Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально–науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

До захисту допущено
Кафедрою комп'ютерних систем та робототехніки
протокол №__ від ____ червня 2026 р.

завідувач кафедри _____ Максим ХРУСЛОВ
(підпис)
«___»_____ 2026 р.

Кваліфікаційна робота

здобувача першого (бакалаврського) рівня вищої освіти

«СИСТЕМА МОНІТОРИНГУ ТА АНАЛІЗУ ДАНИХ ТРАНСПОРТНОЇ ЛОГІСТИКИ НА БАЗІ ВЕБ–ТЕХНОЛОГІЙ»

Спеціальність ***123 – Комп'ютерна інженерія***

Освітня програма ***Комп'ютерна інженерія***

Виконавець _____ **Даніїл ДВОРЕЦЬКИЙ**
(підпис)

Науковий керівник _____ **Ольга ЧУБ**
(підпис)

Харків – 2026

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи містить вступ, 3 розділи основної частини, висновки, список використаних джерел та 3 додатки. Загальний обсяг роботи становить 63 сторінки, з них 39 сторінок основного тексту, 12 рисунків, 2 таблиці, 25 найменувань використаних джерел.

Мета кваліфікаційної роботи полягає у підвищенні ефективності керування залізничними перевезеннями за допомогою розробки веб–застосунку з використанням методів оптимізації маршрутів, зокрема алгоритмів пошуку найкоротшого шляху, таких як Алгоритм Дейкстри.

Об'єктом дослідження є процес керування залізничними рейсами та формування оптимальних маршрутів перевезення пасажирів.

Предмет дослідження – методи та алгоритми оптимізації маршрутів, обробка транспортних даних, а також програмні засоби розробки веб–застосунків для керування залізничними перевезеннями.

У роботі розглянуто проблему ефективного планування маршрутів та розкладу руху поїздів в умовах зростання пасажирських перевезень. Розроблений веб–застосунок забезпечує автоматизацію формування маршрутів, управління розкладом та взаємодії з користувачами, що дозволяє скоротити час пошуку оптимальних маршрутів і підвищити якість обслуговування.

Область застосування – транспортні та логістичні системи, інформаційні системи управління залізничними перевезеннями.

Ключові слова: *інформаційні технології, веб–застосунок, залізничні рейси, оптимізація маршрутів, алгоритм Дейкстри, інформаційні системи, логістика, автоматизація.*

ABSTRACT

The explanatory note to the qualification work contains an introduction, 3 chapters of the main part, conclusions, a list of references, and 3 appendices. The total volume of the work is 63 pages, including 39 pages of the main text, 12 figures, 2 tables, 25 references.

The purpose of the qualification work is to improve the efficiency of railway transportation management through the development of a web application using route optimization methods, particularly shortest path algorithms such as Dijkstra's Algorithm.

The object of research is the process of railway route management and the formation of optimal passenger transportation routes.

The subject of research includes methods and algorithms for route optimization, transport data processing, as well as software tools for developing web applications for railway transportation management.

The work considers the problem of efficient route and train schedule planning under conditions of increasing passenger transportation demand. The developed web application provides automation of route generation, schedule management, and user interaction, which reduces the time required to search for optimal routes and improves service quality.

The area of application includes transport and logistics systems, as well as information systems for railway transportation management.

Keywords: *information technologies, web application, railway routes, route optimization, Dijkstra's algorithm, information systems, logistics, automation.*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ 6

ВСТУП 8

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ СИСТЕМ КЕРУВАННЯ ЗАЛІЗНИЧНИМИ РЕЙСАМИ 10

1.1 Поняття та принципи організації залізничних перевезень 10

1.2 Огляд існуючих систем управління залізничними маршрутами 11

1.3 Огляд та порівняння алгоритмів пошуку маршрутів 13

1.4 Вибір технологічного стеку та його обґрунтування 15

Висновки до розділу 1 18

РОЗДІЛ 2 ПРОЄКТУВАННЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ КЕРУВАННЯ ЗАЛІЗНИЧНИМИ РЕЙСАМИ 19

2.1 Вимоги до системи та постановка задачі проектування 19

2.2 Проектування архітектури веб-застосунку 20

2.3 Проектування бази даних та моделі даних 22

2.4 Опис програмних сутностей та шарів застосунку 24

2.5 Проектування алгоритму пошуку оптимального маршруту 27

2.6 Проектування REST API та міжсервісної взаємодії 30

Висновки до розділу 2 30

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ВЕБ-ЗАСТОСУНКУ 31

3.1 Розгортання середовища розробки та налаштування проекту 31

3.2 Реалізація серверної частини (бекенду) 32

3.3 Реалізація клієнтської частини (фронтенду) 33

3.4 Тестування та перевірка функціональності..... 39

3.4.1 Тестування CRUD-операцій адмін-панелі..... 39

3.4.2 Тестування алгоритму пошуку маршрутів..... 40

Висновки до розділу 3..... 41

ВИСНОВКИ..... 43

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 46

ДОДАТКИ..... 49

Додаток А..... 49

Додаток Б..... 51

Додаток В..... 58

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API (Application Programming Interface) – програмний інтерфейс застосунку; набір правил і протоколів для взаємодії між програмними компонентами.

БД – база даних; організована сукупність структурованих даних, що зберігається в комп'ютерній системі.

CORS (Cross–Origin Resource Sharing) – механізм, що дозволяє або забороняє веб–запити між різними доменами.

DTO (Data Transfer Object) – об'єкт передачі даних; шаблон проектування для обміну даними між шарами застосунку.

HTTP (HyperText Transfer Protocol) – протокол передачі гіпертексту; основний протокол передачі даних у мережі Інтернет.

HTML (HyperText Markup Language) – мова розмітки гіпертексту для створення веб–сторінок.

CSS (Cascading Style Sheets) – каскадні таблиці стилів для визначення зовнішнього вигляду документів HTML.

IDE (Integrated Development Environment) – інтегроване середовище розробки програмного забезпечення.

JPA (Jakarta Persistence API) – специфікація Java для об'єктно–реляційного відображення даних.

JSON (JavaScript Object Notation) – текстовий формат обміну даними, зручний для читання людиною та машинного аналізу.

JS – JavaScript; інтерпретована мова програмування для веб–розробки.

Maven – інструмент автоматизації збірки Java–проектів і управління залежностями.

