МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЧЕМЕС ВЛАДИСЛАВ СТАНІСЛАВОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д-р. техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

РОЗРОБКА БАГАТОКОМПОНЕНТНОЇ СИСТЕМИ ДЛЯ ВІДОБРАЖЕННЯ РОЗКЛАДУ РУХУ ГРОМАДСЬКОГО ТРАНСПОРТУ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Олександр ПАВЛЮК, старший викладач
кафедри інформаційних технологій,
к. т. н., ст. викладач
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2026

АНОТАЦІЯ

Чемес В.С. Розробка багатокомпонентної системи для відображення розкладу руху громадського транспорту. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2026.

У кваліфікаційній (бакалаврській) роботі досліджено та проаналізовано предметну область систем відображення розкладу руху громадського транспорту. Розроблено багатокомпонентну систему, що включає серверну частину на базі Node.js, клієнтський веб-інтерфейс з використанням React.js та базу даних для зберігання та обробки даних про рух транспорту. Система забезпечує відображення та пошук розкладу маршрутів і зупинок, фільтрацію за видами транспорту та зручний інтерфейс користувача.

Ключові слова: громадський транспорт, розклад руху, багатокомпонентна система, веб-застосунок, React.js.

59 ст. 8 рис., 7 табл., 47 джерел.

ABSTRACT

Chemes V.S. Development of a Multi-Component System for Displaying Public Transport Schedules. Specialty 122 «Computer Science», educational program «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia 2026.

The qualification (bachelor's) work investigates and analyses the subject area of public transport schedule display systems. A multi-component system has been developed comprising a server-side component based on Node.js, a client-side web interface using React.js, and a database for storing and processing transport movement data. The system provides schedule display and search for routes and stops, transport type filtering, and a user-friendly interface.

Keywords: public transport, movement schedule, multi-component system, web application, React.js.

59 p., 8 fig., 7 table, 47 sources.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 4
ВСТУП 5
РОЗДІЛ 1 8
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ 8
1.1 Дослідження предметної області систем відображення розкладу руху громадського транспорту 8
1.2 Огляд та порівняння існуючих аналогів систем розкладу транспорту 12
1.3 Формування функціональних та нефункціональних вимог до системи 16
1.4 Вибір технологічного стеку та інструментальних засобів розробки 20
РОЗДІЛ 2 25
ПРОЄКТУВАННЯ БАГАТОКОМПОНЕНТНОЇ СИСТЕМИ 25
2.1 Архітектура багатокомпонентної системи 25
2.2 Проєктування бази даних та моделей даних 28
2.3 Алгоритми обробки та відображення даних розкладу 33
2.4 Проєктування REST API та взаємодії між компонентами 36
2.5 Проєктування інтерфейсу користувача 38
РОЗДІЛ 3 42
РЕАЛІЗАЦІЯ БАГАТОКОМПОНЕНТНОЇ СИСТЕМИ 42
3.1 Реалізація серверної частини системи 42
3.2 Реалізація клієнтської частини системи 43
3.3 Результати реалізації системи та напрями її вдосконалення 50
ВИСНОВКИ 53
СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ 55

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

СУБД – Система управління базами даних

API – Application Programming Interface

CSS – Cascading Style Sheets

DOM – Document Object Model

GPS - Global Positioning System

GTFS – General Transit Feed Specification

HTTP – Hyper Text Transfer Protocol

JSON – JavaScript Object Notation

REST – Representational State Transfer

SQL – Structured Query Language

ВСТУП

Актуальність теми дослідження

В умовах стрімкого розвитку міської інфраструктури та зростання потреб населення у мобільності питання ефективної організації та інформаційного забезпечення громадського транспорту набуває особливої актуальності. Сучасні користувачі потребують зручного, швидкого та надійного доступу до актуальної інформації про рух громадського транспорту, зокрема до розкладу маршрутів, часу відправлення та прибуття, а також зручних інструментів для планування поїздок.

Незважаючи на поширення мобільних застосунків і веб-сервісів для відображення транспортної інформації, більшість існуючих рішень мають суттєві обмеження, такі як відсутність підтримки українських міст, застарілі інтерфейси та обмежений функціонал. Це зумовлює необхідність розробки спеціалізованої багатокомпонентної системи, яка забезпечує відображення актуального розкладу руху громадського транспорту з урахуванням потреб користувачів.

Мета дослідження

Метою кваліфікаційної (бакалаврської) роботи є розробка багатокомпонентної системи для відображення розкладу руху громадського транспорту, яка забезпечує зручний доступ користувачів до актуальної транспортної інформації через веб-інтерфейс.

Завдання дослідження

Для досягнення поставленої мети необхідно вирішити такі завдання:

1) Дослідити предметну область та проаналізувати існуючі системи відображення розкладу громадського транспорту.
2) Виявити переваги та недоліки наявних аналогів і сформулювати функціональні та нефункціональні вимоги до системи.
3) Обґрунтувати вибір технологічного стеку та інструментальних засобів розробки.

4) Спроєктувати архітектуру багатокомпонентної системи, структуру бази даних та алгоритми обробки транспортних даних.
5) Розробити серверну та клієнтську частини системи з реалізацією всіх визначених функціональних вимог.

Об’єкт дослідження

Об’єктом дослідження є процес інформаційного забезпечення користувачів актуальними даними пор рух громадського транспорту в міському середовищі.

Предмет дослідження

Предметом дослідження є методи, алгоритми та технологічні засоби розробки багатокомпонентних веб-систем для відображення розкладу руху громадського транспорту.

Теоретичне та/або практичне значення одержаних результатів

Теоретичне значення роботи полягає в узагальненні підходів до проєктування багатокомпонентних систем для роботи з транспортними даними та аналізі сучасних технологій реалізації подібних рішень. Практичне значення результатів роботи полягає у створенні функціонального програмного продукту, який може бути використаний міськими транспортними підприємствами та місцевими органами самоврядування для інформування населення про розклад руху громадського транспорту.

Апробація результатів дослідження

Результати кваліфікаційної (бакалаврської) роботи апробовано на двох наукових конференціях:

1. X Міжнародна наукова конференція студентів, аспірантів та молодих вчених «Topical Issues of Humanities, Technical, and Natural Sciences», яка відбулась 30 квітня 2026 року на базі Донецького національного університету імені Василя Стуса. Тези на тему: «The impact of information systems on the efficiency of public transport usage» .
2. VI Всеукраїнська науково-практична конференція «Прикладні інформаційні технології» (ПІТ 2026). Тези на тему: «Бібліотеки та

фреймворки React для роботи з картографічними даними у транспортних інформаційних системах».

Структура кваліфікаційної (бакалаврської) роботи

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі проведено аналіз предметної області, огляд і порівняння існуючих систем відображення транспортного розкладу, сформульовано вимоги до розроблювальної системи та обґрунтовано вибір технологічного стеку.

У другому розділі спроєктовано архітектуру багатокомпонентної системи, описано структуру бази даних, розроблено алгоритми обробки даних розкладу та спроєктовано API для взаємодії між компонентами.

У третьому розділі описано реалізацію серверної та клієнтської частин системи, інтеграцію компонентів та розгортання розробленого програмного продукту.

Робота містить 59 сторінок, 8 рисунків, 7 таблиць та список літератури з 47 джерел.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Дослідження предметної області систем відображення розкладу руху громадського транспорту

Громадський транспорт є невід'ємною складовою міської інфраструктури та відіграє важливу роль у сталому розвитку сучасних міст. Він сприяє зменшенню транспортного навантаження на дорожню мережу, скороченню кількості приватних автомобілів у центральних частинах міста, зниженню рівня шкідливих викидів та забезпеченню доступної мобільності для різних груп населення. Ефективна організація громадського транспорту безпосередньо впливає на якість життя міських мешканців, рівень транспортної доступності районів міста та загальну економічну активність.

Ефективність використання громадського транспорту залежить не лише від якості рухомого складу, стану дорожньої інфраструктури та раціональності маршрутної мережі, а й від рівня інформаційного забезпечення пасажирів. Користувачі громадського транспорту мають потребу швидко отримувати зрозумілу інформацію про маршрути, зупинки, напрямки руху, час першого та останнього рейсу, інтервали руху та розклад прибуття на конкретні зупинки. Наявність такої інформації зменшує невизначеність під час поїздки та підвищує довіру до громадського транспорту як до щоденного засобу пересування.

Поняття «транспортна інформаційна система» охоплює широкий клас програмних та апаратних засобів, призначених для збору, зберігання, обробки та відображення даних про функціонування транспортних мереж. У контексті громадського транспорту такі системи виконують кілька ключових функцій: планування та зберігання розкладів руху; відображення актуальної транспортної інформації для пасажирів; аналіз ефективності роботи транспортної мережі [1].

Розклад руху є одним із базових елементів інформаційного забезпечення пасажира. З технічної точки зору розклад являє собою структурований набір даних, який описує часові та просторові характеристики руху транспортних засобів [2]. До основних елементів такого набору належать маршрути, напрямки

руху, зупинки з координатами, рейси як конкретні проходження маршруту у визначений час, а також таблиці часу прибуття транспорту на кожну зупинку. Саме ці дані є основою для побудови системи перегляду розкладу громадського транспорту.

Традиційні паперові розклади мають низку суттєвих обмежень. Вони є статичними, швидко застарівають при зміні маршруту або графіка руху, не дозволяють швидко здійснювати пошук і не забезпечують зручного перегляду розкладу для різних напрямків. Цифрові системи розкладу усувають значну частину цих недоліків, оскільки дозволяють централізовано оновлювати дані, відображати інформацію через веб-інтерфейс або мобільний застосунок, виконувати пошук за маршрутом чи зупинкою та інтегрувати карту для просторового подання транспортної мережі [1, 3].

Фундаментальним відкритим стандартом для представлення даних громадського транспорту у цифровому форматі є GTFS (General Transit Feed Specification) – специфікація, первинно розроблена компанією Google спільно з транспортним агентством TriMet міста Портленд у 2005–2006 роках. Основною метою створення GTFS було забезпечення єдиного формату для публікації розкладів руху та маршрутної інформації, що дозволяє інтегрувати транспортні дані в навігаційні сервіси, системи планування поїздок і мобільні додатки [4].

GTFS визначає структуру набору текстових файлів у форматі TXT, пов’язаних між собою через унікальні ідентифікатори. Кожен файл описує окремий аспект функціонування транспортної мережі. Зокрема, файл agency.txt містить інформацію про транспортного оператора, включаючи його назву та контактні дані; stops.txt зберігає перелік зупинок із географічними координатами; routes.txt описує маршрути перевезень; trips.txt містить інформацію про окремі рейси в межах маршрутів; stop_times.txt визначає час прибуття та відправлення для кожної зупинки на конкретному рейсі. Для опису періодичності руху використовується файл calendar.txt, який задає дні тижня та періоди дії розкладу, тоді як calendar_dates.txt дозволяє задавати винятки, наприклад зміни графіка у святкові дні або під час спеціальних подій [4, 5].

Поряд із GTFS існують й інші міжнародні стандарти обміну транспортними даними. Одним із них є NeTEx (Network Timetable Exchange) – європейський стандарт на основі XML, призначений для детального опису транспортних мереж, маршрутів, тарифів та розкладів. NeTEx забезпечує значно ширші можливості моделювання складних транспортних систем, проте є більш складним у впровадженні [6, 7]. Для передачі оперативної інформації про рух транспорту використовується стандарт SIRI (Service Interface for Real-time Information), який визначає набір інтерфейсів для обміну даними в реальному часі, зокрема щодо поточних позицій транспортних засобів, прогнозів прибуття та повідомлень про затримки [8].

У межах даної роботи не ставиться завдання повної реалізації імпорту або експорту GTFS-даних, однак структура предметної області має багато спільного з цим стандартом. Зокрема, у системі також виділяються маршрути, зупинки, зв’язки між маршрутами і зупинками, рейси та часи прибуття на зупинки. Такий підхід дозволяє побудувати логічно зрозумілу модель даних, яка в майбутньому може бути розширена для підтримки стандартних транспортних форматів.

Класифікувати системи відображення транспортної інформації можна за характером даних, які вони використовують. Статичні системи працюють із плановими розкладами та не враховують фактичне місцезнаходження транспорту. Системи реального часу відображають актуальні координати транспортних засобів і прогнозований час прибуття. Гібридні системи поєднують статичні розклади з окремими динамічними елементами [9]. Розроблювана система належить переважно до статичних систем, оскільки її основною метою є зручне відображення планового розкладу та маршрутної інформації.

За способом доступу до інформації транспортні інформаційні системи можна поділити на стаціонарні інформаційні табло, мобільні застосунки та веб-системи. Стаціонарні табло зручні на зупинках, але потребують додаткового обладнання та обслуговування. Мобільні застосунки забезпечують високий рівень зручності, проте вимагають встановлення на пристрій. Веб-системи є

універсальним варіантом, оскільки доступні з будь-якого пристрою, який має браузер [10]. Дослідження Macedo et al. на основі 655 опитувань у Португалії та Швеції встановили, що молоде покоління надають перевагу мобільним застосункам, тоді як старші пасажири переважно покладаються на фізичні табло на зупинках. Ця відмінність свідчить про необхідність мультиплатформного підходу, зокрема розробки веб-орієнтованих систем, доступних з будь-якого пристрою [11].

Дослідження впливу транспортних інформаційних систем на поведінку пасажирів демонструють переконливі кількісні результати. Walkins et al. провели детальне дослідження у Сіетлі, де порівнювали час очікування користувачів мобільного застосунку OneBusAway та пасажирів, які покладались на статичні розклади. Результати показали, що користувачі застосунку повідомляли про середній час очікування 7,5 хвилин проти 9,9 хвилин у контрольній групі – скорочення на 31%. Навіть статичні системи, які просто відображають плановий розклад у зручному форматі, суттєво підвищують задоволеність пасажирів та сприяють збільшенню кількості користувачів громадського транспорту [12, 13].

Масштабне імітаційне дослідження Paulsen et al. у Копенгагені охопило понад 800 тисяч поїздок на добу та моделювало чотири сценарії рівня інформованості пасажирів. Навіть базовий рівень інформування – відображення статичного планового розкладу (сценарій R1) – дав скорочення узагальнених витрат часу на 43% порівняно зі сценарієм повної відсутності інформації (сценарій R0). Це підтверджує, що розробка системи відображення навіть статичного розкладу має значний практичний ефект для пасажирів [14].

