МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА КОМПЛЕКСНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ УПРАВЛІННЯ СЛУЖБОЮ ТАКСІ ЗАСОБАМИ FLUTTER ТА POSTGRESQL»**

Виконав: студент 4 курсу, групи 6.1212-пі-2
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
А.О. Сапко
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, доцент, к.т.н. Лимаренко Ю.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент директор ТОВ Голден-Тім Калін М.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Сапку Артему Олександровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка комплексної інформаційної системи для управління службою таксі засобами Flutter та PostgreSQL

керівник роботи Лимаренко Юлія Олексіївна, к.т.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 01.06.2026 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та постановка задачі.
2. Проєктування системи.
3. Програмна реалізація та тестування компонентів системи.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 09.12.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	05.01.2026	
2.	Збір вихідних даних.	19.01.2026	
3.	Обробка методичних та теоретичних	09.02.2026	
	джерел.		
4.	Розробка першого та другого розділу.	16.03.2026	
5.	Розробка третього розділу.	18.05.2026	
6.	Оформлення та нормоконтроль	25.05.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2026	

Студент _____ А.О. Сапко
(підпис) (ініціали та прізвище)

Керівник роботи _____ Ю.О. Лимаренко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка комплексної інформаційної системи для управління службою таксі засобами Flutter та PostgreSQL»: 49 с., 12 рис., 3 табл., 12 джерел.

АВТОРИЗАЦІЯ, АДМІН-ПАНЕЛЬ, ТАКСІ, ANDROID, AWS, BUCKET, FLUTTER, GOOGLE-APIS3, POSTGRESQL, TWILIO, VUE.

Об’єкт дослідження – процес проєктування та програмної реалізації клієнт-серверної системи таксі.

Мета роботи – розробка масштабованої клієнт-серверної системи таксі та підвищення продуктивності її розробки за рахунок використання сучасних кросплатформних і вебтехнологій.

Методи дослідження – системний аналіз, об’єктно-орієнтоване проєктування, моделювання архітектури програмного забезпечення, ручне тестування.

В результаті роботи розроблено клієнтський додаток для взаємодії користувачів з цільною системою таксі засобами фреймворку Flutter. Розроблено серверну частину на базі середовища Node.js, з використанням фреймворку Express. Протестована цілісність та стабільність системи методами мануального тестування, як з боку клієнтського застосунку, так і серверної частини.

SUMMARY

Bachelor’s qualifying paper “Development of a Comprehensive Information System for Taxi Service Management using Flutter and PostgreSQL”: 49 pages, 12 figures, 3 tables, 12 references.

ADMIN PANEL, ANDROID, AUTHORIZATION, AWS, BUCKET, FLUTTER, GOOGLE-APIS3, POSTGRESQL, TAXI, TWILIO, VUE.

The object of the study is the process of designing and software implementation of a client-server taxi system.

The purpose of the work is to develop a scalable client-server taxi system and improve its development productivity through the use of modern cross-platform and web technologies.

Research methods include system analysis, object-oriented design, software architecture modeling, and manual testing.

As a result of the work, a client application for user interaction with the taxi system was developed using the Flutter framework, alongside an administrative web panel based on the Vue.js reactive framework, and a server-side application in the Node.js (Express.js) environment with AWS S3 cloud storage integration. The integrity and stability of the system were verified using manual testing methods for both the client applications and the server side.

ЗМІСТ

Завдання на кваліфікаційну роботу……...2
Реферат 4
Summary 5
Вступ..... 7
1 Аналіз предметної області та постановка задачі 9
1.1 Аналіз предметної області 9
1.2 Аналіз ринку предметної області 10
1.3 Обґрунтування вибору технологій 11
2 Проєктування системи 14
2.1 Моделювання системи та архітектурних рішень..... 14
2.2 Проєктування бази даних 19
2.3 Проєктування rest api 23
2.4 Валідація rest api серверної частини системи 32
2.5 Архітектура системи 33
3 Програмна реалізація та тестування компонентів системи 36
3.1 Програмна реалізація клієнтської частини системи..... 36
3.2 Проведення тестів системи 43
Висновки 47
Перелік посилань..... 49

ВСТУП

Мобільні технології докорінно змінили сферу пасажирських перевезень. Якщо раніше замовлення таксі здійснювалося переважно через телефонні дзвінки до диспетчерських служб, то сучасні інформаційні системи дають змогу користувачам викликати авто через мобільний застосунок, автоматично прокладати маршрут, розраховувати вартість поїздки та відстежувати місцезнаходження водія в реальному часі. Для успішного функціонування такого сервісу критично важливо поєднати зручний інтерфейс для різних ролей (пасажир, водій, адміністратор), стабільну серверну частину, надійну базу даних та безперебійну інтеграцію зі сторонніми сервісами.

Кваліфікаційна робота пов’язана з розробкою комплексної інформаційної системи для управління службою таксі з використанням Flutter, Vue.js, Express.js та PostgreSQL. Обрана тема є надзвичайно актуальною, оскільки сучасні сервіси мобільності вимагають високої швидкодії та надійності, а розробка такої системи охоплює одразу кілька ключових напрямів програмної інженерії: побудову кросплатформного мобільного інтерфейсу, розробку вебадмін-панелі, створення REST API, проєктування бази даних, роботу з авторизацією (Twilio), інтеграцію геоінформаційних систем (Google API) та хмарного збереження даних (AWS S3).

У межах роботи розглядається комплексна система, яка реалізує функції реєстрації та авторизації користувачів і водіїв, верифікації документів водіїв через адміністраторську панель, формування замовлення на поїздку із зазначенням точок маршруту на карті, розрахунку вартості та управління статусами поїздок. Мобільна клієнтська частина (для пасажира та водія) побудована як універсальний застосунок на Flutter, адміністраторська панель реалізована за допомогою Vue.js, серверна логіка написана на Express.js, а дані зберігаються у реляційній базі даних PostgreSQL.

Мета роботи – комплексна система управління таксі, планування архітектури клієнт-серверної частини, аналіз вимог цільової аудиторії,

структури даних та програмної реалізації, що передбачає аналіз предметної області, аналіз сучасного ринку, конкурентів, наявних сервісів для оптимізації навантаження на серверну частину та покращення досвіду користувачів.

За об’єкт дослідження взято систему таксі, що складається з універсального мобільного застосунку, призначеного як для пасажирів, так і для водіїв, засобами фреймворку Flutter. Система також включає в себе адміністраторську панель, яка забезпечує контроль над процесом реєстрації та верифікації водіїв, засобами Vue js. Базою даних було обрано PostgreSQL, так як є реляційною базою даних, що використовується для багаточисленного пласту застосунків і має значну кількість sdk, що спрощує інтеграцію з серверною частиною.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків та переліку посилань. У першому розділі розглянуто предметну область, вимоги та обрані технології. У другому розділі описано архітектуру системи, проєктування бази даних, REST API та інтерфейси користувача. У третьому розділі наведено реалізацію основних модулів, інтеграцію third-party сервісів, підхід до тестування та запуску застосунку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

Предметною областю даної системи є сфера пасажирських перевезень та організація взаємодії між пасажирами, водіями й адміністраторами за допомогою сучасних інформаційних технологій. У сучасних умовах цифровізації більшість сервісів таксі переходять у формат мобільних платформ, що дозволяє значно спростити процес виклику автомобіля, скоротити час очікування та підвищити рівень комфорту користувачів.

Основною задачею системи є автоматизація процесу створення, обробки та виконання замовлень на перевезення. Для цього необхідно забезпечити швидкий обмін інформацією між усіма учасниками системи: пасажиром, який створює замовлення, водієм, який його приймає та виконує, а також адміністратором, який контролює функціонування сервісу.

Особливістю предметної області є необхідність роботи системи в режимі реального часу. Це включає передачу геолокації водіїв, оновлення статусу замовлень, обмін повідомленнями та підтримку оперативного зв’язку між клієнтом і водієм. Для реалізації таких функцій необхідно використовувати серверну архітектуру, здатну обробляти велику кількість одночасних запитів та забезпечувати стабільне функціонування системи.

Важливим аспектом предметної області є також безпека та контроль користувачів. Зокрема, для водіїв необхідна процедура верифікації документів, що підтверджують право на здійснення перевезень. Це дозволяє підвищити рівень довіри до сервісу та забезпечити безпеку пасажирів. Для цього в системі передбачено адміністраторську панель, через яку здійснюється перевірка документів, блокування або розблокування користувачів та контроль діяльності системи.

1.2 Аналіз ринку предметної області

На сучасному ринку існує велика кількість сервісів для замовлення таксі, серед яких найбільш відомими є Uber [10], Bolt [11] та Uklon [12]. Дані платформи пропонують користувачам можливість швидкого виклику автомобіля через мобільний застосунок, відстеження маршруту на карті та безготівкову оплату поїздок.

Основною перевагою існуючих рішень є зручність використання та швидкість обробки замовлень. Більшість сервісів використовують мобільні застосунки з інтеграцією геолокації, системи рейтингу водіїв та пасажирів, а також механізми автоматичного підбору найближчого водія. Окрім цього, сучасні платформи забезпечують підтримку онлайн-карт, push-сповіщень та оновлення даних у режимі реального часу.

Проведений аналіз показав, що актуальним напрямком розвитку є створення системи, яка поєднуватиме простий та зрозумілий інтерфейс, швидку роботу мобільного застосунку та підвищений рівень контролю водіїв через процедуру верифікації документів. Особливу увагу необхідно приділити стабільності серверної частини, швидкому оновленню інформації та забезпеченню комфортної взаємодії користувачів із системою.

Отже, аналіз ринку та існуючих рішень дозволив визначити основні функціональні та технічні вимоги до майбутньої системи, що забезпечить конкурентоспроможність та високий рівень захисту персональних даних користувачів. Аналіз ринку також продемонстрував необхідність проєктування гнучкої до інтеграції нових функцій системи. Відповідно, комплексна система має відповідати вимогам сучасних та майбутніх технологій, таких як операційні системи, мобільні пристрої, а також нове серверне обладнання.

1.3 Обґрунтування вибору технологій

Бекенд-частина системи розроблена на базі платформи Node.js [2] із використанням мови програмування TypeScript, що забезпечує типізацію даних та підвищує надійність коду. В основі архітектури лежить фреймворк Express для обробки HTTP-запитів та бібліотека TypeORM [8] для взаємодії з базою даних через об’єктно-реляційне відображення. Для забезпечення реального часу у взаємодії між пасажиром та водієм (передача замовлень, оновлення статусів поїздки) інтегровано технологію WebSockets за допомогою Socket.io [6]. Безпека системи базується на використанні JWT (JSON Web Tokens) для авторизації запитів, а також впроваджено багаторівневу систему перевірки статусів користувачів, що дозволяє миттєво блокувати доступ до функцій системи у разі порушення правил сервісу.

