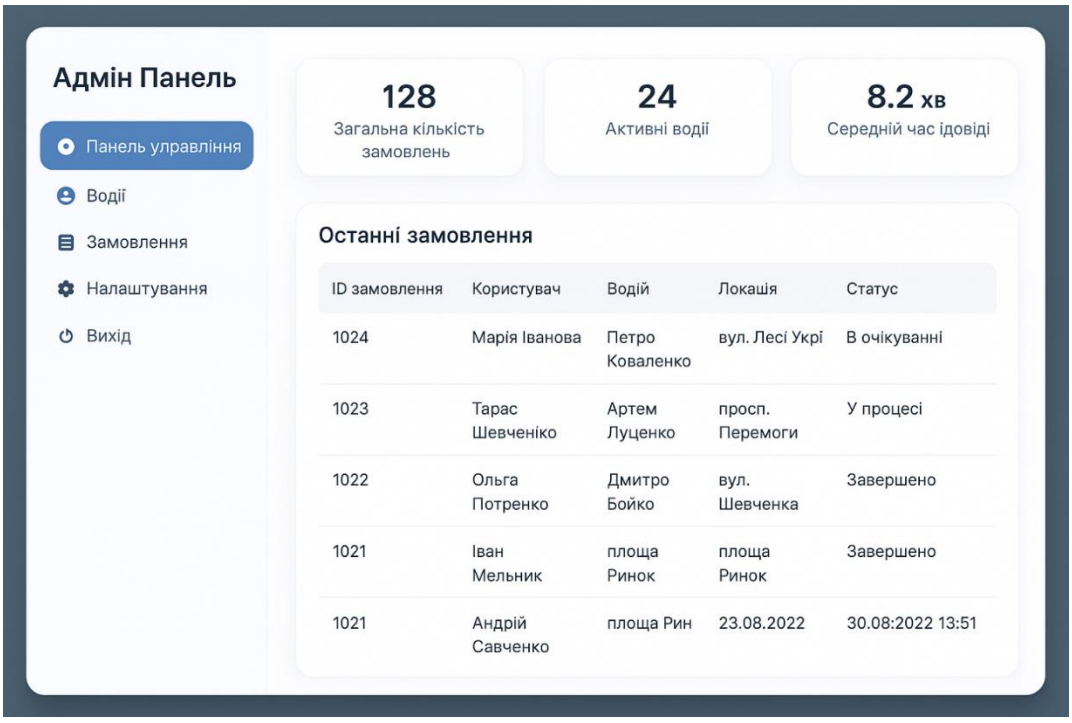

Рисунок 4.6 – Інтерфейс адміністратора

Інтерфейс перевірено на адаптивність до різних розмірів екранів. Застосунок коректно працює на мобільних пристроях, планшетах та настільних ПК. При зміні розміру вікна елементи інтерфейсу масштабуються без втрати функціональності.

Крім основних сценаріїв, перевірялися граничні випадки:

- спроба надіслати пусту форму;
- дублювання логіна;
- відсутність Інтернет-з'єднання;
- повторне натискання кнопки під час обробки запиту.

Застосунок показав стабільну роботу навіть при багаторазовому виконанні одних і тих самих дій. Повідомлення про помилки відображаються своєчасно, а функції блокуються до завершення обробки поточного запиту.

Таким чином, розроблене програмне забезпечення пройшло повне тестування і підтвердило свою працездатність, відповідність функціональним вимогам та готовність до практичного використання. Виявлені незначні недоліки були усунуті в процесі фінальної доробки інтерфейсу, а основні функціональні модулі успішно виконують свої задачі.

4.3 Інструкція користувача

Для початку роботи з вебзастосунком користувачу достатньо перейти за відповідним посиланням у браузері з будь-якого пристрою, що підтримує сучасні вебтехнології. Застосунок не потребує встановлення, оскільки працює повністю у браузерному середовищі. Це забезпечує зручність доступу, мобільність і швидкий старт для користувачів без необхідності інсталяції сторонніх компонентів [18].

Система доступна на десктопних комп'ютерах, ноутбуках, планшетах і смартфонах, а адаптивний інтерфейс дозволяє зручно користуватись сервісом незалежно від розміру екрана чи типу пристрою. Усі дії відбуваються у реальному часі через інтерактивний інтерфейс.

Перед доступом до функціоналу система перевіряє, чи користувач авторизований. Якщо ні – автоматично пропонується виконати вхід або зареєструватися. Без входу користувач не може замовити евакуатор або переглядати історію замовлень.

Після успішної авторизації відкривається головна сторінка з інтерактивною картою, на якій відображаються найближчі евакуатори. Користувач має змогу вибрати точку подачі, ознайомитися з інформацією про водія (тариф, рейтинг, тип транспорту) та викликати евакуатор в один клік. Після виконання заявки користувач має змогу залишити відгук про якість послуги [19].

Для стабільної роботи системи браузер має підтримувати JavaScript, CSS3, HTML5 та мати активне інтернет-з'єднання. Інформація про мінімальну та рекомендовану конфігурацію пристроїв для роботи з вебзастосунком представлена у табл. 4.1 та 4.2.

Таблиця 4.1 – Мінімальна конфігурація:

Тип процесора	MediaTek Helio P90
Об'єм оперативної пам'яті	2 ГБ для 64-разрядної системи
Місце на жорсткому диску	1 ГБ
Операційна система	Android 11

Таблиця 4.2 – Рекомендована конфігурація:

Тип процесора	.4GHz Octa-core Qualcomm Snapdragon 888
Об'єм оперативної пам'яті	4 ГБ для 64-разрядної системи
Місце на жорсткому диску	4 ГБ
Операційна система	Android 14

Таким чином, вебзастосунок може працювати на більшості сучасних пристроїв без встановлення додаткових компонентів. Система перевіряє наявність авторизації перед кожною взаємодією з основними модулями. Увійшовши до облікового запису, користувач отримує повний доступ до функціоналу: створення та перегляду замовлень, оцінювання послуг, керування профілем. Інтерфейс побудований таким чином, щоб взаємодія із сервісом займала мінімум часу, а користування максимально комфортним [20].

4.4 Висновки

У цьому розділі було проведено тестування функціональних можливостей вебзастосунку для автоматизованого виклику евакуатора та наведено інструкцію користувача щодо використання основних можливостей системи. У результаті функціонального та ручного тестування підтверджено працездатність розробленого застосунку, правильність реалізації логіки обробки запитів, стабільну роботу інтерфейсу та відповідність функціоналу технічному завданню. Система успішно виконує всі базові сценарії взаємодії: реєстрацію користувача, авторизацію, виклик евакуатора через інтерактивну мапу, зміну статусу замовлення та перегляд історії. Особливу увагу під час тестування було приділено перевірці роботи адаптивного інтерфейсу, який коректно масштабується для різних типів пристроїв. Розроблена інструкція демонструє покроковий процес взаємодії із застосунком, починаючи з реєстрації, закінчуючи оцінкою наданих послуг, що підтверджує орієнтованість системи на кінцевого користувача. Розглянуто технічні вимоги до клієнтських та серверних пристроїв, що дозволяє однозначно визначити умови, необхідні для стабільної експлуатації вебзастосунку. Таким чином, результати тестування підтверджують повну готовність системи до практичного використання та демонструють її функціональну завершеність і зручність.

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи було реалізовано повноцінний вебзастосунок для автоматизованого пошуку, виклику та моніторингу послуг евакуації транспортних засобів. Вказаний програмний продукт призначений для спрощення взаємодії між користувачем і службою евакуації в критичних або форс-мажорних ситуаціях, коли швидкість реагування та доступність сервісу відіграють вирішальну роль.

На першому етапі виконання роботи було проведено аналіз предметної області, визначено основні проблеми існуючих рішень і сформульовано вимоги до майбутнього застосунку. Було досліджено структуру подібних сервісів (Uklon, Bolt, Autobooking тощо), вивчено механізми геолокації, інтеграції з картою та формування замовлення, після чого побудовано архітектуру майбутньої системи.