Серед основних видів міського наземного транспорту, для яких доцільно створювати інформаційні системи, можна виділити автобуси, тролейбуси, трамваї та маршрутні таксі. Саме ці види транспорту є поширеними в багатьох українських містах, зокрема у Вінниці. Для них характерні часті зупинки, наявність кількох напрямків руху, залежність розкладу від типу дня та потреба в зручному відображенні маршрутів на карті.

В Україні питання цифровізації громадського транспорту набуває особливого значення. Більшість міських транспортних підприємств країни досі не мають повноцінних цифрових систем управління розкладами та інформування пасажирів. Навіть у великих містах якість транспортних застосунків суттєво поступається аналогічним рішенням у країнах ЄС. Ця ситуація обумовлена браком фінансування транспортної галузі, відсутністю стандартизованих підходів до цифровізації та розрізненістю транспортних операторів. Вінниця є одним з небагатьох українських міст, де питанням цифровізації транспорту приділяється підвищена увага, однак і тут існуючі рішення не повністю задовольняють потреби мешканців.

Таким чином, аналіз предметної області демонструє, що системи відображення розкладу громадського транспорту є соціально значущими та технічно складними об’єктами. Їх розробка потребує врахування специфіки транспортної галузі, міжнародних стандартів представлення даних, особливостей поведінки різних груп користувачів та вимог до надійності й продуктивності програмного забезпечення.

1.2 Огляд та порівняння існуючих аналогів систем розкладу транспорту

Ринок систем інформування пасажирів громадського транспорту сьогодні є досить розвиненим і включає як глобальні багатофункціональні платформи, так і локальні міські рішення. Вони відрізняються за підходами до збору даних, архітектурою, рівнем масштабованості, якістю оновлення інформації та доступністю для різних регіонів. Для обґрунтування доцільності розробки нового веб-застосунку необхідно детально проаналізувати наявні аналоги, їх функціональні можливості та обмеження, а також виявити ключові недоліки, які залишаються невирішеними.

Одним із найвідоміших глобальних рішень є Moovit, що належить компанії Moovit (придбана Intel у 2020 році). Система охоплює понад 3400 міст у більш ніж 110 країнах світу та позиціонується як універсальний інструмент для планування поїздок громадським транспортом. Основною особливістю Moovit є

використання краудсорсингової моделі даних, коли інформація про рух транспорту частково формується на основі активності користувачів застосунку. Це дозволяє оперативно оновлювати дані навіть у відсутності офіційних транспортних фідів, таких як GTFS [4]. Водночас саме ця особливість є і слабкою стороною системи: точність даних суттєво залежить від кількості активних користувачів у конкретному регіоні. У великих містах України сервіс працює відносно стабільно, однак у середніх і малих містах, зокрема у Вінниці, якість даних є обмеженою, що ускладнює використання сервісу як основного джерела розкладу [15].

Іншим популярним рішенням є CityMapper, розроблений британською компанією CityMapper Ltd. Його головною перевагою є глибока оптимізація мультимодальних маршрутів, яка включає не лише традиційний громадський транспорт, але й велосипеди, таксі, сервіси прокату самокатів та пішохідні маршрути. Система відзначається високим рівнем UX-дизайну та інтелектуальними алгоритмами побудови найшвидших маршрутів. Однак суттєвим недоліком є географічна обмеженість: CityMapper підтримує лише великі мегаполіси, переважно у США, Великій Британії та Західній Європі. Українські міста практично не охоплені, що робить сервіс непридатним для локального використання [16].

OpenTripPlanner (OTP) є прикладом відкритої системи для планування маршрутів, що розповсюджується під ліцензією LGPL. Це потужний інструмент, який підтримує стандарти GTFS та OpenStreetMap і дозволяє будувати складні мультимодальні транспортні графи. Його архітектура базується на серверному Java-ядрі, що забезпечує високу гнучкість та масштабованість. Проте ця ж особливість створює суттєві труднощі при впровадженні: система потребує значних обчислювальних ресурсів, складного налаштування транспортних графів і глибокої технічної експертизи. Для невеликих міст впровадження OTP часто є надмірним і економічно недоцільним, оскільки функціональність системи перевищує реальні потреби локальних транспортних мереж [17].

Серед рішень, орієнтованих на український ринок, найбільш відомим є EasyWay. Це веб-орієнтований сервіс, який охоплює приблизно 50 міст України та надає інформацію про маршрути громадського транспорту. Його перевагою є доступність українською мовою та відносно проста навігація: користувач може переглядати маршрути автобусів, тролейбусів і трамваїв, а також їх відображення на карті [18]. Водночас система має низку системних недоліків. По-перше, оновлення даних у багатьох містах відбувається нерегулярно, що призводить до розбіжностей між фактичним і відображеним розкладом. По-друге, відсутній повноцінний механізм фільтрації транспорту за типами або напрямками. По-третє, інтерфейс системи морально застарілий і недостатньо адаптований для мобільних пристроїв. Крім того, відсутній детальний перегляд розкладу по конкретних зупинках у структурованому вигляді, що обмежує зручність використання.

Окрему категорію становлять локальні міські рішення, одним із яких є застосунок «Vinnytsia Transport». Це офіційний продукт Вінницької міської ради, який забезпечує моніторинг руху трамваїв у реальному часі на основі GPS-даних. Система є важливим кроком у напрямку цифровізації транспорту, оскільки дозволяє відстежувати рух трамваїв із високою точністю [19]. Проте функціональність сервісу має певні обмеження: користувачі часто відзначають брак зручного перегляду статичного розкладу в структурованій формі для планування поїздок заздалегідь, через що вимушені використовувати альтернативні локальні бази даних.

Подібним прикладом є застосунок «CityBus Харків», який забезпечує доступ до маршрутів метро, автобусів, тролейбусів і трамваїв. Він відображає транспортні маршрути на карті та надає інформацію про розклад руху. Водночас ключовим недоліком системи є її реалізація виключно у вигляді мобільного застосунку для ОС Android без повноцінної вебверсії чи підтримки інших платформ. Це знижує універсальність системи, оскільки обмежує використання лише смартфонами та унеможливлює роботу з настільних пристроїв [20]. Аналогічні рішення існують і в інших містах України, наприклад у Львові та

Дніпрі, проте всі вони мають локальний характер, прив'язані до комунальних підприємств конкретного міста, мають різну архітектуру та не можуть бути масштабовані як єдина універсальна платформа.

Найбільш технологічно розвиненим міським рішенням в Україні є муніципальний застосунок «Київ Цифровий». У межах транспортного модуля він забезпечує відображення руху всього громадського транспорту столиці в реальному часі на основі GPS-трекінгу та повної інтеграції з автоматизованою системою оплати проїзду (АСОП). Система має сучасний інтерфейс, високу стабільність роботи, об'єднує функції трекінгу, купівлі квитків та моніторингу роботи метрополітену. Однак вона залишається повністю локалізованою під інфраструктуру Києва і не передбачає можливості масштабування або інтеграції даних інших міст без глибокої модифікації архітектури та переналаштування джерел даних [21].

Узагальнюючи проведений аналіз, можна виділити кілька ключових проблем існуючих рішень. По-перше, глобальні платформи або не підтримують більшість українських міст, або мають низьку точність даних через відсутність офіційних інтеграцій. По-друге, локальні міські системи є фрагментованими, створеними під конкретні міста і не мають універсальної архітектури. По-третє, значна частина рішень орієнтована на навігацію в реальному часі, тоді як потреба у зручному перегляді статичного розкладу по зупинках залишається недостатньо закритою. По-четверте, не всі системи мають повноцінний веб-інтерфейс, що обмежує доступність для широкого кола користувачів [22, 23].

Проведений аналіз демонструє відсутність універсального рішення, яке б одночасно забезпечувало підтримку різних типів транспорту, зручний веб-інтерфейс, актуальні дані та можливість адаптації під конкретне місто. Це підтверджує доцільність розробки власного рішення, орієнтованого на реальні потреби вітчизняних транспортних операторів та пасажирів.

1.3 Формування функціональних та нефункціональних вимог до системи

Коректне визначення вимог є важливим етапом розробки програмного забезпечення, оскільки саме вимоги визначають функціональність майбутньої системи, її обмеження, якість роботи та відповідність очікуванням користувачів. Вимоги до системи формуються на основі аналізу предметної області, огляду існуючих аналогів та практичних сценаріїв використання веб-застосунку пасажирами громадського транспорту.

Перш ніж формулювати вимоги, визначимо основні групи зацікавлених сторін системи. Основною цільовою аудиторією системи є пасажири громадського транспорту. Їм необхідно швидко знаходити потрібний маршрут, переглядати розклад, визначати зупинки маршруту, бачити транспортну інформацію на карті та зберігати часто використовувані маршрути. Додатковою групою користувачів можуть бути працівники транспортного підприємства або IT-фахівці, які відповідають за наповнення бази даних та підтримку системи. У межах даної роботи основна увага приділяється саме пасажирському веб-інтерфейсу.

Функціональні вимоги описують конкретні можливості, які повинна надавати система. Вони визначають, які дії може виконувати користувач і які дані повинна обробляти система. Для розроблюваного застосунку функціональні вимоги доцільно згрупувати за основними модулями: модуль маршрутів, модуль деталей маршруту та розкладу, модуль карти зупинок, модуль пошуку, модуль обраних маршрутів і модуль взаємодії з API [24].

Модуль перегляду маршрутів повинен забезпечувати відображення повного переліку маршрутів громадського транспорту. Для кожного маршруту необхідно показувати номер, назву, тип транспорту, час першого та останнього рейсу. Система повинна підтримувати фільтрацію маршрутів за видом транспорту: автобус, тролейбус, трамвай і маршрутне таксі. Також має бути реалізований пошук маршруту за номером, назвою або зупинкою, через яку проходить маршрут.

Для детального ознайомлення з обраним напрямком призначений модуль деталей маршруту. Він акумулює повну інформацію, демонструючи користувачеві базові параметри, як-от номер, назву, тип транспорту й часові межі роботи, та доповнює їх переліком усіх зупинок у прямому й зворотному напрямках. Важливою складовою є цифрова карта, де траєкторія руху візуалізується у вигляді суцільної лінії, побудованої за географічними координатами зупинок, а її забарвлення чітко відповідає кольоровому кодуванню конкретного виду транспорту.

Модуль перегляду розкладу повинен забезпечувати відображення часу прибуття транспорту на кожну зупинку маршруту. Розклад повинен мати перемикання за напрямком руху: прямий і зворотний. Також необхідно передбачити фільтрацію за типом дня: будні та вихідні. Часи прибуття повинні відображатися у зручному форматі у вигляді окремих часових позначок.

У випадках, коли необхідно побудувати поїздку між двома конкретними точками, задіюється модуль розширеного пошуку. Пасажир обирає початкову та кінцеву зупинки, після чого алгоритм визначає всі доступні прямі маршрути, які проходять через обидва пункти у правильній хронологічній послідовності. Для кожного знайденого варіанту система розраховує та демонструє найближчі години відправлення, орієнтовну тривалість поїздки до фінішу та точний час очікування на посадку.

Модуль роботи зі зупинками повинен забезпечувати відображення інтерактивної карти з маркерами всіх зупинок транспортної мережі. Користувач повинен мати можливість знайти зупинку за назвою, натиснути на маркер і переглянути список маршрутів, які проходять через цю зупинку. Для маршрутів у спливаючому вікні необхідно відображати номер, назву, тип транспорту, відповідну іконку та колір.

Модуль карти повинен забезпечувати інтерактивну роботу з просторовими даними. На сторінці деталей маршруту карта повинна відображати лінію конкретного маршруту та його зупинки. На сторінці карти зупинок карта повинна відображати всі зупинки, а при виборі зупинки – маршрути, які через

неї проходять. Лінії маршрутів на карті повинні розміщуватися нижче маркерів, щоб не заважати натисканню на зупинки.

Для персоналізації взаємодії розроблено модуль обраних маршрутів. Користувач може формувати власний список швидкого доступу за допомогою кліку на іконку зірочки, яка працює як для додавання, так і для видалення елементів. Щоб інформація не зникала після закриття вкладки, дані зберігаються у локальному сховищі браузера між сесіями. Якщо список обраних маршрутів порожній, система повинна показувати відповідне повідомлення та підказку користувачу.

Усі клієнтські дії обслуговує модуль взаємодії з API, який організовує отримання актуальних даних із серверної частини через HTTP-запити. Він містить набір функцій для запиту загальних списків, детальних описів конкретного маршруту чи зупинки, графіків прибуття, а також передає параметри для звичайного пошуку та розширеного пошуку сполучень між двома зупинками.

Нефункціональні вимоги визначають якісні характеристики системи. До них належать вимоги до зручності використання, продуктивності, надійності, підтримуваності, масштабованості, безпеки та адаптивності інтерфейсу. Для системи розкладу громадського транспорту ці вимоги мають важливе значення, оскільки користувач часто звертається до неї в реальних умовах поїздки та очікує швидкої відповіді.

Зручність використання гарантується простим інтерфейсом, де ключові функції, такі як пошук, перегляд графіків, робота з картою та керування обраним, доступні одразу без блукання складним меню. Візуальний комфорт досягається завдяки зрозумілій іконографіці, кольоровому кодуванню за видами транспорту та чіткому розділенню інформації на окремі картки [25].

З погляду продуктивності система повинна швидко завантажувати основні дані та забезпечувати миттєву реакцію на дії користувача. Фільтрація за типом транспорту й базовий пошук у списку маршрутів мають виконуватися на клієнтській стороні без додаткових затримок. Запити до сервера повинні

виконуватися лише тоді, коли потрібна додаткова інформація: деталі маршруту, розклад, пошук за зупинкою або пошук маршруту між двома зупинками.

Паралельно з цим критерій надійності забезпечує стабільну роботу при виникненні помилок або відсутності зв'язку. Замість некоректного руйнування інтерфейсу, система покаже інформативне повідомлення, якщо рейси не знайдено або сервер тимчасово недоступний.

Критерій підтримуваності системи вимагає чіткого розмежування вихідного коду на незалежні логічні компоненти. Клієнтська частина повинна містити окремі сторінки для списку маршрутів, деталей маршруту, карти зупинок та обраних маршрутів. Запити до API повинні бути винесені в окремий сервісний модуль. Серверна частина повинна мати окремі маршрути для роботи з маршрутами, зупинками, розкладами та пошуком.