У якості системи керування базами даних обрано PostgreSQL [5], що забезпечує високу доступність, масштабованість та відмовостійкість даних. Зберігання медіафайлів, таких як фото профілів та скан-копії документів водіїв для верифікації, реалізовано через хмарне сховище Amazon S3 [1]. Інтеграція з AWS дозволяє системі ефективно обробляти велику кількість одночасних підключень та забезпечує швидкий доступ до даних з мінімальними затримками. Така комбінація сучасних технологій та хмарних сервісів дозволяє створити стабільну, безпечну та продуктивну серверну інфраструктуру, готову до високих навантажень у реальних умовах експлуатації.

Вибір технологій для розробки серверної частини був зумовлений необхідністю створення високопродуктивної, масштабованої та безпечної системи, здатної обробляти велику кількість запитів у режимі реального часу. Використання середовища виконання Node.js разом із мовою TypeScript є стратегічним рішенням: Node.js базується на подійному циклі (Event Loop) та неблокуючому вводу-виводу, що ідеально підходить для I/O-інтенсивних додатків, таких як сервіси таксі, де одночасно підтримуються тисячі сокет-

з’єднань. TypeScript додає до цього рівняння статичну типізацію, що значно знижує ймовірність виникнення помилок на етапі розробки та полегшує підтримку великої кодової бази, роблячи архітектуру прозорою та передбачуваною. Фреймворк Express обраний завдяки своїй мінімалістичності та гнучкості, що дозволяє будувати модульну архітектуру без зайвого навантаження, забезпечуючи при цьому високу швидкість обробки HTTP-запитів.

Для забезпечення надійної взаємодії з даними була обрана реляційна СУБД PostgreSQL у поєднанні з TypeORM. PostgreSQL є однією з найстабільніших систем у світі з потужною підтримкою складних запитів та транзакцій, що критично важливо для фінансових операцій та обробки замовлень. TypeORM, у свою чергу, дозволяє абстрагуватися від низькорівневого SQL та працювати з даними як з об’єктами, що прискорює розробку та забезпечує автоматичну міграцію схем. Окрему увагу приділено технології Socket.io – вона стала фундаментом для реалізації real-time функціоналу (відстеження водія на карті, миттєве отримання замовлень). На відміну від стандартних вебсокетів, socket.io забезпечує автоматичне перепідключення та гарантовану доставку повідомлень. Безпека ж реалізована через JWT, що дозволяє будувати безстатеву (stateless) систему авторизації, яка легко масштабується горизонтально. Такий комплексний підхід до вибору технологій гарантує не лише швидку розробку прототипу, але й довгострокову життєздатність проєкту при зростанні навантаження.

Клієнтська частина системи представлена двома мобільними застосунками (для пасажира та водія), розробленими на базі фреймворку Flutter [3]. Вибір цієї технології зумовлений можливістю створення кросплатформного рішення з єдиною кодовою базою для iOS та Android без втрати продуктивності. Архітектура застосунків побудована на принципах реактивного програмування та розділення відповідальності. Взаємодія з бекендом реалізована через REST API для синхронних операцій (авторизація, історія, профілі) та через протокол WebSockets (socket.io) для миттєвих подій.

Ключовим компонентом є інтеграція з Google Maps SDK [4], що дозволяє будувати маршрути, візуалізувати рух автомобілів та виконувати просторовий пошук адрес.

Панель адміністрування розроблена як сучасний вебзастосунок (SPA) на базі фреймворку Vue.js [9] із використанням TypeScript та Tailwind CSS. Вона слугує центральним вузлом для моніторингу системи, верифікації документів водіїв та оперативного керування доступом користувачів. Архітектура панелі базується на компонентному підході, що дозволяє легко розширювати функціонал. Завдяки системі вкладок (Tabs) адміністратор може одночасно працювати з різними категоріями даних: заявками на реєстрацію, базою клієнтів та активним штатом водіїв.

Вибір Flutter як основного інструменту розробки мобільної частини дипломного проєкту обґрунтований його високою швидкістю рендерингу завдяки власному графічному движку Skia. У контексті сервісу таксі це критично важливо для плавної анімації маркерів на карті та миттєвого відгуку інтерфейсу. Використання мови Dart забезпечує безпеку типів та зручну роботу з асинхронністю (Future/Stream), що є фундаментом для обробки сокет-подій. На відміну від React Native, Flutter не потребує "моста" (bridge) для взаємодії з нативними компонентами, що підвищує загальну стабільність застосунку при тривалому використанні GPS та мережевих запитів.

Для розробки адмін-панелі було обрано Vue.js, оскільки цей фреймворк пропонує оптимальний баланс між продуктивністю та швидкістю розробки. Використання Composition API дозволяє створювати чисту та тестовану логіку керування станом, що важливо для інтерфейсів з великою кількістю таблиць та модальних вікон. Tailwind CSS обраний як основний інструмент стилізації через його utility-first підхід, що дозволяє швидко будувати складні адаптивні інтерфейси без написання громіздких CSS-файлів. Інтеграція з Axios забезпечує надійну обробку HTTP-запитів з автоматичним додаванням JWT-токена в заголовки. Такий стек дозволяє створити інструмент адміністрування, який є легким, швидким та зручним у підтримці, що повністю відповідає вимогам до сучасних корпоративних систем керування.

2 ПРОЄКТУВАННЯ СИСТЕМИ

Другий розділ присвячено проєктуванню системи, що включає в себе як клієнт, так і серверну частину. На цьому етапі визначено загальну архітектуру системи, структуру бази даних, основні API-маршрути та логіку взаємодії користувача з інтерфейсом. Проєктування є необхідним етапом перед описом програмної реалізації, оскільки воно показує, як окремі частини застосунку пов’язані між собою.

2.1 Моделювання системи та архітектурних рішень

Для забезпечення коректної роботи програмного комплексу та розуміння логіки взаємодії між усіма учасниками системи (пасажиром, водієм та сервером) необхідно провести детальне моделювання ключових бізнес-процесів. Функціональне моделювання дозволяє наочно відобразити послідовність виконання операцій, визначити відповідальність кожної організаційної одиниці та проаналізувати можливі розгалуження (сценарії) під час роботи застосунку. Як інструмент візуалізації було обрано нотацію EEPC (Extended Event-Driven Process Chain – розширена подійно-орієнтована ланцюжкова модель). Вибір цієї методології зумовлений її здатністю чітко розмежовувати стани системи (Події) та активні дії (Функції), а також можливістю відображення інформаційних потоків і виконавців для кожного етапу. Оскільки розроблений сервіс таксі є системою, що працює в режимі реального часу (real-time) та сильно залежить від зовнішніх тригерів (поведінка користувачів, зміна геоданих), подійно-орієнтована нотація є найбільш релевантною. Головним бізнес-процесом, що визначає цінність даного програмного продукту, є «Процес оформлення та виконання поїздки». Цей життєвий цикл охоплює всі етапи взаємодії: від виникнення потреби у

перевезенні та алгоритмічного розрахунку вартості сервером, до конкурентного прийняття замовлення водієм та фінального завершення поїздки. На рисунку 2.1 представлена розроблена EEPC-модель цього процесу.

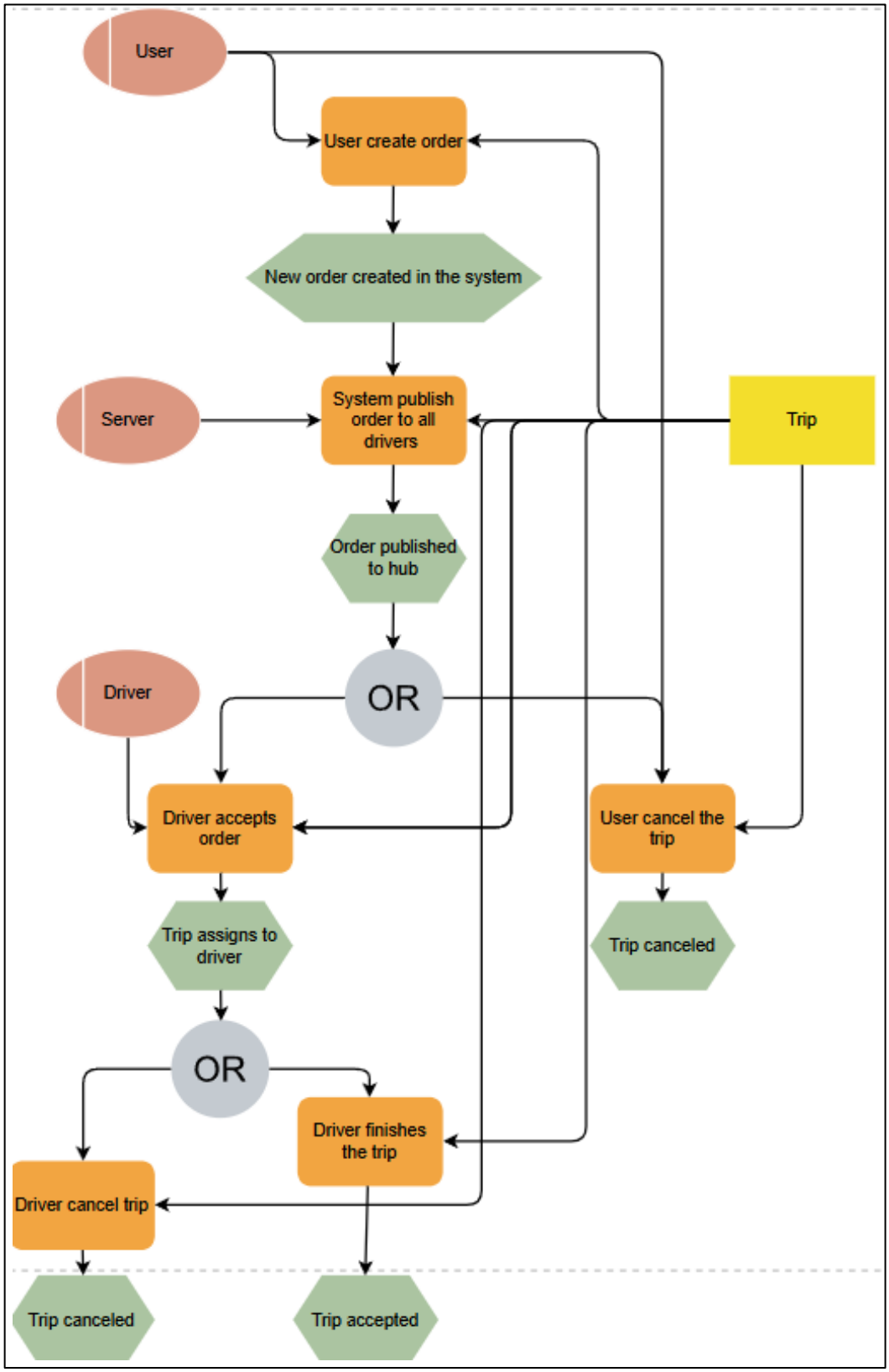


Рисунок 2.1 – EEPC-модель оформлення та завершення поїздки

Діаграма демонструє не лише лінійний хід успішного виконання замовлення, але й обробку альтернативних сценаріїв за допомогою логічних операторів (XOR), таких як скасування пошуку пасажиром або відмова від замовлення. Крім того, на схемі чітко розмежовано зони відповідальності: які саме функції виконуються на боці клієнтських мобільних застосунків, а які автоматизовані та виконуються центральним сервером (REST API та WebSocket-модулями).