Основні задачі, які були виконані в ході реалізації бакалаврської кваліфікаційної роботи:

– проаналізовано предметну галузь і виявлено ключові технічні потреби цільових користувачів;

– розроблено вимоги до програмного забезпечення, сформовано функціональні сценарії та модель взаємодії;

– побудовано загальну архітектуру системи із розділенням на клієнтську та серверну частину;

– реалізовано графічний інтерфейс з інтерактивною картою, формами входу, реєстрації, замовлення, перегляду історії тощо;

– створено backend із використанням Java, Spring Boot і REST API, впроваджено авторизацію за допомогою JWT;

– спроєктовано та реалізовано базу даних у PostgreSQL для збереження користувачів, водіїв, замовлень і відгуків;

– виконано тестування застосунку: перевірено основні функції, валідацію форм, адаптивність інтерфейсу, обробку помилок;

– підготовлено інструкцію користувача, описано технічні вимоги до роботи системи та умови розгортання на сервері.

У результаті виконаних робіт було створено зручний і функціональний застосунок, який забезпечує повний цикл взаємодії користувача з сервісом: від входу в систему до виклику евакуатора та залишення відгуку. Реалізована система є логічно завершеною, протестованою та готовою до подальшого розгортання в реальних умовах. Вона відповідає сучасним вимогам до вебсервісів — зокрема кросплатформності, масштабованості, безпеки та зручності у використанні.

Розроблений вебзастосунок може бути адаптований до мобільного формату або інтегрований із зовнішніми службами (наприклад, службами оплати, смс-сповіщеннями, GPS-трекерами). У подальшому він може стати основою для створення комерційного сервісу або впровадження у рамках міських програм допомоги водіям.

Таким чином, поставлена мета бакалаврської кваліфікаційної роботи досягнута повною мірою, а створений програмний продукт є прикладом якісного, структурованого та функціонально завершеного рішення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Uklon офіційний сайт. – [Електронний ресурс]. – Режим доступу: https://www.uklon.com.ua (дата звернення 16.04.2025)

2. Easy Towing офіційний сайт. – [Електронний ресурс]. – Режим доступу: https://easytowing.app/ (дата звернення 16.04.2025)

3. Evacuator PRO офіційний сайт. – [Електронний ресурс]. – Режим доступу: https://apps.apple.com/ua/app/ (дата звернення 16.04.2025)

4. Гама, Е., Хелм, Р., Джонсон, Р., Влісідес, Дж. Шаблони проектування. Стандартні рішення проблем проектування програмного забезпечення. – Київ: Діалектика, 2002. – 352 с.

5. Еріх Бетта, Річард Гельм та ін. Шаблони проектування. Стандартні рішення повторно використовуваних об'єктно-орієнтованих програм. – Київ: Діалектика, 2015. – 366 с.

6. Сомерс Р. Інженерія програмного забезпечення. – 9-те видання. – К.: Діалектика, 2016. – 800 с.

7. Мартін Р. Чистий код. Створення, аналіз та рефакторинг. – К.: Діалектика, 2017. – 464 с.

8. Практика побудови веборієнтованих інформаційних систем. Підручник / М.С. Матієшин. – Львів: ЛНУ ім. І. Франка, 2018. – 312 с.

9. Архітектура програмних систем: навч. посібник / В.О. Соловйов. – Харків: ХНУРЕ, 2021. – 196 с.

10. Глушков А.В. Інформаційні системи та технології. – К.: Кондор, 2019. – 504 с.

11. Фленаган Д. Java. Повний довідник. 10-те видання. – СПб.: Пітер, 2021. – 1376 с.

12. Deitel P., Deitel H. Java: How to Program. – 11th Edition. – Pearson, 2018. – 1248 p.

13. Бойко В.І. Розробка мобільних застосунків: підручник. – Вінниця: ВНТУ, 2020. – 285 с.

14. Марченко Д.В. Проектування мобільних додатків на базі Android. – Дніпро: ДНУ, 2021. – 300 с.

15. Android Development Patterns / Alex Ruiz. – Addison-Wesley Professional, 2016. – 512 p.

16. Використання API Google Maps у мобільних застосунках / Матеріали конференції ITIMC-2021. – Львів, 2021. – 156 с.

17. Робота з графічними елементами в Java Swing: методичні вказівки / О.В. Романюк. – Вінниця: ВНТУ, 2022. – 87 с.

18. Збірник матеріалів студентської конференції "Інформаційні технології: стан та перспективи", Вінниця, 2024. – ВНТУ, 2024. – 220 с.

19. Методичні вказівки до виконання курсових проєктів з дисципліни "Проєктний практикум" для студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. В.В. Войтко, Г.О. Черноволик. – Вінниця: ВНТУ, 2022 – 44с.

20. Лисенко Г.Л. Методичні вказівки до оформлення курсових проектів (робіт) у Вінницькому національному технічному університеті / Уклад. Г.Л. Лисенко, А.Г. Буда, Р.Р. Обертюх. – Вінниця: ВНТУ, 2006. – 60 с.

ДОДАТКИ

Додаток А. Технічне завдання

62

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Зав. кафедри ПЗ, проф., д.т.н.
_____ О.Н. Романюк
«25» березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційна роботу «Розробка веборієнтованої інформаційної системи для автоматизованого пошуку, виклику та моніторингу послуг евакуації транспортних засобів» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи
_____ викладач. О.В. Мельник
«24» березня 2025 р.
Виконав:
Студент гр. ЗПІ-21б М.В. Великий
«24» березня 2025 р.

м. Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Веборієнтованої інформаційної системи для автоматизованого пошуку, виклику та моніторингу послуг евакуації транспортних засобів».

Галузь застосування – вебтехнології.

2. Підстава для розробки:

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол №18 від 18 лютого 2025 р.

3. Мета і призначення розробки:

Метою є створення інформаційної системи, яка забезпечує користувачам зручний механізм пошуку та виклику евакуатора в екстрених ситуаціях, а також адміністрування та моніторинг замовлень через вебінтерфейс. Система призначена для водіїв та диспетчерських служб.

4. Вихідні дані для проведення НДР

1. Banks, E. Porcello. *Learning React: Modern Patterns for Developing React Apps*, 2nd Edition. O'Reilly Media, 2020. – 307 с.
2. D. Flanagan. *JavaScript. The Definitive Guide*, 7th Edition. O'Reilly Media, 2020. – 706 с.
3. А. В. Столяров. *Основы проектирования информационных систем*. СПб: Питер, 2022. – 384 с.
4. М. М. Іващенко, І. В. Смірнова. *Системи підтримки прийняття рішень*. Харків: ХНУРЕ, 2021. – 236 с.
5. A. Jones. *Mastering TypeScript Programming*. Walzone Press, 2025. – 259 с.

5. Технічні вимоги

Вебзастосунок має працювати в сучасних браузерах, бути адаптивним для мобільних і десктопних пристроїв. Клієнтська частина – React (HTML/JS), серверна – REST API з базою даних PostgreSQL. Підтримка ОС Windows.

6. Конструктивні вимоги

Розроблений програмний продукт повинен мати зручний, адаптивний інтерфейс користувача, відповідати принципам UX/UI дизайну, забезпечувати швидку навігацію між основними екранами та відповідати стандартам розробки вебзастосунків. Уся текстова та графічна документація повинна відповідати чинним стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до бакалаврської кваліфікаційної роботи;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз цифрових сервісів виклику евакуатора та формулювання завдань проєкту	25.03.25 – 06.04.25
2	Проєктування архітектури веб-застосунку та розробка інтерфейсу користувача	07.04.25 – 20.04.25
3	Реалізація клієнтської та серверної частин системи з інтеграцією карти	20.04.25 – 07.05.25
4	Тестування функціоналу та розробка адміністративного модуля	07.05.25 – 20.05.25
5	Оформлення пояснювальної записки та підготовка матеріалів до захисту	20.05.25– 30.05.25

10. Порядок контролю та прийняття.