З погляду масштабованості структура бази даних повинна дозволяти додавання нових маршрутів, зупинок, рейсів і часів прибуття без зміни загальної архітектури системи. У майбутньому система може бути розширена додатковими функціями, наприклад адміністративним модулем для керування даними, імпортом транспортних даних або підтримкою даних у реальному часі.

Безпека програмного комплексу базується на принципі суворої ізоляції внутрішніх ресурсів, що унеможливлює будь-яку пряму взаємодію фронтенду із базою даних. Коригування чи отримання інформації мають здійснюватися виключно через захищений серверний API, який виступає посередником. Сервер повинен перевіряти вхідні параметри запитів і повертати коректні коди помилок у разі некоректних або відсутніх даних.

Для забезпечення високої адаптивності інтерфейс повинен коректно відображатися на різних розмірах екрана. Веб-застосунок має бути доступним як на настільних комп’ютерах, так і на ноутбуках, планшетах і смартфонах. Елементи інтерфейсу повинні мати достатні відступи, зручні для натискання кнопки та читабельні написи.

Отже, сформовані функціональні та нефункціональні вимоги визначають основу для подальшого проєктування системи. Вони охоплюють основні

потреби користувача: швидкий пошук маршруту або зупинки, перегляд розкладу, роботу з картою, пошук транспорту між двома зупинками та збереження обраних маршрутів.

1.4 Вибір технологічного стеку та інструментальних засобів розробки

Вибір технологічного стеку є важливим етапом розробки програмної системи, оскільки він визначає швидкість створення продукту, зручність підтримки, продуктивність і можливості подальшого розширення. Для розроблюваної системи було обрано набір відкритих і поширених технологій, які добре підходять для створення веб-застосунків з інтерактивним інтерфейсом, REST API, реляційною базою даних та картографічним модулем [26].

Для клієнтської частини системи обрано бібліотеку React.js версії 18, розроблену компанією Meta. React реалізує компонентний підхід до побудови інтерфейсів: кожен елемент UI є незалежним компонентом з власним станом та логікою відображення. Використання Virtual DOM дозволяє React мінімізувати кількість операцій із справжнім DOM браузера, що забезпечує продуктивний рендеринг при частих оновленнях даних. Компонентна модель дозволяє повторно використовувати елементи інтерфейсу (картки маршрутів, таблиці розкладів, пошукові поля) в різних частинах застосунку без дублювання коду [27].

Порівняння React з основними альтернативами демонструє обґрунтованість вибору. Vue.js більш доречним фреймворком з нижчим порогом входу, проте з меншою гнучкістю при побудові складних інтерфейсів та меншим ринком праці [28]. Angular – повноцінний enterprise-фреймворк з жорсткою TypeScript-архітектурою, що є надмірно складним для масштабу даного проєкту та значно збільшує час початкового завантаження застосунку [29]. Чистий HTML/CSS/JavaScript без фреймворку не забезпечує зручного управління станом застосунку та призводить до ускладнення коду при зростанні складності інтерфейсу. React є оптимальним вибором завдяки балансу між гнучкістю, продуктивністю, зрілістю екосистеми та широкою документацією [30].

Для керування станом клієнтського застосунку використовуються вбудовані хуки React, зокрема useState та useEffect. Хук useState застосовується для зберігання локального стану компонентів: списку маршрутів, активного фільтра, пошукового рядка, обраних маршрутів, вибраної зупинки або результатів пошуку. Хук useEffect використовується для завантаження даних із сервера при відкритті сторінки або зміні параметрів, наприклад маршруту, напрямку чи типу дня [31].

Використання вбудованих хуків React є достатнім для поточного масштабу системи. Застосунок не має складної багаторівневої глобальної бізнес-логіки, а більшість даних використовується в межах конкретних сторінок. Наприклад, список маршрутів потрібний на головній сторінці, розклад – на сторінці деталей маршруту, а список зупинок – на сторінці карти. Тому використання додаткових бібліотек глобального стану не є обов'язковим і могло б лише ускладнити структуру проєкту.

Для відображення картографічних даних обрано бібліотеку Leaflet.js – провідне відкрите рішення для інтерактивних карт у браузері. Leaflet.js відрізняється невеликим розміром (~40 КБ у стиснутому вигляді), продуктивністю та широкими можливостями розширення через плагіни [32]. Інтеграція з React реалізується через бібліотеку React-Leaflet, що надає React-компоненти для карти, маркерів, ліній та спливаючих підказок [32, 33]. Як картографічну основу обрано OpenStreetMap – відкриту базу картографічних даних з безкоштовними тайлами та детальним покриттям українських міст [34]. Поєднання Leaflet.js та OpenStreetMap є стандартним підходом для відкритих транспортних систем і не потребує платних API-ключів на відміну від Google Maps Platform.

Для серверної частини системи обрано платформу Node.js з фреймворком Express.js [35]. Node.js є середовищем виконання JavaScript поза браузером, побудованим на движку V8 від Google. Ключовою особливістю є асинхронна неблокуюча модель виконання вводу-виводу, що робить його ефективним для обробки HTTP-запитів [36]. Перевага Node.js у контексті даного проєкту полягає

у використанні JavaScript як єдиної мови програмування для обох частин системи. Це спрощує розробку, дозволяє повторно використовувати допоміжний код (наприклад, функції форматування часу або валідації) та зменшує необхідний обсяг знань для підтримки проєкту.

Порівняння альтернатив серверної платформи. Python з фреймворком Django або FastAPI є популярним вибором для веб-API завдяки лаконіному синтаксису та потужним бібліотекам. Однак використання Python на сервері при React означає роботу з двома різними мовами, що ускладнює підтримку. Java зі Spring Boot забезпечує найвищу продуктивність та безпеку, але є надмірно складним для проєкту даного масштабу та суттєво збільшує час розробки. Node.js з Express.js є оптимальним вибором, що забезпечує необхідну продуктивність і при цьому дозволяє зберегти єдину мову програмування в межах всього проєкту.

Express.js – мінімалістичний та гнучкий фреймворк для Node.js, що є стандартом для побудови REST API. Express.js надає базові засоби: маршрутизацію HTTP-запитів, middleware-архітектуру для обробки запитів у конвеєрному режимі, обробку помилок. Незважаючи на переваги NestJS у великих проєктах з командою розробників, Express.js є більш доцільним завдяки простішому входженню та меншій кількості обов’язкових абстракцій [37].

Як систему управління базами даних обрано MySQL 8 – одну з найпоширеніших у світі реляційних СУБД з відкритим вихідним кодом, що активно розвивається компанією Oracle. MySQL є оптимальним вибором для зберігання транспортних даних з огляду на їх структуровану реляційну природу: маршрути мають зупинки, зупинки входять до рейсів, рейси мають часові таблиці – всі ці зв’язки природно виражаються у реляційній моделі з чітко визначеними зовнішніми ключами. MySQL підтримує транзакції з дотриманням властивостями ACID (для таблиць InnoDB), складні SQL-запити із підзапитами та об’єднаннями, а також зберігання координат зупинок у полях типу FLOAT. Порівняно з PostgreSQL MySQL є простішим у встановленні та налаштуванні, що є практичною перевагою для даного проєкту. MongoDB як документо-

орієнтована СУБД є менш придатною, оскільки транспортні дані мають чітко визначену реляційну структуру з множинними зв’язками між сутностями [38].

Для взаємодії Node.js з MySQL обрано ORM-бібліотеку Sequelize у поєднанні з драйвером mysql2. Sequelize забезпечує об'єктно-реляційне відображення та дозволяє описувати таблиці бази даних як JavaScript-класи. Ключові переваги для даного проєкту: автоматичний захист від SQL-ін'єкцій через параметризовані запити; метод sync() для автоматичного створення таблиць на основі визначених моделей; декларативне визначення зв'язків між моделями (маршрути – зупинки – розклади) [39]. Бібліотека mysql2 є сучасним та продуктивним драйвером для MySQL, що підтримує Promise-based API та підготовлені запити [40]. Усі компоненти технологічного стеку розроблюваної системи узагальнено в таблиці 1.1.

Таблиця 1.1 Технологічний стек розроблюваної системи

Компонент системи	Обрана технологія	Основне обґрунтування вибору
Клієнтська частина	React.js 18	Компонентність, Virtual DOM, зрілість екосистеми
Управління станом	React hooks	Простота реалізації, відповідність масштабу проєкту, відсутність зайвих залежностей
Карти та геодані	Leaflet.js + OpenStreetMap	Відкритість, безкоштовність, покриття України
Серверні частина	Node.js + Express.js	Єдина мова з клієнтом, асинхронність
База даних	MySQL 8	Реляційна модель, геодані, ACID
ORM	Sequelize	Міграції, захист від SQL-ін'єкцій
HTTP-клієнт	Axios	Interceptors, зручна обробка помилок

Обраний технологічний стек забезпечує реалізацію всіх визначених функціональних вимог: React.js, React hooks, Axios та React-Leaflet відповідають за побудову інтерфейсу, взаємодію з API, перегляд розкладів, пошук і відображення карти зупинок; Node.js з Express.js забезпечують роботу серверної частини та обробку HTTP-запитів; MySQL у поєднанні із Sequelize використовується для зберігання й отримання структурованих транспортних даних. Використання відкритих і безкоштовних технологій, зокрема OpenStreetMap замість комерційних картографічних сервісів, дозволяє зменшити залежність системи від зовнішніх платних платформ і спростити її подальше розгортання та підтримку.

Висновки до розділу 1

У першому розділі було проаналізовано предметну область систем відображення розкладу руху громадського транспорту. Визначено, що якісне інформаційне забезпечення пасажирів є важливою умовою зручного користування міським транспортом. Також розглянуто стандарт GTFS, логіка якого є близькою до структури даних розроблюваної системи.

Було проведено огляд існуючих аналогів, зокрема Moovit, Citymapper, OpenTripPlanner, EasyWay та міських транспортних застосунків. Аналіз показав, що наявні рішення мають певні обмеження: неповну підтримку українських міст, складність локального розгортання, залежність від зовнішніх сервісів або недостатньо зручний перегляд статичного розкладу.

На основі аналізу сформовано функціональні та нефункціональні вимоги до системи. Розроблюваний застосунок повинен забезпечувати перегляд маршрутів, пошук і фільтрацію, відображення розкладу, роботу з картою зупинок, пошук маршруту між двома зупинками та збереження обраних маршрутів. Також обґрунтовано вибір технологічного стеку: React.js для клієнтської частини, Node.js та Express.js для серверної логіки, MySQL і Sequelize для збереження даних, а також Leaflet і OpenStreetMap для роботи з картою. Обрані технології відповідають вимогам системи та створюють основу для подальшого проєктування й реалізації застосунку.

РОЗДІЛ 2
ПРОЄКТУВАННЯ БАГАТОКОМПОНЕНТНОЇ СИСТЕМИ

2.1 Архітектура багатокомпонентної системи

Проєктування архітектури програмної системи є одним із ключових етапів розробки, оскільки саме архітектура визначає загальну структуру застосунку, принципи взаємодії між його компонентами, спосіб збереження та обробки даних, а також можливості подальшого розширення системи [41]. Для системи відображення розкладу громадського транспорту було обрано класичну триланкову клієнт-серверну архітектуру з розподілом відповідальності між клієнтською частиною, серверною частиною та базою даних [42].

Розроблювана система призначена для надання користувачу зручного доступу до інформації про маршрути громадського транспорту, зупинки, розклад руху, напрямки руху та інтерактивну карту. Основна вимога до архітектури полягає в тому, щоб інтерфейс користувача був швидким і зручним, серверна частина забезпечувала централізовану обробку запитів, а база даних зберігала структуровану транспортну інформацію з урахуванням зв'язків між маршрутами, зупинками, рейсами та часами прибуття.

У межах системи виділено три основні логічні рівні:

1) рівень представлення, який відповідає за інтерфейс користувача;
2) рівень бізнес-логіки, який реалізує REST API та обробляє запити клієнта;
3) рівень даних, який забезпечує збереження транспортної інформації в реляційній базі даних [42, 43].

Рівень представлення реалізований у вигляді односторінкового застосунку (Single Page Application, SPA) на основі бібліотеки React.js. Він відповідає за відображення користувацького інтерфейсу, обробку взаємодій користувача, відображення карти, фільтрацію та пошук на стороні клієнта, а також збереження обраних маршрутів у локальному сховищі браузера. Клієнтська частина виконується безпосередньо в браузері користувача та взаємодіє з сервером виключно через асинхронні HTTP-запити до REST API .

Рівень бізнес-логіки реалізований за допомогою Node.js з використанням фреймворку Express.js. Серверна частина приймає запити від клієнта, звертається до бази даних, формує необхідні структури даних і повертає результат у форматі JSON. Сервер відповідає за отримання списку маршрутів, отримання детальної інформації про маршрут, отримання списку зупинок, пошук маршрутів за зупинками, формування розкладу по зупинках та розширений пошук маршруту між двома зупинками.

Рівень даних реалізований на основі реляційної СУБД MySQL 8. Реляційна база даних зберігає всі структуровані дані системи: інформацію про міста, маршрути, зупинки, порядок зупинок на маршрутах, рейси та часи прибуття на зупинки із чітко визначеними зв'язками між сутностями. Для взаємодії між рівнем бізнес-логіки та рівнем даних використовується ORM-бібліотека Sequelize, яка забезпечує об'єктно-реляційне відображення та захист від SQL-ін'єкцій.

Загальний процес взаємодії між компонентами системи відбувається наступним чином. Користувач відкриває веб-застосунок у браузері та отримує статичні файли (HTML, CSS, JavaScript) від веб-сервера. Після завантаження React-застосунок починає відправляти асинхронні HTTP-запити до REST API для отримання транспортних даних. Express-сервер обробляє запити, звертається до бази даних через Sequelize та повертає відповідь у форматі JSON. React-застосунок отримує дані та оновлює інтерфейс без перезавантаження сторінки.

Для відображення картографічної інформації використовується бібліотека Leaflet.js у поєднанні з тайлами OpenStreetMap. Картографічні дані завантажуються безпосередньо з серверів OpenStreetMap у браузері користувача, не навантажуючи власний сервер системи. Це рішення є оптимальним з точки зору продуктивності та вартості, оскільки не потребує платних API-ключів.

Для збереження обраних маршрутів використовується механізм локального сховища (localStorage) браузера. Це дозволяє зберігати список улюблених маршрутів між сесіями без створення облікового запису користувача

та без додаткового серверного зберігання. Такий варіант є оптимальним для системи, орієнтованої на швидке використання пасажиром без авторизації.

Важливою архітектурною особливістю є підтримка багатомістності. Система розроблена з урахуванням можливості роботи з даними кількох міст. Кожна сутність бази даних (маршрут, зупинка) прив'язана до конкретного міста через зовнішній ключ. Клієнтська частина зберігає вибране місто в локальному сховищі та передає ідентифікатор міста як параметр у відповідних запитах до API, що дозволяє серверу фільтрувати дані відповідно до обраного міста.

Схему загальної архітектури системи наведено на рисунку 2.1. Стрілки відображають напрямок передачі даних між компонентами системи. Клієнт і сервер взаємодіють через HTTP/REST API, сервер і база даних – через ORM-запити Sequelize/SQL.

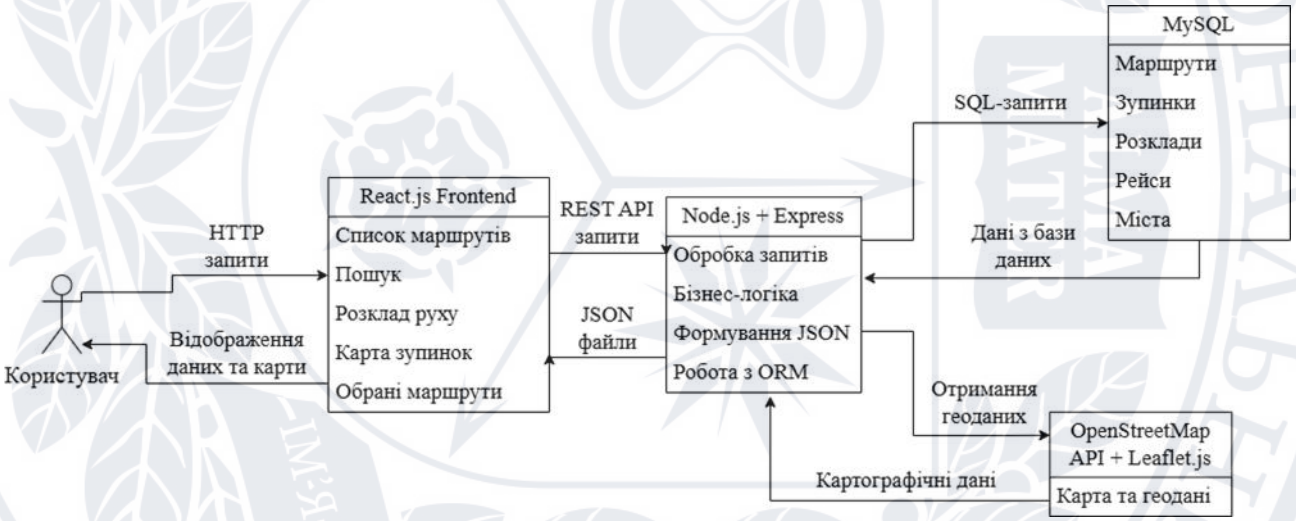

Рисунок 2.1 – Загальна архітектура багатокомпонентної

Обрана архітектура забезпечує логічний поділ системи на незалежні частини. Клієнтська частина може змінюватися без істотних змін у структурі бази даних, серверна частина може розширюватися новими API-ендпоінтами, а база даних може масштабуватися залежно від кількості транспортної інформації. Такий підхід відповідає вимогам до простоти підтримки, модульності та подальшого розвитку системи.

2.2 Проєктування бази даних та моделей даних

Проєктування бази даних є важливим етапом створення системи розкладу громадського транспорту, оскільки саме структура даних визначає можливість коректного збереження маршрутів, зупинок, напрямків руху, рейсів та часу прибуття транспорту на кожну зупинку. Для розроблюваної системи обрано реляційну модель даних, оскільки предметна область має чітко виражені сутності та зв'язки між ними [44].

Концептуальне проєктування бази даних розпочинається з визначення основних сутностей предметної області. У процесі аналізу предметної області було виділено шість основних сутностей: Місто (City), Маршрут (Route), Зупинка (Stop), Порядок зупинок маршруту (RouteStop), Рейс (Trip) та Час зупинки (StopTime). Кожна з цих сутностей відповідає окремій таблиці бази даних і виконує конкретну роль у збереженні транспортної інформації.

Сутність «Місто» (Cities) описує населений пункт, для якого в системі зберігаються транспортні дані. Кожне місто характеризується унікальним ідентифікатором id, назвою міста name, географічною широтою lat, географічною довготою lng та ознакою наявності транспортних даних has_data. Координати міста використовуються для центрування карти при виборі відповідного населеного пункту, а поле has_data дозволяє відрізняти міста, для яких уже внесено транспортні дані, від міст, які можуть бути додані до системи пізніше. Додавання цієї сутності дозволяє масштабувати систему на кілька міст без зміни основної структури бази даних. Атрибути сутності Cities наведено в таблиці 2.2.

Таблиця 2.2 Структура сутності Cities

Поле	Тип даних	Опис
id	INT, PK, AI	Унікальний ідентифікатор міста
name	VARCHAR(255)	Назва міста
lat	FLOAT	Географічна широта центру міста
lng	FLOAT	Географічна довгота центру міста

Поле	Тип даних	Опис
has_data	TINYINT(1)	Ознака наявності транспортних даних
createdAt	DATETIME	Дата створення запису
updatedAt	DATETIME	Дата останнього оновлення запису

Сутність «Маршрут» (Routes) описує транспортний маршрут у межах конкретного міста. Вона містить загальні характеристики маршруту: унікальний ідентифікатор id, номер маршруту number, назву маршруту name та тип транспорту type. Номер маршруту зберігається як рядок, оскільки в транспортних системах можуть використовуватися не лише числові, а й буквено-цифрові позначення. Поле type визначає вид транспорту та має фіксований перелік допустимих значень: bus, trolleybus, tram, minibus. Крім того, таблиця містить зовнішній ключ CityId, який визначає місто, до якого належить маршрут. Атрибути сутності Routes наведено в таблиці 2.3.

Таблиця 2.3 Структура сутності Routes

Поле	Тип даних	Опис
id	INT, PK, AI	Унікальний ідентифікатор маршруту
number	VARCHAR(10)	Номер маршруту
name	VARCHAR(255)	Назва маршруту
type	ENUM	Тип транспорту: bus, trolleybus, tram, minibus
CityId	INT	Посилання на місто, до якого належить маршрут
createdAt	DATETIME	Дата створення запису
updatedAt	DATETIME	Дата останнього оновлення запису

Сутність «Зупинка» (Stops) описує фізичну зупинку громадського транспорту. Атрибути сутності: унікальний ідентифікатор id, назва зупинки name, географічна широта lat та географічна довгота lng. Кожна зупинка характеризується унікальним ідентифікатором id, назвою name географічною широтою lat та географічною довготою lng у форматі FLOAT. Наявність

координат дозволяє відображати зупинки на інтерактивній карті та будувати лінії маршрутів за послідовністю зупинок.

Також таблиця містить зовнішній ключ CityId, який вказує, у межах якого міста розташована зупинка. Завдяки цьому одна база даних може містити зупинки різних міст без змішування транспортних мереж між собою. Атрибути сутності Stops наведено в таблиці 2.4.

Таблиця 2.4 Структура сутності Stops

Поле	Тип даних	Опис
id	INT, PK, AI	Унікальний ідентифікатор зупинки
name	VARCHAR(255)	Назва зупинки
lat	FLOAT	Географічна широта зупинки
lng	FLOAT	Географічна довгота зупинки
CityId	INT	Посилання на місто, у межах якого розташована зупинка
createdAt	DATETIME	Дата створення запису
updatedAt	DATETIME	Дата останнього оновлення запису

Сутність «Порядок зупинок маршруту» (RouteStops) є проміжною таблицею між маршрутами та зупинками. Вона реалізує зв’язок багато-до-багатьох, оскільки один маршрут містить багато зупинок, а одна зупинка може входити до складу багатьох маршрутів. Крім зовнішніх ключів RouteId і StopId, таблиця містить поле order_number, яке визначає порядок зупинки на маршруті, та поле direction, яке визначає напрямок руху. Поле direction має два основні значення: forward для прямого напрямку та backward для зворотного. Атрибути сутності RouteStops наведено в таблиці 2.5.

Таблиця 2.5 Структура сутності RouteStops

Поле	Тип даних	Опис
id	INT, PK, AI	Унікальний ідентифікатор запису
RouteId	INT	Посилання на маршрут

Поле	Тип даних	Опис
StopId	INT	Посилання на зупинку
order_number	INT	Порядковий номер зупинки на маршруті
direction	ENUM	Напрямок руху: forward або backward
createdAt	DATETIME	Дата створення запису
updatedAt	DATETIME	Дата останнього оновлення запису

Сутність «Рейс» (Trips) описує конкретне проходження маршруту у визначений час. Рейс прив’язаний до маршруту через зовнішній ключ RouteId і містить такі атрибути: унікальний ідентифікатор id, тип дня day_type, напрямок руху direction та час відправлення від першої зупинки departure_time. Поле day_type дозволяє розділити розклад для будніх і вихідних днів, а поле direction визначає, чи належить рейс до прямого або зворотного напрямку руху. Атрибути сутності Trips наведено в таблиці 2.6.

Таблиця 2.6 Структура сутності Trips

Поле	Тип даних	Опис
id	INT, PK, AI	Унікальний ідентифікатор рейсу
RouteId	INT	Посилання на маршрут, до якого належить рейс
day_type	ENUM	Тип дня: weekday або weekend
direction	ENUM	Напрямок руху: forward або backward
departure_time	VARCHAR(5)	Час відправлення від першої зупинки
createdAt	DATETIME	Дата створення запису
updatedAt	DATETIME	Дата останнього оновлення запису

Сутність «Час зупинки» (StopTimes) описує час прибуття транспорту на конкретну зупинку в межах конкретного рейсу. Кожен запис містить унікальний ідентифікатор id, зовнішній ключ рейсу TripId, зовнішній ключ зупинки StopId, час прибуття arrival_time у форматі рядка «ГГ:ХХ» та порядковий номер зупинки

stop_order. Ця таблиця є найбільшою за обсягом і основною для формування детального розкладу по зупинках, оскільки саме в ній зберігається інформація про те, о котрій годині конкретний рейс прибуває на конкретну зупинку. Атрибути сутності StopTimes наведено в таблиці 2.7.

Таблиця 2.7 Структура сутності StopTimes

Поле	Тип даних	Опис
id	INT, PK, AI	Унікальний ідентифікатор запису
TripId	INT	Посилання на рейс
StopId	INT	Посилання на зупинку
arrival_time	VARCHAR(5)	Час прибуття транспорту на зупинку
stop_order	INT	Порядковий номер зупинки в межах рейсу
createdAt	DATETIME	Дата створення запису
updatedAt	DATETIME	Дата останнього оновлення запису

Логічну структуру бази даних з позначенням зв'язків між таблицями наведено на рисунку 2.1. Стрілки відображають зв'язки між сутностями із зазначенням кардинальності: один маршрут має багато рейсів, один рейс має багато часів зупинок, маршрут і зупинки пов'язані через таблицю RouteStops.

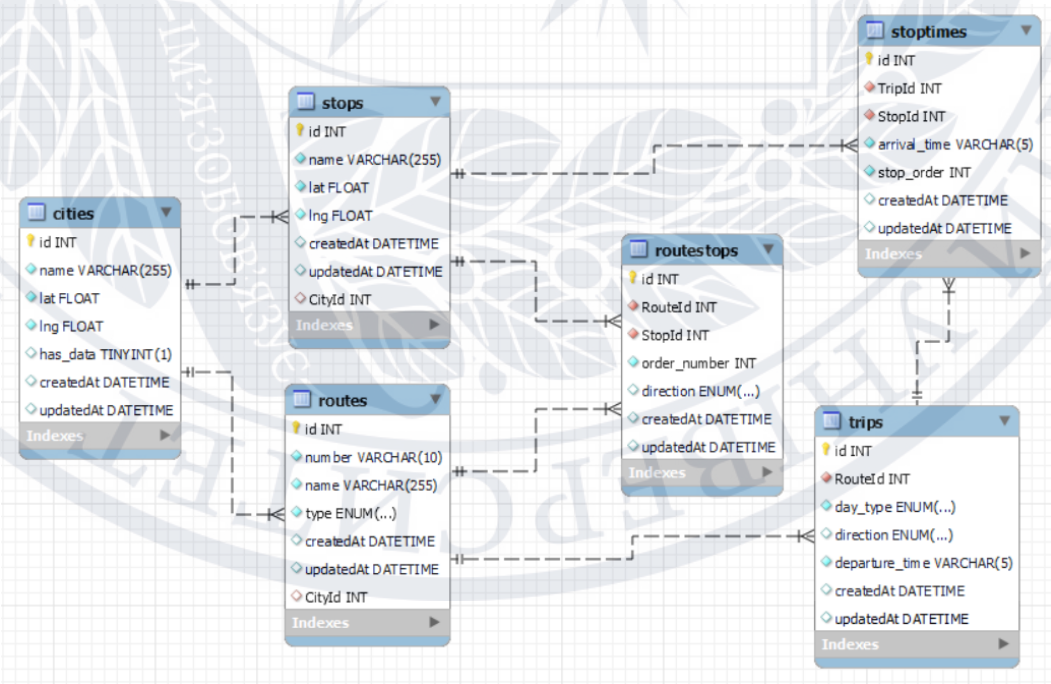

Рисунок. 2.2 – Логічна структура бази даних транспортної системи

У Sequelize ці зв’язки реалізуються за допомогою методів hasMany, belongsTo та belongsToMany. Сутність City має багато маршрутів і багато зупинок. Сутності Route та Stop пов’язуються між собою через проміжну модель RouteStop. Сутність Route має багато рейсів Trip, а кожен рейс має багато записів StopTime. Така структура відповідає реляційній природі транспортних даних і дозволяє ефективно отримувати як список маршрутів міста, так і детальний розклад по кожній зупинці [45].

Нормалізація бази даних дозволяє уникнути дублювання інформації. Наприклад, назва та координати зупинки зберігаються лише один раз у таблиці Stops, а її участь у різних маршрутах описується через таблицю RouteStops. Аналогічно, загальна інформація про маршрут зберігається в таблиці Routes, а конкретні рейси та часи прибуття винесені в окремі таблиці Trips і StopTimes.

Додавання таблиці Cities робить базу даних більш масштабованою. Якщо система використовується для одного міста, усі маршрути та зупинки прив’язуються до одного запису в таблиці Cities. Якщо в майбутньому потрібно буде додати інші міста, достатньо створити нові записи в таблиці Cities і пов’язати з ними відповідні маршрути та зупинки через поле CityId.

2.3 Алгоритми обробки та відображення даних розкладу

Основним функціональним завданням системи є формування зручного для користувача подання транспортних даних. Ця задача є нетривіальною з точки зору обробки даних, оскільки потребує агрегації інформації з кількох таблиць бази даних та формування структурованої відповіді, зручної для відображення в інтерфейсі. Для цього на серверній та клієнтській частинах реалізовано кілька алгоритмів: отримання списку маршрутів, формування розкладу по зупинках, пошук маршруту між двома зупинками, пошук маршрутів за проміжною зупинкою, побудова лінії маршруту на карті та збереження обраних маршрутів.

Алгоритм отримання списку маршрутів є найпростішим. Клієнтська частина виконує GET-запит до API, сервер отримує всі записи з таблиці Routes, сортує їх за номером маршруту та повертає у форматі JSON. На клієнті ці дані

відображаються у вигляді списку карток маршрутів. Кожна картка містить іконку типу транспорту, номер маршруту, тип, назву та час першого й останнього рейсу.

Алгоритм фільтрації маршрутів за типом транспорту виконується на клієнтській стороні. У стані компонента зберігається активний фільтр: all, bus, trolleybus, tram або minibus. Якщо обрано значення all, відображаються всі маршрути. Якщо користувач обирає конкретний тип транспорту, компонент автоматично перераховує відфільтрований список без звернення до сервера, що забезпечує миттєву відповідь інтерфейсу.

Пошук за номером або назвою маршруту також може виконуватися на клієнті, оскільки список маршрутів уже завантажений у пам'ять браузера. При введенні тексту в поле пошуку значення приводиться до нижнього регістру, після чого порівнюється з номером і назвою маршруту. Це забезпечує швидку реакцію інтерфейсу на введення користувача.

Окремо реалізовано пошук маршрутів за проміжною зупинкою, який вирішує задачу відображення контексту знайденої зупинки в межах маршруту. Для цього використовується серверний endpoint пошуку зупинок. Користувач вводить назву зупинки, сервер знаходить відповідні записи в таблиці Stops, після чого через таблицю RouteStops визначає маршрути, які проходять через знайдені зупинки. Для кожного маршруту сервер отримує повний список зупинок у прямому напрямку, визначає позицію знайденої зупинки та формує скорочене подання маршруту у форматі: «початкова зупинка – ... – знайдена зупинка – ... – кінцева зупинка», де знайдена зупинка виділяється жирним шрифтом. Таке подання дозволяє користувачу швидко зрозуміти, що маршрут проходить через потрібну йому проміжну зупинку.

При цьому початкова та кінцева зупинка маршруту не повинні оброблятися як проміжні. Якщо знайдена зупинка є першою або останньою у списку зупинок маршруту, маршрут може бути знайдений за назвою, але не повинен відображатися як маршрут із проміжною зупинкою. Це дозволяє уникнути ситуації, коли початкова зупинка підсвічується як проміжна.

Алгоритм формування розкладу по зупинках виконується на серверній стороні. Клієнт передає ідентифікатор маршруту, тип дня та напрямок руху. Сервер спочатку отримує всі зупинки маршруту з таблиці RouteStops відповідно до заданого напрямку та сортує їх за order_number. Далі з таблиці Trips вибираються всі рейси відповідного маршруту, типу дня та напрямку. Після цього для кожної зупинки та кожного рейсу виконується пошук відповідного запису в таблиці StopTimes. У результаті формується масив об'єктів, кожен з яких містить інформацію про зупинку та масив часів прибуття.

Отримана структура є зручною для відображення на сторінці деталей маршруту. Клієнтська частина проходить по масиву зупинок і для кожної з них виводить назву та набір часових позначок. Часи відображаються у вигляді невеликих кольорових чіпів. Колір чіпів залежить від типу транспорту, що дозволяє зберегти єдиний візуальний стиль на сторінці маршруту.

Розширений пошук маршруту між двома зупинками реалізований через endpoint /api/search/trip. Користувач обирає початкову та кінцеву зупинку. Сервер знаходить усі записи RouteStop для початкової зупинки та всі записи RouteStop для кінцевої зупинки. Далі маршрути порівнюються за RouteId і direction. Маршрут вважається придатним, якщо обидві зупинки належать одному маршруту в одному напрямку, а порядковий номер початкової зупинки менший за порядковий номер кінцевої. Таким чином система знаходить лише ті маршрути, які реально рухаються від вибраної початкової зупинки до вибраної кінцевої.

Після визначення придатних маршрутів сервер аналізує рейси для поточного типу дня. Для кожного рейсу визначається час прибуття транспорту на початкову зупинку. Далі відбираються найближчі майбутні рейси відносно поточного часу. Результати сортуються за кількістю хвилин до найближчого відправлення. Клієнт отримує список маршрутів, найближчі часи відправлення, орієнтовний час прибуття на кінцеву зупинку та кількість хвилин до найближчого рейсу.

Алгоритм відображення маршруту на карті базується на послідовності координат зупинок. На сторінці деталей маршруту клієнт отримує маршрут разом із routeStops, фільтрує зупинки за активним напрямком, сортує їх за order_number і формує масив координат у форматі [lat, lng]. Цей масив передається компоненту Polyline бібліотеки React-Leaflet, який будує лінію маршруту на карті. Зупинки відображаються окремими маркерами.

На сторінці карти зупинок реалізовано інший сценарій роботи з картою. Спочатку завантажується список усіх зупинок. Кожна зупинка відображається на карті як маркер. При кліку на зупинку виконується запит до API для отримання зупинки разом із маршрутами, які через неї проходять. У спливаючому вікні відображається список таких маршрутів з іконками та кольорами відповідного типу транспорту. Додатково для вибраних маршрутів можуть будуватися лінії на карті, що дозволяє наочно побачити, які маршрути проходять через обрану зупинку.

Алгоритм збереження обраних маршрутів реалізований на клієнтській стороні за допомогою localStorage [46]. При натисканні на іконку зірочки ідентифікатор маршруту додається до масиву обраних маршрутів або видаляється з нього, якщо маршрут уже був обраний. Оновлений масив зберігається в localStorage у форматі JSON. При повторному відкритті застосунку цей масив зчитується та використовується як початковий стан компонента.

2.4 Проєктування REST API та взаємодії між компонентами

REST API є центральним інтерфейсом взаємодії між клієнтською та серверною частинами системи. Коректне проєктування API визначає зручність розробки клієнта, зрозумілість інтерфейсу та можливість майбутнього розширення системи без порушення наявної функціональності. REST (Representational State Transfer) є архітектурним стилем для побудови розподілених веб-сервісів, що базується на використанні стандартних HTTP-методів та ресурсно-орієнтованого підходу до організації ендпоінтів.

Ключовими принципами REST є: однакова взаємодія з ресурсами через стандартні методи HTTP (GET, POST, PUT, DELETE); відсутність збереження стану між запитами (stateless); адресація кожного ресурсу через унікальний URI; передача даних у стандартизованому форматі – у даній системі використовується JSON [47].

Проєктоване API системи побудоване навколо основних сутностей предметної області транспортної системи. Кожна сутність представлена окремим ресурсом, що забезпечує логічну декомпозицію функціональності та спрощує розширення системи в майбутньому.

Перший ресурс відповідає за роботу з містами та забезпечує отримання переліку доступних населених пунктів, для яких наявні транспортні дані. Цей ресурс використовується на початковому етапі роботи клієнтського застосунку для вибору міста та ініціалізації подальших запитів.

Другий ресурс відповідає за маршрути громадського транспорту. Він забезпечує отримання списку маршрутів із можливістю фільтрації за містом, а також отримання детальної інформації про конкретний маршрут. Такий підхід дозволяє використовувати один ресурс як для загального перегляду маршрутів, так і для детального аналізу окремого маршруту на стороні клієнта.

Третій ресурс охоплює зупинки громадського транспорту. Він забезпечує отримання списку зупинок у межах обраного міста, а також доступ до детальної інформації про конкретну зупинку. Окремо передбачено функціональність пошуку, яка дозволяє знаходити зупинки та пов’язані з ними маршрути за текстовим запитом, що підвищує зручність навігації в системі.

Четвертий ресурс відповідає за розклади руху транспорту. Його основне призначення полягає у наданні інформації про часи прибуття транспорту на зупинки з урахуванням типу дня та напрямку руху. Це дозволяє клієнтській частині формувати актуальний розклад для конкретного маршруту та сценарію використання.

П’ятий ресурс реалізує функціональність пошуку маршруту між двома зупинками. Він дозволяє визначити можливі варіанти поїздки, включаючи

інформацію про найближчі відправлення та орієнтовний час прибуття. Даний ресурс є ключовим для реалізації функції побудови маршруту користувача.

Для забезпечення масштабованості системи кожен ресурс проєктується як незалежний компонент API. Такий підхід дозволяє розширювати функціональність без необхідності зміни існуючих інтерфейсів взаємодії.

Особливістю системи є підтримка багатомістності. Для цього в більшості запитів передбачається передача ідентифікатора міста, що дозволяє серверу повертати лише ті транспортні дані, які належать обраному населеному пункту.

В результаті, спроєктоване REST API забезпечує стандартизований, зрозумілий та масштабований механізм взаємодії між клієнтською частиною, сервером і базою даних та створює основу для реалізації функціональності системи в наступному розділі.

2.5 Проєктування інтерфейсу користувача

Інтерфейс користувача системи розкладу громадського транспорту спроєктовано з урахуванням основної потреби пасажира: швидко знайти потрібний маршрут, переглянути його розклад або визначити, яким транспортом можна доїхати від однієї зупинки до іншої. Проєктування інтерфейсу орієнтується на принципи мінімальної кількості дій для досягнення результату, передбачуваної поведінки елементів та зрозумілості без попереднього навчання.

На початковому етапі було визначено основні сценарії використання системи, на основі яких сформовано структуру інтерфейсу. До таких сценаріїв належать: перегляд списку маршрутів із можливістю фільтрації за типом транспорту, пошук маршруту за номером або назвою, пошук маршруту між двома зупинками, перегляд детальної інформації про маршрут із картою та розкладом руху, перегляд карти зупинок міста, збереження обраних маршрутів, а також вибір міста під час першого запуску застосунку. Кожному сценарію відповідає окрема сторінка або функціональний модуль системи.

Для забезпечення зручної навігації застосовано нижню панель навігації, яка містить три основні розділи: «Маршрути», «Карта» та «Обрані». Таке рішення надає користувачу швидкий доступ до ключових функцій застосунку

незалежно від поточної сторінки. У верхній частині інтерфейсу розміщено назву застосунку та інформацію про вибране місто. Натискання на назву міста відкриває інтерфейс його зміни. Перехід між сторінками виконується без перезавантаження, що забезпечує швидку та плавну взаємодію із системою.

Сторінку вибору міста реалізовано у вигляді простого списку з полем пошуку для швидкого знаходження необхідного населеного пункту. Для міст, транспортні дані яких ще не додано до системи, передбачено відображення позначки «Незабаром». Таке рішення демонструє можливість подальшого розширення системи та підтримки нових міст.

Головна сторінка є центральним елементом інтерфейсу та об'єднує основні засоби роботи з маршрутами. На ній відображається перелік усіх маршрутів обраного міста у вигляді інформаційних карток. Кожна картка маршруту містить основну інформацію про маршрут: номер, тип транспорту, назву, час роботи та кнопку додавання до списку обраних. Для швидкого візуального розпізнавання різних видів транспорту використовується система кольорового кодування.

Кольорове кодування типів транспорту визначено як єдине для всіх сторінок застосунку. Таке проєктне рішення забезпечує візуальну узгодженість: побачивши колір один раз, користувач одразу розпізнає тип транспорту на будь-якій сторінці – чи то картка на головній, чи маркер на карті, чи заголовок сторінки деталей.

Над списком розташовано поле пошуку для знаходження маршрутів за номером або назвою, а також набір фільтрів за типом транспорту: автобуси, тролейбуси, трамваї та маршрутні таксі. Додатково передбачено блок розширеного пошуку маршруту між двома зупинками, який може бути розгорнутий за потреби користувача.

Блок пошуку маршруту між двома зупинками дозволяє користувачу обрати початкову та кінцеву точки поїздки й отримати перелік доступних маршрутів. Результати пошуку відображаються у порядку наближеності часу відправлення найближчого рейсу із зазначенням часу очікування. Це дає можливість швидко обрати найбільш зручний варіант пересування.

Сторінка деталей маршруту призначена для перегляду повної інформації про обраний маршрут. У верхній частині сторінки відображаються номер маршруту, тип транспорту та кнопка додавання до обраних. Нижче розташовано інформаційні блоки з даними про час першого та останнього рейсу. Для перегляду маршруту в різних напрямках передбачено перемикач між прямим та зворотним напрямками руху.

Карта маршруту відображає траєкторію його проходження через усі зупинки. Лінія маршруту та маркери зупинок забарвлюються відповідно до типу транспорту, що забезпечує візуальну узгодженість інтерфейсу. Розклад руху представлено у вигляді послідовності зупинок із зазначенням часу прибуття транспортного засобу. Передбачено окремий перегляд розкладу для будніх і вихідних днів.

Сторінка карти зупинок містить інтерактивну карту з маркерами всіх зупинок обраного міста та поле пошуку. Після вибору зупинки користувачу відображається перелік маршрутів, що проходять через неї, а також їхнє розташування на карті. Це дозволяє оцінити транспортну доступність певної частини міста та спростити планування поїздок.

Сторінка обраних маршрутів забезпечує швидкий доступ до маршрутів, які користувач додав до списку за допомогою спеціальної кнопки. Для збереження обраних маршрутів і вибраного міста використовується локальне сховище браузера, що не потребує створення облікового запису або проходження процедури авторизації. Такий підхід спрощує використання системи та відповідає потребам більшості пасажирів. Водночас слід враховувати, що збережені дані прив'язані до конкретного браузера та пристрою й не синхронізуються між різними пристроями.