Для забезпечення безпеки та контролю якості послуг перевезення, одним із найважливіших бізнес-процесів розробленої системи є реєстрація та верифікація водіїв. Цей процес є багатоетапним і вимагає чіткої взаємодії між мобільним застосунком клієнта, серверним ядром та вебпанеллю адміністратора. Для проєктування та візуалізації цієї взаємодії було обрано стандарт моделювання BPMN (Business Process Model and Notation). Вибір нотації BPMN зумовлений її здатністю розподіляти зони відповідальності за допомогою пулів (Pools) та доріжок (Lanes), що дозволяє наочно продемонструвати рух даних між фізично розділеними підсистемами.

На рисунку 2.2 представлена BPMN-діаграма процесу «Реєстрація, верифікація та авторизація водія».

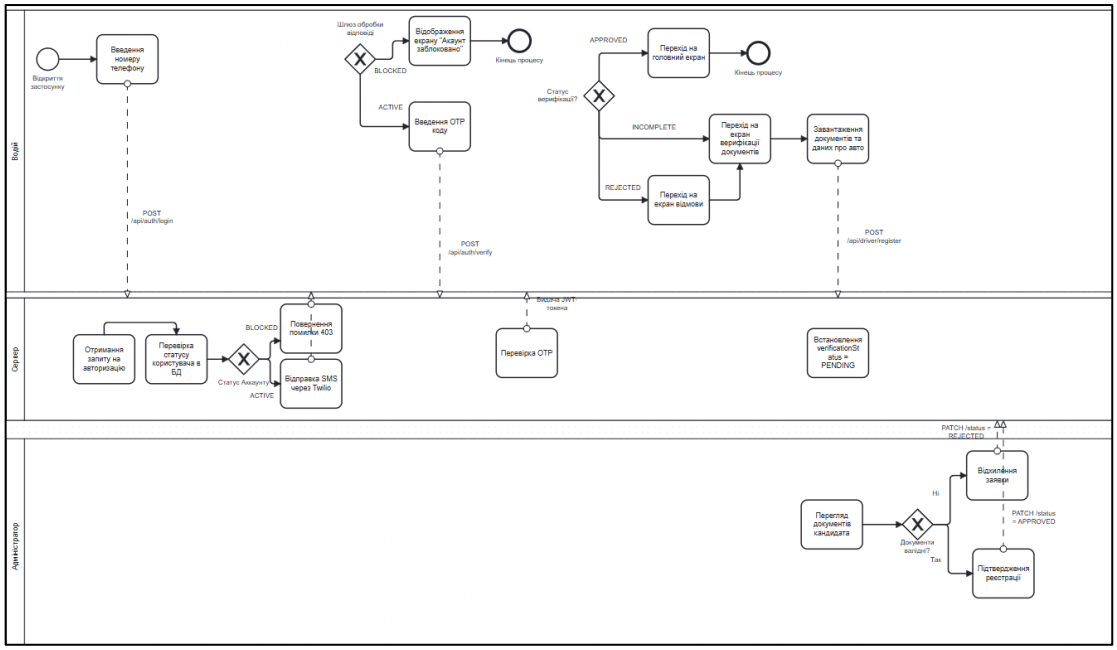


Рисунок 2.2 – Процес авторизації та реєстрації водія схемою BPMN 2.0

Розроблена модель детально описує життєвий цикл профілю водія: від первинної OTP-авторизації за номером телефону до завантаження цифрових копій документів (посвідчення водія, техпаспорт) та фінальної модерації. На схемі відображено обробку сервером різних станів профілю (наприклад, блокування доступу), а також роботу логічних шлюзів (Gateways), які визначають подальші дії адміністратора у разі виявлення невалідних документів. Побудова даної моделі дозволила спроєктувати надійний механізм контролю доступу (Middlewares) на рівні серверної архітектури.

Відповідно до наведеної діаграми на рисунку 2.3, варіантів використання, з боку пасажира (користувача) програмний застосунок має реалізовувати комплекс функціональних можливостей, що забезпечують повноцінну взаємодію із сервісом таксі. Насамперед система повинна містити механізм авторизації для безпечного доступу до персональних даних.

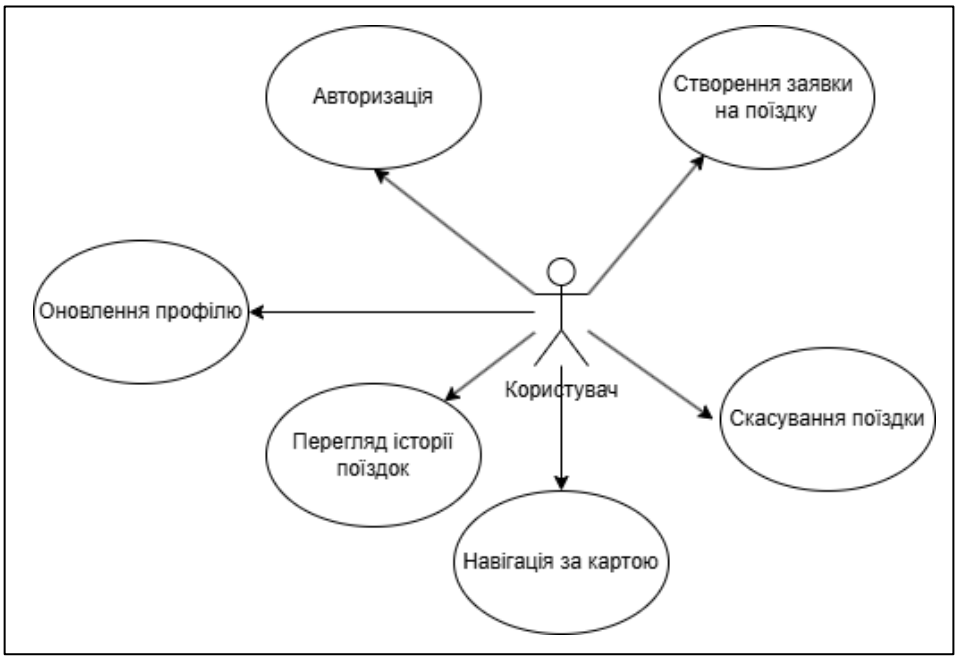


Рисунок 2.3 – Діаграма використання застосунку з боку користувача

Після успішного входу користувач повинен мати змогу керувати своїм обліковим записом через функцію оновлення профілю, а також переглядати інформацію про власну активність у розділі історії поїздок. Головний операційний процес реалізується через функціонал навігації за картою, який

слугує основою для ініціації ключової дії – створення заявки на поїздку. Крім того, архітектура програми має бути гнучкою та передбачати сценарій переривання цього процесу, надаючи користувачеві можливість скасування створеної поїздки у разі зміни обставин.

Найважливішою частиною клієнтського застосунку є проєктування загальної системи класів, яка буде відповідати цілям застосунку і буде мати графічний вигляд для зручного використання та швидкого орієнтування в класах та зв’язках об’єктів (див. рис. 2.4).

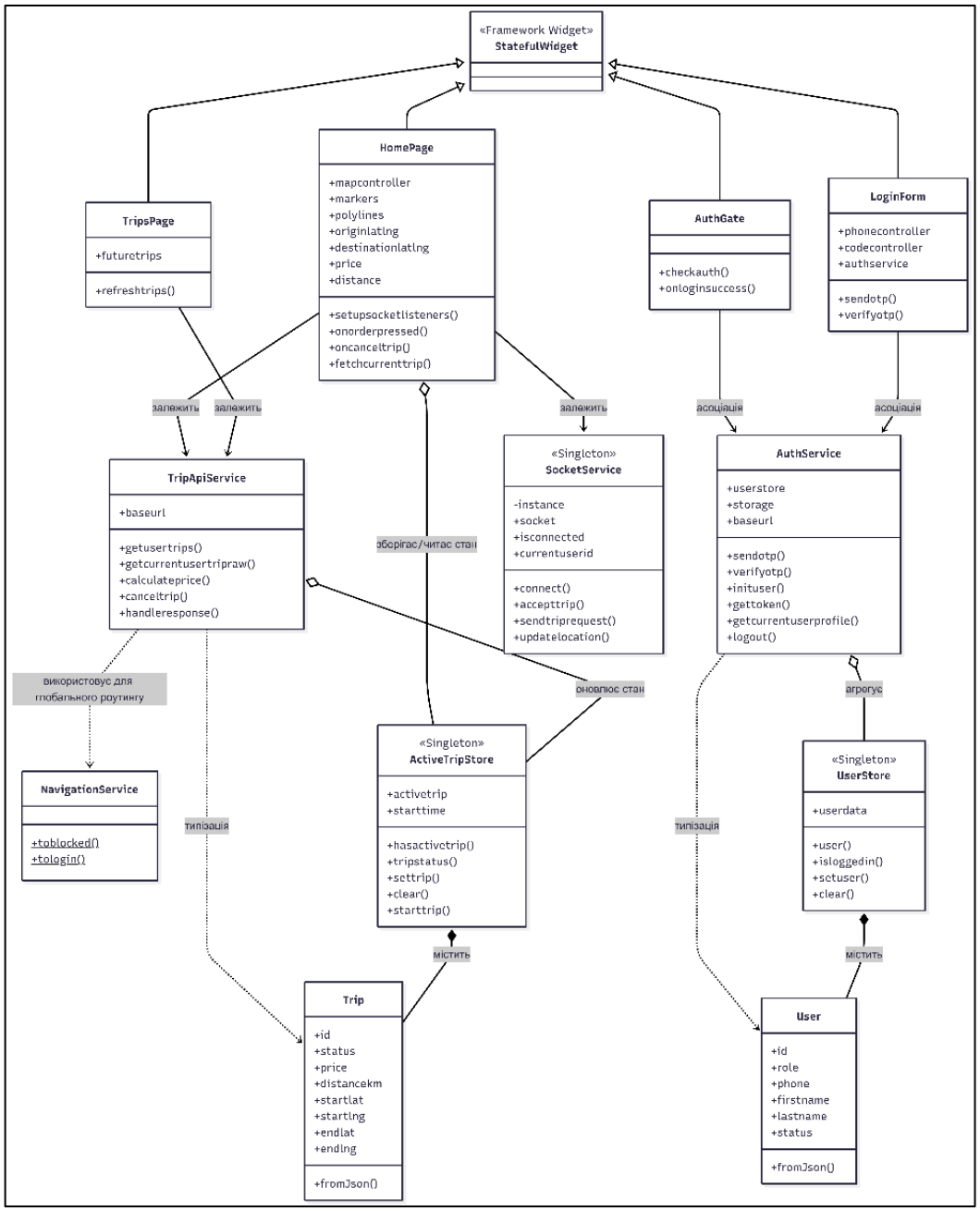


Рисунок 2.4 – Діаграма класів клієнтського застосунку

2.2 Проєктування бази даних

Проєктування інформаційного забезпечення системи базується на використанні реляційної моделі даних, що реалізована засобами системи керування базами даних PostgreSQL. Логічна структура бази даних складається з набору взаємопов'язаних сутностей, що забезпечують збереження інформації про користувачів, транспортні засоби та процеси перевезення.