Усі етапи бакалаврської роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

Додаток Б. Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Веборієнтованої інформаційної системи для автоматизованого пошуку, виклику та моніторингу послуг евакуації транспортних засобів

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, ФІТКІ, ЗПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 3,4 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ к.т.н., доц. каф. ПЗ Мельник О. В.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Великий М. В.
(підпис) (прізвище, ініціали)

Додаток В. Лістинг програми

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
import java.util.Random;

public class EvacuatorFullApp {
    private JFrame frame;
    private CardLayout layout;
    private JPanel appContent;
    private JPanel sideMenu;
    private boolean isMenuVisible = false;

    private final int MIN_EVACS = 3;
    private final int MAX_EVACS = 6;
    private final int MAP_WIDTH = 360;
    private final int MAP_HEIGHT = 250;
    private final int CENTER_X = MAP_WIDTH / 2;
    private final int CENTER_Y = MAP_HEIGHT / 2;
    private final int SAFE_RADIUS = 60;
    private final int ICON_SIZE = 24;

    public EvacuatorFullApp() {
        frame = new JFrame("Evacuator App");
        frame.setUndecorated(true);
        frame.setSize(400, 700);
        frame.setLocationRelativeTo(null);
```

```
frame.setLayout(null);

JPanel phoneFrame = new JPanel(null) {
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.BLACK);
        g.fillRect(0, 0, getWidth(), 4); // верхня
        g.fillRect(0, 0, 4, getHeight()); // ліва
        g.fillRect(getWidth() - 4, 0, 4, getHeight()); // права
        g.fillRect(0, getHeight() - 4, getWidth(), 4); // нижня
    }
};
phoneFrame.setBounds(0, 0, 400, 700);
phoneFrame.setOpaque(false);
frame.setContentPane(phoneFrame);

layout = new CardLayout();
appContent = new JPanel(layout);
appContent.setBounds(6, 6, 388, 688);
appContent.setBackground(new Color(245, 250, 255));
phoneFrame.add(appContent);

appContent.add(startScreen(), "start");
appContent.add(loginScreen(), "login");
appContent.add(registerScreen(), "register");
appContent.add(mainAppScreen(), "main");
appContent.add(faqScreen(), "faq");
```

```
    appContent.add(ordersScreen(), "orders");
    appContent.add(infoScreen(), "info");

    frame.setVisible(true);
}

private JPanel startScreen() {
    JPanel panel = new JPanel(null);
    panel.setBackground(new Color(230, 240, 255));

    JLabel logo = new JLabel(new ImageIcon("logo.png"));
    logo.setBounds(40, 20, 320, 359);
    panel.add(logo);

    JButton login = styledButton("Вхід");
    login.setBounds(90, 400, 180, 45);
    login.addActionListener(e -> layout.show(appContent, "login"));
    panel.add(login);

    JButton register = styledButton("Реєстрація");
    register.setBounds(90, 465, 180, 45);
    register.addActionListener(e -> layout.show(appContent, "register"));
    panel.add(register);

    return panel;
}

private JPanel loginScreen() {
```

```
    JPanel panel = new JPanel(null);
    panel.setBackground(new Color(230, 240, 255));

    JLabel label = new JLabel("Вхід до акаунта", SwingConstants.CENTER);
    label.setBounds(100, 100, 180, 30);
    panel.add(label);

    JTextField login = new JTextField("Користувач");
    login.setBounds(90, 150, 200, 30);
    panel.add(login);

    JPasswordField pass = new JPasswordField("Пароль");
    pass.setBounds(90, 190, 200, 30);
    panel.add(pass);

    JButton confirm = styledButton("Увійти");
    confirm.setBounds(110, 240, 160, 40);
    confirm.addActionListener(e -> layout.show(appContent, "main"));
    panel.add(confirm);

    return panel;
}

private JPanel registerScreen() {
    JPanel panel = new JPanel(null);
    panel.setBackground(new Color(230, 240, 255));

    JLabel label = new JLabel("Реєстрація", SwingConstants.CENTER);
```

```
    label.setBounds(100, 50, 180, 30);
    panel.add(label);

    JTextField name = new JTextField("Ім’я");
    name.setBounds(90, 100, 200, 30);
    panel.add(name);

    JTextField email = new JTextField("Email");
    email.setBounds(90, 140, 200, 30);
    panel.add(email);

    JPasswordField pass = new JPasswordField("Пароль");
    pass.setBounds(90, 180, 200, 30);
    panel.add(pass);

    JPasswordField confirm = new JPasswordField("Підтвердження");
    confirm.setBounds(90, 220, 200, 30);
    panel.add(confirm);

    JButton confirmBtn = styledButton("Зареєструватися");
    confirmBtn.setBounds(100, 270, 180, 40);
    confirmBtn.addActionListener(e -> layout.show(appContent, "main"));
    panel.add(confirmBtn);

    return panel;
}

private JButton styledButton(String text) {
```

```
        JButton btn = new JButton(text);
        btn.setBackground(new Color(33, 150, 243));
        btn.setForeground(Color.WHITE);
        btn.setFont(new Font("Arial", Font.BOLD, 14));
        btn.setFocusPainted(false);
        return btn;
    }
    private JPanel mainAppScreen() {
        JPanel panel = new JPanel(null);
        panel.setBackground(Color.WHITE);

        // JLayeredPane для карти та маркерів
        JLayeredPane layeredMap = new JLayeredPane();
        layeredMap.setBounds(10, 10, MAP_WIDTH, MAP_HEIGHT);

        JLabel map = new JLabel(new ImageIcon("vinnytsia_map.png"));
        map.setBounds(0, 0, MAP_WIDTH, MAP_HEIGHT);
        layeredMap.add(map, Integer.valueOf(0));

        JPanel markerLayer = new JPanel(null);
        markerLayer.setOpaque(false);
        markerLayer.setBounds(0, 0, MAP_WIDTH, MAP_HEIGHT);
        layeredMap.add(markerLayer, Integer.valueOf(1));

        panel.add(layeredMap);

// Синя точка користувача з радіусом
        JLabel user = new JLabel() {
```

```
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                g.setColor(new Color(0, 0, 255, 80)); // прозорий радіус
                g.fillOval(0, 0, SAFE_RADIUS * 2, SAFE_RADIUS * 2);
                g.setColor(Color.BLUE); // центр
                g.fillOval(SAFE_RADIUS - 6, SAFE_RADIUS - 6, 12, 12);
            }
        };
        user.setBounds(CENTER_X - SAFE_RADIUS, CENTER_Y - SAFE_RADIUS,
SAFE_RADIUS * 2, SAFE_RADIUS * 2);
        markerLayer.add(user);

// Спавн евакуаторів як зображення
        int count = new Random().nextInt(3) + 3; // 3–5
        ArrayList<Point> placed = new ArrayList<>();
        for (int i = 0; i < count; i++) {
            int x, y;
            boolean overlap;
            do {
                x = 40 + new Random().nextInt(MAP_WIDTH - 80);
                y = 40 + new Random().nextInt(MAP_HEIGHT - 80);
                overlap = distance(x, y, CENTER_X, CENTER_Y) < SAFE_RADIUS + 16;
                for (Point p : placed) {
                    if (distance(x, y, p.x, p.y) < ICON_SIZE + 10) {
                        overlap = true;
                        break;
                    }
```

```
            }
        } while (overlap);
        placed.add(new Point(x, y));

        JLabel evac = new JLabel(new ImageIcon("evac.png"));
        evac.setBounds(x, y, ICON_SIZE, ICON_SIZE);
        markerLayer.add(evac);
    }

// Додай в кінець обов’язково:
    panel.add(markerLayer);

    JTextField addressField = new JTextField("Введіть адресу (неактивно)");
    addressField.setBounds(20, MAP_HEIGHT + 20, 348, 30);
    panel.add(addressField);

    // Список евакуаторів
    int yStart = MAP_HEIGHT + 60;
    for (int i = 0; i < placed.size(); i++) {
        JPanel card = new JPanel(null);
        card.setBounds(20, yStart, 348, 60);
        card.setBackground(new Color(240, 248, 255));
        card.setBorder(BorderFactory.createLineBorder(new Color(200, 220, 240)));

        JLabel name = new JLabel("Евакуатор №" + (i + 1));
        name.setFont(new Font("Arial", Font.BOLD, 13));
        name.setBounds(10, 5, 160, 20);
        card.add(name);
```

```
        JLabel info = new JLabel("~" + (5 + i * 3) + " хв | " + (500 + i * 30) + " грн");
        info.setBounds(10, 25, 140, 20);
        card.add(info);

        JButton orderBtn = new JButton("Замовити");
        orderBtn.setBounds(220, 15, 110, 30);
        final int idx = i;
        orderBtn.addActionListener(e -> showOrderDialog("Евакуатор №" + (idx + 1),
(500 + idx * 30) + " грн", (5 + idx * 3) + " хв"));
        card.add(orderBtn);

        panel.add(card);
        yStart += 70;
    }

    // Кнопка меню
    JButton menuBtn = new JButton("≡");
    menuBtn.setBounds(10, 10, 40, 30);
    menuBtn.setBackground(new Color(33, 150, 243));
    menuBtn.setForeground(Color.WHITE);
    menuBtn.setFont(new Font("Arial", Font.BOLD, 18));
    menuBtn.setFocusPainted(false);
    menuBtn.setBorderPainted(false);
    menuBtn.addActionListener(e -> {
        if (!isMenuVisible) {
            sideMenu = createSidebarMenu(panel);
            frame.getLayeredPane().add(sideMenu, JLayeredPane.POPUP_LAYER);
```

```
            sideMenu.setVisible(true);
            isMenuVisible = true;
        }
    });
    panel.setComponentZOrder(menuBtn, 0);
    panel.add(menuBtn);

    JPanel exitBar = new JPanel();
    exitBar.setBounds(150, MAP_HEIGHT + 380, 100, 8);
    exitBar.setBackground(new Color(180, 180, 180));
    exitBar.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
    exitBar.addMouseListener(new MouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e) {
            System.exit(0);
        }
    });
    layeredMap.add(exitBar, Integer.valueOf(2));

    return panel;
}

private double distance(int x1, int y1, int x2, int y2) {
    return Math.sqrt(Math.pow(x1 - x2, 2) + Math.pow(y1 - y2, 2));
}
private JPanel createSidebarMenu(JPanel parent) {
    JPanel menu = new JPanel(null);
    menu.setBounds(0, 0, 220, 700);
```

```
menu.setBackground(new Color(225, 240, 255));

JButton backBtn = new JButton("←");
backBtn.setBounds(10, 10, 40, 30);
backBtn.addActionListener(e -> {
    frame.getLayeredPane().remove(menu);
    frame.repaint();
    isMenuVisible = false;
});
menu.add(backBtn);

String[] items = {
        "Профіль", "Безпека", "Налаштування", "Мої замовлення",
        "FAQ", "Про сервіс", "Вийти"
};
String[] screens = {
        "main", "main", "main", "orders",
        "faq", "info", "start"
};

int y = 60;
for (int i = 0; i < items.length; i++) {
    JButton btn = new JButton(items[i]);
    btn.setBounds(20, y, 180, 35);
    btn.setBackground(new Color(200, 220, 240));
    btn.setFocusPainted(false);
    final String screen = screens[i];
    btn.addActionListener(e -> {
```

```
            layout.show(appContent, screen);
            frame.getLayeredPane().remove(menu);
            frame.repaint();
            isMenuVisible = false;
        });
        menu.add(btn);
        y += 45;
    }
    return menu;
}

private JPanel ordersScreen() {
    JPanel panel = new JPanel(null);
    panel.setBackground(new Color(240, 248, 255));

    JLabel title = new JLabel("Мої замовлення:");
    title.setBounds(20, 50, 200, 20);
    panel.add(title);

    JLabel info = new JLabel("Немає активних замовлень");
    info.setBounds(20, 80, 300, 20);
    panel.add(info);

    JButton back = styledButton("← Назад");
    back.setBounds(20, 10, 100, 30);
    back.addActionListener(e -> layout.show(appContent, "main"));
    panel.add(back);
```

```
    return panel;
}

class Evacuator {
    private String name;
    private int price;
    private int eta;

    public Evacuator(String name, int price, int eta) {
        this.name = name;
        this.price = price;
        this.eta = eta;
    }

    public String getName() {
        return name;
    }

    public int getPrice() {
        return price;
    }

    public int getEta() {
        return eta;
    }

    @Override
    public String toString() {
```

```
        return name + " | " + price + " грн | " + eta + " хв";
    }
}

class OrderHistory {
    private ArrayList<String> orders = new ArrayList<>();

    public void add(String order) {
        orders.add(order);
    }

    public void printAll() {
        System.out.println("Історія замовлень:");
        for (String o : orders) {
            System.out.println("- " + o);
        }
    }

    public int count() {
        return orders.size();
    }
}

class ServerStatus {
    private boolean isConnected = true;

    public void reconnect() {
        System.out.println("Спроба перепідключення до сервера...");
```

```
            isConnected = true;
        }

        public void disconnect() {
            System.out.println("Втрачено з'єднання із сервером.");
            isConnected = false;
        }

        public boolean isConnected() {
            return isConnected;
        }
    }


    private void showOrderSuccess(String evacName) {
        JDialog successDialog = new JDialog(frame, "Підтверджено", true);
        successDialog.setLayout(null);
        successDialog.setSize(300, 150);
        successDialog.setLocationRelativeTo(frame);

        JLabel msg = new JLabel("Ваше замовлення на " + evacName + " успішно
підтверджено.");
        msg.setBounds(20, 30, 260, 30);
        successDialog.add(msg);

        JButton ok = new JButton("OK");
        ok.setBounds(100, 70, 100, 30);
        ok.addActionListener(e -> successDialog.dispose());
```

```
    successDialog.add(ok);

    successDialog.setVisible(true);
}

private String getOrderStatus() {
    String[] statuses = {"Очікує водія", "Водій прямує", "Виконано"};
    return statuses[new Random().nextInt(statuses.length)];
}

private JPanel faqScreen() {
    JPanel panel = new JPanel(null);
    panel.setBackground(new Color(240, 248, 255));

    JLabel title = new JLabel("FAQ:");
    title.setBounds(20, 50, 200, 20);
    panel.add(title);

    JLabel q1 = new JLabel("Як викликати евакуатор?");
    q1.setBounds(20, 80, 300, 20);
    panel.add(q1);

    JLabel a1 = new JLabel("Натисніть кнопку 'Замовити'");
    a1.setBounds(20, 100, 300, 20);
    panel.add(a1);

    JButton back = styledButton("← Назад");
    back.setBounds(20, 10, 100, 30);
```

```
    back.addActionListener(e -> layout.show(appContent, "main"));
    panel.add(back);

    return panel;
}

private JPanel infoScreen() {
    JPanel panel = new JPanel(null);
    panel.setBackground(new Color(240, 248, 255));

    JLabel info = new JLabel("Про сервіс:");
    info.setBounds(20, 50, 200, 20);
    panel.add(info);

    JLabel desc = new JLabel("EvacuatorApp — цілодобова служба евакуації.");
    desc.setBounds(20, 80, 340, 20);
    panel.add(desc);

    JButton back = styledButton("← Назад");
    back.setBounds(20, 10, 100, 30);
    back.addActionListener(e -> layout.show(appContent, "main"));
    panel.add(back);

    return panel;
}

private void showOrderDialog(String name, String price, String time) {
    JDialog dialog = new JDialog(frame, name, false);
```

```
        dialog.setLayout(null);
        dialog.setSize(340, 180);
        dialog.setLocation(frame.getX() + 30, frame.getY() + 500);

        JLabel label = new JLabel("Ціна: " + price);
        label.setBounds(20, 20, 300, 20);
        dialog.add(label);

        JLabel timeLabel = new JLabel("Очікуваний час прибуття: " + time);
        timeLabel.setBounds(20, 50, 300, 20);
        dialog.add(timeLabel);

        JButton pay = new JButton("Підтвердити");
        pay.setBounds(60, 90, 200, 30);
        pay.setBackground(new Color(33, 150, 243));
        pay.setForeground(Color.WHITE);
        pay.addActionListener(e -> dialog.dispose());
        dialog.add(pay);

        dialog.setVisible(true);
    }


    public static void main(String[] args) {
        SwingUtilities.invokeLater(EvacuatorFullApp::new);
    }
}
```

Додаток Г. Ілюстративна частина

ГРАФІЧНА ЧАСТИНА

РОЗРОБКА ВЕБОРІЄНТОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ АВТОМАТИЗОВАНОГО ПОШУКУ, ВИКЛИКУ ТА МОНІТОРИНГУ ПОСЛУГ ЕВАКУАЦІЇ ТРАНСПОРТНИХ ЗАСОБІВ

Рисунок Г.1 – Титульний аркуш

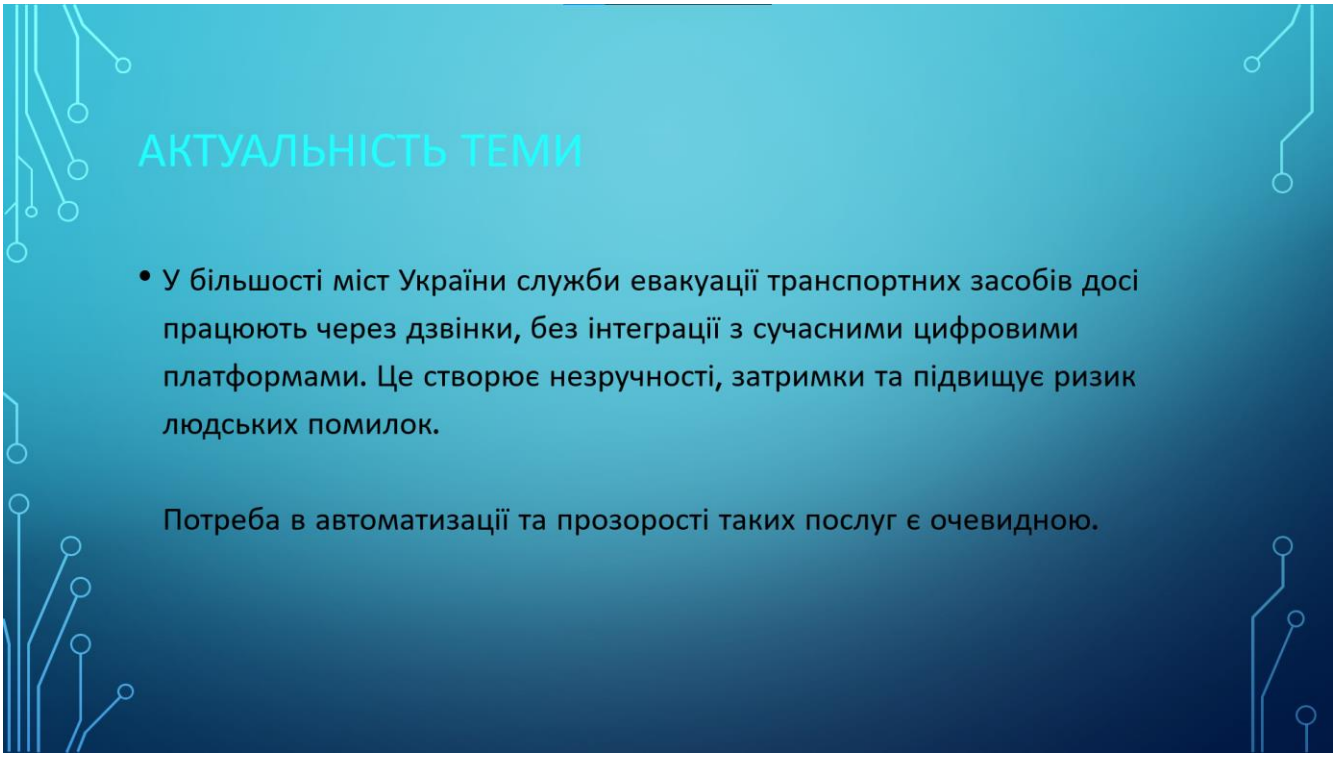

Рисунок Г.2 – Актуальність теми

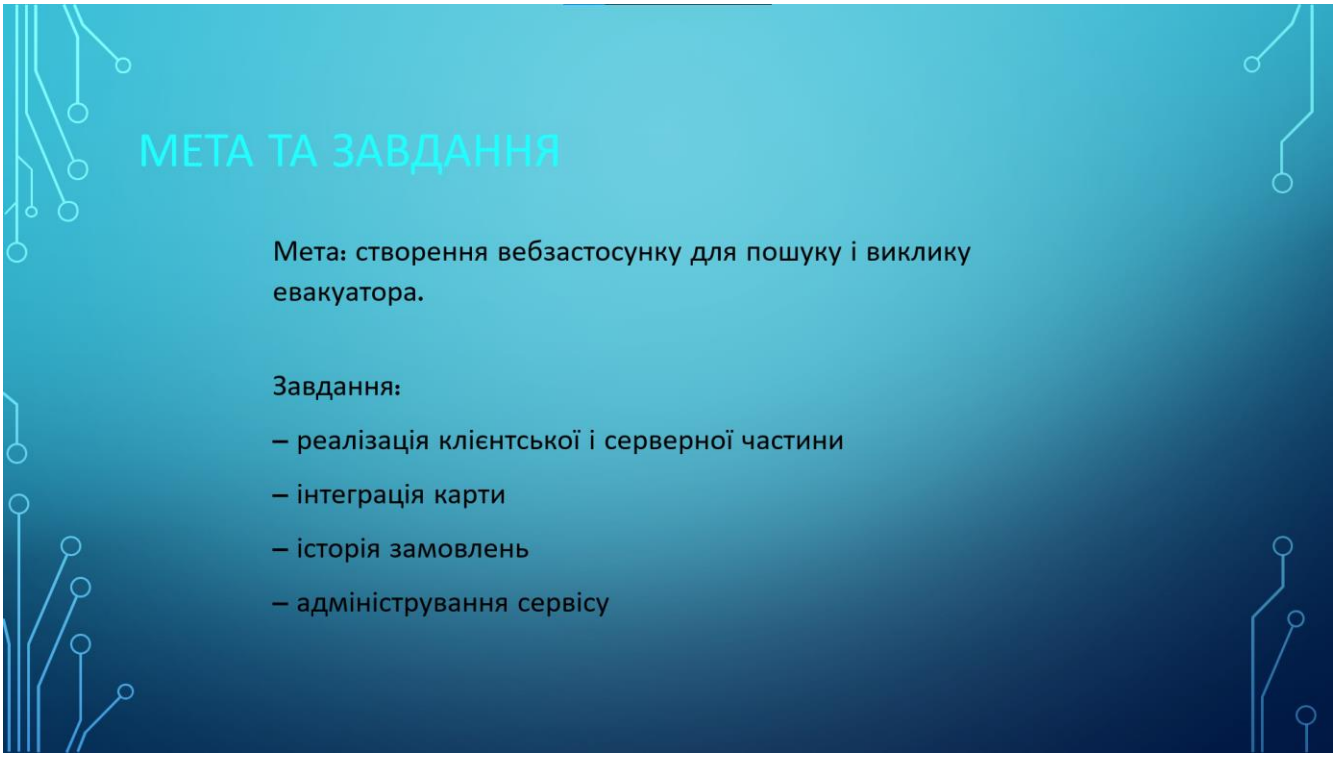

Рисунок Г.3 – Мета, предмет та завдання

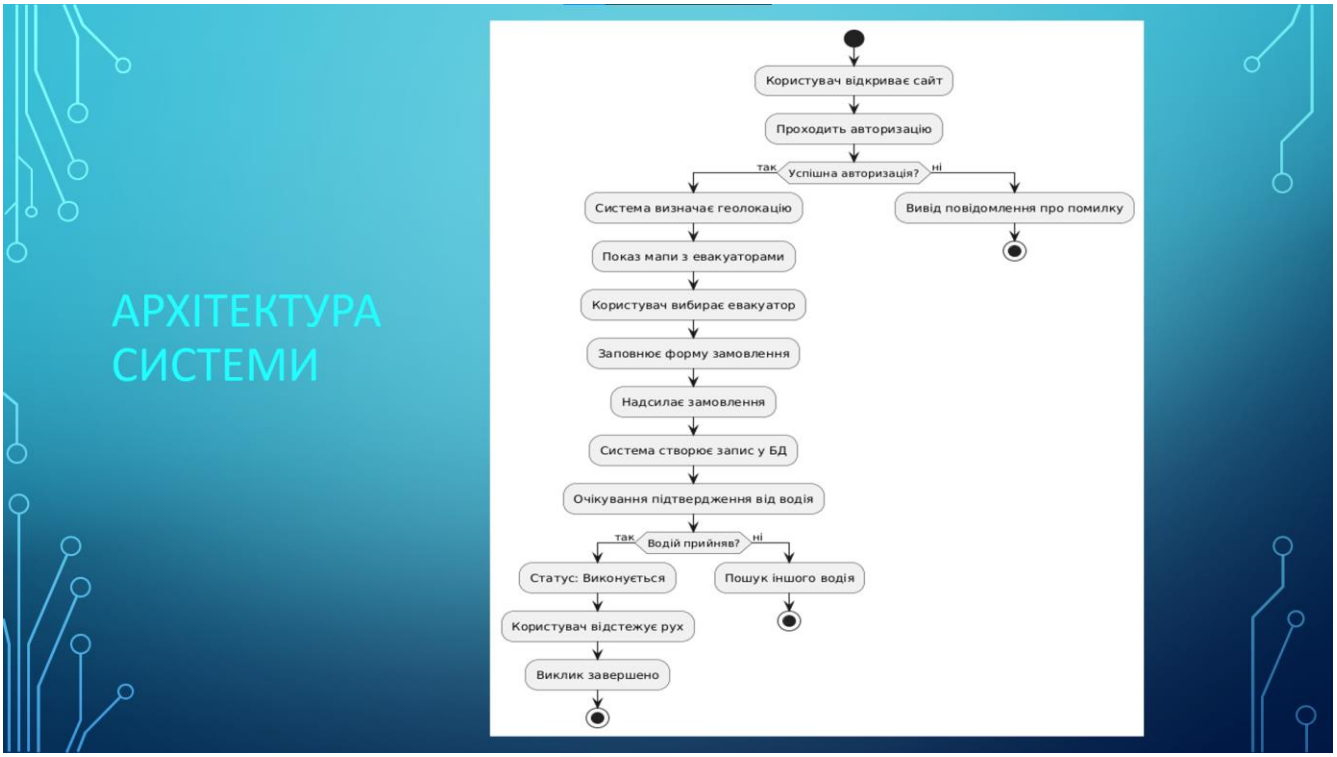

Рисунок Г.4 – Архітектура системи

Рисунок Г.5 – Технології реалізації

Рисунок Г.6 – Інтерфейс користувача

Рисунок Г.7 – Мапа та евакуатори

Рисунок Г.8 – Список евакуаторів

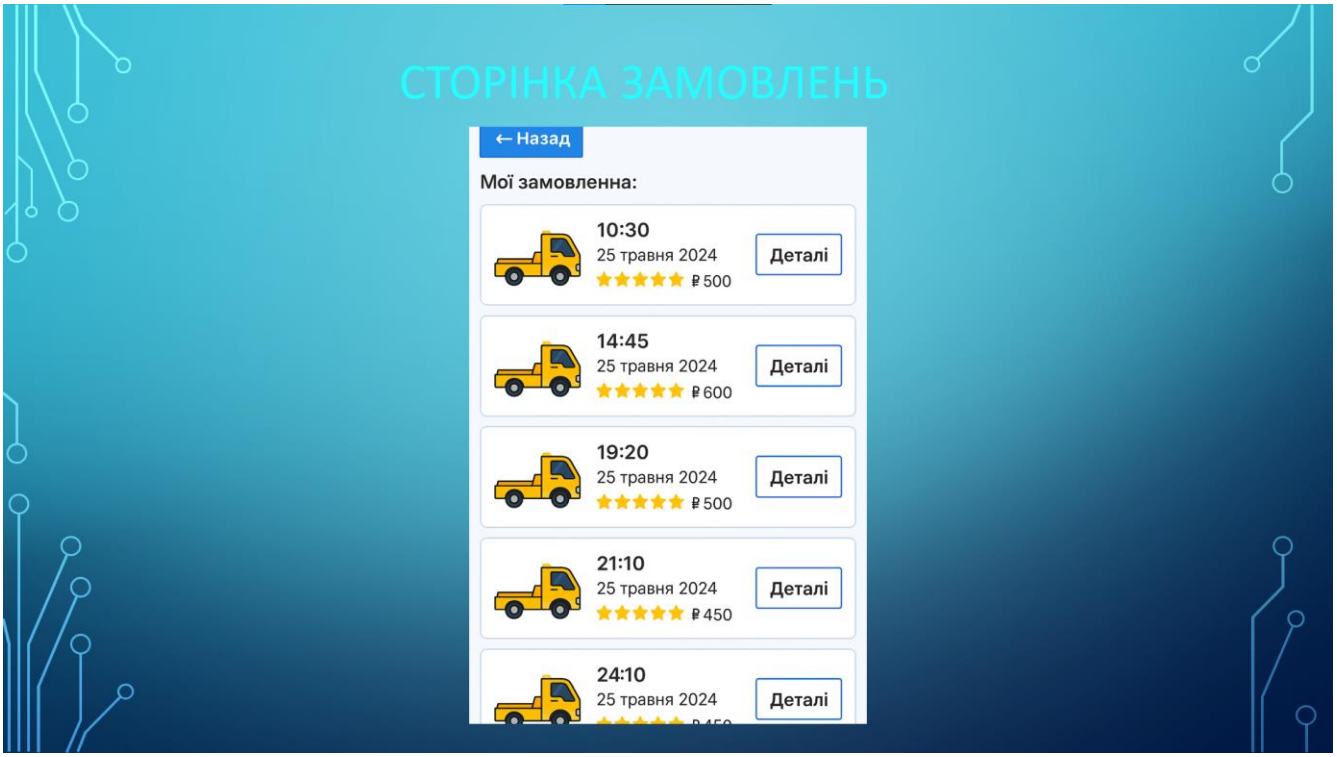

Рисунок Г.9 – Сторінка замовлень

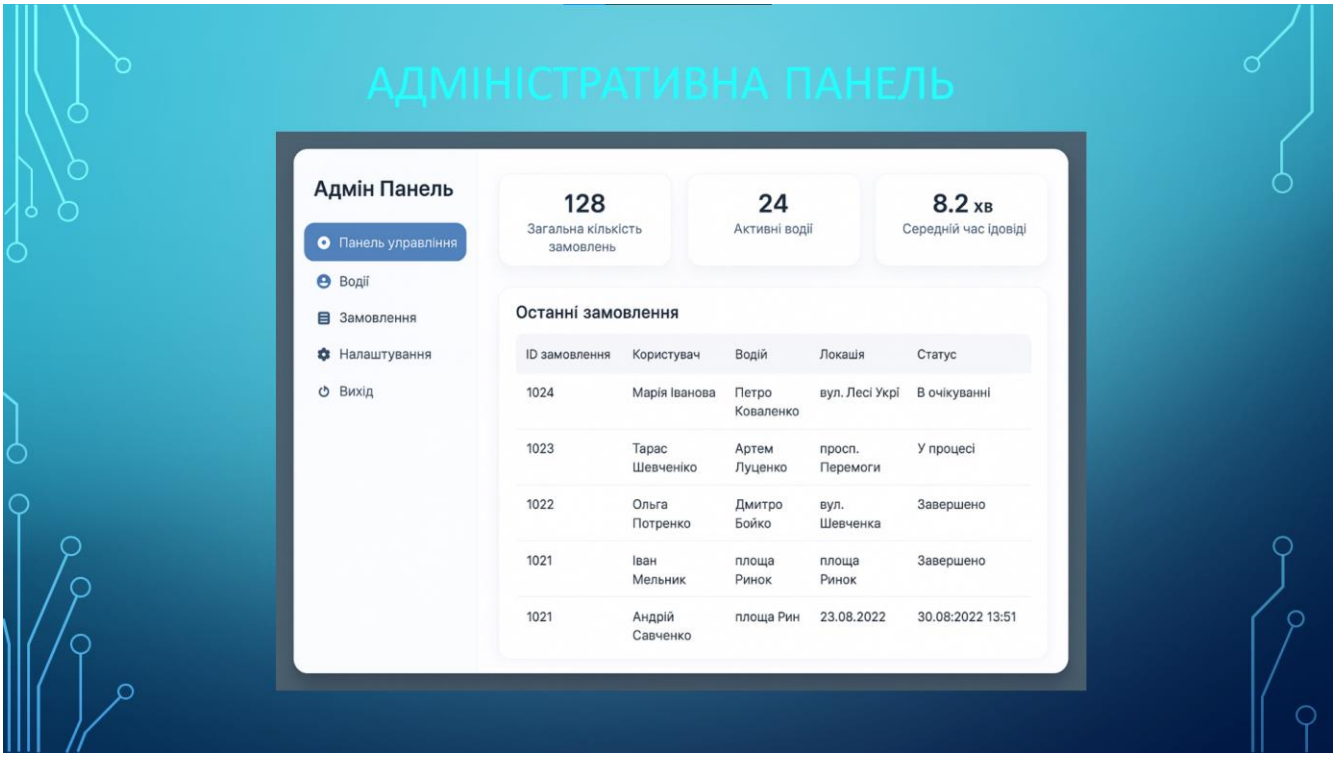

Рисунок Г.10 – Адміністративна панель

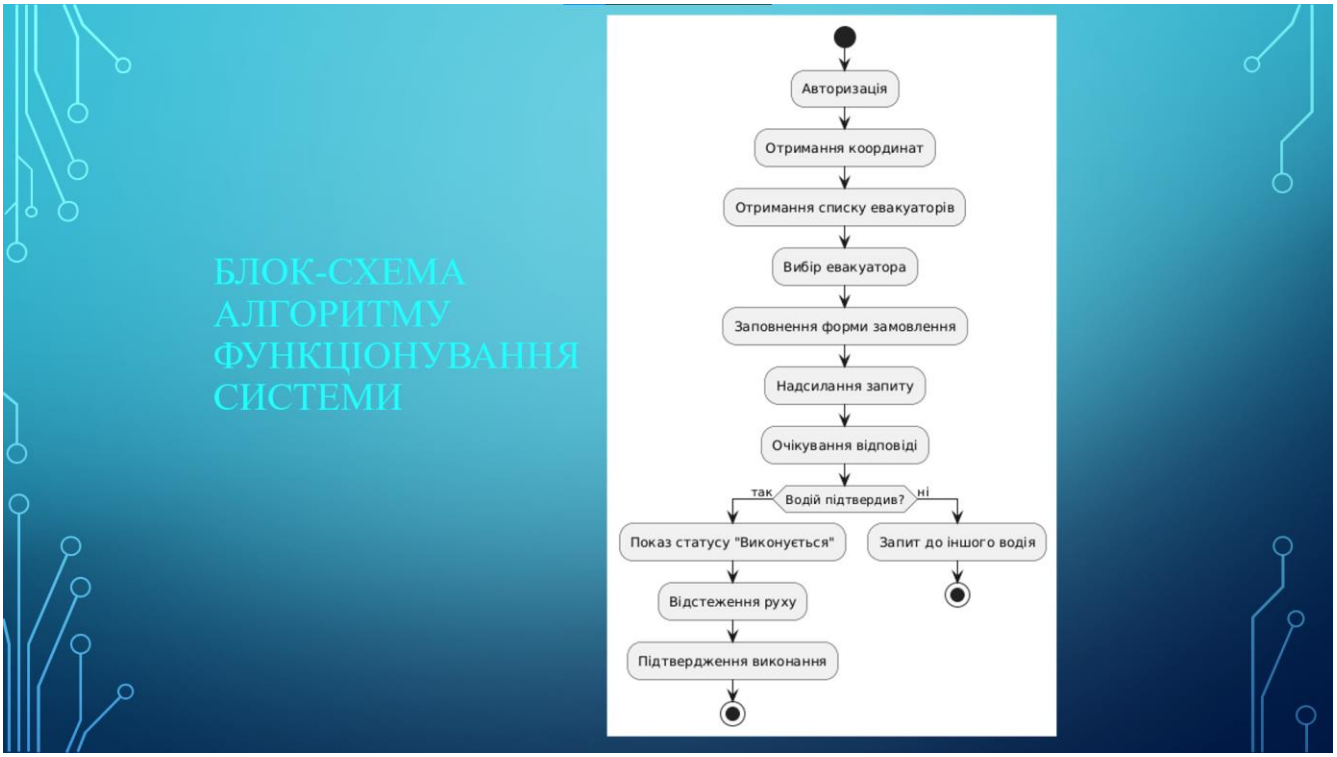

Рисунок Г.11 – Блок-схема алгоритму функціонування системи

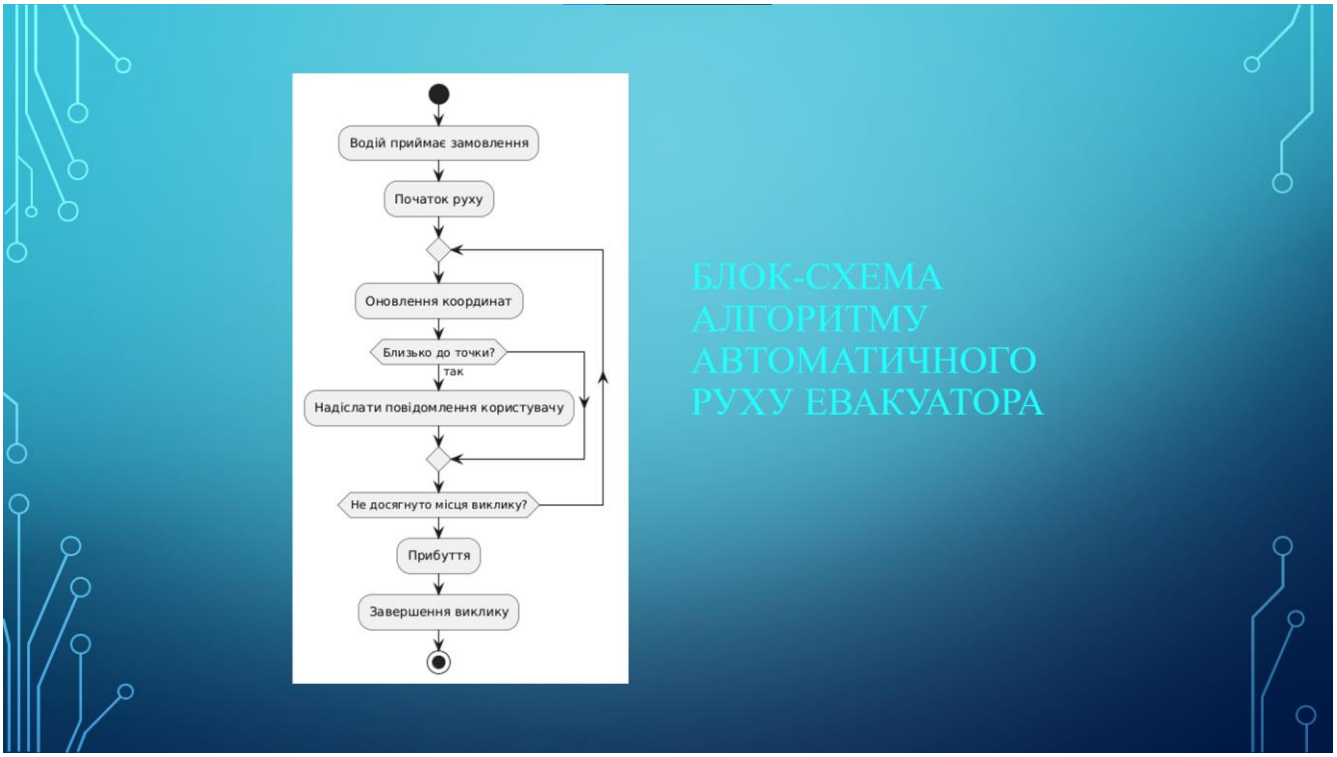

Рисунок Г.12 – Блок-схема алгоритму автоматичного руху евакуаторів

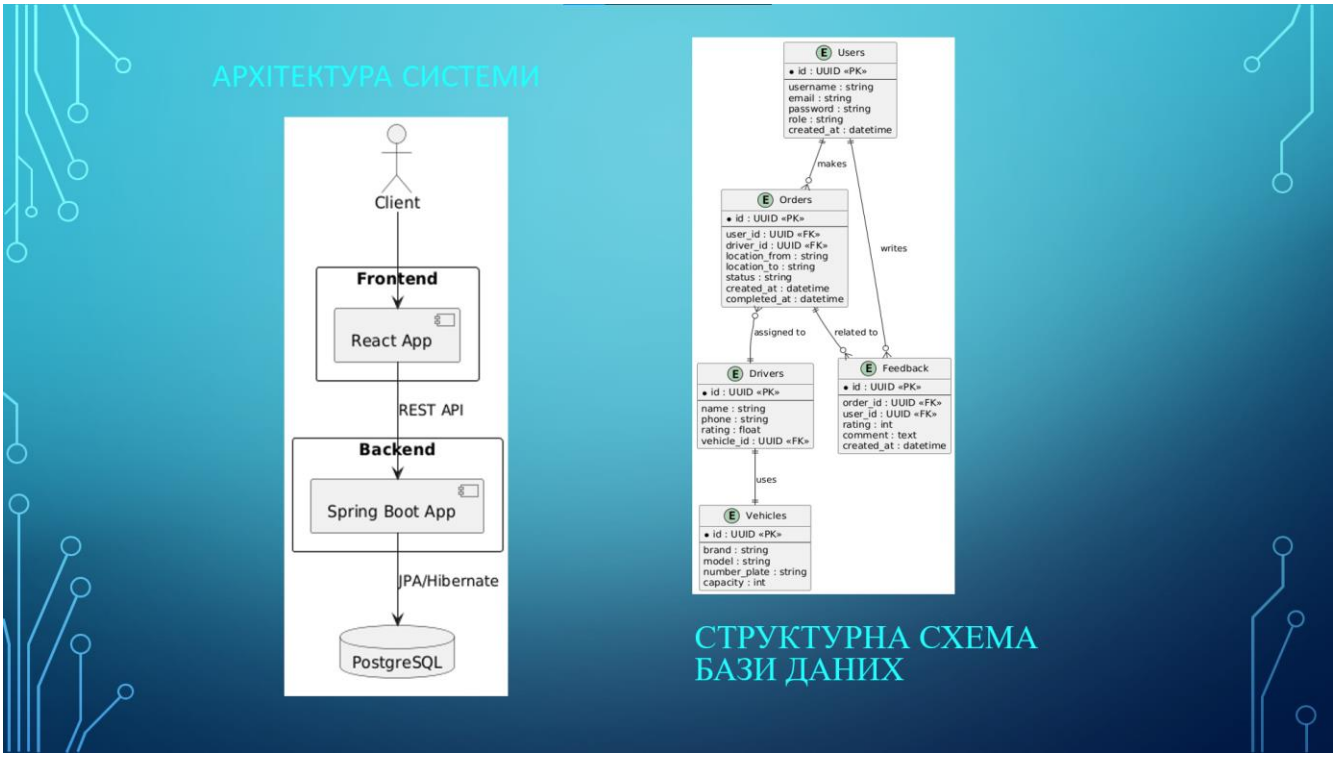

Рисунок Г.13 – Блок-схеми архітектури системи та структурна схема бази даних

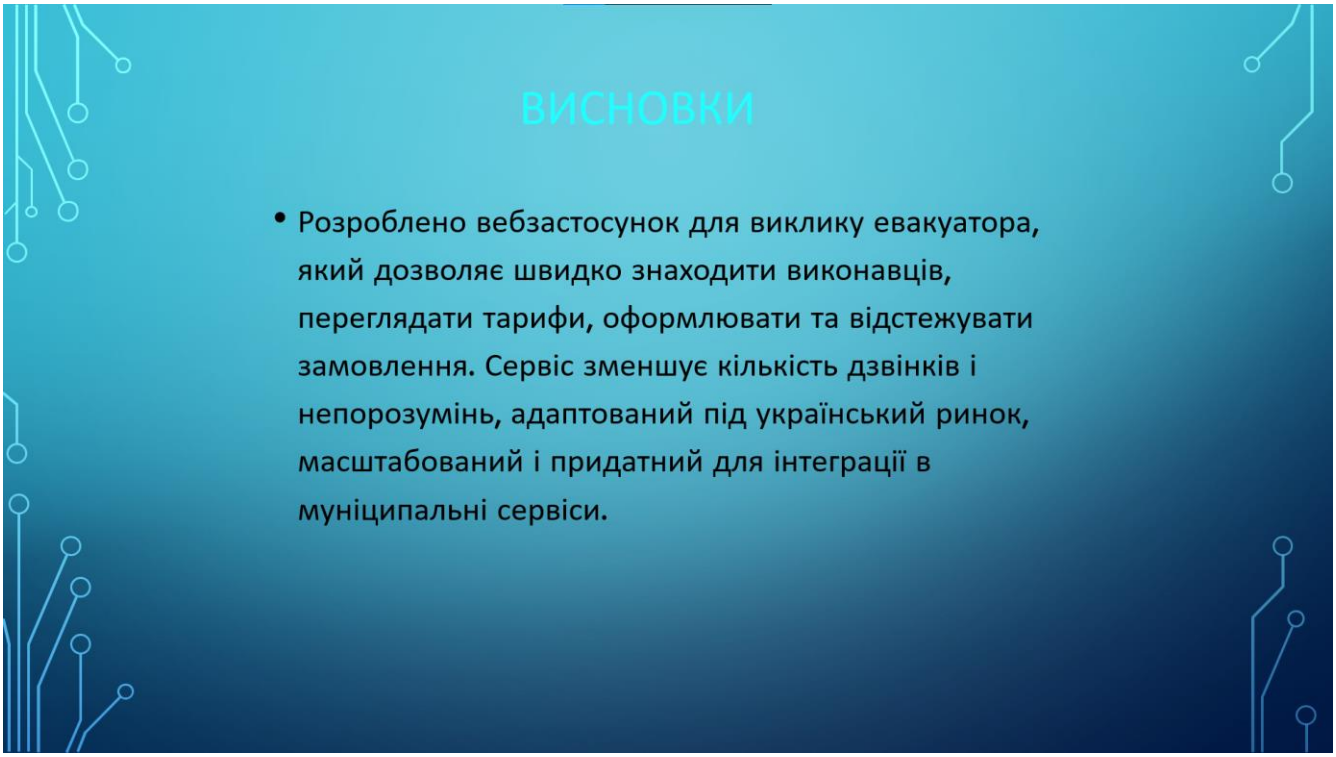

Рисунок Г.14 – Висновки

Рисунок Г.15 – Закінчення презентації