Загалом інтерфейс системи побудований за принципами простоти та швидкого доступу до інформації. Кольорове кодування типів транспорту, наявність пошуку, фільтрів та інтерактивної карти, а також механізм збереження обраних маршрутів і вибраного міста між сесіями забезпечують зручну та

ефективну взаємодію пасажира з розкладом громадського транспорту незалежно від використовуваного пристрою.

Висновки до розділу 2

У другому розділі спроєктовано повну архітектуру багатокомпонентної системи відображення розкладу громадського транспорту. Обрано та обґрунтовано триланкову клієнт-серверну архітектуру з чітким поділом на рівень представлення, рівень бізнес-логіки та рівень даних. Така архітектура забезпечує незалежність компонентів, масштабованість та зручність підтримки.

Спроєктовано реляційну базу даних з п'яти взаємопов'язаних таблиць: Routes, Stops, RouteStops, Trips та StopTimes. База даних нормалізована до 3НФ та забезпечує зберігання повної транспортної інформації – від загальних характеристик маршруту до точних часів прибуття на кожну зупинку для кожного рейсу. Визначено всі необхідні обмеження цілісності, включаючи зовнішні ключі з каскадним видаленням.

Розроблено алгоритми обробки даних для формування розкладу по зупинках, пошуку маршрутів і зупинок, фільтрації за видом транспорту та управління обраними маршрутами. Спроєктовано REST API та визначено структуру інтерфейсу користувача з двома основними екранами та навігаційною панеллю, що забезпечують отримання потрібної інформації не більше ніж за два кліки. Результати проєктування є повною технічною специфікацією для реалізації системи в наступному розділі.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ БАГАТОКОМПОНЕНТНОЇ СИСТЕМИ

3.1 Реалізація серверної частини системи

Реалізація серверної частини системи здійснювалась відповідно до архітектурних рішень, прийнятих у попередньому розділі. Серверна частина розроблена на базі платформи Node.js із використанням фреймворку Express.js, що забезпечує гнучку маршрутизацію запитів та обробку HTTP-взаємодії. Для взаємодії з реляційною базою даних MySQL 8 використовується ORM-бібліотека Sequelize, яка дозволяє працювати з даними на рівні об’єктної моделі.

Архітектура серверного застосунку побудована за модульним принципом і розділена на кілька основних шарів: маршрутизація запитів, бізнес-логіка та доступ до даних. Точкою входу серверного застосунку є файл index.js, у якому виконується ініціалізація Express-застосунку, підключення middleware, реєстрація маршрутизаторів та запуск HTTP-сервера після успішного підключення до бази даних. Для обробки JSON-запитів використовується middleware express.json(), а для підтримки міждоменних запитів – cors(). Конфігураційні параметри підключення до бази даних зберігаються у змінних середовища та завантажуються через бібліотеку dotenv, що підвищує безпеку зберігання конфігураційних даних.

Підключення до бази даних реалізовано у окремому модулі конфігурації, де створюється та ініціалізується екземпляр Sequelize. Під час запуску сервера виконується перевірка з’єднання з базою даних, що дозволяє виявити можливі помилки конфігурації на ранньому етапі.

Модель даних реалізована за допомогою Sequelize-моделей, які відображають основні сутності предметної області: міста, маршрути, зупинки, рейси та розклади руху. Моделі пов’язані між собою через асоціації, що дозволяє коректно відображати структуру транспортної системи на рівні бази даних.

Реалізація API виконана через окремі маршрутизатори Express, кожен з яких відповідає за певний ресурс системи. Після отримання запиту сервер

виконує перевірку вхідних параметрів, звертається до відповідних моделей даних та формує відповідь у форматі JSON.

Для підвищення продуктивності при обробці запитів, що містять множинні звернення до бази даних, використовується асинхронна обробка з Promise.all(), що дозволяє виконувати частину запитів паралельно та зменшувати загальний час відповіді сервера.

Окрему увагу приділено реалізації функціональності пошуку маршруту між двома зупинками. Для цього серверна частина аналізує зв’язки між маршрутами та зупинками і формує можливі варіанти поїздки на основі даних, отриманих із бази даних. Логіка обробки запиту інкапсульована на сервері, що дозволяє зменшити навантаження на клієнтську частину та забезпечити централізовану обробку алгоритму пошуку.

Усі API-запити реалізовані у відповідності до REST-принципів, що забезпечує уніфікований інтерфейс взаємодії між клієнтом і сервером. Дані передаються у форматі JSON, що спрощує їх обробку на стороні клієнтського застосунку.

3.2 Реалізація клієнтської частини системи

Клієнтську частину системи реалізовано як односторінковий React-застосунок, створений за допомогою утиліти Create React App. Маршрутизацію між сторінками забезпечує React Router DOM v6, що дозволяє декларативно описувати навігаційну структуру через компонент Routes з вкладеними елементами Route.

Кореневий компонент App реалізує логіку вибору міста. При завантаженні застосунку перевіряється наявність збереженого міста в локальному сховищі (localStorage). Якщо місто не обрано, відображається сторінка вибору міст (CitySelectPage). Після вибору міста об'єкт міста зберігається в localStorage та в стані компонента App, після чого відображається основний інтерфейс із навігаційними панелями та сторінками. При натисканні на кнопку зміни міста,

місто видаляється з localStorage та стану, і застосунок повертається до сторінки вибору міста.

Сторінку вибору міста (CitySelectPage) реалізовано з підтримкою пошуку міста за назвою. При завантаженні сторінка завантажує список міст з API. Міста з ознакою has_data: false відображаються з позначкою «Незабаром» та недоступні для вибору. Міста з наявними даними відображаються активними елементами, натискання на які зберігає місто та переходить до головної сторінки. Зовнішній вигляд сторінки вибору міста наведено на рисунку 3.1.

Рисунок 3.1 – Сторінка вибору міста

Головну сторінку застосунку (RoutesPage) реалізовано як багатофункціональний інтерфейс, що об’єднує декілька взаємопов’язаних функціональних блоків для забезпечення зручної та ефективної взаємодії користувача з системою. Основним елементом сторінки є блок пошуку маршрутів, який забезпечує швидку фільтрацію списку маршрутів у режимі реального часу. Пошук виконується без додаткових звернень до сервера, оскільки обробка даних здійснюється локально в межах уже завантаженого масиву маршрутів. Окрім пошуку, на сторінці реалізовано блок фільтрів, який надає можливість користувачеві відображати лише маршрути певного виду транспорту. Завдяки цьому користувач може швидко адаптувати список маршрутів до власних потреб, обираючи, наприклад, лише автобусні, трамвайні

або тролейбусні маршрути. Використання системи фільтрації сприяє зручнішій навігації та спрощує процес пошуку необхідного маршруту серед великої кількості доступних варіантів.

Список маршрутів представлено у вигляді окремих карток, кожна з яких містить основну інформацію про маршрут та елементи інтерактивної взаємодії. Для покращення візуального сприйняття використано кольорове кодування залежно від типу транспорту, що дозволяє користувачеві швидко розрізняти категорії маршрутів. Використані кольори відповідають офіційному кольоровому оформленню транспорту Вінницького депо. Зокрема, зелений колір використовується для автобусів, червоний – для тролейбусів, синій – для трамваїв, а жовтий – для маршрутних таксі. Додатково в інтерфейсі застосовано іконки з бібліотеки react-icons, які роблять сторінку більш наочною та інтуїтивно зрозумілою. Кожна картка також містить кнопку додавання маршруту до списку обраних, що забезпечує швидкий доступ до найбільш важливих або часто використовуваних маршрутів.

Завдяки поєднанню механізмів пошуку, фільтрації, візуального оформлення та інтерактивних елементів головна сторінка забезпечує зручний і сучасний інтерфейс для роботи з маршрутами громадського транспорту. Зовнішній вигляд головної сторінки наведено на рисунку 3.2.

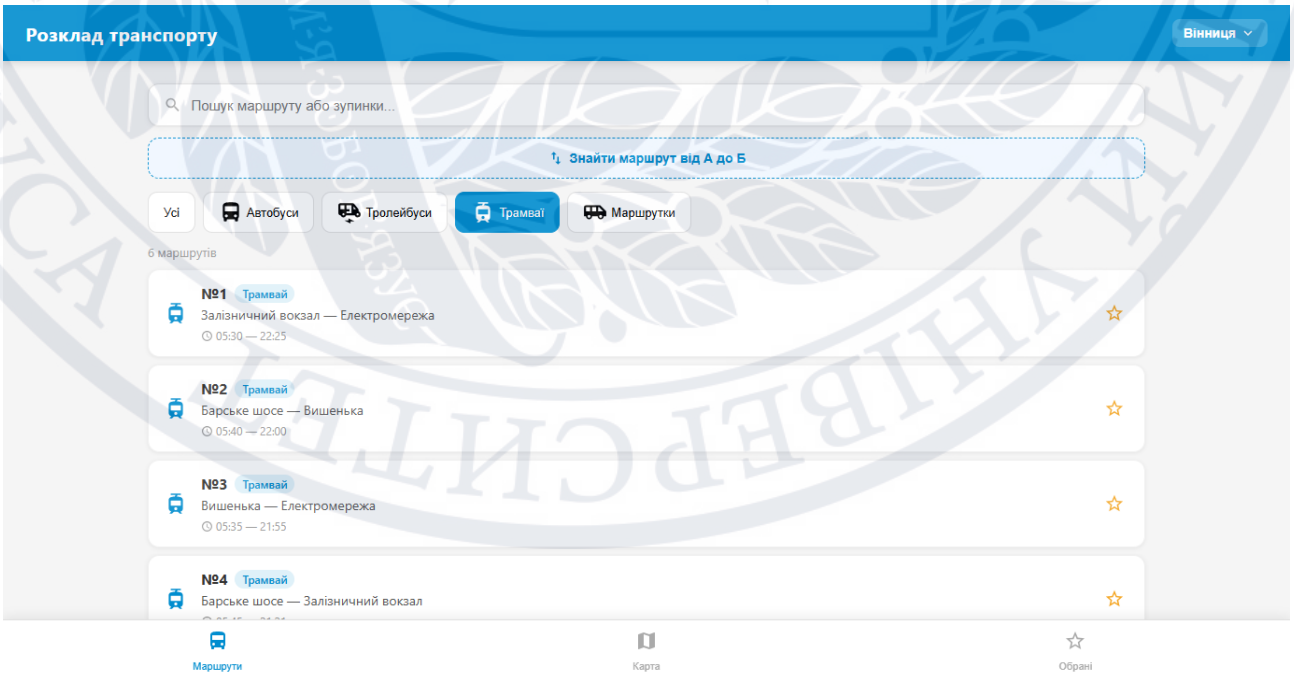

Рисунок 3.2 – Головна сторінка застосунку зі списком маршрутів

Блок розширеного пошуку «від А до Б» реалізований як компонент, що розгортається при натисканні відповідної кнопки. Для вибору зупинок розроблено компонент StopDropdown – кастомний випадаючий список з підтримкою введення тексту для фільтрації. При відкритті компонента завантажується повний список зупинок обраного міста. При введенні тексту список фільтрується за частковим збігом назви. Після вибору обох зупинок та натискання кнопки «Знайти маршрути» виконується запит до API, а результати відображаються відсортованими за часом очікування до найближчого рейсу. Зовнішній вигляд блоку розширеного пошуку наведено на рисунку 3.3.

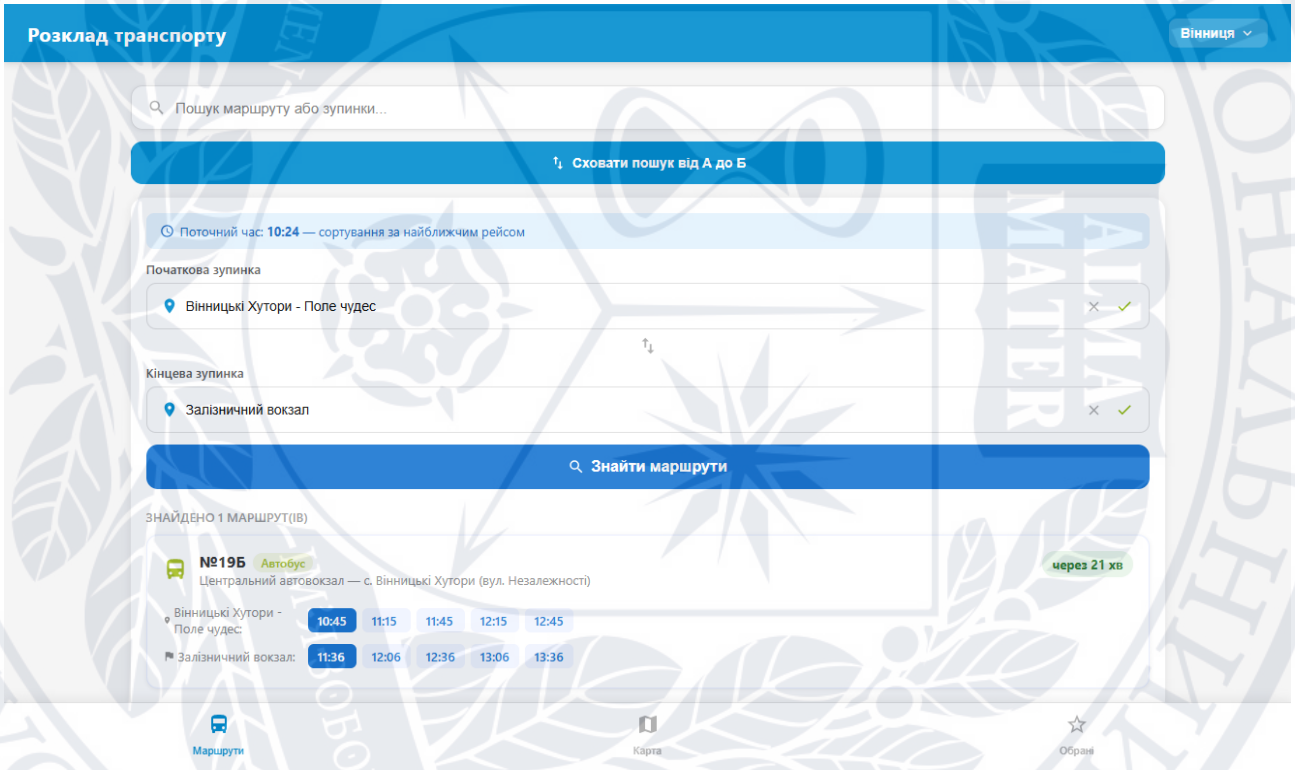

Рисунок 3.3 – Блок розширеного пошуку маршрутів від А до Б

Сторінку деталей маршруту реалізовано як комплексний інформаційний модуль, що складається з трьох основних функціональних блоків: картки з ключовими характеристиками маршруту, інтерактивної карти на основі бібліотеки Leaflet.js та блоку розкладу руху по зупинках. Така структура дозволяє користувачеві отримати повну інформацію про маршрут у межах однієї сторінки та забезпечує зручну взаємодію з даними.

Перший блок представлений у вигляді інформаційної картки маршруту, яка містить основні характеристики обраного маршруту. У картці відображається номер маршруту, тип транспорту, початкова та кінцева зупинки, а також додаткова інформація, необхідна для швидкої ідентифікації маршруту користувачем. Для покращення візуального сприйняття використовується кольорове оформлення відповідно до типу транспорту, що забезпечує стилістичну узгодженість із загальним дизайном системи.

Другий блок реалізовано у вигляді інтерактивної карти з використанням бібліотеки Leaflet.js. Карта дозволяє наочно відобразити географічне розташування маршруту та його зупинок. Лінія маршруту виводиться за допомогою компонента Polyline, який будує траєкторію руху транспорту між зупинками. Для позначення зупинок використано компонент CircleMarker, що забезпечує компактне та зрозуміле відображення точок зупинок на карті. Маркери мають кольорове оформлення відповідно до типу транспорту, завдяки чому користувач може легко розрізняти різні категорії маршрутів.

Інтерактивна карта підтримує масштабування та навігацію, що дозволяє детально переглядати окремі ділянки маршруту. Крім того, взаємодія з картою відбувається без перезавантаження сторінки, що позитивно впливає на швидкодію та комфорт використання вебзастосунку.

Третій блок сторінки містить розклад руху транспорту по зупинках. Реалізовано можливість перемикання між прямим і зворотним напрямком маршруту, що дозволяє переглядати час прибуття транспорту для обох напрямків руху. Також передбачено перемикання між будніми та вихідними днями, оскільки графік руху транспорту може відрізнятися залежно від дня тижня. Завдяки цьому користувач отримує актуальну інформацію про розклад руху відповідно до власних потреб. Зовнішній вигляд сторінки деталей маршруту наведено на рисунку 3.4.

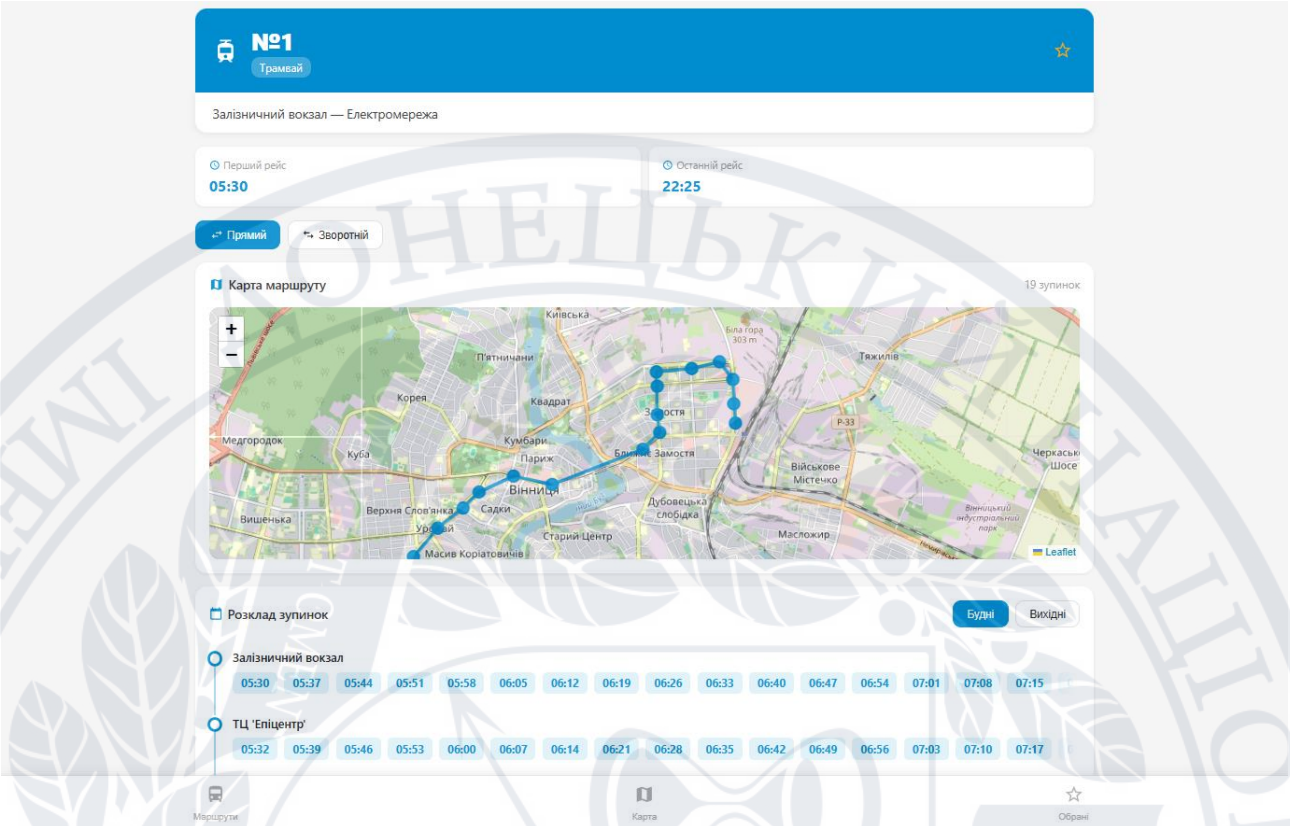

Рисунок 3.4 – Сторінка деталей маршруту з картою та розкладом

Сторінку карти зупинок реалізовано із застосуванням бібліотеки Leaflet, яка забезпечує інтерактивне відображення картографічних даних та підтримує роботу з різними шарами карти. Для коректного відображення елементів інтерфейсу використано кастомні шари Leaflet (Pane), що дозволяють керувати порядком розташування об’єктів на карті та визначати пріоритет їхнього відображення.

У процесі реалізації було налаштовано окремі шари для маршрутних ліній та маркерів зупинок. Лінії маршрутів розміщено на нижньому рівні відображення, тоді як маркери зупинок знаходяться поверх них. Таким чином користувач може безперешкодно взаємодіяти з маркерами зупинок незалежно від кількості маршрутів, що проходять через певну ділянку карти.

Кожна зупинка на карті представлена окремим маркером, який підтримує інтерактивну взаємодію. Після натискання на маркер виконується завантаження інформації про маршрути, що проходять через вибрану зупинку. Отримані дані обробляються та відображаються у вигляді ліній маршрутів безпосередньо на

карті. Це дозволяє користувачеві швидко переглядати транспортні зв’язки конкретної зупинки та аналізувати доступні маршрути громадського транспорту.

Для підвищення зручності використання сторінки реалізовано динамічне оновлення карти без її повного перезавантаження. Усі зміни відображаються в реальному часі, що забезпечує плавну роботу інтерфейсу та покращує користувацький досвід. Зовнішній вигляд сторінки карти зупинок наведено на рисунку 3.5.

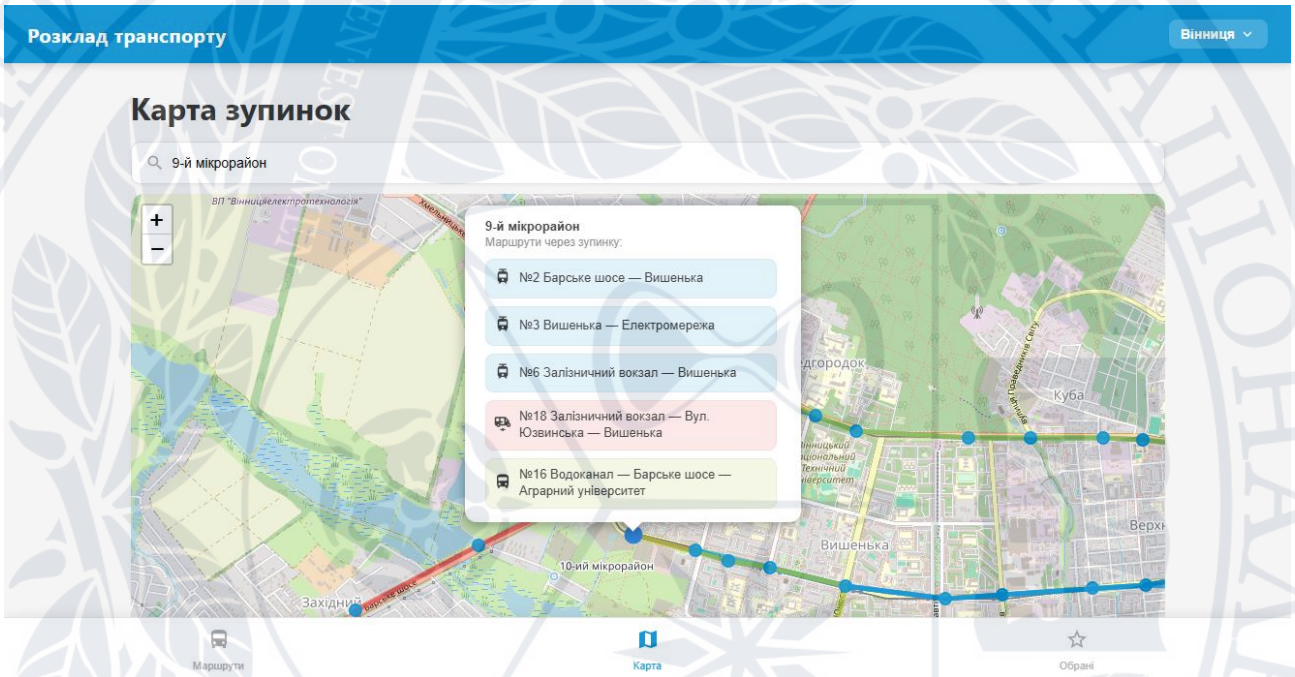

Рисунок 3.5 – Сторінка карти зупинок

Сторінку «Обране» реалізовано для швидкого доступу до маршрутів, які користувач найчастіше використовує. Додавання маршрутів до списку обраного здійснюється за допомогою спеціальної кнопки на картках маршрутів, після чого вони автоматично зберігаються у локальному сховищі браузера.

Список обраних маршрутів відображається у вигляді карток із основною інформацією про маршрут, кольоровим кодуванням типу транспорту та іконками. Особливістю реалізації є те, що список обраного є спільним для всіх міст, доступних у застосунку. Це дозволяє користувачеві зберігати важливі маршрути незалежно від обраного міста та швидко отримувати до них доступ у будь-який момент. Приклад сторінки наведено на рисунку 3.6.

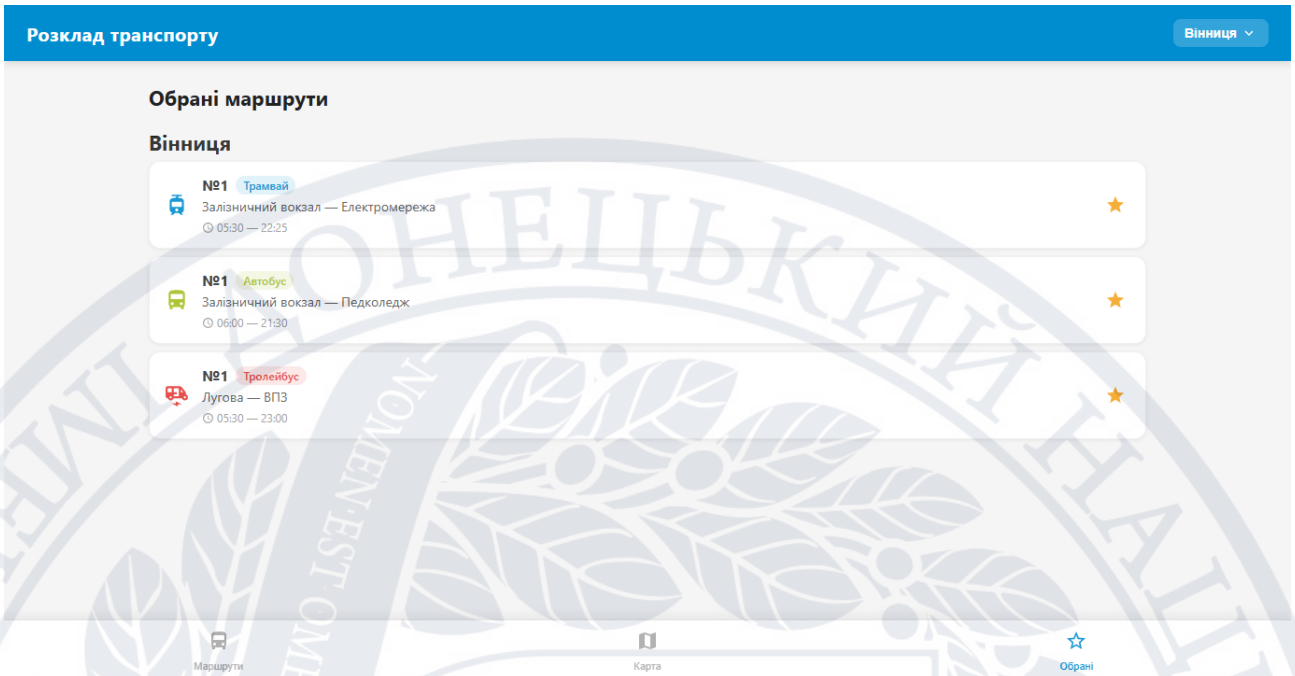

Рисунок 3.6 – Сторінка «Обрані»

3.3 Результати реалізації системи та напрями її вдосконалення

За результатами реалізації розроблена система відповідає визначеним функціональним та нефункціональним вимогам. Усі шість модулів застосунку – вибір міста, перегляд маршрутів з розширеним пошуком, деталі маршруту з картою та розкладом по зупинках, карта зупинок, текстовий пошук та обрані маршрути – повністю реалізовані та функціонують коректно.

Практична цінність системи полягає в кількох аспектах. По-перше, реалізовано зручний веб-інтерфейс для перегляду розкладу громадського транспорту, доступний з будь-якого пристрою через браузер без встановлення додаткового програмного забезпечення. По-друге, функція розширеного пошуку маршрутів між двома зупинками з відображенням найближчих рейсів та часу очікування є практично корисним інструментом для щоденного використання пасажирами. По-третє, інтерактивна карта зупинок з відображенням маршрутів дозволяє отримати просторове уявлення про транспортну мережу міста. По-четверте, архітектура системи підтримує багатомістність – при додаванні нових міст достатньо внести відповідні дані до бази без зміни програмного коду.

Порівняно з існуючими аналогами, розглянутими у першому розділі, розроблена система має ряд переваг. На відміну від EasyWay вона надає повний

розклад по зупинках у табличному форматі для обох напрямків руху. На відміну від «Vinnytsia Transport» охоплює всі види транспорту – трамваї, тролейбуси, автобуси та маршрутне таксі. Порівняно з OpenTripPlanner система є значно простішою у розгортанні та налаштуванні. При цьому, на відміну від більшості глобальних сервісів, вона повністю підтримує українські міста та надає україномовний інтерфейс.

Разом з тим у процесі реалізації виявлено технічні обмеження, що можуть бути усунені у подальшому розвитку. Поточна реалізація підтримує лише прямі маршрути без пересадок. Розклад є статичним і не враховує фактичний стан руху транспорту на дорогах. Наповнення бази даних виконується вручну через SQL-скрипти, що потребує технічних знань від адміністратора системи.

Перспективними напрямками розширення функціональності системи є інтеграція з GPS-трекерами транспортних засобів для відображення їх реального місцезнаходження та точного прогнозування часу прибуття. Розробка адміністративного веб-інтерфейсу дозволить управляти даними про маршрути і розклади без прямого доступу до бази даних. Реалізація алгоритму пошуку маршрутів з пересадками суттєво підвищить практичну цінність функції розширеного пошуку. Підтримка імпорту даних у форматі GTFS забезпечить можливість легкої інтеграції даних від транспортних операторів, що вже використовують цей стандарт. Нарешті, розширення бази даних транспортними мережами інших українських міст перетворить систему на повноцінний загальнонаціональний інструмент інформування пасажирів.

Висновки до розділу 3

У третьому розділі описано реалізацію багатокомпонентної системи відображення розкладу громадського транспорту. Реалізовано серверну частину на Node.js з Express.js та Sequelize з маршрутизаторами для п'яти ресурсів REST API. Виконано ініціалізацію бази даних MySQL реальними даними транспортних мереж України.

Реалізовано клієнтський React-застосунок із шістьма сторінками: вибір міста, перелік маршрутів з розширеним пошуком від А до Б, деталі маршруту з

картою та розкладом по зупинках, карта зупинок, їх пошук та обрані маршрути. Застосовано кольорове кодування типів транспорту через CSS-змінні, іконки з бібліотеки react-icons та інтерактивну карту на Leaflet.js з OpenStreetMap.

Проведено аналіз результатів реалізації, що підтвердив відповідність системи сформованим вимогам та її переваги порівняно з існуючими аналогами. Визначено технічні обмеження та перспективні напрямки подальшого розширення функціональності системи.

ВИСНОВКИ

У кваліфікаційні (бакалаврській) роботі вирішено науково-практичне завдання розробки багатокомпонентної системи для відображення розкладу руху громадського транспорту. На підставі проведеного дослідження зроблено такі висновки.

Проведено аналіз предметної області систем відображення розкладу громадського транспорту. Встановлено, що якісне інформаційне забезпечення пасажирів є визначальним чинником ефективності використання громадського транспорту. Розглянуто міжнародний стандарт представлення транспортних даних GTFS, логіка якого стала основою для проєктування структури бази даних систем.

Здійснено огляд та порівняльний аналіз існуючих систем аналогів та було встановлено, що жодне з рішень не поєднує одночасно зручний перегляд повного статичного розкладу по зупинках, карту зупинок та їх пошук, україномовний інтерфейс та відкритість для локального розгортання. Це підтвердило доцільність розробки власного рішення.

У процесі роботи було визначено основні вимоги до системи, що охоплюють функціональні можливості перегляду маршрутів, розкладів, пошуку та взаємодії з картою, а також нефункціональні характеристики, пов’язані з продуктивністю, надійністю та зручністю використання. На основі цих вимог було обрано сучасний технологічний стек, придатний для реалізації вебзастосунку з клієнт-серверною архітектурою.

Спроєктовано загальну триланкову клієнт-серверну архітектуру системи з підтримкою багатомістності. Архітектурне проєктування дозволило створити цілісну структуру системи з чітко визначеною реляційною моделлю даних, що забезпечує збереження маршрутів, зупинок і розкладів та підтримує цілісність інформації.

Спроєктовано серверну частину із REST API та реалізовано логіку обробки транспортних даних, зокрема механізми пошуку маршрутів і визначення

найближчих рейсів. Це дозволило забезпечити коректну роботу з актуальними даними та зручну взаємодію клієнтської частини із сервером.

Клієнтська частина системи реалізована як вебзастосунок із інтерактивним інтерфейсом, що забезпечує перегляд маршрутів, розкладів і карти зупинок, а також підтримує пошук та збереження обраних маршрутів. Використані інструменти дозволили створити зручний і зрозумілий користувацький інтерфейс.

Практична цінність отриманих результатів полягає у створенні функціонального веб-застосунку для перегляду розкладу вінницького громадського транспорту, доступного з будь-якого пристрою через браузер без встановлення додаткового програмного забезпечення. Система реалізує унікальну для вітчизняного ринку функцію пошуку маршрутів між двома зупинками з відображенням найближчих рейсів в реальному часі. Архітектура системи підтримує роботу з кількома містами та є готовою до масштабування шляхом додавання нових міст і розширення функціональності.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Ganesh K., Thrivikraman M., Kuri J., Dagale H., Sudhakar G., Sanyal S. Implementation of a Real Time Passenger Information System. arXiv. 2012. URL: https://arxiv.org/abs/1206.0447 (дата звернення: 10.03.2026).
2. Розклад руху. Термінологія законодавства України. Законодавство України. URL: https://zakon.rada.gov.ua/laws/term/26225 (дата звернення: 10.03.2026).
3. Webb L., Higgs G., Langford M., Berry R. Evaluating Real-Time and Scheduled Public Transport Data: Challenges and Opportunities. ISPRS International Journal of Geo-Information. 2025. URL: https://www.mdpi.com/2220-9964/14/7/243 (дата звернення: 10.03.2026).
4. General Transit Feed Specification Reference. Google LLC, 2022. URL: https://gtfs.org/documentation/schedule/reference/ (дата звернення: 12.03.2026).
5. Liu D., Guo J., Gu Y. et al. GTFS2STN: Analyzing GTFS Transit Data by Generating Spatiotemporal Transit Network. arXiv. 2024. URL: https://arxiv.org/abs/2405.02760 (дата звернення: 14.03.2026).
6. NeTEx – Network Timetable Exchange: CEN/TS 16614. European Committee for Standardization, 2017. URL: https://transmodel-cen.eu/index.php/netex/ (дата звернення: 14.03.2026).
7. Scrocca M., Comerio M., Carenini A., Celino I. Turning Transport Data to Comply with EU Standards while Enabling a Multimodal Transport Knowledge Graph. arXiv 2020. URL: https://arxiv.org/abs/2011.06423 (дата звернення: 15.03.2026).
8. SIRI – Service Interface for Real-time Information. Transmodel. URL: https://transmodel-cen.eu/index.php/siri/ (дата звернення: 15.03.2026).
9. Chen Z., Botta F. A scalable framework for correcting public transport timetables using real-time data for accessibility analysis. arXiv. 2026. URL: https://arxiv.org/abs/2603.11477 (дата звернення: 16.03.2026).

10. Zheng Y., Capra L., Wolfson O., Yang H. Urban Computing: Concepts, Methodologies, and Applications. ACM Transactions on Intelligent Systems and Technology. 2014. URL: https://dl.acm.org/doi/10.1145/2629592 (дата звернення: 16.03.2026).
11. Macedo E., Teixeira J., Sampaio C., Silva N., Coelho M. C., Glinos M., Bandeira J. M. Real-time information systems for public transport: user perspective. Transportation Research Procedia. 2021. Vol. 52. P. 732–739. URL: https://doi.org/10.1016/j.trpro.2021.01.088 (дата звернення: 07.03.2026).
12. Watkins K. E., Ferris B., Borning A., Rutherford G. S., Layton D. Where Is My Bus? Impact of mobile real-time information on the perceived and actual wait time of transit riders. Transportation Research Part A: Policy and Practice. 2011. Vol. 45, No. 8. P. 839–848. URL: https://doi.org/10.1016/j.tra.2011.06.010 (дата звернення: 07.03.2026).
13. Open Transit Software Foundation. OneBusAway: Transit Rider Experience Platform. URL: https://onebusaway.org/ (дата звернення: 07.03.2026).
14. Paulsen M., Rasmussen T. K., Nielsen O. A. Impacts of real-time information levels in public transport: A large-scale case study using an adaptive passenger path choice model. Transportation Research Part A: Policy and Practice. 2021. Vol. 148. P. 155–182. URL: https://doi.org/10.1016/j.tra.2021.02.015 (дата звернення: 07.03.2026).
15. Moovit Inc. Moovit Public Transit App. URL: https://moovitapp.com (дата звернення: 19.03.2026).
16. CityMapper Ltd. CityMapper App. URL: https://citymapper.com (дата звернення: 19.03.2026).
17. OpenTripPlanner Project. OpenTripPlanner (Open Source Multimodal Trip Planning System). URL: https://www.opentripplanner.org/ (дата звернення: 19.03.2026).
18. EasyWay. EasyWay Public Transport App (Ukraine & Europe Transit Information System). URL: https://www.eway.in.ua/ (дата звернення: 20.03.2026).

19. Вінницька міська рада. У Вінниці запрацював додаток «Vinnytsia Transport». Офіційний сайт Вінницької міської ради. URL: https://www.vmr.gov.ua/u-vinnytsi-zapratsiuvav-dodatok-vinnytsia-transport (дата звернення: 20.03.2026)
20. CityBus. Найрозумніший додаток для моніторингу громадського транспорту міст України. URL: http://citybus.in.ua/ (дата звернення: 20.03.2026)
21. Департамент інформаційно-комунікаційних технологій КМДА. Київ Цифровий: міський застосунок. Офіційний портал Київської міської державної адміністрації. URL: https://kyivcity.gov.ua/miskiy_zastosunok_kyiv_tsifroviy/ (дата звернення: 20.03.2026)
22. CityMonitor Editorial Team. Best Transit Apps for New York City: A Complete Review. CityMonitor. 2026. URL: https://www.nycmoov.com/guide/best-transit-apps-for-new-york-city-a-complete-review (дата звернення: 22.03.2026).
23. Santos G., Nikolaev N. Mobility as a Service and Public Transport: A Rapid Literature Review and the Case of Moovit. Sustainability. 2021. URL: https://www.mdpi.com/2071-1050/13/7/3666 (дата звернення: 22.03.2026).
24. Fielding R. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California. Irvine. 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (дата звернення: 25.03.2026).
25. Nielsen J. Usability Engineering. Morgan Kaufmann, 1994. URL: https://www.nngroup.com/books/usability-engineering/ (дата звернення: 26.03.2026).
26. MDN Web Docs. Modern web development architecture overview. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development (дата звернення: 29.03.2026).

27. React Documentation. Building User Interfaces with React 18 . URL: https://react.dev/learn (дата звернення: 30.03.2026).
28. Vue.js Official Guide. Comparison with Other Frameworks. URL: https://vuejs.org/guide/introduction.html (дата звернення: 30.03.2026).
29. Angular Documentation. Architecture Overview. URL: https://angular.dev/reference/configs/file-structure (дата звернення: 30.03.2026).
30. Comparison of Angular, React, and Vue Technologies in the Process of Creating Web Applications. Journal of Education, Technology and Computer Science. 2023. URL: https://journals.ur.edu.pl/jetacomps/article/view/9124 (дата звернення: 30.03.2026)
31. React Hooks API Reference. URL: https://react.dev/reference/react (дата звернення: 04.05.2026).
32. Leaflet. Leaflet – an open-source JavaScript library for interactive maps. URL: https://leafletjs.com/ (дата звернення: 04.05.2026).
33. React Leaflet Documentation. URL: https://react-leaflet.js.org/ (дата звернення: 04.05.2026).
34. OpenStreetMap. About OpenStreetMap. URL: https://www.openstreetmap.org/about (дата звернення: 05.05.2026).
35. Node.js Official Documentation. URL: https://nodejs.org/en/docs (дата звернення: 05.05.2026).
36. Node.js. Event Loop Timers and process.nextTick (Official Guide). URL: https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/ (дата звернення: 05.05.2026).
37. Express.js Official Documentation. URL: https://expressjs.com/ (дата звернення: 05.05.2026).
38. MySQL 8.0 Reference Manual. Oracle. URL: https://dev.mysql.com/doc/refman/8.0/en/ (дата звернення: 06.05.2026).
39. Sequelize Documentation. URL: https://sequelize.org/ (дата звернення: 06.05.2026).

40. Sidorares. mysql2 GitHub Repository. URL: https://github.com/sidorares/node-mysql2 (дата звернення: 06.05.2026).
41. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Addison-Wesley, 2021. URL: https://www.scribd.com/document/771670822/Software-Architecture-in-Practice-4th-Edition (дата звернення: 08.05.2026).
42. Microsoft. N-tier architecture style. Microsoft Learn. 2024. URL: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier (дата звернення: 08.05.2026).
43. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002. URL: https://martinfowler.com/books/eaa.html (дата звернення: 10.05.2026).
44. Silberschatz A., Korth H. F., Sudarshan S. Database System Concepts. 7th ed. McGraw-Hill Education, 2020. URL: https://www.db-book.com/ (дата звернення: 12.05.2026).
45. Sequelize. Associations. Sequelize Documentation. URL: https://sequelize.org/docs/v6/core-concepts/assocs/ (дата звернення: 06.05.2026).
46. MDN Web Docs. Web APIs – localStorage. Mozilla Corporation, 2024. URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage (дата звернення: 16.05.2026).
47. Rodríguez J. A., García-Lozano M., Solé-Casals J. Do RESTful API design rules have an impact on the understandability of Web APIs? Empirical Software Engineering. 2023. Vol. 28. URL: https://link.springer.com/article/10.1007/s10664-023-10367-y (дата звернення: 17.05.2026).