Центральною ланкою архітектури є таблиця користувачів яка містить контактні дані, рольову модель, рейтингові показники та поточний статус доступу до системи.

Сутність водія розширює дані користувача та містить спеціалізовану інформацію про ліцензії та стан верифікації документів. Опис транспортних засобів винесено в окрему структуру що включає технічні характеристики, державні номери та тип комфорту автомобіля.

Облік поїздок реалізовано через таблицю замовлень яка зберігає просторові координати точок маршруту, розраховану вартість, дистанцію та часові мітки життєвого циклу замовлення.

Зв'язок між пасажиром та водієм фіксується через зовнішні ключі що забезпечує цілісність даних на рівні бази, комунікаційні процеси представлені сутностями чатів та викликів які мають пряму прив'язку до конкретного ідентифікатора поїздки, використання механізмів індексації за ідентифікаторами id та унікальними полями телефонів забезпечує високу швидкість виконання пошукових операцій та запобігає дублюванню записів.

Структура даних спроєктована з дотриманням вимог нормалізації що мінімізує надмірність інформації та спрощує підтримку цілісності системи при інтенсивних навантаженнях.

На рисунку 2.5 зображено графічне представлення бази даних у вигляді схеми, яка демонструє логічні зв'язки між сутностями. Вона складається з основних таблиць: users, cars, trips та drivers.

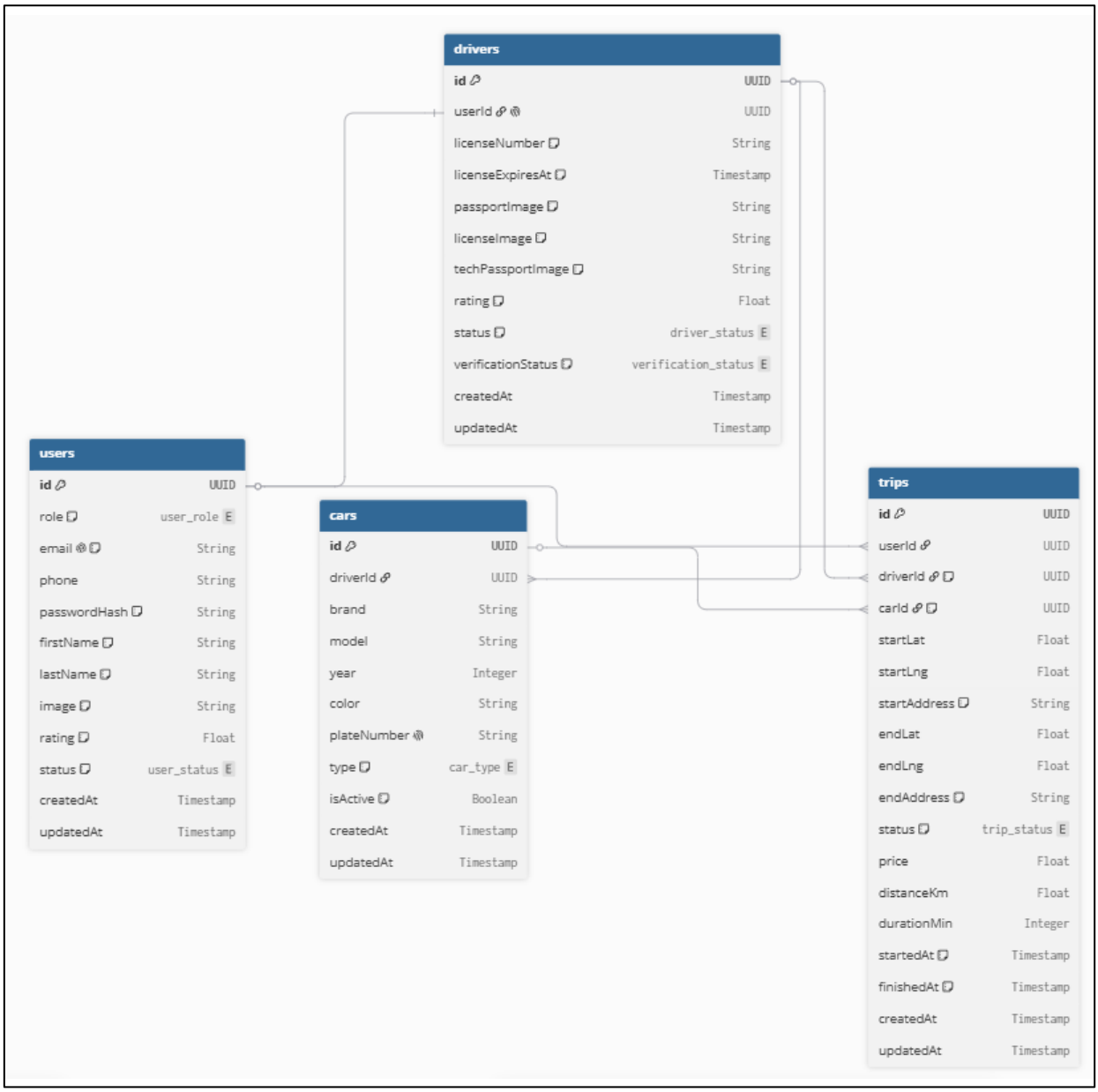


Рисунок 2.5 – Схема бази даних

Проєктування реляційної бази даних розпочинається з концептуального та логічного моделювання предметної області. Важливим етапом цього процесу є виділення та формування структурованого переліку сутностей, які відображають ключові об’єкти системи таксі: користувачів, водіїв, транспортні засоби та поїздки. Чітке визначення сутностей та їхніх атрибутів дозволяє провести нормалізацію даних, забезпечити їхню цілісність, уникнути дублювання інформації та закласти правильну основу для побудови зв’язків між таблицями. Структурований перелік базових сутностей бази даних розробленої системи наведено у таблиці 2.1.

Таблиця 2.1 – Основні таблиці бази даних застосунку

Таблиця	**Поля**	**Призначення**
users	Id, role, email, phone, passwordHash, firstName, lastName, image, rating, status, createdAt, updatedAt	Зберігання основної інформації про всіх суб’єктів системи (пасажирів, адміністраторів та водіїв), їх автентифікаційних даних, ролей та загального статусу доступу.
drivers	Id, userId, licenseNumber, licenseExpiresAt, passportImage, licenseImage, techPassportImage, rating, status, verificationStatus, createdAt, updatedAt	Розширення профілю користувача специфічними даними водія: інформацією про посвідчення, посиланнями на документи для верифікації та поточним робочим станом (на лінії/офлайн).
cars	id, driverId, brand, model, year, color, plateNumber, type, isActive, createdAt, updatedAt	Зберігання характеристик транспортних засобів, прив’язаних до водіїв, визначення класу комфорту авто (для розрахунку тарифу) та контролю його активного використання.
trips	id, userId, driverId, carId, startLat, startLng, startAddress, endLat, endLng, endAddress, status, price, distanceKm, durationMin, startedAt, finishedAt, createdAt, updatedAt	Фіксація повного життєвого циклу замовлення: від координат і адрес маршруту до прив’язки учасників (пасажир, водій, авто), розрахунку вартості, відстані та відстеження статусу поїздки.

Для забезпечення коректної роботи інформаційної системи спроєктовано реляційну базу даних, що складається з низки взаємопов’язаних таблиць. Нижче наведено детальний опис структури основних сутностей бази даних та їхніх атрибутів.

Основною таблицею для ідентифікації та автентифікації суб’єктів системи є таблиця users. Вона містить поле id, що виступає первинним ключем у форматі UUID, поле role, яке визначає системну роль особи (пасажир, водій або адміністратор), а також контактні дані: email та phone. З метою гарантування безпеки пароль користувача зберігається у вигляді криптографічного хешу в полі passwordHash. Особисті дані зберігаються у полях firstName та lastName, а посилання на фотографію профілю у хмарному сховищі (AWS S3) – у полі image. Поле rating фіксує середню оцінку користувача, тоді як status відображає поточний стан облікового запису (активний, заблокований або видалений). Поля createdAt та updatedAt виконують функцію системних часових міток.

Дані про надавачів послуг винесено в окрему таблицю drivers, що має зв’язок один-до-одного з таблицею користувачів через зовнішній ключ userId. Ця таблиця зберігає професійну інформацію водія, зокрема номер посвідчення (licenseNumber) та термін його дії (licenseExpiresAt). Для проходження процедури верифікації передбачено поля passportImage, licenseImage та techPassportImage, що містять посилання на завантажені скан-копії відповідних документів. Поле verificationStatus відображає поточний етап перевірки профілю адміністратором, status визначає робочий стан водія (на лінії, офлайн або у поїздці), а окреме поле rating містить оцінку якості надання послуг.

Інформація про автопарк системи консолідується в таблиці cars, яка пов’язана з профілем водія через зовнішній ключ driverId. Транспортний засіб описується атрибутами brand (марка), model (модель), year (рік випуску), color (колір) та plateNumber (унікальний державний реєстраційний номер). Для автоматизованого розрахунку вартості послуг передбачено поле type, що

визначає клас комфорту автомобіля (наприклад: економ, стандарт, бізнес). Логічне поле isActive слугує індикатором того, чи використовується дане авто водієм для виконання замовлень у поточний момент.

Життєвий цикл надання послуги перевезення повною мірою фіксується у таблиці trips. Вона виступає центральною сутністю, яка об’єднує замовника (userId), виконавця (driverId) та транспортний засіб (carId). Для забезпечення маршрутизації використовуються поля координат (startLat, startLng, endLat, endLng) та текстових адрес (startAddress, endAddress) початкової та кінцевої точок поїздки. Економічні та логістичні показники замовлення зберігаються у полях price (розрахована вартість), distanceKm (фізична відстань маршруту) та durationMin (тривалість). Поле status відображає актуальний етап виконання поїздки: від створення заявки до успішного завершення або скасування. Для точного хронометражу використовуються мітки startedAt (момент посадки пасажира) та finishedAt (момент завершення маршруту). Саме ця таблиця має ключову роль у функціонуванні системи.

2.3 Проєктування Rest API

REST API програмного застосунку реалізовано за клієнт-серверною архітектурою та призначено для забезпечення взаємодії між мобільними клієнтами та серверною частиною системи. API відповідає за авторизацію користувачів, обробку замовлень, управління поїздками, передачу інформації між водіями та пасажирами, а також взаємодію з базою даних.