MVC (Model–View–Controller) – архітектурний патерн розділення застосунку на модель, подання та контролер.

ORM (Object–Relational Mapping) – технологія об'єктно–реляційного відображення між класами Java і таблицями бази даних.

REST (Representational State Transfer) – архітектурний стиль для розподілених гіпермедіасистем.

SQL (Structured Query Language) – мова структурованих запитів для роботи з реляційними базами даних.

SPA (Single Page Application) – односторінковий застосунок, що динамічно оновлює вміст без перезавантаження сторінки.

UI (User Interface) – інтерфейс користувача; сукупність засобів взаємодії між користувачем та системою.

UML (Unified Modeling Language) – уніфікована мова моделювання програмних систем.

URL (Uniform Resource Locator) – уніфікований покажчик ресурсу в мережі Інтернет.

ПЗ – програмне забезпечення.

СУБД – система управління базами даних.

ВСТУП

Актуальність роботи. У сучасному світі транспортна інфраструктура відіграє ключову роль у забезпеченні економічного розвитку та мобільності населення. Зростання обсягів пасажирських і вантажних перевезень залізничним транспортом потребує ефективних засобів управління рейсами, розкладом та ресурсами. Традиційні підходи до організації залізничних перевезень часто є недостатньо гнучкими, залежать від людського фактора та не забезпечують належного рівня оптимізації маршрутів і використання ресурсів. Особливо це стає критичним в умовах великого навантаження на транспортні мережі, коли необхідно оперативно реагувати на зміни, уникати затримок і забезпечувати безперебійність перевезень. Впровадження сучасних веб–застосунків для керування залізничними рейсами дозволяє автоматизувати основні процеси, підвищити точність планування та ефективність прийняття рішень. Застосування методів оптимізації маршрутів, зокрема алгоритмів пошуку найкоротшого шляху, таких як алгоритм Дейкстри, дає змогу знаходити оптимальні маршрути перевезень, зменшувати час у дорозі та підвищувати ефективність використання інфраструктури. Це робить розробку подібних систем актуальним і перспективним напрямом, особливо з урахуванням розвитку інформаційних технологій та веб–платформ.

Метою дослідження є підвищення ефективності керування залізничними перевезеннями за допомогою розробки веб–застосунку з використанням методів оптимізації маршрутів, зокрема алгоритмів пошуку найкоротшого шляху, таких як Алгоритм Дейкстри.

Об’єктом дослідження є процес керування залізничними рейсами та формування оптимальних маршрутів перевезення пасажирів.

Методи дослідження: аналіз наукових джерел та існуючих інформаційних систем, порівняльний аналіз, проєктування інформаційних

систем, моделювання бізнес–процесів, програмна реалізація та тестування веб–застосунку.

Предметом дослідження є методи та алгоритми оптимізації маршрутів, обробка транспортних даних, а також програмні засоби розробки веб–застосунків для керування залізничними перевезеннями.

Для досягнення сформульованої мети були виділені такі **завдання дослідження:**

- виконати аналіз предметної області та існуючих інформаційних систем керування залізничними рейсами, їх функціональних можливостей, переваг і недоліків;
- спроєктувати архітектуру веб–застосунку, включаючи модель даних, ролі користувачів та бізнес–логіку;
- розробити веб–застосунок для керування залізничними рейсами з реалізацією основних функцій (управління рейсами, розкладом, станами та користувачами);
- провести тестування та оцінку ефективності роботи застосунку, перевірити коректність функціонування та зручність використання;
- проаналізувати можливості подальшого розвитку та масштабування системи, зокрема інтеграцію з іншими сервісами та впровадження аналітичних інструментів.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ СИСТЕМ КЕРУВАННЯ ЗАЛІЗНИЧНИМИ РЕЙСАМИ

1.1 Поняття та принципи організації залізничних перевезень

Залізничний транспорт є одним із найважливіших видів громадського транспорту, що забезпечує перевезення пасажирів і вантажів на значні відстані. В Україні залізниця відіграє ключову роль у забезпеченні транспортної доступності між населеними пунктами: мережа залізничних доріг охоплює більшість регіонів країни і щороку обслуговує мільйони пасажирів [1]. Зростання пасажиропотоку та ускладнення логістичних схем висувають нові вимоги до якості управління залізничними маршрутами і автоматизації процесів бронювання квитків.

Сучасний залізничний транспорт є складною соціотехнічною системою, що включає інфраструктуру (колії, станції, депо), рухомий склад (локомотиви, вагони), системи управління рухом поїздів, а також засоби обслуговування пасажирів [2]. Ефективне функціонування всіх цих елементів вимагає узгодженого обміну даними та автоматизованої підтримки прийняття рішень. Перехід до цифрових технологій управління залізничними перевезеннями є глобальним трендом, що відображено у стратегічних документах Міжнародного союзу залізниць (UIC) [3].

Предметна область задачі, що вирішується в рамках даної роботи, охоплює такі ключові поняття та сутності:

Маршрут (Route) – визначений шлях руху поїзда між двома станціями з фіксованим переліком проміжних зупинок, відстанню та тривалістю поїздки. Кожен маршрут характеризується впорядкованим набором станцій (RouteStation) із зазначенням часу стоянки на кожній з них [4].

Рейс (Trip) – конкретне відправлення поїзда за заданим маршрутом у певну дату та час. Рейс пов'язується з конкретним поїздом (Train) і включає

інформацію про дату та час відправлення і прибуття, кількість доступних місць та базову ціну квитка.

Поїзд (Train) – рухомий склад, що характеризується унікальним номером, найменуванням, типом (приміський, швидкісний, міжнародний) і загальною кількістю місць. Кожен поїзд має перелік місць (Seat) із прапорцем доступності для бронювання.

Станція (Station) – вузловий пункт залізничної інфраструктури з унікальним кодом, назвою, адресою та прив'язкою до міста (City). Станції є вершинами транспортного графа при побудові маршрутів.

Бронювання (Booking) – операція резервування конкретного місця у рейсі для пасажира з фіксацією дати бронювання та ціни. Бронювання пов'язує пасажира (Passenger), рейс (Trip) та місце (Seat).

Розклад рейсу (TripSchedule) – детальний план проходження рейсом кожної зі станцій маршруту із зазначенням часу прибуття та відправлення з кожної станції.

Зазначені сутності утворюють зв'язну предметну модель залізничних перевезень, що відображає реальні бізнес–процеси управління пасажирським транспортом [5]. Взаємодія між ними реалізована через зв'язки типу «один до багатьох» та «багато до одного», що відповідає нормалізованій реляційній моделі даних.

1.2 Огляд існуючих систем управління залізничними маршрутами

Аналіз наявних систем управління залізничними перевезеннями дозволяє виявити типові підходи до організації даних, функціональні можливості та технологічні рішення, що застосовуються у галузі.

Система «Укрзалізниця» (UZ Online) є офіційним сервісом бронювання квитків Державного підприємства «Укрзалізниця». Вона надає

функціонал пошуку поїздів за маршрутом і датою, вибору місць у вагонах різних класів, онлайн–оплати та управління бронюваннями. Система побудована на мікросервісній архітектурі з використанням Java на бекенді та React на фронтенді [6]. До обмежень системи можна віднести закритість API, відсутність відкритих засобів оптимізації маршрутів з пересадками, а також складність адміністрування через множинні спадкові модулі.

Система RailEurope – міжнародний агрегатор залізничних квитків, що об'єднує понад 40 залізничних операторів Європи. Платформа надає єдиний інтерфейс для пошуку та бронювання квитків, підтримує пошук маршрутів із пересадками і реалізована як хмарний SaaS–сервіс [7]. Недоліком є висока вартість інтеграції та неможливість кастомізації під специфічні вимоги окремих операторів.

Система DB Navigator (Deutsche Bahn) – мобільний застосунок і веб–портал найбільшого залізничного оператора Німеччини. Відрізняється розвиненим функціоналом пошуку оптимальних маршрутів, включно з мультимодальними перевезеннями (поїзд + метро + автобус), відстеженням поїздів у реальному часі та інтелектуальними підказками щодо пересадок [8]. Система є закритою корпоративною розробкою.

Порівняльний аналіз наявних систем виявив такі спільні риси: наявність RESTful API для взаємодії між клієнтом і сервером; використання реляційних баз даних для структурованих даних про рейси, маршрути та бронювання; реалізація алгоритмів пошуку оптимальних шляхів. Разом із тим для навчальних і демонстраційних цілей відсутні відкриті повнофункціональні реалізації, що охоплювали б увесь стек від бази даних до інтерфейсу користувача [9].

1.3 Огляд та порівняння алгоритмів пошуку маршрутів

Задача пошуку оптимального маршруту між двома станціями є класичною задачею теорії графів. Мережа залізничних маршрутів природно

моделюється як граф, де вершинами є станції, а ребрами – з'єднання між суміжними станціями в межах маршрутів. Вага ребра може відповідати відстані (у кілометрах), тривалості (у хвилинах) або вартості проїзду [10].

Алгоритм Дейкстри є найбільш широко застосовуваним алгоритмом пошуку найкоротшого шляху у графах із невід'ємними вагами ребер. Алгоритм послідовно розширює множину «вирішених» вершин, обираючи на кожному кроці вершину з мінімальною відомою відстанню від початкової. Часова складність алгоритму з пріоритетною чергою становить $O((V + E) \log V)$, де V – кількість вершин, E – кількість ребер [11]. Саме цей алгоритм реалізовано у класі RouteFinderService проекту railway–web–app для знаходження найкоротшого шляху за тривалістю між двома станціями.

Алгоритм A* (A–зірка) є розширенням алгоритму Дейкстри з використанням евристичної функції $h(n)$, що оцінює вартість шляху від поточної вершини до цільової. Застосування евристики дозволяє суттєво пришвидшити пошук у великих графах, спрямовуючи його в бік цільової вершини [12]. Для залізничних мереж як евристика може використовуватися географічна відстань між станціями.

Алгоритм Белмана–Форда, на відміну від алгоритму Дейкстри, може опрацьовувати графи з ребрами від'ємної ваги і виявляти від'ємні цикли. Часова складність алгоритму $O(V \cdot E)$ робить його менш ефективним для великих мереж, проте він є незамінним у задачах із від'ємними вагами [13].

Алгоритм Флойда–Варшалла знаходить найкоротші шляхи між усіма парами вершин графа за час $O(V^3)$, що робить його придатним для відносно невеликих мереж із попереднім кешуванням результатів [14].

Таблиця 1.1

Порівняння алгоритмів пошуку найкоротшого шляху

Назва алгоритму	**Часова складність**	**Підтримка від'ємних ваг**	**Доцільність для залізничних маршрутів**
Алгоритм Дейкстри	O((V + E) log V)	Ні	Висока ефективність для маршрутів із додатними вагами (час, відстань). Оптимальний для реального часу та великих мереж.
Алгоритм A* (A–зірка)	O((V + E) log V) (залежить від евристики)	Ні	Дуже ефективний для великих мереж за наявності хорошої евристики (наприклад, географічної відстані). Підходить для швидкого пошуку конкретного маршруту.
Алгоритм Белмана–Форда	O(V · E)	Так	Доцільний у випадках наявності від'ємних ваг (наприклад, штрафів або знижок). Менш ефективний для великих залізничних систем.
Алгоритм Флойда–Варшалла	$O(V^3)$	Так	Підходить для невеликих мереж або при необхідності попереднього обчислення всіх маршрутів між станціями (кешування).
Алгоритм Джонсона	$O(V^2 \log V + V \cdot E)$	Так	Ефективний для розріджених графів із можливими від'ємними вагами. Може використовуватись для великих транспортних мереж при складних тарифних моделях.

Для реалізованого проекту обрано алгоритм Дейкстри як найбільш оптимальний для задачі пошуку шляху мінімальної тривалості в графі залізничних маршрутів, де всі ваги ребер є невід'ємними. Бідирекціональний характер побудованого графа (метод buildGraph() клас RouteFinderService додає ребра в обидва напрямки) забезпечує коректне знаходження шляху незалежно від напряму руху [15].

1.4 Вибір технологічного стеку та його обґрунтування

Вибір технологій для розробки веб–застосунку здійснювався на основі аналізу вимог предметної області, доцільності застосування, зрілості екосистеми та наявності відповідної документації.

Java 21 та Spring Boot 3.2.0 обрані для реалізації бекенду. Java є однією з найбільш поширених мов програмування для корпоративних застосунків;

Spring Boot забезпечує швидке налаштування серверної частини, вбудовану підтримку REST API та підключення до баз даних [16]. Анотаційний підхід до конфігурації (@RestController, @Service, @Repository) зменшує обсяг шаблонного коду, підвищуючи продуктивність розробки.

Spring Data JPA з Hibernate ORM використовуються для роботи з базою даних. JPA (Jakarta Persistence API) забезпечує об'єктно–реляційне відображення (ORM), що дозволяє маніпулювати даними через Java–об'єкти без написання SQL–запитів вручну [17]. Інтерфейси JpaRepository надають стандартні CRUD–операції з мінімальним обсягом коду.

PostgreSQL 16 обрано як систему управління базами даних завдяки надійності, підтримці складних запитів, транзакцій ACID, а також використанню схем (schema), що дозволяє логічно ізолювати таблиці залізничної системи у схемі railway [18].

Lombok – бібліотека Java, що усуває шаблонний код завдяки анотаціям @Data, @Builder, @NoArgsConstructor, @AllArgsConstructor [19]. Це підвищує читабельність сутностей і зменшує їхній обсяг.

MapStruct 1.5.5 використовується для генерації типобезпечних маперів між Entity і DTO на етапі компіляції [20]. Відсутність рефлексії забезпечує вищу продуктивність порівняно з аналогами (ModelMapper).

Springdoc OpenAPI 2.2.0 (Swagger UI) забезпечує автоматичну генерацію інтерактивної документації REST API на основі анотацій Spring MVC, що спрощує тестування та інтеграцію.

React 19 та Tailwind CSS 3.4 обрані для фронтенду. React – найпопулярніша JavaScript–бібліотека для побудови компонентних інтерфейсів із ефективним оновленням DOM через Virtual DOM [21]. Tailwind CSS – утилітарний фреймворк, що дозволяє стилізувати компоненти безпосередньо в JSX без окремих CSS–файлів.

Docker та Docker Compose використовуються для контейнеризації та оркестрації: три контейнери (railway–db, railway–backend, railway–frontend) об'єднані в єдину мережу, що спрощує розгортання і забезпечує ізоляцію середовищ [22].

Таблиця 1.2

Зведена таблиця технологічного стеку проекту railway–web–app.

Технологія	**Версія**	**Призначення в проекті**
Java	21	Основна мова програмування бекенду; забезпечує високу продуктивність, багатопоточність та надійність корпоративного рівня
Spring Boot	3.2.0	Фреймворк для створення REST API; спрощує конфігурацію, інтеграцію компонентів та запуск серверної частини
Spring Data JPA	3.x	Спрощує доступ до бази даних через ORM, надає CRUD–операції без написання SQL
Hibernate ORM	6.x	Реалізація JPA; забезпечує об'єктно–реляційне відображення між сутностями Java та таблицями БД
PostgreSQL	16	Реляційна СУБД для зберігання даних; підтримує складні запити, ACID–транзакції та схеми
Lombok	1.18.x	Генерація шаблонного коду (гетери, сетери, конструктори) через анотації
MapStruct	1.5.5	Генерація типобезпечних маперів між Entity і DTO на етапі компіляції
Springdoc OpenAPI	2.2.0	Автоматична генерація документації REST API та інтерактивний інтерфейс Swagger UI
React	19	Бібліотека для побудови користувацького інтерфейсу (frontend), компонентний підхід
Tailwind CSS	3.4	Утилітарний CSS–фреймворк для швидкої стилізації інтерфейсу без окремих CSS–файлів
Docker	24.x	Контейнеризація додатку для ізольованого середовища виконання
Docker Compose	2.x	Оркестрація контейнерів (бекенд, фронтенд, база даних) в єдиній мережі
Maven / Gradle	x	Система збірки та керування залежностями Java–проекту

Висновки до розділу 1

У першому розділі проведено аналіз предметної області залізничних перевезень та виявлено основні сутності системи: маршрут, рейс, поїзд, станція, місце, пасажир, бронювання і розклад. Виконано огляд існуючих

систем управління залізничними маршрутами (UZ Online, RailEurope, DB Navigator) і визначено їхні переваги та обмеження. Здійснено порівняльний аналіз алгоритмів пошуку найкоротших шляхів, за результатами якого обрано алгоритм Дейкстри як найбільш відповідний задачі пошуку оптимального маршруту за тривалістю в графах із невід'ємними вагами. Обґрунтовано вибір технологічного стеку: Java 21 + Spring Boot 3.2 на бекенді, PostgreSQL як СУБД, React 19 + Tailwind CSS на фронтенді, Docker Compose для оркестрації. Визначений стек дозволяє реалізувати повнофункціональний веб–застосунок управління залізничними маршрутами відповідно до сучасних вимог.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ВЕБ–ЗАСТОСУНКУ ДЛЯ КЕРУВАННЯ ЗАЛІЗНИЧНИМИ РЕЙСАМИ

2.1 Вимоги до системи та постановка задачі проектування

На основі аналізу предметної області та огляду існуючих систем сформульовано функціональні та нефункціональні вимоги до проектованого веб–застосунку управління залізничними маршрутами.

Функціональні вимоги включають такі групи:

Управління довідниками: повний CRUD–функціонал для сутностей – міста (City), станції (Station), поїзди (Train), маршрути (Route), проміжні станції маршрутів (RouteStation), місця у вагонах (Seat), пасажири (Passenger), рейси (Trip), розклади рейсів (TripSchedule) та бронювання (Booking).

Пошук маршрутів: надання функції пошуку оптимального шляху між двома станціями з використанням алгоритму Дейкстри. Результат повинен містити перелік сегментів шляху з назвами маршрутів, відстанню, тривалістю та кількістю пересадок.

Панель користувача: можливість перегляду доступних рейсів, вибору місця і оформлення бронювання з перевіркою доступності обраного місця.

Адмін–панель: інструменти для управління всіма сутностями системи через веб–інтерфейс з підтримкою операцій перегляду, створення, редагування і видалення.

Нефункціональні вимоги визначають такі характеристики системи:

Продуктивність: час відповіді REST API не більше 1 секунди для CRUD–операцій і не більше 3 секунд для алгоритму пошуку маршрутів у мережі до 1000 станцій.

Масштабованість: архітектура має передбачати можливість горизонтального масштабування завдяки розділенню бекенду та фронтенду на незалежні контейнери.

Сумісність: фронтенд повинен коректно відображатися у сучасних браузерах (Chrome, Firefox, Edge) без встановлення додаткових плагінів.

Безпека: система повинна підтримувати CORS–захист для обмеження доступу до API лише з дозволених джерел [23].

2.2 Проектування архітектури веб–застосунку

Архітектура розробленого веб–застосунку railway–web–app базується на класичній триланцюговій (three–tier) моделі: рівень представлення (фронтенд), рівень бізнес–логіки (бекенд) та рівень даних (база даних) [24].

Рівень представлення реалізовано як SPA–застосунок на React 19. Головний компонент App організовує навігацію між сторінками: HomePage (головна), AdminPanel (адмін–панель), AdminResourcePage (управління ресурсом), UserPanel (панель користувача) та ShortestPathPage (пошук маршруту). Взаємодія з бекендом відбувається через єдиний об'єкт api, що інкапсулює виклики Fetch API до REST–ендпоінтів.

Рівень бізнес–логіки реалізовано як REST API на Spring Boot 3.2. Кожен ресурс системи має відповідний контролер (@RestController), що делегує виконання відповідним сервісам (@Service). Рівень доступу до даних реалізовано через репозиторії (@Repository), що розширюють JpaRepository [4].

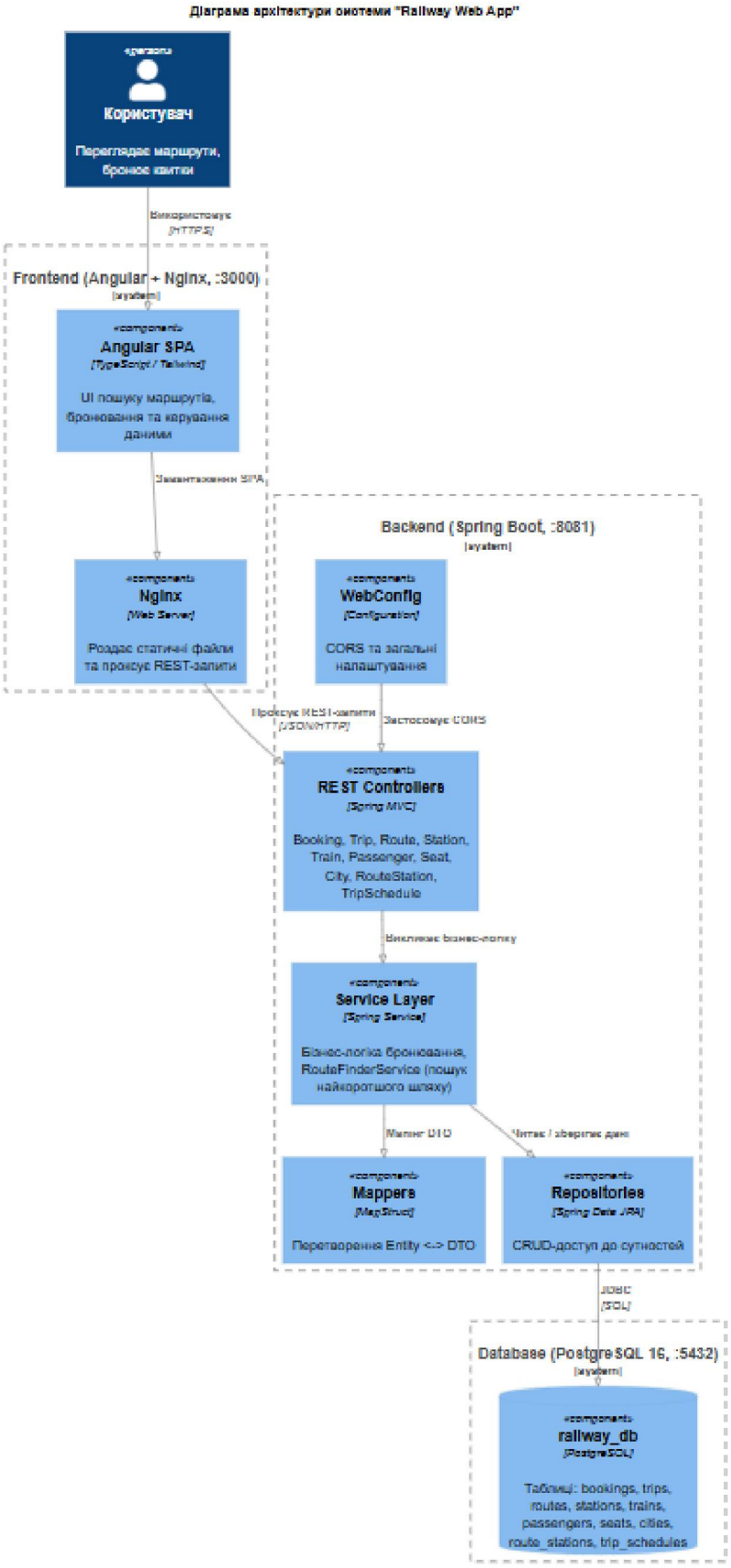


Рисунок 2.1 – Діаграма архітектури системи.

2.3 Проектування бази даних та моделі даних

База даних проекту розгорнута в PostgreSQL 16 у схемі railway. Структура бази даних спроектована відповідно до принципів нормалізації (третя нормальна форма) для виключення надлишковості та забезпечення цілісності даних [18].

Таблиця cities зберігає інформацію про міста. Таблиця stations зберігає залізничні станції з прив'язкою до міст через зовнішній ключ city_id; кожна станція має унікальний код (code) та адресу. Таблиця trains містить дані про рухомий склад: номер (number, унікальний), найменування, тип (train_type) та загальну кількість місць (total_seats). Таблиця seats деталізує місця у вагоні: номер місця (seat_number) та прапорець доступності (is_available), що автоматично змінюється при бронюванні або скасуванні.

Таблиця routes визначає маршрути між станціями, включаючи назву, початкову (departure_station_id) і кінцеву (arrival_station_id) станції, відстань (distance_km) і тривалість (duration_minutes). Таблиця route_stations є реалізацією зв'язку між routes та stations з атрибутами: порядковий номер станції (station_order) і тривалість зупинки (stop_duration_minutes). Порядок станцій є критично важливим для побудови графа в RouteFinderService.

Таблиця trips представляє конкретні рейси з прив'язкою до поїзда і маршруту, датами та часом відправлення і прибуття, кількістю доступних місць та базовою ціною. Таблиця trip_schedules зберігає детальний розклад зупинок рейсу. Таблиця passengers зберігає дані пасажирів. Таблиця bookings пов'язує пасажира, рейс і місце, фіксуючи дату бронювання та ціну [5].

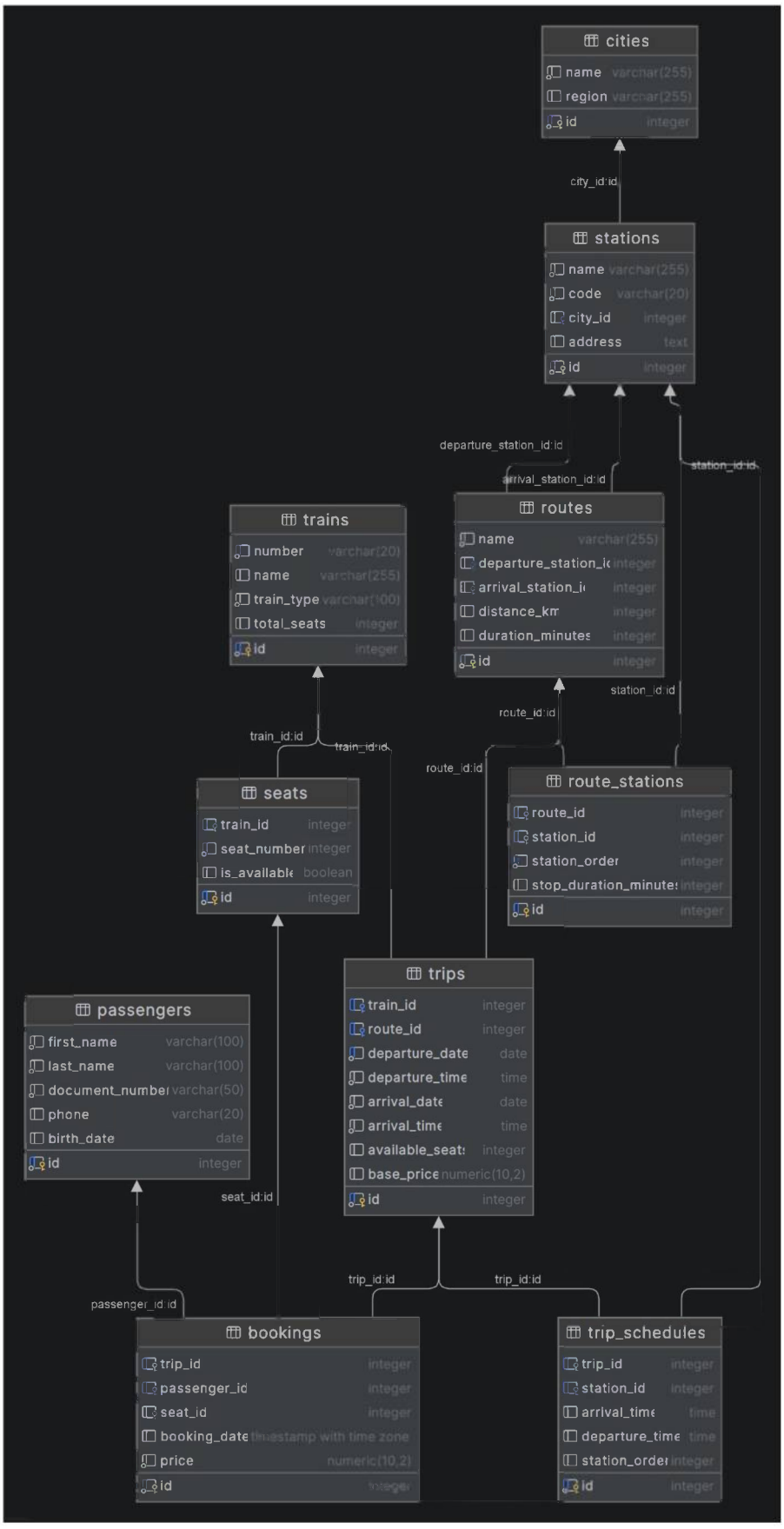


Рисунок 2.2 – ER–діаграма бази даних схеми railway з усіма таблицями та зв'язками.

2.4 Опис програмних сутностей та шарів застосунку

Пакет org.example.entities містить JPA–сутності, що відображають таблиці бази даних. Анотація @Entity позначає клас як JPA–сутність, @Table задає назву таблиці та схему. Анотації Lombok (@Data, @Builder, @NoArgsConstructor, @AllArgsConstructor) генерують шаблонний код.

City – сутність міста, що слугує довідниковою сутністю для географічної прив'язки станцій. Анотація @Table(name = "cities", schema = "railway") визначає розташування в схемі railway. Поля: id, name (обов'язкова назва міста), region (область). Зв'язок OneToMany до Station задано з FetchType.LAZY та CascadeType.ALL, що забезпечує каскадне збереження пов'язаних станцій при створенні міста.

Station – сутність залізничної станції. Анотація @Table(name = "stations", schema = "railway") вказує розташування в схемі. Поля: id, name, code (унікальний), city (ManyToOne до City), address. Зв'язки OneToMany до RouteStation і TripSchedule задані з FetchType.LAZY для оптимізації запитів.

Train – сутність поїзда з полями number (унікальний), name, trainType, totalSeats. Пов'язаний зі Seat і Trip через OneToMany.

Seat – сутність окремого місця у поїзді. @Table(name = "seats", schema = "railway"). Поля: id, seatNumber (обов'язковий номер місця), isAvailable (прапорець доступності, за замовчуванням true). Зв'язок ManyToOne до Train визначає належність місця конкретному поїзду, зв'язок OneToMany до Booking дозволяє відстежувати історію бронювань цього місця. Прапорець isAvailable є критичним для логіки BookingService: перевіряється при створенні бронювання і автоматично перемикається між значеннями true/false під час створення та скасування бронювання.

Route – сутність маршруту. Містить посилання на departureStation і arrivalStation (ManyToOne), дистанцію та тривалість, списки RouteStation і Trip (каскадне збереження CascadeType.ALL).

RouteStation – сутність проміжної станції маршруту. Зберігає посилання на route і station, stationOrder і stopDurationMinutes. Є ключовою для побудови графа в RouteFinderService.

Trip – сутність рейсу з посиланнями на train і route, departureDate, departureTime, arrivalDate, arrivalTime, availableSeats і basePrice. Каскадно пов'язані Booking і TripSchedule.

TripSchedule – сутність детального розкладу зупинок рейсу на проміжних станціях. @Table(name = "trip_schedules", schema = "railway"). Поля: id, arrivalTime (час прибуття на станцію), departureTime (час відправлення зі станції), stationOrder (порядковий номер станції у рейсі). Зв'язки ManyToOne до Trip і Station з FetchType.LAZY прив'язують розклад до конкретного рейсу та станції, що дозволяє формувати повний маршрут рейсу з часовими мітками для кожної зупинки.

Passenger – сутність пасажира, який здійснює бронювання квитків. @Table(name = "passengers", schema = "railway"). Поля: id, firstName (ім'я), lastName (прізвище), documentNumber (обов'язковий номер документа), phone (контактний номер телефону), birthDate (дата народження). Зв'язок OneToMany до Booking з каскадним збереженням CascadeType.ALL дозволяє отримати всі бронювання пасажира та забезпечує видалення пов'язаних бронювань разом із пасажиром.

Booking – сутність бронювання, що пов'язує Trip, Passenger і Seat. Поле bookingDate автоматично встановлюється через @PrePersist при відсутності явно заданого значення.

Пакет org.example.dto містить об'єкти передачі даних (DTO) – «пласке» представлення сутностей для серіалізації в JSON. DTO ізолюють бізнес–логіку від зовнішнього представлення, запобігаючи витоку внутрішньої структури бази даних через API.

Пакет org.example.mapper містить MapStruct–маппери для перетворення між Entity і DTO. Маппери генеруються на етапі компіляції, забезпечуючи типобезпечне відображення без рефлексії та з максимальною продуктивністю [20].

Пакет org.example.repository містить Spring Data JPA репозиторії – інтерфейси, що розширюють JpaRepository<T, ID> і надають стандартні методи findAll(), findById(), save(), deleteById() без необхідності реалізації SQL–запитів.

Пакет org.example.service містить сервісні класи з бізнес–логікою. BookingService реалізує логіку бронювання: перевірку доступності місця, зміну статусу is_available, транзакційну цілісність через @Transactional. RouteFinderService реалізує алгоритм Дейкстри для пошуку оптимального маршруту.

Пакет org.example.controller містить REST–контролери. Кожен ресурс має п'ять стандартних ендпоінтів: GET /{resource} (список), GET /{resource}/{id} (одна сутність), POST /{resource} (створення), PUT /{resource}/{id} (оновлення), DELETE /{resource}/{id} (видалення).

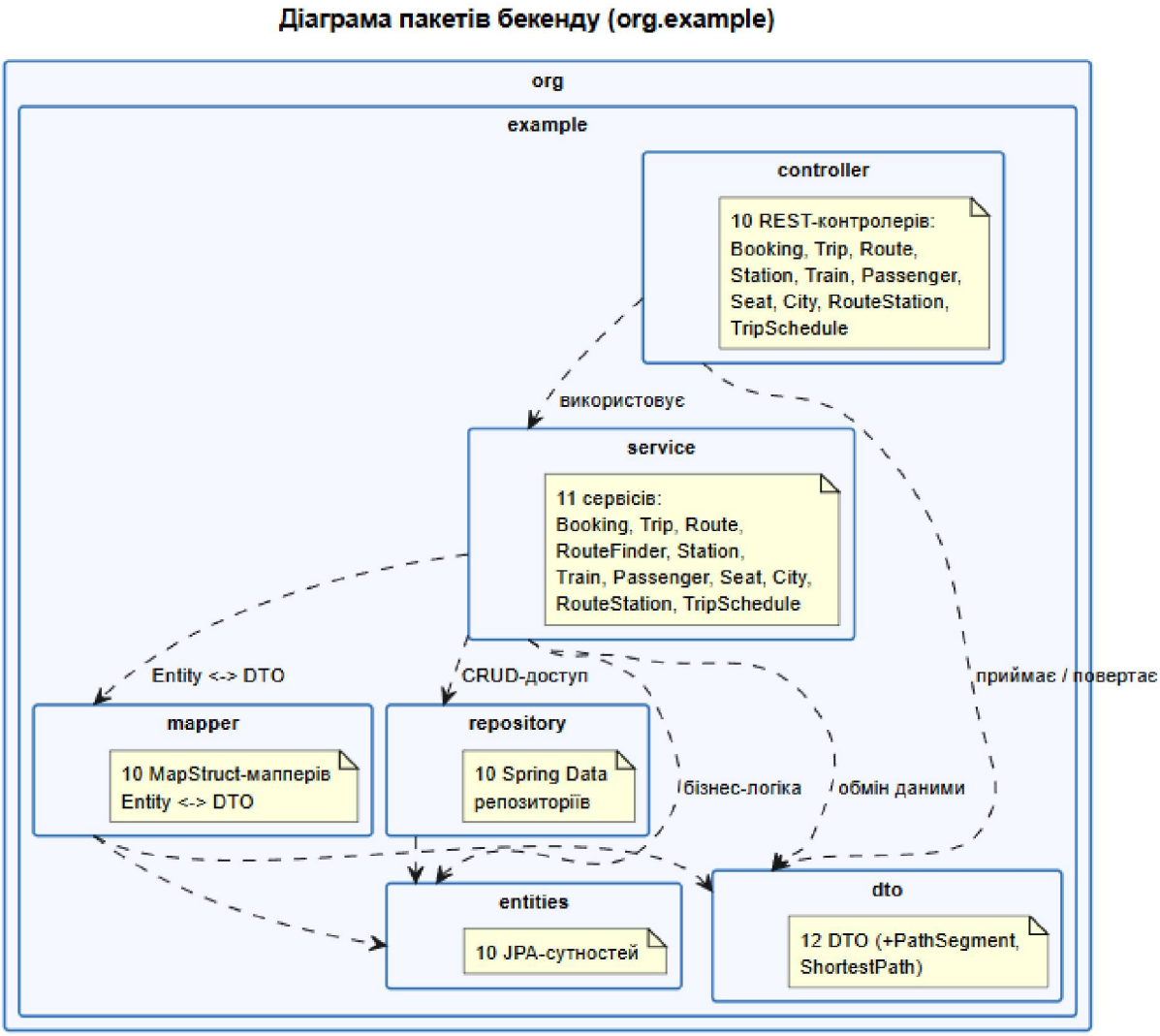


Рисунок 2.3 – Діаграма пакетів бекенду.

2.5 Проектування алгоритму пошуку оптимального маршруту

Клас RouteFinderService реалізує алгоритм Дейкстри для пошуку оптимального маршруту між двома станціями з мінімальною сумарною тривалістю [11]. Метод findShortestPath(Long fromId, Long toId) приймає ідентифікатори початкової і кінцевої станцій і повертає ShortestPathDTO з повним описом маршруту.

Алгоритм складається з трьох основних етапів:

Побудова графа (buildGraph()): завантажуються всі маршрути з бази даних. Для кожного маршруту станції сортуються за stationOrder. Для

кожної пари суміжних станцій додається двонаправлене ребро типу Edge з вагою, що відповідає пропорційній тривалості сегмента (totalDuration / кількість_сегментів).

Виконання Дейкстри: використовується PriorityQueue<long[]>, де кожен елемент містить ідентифікатор вершини та поточну відому відстань. Мапи prevEdge і prevNode зберігають попередні ребро та вузол для відновлення шляху. Обробка завершується при досягненні цільової вершини або спустошенні черги.

Відновлення шляху: від цільової вершини за допомогою prevEdge і prevNode відновлюється ланцюжок сегментів PathSegmentDTO. Підраховуються загальна тривалість, відстань та кількість пересадок (countTransfers()).



Рисунок 2.4 – Блок–схема алгоритму Дейкстри в RouteFinderService.

2.6 Проектування REST API та міжсервісної взаємодії

REST API бекенду організований згідно з принципами RESTful–архітектури [25]: стандартні HTTP–методи (GET, POST, PUT, DELETE), ресурсно–орієнтована адресація, відсутність стану (stateless), стандартні коди відповіді HTTP (200 OK, 201 Created, 204 No Content, 400 Bad Request, 404 Not Found).

Налаштування CORS здійснюється у класі WebConfig (@Configuration, @EnableWebMvc), що реалізує WebMvcConfigurer. Метод addCorsMappings дозволяє запити з http://localhost:3000 до всіх ендпоінтів /api/** методами GET, POST, PUT, DELETE.

Серіалізацію Java–об'єктів у JSON забезпечує Jackson із підтримкою типів Java Date/Time (jackson–datatype–jsr310). Це дозволяє коректно серіалізувати поля LocalDate, LocalTime і ZonedDateTime у форматі ISO 8601. Валідація вхідних даних здійснюється через @Valid у контролерах, що при помилці автоматично повертає 400 Bad Request.

Висновки до розділу 2

У другому розділі сформульовано функціональні та нефункціональні вимоги до системи управління залізничними маршрутами. Спроектовано триланцюгову архітектуру застосунку: React SPA–фронтенд, Spring Boot REST API бекенд і PostgreSQL база даних, розгорнуті у Docker–контейнерах. Розроблено ER–діаграму бази даних у схемі railway, що включає 10 взаємопов'язаних таблиць. Детально описано структуру пакетів бекенду та призначення кожного шару: entities, dto, mapper, repository, service, controller. Розглянуто алгоритм Дейкстри, реалізований у RouteFinderService, та його адаптацію до задачі пошуку залізничних маршрутів. Спроектовано REST API з 50+ ендпоінтами, CORS–захистом та валідацією вхідних даних.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ВЕБ–ЗАСТОСУНКУ

3.1 Розгортання середовища розробки та налаштування проекту

Розробка веб–застосунку railway–web–app здійснювалася у середовищі IntelliJ IDEA як для бекенду (Spring Boot), так і для редагування фронтенду (React). Система контролю версій Git використовувалася для відстеження змін у коді. Інфраструктура проекту розгорнута за допомогою Docker Compose, що забезпечує ізольоване та відтворюване середовище запуску.

Файл docker–compose.yml визначає три сервіси: db (PostgreSQL 16 Alpine на порту 5432 з volume railway_pgdata для персистентності даних), backend (Spring Boot на порту 8081) і frontend (React–застосунок, зібраний Webpack і обслуговуваний Nginx на порту 3000). Між сервісами налаштована залежність: backend очікує healthcheck бази даних (pg_isready), frontend запускається після бекенду.

Конфігурація підключення передається через змінні середовища Docker:

- SPRING_DATASOURCE_URL
- SPRING_DATASOURCE_USERNAME
- SPRING_DATASOURCE_PASSWORD.

Такий підхід відповідає принципу 12–Factor App та дозволяє змінювати конфігурацію без перезбірки Docker–образу [16].

Збірка бекенду здійснюється за допомогою Maven із POM–файлом (pom.xml), де визначено parent–артефакт spring–boot–starter–parent версії 3.2.0. Maven–плагін Spring Boot пакує застосунок у виконуваний JAR–файл, що запускається всередині контейнера.

3.2 Реалізація серверної частини (бекенду)

Точкою входу бекенд–застосунку є клас Main.java з анотацією @SpringBootApplication, що активує автоконфігурацію Spring Boot, сканування компонентів і конфігурацію JPA.

Контролери реалізують RESTful API за єдиним шаблоном. BookingController анотований @RestController та @RequestMapping("/api/bookings"). Ін'єкція залежності BookingService здійснюється через конструктор (@RequiredArgsConstructor від Lombok). Метод createBooking приймає @Valid @RequestBody BookingDTO, що активує валідацію вхідних даних, і повертає ResponseEntity із кодом 201 Created. Аналогічну структуру мають контролери TripController, RouteController, StationController, TrainController та інші – всього 10 REST–контролерів, кожен з яких відповідає одному ресурсу.

Транзакційне управління реалізовано на рівні сервісів через @Transactional. У BookingService методи createBooking, updateBooking і deleteBooking позначені цією анотацією, що гарантує атомарність: при бронюванні місця оновлення Booking і зміна is_available у Seat відбуваються в одній транзакції. При виключенні транзакція автоматично відкочується.

Логіка перевірки доступності місця в BookingService: отримується пов'язаний Seat і перевіряється is_available. Якщо місце зайняте – кидається IllegalStateException. При успішному бронюванні seat.setIsAvailable(false) зберігається через seatRepository, після чого зберігається Booking. Метод deleteBooking встановлює is_available = true, звільняючи місце для наступних бронювань.

RouteFinderService реалізує алгоритм Дейкстри. Внутрішній клас Edge зберігає параметри ребра: ідентифікатор кінцевої вершини, тривалість сегмента, відстань, ідентифікатор та назву маршруту. