Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Комп'ютерних наук і технологій
(повне найменування факультету)

Комп'ютерні системи та мережі
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалаврський
(ступінь вищої освіти)

на тему: РОЗРОБКА ІНТЕРАКТИВНОЇ СИСТЕМИ ДИСПЕТЧЕРА МІСЬКОГО ТРАНСПОРТУ

Виконав(ла): студент(ка) 4 курсу,
групи КНТз-512

Спеціальності
123 Комп'ютерна інженерія
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерна інженерія
(назва освітньої програми (спеціалізації)

ІСТОМІН К.І.
(ПРІЗВИЩЕ та ініціали)

Керівник ЗЕЛЕНЬОВА І.Я.
(ПРІЗВИЩЕ та ініціали)
Рецензент ХАРИТОНОВ О. Б.
(ПРІЗВИЩЕ та ініціали)

2026

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
Кафедра комп'ютерних систем та мереж
Ступінь вищої освіти бакалаврський
Спеціальність 123 комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) комп'ютерна інженерія
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ
Завідувач кафедри КУДЕРМЕТОВ Р.К.
_____
«_____»_____2026 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ІСТОМІНА Костянтина Ігоровича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка інтерактивної системи диспетчера міського транспорту
керівник проєкту (роботи) к.т.н., доцент, ЗЕЛЕНЬОВА І.Я.,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від « 14 » квітня 2026 року № 160
2. Строк подання студентом проєкту (роботи) 01.06.2026 р.
3. Вихідні дані до проєкту (роботи) опис предметної області, технології розробки клієнтської та серверної частини

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) Аналіз технологій розробки клієнт-серверних систем;
2) Розробка серверної частини системи;
3) Розробка клієнтської частини системи;
4) Випробування системи.

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

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	ЗЕЛЕНЬОВА І.Я.		
Нормоконтроль	ДЬЯЧУК Т.С.		

7. Дата видачі завдання «14» квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	**Примітка**
1	Аналіз предметної області та аналогів	до 18.04.2026	
2	Визначення та аналіз вимог до системи	до 21.04.2026	
3	Проєктування бази даних системи	до 25.04.2026	
4	Проєктування архітектури сервера	до 28.04.2026	
5	Розробка серверної частини	до 10.05.2026	
6	Проєктування клієнтської частини	до 14.05.2026	
7	Розробка користувацького інтерфейсу	до 22.05.2026	
8	Випробування комп’ютерної системи	до 24.05.2026	
9	Оформлення пояснювальної записки	до 26.05.2026	
10	Проходження нормоконтролю	до 30.05.2026	
11	Перевірка на наявність академічного плагіату	до 01.06.2026	
12	Проходження рецензування	до 02.06.2026	

Студент(ка) _____ Костянтин ІСТОМІН
(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Ірина ЗЕЛЕНЬОВА
(підпис) (Ім’я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 84 с., 37 рис., 5 табл., 3 дод., 17 джерел.

JAVA, JAVAFX, POSTGRESQL, ДИСПЕТЧЕР, МАРШРУТ, ТРАНСПОРТНИЙ ЗАСІБ

Предмет розробки – методи та програмні засоби автоматизації процесів диспетчеризації міського транспорту на основі клієнт-серверних технологій.

Мета роботи – підвищення ефективності роботи диспетчера міського транспорту та спрощення виконання диспетчерських операцій.

Новизна роботи полягає у програмній реалізації інтерактивної системи диспетчера міського транспорту на основі клієнт-серверної архітектури для централізованого керування транспортними засобами, маршрутами та диспетчерськими операціями.

Практична цінність роботи полягає у можливості використання розробленої системи для спрощення роботи диспетчера шляхом централізованого керування транспортом і маршрутами та автоматизації основних операцій обліку.

В межах роботи виконано аналіз предметної області диспетчеризації міського транспорту, сформульовано вимоги до програмної системи, обґрунтовано вибір клієнт-серверної архітектури та програмних засобів реалізації. Спроєктовано структуру бази даних, реалізовано серверну частину системи мовою Java з використанням PostgreSQL, а також клієнтський застосунок на основі JavaFX. Проведено тестування основних функцій системи та аналіз її працездатності. Проведено випробування програмного забезпечення, які підтвердили коректність роботи основних функціональних модулів системи, правильність мережевої взаємодії між клієнтом і сервером та коректність відображення даних у графічному інтерфейсі користувача.

ЗМІСТ

ВСТУП

Міський транспорт є одним з ключових елементів інфраструктури сучасного міста. Від ефективності його роботи залежить організація пересування населення, дотримання розкладу та загальна якість транспортного обслуговування всього міста. Координація роботи транспортних засобів, контроль маршрутів і оперативне реагування на зміни у процесі перевезень традиційно виконуються диспетчерськими службами. Разом із тим використання паперового обліку або великої кількості розрізнених програмних засобів ускладнює виконання цих завдань та збільшує ймовірність виникнення помилок [1].

Сучасні інформаційні технології дозволяють автоматизувати значну частину диспетчерських процесів [2]. Використання програмних систем дає можливість централізовано зберігати інформацію про транспорт, маршрути та виконані рейси, швидко отримувати необхідні дані та спрощувати виконання типових диспетчерських операцій. Особливо актуальним є застосування клієнт–серверної архітектури, яка забезпечує розділення функцій обробки даних і користувацького інтерфейсу, підтримує масштабованість та спрощує подальший розвиток програмного продукту.

На практиці диспетчеру необхідно одночасно працювати з декількома категоріями даних: транспортними засобами, маршрутами, поточним станом виконання рейсів та службовою інформацією. У разі відсутності єдиної системи ці процеси часто виконуються окремо, що ускладнює керування та знижує ефективність роботи [1, 2].

Тому створення програмного засобу, який об’єднує основні функції диспетчеризації в одному середовищі, є доцільним і практично значущим завданням.

В межах даної роботи пропонується розробка інтерактивної системи диспетчера міського транспорту, орієнтованої на автоматизацію обліку

транспортних засобів, керування маршрутами та підтримку роботи диспетчера через зручний користувацький інтерфейс.

Реалізація системи передбачається у вигляді клієнт–серверного застосунку з використанням мови програмування Java, графічного інтерфейсу JavaFX та системи керування базами даних PostgreSQL [3, 4].

Для досягнення поставленої мети необхідно вирішити такі завдання:

– проаналізувати особливості організації роботи диспетчерських служб міського транспорту;

– дослідити існуючі програмні рішення у сфері диспетчеризації;

– спроєктувати структуру програмної системи та модель даних;

– реалізувати серверну частину застосунку;

– розробити клієнтський інтерфейс користувача;

– створити базу даних для зберігання інформації;

– провести тестування реалізованої системи та оцінити результати її роботи.

1 АНАЛІЗ ТЕХНОЛОГІЙ РОЗРОБКИ КЛІЄНТ-СЕРВЕРНИХ СИСТЕМ

1.1 Призначення та функціональні вимоги до системи

Організація роботи міського транспорту передбачає постійну взаємодію між диспетчерською службою, транспортними засобами та маршрутною мережею. Навіть у випадках, коли не використовується автоматичне позиціонування транспорту або системи навігації, диспетчеру необхідно забезпечувати актуальність інформації про наявний транспорт, контролювати його закріплення за маршрутами та вести облік виконання рейсів. При збільшенні кількості транспортних одиниць виконання таких операцій вручну ускладнюється та потребує додаткових часових витрат [1].

Одним із підходів до підвищення ефективності організації роботи може бути використання інформаційних систем диспетчеризації. Такі системи дозволяють централізовано зберігати інформацію, забезпечувати швидкий доступ до необхідних даних та спрощувати виконання типових операцій. На практиці це дає можливість зменшити кількість повторного введення інформації та підтримувати єдиний інформаційний простір для роботи диспетчера [2].

В межах даної роботи передбачається розробка інтерактивної системи диспетчера міського транспорту, побудованої за клієнт–серверною архітектурою. Обраний підхід дозволяє відокремити рівень представлення даних від логіки обробки та механізмів зберігання інформації, що спрощує підтримку програмного продукту та забезпечує можливість його подальшого розвитку [3, 4].

Основним призначенням системи є автоматизація роботи диспетчера шляхом створення єдиного програмного середовища для ведення обліку транспортних засобів, керування маршрутами та реєстрації виконання транспортних операцій.

Розроблювана система орієнтована на одного користувача – диспетчера, який працює через графічний інтерфейс клієнтської частини застосунку. Обробка запитів та взаємодія з базою даних здійснюються серверною частиною системи.

Відповідно до поставленої задачі визначено такі функціональні вимоги до системи, які наведені нижче.

Система повинна забезпечувати можливість роботи з довідником транспортних засобів. Користувач повинен мати можливість додавати інформацію про транспорт, переглядати наявні записи, змінювати їх та видаляти за потреби. Для кожного транспортного засобу передбачається зберігання основних характеристик, необхідних для організації роботи.

Система повинна підтримувати ведення довідника маршрутів. Передбачено можливість створення нового маршруту, редагування його параметрів та перегляду інформації про існуючі маршрути.

Окремою функцією системи є призначення транспортного засобу на маршрут. Диспетчер повинен мати можливість обрати транспорт, визначити маршрут та встановити поточний стан виконання рейсу.

Для забезпечення зручності роботи передбачається формування узагальненої інформації про стан системи. До статистичних даних належать кількість зареєстрованого транспорту, кількість маршрутів та інформація щодо виконаних рейсів.

Таким чином, сформовані функціональні вимоги визначають основні можливості майбутньої системи та створюють основу для подальшого проєктування її структури, моделі даних та програмної реалізації.

1.2 Нефункціональні вимоги до системи

Під час розробки програмного забезпечення поряд із визначенням функціональних можливостей необхідно встановити нефункціональні вимоги, які описують умови роботи системи, обмеження реалізації та показники якості програмного продукту.

На відміну від функціональних вимог, які визначають перелік операцій, що виконує система, нефункціональні вимоги характеризують особливості її функціонування та впливають на зручність використання, продуктивність, підтримуваність і стабільність роботи [2].

Для розроблюваної інтерактивної системи диспетчера міського транспорту нефункціональні вимоги визначаються з урахуванням її призначення, обраної клієнт–серверної архітектури та передбачуваного сценарію використання.

Однією з основних вимог є зручність використання системи. Оскільки взаємодія користувача із застосунком здійснюється через графічний інтерфейс, структура екранів повинна бути зрозумілою та не вимагати додаткового навчання. Елементи керування мають бути розташовані логічно, а основні операції – виконуватися за мінімальну кількість дій користувача.

Наступною вимогою є продуктивність системи. Незважаючи на відсутність великого навантаження та складних обчислювальних процесів, система повинна забезпечувати достатньо швидке виконання типових операцій: відкриття форм, отримання даних із сервера, відображення списків транспорту та маршрутів, а також запис інформації до бази даних. При роботі користувача не повинні виникати помітні затримки під час виконання стандартних операцій.

Важливою характеристикою є надійність функціонування. Система повинна забезпечувати коректну обробку введених даних та не допускати появи некомпетентної інформації в базі даних. У випадку виникнення помилок під час обміну даними або введення некоректних значень користувач повинен отримувати відповідне повідомлення без завершення роботи програми.

Окрему увагу необхідно приділити цілісності та збереженню даних. Усі зміни, внесені користувачем, повинні коректно зберігатися в базі даних та бути доступними під час наступного запуску системи. Передбачено використання централізованого зберігання інформації на стороні сервера, що дозволяє уникнути дублювання даних і забезпечує єдине джерело актуальної інформації [4, 5].

Ще однією вимогою є масштабованість рішення. Хоча в межах даної роботи система орієнтована на використання одним диспетчером, обрана клієнт–серверна

архітектура повинна дозволяти подальше розширення функціональності без необхідності суттєвої зміни структури програмного забезпечення [3].

Для забезпечення можливості подальшого супроводу система повинна відповідати вимогам підтримуваності програмного коду. Реалізація застосунку має передбачати логічний поділ клієнтської частини, серверної логіки та рівня доступу до даних. Такий підхід спрощує внесення змін та зменшує залежність між окремими компонентами [5].

Також до нефункціональних вимог належить сумісність програмного забезпечення. Розроблена система повинна коректно працювати у середовищі операційної системи Windows та використовувати стандартні програмні засоби без необхідності встановлення спеціалізованого обладнання.

Отже, визначення нефункціональних вимог дозволяє сформувати додаткові критерії оцінювання якості майбутньої системи та створює основу для вибору архітектурних і технологічних рішень на наступних етапах проєктування.

1.3 Вибір технологій реалізації системи

Після визначення вимог до програмної системи необхідно обрати технологічні засоби її реалізації. Вибір технологій впливає не лише на процес розробки, але й на подальшу підтримку програмного забезпечення, складність внесення змін та відповідність поставленим функціональним вимогам.

Оскільки в межах даної роботи передбачається створення настільної клієнт–серверної системи диспетчера міського транспорту, під час вибору враховувалися такі критерії:

– простота реалізації;

– підтримка клієнт–серверної взаємодії;

– наявність засобів створення графічного інтерфейсу;

– сумісність із реляційними базами даних;

– можливість використання без додаткової інфраструктури;

– зручність супроводу програмного продукту.

1.3.1 Вибір мови програмування та платформи реалізації

Для реалізації програмної системи розглядалися три мови програмування:

– Java;

– C# (.NET);

– Python.

Таблиця 1.1 – Порівняння мов програмування для реалізації системи

Критерій	Java	C#	Python
Кросплатформеність	+	+-	+
Робота з GUI	+	+	+-
Підтримка клієнт–серверної архітектури	+	+	+
Інтеграція з PostgreSQL	+	+	+
Простота розгортання	+	+	+
Готовність до мобільного клієнта	+	+-	+

Мова Java була обрана як основна технологія реалізації. Вона забезпечує незалежність від операційної системи, підтримує об’єктно-орієнтований підхід та містить засоби для створення мережевих застосунків і роботи з базами даних. На відміну від C#, використання Java не прив’язує систему до конкретної платформи. Порівняно з Python, Java забезпечує більш структурований підхід до розробки великих прикладних застосунків та є поширеним рішенням для реалізації клієнт–серверних систем [3].

1.3.2 Вибір технології створення інтерфейсу

Для реалізації клієнтської частини розглядалися:

– JavaFX;

– Swing;

– веб-інтерфейс (HTML/CSS/JavaScript).

Таблиця 1.2 – Порівняння технологій створення інтерфейсу

Критерій	JavaFX	Swing	Web
Сучасний інтерфейс	+	-	+
Інтеграція з Java	+	+	+-
Простота реалізації	+	+	-
Швидкість розробки	+	+	-
Не потребує браузера	+	+	-

Для реалізації клієнтської частини було обрано JavaFX**.** Це рішення дозволяє створити графічний інтерфейс без використання додаткових вебтехнологій та забезпечує повну інтеграцію із серверною логікою, реалізованою мовою Java. Використання вебінтерфейсу в межах даної роботи визнано надлишковим через збільшення складності проєкту.

1.3.3 Вибір системи керування базами даних

Для організації зберігання інформації були розглянуті:

– PostgreSQL;

– MySQL;

– SQLite.

Таблиця 1.3 – Порівняння систем керування базами даних

Критерій	PostgreSQL	MySQL	SQLite
Робота з багатьма таблицями	+	+	+_
Підтримка транзакцій	+	+	+-
Масштабованість	+	+	-
Простота інтеграції	+	+	+
Підходить для клієнт–серверної системи	+	+	-

В результаті аналізу для роботи обрано PostgreSQL [6, 7]. Використання цієї системи дозволяє централізовано зберігати дані та підтримує подальше розширення структури бази без зміни архітектури програмного продукту [4, 6].

Для передавання даних між компонентами системи обрано формат JSON, який забезпечує компактне представлення інформації та спрощує її обробку під час обміну між клієнтом і сервером.

Таким чином, за результатами проведеного порівняльного аналізу для реалізації системи обрано стек технологій Java + JavaFX + PostgreSQL + JSON, який забезпечує відповідність функціональним вимогам і не створює надлишкової складності реалізації [8, 9].

1.4 Висновки по розділу 1

У першому розділі проведено аналіз підходів до розробки клієнт–серверних програмних систем та визначено основні вимоги до розроблюваної інтерактивної системи диспетчера міського транспорту.

В результаті аналізу предметної області встановлено, що для забезпечення ефективної роботи диспетчера необхідно реалізувати централізоване зберігання інформації про транспортні засоби, маршрути та результати виконання транспортних операцій. На основі цього сформовано перелік функціональних вимог, які визначають основні можливості майбутньої системи, а також нефункціональних вимог, що встановлюють критерії її якості, продуктивності, зручності використання та підтримуваності [11-13].

Проведено порівняльний аналіз технологій реалізації програмного забезпечення та обґрунтовано вибір інструментальних засобів розробки [14, 15]. Для реалізації серверної частини та клієнтського застосунку обрано мову програмування Java [14], для створення графічного інтерфейсу — технологію JavaFX [15], а для організації централізованого зберігання даних — систему керування базами даних PostgreSQL [6, 7, 13]. Для обміну інформацією між компонентами системи передбачено використання формату JSON.

Отримані результати створюють основу для переходу до етапу проєктування програмної системи, побудови її архітектури та визначення структури бази даних.

2 РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ СИСТЕМИ

2.1 Проєктування серверної архітектури

Після визначення функціональних вимог та вибору технологій реалізації є проєктування серверної частини системи. Сервер виступає центральним компонентом програмного забезпечення та забезпечує взаємодію між клієнтським застосунком і базою даних.

В межах розроблюваної інтерактивної системи диспетчера міського транспорту серверна частина виконує функції прийому запитів від користувача, обробки даних, виконання перевірок коректності введеної інформації та організації доступу до бази даних. Такий підхід дозволяє відокремити логіку обробки інформації від графічного інтерфейсу клієнтського застосунку та забезпечити централізоване керування даними [3, 9].

Під час проєктування було обрано клієнт–серверну архітектуру, у якій клієнтська частина відповідає за взаємодію з користувачем, а серверна — за виконання основних операцій системи.

На рисунку 2.1 наведено структуру серверної частини розроблюваної системи. Серверний застосунок організовано у вигляді набору функціональних пакетів, що забезпечують розподіл відповідальності між окремими компонентами програмного забезпечення.

Основою побудови серверної частини є багаторівневий підхід, відповідно до якого різні задачі системи реалізуються незалежними компонентами. Така організація дозволяє зменшити залежність між модулями та спрощує подальший супровід програмного забезпечення.

Пакет controller призначений для обробки запитів, що надходять від клієнтської частини системи. На цьому рівні виконується прийом даних, передача їх до внутрішніх компонентів та формування відповіді.

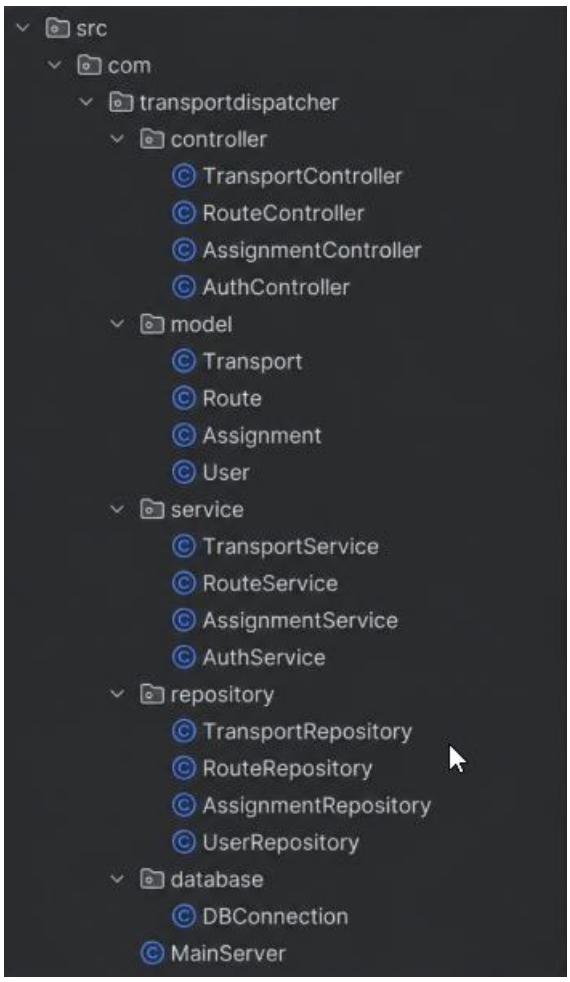

Рисунок 2.1 – Стуктура серверної частини системи

Пакет model містить структури даних, які використовуються для представлення інформації предметної області. До таких даних належать транспортні засоби, маршрути та результати диспетчерських операцій.

Пакет service реалізує прикладну логіку роботи системи. Саме на цьому рівні виконуються основні операції обробки інформації та перевірка коректності виконання дій користувача.

Пакет repository забезпечує взаємодію із системою керування базами даних та реалізує механізми отримання, збереження та зміни інформації.

Пакет database використовується для організації підключення до бази даних та керування доступом до неї.

Окремо виділяється модуль запуску серверного застосунку, який забезпечує ініціалізацію компонентів та запуск системи.

Запропонована структура серверної частини дозволяє забезпечити логічне розділення функціональності та створює основу для подальшої реалізації програмного забезпечення.

2.2 Апаратні вимоги до сервера

Для забезпечення роботи серверної частини необхідно визначити мінімальні апаратні характеристики, достатні для запуску програмного забезпечення та підтримки взаємодії між компонентами системи. Оскільки в межах роботи реалізується навчальна клієнт–серверна система диспетчера міського транспорту, сервер не орієнтований на одночасне обслуговування великої кількості користувачів та не виконує ресурсоємних обчислень.

Основними функціями серверної частини є прийом запитів від клієнтського застосунку, обробка інформації та взаємодія із системою керування базами даних PostgreSQL [6, 7, 13]. Для виконання зазначених задач рекомендується використання обчислювального середовища із процесором не нижче рівня двоядерного процесора із тактовою частотою від 2,0 ГГц.

Для забезпечення одночасної роботи серверного застосунку, середовища виконання Java [14] та системи керування базами даних необхідний обсяг оперативної пам’яті не менше 4 ГБ. Для більш стабільної роботи та зменшення часу обробки запитів рекомендованим є використання 8 ГБ оперативної пам’яті.

Для розміщення серверного застосунку, службових файлів та бази даних необхідно передбачити не менше 10 ГБ вільного дискового простору. Для підвищення швидкодії бажаним є використання твердотільного накопичувача.

Для організації взаємодії між клієнтською та серверною частинами сервер повинен мати підключення до локальної мережі зі швидкістю не менше 100 Мбіт/с.

Серверна частина системи повинна працювати під керуванням сучасної операційної системи із підтримкою встановлення середовища виконання Java та системи керування базами даних PostgreSQL[6, 7, 13]. В межах реалізації проєкту передбачається використання операційної системи Windows 10 або новішої версії.

Таким чином, визначені апаратні вимоги забезпечують можливість стабільного функціонування серверної частини системи без використання спеціалізованого серверного обладнання та відповідають масштабу розроблюваного програмного забезпечення.

2.3 Проєктування бази даних

Одним із ключових етапів розробки серверної частини системи є проєктування структури бази даних. Від способу організації даних залежить коректність роботи програмного забезпечення, зручність обробки інформації та можливість подальшого розширення функціональності системи.

Оскільки розроблювана система призначена для автоматизації роботи диспетчера міського транспорту, структура бази даних повинна забезпечувати зберігання інформації про транспортні засоби, маршрути та результати їх взаємного призначення.

Під час проєктування використовувався реляційний підхід до організації даних. Такий підхід дозволяє розділити інформацію на окремі сутності та встановити зв’язки між ними без дублювання даних [2, 5].

З урахуванням функціональних вимог було визначено три основні інформаційні сутності системи:

– транспортний засіб;

– маршрут;

– призначення транспорту на маршрут.

Сутність «Транспортний засіб» використовується для зберігання інформації про транспорт, доступний для виконання рейсів. Для кожного запису передбачається збереження унікального ідентифікатора, номера транспортного засобу, його типу та поточного стану.

Сутність «Маршрут» призначена для опису транспортних маршрутів. Для кожного маршруту зберігається ідентифікатор, номер маршруту та інформація про початковий і кінцевий пункти руху.

Сутність «Призначення» використовується для фіксації відповідності між транспортним засобом та маршрутом. У цій сутності також зберігається поточний статус виконання рейсу.

Між сутностями встановлюються зв’язки для забезпечення узгодженості даних. Один маршрут може використовуватися у декількох записах призначення, аналогічно один транспортний засіб може бути пов’язаний із декількома записами призначення.

Для ідентифікації записів у кожній таблиці використовується первинний ключ [6]. Для реалізації зв’язків між таблицями застосовуються зовнішні ключі.

На етапі проєктування передбачається приведення структури бази даних до такого стану, який дозволяє уникнути дублювання інформації та забезпечує цілісність збережених даних.

Запропонована структура бази даних створює основу для подальшої реалізації серверної логіки та взаємодії із клієнтською частиною системи.

На рисунку 2.2 наведено ER-діаграму бази даних розроблюваної системи. Діаграма відображає основні сутності предметної області та зв’язки між ними.

Після проведення аналізу предметної області було визначено набір сутностей, необхідних для реалізації функціональності системи диспетчера міського транспорту. У базі даних зберігається інформація про транспортні засоби, маршрути та результати призначення транспорту на маршрут. Такий підхід дозволяє організувати централізоване зберігання даних та забезпечити взаємозв’язок між окремими компонентами системи.

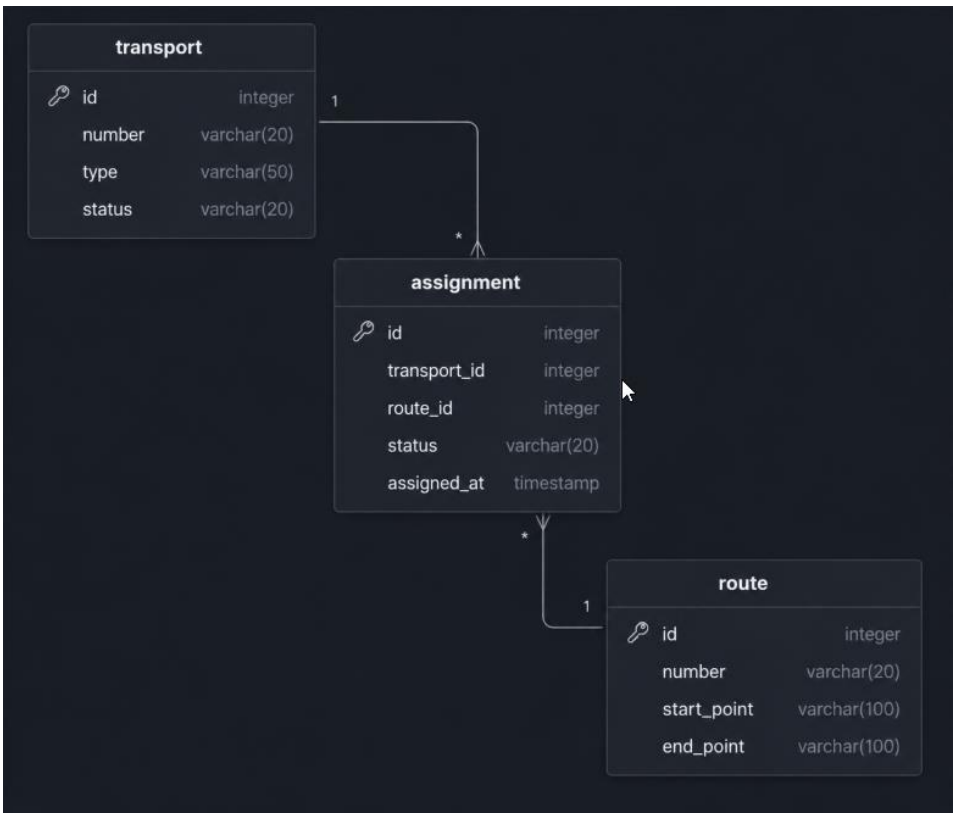

Рисунок 2.2 – ER-діаграма бази даних системи

Основною сутністю системи є транспортний засіб, для якого зберігаються ідентифікаційні дані, тип транспорту та поточний стан. Окремо виділено сутність маршрутів, що містить інформацію про транспортні маршрути та основні параметри руху.

Для реалізації диспетчерських операцій передбачено сутність призначення транспорту на маршрут. Вона використовується для встановлення зв’язку між транспортним засобом та маршрутом, а також дозволяє зберігати інформацію про поточний стан виконання рейсу.

Запропонована структура бази даних забезпечує реалізацію поточного функціоналу системи та створює основу для подальшого розширення програмного забезпечення без необхідності зміни основної структури даних.

Сутність Transport (лістинг 2.1) призначена для зберігання інформації про транспортні засоби. Вона містить унікальний ідентифікатор транспортного засобу, його номер, тип та поточний стан.

Лістинг 2.1 – DBML-опис сутності Transport

```
Table transport {
id int [pk, increment]
number varchar [not null]
type varchar [not null]
status varchar
}
```

Сутність Route (лістинг 2.2) використовується для зберігання інформації про маршрути. Для кожного маршруту зберігається номер маршруту, початковий та кінцевий пункти руху.

Лістинг 2.2 – DBML-опис сутності Route

```
Table route {
id int [pk, increment]
number varchar [not null]
start_point varchar [not null]
end_point varchar [not null]
}
```

Сутність Assignment (лістинг 2.3) використовується для збереження інформації про призначення транспортного засобу на маршрут. Сутність містить посилання на транспортний засіб та маршрут, а також поточний статус виконання рейсу.

Лістинг 2.3 – DBML-опис сутності Assignment

```
Table assignment {
id int [pk, increment]
transport_id int [ref: > transport.id]
route_id int [ref: > route.id]
status varchar
}
description text
}
```

У сукупності описані сутності формують цілісну структуру бази даних, необхідну для реалізації функціональності системи диспетчера міського транспорту. Запропонована модель забезпечує зберігання інформації про транспортні засоби, маршрути та результати призначення транспорту на маршрут, а також підтримує зв’язки між основними сутностями системи.

Структура бази даних побудована таким чином, щоб уникнути дублювання інформації та забезпечити цілісність даних під час роботи серверної частини застосунку. Використання окремих таблиць для транспорту, маршрутів і диспетчерських операцій спрощує обробку інформації та дозволяє виконувати подальше розширення системи без зміни основної логіки організації даних.

Первинне наповнення бази даних виконується за допомогою SQL-запитів створення початкових записів. Фрагмент наповнення бази даних наведено в лістингу 2.4.

Лістинг 2.4 – Фрагмент первинного наповнення бази даних

```
INSERT INTO transport (id, number, type, status) VALUES
(1, 'AE1023BM', 'Автобус', 'Активний'),
(2, 'AP4587CK', 'Тролейбус', 'Активний'),
(3, 'AE7712HP', 'Автобус', 'На обслуговуванні');
INSERT INTO route (id, number, start_point, end_point) VALUES
(1, '3', 'Вокзал', 'Центральний ринок'),
(2, '7', 'Проспект Металургів', 'Автовокзал'),
(3, '12', 'Бульвар Шевченка', 'Південний мікрорайон');
INSERT INTO assignment (id, transport_id, route_id, status) VALUES
(1, 1, 1, 'Виконується'),
(2, 2, 2, 'Призначено'),
(3, 3, 3, 'Завершено');
```

У процесі роботи серверний застосунок виконує звернення до бази даних для отримання, зміни та збереження інформації. Клієнтська частина отримує необхідні дані через сервер за допомогою мережевої взаємодії.

2.4 Реалізація серверних модулів

Реалізація серверної частини системи виконується у вигляді набору окремих програмних модулів, кожен із яких відповідає за визначений набір функцій. Основою серверної логіки є моделі даних, які використовуються для

представлення сутностей предметної області та забезпечують обмін інформацією між серверною частиною і базою даних.

Початковим етапом реалізації є створення моделей даних. Як приклад у лістингу 2.5 наведено клас Transport, який використовується для представлення транспортного засобу в системі.

Набір полів класу дозволяє зберігати основну інформацію про транспортний засіб: ідентифікатор, номер транспорту, його тип та поточний стан. Використання окремого класу для представлення транспортного засобу спрощує передачу даних між компонентами системи та дозволяє централізовано працювати з інформацією про транспорт.

Лістинг 2.5 – Реалізація класу Transport

```
public class Transport {
private int id;
private String number;
private String type;
private String status;
public Transport() {
}
public Transport(int id, String number, String type, String status)
{
this.id = id;
this.number = number;
this.type = type;
this.status = status;
}
public int getId() {
return id;
}
public void setId(int id) {
this.id = id;
}
public String getNumber() {
return number;
}
public void setNumber(String number) {
this.number = number;
}
public String getType() {
return type;
}
public void setType(String type) {
this.type = type;
}
```

```
public String getStatus() {
return status;
}
public void setStatus(String status) {
this.status = status;
}}
```

Аналогічним чином реалізуються моделі маршрутів та призначень транспорту. Використання окремих класів для представлення сутностей дозволяє забезпечити логічне розділення даних та спрощує подальшу реалізацію серверної логіки.

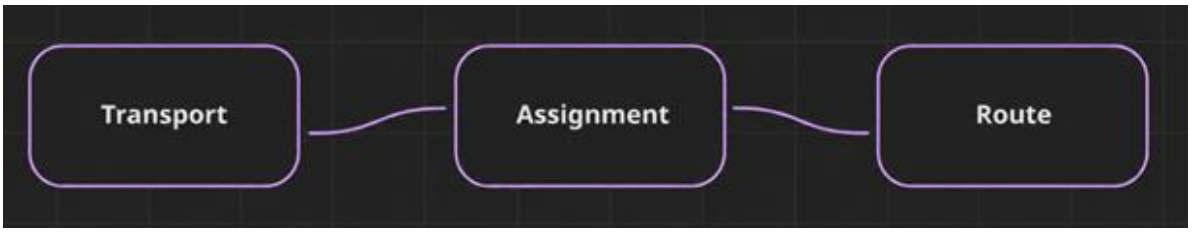

Рисунок 2.3 – Співвідношення між класами моделі

Далі реалізуються серверні модулі, які забезпечують роботу з базою даних та обробку запитів клієнтської частини. Одним із основних елементів серверної частини є клас DB, призначений для організації взаємодії з базою даних. Він забезпечує встановлення з’єднання із СКБД PostgreSQL [6, 7, 13], створення необхідних таблиць, первинне наповнення бази даних, а також виконання операцій читання, додавання, зміни та видалення записів.

В межах розроблюваної системи клас DB використовується для роботи з трьома основними групами даних: транспортними засобами, маршрутами та призначеннями транспорту на маршрут. Такого набору методів достатньо для мінімальної реалізації серверної частини, оскільки клієнтський застосунок повинен мати можливість переглядати наявні записи, додавати нові дані та змінювати інформацію, необхідну для роботи диспетчера.

Перелік ключових методів класу DB наведено в таблиці 2.1.

Таблиця 2.1 – Основні методи класу DB

Назва методу	Аргументи	Призначення
connect()	–	Встановлення з’єднання з базою даних
createTables()	–	Створення таблиць бази даних
insertInitialData()	–	Первинне наповнення бази даних
getTransports()	–	Отримання списку транспортних засобів
getRoutes()	–	Отримання списку маршрутів
getAssignments()	–	Отримання списку призначень
addTransport()	Transport transport	Додавання транспортного засобу
addAssignment()	Assignment assignment	Створення призначення транспорту на маршрут
updateAssignmentStatus()	int id, String status	Зміна статусу призначення

Покажемо код методу getTransportById класу DB у лістингу 2.6. Метод використовується для отримання інформації про транспортний засіб за його ідентифікатором. Під час виконання методу формується SQL-запит до бази даних, після чого отримані дані перетворюються в об’єкт класу Transport.

Лістинг 2.6 – Код методу getTransportById класу DB

```
public Transport getTransportById(int id) {
Transport transport = null;
String sql = "SELECT * FROM transport WHERE id = ?";
try (PreparedStatement statement =
connection.prepareStatement(sql)) {
statement.setInt(1, id);
ResultSet resultSet = statement.executeQuery();
if (resultSet.next()) {
transport = new Transport(
resultSet.getInt("id"),
resultSet.getString("number"),
resultSet.getString("type"),
resultSet.getString("status"));
}
} catch (SQLException e) {
e.printStackTrace();
}
return transport;
}
```

Метод використовує механізм підготовлених SQL-запитів (PreparedStatement), що дозволяє безпечно передавати параметри до запиту та зменшує ризик помилок під час роботи з базою даних. Отримані з таблиці дані використовуються для створення об’єкта класу Transport, який надалі може бути переданий іншим компонентам серверної частини або клієнтському застосунку.

Метод getModelById реалізує повний цикл отримання однієї автомобільної моделі з бази даних та побудови відповідного об’єкта доменної моделі в оперативній пам’яті сервера. На вході метод приймає ідентифікатор моделі modelId, який однозначно визначає запис у таблиці model. Далі формується параметризований SQL-запит, у якому таблиця model виступає центральною, а всі пов’язані з нею сутності підтягуються через JOIN. Використання JOIN дозволяє за один запит отримати всі необхідні дані без додаткових звернень до бази. Запит явно вибирає поля кожної сутності з псевдонімами (AS ...). Це робиться для уникнення конфліктів імен колонок (id, name) та для чіткого зіставлення значень із параметрами конструкторів модельних класів. Наприклад, окремо вибираються ідентифікатор, назва та опис марки (brand_id, brand_name, brand_description), атрибути типу кузова, технічні характеристики двигуна, параметри трансмісії та дані кольору. Після підготовки запиту створюється PreparedStatement, у який передається значення modelId. Використання PreparedStatement забезпечує захист від SQL-ін’єкцій та дозволяє СКБД повторно використовувати план виконання запиту. Далі виконується запит і отримується ResultSet. Якщо результат порожній, метод повертає null, що сигналізує викликаючому коду про відсутність моделі з таким ідентифікатором. Якщо запис знайдено, відбувається поетапне створення об’єктів модельного рівня. Спочатку створюються об’єкти базових довідникових сутностей - Brand, BodyType, Engine, Transmission та Color. Кожен з них ініціалізується безпосередньо через конструктор з повним набором параметрів, що відповідає структурі таблиць бази даних. Після цього створюється об’єкт Model, який виступає інкапсулюючою сутністю. У його конструктор передається ідентифікатор моделі, назва, базова ціна, текстовий опис та посилання на PDF-документ, а також раніше створені об’єкти Brand, BodyType, Engine, Transmission

та Color. Таким чином формується повністю зв’язаний об’єктний граф, що відображає одну конкретну модель автомобіля в тому вигляді, в якому вона зберігається в базі даних. У результаті метод повертає ініціалізований екземпляр класу Model, який може бути без додаткової обробки використаний у серверній логіці, серіалізований у JSON та переданий клієнтському застосунку. Такий підхід відокремлює доступ до даних від бізнес-логіки та забезпечує однозначне відображення реляційної структури бази даних у об’єктній моделі сервера.

Продемонструємо результат виклику метода getModelById(1) з ідентифікатором моделі 1 у вигляді віртуальної таблиці на рис. 2.4.

Рисунок 2.4 – Результат перетворення даних у об’єкт Transport

Метод getTransportById() використовується для отримання інформації про конкретний транспортний засіб за його ідентифікатором. У результаті виконання SQL-запиту з бази даних вибирається запис із таблиці transport, після чого отримані дані перетворюються в об’єкт класу Transport. Такий підхід дозволяє відокремити логіку роботи з базою даних від інших компонентів серверної частини та спрощує подальшу обробку інформації.

Аналогічним чином реалізуються методи отримання маршрутів та призначень транспорту. Усі операції доступу до бази даних зосереджені в класі DB, що забезпечує централізовану роботу з даними та спрощує супровід програмного коду.

Після реалізації модуля доступу до бази даних необхідно організувати мережеву взаємодію між сервером та клієнтським застосунком. Для цього використовується клас Networking, який відповідає за створення серверного сокета, очікування клієнтських підключень та передавання даних між компонентами системи.

Клас Networking забезпечує безперервну роботу сервера та підтримує можливість одночасного обслуговування декількох клієнтів. При надходженні нового підключення сервер створює окремий потік виконання для його обробки, що дозволяє уникнути блокування роботи інших клієнтів і забезпечує стабільність функціонування системи.

Код класу Networking наведено в лістингу 2.7.

Лістинг 2.7 – Код класу Networking

```
public class Networking {
private final int port;
public Networking(int port) {
this.port = port;
}
public void start() {
try (ServerSocket serverSocket = new ServerSocket(port)) {
System.out.println("Server started on port " + port);
while (true) {
Socket clientSocket = serverSocket.accept();
System.out.println(
"Client connected: "
+ clientSocket.getInetAddress()
);
Client client = new Client(clientSocket);
Thread thread = new Thread(client);
thread.start();
}
} catch (IOException e) {
e.printStackTrace();
}}}
```

У наведеному фрагменті коду реалізовано базову логіку мережевої взаємодії сервера з клієнтськими застосунками. Після запуску сервер створює об’єкт ServerSocket та переходить у режим очікування підключень. Для кожного нового клієнта створюється окремий потік виконання, що дозволяє обслуговувати

декілька підключень одночасно. Такий підхід забезпечує незалежність клієнтських сесій та підвищує стабільність роботи серверної частини.

Після встановлення мережевого з’єднання керування передається класу Client, який відповідає за обробку запитів конкретного клієнта. Для кожного нового підключення створюється окремий екземпляр цього класу, що дозволяє ізолювати роботу окремих клієнтів та уникнути взаємного впливу між ними.

Основним завданням класу Client є приймання запитів від клієнтського застосунку, отримання необхідних даних із бази даних та формування відповіді для подальшого передавання по мережі. Під час роботи клієнтський потік взаємодіє з класом DB, який виконує безпосередній доступ до бази даних. Отримані результати перетворюються у модельні об’єкти та передаються клієнту.

В межах розроблюваної системи клієнтський застосунок може запитувати інформацію про транспортні засоби, маршрути та призначення транспорту на маршрут. Після отримання відповідного запиту клас Client викликає необхідні методи класу DB, формує відповідь та надсилає її клієнту через активне мережеве з’єднання.

Код класу Client наведено в лістингу 2.8.

Лістинг 2.8 – Фрагмент реалізації класу Client

```
public class Client implements Runnable {
private final Socket socket;
private final DB db;
public Client(Socket socket) {
this.socket = socket;
this.db = new DB();
}
@Override
public void run() {
try (
BufferedReader reader =
new BufferedReader(
new InputStreamReader(
socket.getInputStream()));
PrintWriter writer =
new PrintWriter(
socket.getOutputStream(), true)) {
String request = reader.readLine();
if ("GET_TRANSPORTS".equals(request)) {
```

```
List<Transport> transports =
db.getTransports();
writer.println(transports.toString());
}
} catch (IOException e) {
e.printStackTrace();
}}}
```

У наведеному фрагменті коду клас Client реалізує інтерфейс Runnable, що дозволяє виконувати його в окремому потоці. Після отримання запиту від клієнтського застосунку виконується звернення до класу DB, який отримує необхідні дані з бази даних. Результат роботи передається клієнту через мережеве з'єднання. Такий підхід дозволяє розділити логіку мережевої взаємодії та логіку доступу до даних, що спрощує подальший супровід програмного забезпечення.

Такий підхід дозволяє не змішувати логіку мережевого обміну з логікою роботи бази даних. Клас Client відповідає лише за приймання запиту, виклик потрібного методу та передавання відповіді. Усі операції з таблицями бази даних залишаються в класі DB.

Фрагмент обробки запиту на отримання списку транспортних засобів наведено в лістингу 2.9.

Лістинг 2.9 – Обробка запиту на отримання списку транспорту

```
String request = reader.readLine();
if ("GET_TRANSPORTS".equals(request)) {
List<Transport> transports = db.getTransports();
for (Transport transport : transports) {
writer.println(
transport.getId() + ";" +
transport.getNumber() + ";" +
transport.getType() + ";" +
transport.getStatus()
);
}
writer.println("END");
}
```

У цьому фрагменті сервер отримує від клієнта команду GET_TRANSPORTS. Після цього викликається метод getTransports() класу DB, який повертає список транспортних засобів. Кожен об'єкт класу Transport

перетворюється у текстовий рядок і передається клієнту через вихідний потік. Рядок END використовується як ознака завершення передавання даних.

Аналогічно може бути організована обробка запитів на отримання маршрутів і призначень транспорту. Для цього достатньо додати відповідні команди, наприклад GET_ROUTES та GET_ASSIGNMENTS, і викликати потрібні методи класу DB.

Наведені програмні модулі формують основу серверної частини системи диспетчера міського транспорту. Клас DB забезпечує централізований доступ до бази даних та виконує всі операції читання і зміни інформації. Клас Networking відповідає за організацію мережевої взаємодії та приймання клієнтських підключень, а клас Client виконує обробку запитів, що надходять від клієнтського застосунку.

Використання окремих модельних класів дозволяє відокремити представлення даних від логіки їх обробки та спрощує взаємодію між компонентами системи. Такий підхід забезпечує зрозумілу структуру програмного коду, зменшує зв’язність між модулями та полегшує подальший супровід програмного забезпечення.

Реалізована серверна частина забезпечує зберігання та обробку інформації про транспортні засоби, маршрути та призначення транспорту, а також підтримує мережевий обмін даними з клієнтським застосунком.

2.5 Висновки по розділу 2

У другому розділі виконано проєктування та реалізацію серверної частини системи диспетчера міського транспорту. Розроблено загальну архітектуру серверного застосунку, визначено склад основних модулів та принципи їх взаємодії між собою.

У процесі проєктування було сформовано структуру серверного проєкту та визначено призначення його основних компонентів. Для організації зберігання інформації спроєктовано реляційну базу даних, яка забезпечує роботу з даними про транспортні засоби, маршрути та призначення транспорту на маршрути. Також побудовано ER-діаграму бази даних та визначено структуру основних сутностей системи.

В межах реалізації серверної частини розроблено модельні класи, що відображають сутності предметної області, а також модулі доступу до бази даних і мережевої взаємодії. Для роботи з даними використано систему керування базами даних PostgreSQL та технологію JDBC. Організацію мережевого обміну між сервером і клієнтом реалізовано за допомогою сокетів, що забезпечує прямий обмін даними між компонентами системи.

Розроблений серверний застосунок підтримує приймання клієнтських підключень, обробку запитів та отримання інформації з бази даних. Для забезпечення можливості одночасної роботи декількох клієнтів використовується багатопотокова обробка підключень.

Таким чином, у результаті виконання другого розділу створено серверну частину системи, яка забезпечує зберігання, обробку та передавання даних клієнтському застосунку і створює основу для реалізації користувацького інтерфейсу, розробка якого розглядається в наступному розділі.

3 РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ СИСТЕМИ

3.1 Проєктування клієнтської архітектури

Проєктування клієнтської частини системи починається з визначення основних сценаріїв роботи користувача та структури інтерфейсу. Так як розроблювана система призначена для диспетчера міського транспорту, то клієнтський застосунок має забезпечувати швидкий доступ до основних операцій:

перегляду транспорту, перегляду маршрутів, створення призначень та перегляду статистичної інформації.

Клієнтська частина реалізується у вигляді настільного JavaFX-застосунку [15]. Такий підхід дозволяє створити графічний інтерфейс користувача без використання веббраузера або мобільної платформи. Застосунок орієнтований на роботу на персональному комп’ютері диспетчера, тому структура екранів повинна бути простою, зрозумілою та зручною для щоденного використання.

Почнемо проєктування з діаграми варіантів використання клієнтського застосунку системи диспетчера міського транспорту, наведеної на рисунку 3.1.

Рисунок 3.1 – Use Case діаграма клієнтського застосунку системи диспетчера міського транспорту

Наведена діаграма відображає основні функціональні можливості клієнтської частини системи. Центральною дійовою особою є диспетчер, який

взаємодіє із застосунком для виконання повсякденних операцій з керування транспортом.

Перегляд транспорту дозволяє отримувати інформацію про наявні транспортні засоби та їх поточний стан. Перегляд маршрутів забезпечує доступ до відомостей про маршрути, що використовуються в системі. Функція призначення транспорту на маршрут використовується для формування відповідності між транспортним засобом та маршрутом. У процесі роботи диспетчер також має можливість змінювати статус призначення відповідно до поточного стану виконання рейсу.

Окремим варіантом використання є перегляд статистики, який дозволяє отримувати узагальнену інформацію щодо роботи транспорту та виконаних призначень. Сукупність наведених варіантів використання повністю відповідає функціональним вимогам, сформульованим у першому розділі роботи, та визначає структуру клієнтської частини програмної системи.

Користувач може використовувати застосунок у різних варіантах.

Запуск застосунку. Диспетчер відкриває настільний застосунок. Система ініціалізує головне вікно, встановлює з’єднання із сервером та готує інтерфейс до роботи. Після успішного запуску користувачу відображається головне меню з доступом до основних розділів.

Перегляд транспорту. Диспетчер відкриває розділ транспорту та переглядає список транспортних засобів. Для кожного запису відображаються номер, тип транспорту та його поточний стан.

Перегляд маршрутів. Користувач переходить до розділу маршрутів, де відображається перелік маршрутів із зазначенням номера маршруту, початкового та кінцевого пунктів.

Призначення транспорту на маршрут. Диспетчер обирає транспортний засіб і маршрут, після чого створює призначення. Дані передаються на сервер і зберігаються в базі даних.

Зміна статусу призначення. У процесі роботи диспетчер може змінити стан призначення, наприклад встановити статус «Виконується» або «Завершено»;

Перегляд статистики. Користувач відкриває розділ статистики, де відображається узагальнена інформація про кількість транспортних засобів, маршрутів та призначень.

Use case «Запуск застосунку» включає можливість переходу до основних розділів системи. Use case «Перегляд транспорту» та «Перегляд маршрутів» використовуються як основа для виконання use case «Призначення транспорту на маршрут». Use case «Зміна статусу призначення» є продовженням роботи з уже створеним призначенням. Use case «Перегляд статистики» є окремим сценарієм, який дозволяє отримати узагальнену інформацію про роботу системи.

Структура інтерфейсу клієнтського застосунку будується навколо головного вікна та декількох робочих вікон. Основними екранами настільного застосунку є:

- головне вікно;
- вікно транспорту;
- вікно маршрутів;
- вікно призначень;
- вікно статистики.

Головне вікно відображається після запуску програми та містить кнопки переходу до розділів «Транспорт», «Маршрути», «Призначення» та «Статистика».

Вікно транспорту призначене для перегляду переліку транспортних засобів у табличному вигляді.

Вікно маршрутів містить список маршрутів, що використовуються в системі.

Вікно призначень використовується для створення призначення транспорту на маршрут і зміни його статусу.

Вікно статистики відображає узагальнені показники роботи системи.

Для попереднього проєктування інтерфейсу користувача було розроблено wireframe клієнтського застосунку. Він дає можливість оцінити загальне розташування основних елементів інтерфейсу ще до етапу програмної реалізації. Оскільки клієнтська частина системи створюється як настільний JavaFX-

застосунок, wireframe відображає не мобільні екрани, а робочі вікна програми [15].

Wireframe клієнтського застосунку представлено на рисунку 3.2.

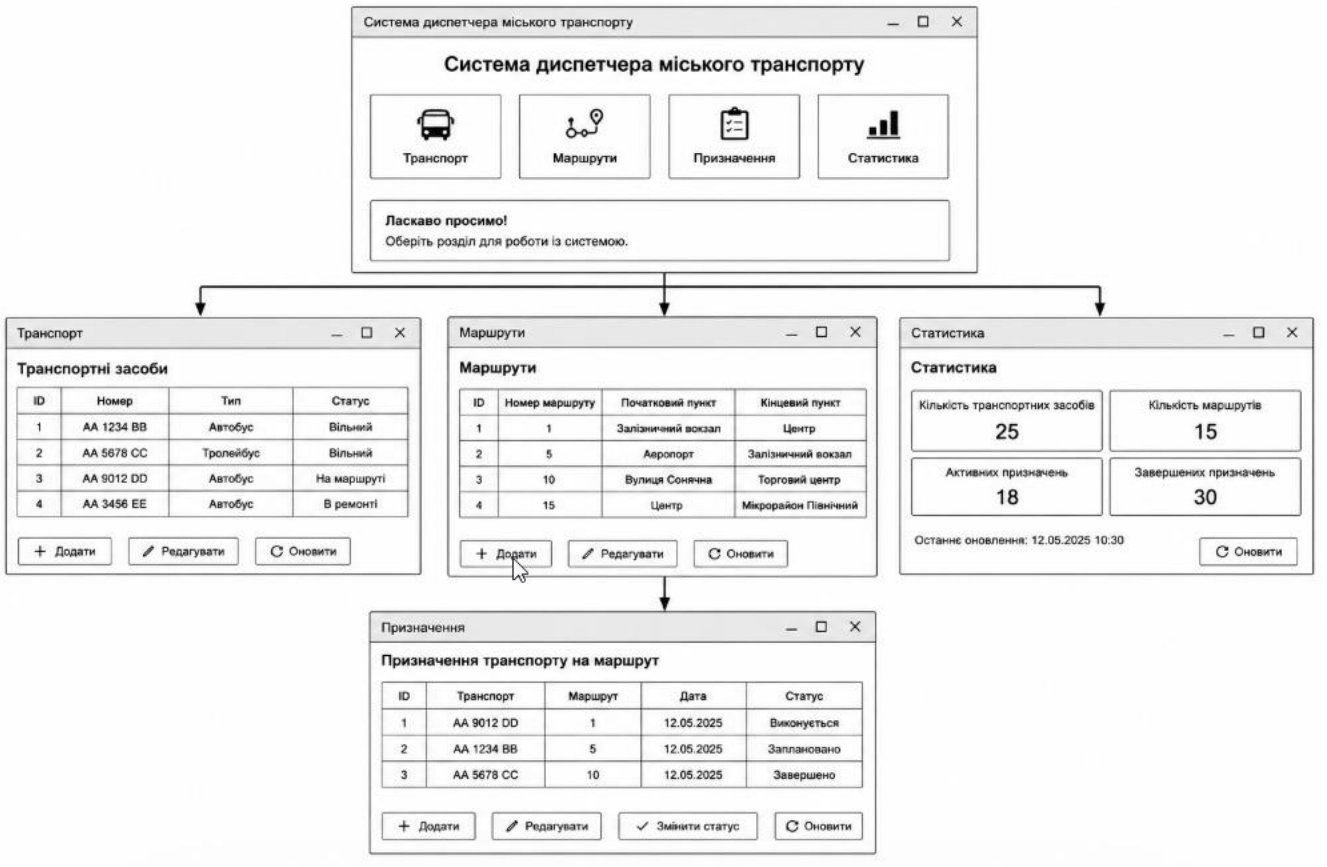

Рисунок 3.2 – Wireframe клієнтського застосунку

Головне вікно є початковою точкою роботи користувача із системою. Воно містить навігаційні кнопки для переходу до основних розділів застосунку. Такий підхід дозволяє не перевантажувати стартовий екран зайвою інформацією та швидко переходити до потрібної операції.

Вікно транспорту призначене для перегляду інформації про транспортні засоби. Основним елементом цього вікна є таблиця, у якій відображаються номер транспорту, його тип та поточний стан. Такий формат подання є зручним для диспетчера, оскільки дозволяє швидко переглядати наявні транспортні одиниці.

Вікно маршрутів використовується для відображення переліку маршрутів. У ньому подаються номер маршруту, початковий та кінцевий пункти. Ці дані надалі використовуються під час створення призначення транспорту на маршрут.

Вікно призначень є основним робочим екраном диспетчера. На ньому користувач може обрати транспортний засіб, маршрут та встановити статус призначення. Після підтвердження операції дані передаються до серверної частини та зберігаються в базі даних.

Вікно статистики призначене для перегляду узагальненої інформації. У ньому можуть відображатися кількість транспортних засобів, маршрутів і створених призначень. Це дозволяє диспетчеру швидко оцінити загальний стан системи.

Таким чином, клієнтська архітектура побудована навколо кількох простих робочих вікон, кожне з яких відповідає окремому сценарію роботи диспетчера. Така структура є достатньою для реалізації функціональних вимог системи та не ускладнює користувацьку взаємодію.

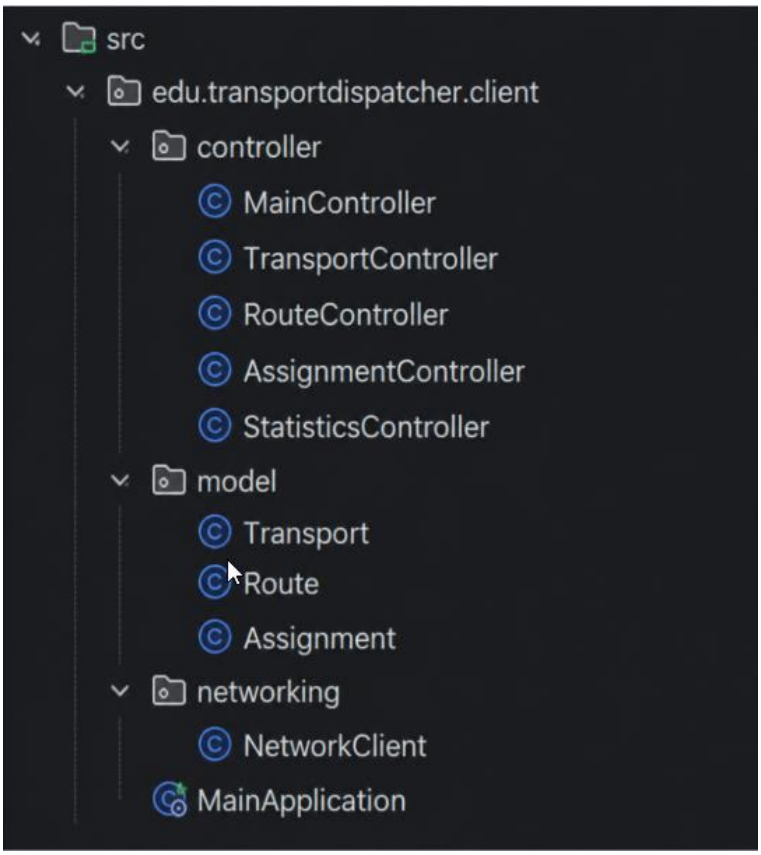

Рисунок 3.3 – Структура проєкту клієнтського застосунку

3.2 Програмно-апаратні вимоги до клієнта

Клієнтська частина системи реалізується у вигляді настільного JavaFX-застосунку, тому вимоги до клієнтського середовища визначаються з урахуванням роботи на персональному комп’ютері диспетчера [15]. На відміну від мобільного застосунку, клієнтська частина не залежить від версії Android або характеристик смартфона. Основними умовами її роботи є наявність операційної системи, середовища виконання Java та доступу до мережі для обміну даними із сервером [14].

Мінімальною підтримуваною операційною системою доцільно визначити Windows 10 або новішу версію. Такий вибір пояснюється тим, що саме настільні комп’ютери під керуванням Windows найчастіше використовуються як робочі місця диспетчерів. Крім того, Windows 10 забезпечує стабільну роботу Java-застосунків, підтримку мережевих підключень та коректне відображення графічного інтерфейсу JavaFX [14, 15].

З точки зору апаратної платформи клієнтський застосунок не висуває високих вимог до обчислювальних ресурсів. Основні операції клієнта полягають у відображенні таблиць, обробці дій користувача, формуванні запитів до сервера та прийманні відповідей. Для коректної роботи достатньо персонального комп’ютера з двоядерним процесором із тактовою частотою від 2,0 ГГц.

Мінімально рекомендований обсяг оперативної пам’яті становить 4 ГБ. Цього достатньо для запуску операційної системи, середовища виконання Java та роботи клієнтського застосунку з основними таблицями даних. Для більш комфортної роботи рекомендовано використовувати 8 ГБ оперативної пам’яті, особливо якщо одночасно з клієнтським застосунком на комп’ютері відкриті інші програми.

Для встановлення та запуску клієнтської частини необхідно передбачити не менше 500 МБ вільного дискового простору. Основний обсяг займають файли застосунку, бібліотеки JavaFX та службові файли [15]. Оскільки клієнтська

частина не зберігає базу даних локально, значні обсяги дискового простору не потрібні.

Клієнтський застосунок орієнтований на постійну мережеву взаємодію із сервером. Для коректної роботи необхідна наявність підключення до локальної мережі або іншого мережевого середовища, у якому доступний серверний застосунок. Обсяг переданих даних є невеликим, оскільки між клієнтом і сервером передається структурована текстова інформація про транспорт, маршрути та призначення.

Для роботи клієнтської частини необхідно встановити середовище виконання Java, сумісне з використаною версією JavaFX [14, 15]. Клієнтський застосунок запускається як окрема програма та взаємодіє із сервером через мережеве з’єднання.

У лістингу 3.1 наведено приклад структури даних, яку сервер може передавати клієнтському застосунку для відображення інформації про транспортні засоби.

Лістинг 3.1 – Фрагмент структури даних, яку сервер передає клієнту

```
{
"transports": [
{
"id": 1,
"number": "AA 1234 BB",
"type": "Автобус",
"status": "Вільний"
},
{
"id": 2,
"number": "AA 5678 CC",
"type": "Тролейбус",
"status": "На маршруті"
}
]
}
```

Отримані дані використовуються клієнтським застосунком для заповнення таблиць інтерфейсу. Після приймання відповіді від сервера інформація перетворюється у відповідні об’єкти моделі та відображається у вікні JavaFX [15].

Такий підхід дозволяє не зберігати дані на стороні клієнта постійно, а отримувати актуальну інформацію із серверної частини системи.

3.3 Реалізація клієнтських модулів

Класи моделі в клієнтському застосунку є аналогічними класам моделі серверної частини. Такий підхід дозволяє використовувати однакову структуру даних під час їх передавання між компонентами системи. Після отримання відповіді від сервера клієнтський застосунок створює відповідні об’єкти класів Transport, Route та Assignment, які надалі використовуються для відображення інформації в графічному інтерфейсі [16].

Головним класом клієнтського застосунку є MainApplication. Саме він відповідає за запуск JavaFX-застосунку, створення головного вікна та ініціалізацію основних компонентів інтерфейсу [15]. Крім того, у цьому класі виконується завантаження початкового представлення користувацького інтерфейсу та підготовка середовища до роботи користувача.

Клас MainApplication виконує роль центральної точки входу до клієнтського застосунку. Після запуску програми створюється головне вікно, завантажується відповідний FXML-файл та ініціалізуються контролери, які надалі забезпечують взаємодію користувача із системою [15].

Фрагмент коду класу MainApplication наведено в лістингу 3.2.

Лістинг 3.2 – Фрагмент коду класу MainApplication

```
public class MainApplication extends Application {
@Override
public void start(Stage stage) throws Exception {
FXMLLoader loader =
new FXMLLoader(
getClass().getResource("main.fxml"));
Scene scene =
new Scene(loader.load());
stage.setTitle(
```

```
"Система диспетчера міського транспорту");
stage.setScene(scene);
stage.show();
}
public static void main(String[] args) {
launch(args);
}
}
```

У наведеному фрагменті коду реалізовано запуск клієнтського застосунку та створення головного вікна. Після завантаження FXML-файлу формується графічний інтерфейс користувача, який надалі використовується для взаємодії із серверною частиною системи [15].

Навігація між окремими частинами інтерфейсу реалізується за допомогою контролерів та FXML-представлень. Такий підхід дозволяє розділити логіку роботи програми та графічний інтерфейс користувача [15].

Клас MainApplication виконує роль центральної точки запуску клієнтського застосунку. Після старту програми він створює головне вікно та завантажує початкове представлення інтерфейсу. Подальша взаємодія користувача із системою здійснюється через відповідні контролери, які відповідають за роботу окремих екранів.

Для реалізації мережевої взаємодії використовується клас NetworkClient. Він відповідає за встановлення з’єднання із сервером, передавання запитів та отримання відповідей. Виконання мережевих операцій здійснюється в окремому потоці, що дозволяє уникнути блокування графічного інтерфейсу під час очікування відповіді від сервера.

Клас NetworkClient інкапсулює всю логіку обміну даними між клієнтом і сервером. Завдяки цьому контролери інтерфейсу не містять програмного коду роботи з сокетами та використовують лише готові методи отримання даних. Такий підхід спрощує супровід програмного забезпечення та забезпечує чіткий розподіл відповідальності між компонентами системи.

Фрагмент класу NetworkClient наведено в лістингу 3.3.

Лістинг 3.3 – Фрагмент класу NetworkClient

```
public class NetworkClient {
private final String host;
private final int port;
public NetworkClient(String host, int port) {
this.host = host;
this.port = port;
}
public String sendRequest(String request)
throws IOException {
try (
Socket socket =
new Socket(host, port);
PrintWriter writer =
new PrintWriter(
socket.getOutputStream(),
true);
BufferedReader reader =
new BufferedReader(
new InputStreamReader(
socket.getInputStream()))
) {
writer.println(request);
return reader.readLine();
}
}
}
```

У наведеному фрагменті коду реалізовано встановлення TCP-з’єднання із сервером та передавання текстового запиту. Після отримання відповіді дані повертаються викликаючому компоненту для подальшої обробки. Такий підхід дозволяє ізолювати мережеву взаємодію від інших частин клієнтського застосунку та спрощує використання мережевих функцій в контролерах інтерфейсу [17].

Поля host та port призначені для зберігання параметрів підключення до серверної частини системи. Вони визначають мережеву адресу сервера та порт, через який здійснюється обмін даними між клієнтом і сервером.

Конструктор класу NetworkClient виконує ініціалізацію параметрів підключення та забезпечує підготовку об’єкта до подальшої роботи. Такий підхід дозволяє відокремити параметри мережевого підключення від логіки обробки даних.

Метод sendRequest є основним методом класу. Під час його виконання створюється TCP-з’єднання із сервером, після чого клієнт надсилає текстовий запит через вихідний потік даних. Сервер обробляє отриманий запит та формує відповідь, яка повертається клієнту через вхідний потік.

Для роботи з мережевими ресурсами використовується конструкція try-with-resources, яка забезпечує автоматичне закриття сокета та потоків після завершення обміну даними. Це дозволяє уникнути витоків ресурсів та підвищує надійність роботи застосунку.

Отримана відповідь повертається викликаючому компоненту у вигляді рядка. Надалі ці дані можуть бути перетворені у відповідні об’єкти моделі та використані для відображення інформації в елементах графічного інтерфейсу.

Таким чином, клас NetworkClient реалізує клієнтську частину мережевої взаємодії та забезпечує обмін даними між користувацьким інтерфейсом і серверною частиною системи.

Після реалізації класу NetworkClient визначаємо структура контролерів клієнтської частини. Оскільки застосунок реалізується з використанням JavaFX, взаємодія між графічним інтерфейсом і програмною логікою організовується через контролери. Кожен контролер відповідає за окрему частину інтерфейсу та обробляє дії користувача в межах відповідного вікна.

У клієнтській частині системи передбачено такі основні контролери:

– MainController відповідає за головне вікно застосунку та перехід до основних розділів системи;

– TransportController забезпечує відображення списку транспортних засобів;

– RouteController відповідає за відображення маршрутів;

– AssignmentController використовується для створення призначення транспорту на маршрут та зміни його статусу;

– StatisticsController забезпечує відображення узагальненої статистичної інформації.

Такий поділ відповідає структурі клієнтського інтерфейсу, визначеній на етапі проєктування. Кожен контролер має окрему зону відповідальності, що дозволяє не змішувати логіку різних екранів в одному класі та спрощує подальший супровід програмного забезпечення.

Першим реалізується клас MainController, який є основним контролером головного вікна клієнтського застосунку. Він забезпечує обробку натискань навігаційних кнопок та виконує перехід до відповідних розділів системи.

Фрагмент коду класу MainController наведено в лістингу 3.4.

Лістинг 3.4 – Фрагмент коду класу MainController

```
public class MainController {
@FXML
private Button transportButton;
@FXML
private Button routeButton;
@FXML
private Button assignmentButton;
@FXML
private Button statisticsButton;
@FXML
private void openTransport() {
System.out.println("Transport section opened");
}
@FXML
private void openRoutes() {
System.out.println("Routes section opened");
}
@FXML
private void openAssignments() {
System.out.println("Assignments section opened");
}
@FXML
private void openStatistics() {
System.out.println("Statistics section opened");
}
}
```

У наведеному фрагменті коду показано зв’язок елементів головного вікна з методами контролера. Поля з анотацією @FXML відповідають кнопкам інтерфейсу, описаним у FXML-файлі [15]. Методи openTransport, openRoutes,

openAssignments та openStatistics викликаються після натискання відповідних кнопок і забезпечують перехід користувача до потрібного розділу системи.

На цьому етапі в лістингу наведено базову логіку головного контролера. Подальша деталізація переходів між вікнами виконується під час реалізації відповідних контролерів і FXML-представлень [15].

Після реалізації головного контролера розробляється контролер перегляду транспортних засобів. Клас TransportController відповідає за отримання даних від серверної частини та відображення їх у таблиці користувацького інтерфейсу.

Основними елементами інтерфейсу цього контролера є таблиця транспортних засобів та стовпці, що відображають номер транспорту, його тип і поточний стан. Під час ініціалізації контролер виконує заповнення таблиці отриманими даними.

Фрагмент коду класу TransportController наведено в лістингу 3.5.

Лістинг 3.5 – Фрагмент коду класу TransportController

```
public class TransportController {
@FXML
private TableView<Transport> transportTable;
@FXML
private TableColumn<Transport, Integer> idColumn;
@FXML
private TableColumn<Transport, String> numberColumn;
@FXML
private TableColumn<Transport, String> typeColumn;
@FXML
private TableColumn<Transport, String> statusColumn;
@FXML
public void initialize() {
idColumn.setCellValueFactory(
new PropertyValueFactory<>("id"));
numberColumn.setCellValueFactory(
new PropertyValueFactory<>("number"));
typeColumn.setCellValueFactory(
new PropertyValueFactory<>("type"));
statusColumn.setCellValueFactory(
new PropertyValueFactory<>("status"));
}
public void setTransports(
List<Transport> transports) {
transportTable.getItems().clear();
transportTable.getItems().addAll(
```

```
transports);
}
}
```

У наведеному фрагменті коду реалізовано контролер відображення транспортних засобів. Таблиця transportTable використовується для подання інформації про транспорт у табличному вигляді. Під час ініціалізації контролера виконується прив’язка стовпців таблиці до відповідних полів класу Transport. Метод setTransports призначений для заповнення таблиці даними, отриманими від серверної частини системи.

Аналогічним чином реалізовано контролери RouteController, AssignmentController та StatisticsController. Вони використовують відповідні модельні класи та забезпечують відображення інформації у графічному інтерфейсі клієнтського застосунку.

Після реалізації контролерів виконується створення FXML-представлень, які описують структуру вікон клієнтського застосунку. У JavaFX FXML-файл використовується для декларативного опису елементів інтерфейсу, а відповідний контролер забезпечує їхню програмну обробку [15].

Такий підхід дозволяє відокремити зовнішній вигляд інтерфейсу від логіки роботи застосунку. Це спрощує внесення змін до розташування елементів, оскільки структура вікна може редагуватися без зміни основного програмного коду.

Фрагмент FXML-розмітки вікна перегляду транспорту наведено в лістингу 3.6.

Лістинг 3.6 – Фрагмент FXML-розмітки вікна транспорту

```
<VBox spacing="10" xmlns:fx="http://javafx.com/fxml"
fx:controller="controller.TransportController">
<Label text="Транспортні засоби" />
<TableView fx:id="transportTable">
<columns>
<TableColumn fx:id="idColumn" text="ID" />
<TableColumn fx:id="numberColumn" text="Номер" />
<TableColumn fx:id="typeColumn" text="Тип" />
<TableColumn fx:id="statusColumn" text="Статус" />
```

```
</columns>
</TableView>
</VBox>
```

У наведеному фрагменті описано вікно перегляду транспортних засобів. Основним елементом інтерфейсу є таблиця TableView, яка містить стовпці для відображення ідентифікатора, номера, типу та статусу транспорту. Атрибути fx:id пов’язують елементи FXML-розмітки з відповідними полями класу TransportController.

Аналогічним способом створюються FXML-представлення для вікон маршрутів, призначень та статистики. Кожне представлення має власний контролер і відповідає окремому сценарію роботи диспетчера.

3.4 Реалізація інтерфейсу

Після реалізації програмних модулів клієнтської частини було створено графічний інтерфейс користувача. Основною вимогою до інтерфейсу була простота використання та швидкий доступ до основних функцій системи. Усі операції, необхідні диспетчеру для роботи з транспортом і маршрутами, виконуються через декілька взаємопов’язаних вікон.

Головне вікно застосунку відображається після запуску програми та надає доступ до основних розділів системи. З нього користувач може перейти до перегляду транспортних засобів, маршрутів, призначень або статистичної інформації.

Зовнішній вигляд головного вікна наведено на рисунку 3.4.

На рисунку 3.4 наведено головний екран системи диспетчера міського транспорту. З нього користувач отримує доступ до основних функціональних модулів застосунку, зокрема до перегляду транспортних засобів, маршрутів, призначень та статистичної інформації. Навігація між розділами здійснюється за

допомогою меню, розташованого в лівій частині вікна. Така організація інтерфейсу забезпечує швидкий доступ до основних функцій системи та спрощує роботу диспетчера під час щоденної експлуатації програмного забезпечення.

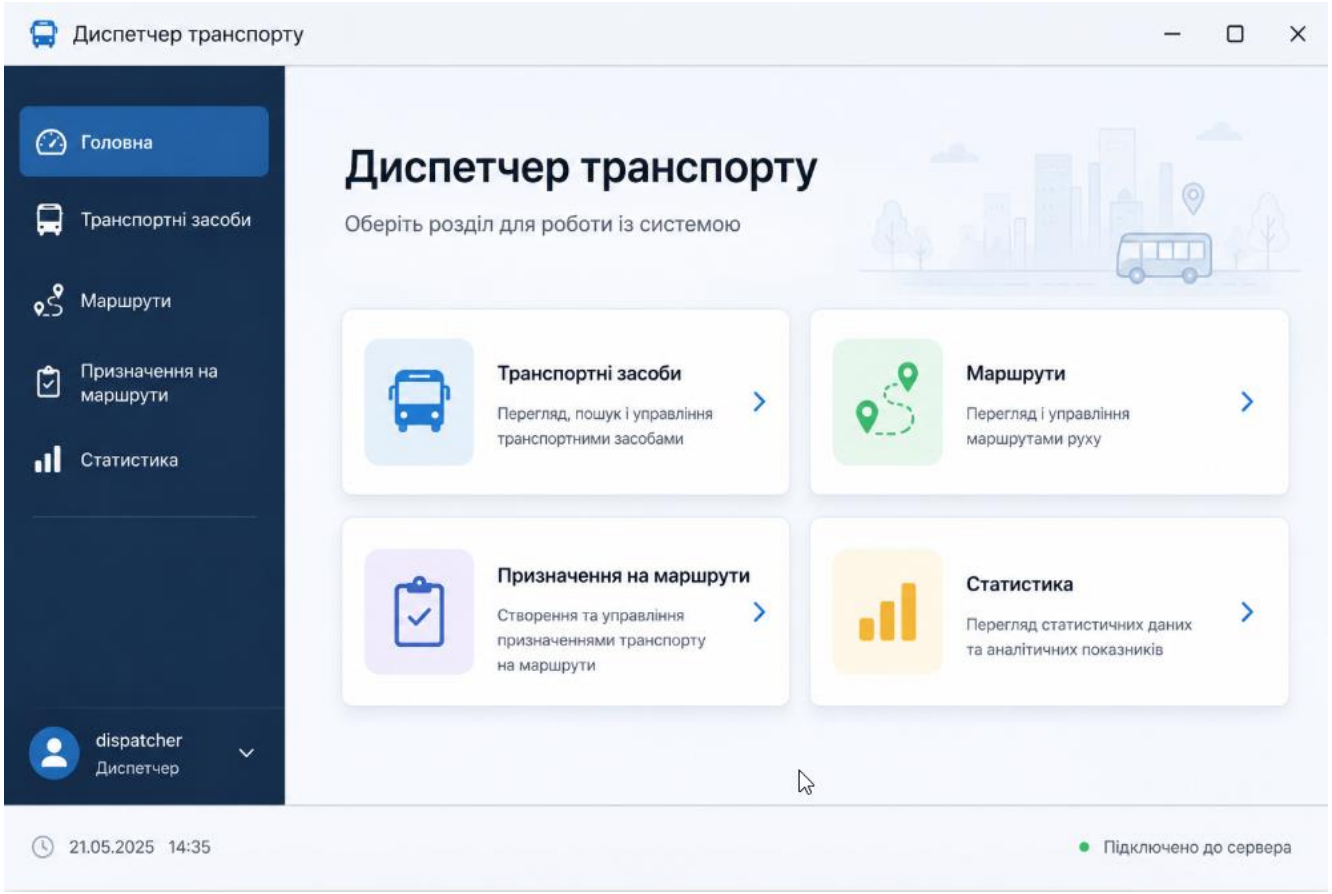

Рисунок 3.4 – Головний екран системи диспетчера міського транспорту

На рисунку 3.5 наведено вікно перегляду транспортних засобів. Воно призначене для отримання та відображення інформації про транспорт, що використовується в системі диспетчеризації. Основним елементом інтерфейсу є таблиця, у якій подано перелік транспортних засобів із їх основними характеристиками.

Для кожного запису відображається унікальний ідентифікатор, реєстраційний номер, тип транспортного засобу та його поточний статус. Така структура подання даних дозволяє диспетчеру швидко отримувати інформацію про наявний транспорт та його поточний стан.

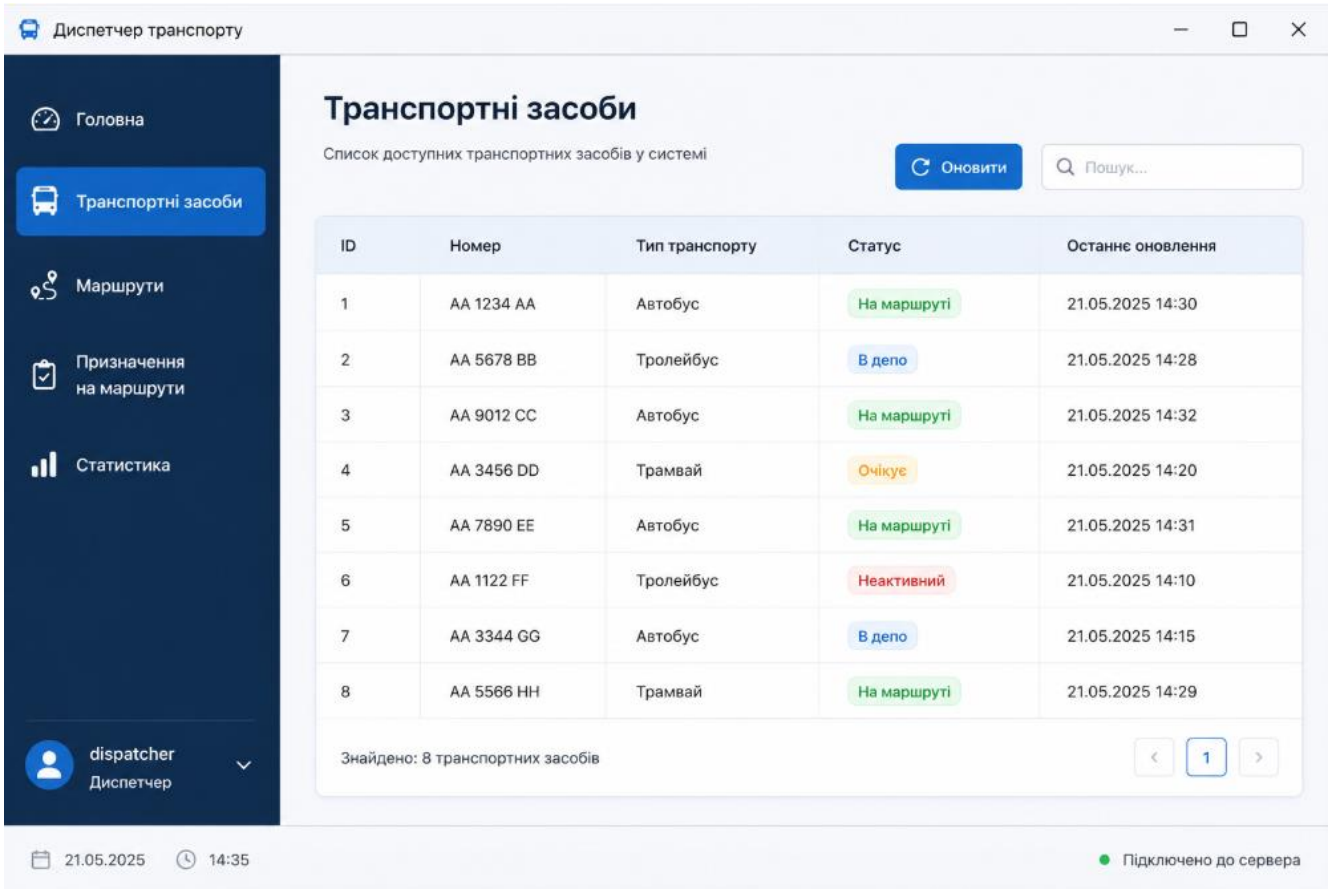

Рисунок 3.5 – Вікно перегляду транспортних засобів

У верхній частині вікна розташовано поле пошуку та кнопку оновлення даних. Пошук використовується для швидкого знаходження потрібного транспортного засобу за його номером або іншими параметрами. Кнопка оновлення дозволяє повторно отримати актуальні дані із серверної частини системи без перезапуску застосунку.

Ліва частина вікна містить навігаційне меню, за допомогою якого користувач може перейти до інших функціональних розділів системи. Такий підхід забезпечує єдиний стиль інтерфейсу для всіх вікон клієнтського застосунку та спрощує навігацію між окремими модулями.

Після реалізації перегляду транспортних засобів було реалізовано модуль роботи з маршрутами, який забезпечує відображення маршрутної мережі та використовується під час створення призначень транспорту.

Зовнішній вигляд вікна перегляду маршрутів наведено на рисунку 3.6.

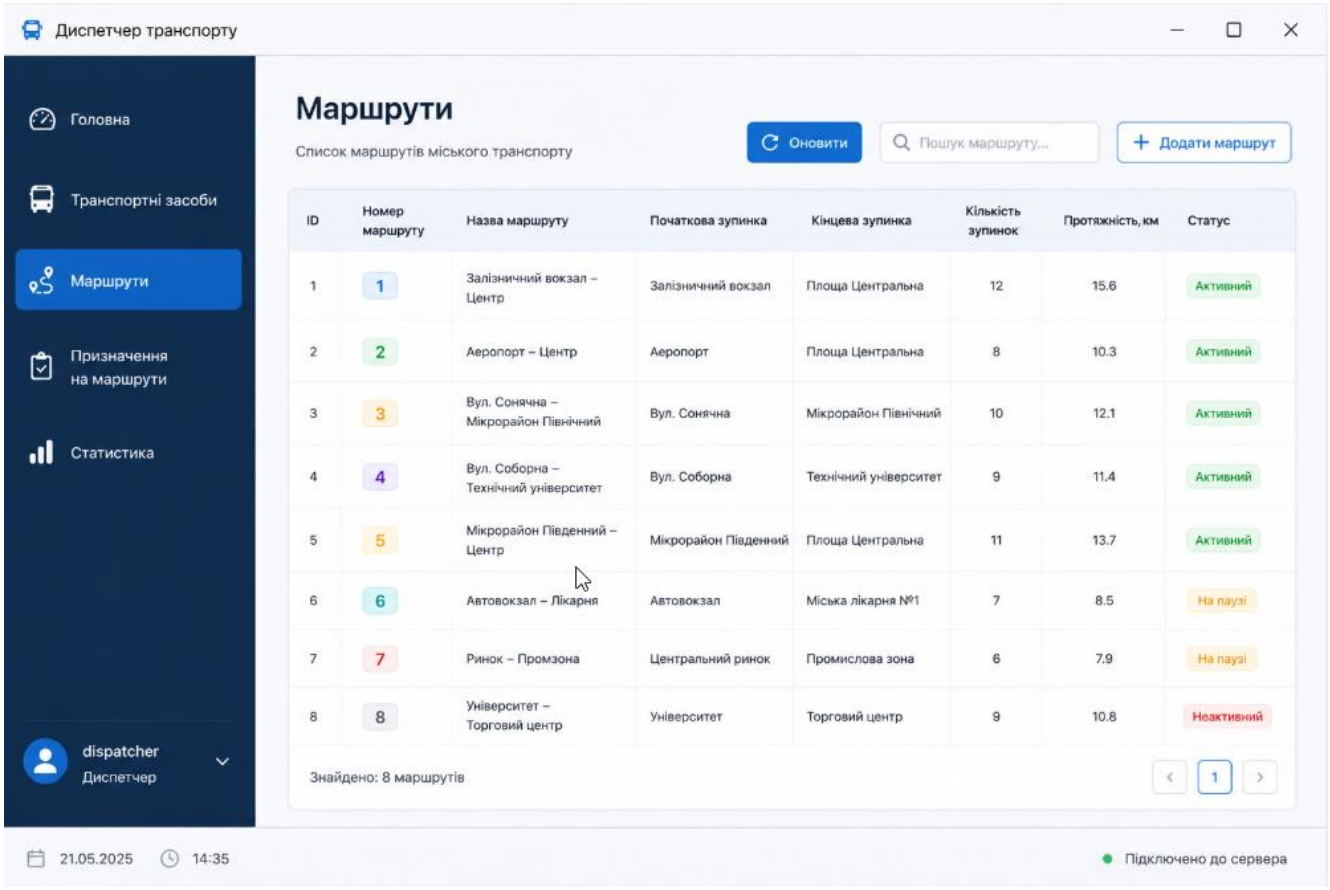

Рисунок 3.6 – Вікно перегляду маршрутів

На рисунку 3.6 наведено вікно перегляду маршрутів. Воно призначене для відображення маршрутної мережі, яка використовується в системі диспетчеризації міського транспорту. Основним елементом інтерфейсу є таблиця маршрутів, що містить інформацію про номер маршруту, його назву, початкову та кінцеву зупинки, кількість зупинок, протяжність та поточний статус.

Для зручності роботи користувача передбачено поле пошуку, яке дозволяє швидко знаходити потрібний маршрут за його номером або назвою. Кнопка оновлення використовується для отримання актуальної інформації із серверної частини системи.

Відображення маршрутів у табличному вигляді забезпечує швидкий доступ до необхідної інформації та спрощує роботу диспетчера під час планування транспортних перевезень і контролю стану маршрутної мережі.

Після реалізації модулів перегляду транспорту та маршрутів було реалізовано вікно створення призначень транспорту на маршрути, яке забезпечує основну функціональність системи диспетчеризації.

Зовнішній вигляд вікна призначення транспорту на маршрут наведено на рисунку 3.7.

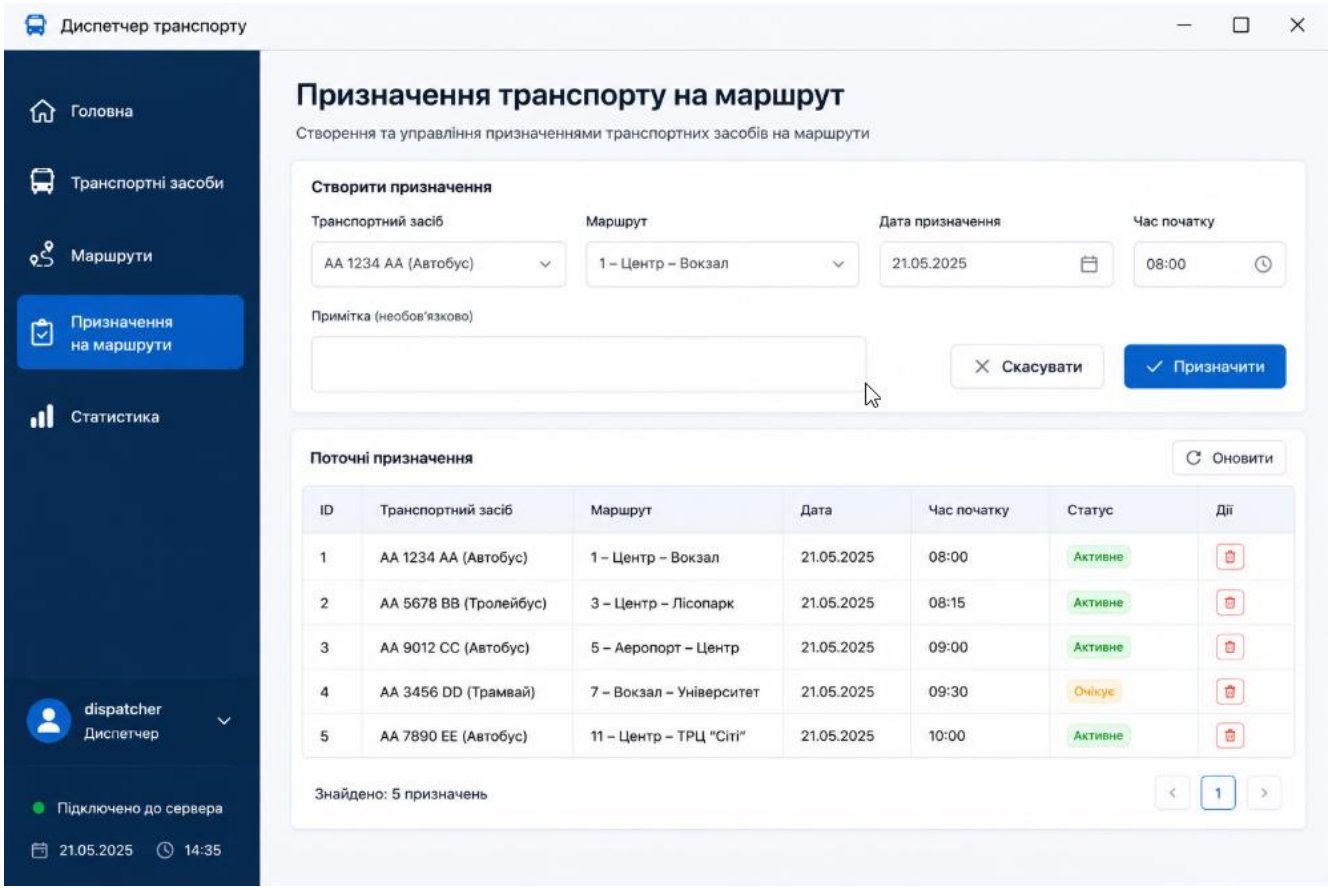

Рисунок 3.7 – Вікно призначення транспорту на маршрут

На рисунку 3.7 наведено вікно призначення транспорту на маршрут. Саме цей модуль забезпечує основну функціональність системи диспетчеризації, оскільки дозволяє пов’язувати транспортні засоби з маршрутами та формувати графік їх роботи.

У верхній частині вікна розташована форма створення нового призначення. Для формування запису диспетчер обирає транспортний засіб та маршрут зі списків, після чого вказує дату і час початку виконання рейсу. За необхідності може бути додана додаткова службова примітка. Після підтвердження введених

даних інформація передається до серверної частини системи та зберігається в базі даних.

Нижня частина вікна містить таблицю поточних призначень. Для кожного запису відображається транспортний засіб, маршрут, дата виконання, час початку роботи та поточний статус призначення. Такий спосіб подання інформації дозволяє диспетчеру швидко контролювати стан транспортних засобів та їх розподіл між маршрутами.

Для підтримання актуальності інформації передбачено можливість оновлення даних без перезапуску застосунку. Отримання актуальних відомостей виконується шляхом взаємодії із серверною частиною системи через мережеве з’єднання.

Реалізація вікна призначень завершує формування основного функціоналу клієнтського застосунку. Для отримання узагальненої інформації про роботу системи передбачено окремий модуль статистики.

Зовнішній вигляд вікна статистики наведено на рисунку 3.8.

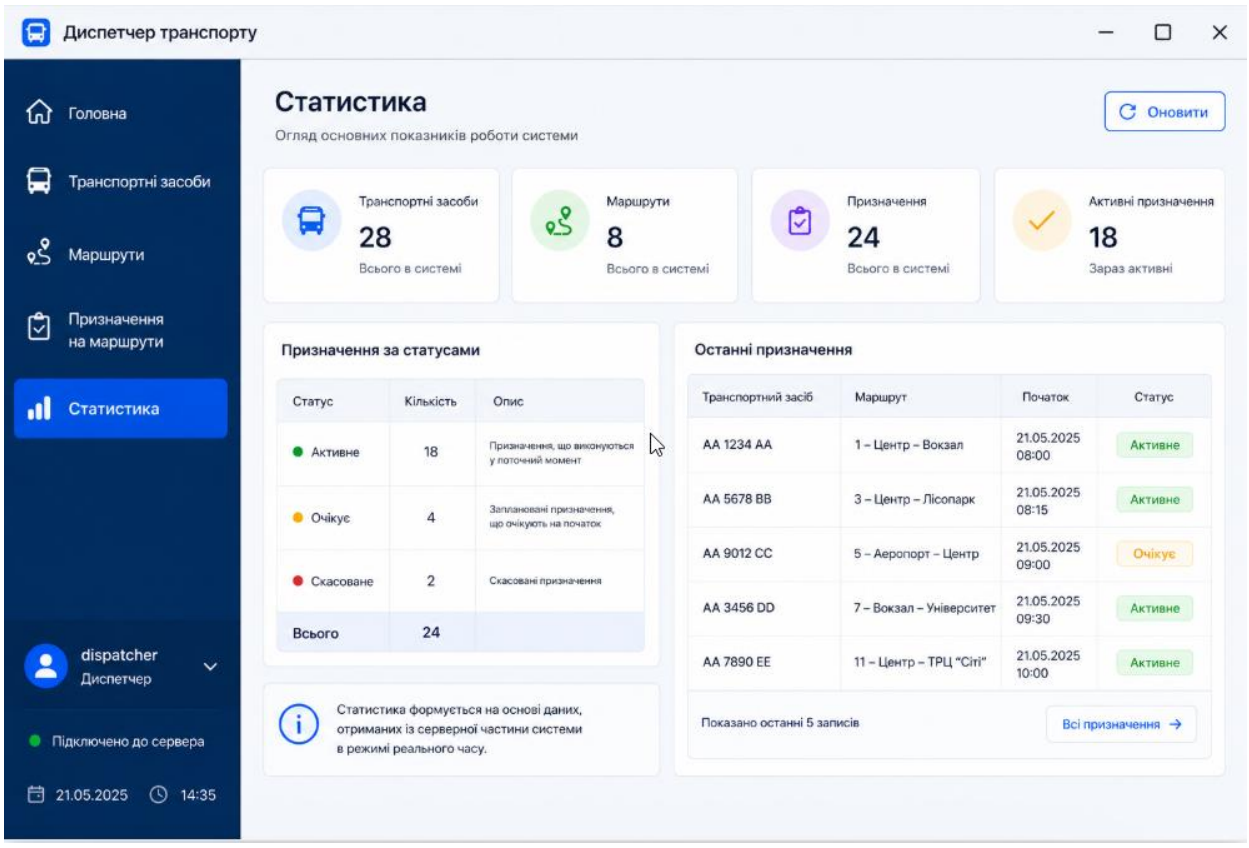

Рисунок 3.8 – Вікно статистики

Вікно статистики призначене для отримання узагальненої інформації про поточний стан системи диспетчеризації міського транспорту.

У верхній частині вікна відображаються основні показники роботи системи, зокрема кількість транспортних засобів, маршрутів, створених призначень та активних призначень. Ці показники дозволяють диспетчеру швидко оцінити поточний стан транспортної мережі без переходу до окремих розділів застосунку.

Для більш детального аналізу інформації передбачено таблицю розподілу призначень за статусами. Вона відображає кількість активних, запланованих та скасованих призначень, що дозволяє контролювати процес розподілу транспорту між маршрутами.

У правій частині вікна розташована таблиця останніх призначень, яка містить інформацію про транспортний засіб, маршрут, дату та поточний статус призначення. Це забезпечує оперативний доступ до актуальних даних та спрощує контроль за роботою системи.

Таким чином, модуль статистики надає користувачу зведену інформацію про роботу системи та дозволяє швидко отримувати відомості, необхідні для прийняття диспетчерських рішень.

3.5 Висновки по розділу 3

У третьому розділі виконано проєктування та реалізацію клієнтської частини системи диспетчера міського транспорту. На початковому етапі було розроблено архітектуру клієнтського застосунку, визначено основні сценарії взаємодії користувача із системою та сформовано структуру графічного інтерфейсу.

Для реалізації клієнтської частини обрано технологію JavaFX, що дозволило створити повноцінний настільний застосунок із графічним інтерфейсом користувача [15]. Визначено програмно-апаратні вимоги до клієнтського

робочого місця та описано особливості його мережевої взаємодії із серверною частиною системи.

У процесі розробки реалізовано основні програмні модулі клієнтського застосунку, зокрема клас запуску програми, модуль мережевої взаємодії та контролери інтерфейсу користувача. Для побудови графічного інтерфейсу використано FXML-представлення, що забезпечують розділення логіки роботи застосунку та його візуальної складової [15].

Також реалізовано основні екрани клієнтського застосунку: головний екран системи, вікно перегляду транспортних засобів, вікно перегляду маршрутів, вікно призначення транспорту на маршрути та вікно статистики. Розроблений інтерфейс забезпечує доступ до всіх основних функцій системи та створює зручні умови для роботи диспетчера.

Отримані результати підтверджують можливість використання розробленого клієнтського застосунку як складової частини комп’ютерної системи диспетчеризації міського транспорту та забезпечують основу для подальшого тестування і оцінювання роботи програмного забезпечення.

4 ВИПРОБУВАННЯ СИСТЕМИ

4.1 Випробування користувацького інтерфейсу

Одним із найважливіших етапів перевірки працездатності системи є тестування користувацького інтерфейсу. Оскільки взаємодія диспетчера із системою здійснюється через графічний інтерфейс, необхідно переконатися в коректності роботи всіх елементів керування, навігації між вікнами та відображення інформації.

Під час випробувань перевірялися такі сценарії використання:

– запуск клієнтського застосунку;

– відкриття головного екрана системи;

– перегляд списку транспортних засобів;

– перегляд маршрутів;

– створення нового призначення транспорту на маршрут;

– перегляд статистичної інформації;

– отримання та відображення даних із серверної частини системи;

– коректність навігації між вікнами застосунку.

На рисунку 4.1 наведено головний екран системи під час проведення випробувань. В процесі тестування було перевірено коректність відображення елементів інтерфейсу, роботу навігаційного меню та можливість переходу між основними функціональними модулями системи. Під час запуску застосунку помилок відображення або порушень роботи інтерфейсу не було виявлено.

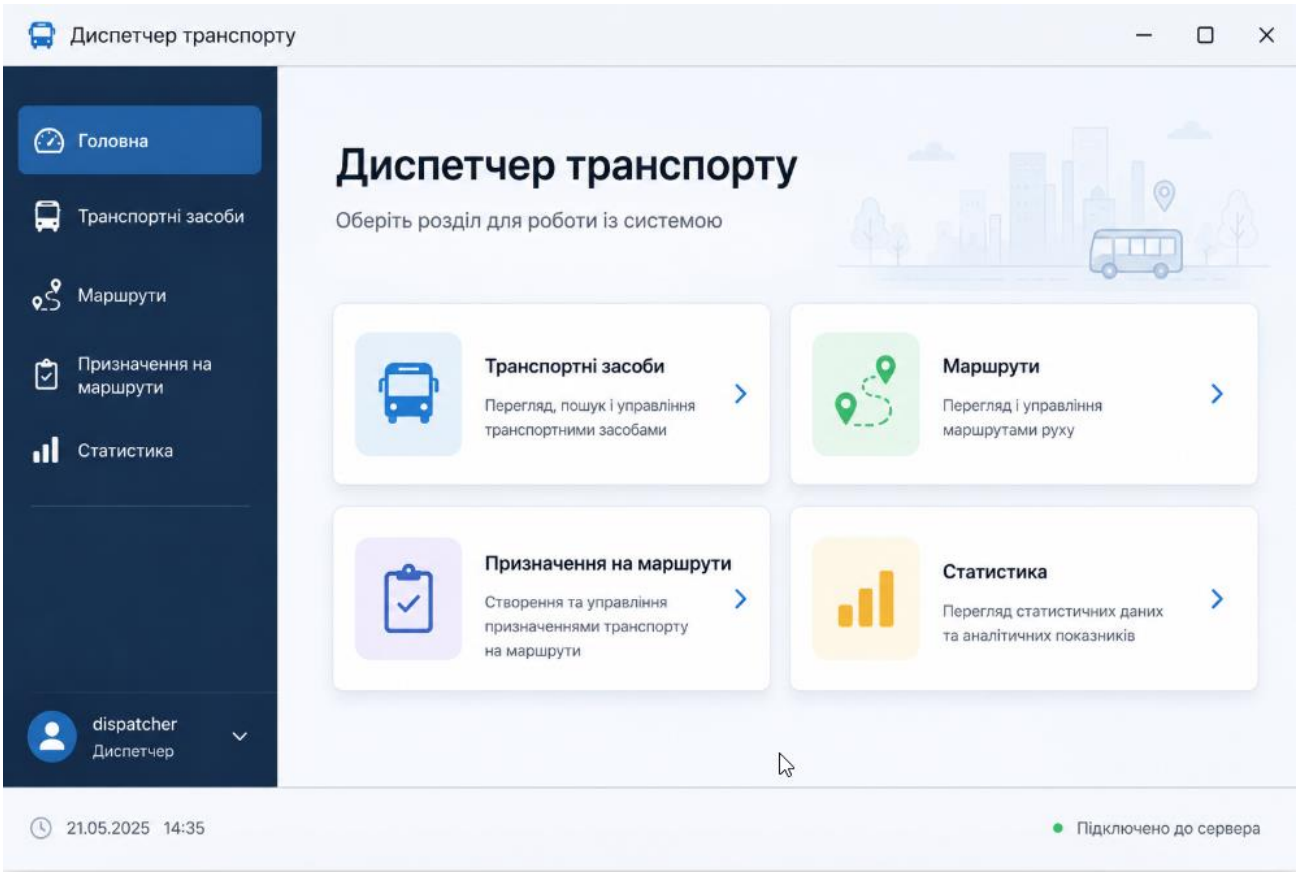

Рисунок 4.1 – Головний екран системи під час випробувань

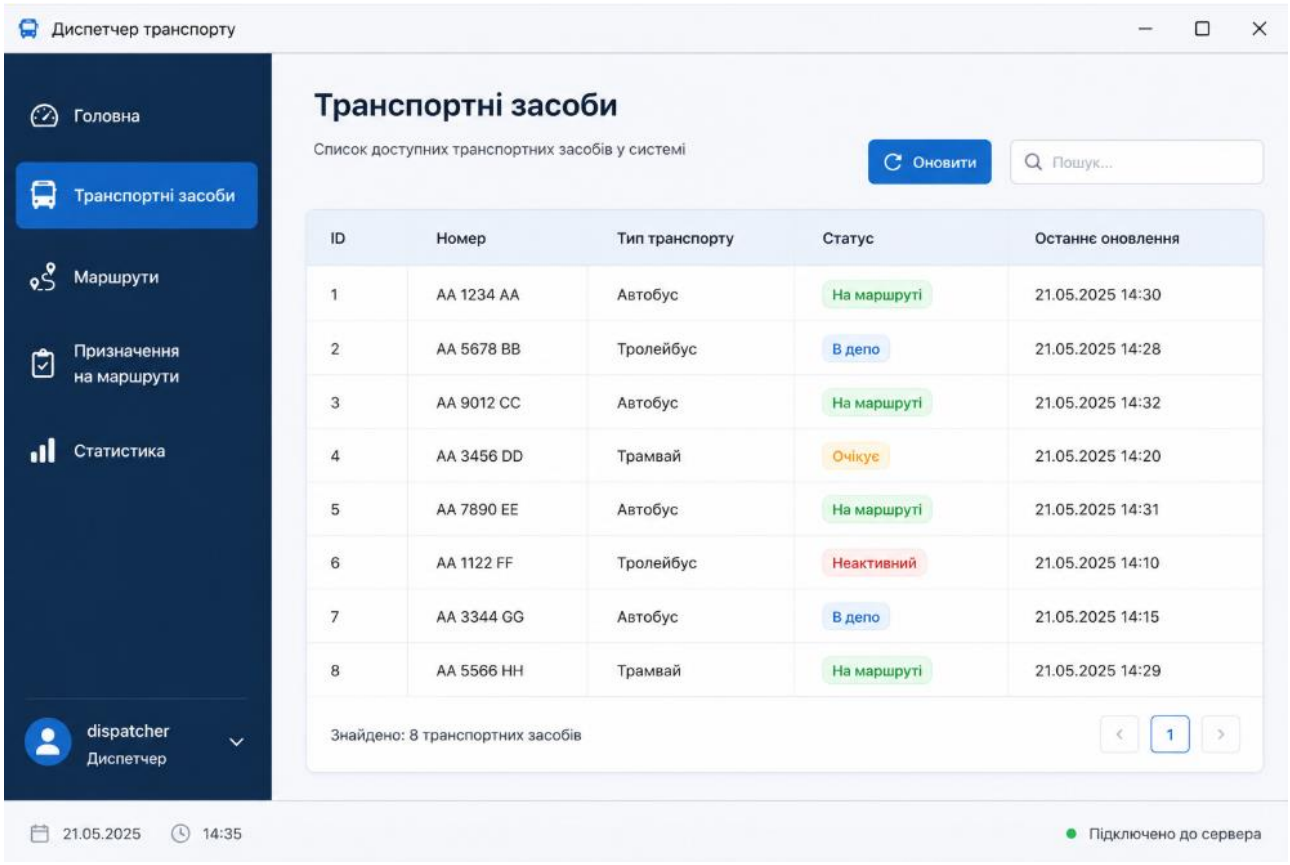

Рисунок 4.2 – Вікно перегляду транспортних засобів під час випробувань

На рисунку 4.2 наведено вікно перегляду транспортних засобів під час проведення випробувань. Основною метою тестування даного модуля була перевірка коректності отримання інформації із серверної частини системи та її відображення у графічному інтерфейсі клієнтського застосунку.

Під час випробувань перевірялася правильність завантаження списку транспортних засобів, відображення даних у табличному вигляді та робота елементів керування інтерфейсом. Особливу увагу було приділено коректності відображення ідентифікаторів транспортних засобів, реєстраційних номерів, типів транспорту та їх поточних статусів.

Окремо було перевірено роботу механізму пошуку. В результаті тестування встановлено, що введення пошукового запиту дозволяє швидко знаходити необхідні записи без помилок та затримок у роботі інтерфейсу. Також було протестовано кнопку оновлення даних, яка забезпечує повторне отримання інформації із сервера та актуалізацію вмісту таблиці.

За результатами проведених випробувань встановлено, що модуль перегляду транспортних засобів коректно виконує покладені на нього функції, забезпечує відображення актуальних даних та може використовуватися в процесі експлуатації системи.

Результати тестування модуля роботи з маршрутами наведено на рисунку 4.3.

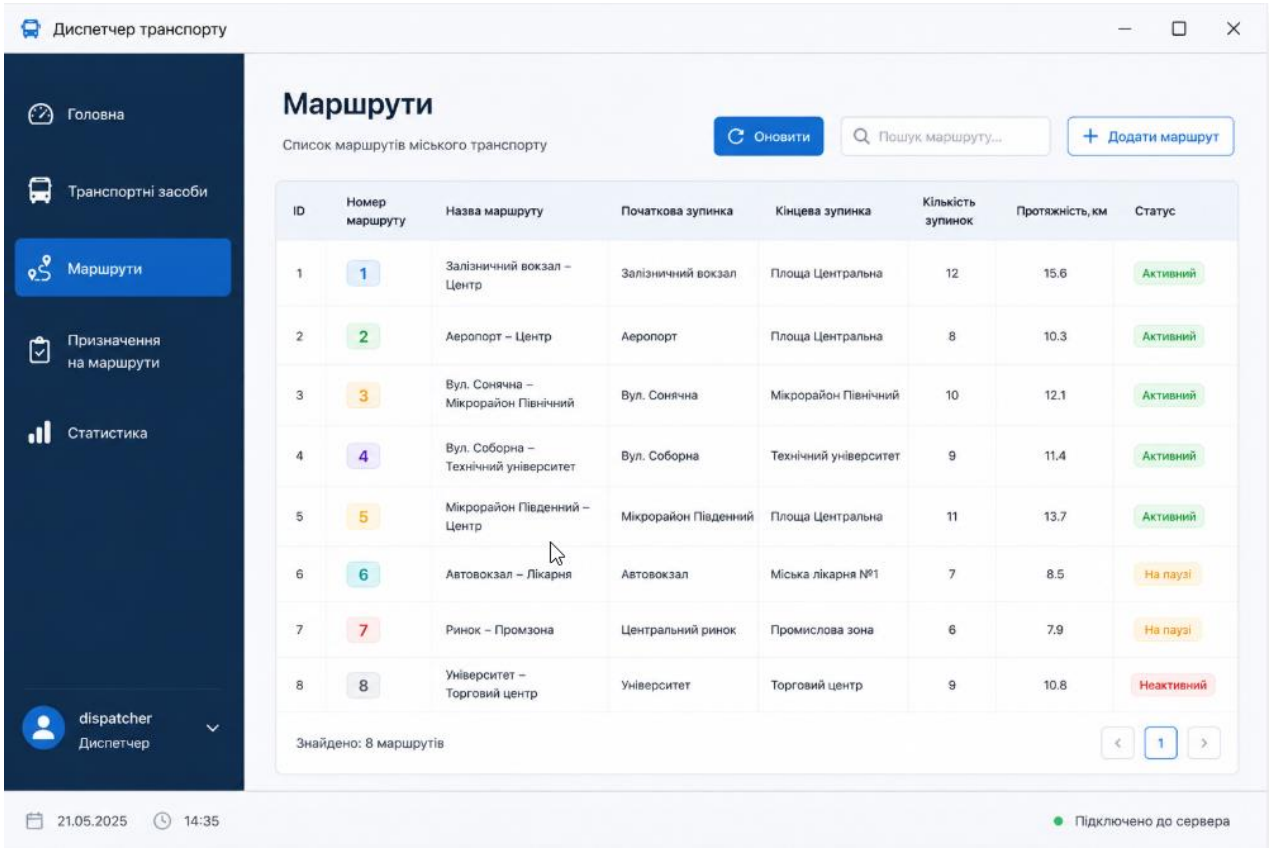

Рисунок 4.3 – Вікно перегляду маршрутів під час випробувань

На рисунку 4.3 наведено вікно перегляду маршрутів під час проведення випробувань. Тестування даного модуля виконувалося з метою перевірки коректності отримання інформації про маршрути із серверної частини системи та її подальшого відображення в інтерфейсі користувача.

У процесі випробувань було перевірено завантаження переліку маршрутів, відображення їх основних характеристик у таблиці та роботу елементів навігації. Особливу увагу приділено правильності відображення номерів маршрутів, початкових і кінцевих зупинок, а також поточних статусів маршрутів.

Також було перевірено функціонування механізму пошуку маршрутів та оновлення інформації. Результати тестування показали, що клієнтський застосунок коректно обробляє дані, отримані від сервера, та забезпечує їх своєчасне відображення без помітних затримок у роботі інтерфейсу.

Під час випробувань помилок завантаження або відображення інформації не виявлено. Усі функції модуля працювали відповідно до визначених вимог.

Результати тестування модуля призначення транспорту на маршрути наведено на рисунку 4.4.

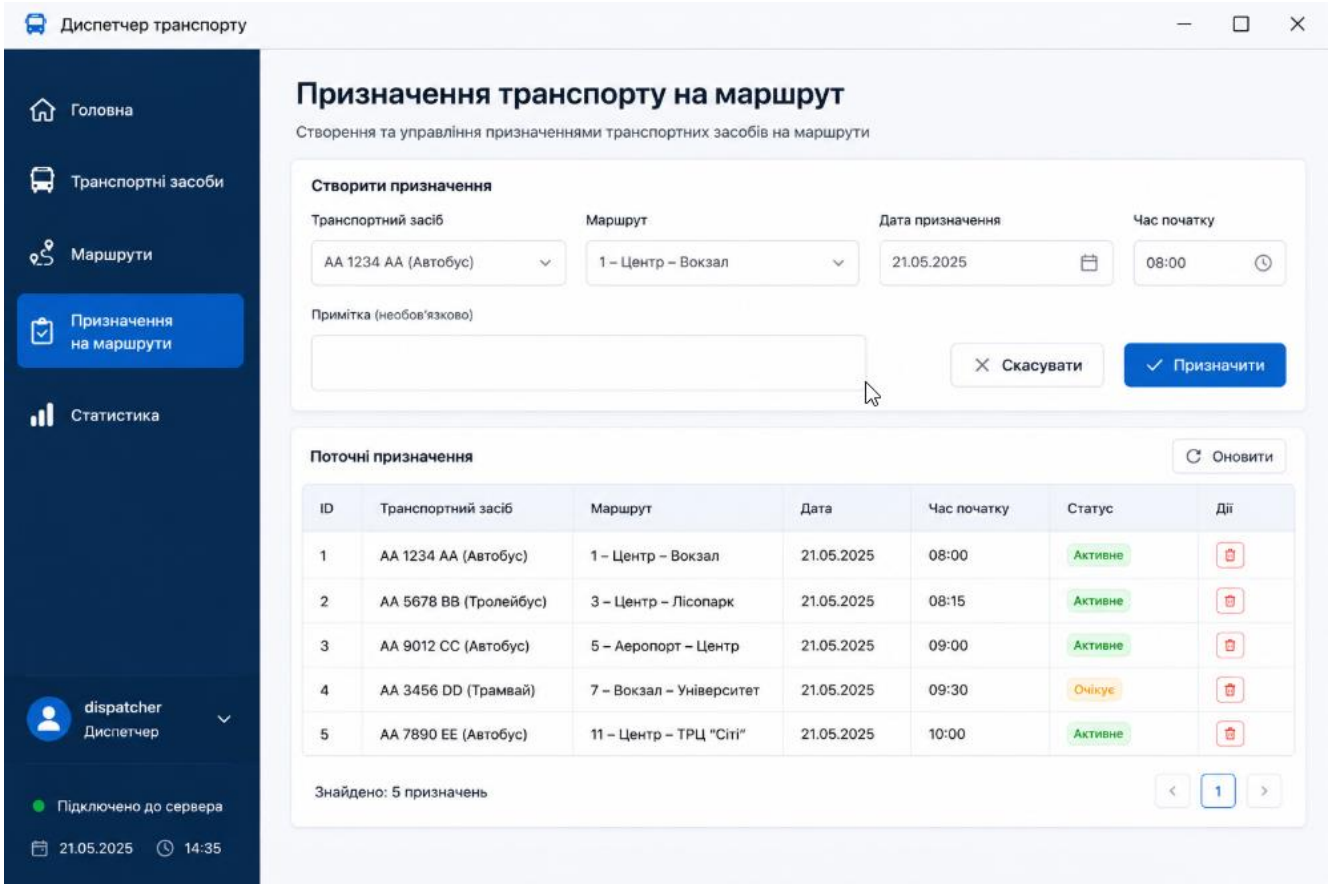

Рисунок 4.4 – Результати тестування модуля призначення транспорту на маршрути

Основною метою тестування даного модуля була перевірка коректності створення нових призначень, взаємодії із серверною частиною системи та відображення актуальної інформації у графічному інтерфейсі.

У процесі випробувань перевірялася робота елементів введення даних, зокрема вибір транспортного засобу, вибір маршруту, введення дати та часу призначення. Також було перевірено коректність передачі введених даних на сервер та подальше збереження інформації у базі даних.

Окрему увагу приділено перевірці відображення списку створених призначень. У результаті тестування встановлено, що після створення нового призначення інформація коректно відображається в таблиці без необхідності перезапуску застосунку. Також було перевірено роботу механізму оновлення даних та відображення статусів призначень.

Результати випробувань показали стабільну роботу модуля та коректне виконання всіх основних операцій, пов’язаних із призначенням транспортних засобів на маршрути.

Результати тестування модуля статистики наведено на рисунку 4.5.

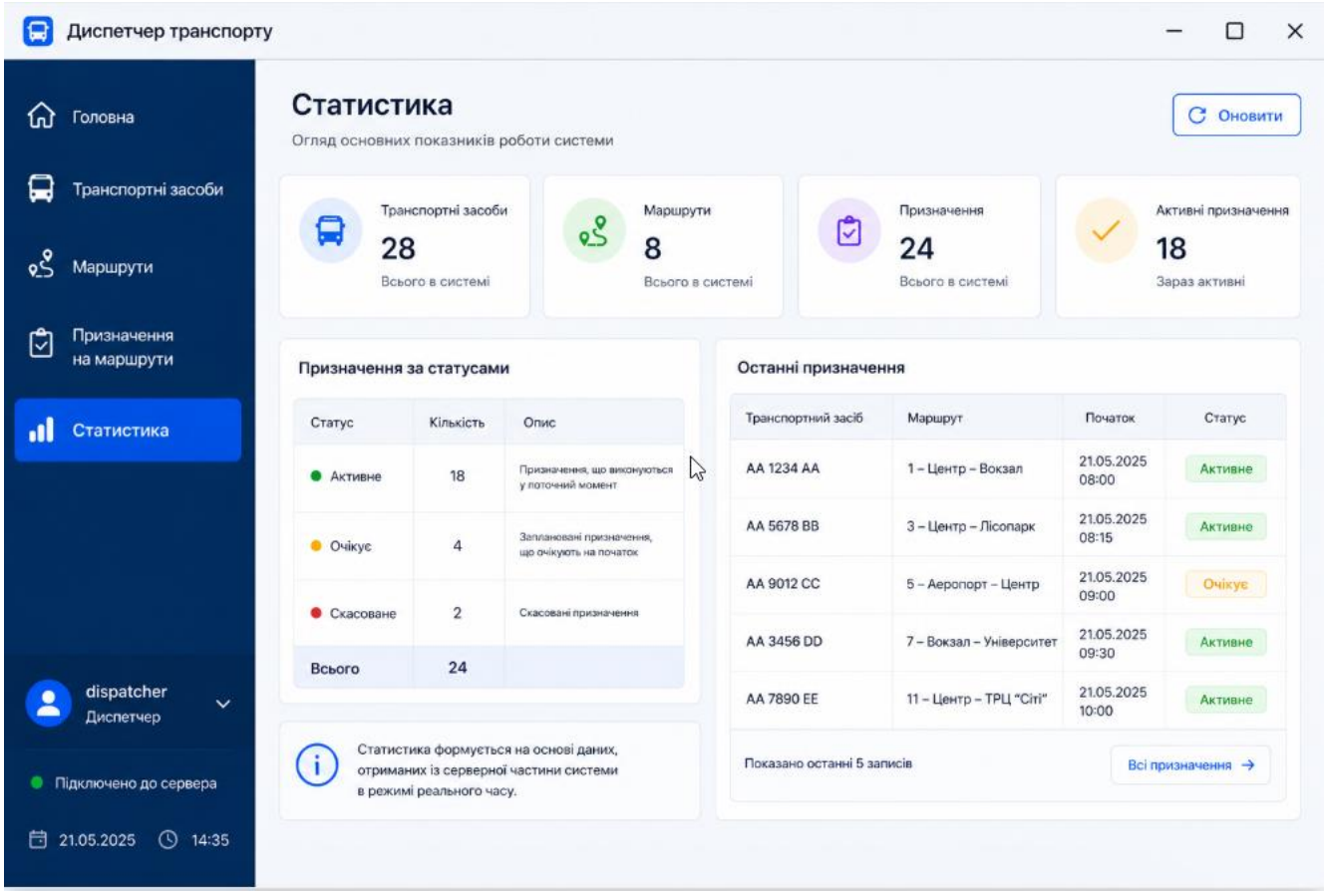

Рисунок 4.5 – Вікно статистики під час випробувань

Метою тестування даного модуля була перевірка коректності формування та відображення інформації про роботу системи диспетчеризації міського транспорту.

У процесі випробувань було перевірено правильність обчислення та відображення основних статистичних показників, зокрема кількості транспортних засобів, маршрутів, створених призначень та активних призначень. Особлива увага приділялася відповідності відображених значень фактичним даним, що зберігаються у базі даних системи.

Також було перевірено роботу таблиць статистичної інформації та коректність відображення відомостей про поточний стан призначень. Під час оновлення даних застосунок успішно отримував актуальну інформацію із серверної частини та відображав її без помилок і затримок.

Результати випробувань підтвердили працездатність модуля статистики та коректність формування зведеної інформації на основі даних, отриманих із бази даних. Усі елементи інтерфейсу функціонували відповідно до поставлених вимог, а відображені статистичні показники відповідали поточному стану системи.

Таким чином, проведені випробування користувацького інтерфейсу показали коректну роботу всіх основних модулів клієнтського застосунку та підтвердили можливість його використання для роботи з системою диспетчеризації міського транспорту.

4.2 Аналіз продуктивності системи

Після перевірки функціональності користувацького інтерфейсу було виконано оцінювання роботи розробленої системи під час виконання основних користувацьких операцій. Аналіз проводився з метою перевірки коректності взаємодії клієнтської та серверної частин системи, а також оцінювання швидкості виконання типових операцій.

Під час оцінювання розглядалися такі сценарії роботи:

– запуск клієнтського застосунку;

– підключення до серверної частини системи;

– отримання списку транспортних засобів;

– отримання списку маршрутів;

– створення нового призначення транспорту на маршрут;

– оновлення статистичної інформації.

У процесі випробувань перевірялася коректність обміну даними між клієнтом і сервером, а також відображення отриманої інформації у графічному інтерфейсі користувача. Особлива увага приділялася роботі мережевої взаємодії та виконанню операцій доступу до бази даних.

Клієнтська частина системи реалізована на основі технології JavaFX, а серверна частина забезпечує обробку запитів та взаємодію з базою даних PostgreSQL. Така архітектура дозволяє розділити функції зберігання, обробки та відображення інформації між окремими компонентами програмної системи.

Таблиця 4.1 – Результати перевірки основних операцій системи

Операція	Результат
Запуск клієнтського застосунку	Виконується успішно
Підключення до сервера	Виконується успішно
Отримання списку транспортних засобів	Виконується успішно
Отримання списку маршрутів	Виконується успішно
Створення призначення транспорту	Виконується успішно
Оновлення статистичної інформації	Виконується успішно

Перевірка показала коректне виконання всіх основних операцій системи. Не було виявлено помилок отримання даних із серверної частини, порушень роботи користувацького інтерфейсу або втрати інформації під час взаємодії з базою даних.

Отримані результати підтверджують працездатність реалізованої клієнт-серверної архітектури та коректну взаємодію між основними компонентами програмної системи.

4.3 Висновки по розділу 4

За результатами проведених випробувань користувацького інтерфейсу та аналізу роботи системи можна зробити висновок про коректне функціонування основних компонентів розробленого програмного забезпечення.

В процесі тестування було перевірено роботу модулів перегляду транспортних засобів, маршрутів, призначень та статистичної інформації. Проведені випробування підтвердили правильність відображення даних у графічному інтерфейсі та коректність навігації між окремими вікнами клієнтського застосунку.

Також було перевірено взаємодію клієнтської та серверної частин системи. Результати випробувань показали коректне отримання даних із бази даних, їх передавання мережею та подальше відображення в інтерфейсі користувача.

Під час виконання тестових сценаріїв не було виявлено помилок, які б перешкоджали роботі основних функцій системи. Отримані результати підтверджують працездатність розробленої системи диспетчера міського транспорту та можливість її подальшого використання для автоматизації роботи диспетчерських служб.

ВИСНОВКИ

В межах дипломної роботи було розроблено систему диспетчера міського транспорту, призначену для автоматизації роботи диспетчера та централізованого керування інформацією про транспортні засоби, маршрути та призначення транспорту на маршрути.

На початковому етапі виконано аналіз предметної області та досліджено особливості організації роботи диспетчерських служб міського транспорту. Проведено аналіз вимог до програмної системи, визначено її функціональні та нефункціональні характеристики, а також обґрунтовано доцільність використання клієнт-серверної архітектури.

У результаті порівняльного аналізу технологій для реалізації програмного забезпечення обрано мову програмування Java, технологію JavaFX для створення графічного інтерфейсу користувача та систему керування базами даних PostgreSQL для централізованого зберігання інформації. Для обміну даними між компонентами системи використано формат JSON.

Під час виконання роботи спроєктовано структуру бази даних, яка включає сутності транспортних засобів, маршрутів та призначень транспорту на маршрути. Для забезпечення цілісності даних між сутностями реалізовано відповідні зв’язки на рівні бази даних.

У ході реалізації серверної частини розроблено програмні модулі для роботи з базою даних, обробки запитів клієнтського застосунку та організації мережевої взаємодії між компонентами системи. Серверна частина забезпечує отримання, збереження та передавання інформації між базою даних і клієнтським застосунком.

Для взаємодії користувача із системою реалізовано клієнтський застосунок на основі JavaFX. Розроблено основні вікна системи, призначені для перегляду транспортних засобів, маршрутів, призначень та статистичної інформації.

Використання графічного інтерфейсу дозволило забезпечити зручну роботу користувача з даними системи.

На завершальному етапі проведено випробування реалізованого програмного забезпечення. Під час тестування було перевірено роботу основних функціональних модулів, коректність обміну даними між клієнтською та серверною частинами системи, а також правильність відображення інформації в інтерфейсі користувача. Результати випробувань підтвердили працездатність розробленої системи та коректне виконання основних операцій.

Отримані результати свідчать про досягнення поставленої мети роботи. Розроблена система може використовуватися як основа для подальшого розвитку програмних засобів автоматизації диспетчерських служб міського транспорту та розширення їх функціональних можливостей.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1.Осєтрін М. М. Міські дорожньо-транспортні споруди : навч. посіб. Київ: ІЗМН, 1997. 196 с.

2.Осєтрін М. М., Беспалов Д. О., Тарасюк В. П. Міські дорожньо-транспортні споруди: конспект лекцій. Київ: КНУБА, 2022. 52 с.

3.Безлюбченко О. С., Гордієнко С. М., Завальний О. В. Планування міст і транспорт: навч. посіб. Харків: ХНУМГ ім. О. М. Бекетова, 2019. 271 с.

4.Ільчук Н. І. Міський транспорт: навч. посіб. Луцьк: ЛНТУ, 2010. 96 с.

5.Пасічник В. В., Резніченко В. А. Організація баз даних та знань: підручник. Київ: Видавнича група BHV, 2006. 384 с.

6.Берко А. Ю., Верес О. М., Пасічник В. В. Системи баз даних та знань. Кн. 1. Організація баз даних та знань: підручник. Львів: Магнолія-2006, 2015. 440 с.

7.Берко А. Ю., Верес О. М., Пасічник В. В. Системи баз даних та знань. Кн. 2. Системи управління базами даних та знань: навч. посіб. Львів : Магнолія-2006, 2012. 584 с.

8.Гайна Г. А. Основи проєктування баз даних: навч. посіб. Київ: Кондор, 2008. 200 с.

9.Мороз О. В., Берко А. Ю. Розподілені інформаційні системи: навч. посіб. Львів: Видавництво Львівської політехніки, 2014. 324 с.

10.Кучук Г. А. Комп’ютерні мережі: навч. посіб. Харків : ХНУПС, 2011. 312 с.

11.Пасічник В. В. Інформаційні системи та мережі : навч. посіб. Львів : Видавництво Львівської політехніки, 2010. 348 с.

12.Sommerville I. Software Engineering. 11th ed. Harlow: Pearson Education, 2020. 832 p.

13.PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 25.05.2026).

14.Java Platform, Standard Edition Documentation. URL:

https://docs.oracle.com/javase/ (дата звернення: 25.05.2026).

15.OpenJFX Documentation. URL: https://openjfx.io/ (дата звернення: 25.05.2026).

16.Oracle. JDBC API Documentation. URL: https://docs.oracle.com/javase/tutorial/jdbc/ (дата звернення: 25.05.2026).

17.JSON Data Interchange Format. URL: https://www.json.org/ (дата звернення: 25.05.2026).

ДОДАТОК А
ЛІСТИНГИ СЕРВЕРНОЇ ЧАСТИНИ СИСТЕМИ

Лістинг А.1 – DBML-опис сутності Transport

```
Table transport {
id int [pk, increment]
number varchar [not null]
type varchar [not null]
status varchar
}
```

Лістинг А.2 – DBML-опис сутності Route

```
Table route {
id int [pk, increment]
number varchar [not null]
start_point varchar [not null]
end_point varchar [not null]
}
```

Лістинг А.3 – DBML-опис сутності Assignment

```
Table assignment {
id int [pk, increment]
transport_id int [ref: > transport.id]
route_id int [ref: > route.id]
status varchar
}
description text
}
```

Лістинг А.4 – Фрагмент первинного наповнення бази даних

```
INSERT INTO transport (id, number, type, status) VALUES
(1, 'AE1023BM', 'Автобус', 'Активний'),
(2, 'AP4587CK', 'Тролейбус', 'Активний'),
(3, 'AE7712HP', 'Автобус', 'На обслуговуванні');
INSERT INTO route (id, number, start_point, end_point) VALUES
(1, '3', 'Вокзал', 'Центральний ринок'),
(2, '7', 'Проспект Металургів', 'Автовокзал'),
(3, '12', 'Бульвар Шевченка', 'Південний мікрорайон');
INSERT INTO assignment (id, transport_id, route_id, status) VALUES
(1, 1, 1, 'Виконується'),
(2, 2, 2, 'Призначено'),
(3, 3, 3, 'Завершено');
```

Лістинг А.5 – Реалізація класу Transport

```
public class Transport {
private int id;
private String number;
private String type;
private String status;
public Transport() {
}
public Transport(int id, String number, String type, String status)
{
this.id = id;
this.number = number;
this.type = type;
this.status = status;
}
public int getId() {
return id;
}
public void setId(int id) {
this.id = id;
}
public String getNumber() {
return number;
}
public void setNumber(String number) {
this.number = number;
}
public String getType() {
return type;
}
public void setType(String type) {
this.type = type;
}
public String getStatus() {
return status;
}
public void setStatus(String status) {
this.status = status;
}}
```

Лістинг А.6 – Код методу getTransportById класу DB

```
public Transport getTransportById(int id) {
Transport transport = null;
String sql = "SELECT * FROM transport WHERE id = ?";
try (PreparedStatement statement =
connection.prepareStatement(sql)) {
statement.setInt(1, id);
ResultSet resultSet = statement.executeQuery();
if (resultSet.next()) {
transport = new Transport(
resultSet.getInt("id"),
```

```
resultSet.getString("number"),
resultSet.getString("type"),
resultSet.getString("status"));
}
} catch (SQLException e) {
e.printStackTrace();
}
return transport;
}
```

Лістинг А.7 – Код класу Networking

```
public class Networking {
private final int port;
public Networking(int port) {
this.port = port;
}
public void start() {
try (ServerSocket serverSocket = new ServerSocket(port)) {
System.out.println("Server started on port " + port);
while (true) {
Socket clientSocket = serverSocket.accept();
System.out.println(
"Client connected: "
+ clientSocket.getInetAddress()
);
Client client = new Client(clientSocket);
Thread thread = new Thread(client);
thread.start();
}
} catch (IOException e) {
e.printStackTrace();
}}}
```

Лістинг А.8 – Фрагмент реалізації класу Client

```
public class Client implements Runnable {
private final Socket socket;
private final DB db;
public Client(Socket socket) {
this.socket = socket;
this.db = new DB();
}
@Override
public void run() {
try (
BufferedReader reader =
new BufferedReader(
new InputStreamReader(
socket.getInputStream()));
PrintWriter writer =
new PrintWriter(
```

```
socket.getOutputStream(), true)) {
String request = reader.readLine();
if ("GET_TRANSPORTS".equals(request)) {
List<Transport> transports =
db.getTransports();
writer.println(transports.toString());
}
} catch (IOException e) {
e.printStackTrace();
}}}
```

Лістинг А.9 – Обробка запиту на отримання списку транспорту

```
String request = reader.readLine();
if ("GET_TRANSPORTS".equals(request)) {
List<Transport> transports = db.getTransports();
for (Transport transport : transports) {
writer.println(
transport.getId() + ";" +
transport.getNumber() + ";" +
transport.getType() + ";" +
transport.getStatus()
);
}
writer.println("END");
}
```

ДОДАТОК Б
ЛІСТИНГИ КЛІЄНТСЬКОЇ ЧАСТИНИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Лістинг Б.1 – Фрагмент структури даних, яку сервер передає клієнту

```
{
"transports": [
{
"id": 1,
"number": "AA 1234 BB",
"type": "Автобус",
"status": "Вільний"
},
{
"id": 2,
"number": "AA 5678 CC",
"type": "Тролейбус",
"status": "На маршруті"
}
]
}
```

Лістинг Б.2 – Фрагмент коду класу MainApplication

```
public class MainApplication extends Application {
@Override
public void start(Stage stage) throws Exception {
FXMLLoader loader =
new FXMLLoader(
getClass().getResource("main.fxml"));
Scene scene =
new Scene(loader.load());
stage.setTitle(
"Система диспетчера міського транспорту");
stage.setScene(scene);
stage.show();
}
public static void main(String[] args) {
launch(args);
}
}
```

Лістинг 3.3 – Фрагмент класу NetworkClient

```
public class NetworkClient {
private final String host;
private final int port;
public NetworkClient(String host, int port) {
```

```
this.host = host;
this.port = port;
}
public String sendRequest(String request)
throws IOException {
try (
Socket socket =
new Socket(host, port);
PrintWriter writer =
new PrintWriter(
socket.getOutputStream(),
true);
BufferedReader reader =
new BufferedReader(
new InputStreamReader(
socket.getInputStream()))
) {
writer.println(request);
return reader.readLine();
}
}
}
```

Лістинг Б.4 – Фрагмент коду класу MainController

```
public class MainController {
@FXML
private Button transportButton;
@FXML
private Button routeButton;
@FXML
private Button assignmentButton;
@FXML
private Button statisticsButton;
@FXML
private void openTransport() {
System.out.println("Transport section opened");
}
@FXML
private void openRoutes() {
System.out.println("Routes section opened");
}
@FXML
private void openAssignments() {
System.out.println("Assignments section opened");
}
@FXML
private void openStatistics() {
System.out.println("Statistics section opened");
}
}
```

Лістинг Б.5 – Фрагмент коду класу TransportController

```
public class TransportController {
@FXML
private TableView<Transport> transportTable;
@FXML
private TableColumn<Transport, Integer> idColumn;
@FXML
private TableColumn<Transport, String> numberColumn;
@FXML
private TableColumn<Transport, String> typeColumn;
@FXML
private TableColumn<Transport, String> statusColumn;
@FXML
public void initialize() {
idColumn.setCellValueFactory(
new PropertyValueFactory<>("id"));
numberColumn.setCellValueFactory(
new PropertyValueFactory<>("number"));
typeColumn.setCellValueFactory(
new PropertyValueFactory<>("type"));
statusColumn.setCellValueFactory(
new PropertyValueFactory<>("status"));
}
public void setTransports(
List<Transport> transports) {
transportTable.getItems().clear();
transportTable.getItems().addAll(
transports);
}
}
```

Лістинг Б.6 – Фрагмент FXML-розмітки вікна транспорту

```
<VBox spacing="10" xmlns:fx="http://javafx.com/fxml"
fx:controller="controller.TransportController">
<Label text="Транспортні засоби" />
<TableView fx:id="transportTable">
<columns>
<TableColumn fx:id="idColumn" text="ID" />
<TableColumn fx:id="numberColumn" text="Номер" />
<TableColumn fx:id="typeColumn" text="Тип" />
<TableColumn fx:id="statusColumn" text="Статус" />
</columns>
</TableView>
</VBox>
```

ДОДАТОК В
СЛАЙДИ ПРЕЗЕНТАЦІЇ

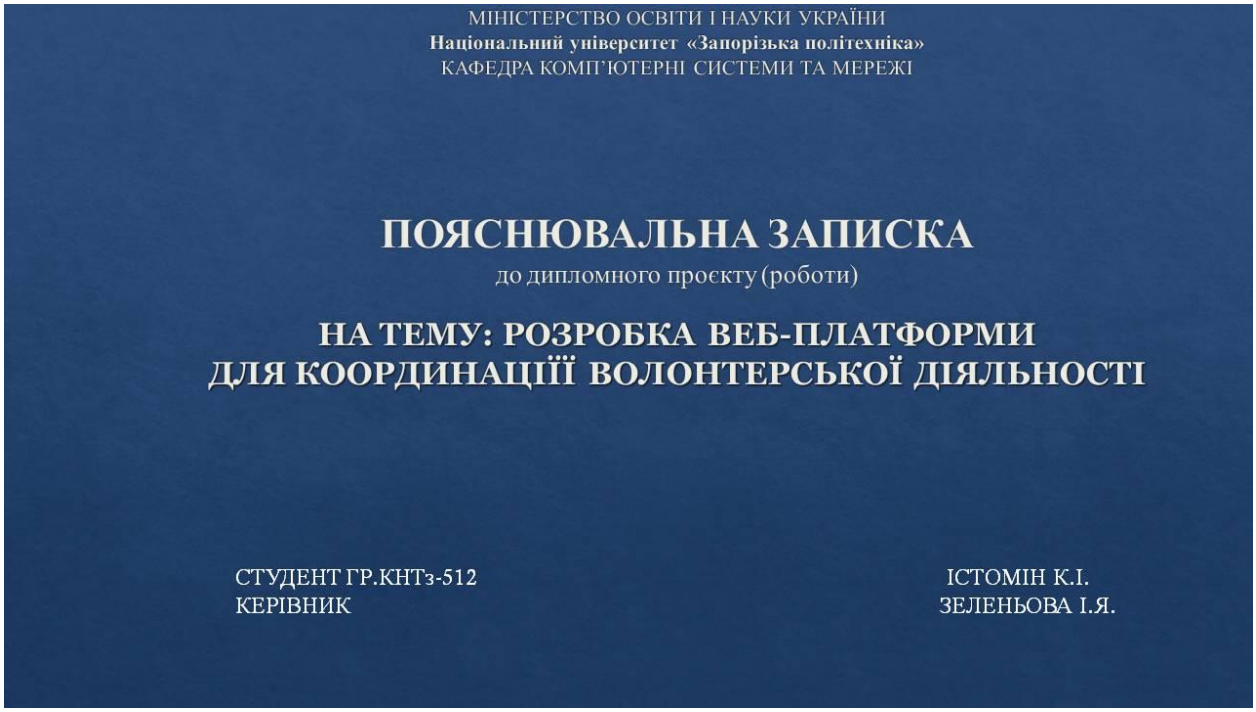

Рисунок В.1 – Слайд 1

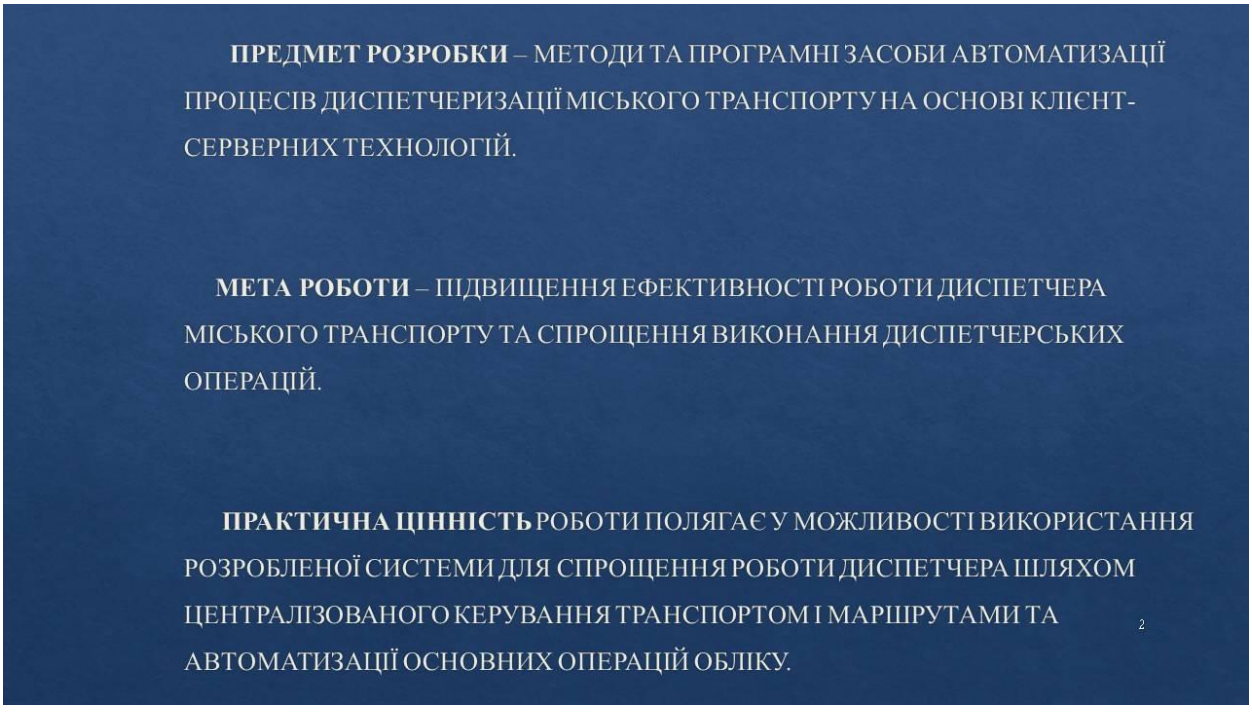

Рисунок В.2 – Слайд 2

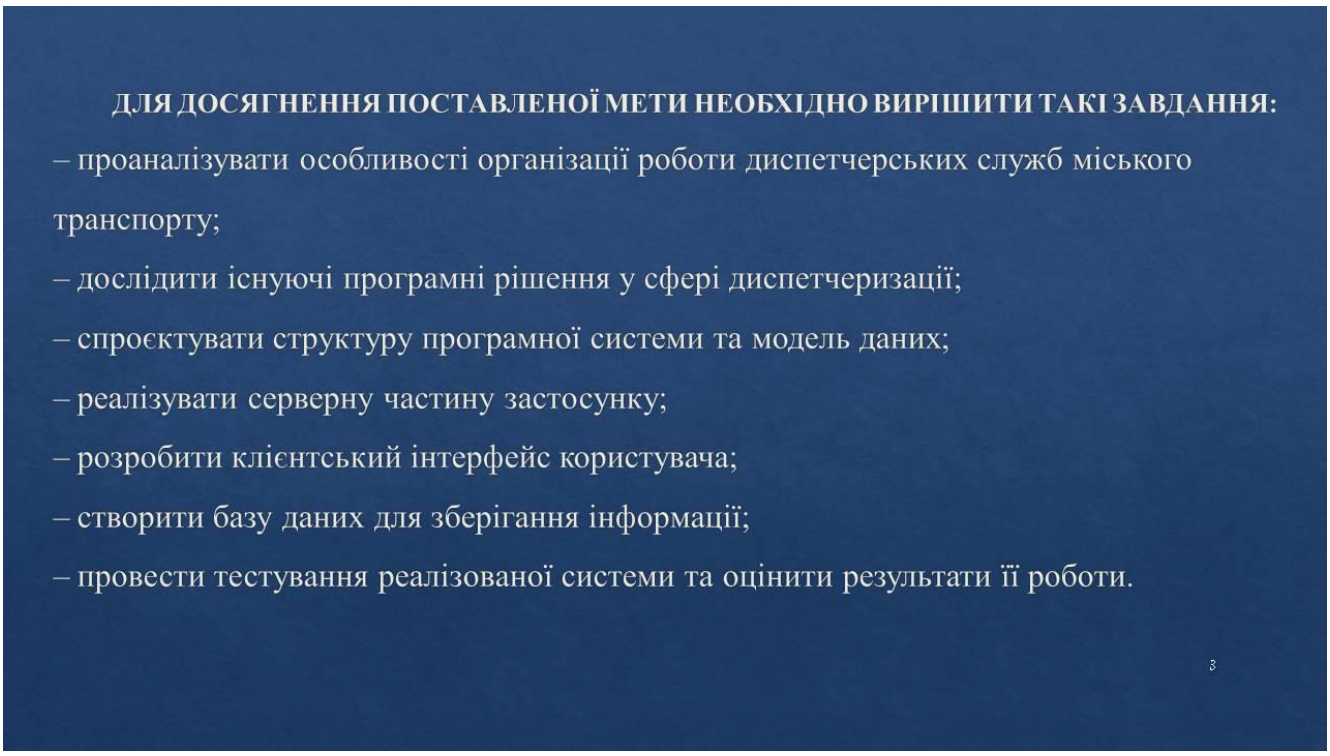

Рисунок В.3 – Слайд 3

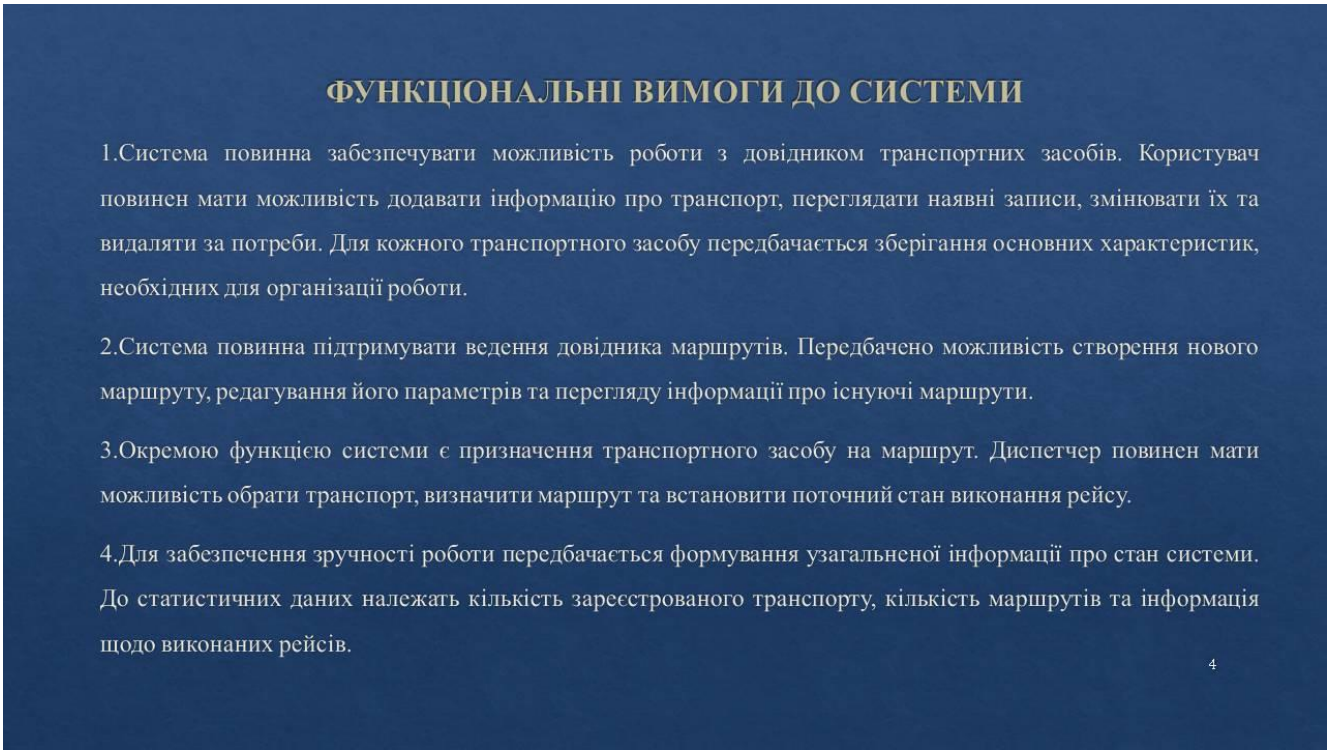

Рисунок В.4 – Слайд 4

НЕФУНКЦІОНАЛЬНІ ВИМОГИ ДО СИСТЕМИ

1.Зручність використання системи.

2.Продуктивність системи.

3.Надійність функціонування.

4.Цілісність та збереження даних.

5.Масштабованість рішення.

6.Система повинна відповідати вимогам підтримуваності програмного коду.

7.Сумісність програмного забезпечення.

5

Рисунок В.5 – Слайд 5

ВИБІР МОВИ ПРОГРАМУВАННЯ ТА ПЛАТФОРМИ РЕАЛІЗАЦІЇ ПОРІВНЯННЯ МОВ ПРОГРАМУВАННЯ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ

Критерій	Java	C#	Python
Кросплатформеність	+	+-	+
Робота з GUI	+	+	+-
Підтримка клієнт–серверної архітектури	+	+	+
Інтеграція з PostgreSQL	+	+	+
Простота розгортання	+	+	+
Готовність до мобільного клієнта	+	+-	+

6

Рисунок В.6 – Слайд 6

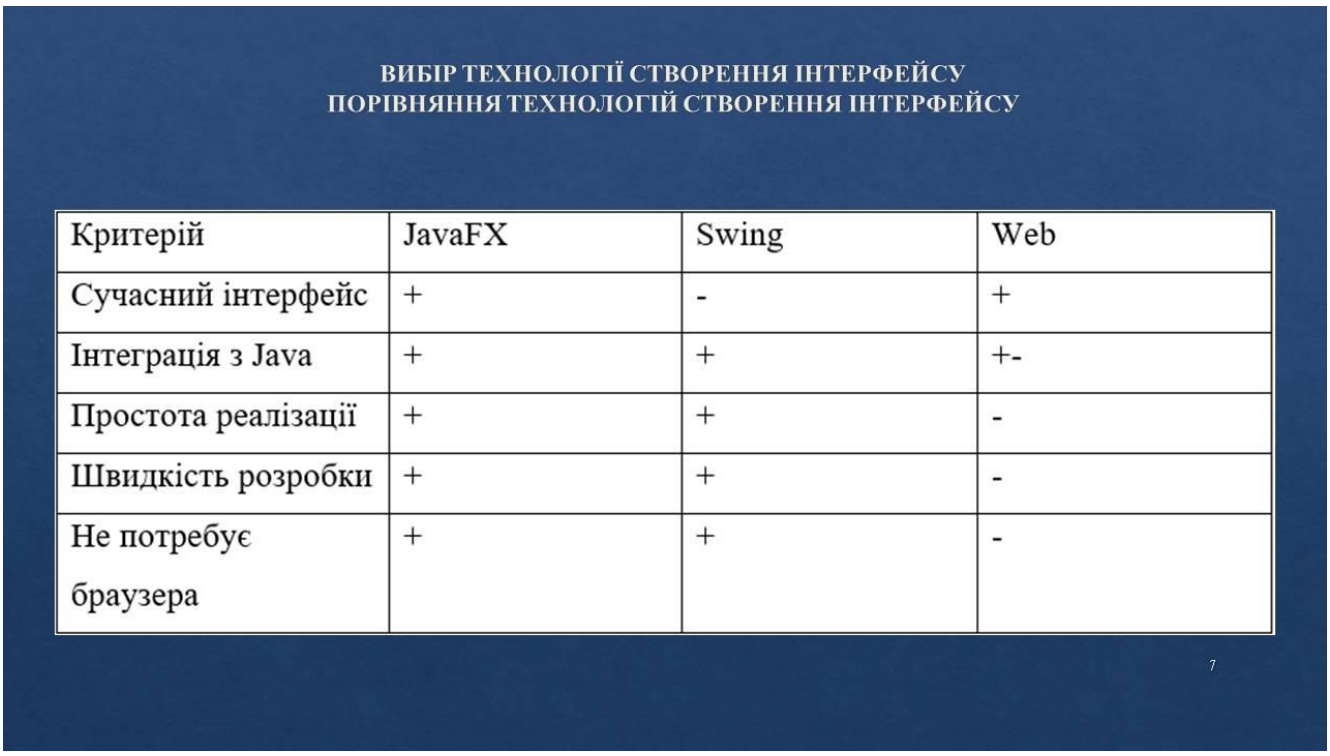

Рисунок В.7 – Слайд 7

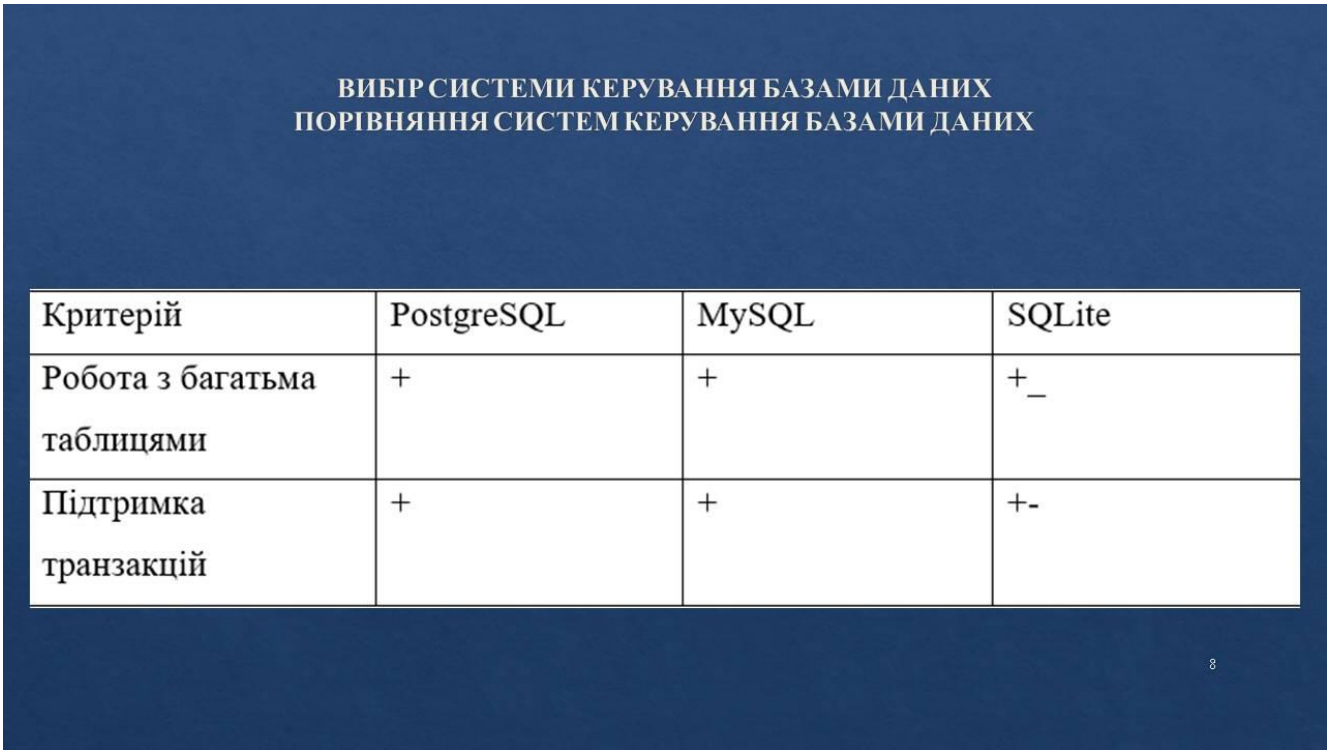

Рисунок В.8 – Слайд 8

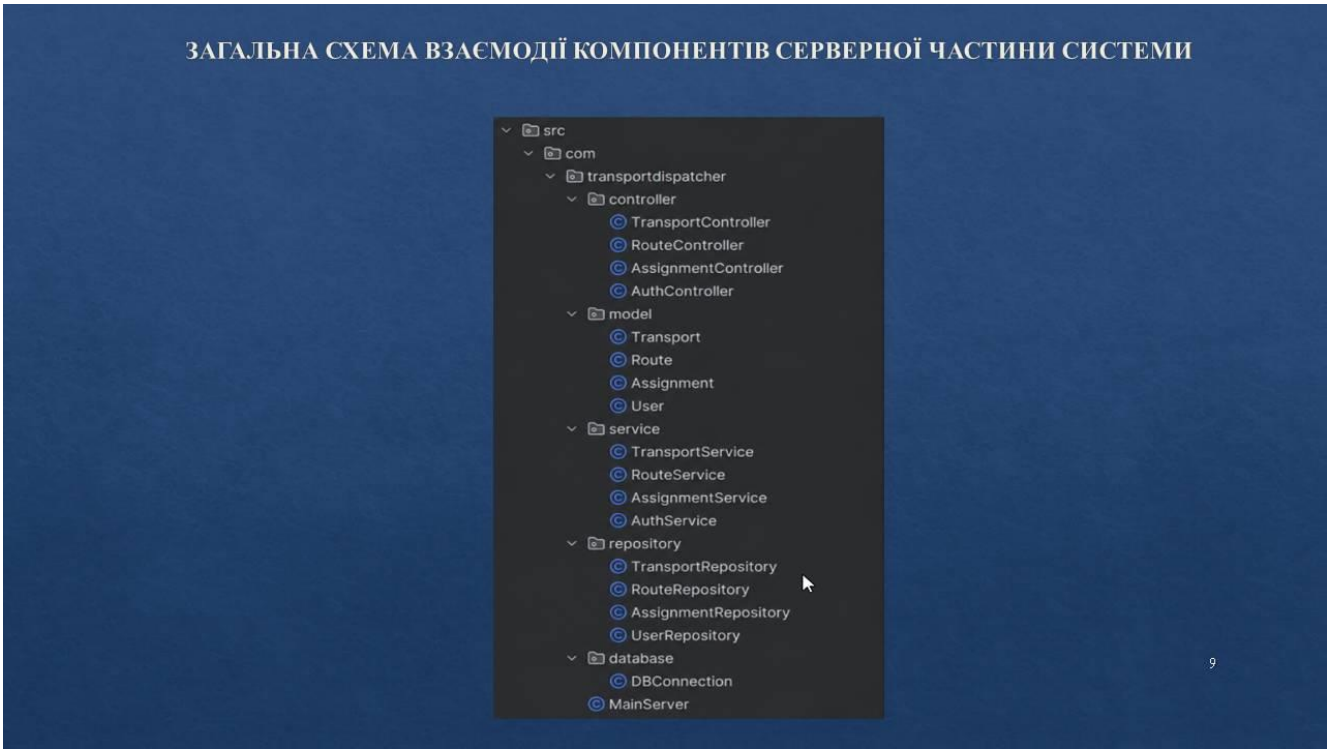

Рисунок В. – Слайд 9

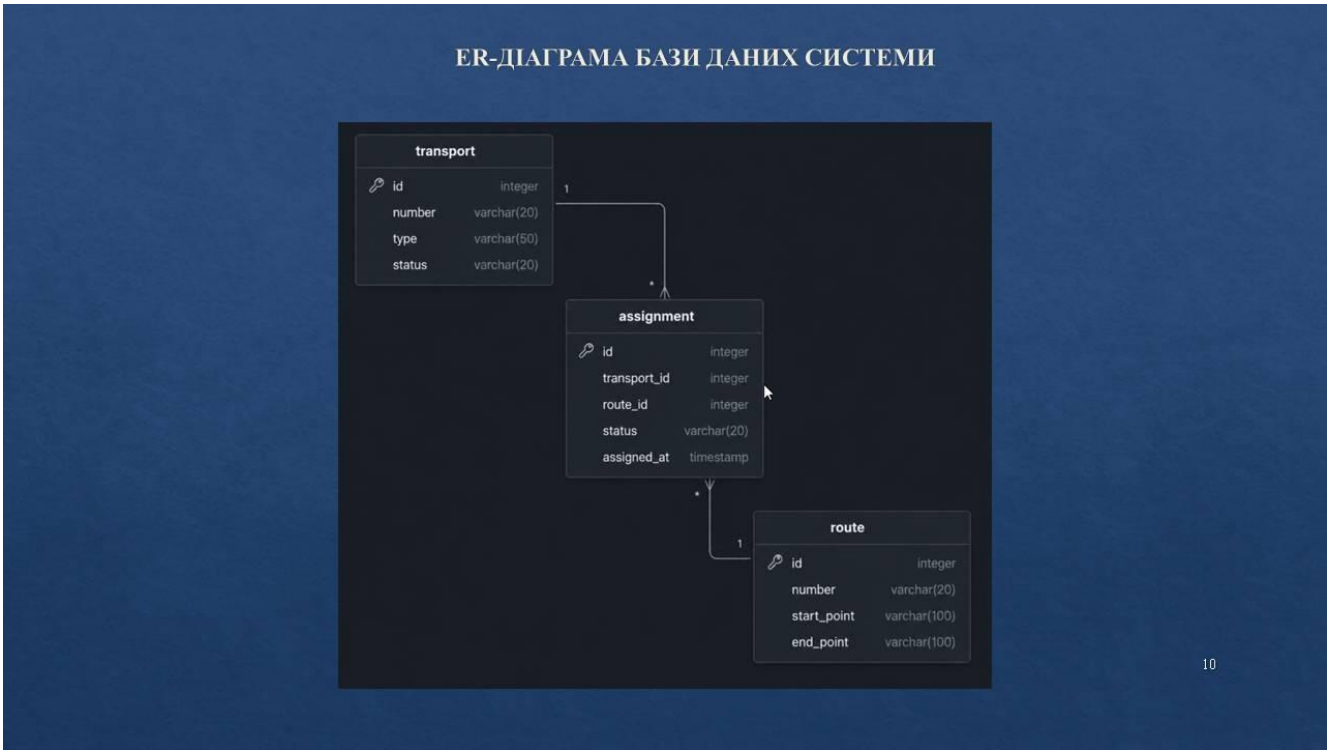

Рисунок В.10 – Слайд 10

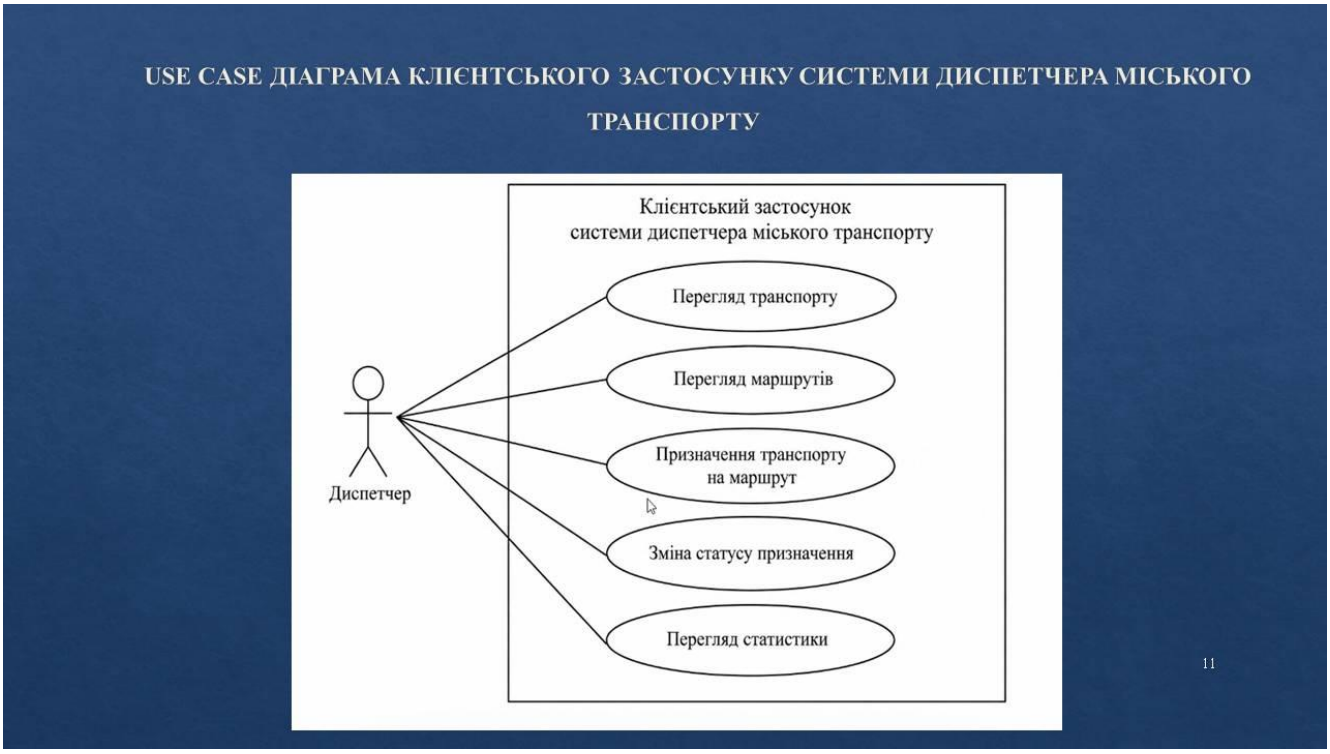

Рисунок В.11 – Слайд 11

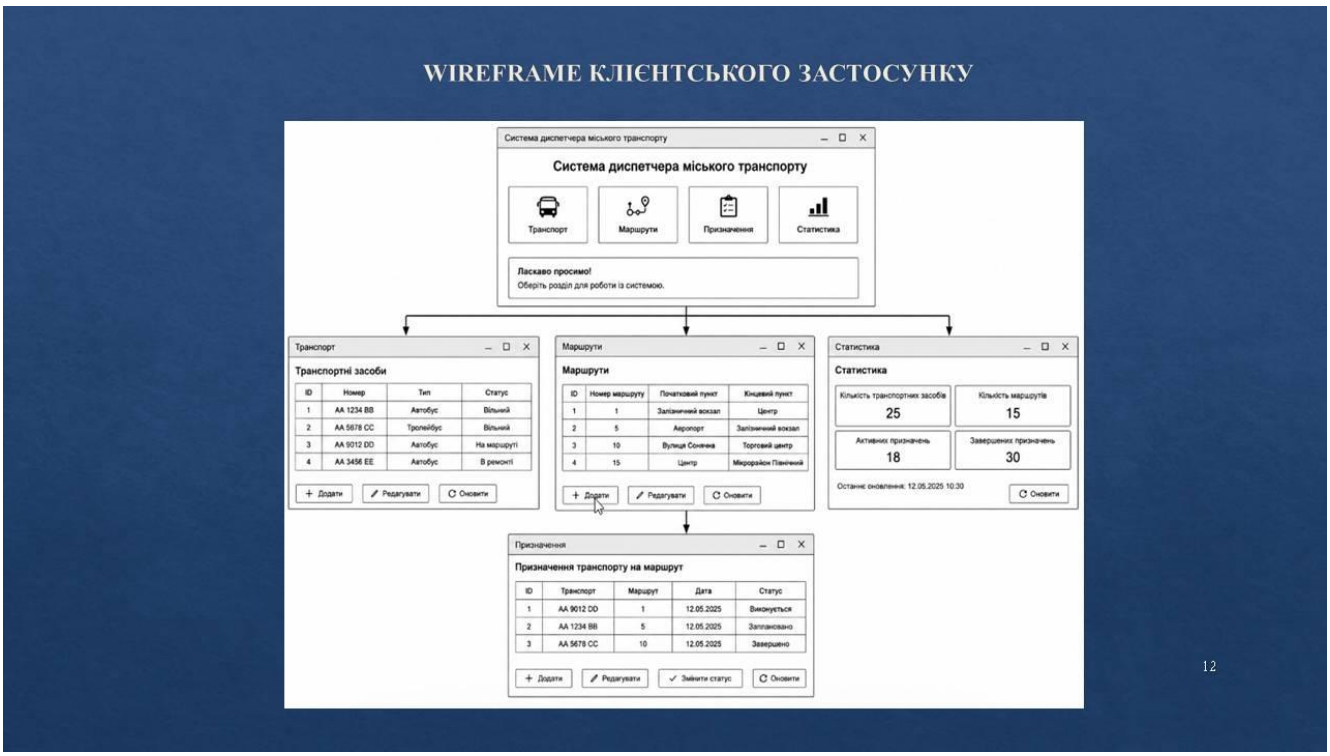

Рисунок В.12 – Слайд 12

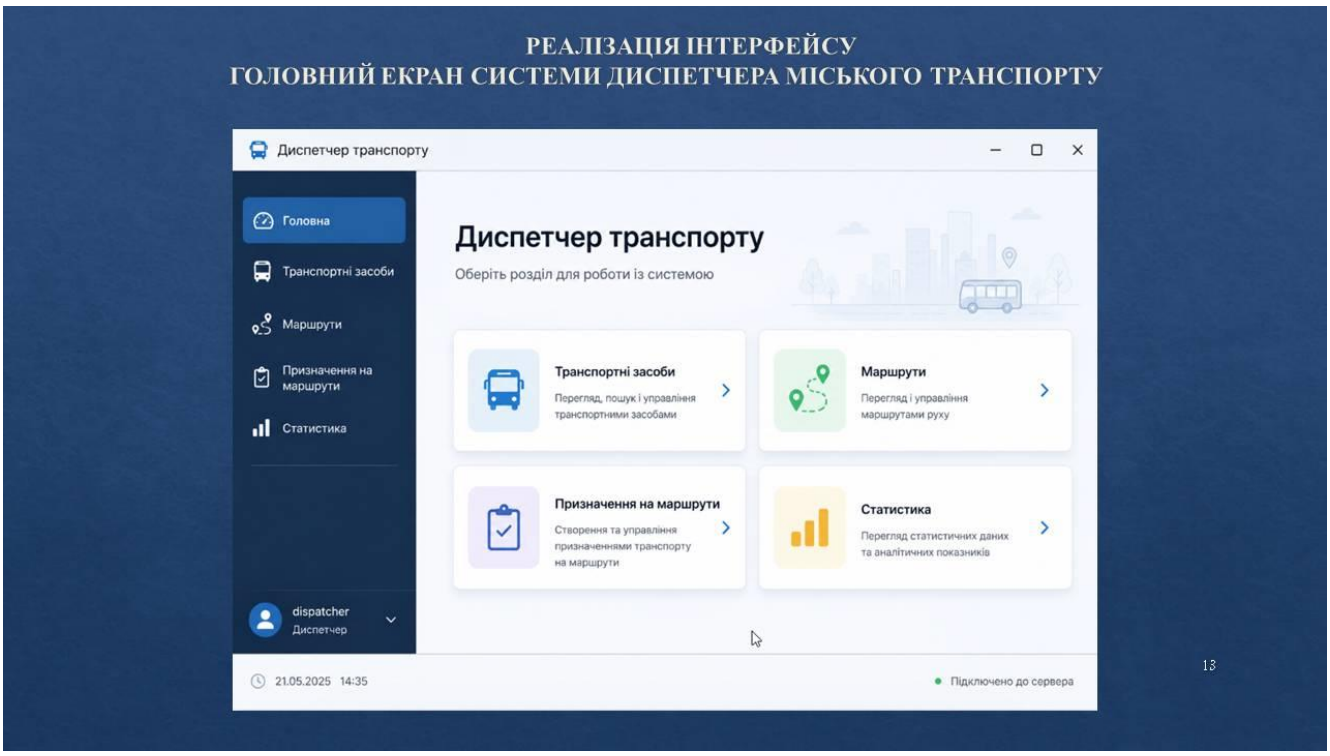

Рисунок В.13 – Слайд 13

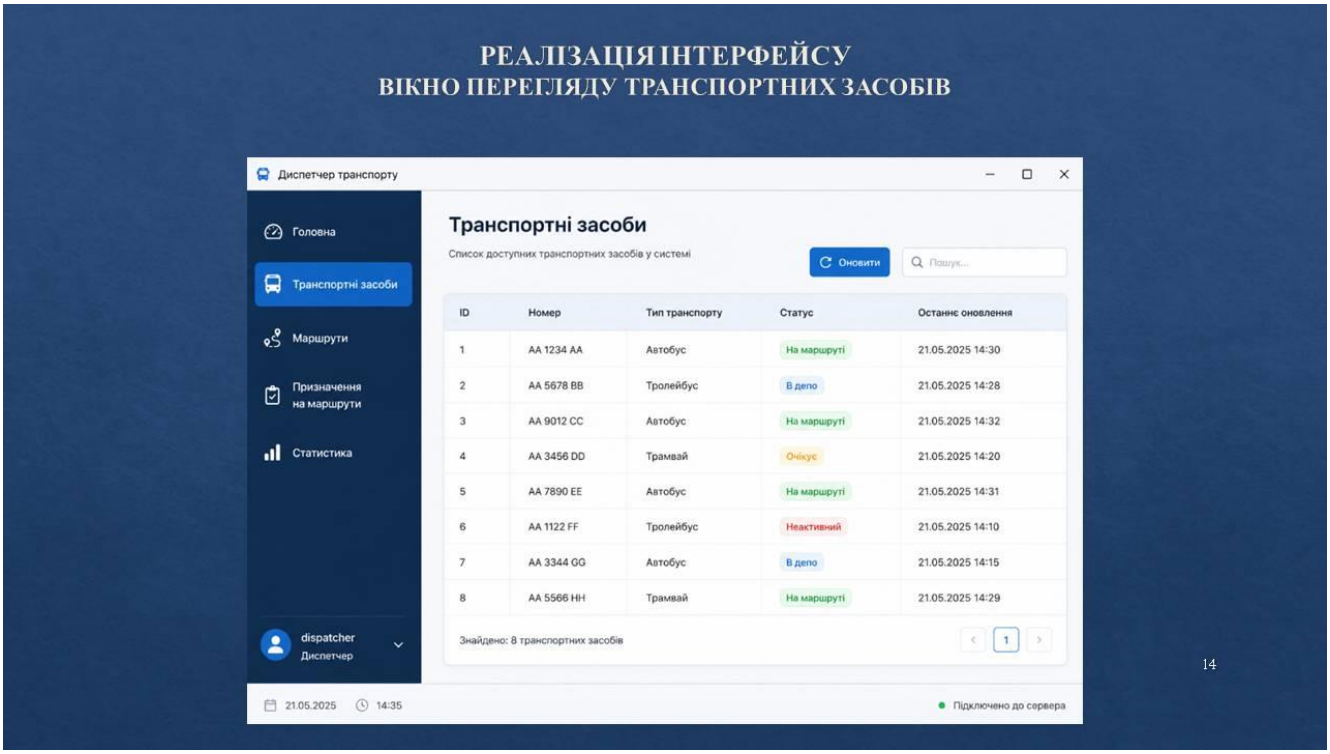

Рисунок В.14 – Слайд 14

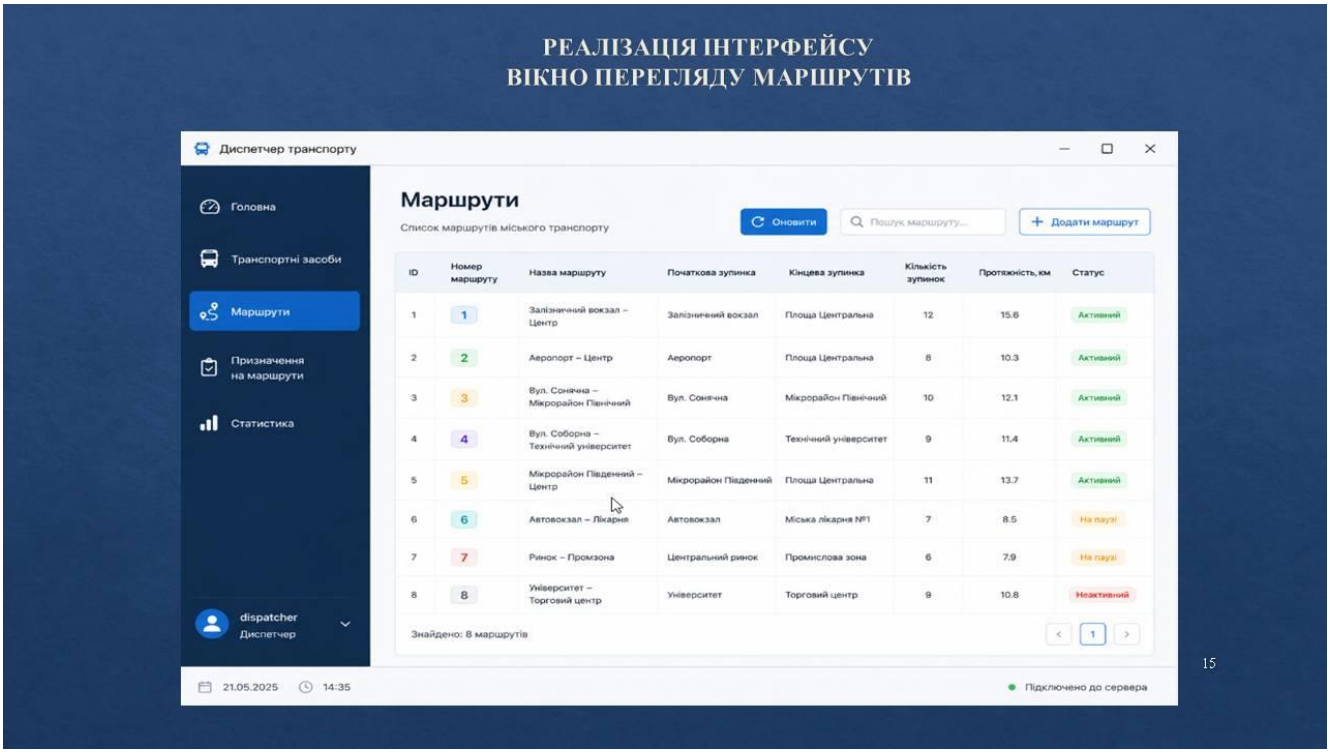

Рисунок В.15 – Слайд 15

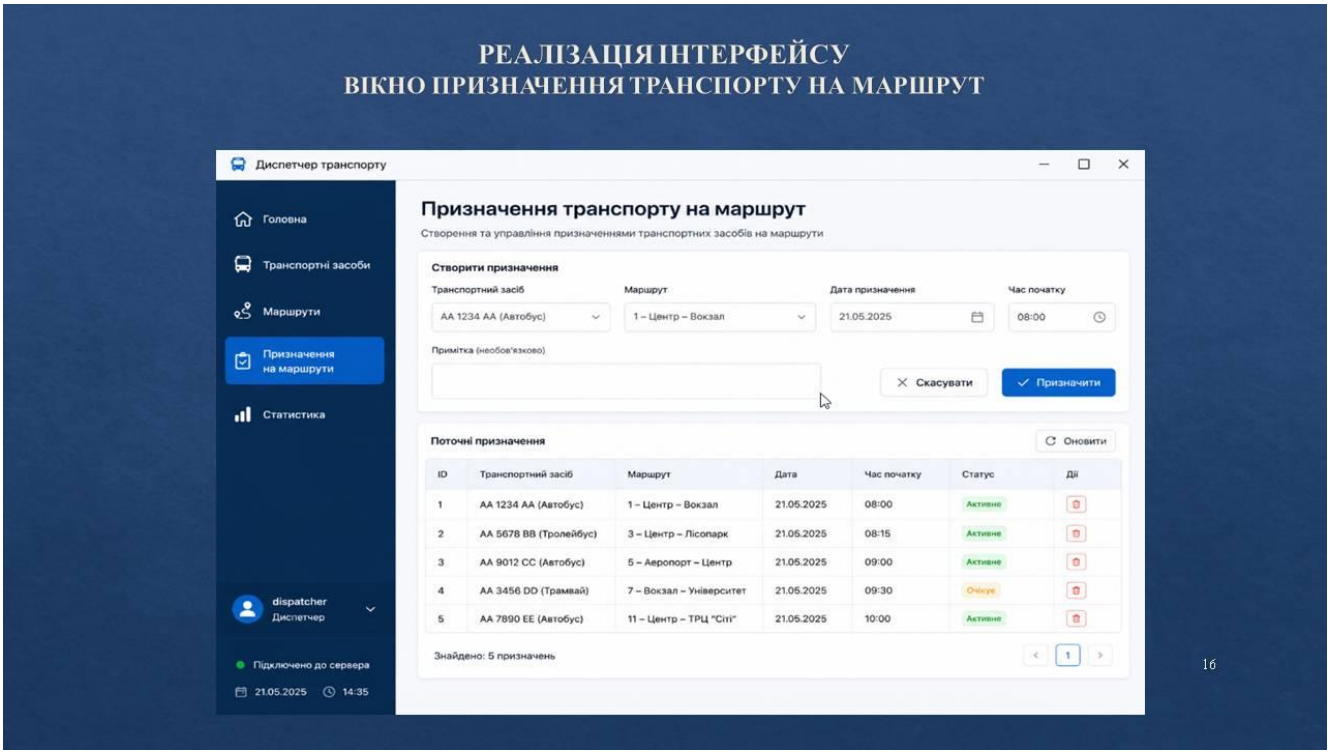

Рисунок В.16 – Слайд 16

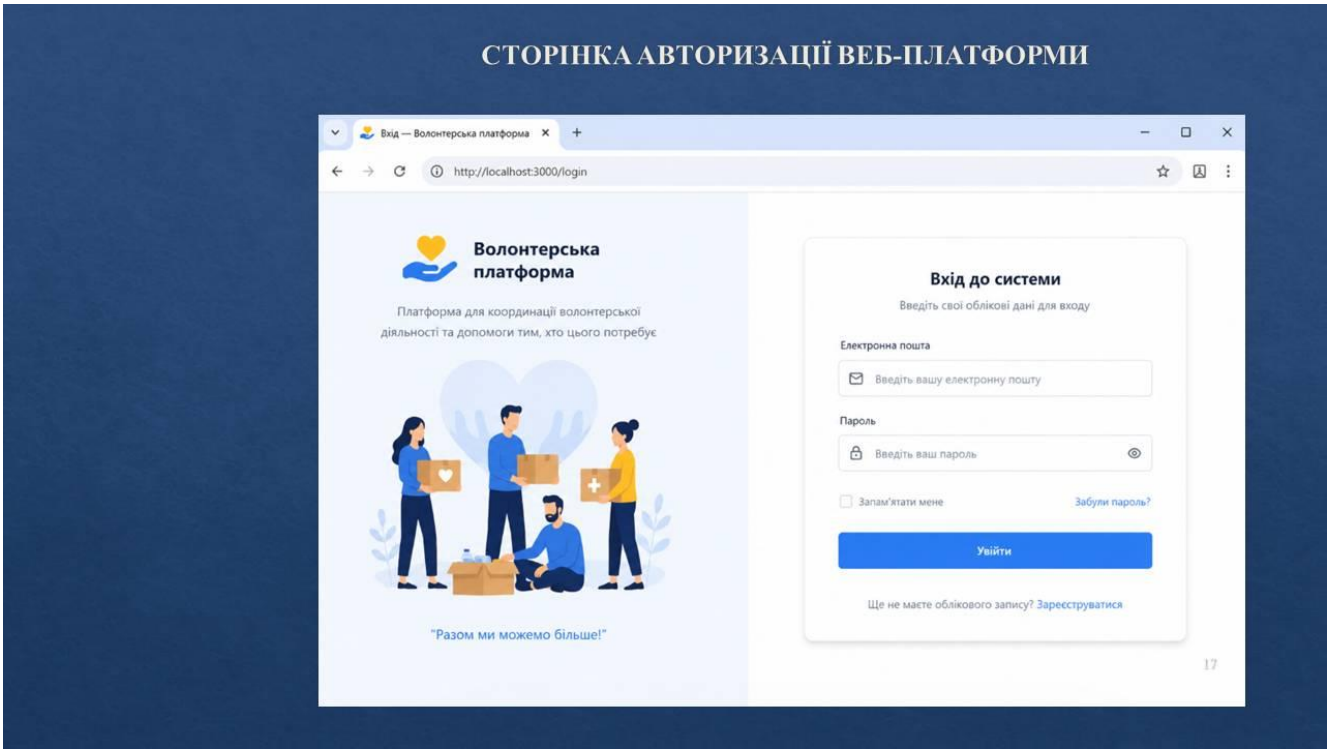

Рисунок В.17 – Слайд 17

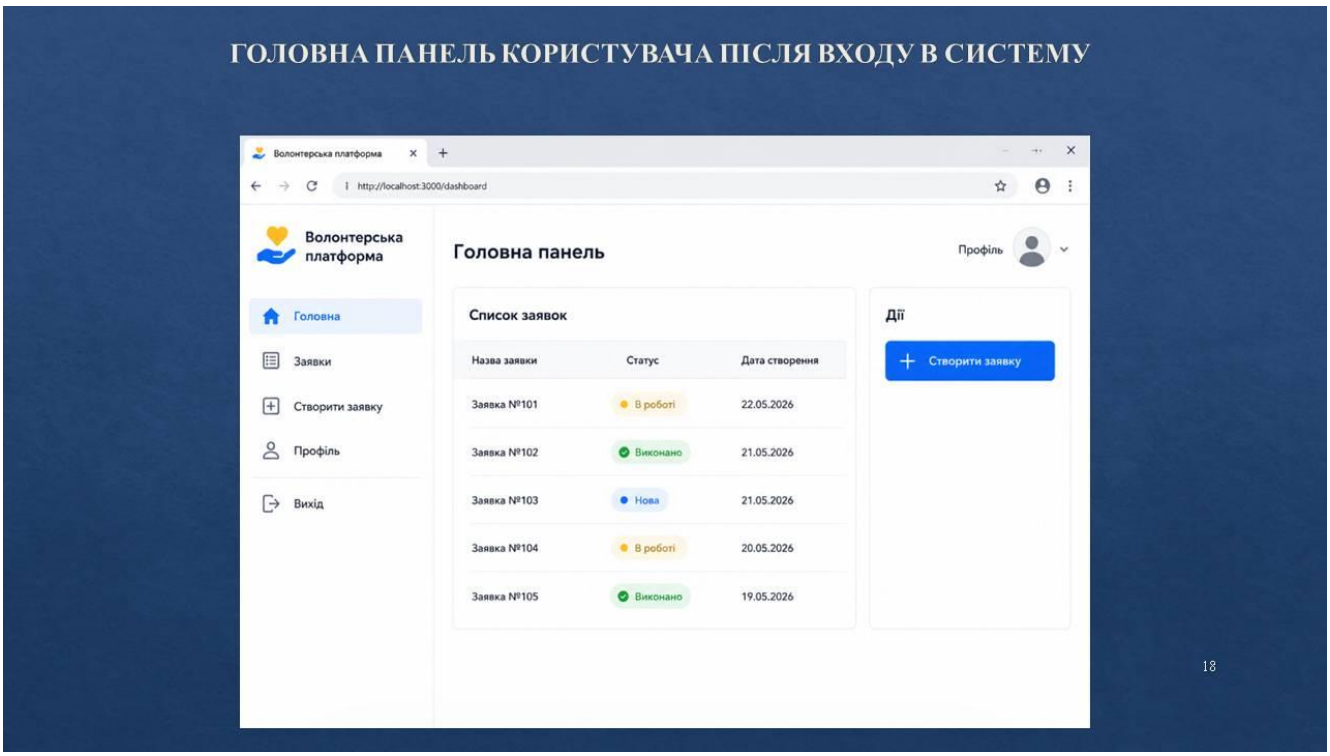

Рисунок В.18 – Слайд 18

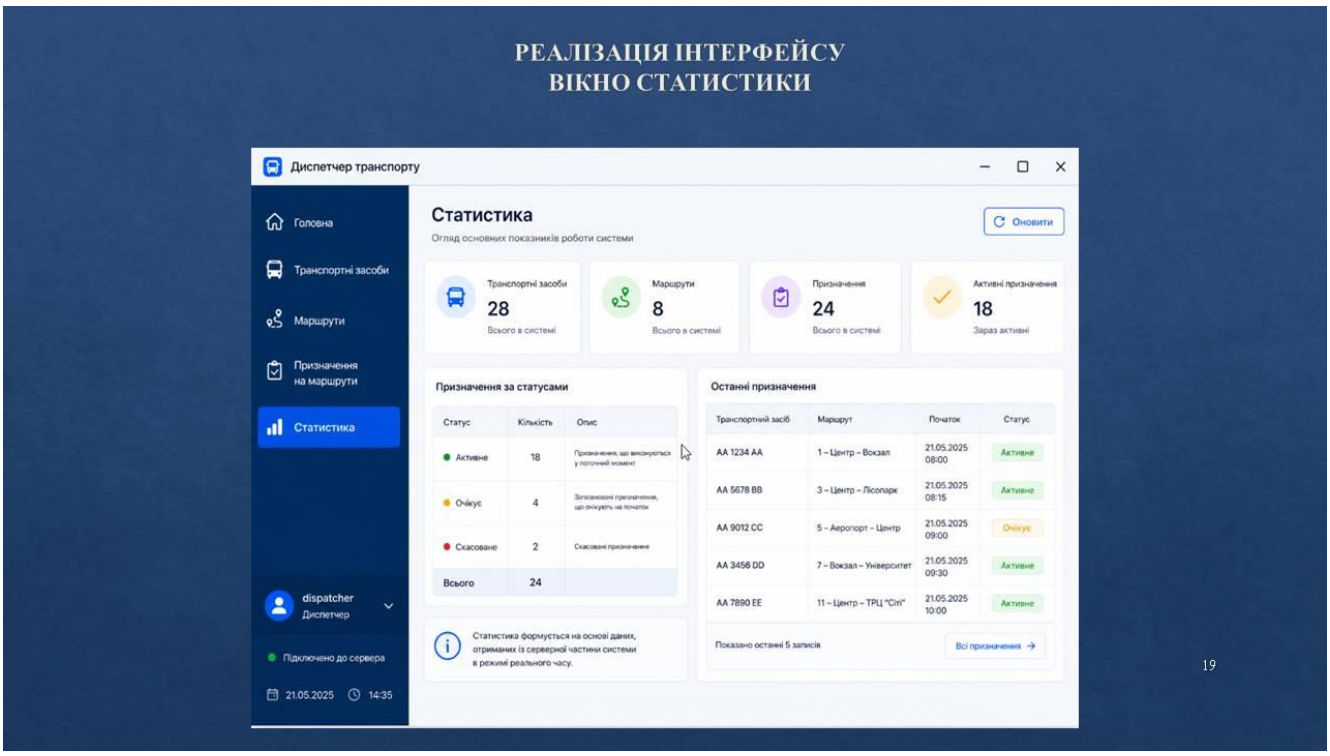

Рисунок В.19 – Слайд 19

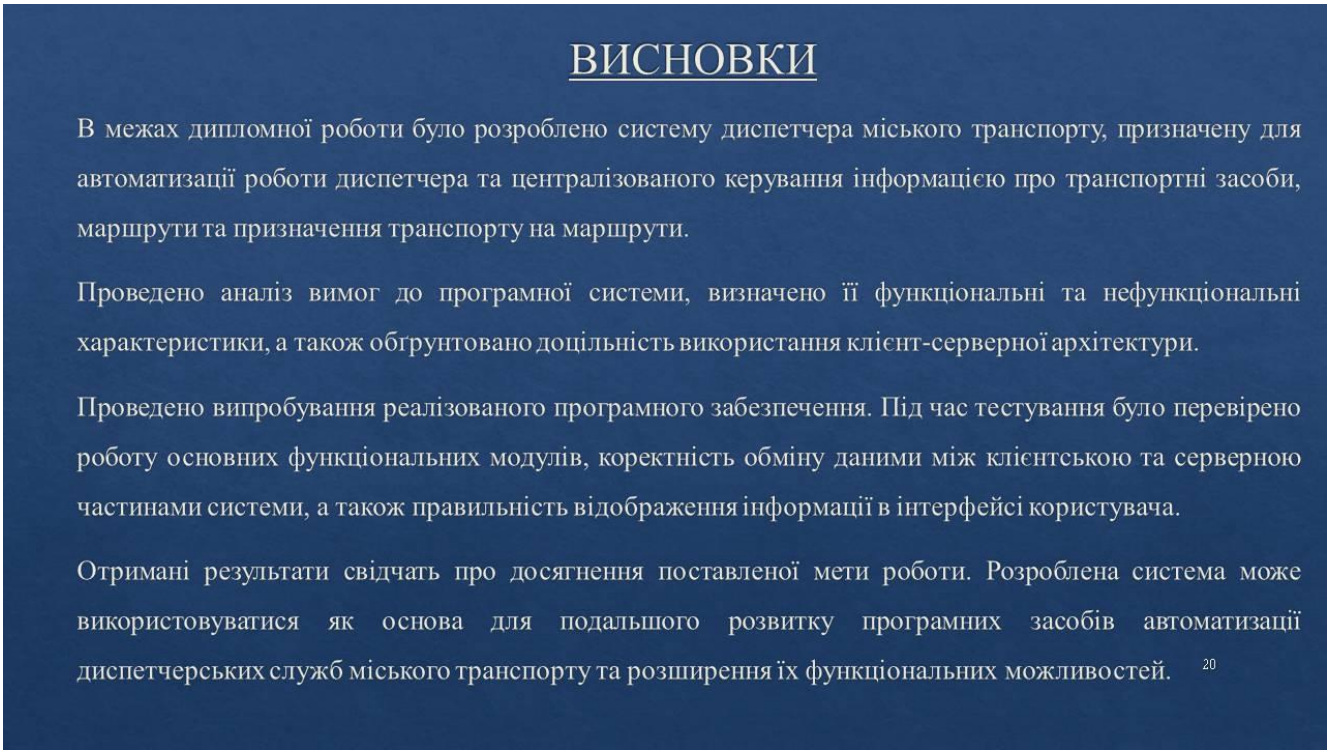

Рисунок В.20 – Слайд 20