Для передачі даних використовується формат JSON. Усі запити виконуються через протокол HTTP із використанням стандартних методів GET, POST, PUT та DELETE. Серверна частина реалізує перевірку валідності вхідних даних, контроль доступу та обробку помилок.

Система авторизації побудована на основі jwt-токенів. Після підтвердження номера телефону користувач отримує токен доступу, який

використовується для подальших захищених запитів до API. Токен передається в http-заголовку Authorization. На сервері виконується перевірка підпису токена, після чого визначається користувач та його роль у системі.

REST API містить окремі групи ендпоінтів для різних функціональних модулів застосунку. До них належать наступні модулі: авторизації користувачів, управління поїздками, модуль роботи з профілями водіїв, адміністративного керування, розрахунку вартості поїздок.

Процес авторизації складається з двох основних етапів. На першому етапі користувач надсилає номер телефону, після чого сервер генерує та відправляє OTP-код через зовнішній SMS-сервіс. На другому етапі виконується перевірка коду підтвердження та створення JWT-токена доступу.

API для роботи з поїздками забезпечує створення замовлень, пошук водія, прийняття або скасування поїздки, а також зміну статусу замовлення. Під час створення поїздки сервер виконує розрахунок приблизної дистанції та вартості маршруту на основі координат початкової та кінцевої точки.

Для забезпечення актуального обміну даними між клієнтами система додатково використовує технологію socket.io, яка дозволяє реалізувати механізм обміну подіями в режимі реального часу. Це використовується для миттєвого отримання нових замовлень водіями, оновлення статусу поїздок та синхронізації дій між учасниками системи.

Усі запити до захищених ресурсів проходять через middleware-перевірку. Middleware відповідає за перевірку JWT-токена, визначення ролі користувача, наприклад, чи є він адміном, та контроль статусу облікового запису, . У випадку блокування користувача сервер автоматично обмежує доступ до функціоналу системи незалежно від наявності дійсного токена.

У таблиці 2.2 надано стислий перелік стандартних http кодів, які повертає сервер у відповідь на операції пов’язані як з внутрішнім функціоналом, так і з third-party сервісами, такими як Amazon AWS S3, Twilio та Google Api. Коди розподілені на декілька категорій: успішні, помилка запиту та помилка з боку сервера.

Таблиця 2.2 – Список HTTP-кодів, які повертає сервер

Код	**Призначення**
200	Успішне виконання запиту
400	Помилка вхідних даних
401	Відсутність авторизації
403	Заборона доступу
404	Ресурс не знайдено
500	Внутрішня помилка сервера

У таблиці 2.3 наведено детальний структурований зміст Rest api, який має на меті максимально лаконічно пояснити та продемонструвати список та логіку серверної частини системи менеджменту служби таксі, що, як наведено у таблиці, включає в собі як модулі авторизації та верифікації, так і адміністрацію.

Таблиця 2.3 – Таблиця Rest Api запитів

Модуль	**Метод**	**Маршрут**	**Дані запиту**	**Результат**
Auth	POST	/api/auth/login	phone, role	Відправка OTP-коду на мобільний пристрій користувача згідно його вхідним даним.
Auth	POST	/api/auth/verify	phone, code, role	Створення нового користувача у системі та повернення його об’єкта.
Auth	GET	/api/auth/me	Auth token, JWT	Отримання об’єкту користувача згідно відповідному токену, який було використано в якості authorization.

Продовження таблиці 2.3

Модуль	**Метод**	**Маршрут**	**Дані запиту**	**Результат**
Trips	POST	/api/trips/calculate	startLat, startLng, endLat, endLng	Розрахунок суми вартості поїздки, згідно певним сталим значенням тарифу на сервері, яка повертається на клієнтський застосунок в якості відповіді.
Trips	GET	/api/trips/available	JWT	Отримання списку поточних доступних поїздок, які очікують на водіїв, мають статус, PENDING від користувачів які успішно верифікувались у системі.
Trips	POST	/api/trips/cancel/id	id, role	Відміна певної поїздки як з боку користувача-замовника, на будь-якому етапі поїздки, окрім випадку, коли водій вже почав поїздку так і з боку водія, коли він вже прив’язаний до поїздки.
Trips	GET	/api/trips/current/driver/:userId	userId	Отримання списку подорожей певного водія за його id, враховуючи інформацію про клієнтів, ціну подорожі, поточний статус та маршрут.

Продовження таблиці 2.3

Модуль	**Метод**	**Маршрут**	**Дані запиту**	**Результат**
Trips	GET	/api/trips/current/user/:userId	userId	Отримання списку поїздок користувача.
Profile	PUT	/api/users/:id	image, name, id	Оновлення профілю користувача, відповідно до наданої інформації у тілі запиту та повернення оновленого об’єкту користувача.
Profile	POST	/api/driver/register	Passport, license, tech Passport, transport	Відправка заявки на реєстрацію водія, що включає в себе паспорт особи, технічний паспорт транспорту, посвідчення водія відповідної категорії та об’єкт транспорту, що включає в себе певні параметри автомобіля.
Admin	GET	/api/admin/drivers/pending	JWT	Отримання списку водіїв, що очікують на верифікацію, та їх повної інформації(документи, номер телефону та автомобіль).
Admin	GET	/api/admin/users	Jwt	Отримання списку користувачів та максимальної інформації про них.

Продовження таблиці 2.3

Модуль	**Метод**	**Маршрут**	**Дані запиту**	**Результат**
Admin	PATCH	/api/admin/drivers/:id/status	Id, status	Оновлення статусу водія відповідно до його заявки. В оновлення статусу входе як верифікація подія, так і блокування, що призводе до скасування його заявки та тимчасового профілю з можливістю повторно подати документи на реєстрацію.
Admin	GET	/api/admin/drivers	Jwt	Отримання списку водіїв та максимальної інформації про них.
Admin	PATCH	/api/admin/users/:id/status	Id, status	Оновлення статусу користувачів, в які входять як пасажири, так і водії. Це призводить до блокування або розблокування профілю користувача і доступу до системи замовлень.

Спроєктований REST API є центральною частиною серверної архітектури програмного застосунку та забезпечує взаємодію між клієнтськими мобільними застосунками, адміністративною панеллю та сервером. API реалізовано відповідно до принципів REST-архітектури, що дозволяє забезпечити стандартизований підхід до передачі даних, масштабованість системи та простоту інтеграції між компонентами.

Усі запити між клієнтською та серверною частиною виконуються через HTTP-протокол із використанням JSON як основного формату обміну даними. Для реалізації захищеного доступу використовується JWT-автентифікація, що дозволяє підтримувати механізм авторизованих сесій без необхідності збереження стану на сервері.

REST API умовно поділено на декілька функціональних модулів, кожен із яких відповідає за окрему бізнес-логіку системи. Основними модулями є: автентифікації, подорожей, керуванням над профілями та адмін-панеллю.

Модуль автентифікації забезпечує процес входу користувачів до системи, перевірку особи через OTP-коди та підтримку захищеного доступу до функціоналу застосунку.

Ендпоінт POST /api/auth/login. Даний ендпоінт використовується для ініціалізації процесу входу або реєстрації користувача. Клієнтський застосунок передає номер телефону користувача та його роль у системі. Роль визначає тип облікового запису, з яким буде виконуватися подальша робота в системі. Це може бути звичайний пасажир або водій.

Після отримання запиту сервер виконує перевірку існування користувача в базі даних. Додатково виконується перевірка статусу облікового запису. Якщо користувач має статус блокування, сервер обмежує подальший доступ до системи.

У випадку успішної перевірки сервер генерує OTP-код підтвердження через інтеграцію із зовнішнім SMS-сервісом Twilio [7]. Код надсилається на мобільний номер користувача, після чого сервер повертає клієнту інформацію про успішність операції.

Основним призначенням цього ендпоінта є забезпечення безпечної аутентифікації користувача без необхідності використання паролів.

Ендпоінт POST /api/auth/verify Після отримання SMS-коду користувач надсилає його на сервер для перевірки. Сервер виконує валідацію OTP-коду та перевіряє його відповідність номеру телефону користувача. Якщо код підтвердження є правильним, сервер виконує пошук користувача в базі даних.

У випадку відсутності запису система автоматично створює нового користувача з базовими параметрами профілю. Після успішної авторизації генерується JWT-токен доступу. До токена записуються основні дані користувача, зокрема його унікальний ідентифікатор, номер телефону та роль у системі. Надалі цей токен використовується для виконання всіх захищених API-запитів. Крім токена, сервер повертає повний об’єкт користувача та додатковий статус верифікації водія. Це дозволяє клієнтському застосунку визначати доступність функціоналу для конкретного типу користувача.

Ендпоінт GET /api/auth/me Даний ендпоінт призначений для отримання актуальної інформації про авторизованого користувача. Клієнт передає JWT-токен у HTTP-заголовку Authorization, після чого сервер виконує його перевірку. Після успішної валідації токена сервер отримує інформацію про користувача з бази даних та повертає повний профіль, включаючи персональні дані, роль, статус облікового запису та додаткові параметри профілю. Використання цього ендпоінта дозволяє клієнтському застосунку підтримувати актуальний стан авторизації користувача та синхронізувати дані між сервером і мобільним застосунком.

Ендпоінт POST /api/trips/calculate Ендпоінт використовується для попереднього розрахунку параметрів майбутньої поїздки. Користувач передає координати початкової та кінцевої точки маршруту. На основі отриманої дистанції система виконує розрахунок вартості поїздки відповідно до внутрішньої тарифної моделі. Результатом роботи ендпоінта є орієнтовна дистанція та розрахована ціна замовлення.

Ендпоінт GET /api/trips/available Даний ендпоінт використовується водіями для отримання списку доступних замовлень. Сервер повертає всі активні поїздки, що мають статус очікування водія. Перед виконанням запиту виконується перевірка JWT-токена та ролі користувача. Доступ до цього функціоналу мають лише авторизовані водії. Кожне замовлення містить інформацію про маршрут, орієнтовну дистанцію, вартість поїздки та додаткові дані, необхідні для прийняття рішення водієм.

Ендпоінт POST /api/trips/cancel/:id Ендпоінт призначений для скасування активної поїздки. Скасування може бути ініційоване як пасажиром, так і водієм. Після отримання запиту сервер виконує пошук поїздки за її унікальним ідентифікатором. Далі перевіряється поточний статус замовлення та можливість його скасування. У випадку успішного виконання операції статус поїздки змінюється на скасований. Додатково система ініціює Socket.io-подію для миттєвої синхронізації стану між усіма учасниками поїздки. Це дозволяє в режимі реального часу оновити інтерфейс пасажира та водія без необхідності повторного виконання HTTP-запитів.

Ендпоінти отримання активних поїздок використовуються для отримання поточної активної поїздки користувача або водія. Сервер виконує пошук поїздок за ідентифікатором користувача та повертає замовлення, яке перебуває в активному статусі. Це дозволяє відновлювати поточний стан застосунку після перезапуску або втрати з’єднання.

Ендпоінт PUT /api/users/:id Даний ендпоінт використовується для оновлення профілю користувача. Користувач може змінювати власне ім’я, фотографію профілю та інші персональні дані. Для завантаження зображень використовується формат Multipart/Form-Data. Після отримання файлу сервер виконує його обробку та завантаження до хмарного файлового сховища Amazon S3. У результаті сервер повертає оновлений об’єкт користувача та посилання на нове зображення профілю.

Ендпоінт POST /api/driver/register Цей ендпоінт використовується для подання заявки на реєстрацію водія. Користувач завантажує набір документів, необхідних для проходження верифікації. До обов’язкових документів належать: паспорт, водійське посвідчення та технічний паспорт транспортного засобу. Після отримання документів сервер створює профіль водія та встановлює статус перевірки. Надалі інформація передається до адміністративної панелі для ручної модерації.

Ендпоінт GET /api/admin/drivers/pending Ендпоінт використовується адміністраторами для отримання списку водіїв, заявки яких очікують

перевірки. Сервер виконує перевірку адміністративної ролі користувача, після чого повертає список профілів зі статусом очікування.

Ендпоінт PATCH /api/admin/drivers/:id/status Після перевірки документів адміністратор може змінити статус водія. Система підтримує декілька станів, зокрема підтвердження або відхилення заявки. Зміна статусу автоматично впливає на доступність функціоналу водія в мобільному застосунку.

Ендпоінти керування користувачами дозволяють адміністраторам отримувати повний список користувачів системи, включаючи водіїв та пасажирів. Система надає доступ до основної інформації профілів, статусів облікових записів та даних транспортних засобів.

Ендпоінт PATCH /api/admin/users/:id/status Даний ендпоінт використовується для блокування або розблокування користувачів системи. Після зміни статусу сервер автоматично обмежує доступ користувача до API навіть у випадку наявності дійсного JWT-токена. Це забезпечується додатковою перевіркою статусу користувача під час кожного захищеного запиту.

2.4 Валідація REST API серверної частини системи

Для забезпечення стабільної та безпечної роботи серверної частини застосунку у системі реалізовано комплексний механізм валідації запитів, перевірки доступу та централізованої обробки помилок. Даний механізм дозволяє контролювати коректність вхідних даних, обмежувати несанкціонований доступ до ресурсів системи та забезпечувати контрольовану обробку виняткових ситуацій під час виконання REST API-запитів.

Серверна частина побудована на базі середовища Node.js із використанням фреймворку Express. Для реалізації перевірок застосовуються middleware-компоненти, які виконуються послідовно перед передачею запиту до основної бізнес-логіки контролерів.

Усі захищені маршрути REST API використовують middleware verifyToken, який відповідає за перевірку автентичності користувача та валідність JWT-токена.

Основним завданням даного middleware є: отримання токена із заголовка Authorization, перевірка цифрового підпису JWT, пошук користувача у базі даних, перевірка статусу облікового запису, передача даних користувача до наступних етапів обробки запиту, що детально продемонстровано на рисунку 2.2, на якому зображено логіку серверної частини, а саме middleware, який відповідає за верифікацію токенів, перед обробкою запитів, які надсилає клієнтський додаток, або адміністраторська панель.

2.5 Архітектура системи

Програмний комплекс побудований на засадах клієнт-серверної архітектури з використанням трирівневої моделі розподілу функцій: рівень представлення, рівень бізнес логіки та рівень керування даними.

Взаємодія між компонентами здійснюється за протоколами HTTP REST та WebSocket. Архітектура включає наступні функціональні блоки: клієнтський рівень, рівень бізнес логіки, рівень керування даними та модель обміну даними.

Клієнтський рівень відповідає за взаємодію з кінцевими користувачами та розділений на три окремі інтерфейси відповідно до ролей у системі: мобільний інтерфейс водія, мобільний інтерфейс пасажира та вебінтерфейс адміністратора.

Інтерфейс водія призначений для обробки вхідних заявок від користувачів, скасування замовлень, коли інтерфейс пасажира забезпечує формування запитів на перевезення та візуалізацію просторових даних. Інтерфейс адміністратора відповідає за моніторинг акаунтів користувачів, зміну статусу акаунтів та верифікацію водіїв у системі.

Рівень бізнес-логіки, тобто серверну частину, реалізовано за принципом внутрішньої багаторівневої архітектури. Через це гарантовано забезпечення інкапсуляція бізнес-правил.

Рівень керування даними відповідає за надійне зберігання, консистентність та швидкий доступ до інформації. Розділений на дві підсистеми: реляційна база даних та об’єктне сховище. Реляційна база даних використовується для зберігання реляційних структур даних. На цьому рівні декларується структура таблиць, типи даних та зв’язки між ними. Об’єктне сховище застосовується для безпечного та масштабованого зберігання бінарних медіа-файлів.

Модель обміну даними працює у синхронному та асинхронному режимах. Синхронний режим використовується для операцій, що не потребують миттєвого оновлення стану та виконуються за запитом клієнта. Асинхронний режим критично важливий для функцій реального часу. Через постійне з’єднання здійснюється безперервний обмін геоданими та миттєва зміна статусів поїздок. Маршрутизація повідомлень у цьому режимі реалізована через механізм віртуальних кімнат, які ідентифікуються за унікальним ідентифікатором користувача, що гарантує надсилання повідомлень виключно цільовим отримувачам.

На рисунку 2.6 наведено фрагмент коду серверної частини, що відповідає за авторизацію запитів.

Перевірка поділена на 2 етапи: автентифікація та авторизація з рольовою моделлю. Автентифікація відповідає за перевірку ідентичності користувача. Алгоритм вилучає токен із HTTP-заголовка Authorization, верифікує його криптографічний підпис та декодує унікальний ідентифікатор.

Важливим елементом безпеки є блокування доступу для акаунтів зі статусом BLOCKED. Рольова модель реалізує систему контролю доступу на основі ролей. Якщо атрибут ролі не відповідає значенню ADMIN, доступ до захищених запитів блокується на рівні маршрутизатора, що запобігає несанкціонованому виконанню адміністративних функцій.

```
export interface AuthRequest extends Request {
  user?: User;
}

export const verifyToken = async (
  req: AuthRequest,
  res: Response,
  next: NextFunction,
) => {
  const token = req.headers.authorization?.split(" ")[1];

  if (!token) {
    return res.status(401).json({ message: "No token" });
  }

  try {
    const decoded = jwt.verify(
      token,
      process.env.JWT_SECRET || "default_secret",
    ) as { userId: string };
    const userRepo = AppDataSource.getRepository(User);
    const user = await userRepo.findOne({ where: { id: decoded.userId } });
    if (!user) {
      return res.status(401).json({ message: "User not found" });
    }

    if (user.status === "BLOCKED") {
      return res.status(403).json({ message: "ACCOUNT_BLOCKED" });
    }

    req.user = user;
    next();
  } catch (err) {
    return res.status(401).json({ message: "Invalid or expired token" });
  }
};
```

Рисунок 2.6 – Фрагмент коду auth middleware серверу

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ КОМПОНЕНТІВ СИСТЕМИ

Третій розділ присвячено детальній програмній реалізації компонентів клієнт-серверного онлайн-застосунку для перегляду кіно. У розділі наведено обґрунтування структурних та поведінкових моделей системи, описано архітектурні рішення клієнтської та серверної частин, алгоритми критичних бізнес-процесів. Також представлено комплексний план функціонального тестування та інструкцію з розгортання проєкту.

3.1 Програмна реалізація клієнтської частини системи

У сучасній індустрії розробки програмного забезпечення клієнтська частина (frontend) відіграє визначальну роль у формуванні загального враження користувача від взаємодії з програмним продуктом. Від якості проєктування, стабільності роботи та ергономічності інтерфейсів залежить не лише рівень задоволеності кінцевого споживача, але й комерційний успіх проєкту загалом. З огляду на це, оцінювання якості розробленої клієнтської частини вимагає комплексного підходу, що базується на низці чітко визначених критеріїв, які охоплюють як візуальну, так і технічну складові системи.

Продуктивність є одним із найважливіших показників якості клієнтського застосунку, оскільки вона безпосередньо впливає на швидкість відгуку системи на дії користувача. У контексті мобільних застосунків для пасажирів та водіїв, розроблених за допомогою фреймворку Flutter, ключовим критерієм є забезпечення стабільної частоти кадрів (FPS) на рівні 60 кадрів на секунду. Використання власного графічного рушія Skia дозволяє Flutter обходити обмеження, притаманні гібридним технологіям (наприклад,

відсутність необхідності використання JavaScript-моста для доступу до нативних компонентів), що мінімізує затримки при рендерингу складних анімацій, таких як плавне переміщення маркерів автомобілів на карті. Оцінка продуктивності також включає аналіз споживання оперативної пам’яті та енергоефективності, що є критичним фактором для мобільних пристроїв, які тривалий час працюють із ввімкненим модулем геолокації.

Для вебпанелі адміністратора, побудованої на базі Vue.js, критерії продуктивності фокусуються на швидкості початкового завантаження сторінки (First Contentful Paint, Time to Interactive) та оптимізації розміру бандлу застосунку (bundle size). Застосування концепції Single Page Application (SPA) у поєднанні з реактивністю Vue.js дозволяє мінімізувати кількість запитів до сервера під час навігації між розділами адмін-панелі, забезпечуючи миттєве оновлення даних у таблицях без необхідності повного перезавантаження сторінки.

Враховуючи високу фрагментацію сучасного ринку мобільних та десктопних пристроїв, адаптивність інтерфейсу є фундаментальним критерієм якості. Система повинна коректно відображатися та функціонувати на екранах з різними співвідношеннями сторін та роздільною здатністю. Завдяки природі фреймворку Flutter, мобільні застосунки автоматично адаптуються під особливості операційних систем Android та iOS, дотримуючись при цьому відповідних гайдлайнів (Material Design для Android та Cupertino для iOS там, де це доцільно). Це забезпечує консистентний досвід користування незалежно від платформи.

Для адмін-панелі адаптивність реалізована за допомогою фреймворку Tailwind CSS, який базується на принципі utility-first та підтримує концепцію Mobile First. Це означає, що інтерфейс адміністратора залишається зручним та повнофункціональним як на широкоформатних моніторах десктопних комп’ютерів, так і на екранах планшетів чи мобільних телефонів, забезпечуючи адміністратору можливість оперативного реагування на критичні події в системі з будь-якого пристрою.

Невидимим для кінцевого користувача, але критично важливим для розробників критерієм є внутрішня якість кодової бази клієнтської частини. Оцінка за цим критерієм базується на дотриманні принципів SOLID, рівні зв'язності (coupling) та згуртованості (cohesion) компонентів. У проєкті застосовано суворе розділення логіки (наприклад, винесення мережевих запитів у відповідні класи Service, а управління станом – у класи Singleton). Такий підхід не лише спрощує процес пошуку та виправлення помилок (дебагінг), але й забезпечує високий ступінь масштабованості системи. У майбутньому розширення функціоналу (додавання нових способів оплати, розширення профілю) не вимагатиме глобального переписування існуючого коду, що значно знижує вартість підтримки програмного продукту.

Останнім, але не менш важливим критерієм є безпека клієнтських застосунків. Хоча основний рівень захисту реалізується на сервері, фронтенд повинен забезпечувати надійне зберігання конфіденційних даних. Оцінка безпеки включає перевірку методів зберігання токенів авторизації. Використання бібліотеки flutter_secure_storage у мобільних застосунках гарантує шифрування JWT-токенів на рівні операційної системи (Keychain для iOS та Keystore для Android), унеможливлюючи їх компрометацію навіть за умови несанкціонованого фізичного доступу до пристрою. Крім того, якісна реалізація перехоплення HTTP-помилок доступу (коди 401 та 403) з миттєвим примусовим розлогіненням користувача або перенаправленням на екран блокування є показником високого рівня зрілості системи безпеки клієнтської частини.

На рисунку 3.1 зображено графічне представлення інтерфейсу панелі адміністратора, яка слідує стилю лаконічності та переслідує ціль надати максимальний функціонал для маніпуляцій над даними профілів користувачів. До переліку взаємодій відноситься верифікація водіїв, редагування статусу блокування користувачів та водіїв. Основна мета цього компонента – надати адміністратору централізований та інтуїтивно зрозумілий інструментарій для управління профілями користувачів платформи.

Функціонал панелі охоплює критичні бізнес-процеси: верифікацію наданих водіями документів, моніторинг активності в системі, а також оперативне управління статусами доступу, зокрема, блокування чи розблокування облікових записів водіїв та пасажирів.

Використання фреймворку Tailwind CSS забезпечило зручну табличну візуалізацію даних та швидку реакцію елементів керування на дії адміністратора.



Рисунок 3.1 – Інтерфейс панелі адміністратора

На рисунку 3.2 зображено екран авторизації логіну мобільного застосунку.

Дизайн цієї сторінки виконано у лаконічному стилі clean design, де візуальний акцент зроблено виключно на ключових елементах взаємодії – полях введення облікових даних та кнопці підтвердження. Такий підхід не лише сприяє швидкій орієнтації користувача, покращуючи загальний користувацький досвід – UX, але й технічно оптимізує роботу додатка: відсутність важких графічних елементів мінімізує час початкового рендерингу екрана та знижує навантаження на ресурси мобільного пристрою. Саме на цьому етапі відбувається первинна валідація даних форми перед їх відправкою на сервер для отримання JWT-токена.

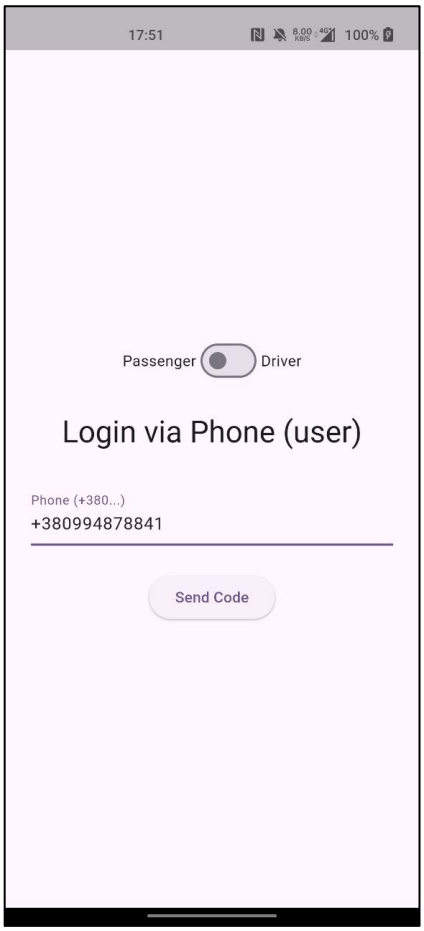


Рисунок 3.2 – Сторінка логіну застосунку

На рисунку 3.3 зображено головну сторінку застосунку, на якій відбувається взаємодія з системою та серверною частиною. Для покращення орієнтування у інтерфейсі було прийнято рішення максимально спростити дизайн та використати сторонні сервіси, як видно з карти google, яка є розповсюдженим варіантом у багатьох застосунках і маю глибоку інтеграцію з великою кількістю фреймворків, в тому числі і Flutter.

На рисунку 3.4 зображено деталізований інтерфейс керування профілями користувачів в адміністраторській панелі. Цей модуль надає уповноваженому персоналу розширений інструментарій для моніторингу активності конкретного суб’єкта системи. Зокрема, у разі виявлення шахрайських дій, скарг або порушення умов використання сервісу, адміністратор може ініціювати процедуру примусового обмеження прав доступу, змінивши статус акаунта на BLOCKED відповідною кнопкою в інтерфейсі.

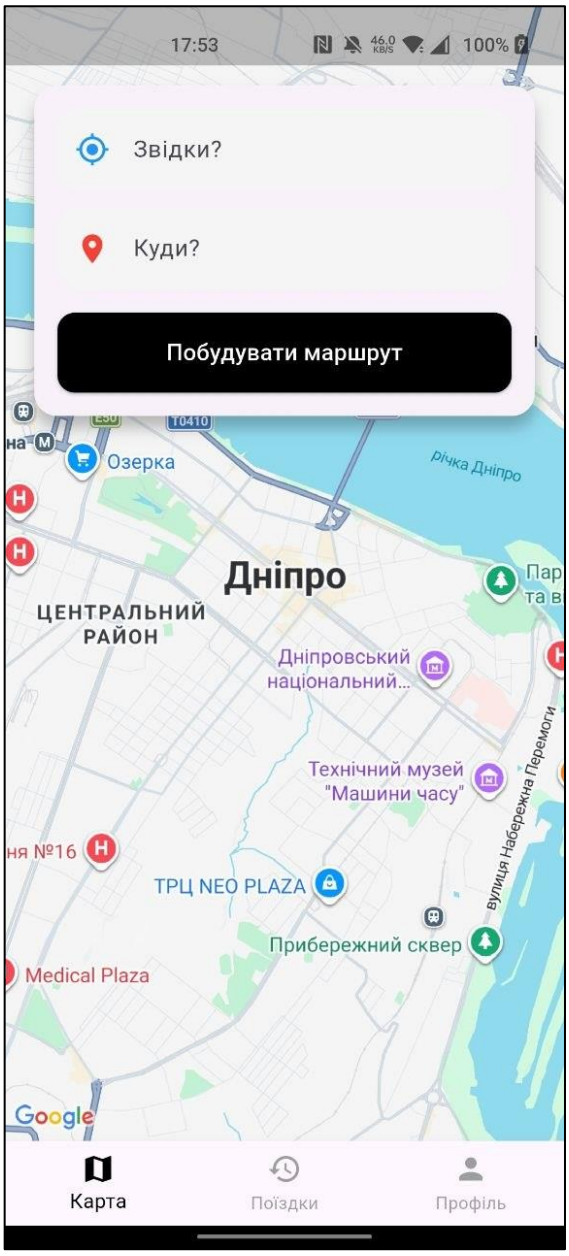


Рисунок 3.3 – Головна сторінка застосунку

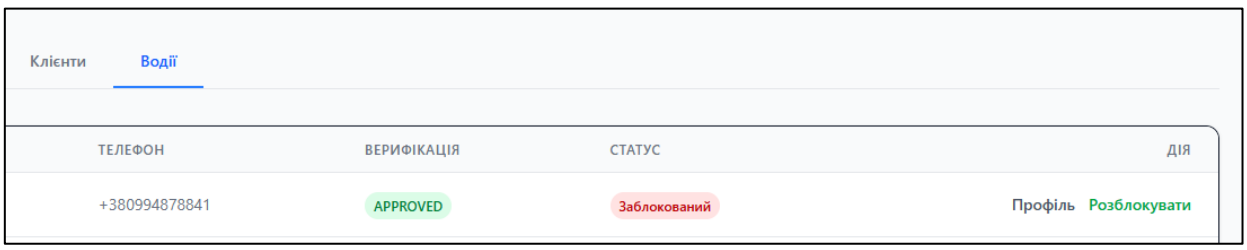


Рисунок 3.4 – Графічний інтерфейс адміністраторської панелі

Результатом блокуванням користувача є рисунок 3.5, на якому зображено блокування користувача, яке забороняє доступ до системи. Як тільки адміністратор застосовує блокування, сервер починає відхиляти всі нові

запити від цього користувача. Цей екран повністю ізолює користувача від основного функціоналу застосунку.

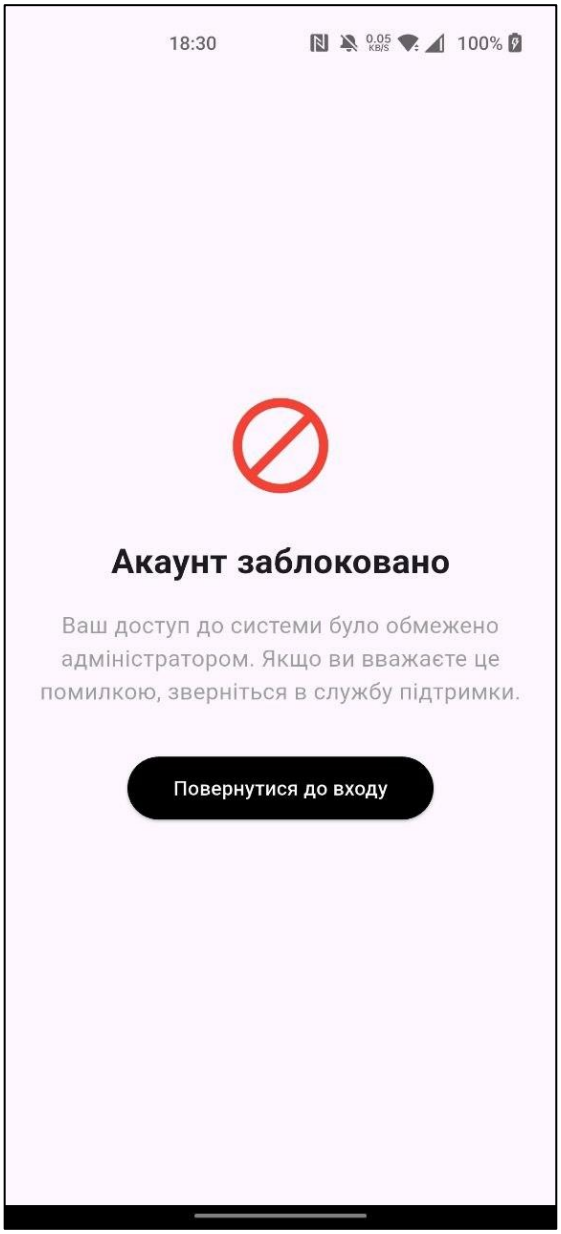


Рисунок 3.5 – Графічне зображення результату обробки клієнтського застосунку отримання заблокованого користувача

Важливою частиною у покращенні користувацього досвіду під час використання клієнтського застосунку є AuthGate, який створений для автоматичної автентифікації у випадку, коли користувач нещодавно вже був верифікований. В такому випадку сесія користувача разом з поточними даними буде збережено. Фрагмент цього коду зображено на рисунку 3.6.

```
class AuthGate extends StatefulWidget {
  const AuthGate({super.key});

  @override
  State<AuthGate> createState() => _AuthGateState();
}

class _AuthGateState extends State<AuthGate> {
  bool _isLoading = true;
  bool _isLoggedIn = false;

  @override
  void initState() {
    super.initState();
    _checkAuth();
  }

  Future<void> _checkAuth() async {
    final authService = AuthService();
    try {
      final isLoggedIn = await authService.initUser();

      setState(() {
        _isLoggedIn = isLoggedIn;
        _isLoading = false;
      });
    } catch (e) {
      if (e.toString().contains("ACCOUNT_BLOCKED")) {
        if (mounted) {
          Navigator.pushReplacementNamed(context, '/blocked');
        }
      } else {
        setState(() {
          _isLoggedIn = false;
          _isLoading = false;
```

Рисунок 3.6 – Фрагмент коду auth gate

3.2 Проведення тестів системи

Забезпечення високого рівня надійності, відмовостійкості та безпеки програмного комплексу вимагає проведення ретельного тестування на всіх етапах життєвого циклу розробки. Враховуючи клієнт-серверний характер системи, використання дуплексних каналів зв’язку (WebSocket) для обміну

геоданими в реальному часі та наявність фінансової бізнес-логіки, процес тестування було розділено на кілька фундаментальних рівнів.

Метою функціонального тестування є перевірка відповідності розробленого програмного забезпечення початковим вимогам та коректності виконання закладених бізнес-процесів. Основна увага на цьому етапі приділяється життєвому циклу замовлення поїздки: сценарії пасажира, сценарії водія та сценарії адміністратора.

До сценаріїв пасажира відноситься перевірка коректності визначення початкових координат, формування маршруту до точки призначення, розрахунку попередньої вартості поїздки залежно від дистанції та обраного тарифу, а також можливості скасування замовлення.

До сценаріїв водія можна віднести тестування механізму отримання вхідних заявок (push-сповіщень або подій через сокети), прийняття та відхилення замовлення, зміни статусів поїздки («Водій в дорозі», «Очікування пасажира», «Поїздка розпочата», «Поїздка завершена»).

До сценаріїв адміністратора відноситься перевірка функцій блокування та розблокування облікових записів, що повинно миттєво відбиватися на клієнтському застосунку, як показано на рис. 3.5, верифікації документів водіїв та відображення актуальної статистики системи.

Оскільки система складається з трьох незалежних клієнтських інтерфейсів та єдиного серверного ядра, критично важливою є перевірка їхньої коректної взаємодії за протоколами HTTP та WebSocket.

За допомогою інструментів тестування, наприклад, Postman або автоматизованих тестів на базі Jest/Supertest, перевірено всі ендпоінти. Особлива увага приділялася валідації вхідних даних: система повинна повертати відповідні HTTP-коди (400 Bad Request) у разі передачі некоректних параметрів (наприклад, невірний формат email або відсутність обов’язкових координат).

Тестування асинхронного обміну даними охоплювало перевірку механізму віртуальних кімнат. Перевірялося, чи отримує конкретний пасажир

геодані виключно того водія, який прийняв його замовлення, а також тестувалася поведінка системи при розриві з’єднання, наприклад, при втраті мобільної мережі, та механізм автоматичного перепідключення з відновленням останнього стану state recovery.

Зважаючи на наявність персональних даних користувачів, система пройшла сувору перевірку механізмів автентифікації та рольової моделі доступу. Перевірено роботу проміжного програмного забезпечення перевірки JWT-токенів. Було зімітовано запити з простроченими, підробленими та відсутніми токенами – система успішно блокує такі запити, повертаючи код 401 unauthorized. Перевірено розділення прав: спроби виконання адміністративних дій (наприклад, виклик ендпоінту блокування користувача) від імені звичайного пасажира або водія відхиляються з кодом 403 forbidden. На стороні мобільного клієнта протестовано механізм локального захисту сесії та безпечного зберігання токенів за допомогою засобів ОС пристрою. Перевірено примусове завершення сесії користувача у разі зміни його статусу адміністратором на BLOCKED.

Сервіси виклику таксі характеризуються піковими навантаженнями у певні години або за несприятливих погодних умов. Для оцінки стійкості серверної частини було проведено симуляцію одночасних підключень. Перевірялася здатність WebSocket-сервера обробляти сотні паралельних потоків трансляції геокоординат без критичних затримок та витоків пам’яті. Також тестувалася швидкість виконання транзакцій базою даних під час одночасного створення великої кількості запитів на поїздку. На стороні мобільного застосунку перевірялася стабільність частоти кадрів графічного рушія при відображенні карти з великою кількістю активних об’єктів.

Останнім етапом стало ручне та автоматизоване тестування адаптивності та зручності інтерфейсів.

Для вебпанелі адміністратора перевірено коректність верстки на різних роздільних здатностях моніторів та в різних браузерах (Cross-browser compatibility).

Для мобільних застосунків на базі Flutter було проведено тестування на фізичних пристроях з різними діагоналями екранів (смартфони та планшети) під управлінням як Android, так і iOS. Перевірялося коректне відображення клавіатури, робота жестів, уникнення перекриття важливих елементів системними панелями («чубчиками» екранів) та плавність анімацій переходів між екранами.

Завдяки комплексному підходу до тестування було виявлено та усунуто потенційні вразливості, оптимізовано алгоритми обробки даних та забезпечено стабільну роботу програмного комплексу в умовах, наближених до реальної експлуатації.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальне науково-практичне завдання розробки комплексної інформаційної системи для управління службою таксі. Мета роботи, яка полягала у створенні надійної, масштабованої та зручної програмної платформи для взаємодії пасажирів, водіїв та адміністраторів сервісу, успішно досягнута. У процесі виконання роботи було отримано такі теоретичні та практичні результати.

Проведено детальний системний аналіз предметної області та бізнес-процесів у сфері пасажирських перевезень. На основі аналізу сформовано вичерпні функціональні та нефункціональні вимоги до програмного забезпечення. Науково та практично обґрунтовано вибір сучасного технологічного стека: фреймворк Flutter для мобільного клієнта, Vue.js для розробки адміністративної вебпанелі, Express.js (Node.js) для побудови серверної логіки та СУБД PostgreSQL для безпечного збереження даних.

Спроєктовано та реалізовано нормалізовану реляційну базу даних, яка забезпечує ефективне зберігання та обробку інформації про профілі користувачів, транспортні засоби, історію поїздок, а також логування комунікації (чати та дзвінки). Продумана структура бази даних гарантує цілісність інформації та високу швидкість виконання запитів під навантаженням.

Розроблено серверну частину системи (API) за принципами REST-архітектури. Успішно реалізовано алгоритми математичного розрахунку відстані між географічними координатами (за формулою Гаверсину) та динамічного формування вартості поїздки. Для забезпечення миттєвої реакції системи на дії користувачів та синхронізації станів (зокрема під час пошуку вільного водія та відстеження його переміщення) інтегровано протокол WebSocket, що дозволило реалізувати повноцінну взаємодію в режимі реального часу.

Створено кросплатформний мобільний застосунок засобами Flutter, який поєднує в собі інтерфейси для двох ключових ролей: пасажира та водія. Реалізовано зручний інтерфейс користувача з підтримкою інтерактивних карт завдяки інтеграції Google Maps API. Додаток забезпечує повний цикл обслуговування: від авторизації та вказування точок маршруту до відстеження автомобіля, внутрішньої комунікації та завершення поїздки.

Спроєктовано та впроваджено адміністративну панель на базі Vue.js. Даний модуль надає інструментарій для управління базою клієнтів, моніторингу активних замовлень та проведення суворої верифікації водіїв. Для безпечного та децентралізованого зберігання конфіденційних медіафайлів, зокрема скан-копій водійських посвідчень та технічних паспортів, налаштовано інтеграцію з хмарним сховищем AWS S3.

Забезпечено високий рівень безпеки даних. Процес автентифікації побудовано на базі JWT-токенів з попереднім підтвердженням номера телефону через SMS-коди за допомогою сервісу Twilio. Паролі та чутливі дані зберігаються виключно у вигляді криптографічних хешів, а доступ до серверних ресурсів захищено багаторівневим проміжним програмним забезпеченням.

Проведено комплексне тестування розробленої системи, яке підтвердило коректність роботи всіх алгоритмів, стабільність клієнт-серверної взаємодії та відповідність кінцевого продукту початковим вимогам.

Практичне значення одержаних результатів полягає у створенні повністю функціонального, готового до розгортання програмного комплексу.. Подальший розвиток проєкту може передбачати впровадження алгоритмів машинного навчання для динамічного ціноутворення на основі попиту, підключення додаткових платіжних шлюзів (наприклад, Stripe або LiqPay) для безготівкової оплати, а також розширення аналітичного функціоналу адміністративної панелі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Amazon Simple Storage Service (Amazon S3) Documentation. *AWS Documentation*. URL: https://docs.aws.amazon.com/s3/ (дата звернення: 07.01.2026).

2. Express – Node.js web application framework. *Express*. URL: https://expressjs.com/ (дата звернення: 18.05.2026).

3. Flutter documentation. *Flutter*. URL: https://docs.flutter.dev/ (дата звернення: 07.01.2026).

4. Google Maps Platform Documentation. *Google Developers*. URL: https://developers.google.com/maps/documentation (дата звернення: 07.01.2026).

5. PostgreSQL: Documentation. *PostgreSQL*. URL: https://www.postgresql.org/docs/ (дата звернення: 08.01.2026).

6. Socket.IO Documentation. *Socket.IO*. URL: https://socket.io/docs/v4/ (дата звернення: 08.01.2026).

7. Twilio Docs. *Twilio*. URL: https://www.twilio.com/docs (дата звернення: 08.01.2026).

8. TypeORM – Amazing ORM for TypeScript and JavaScript. *TypeORM*. URL: https://typeorm.io/ (дата звернення: 09.01.2026).

9. Vue.js – The Progressive JavaScript Framework. *Vue.js*. URL: https://vuejs.org/guide/introduction.html (дата звернення: 09.01.2026).

10. Uber. *About Uber*. URL: https://www.uber.com/us/en/about/ (дата звернення: 10.01.2026).

11. Bolt. *Bolt Business Model and Services*. URL: https://bolt.eu/uk-ua/company/ (дата звернення: 10.01.2026).

12. Uklon. *About Uklon*. URL: https://uklon.com.ua/about-uklon (дата звернення: 10.01.2026).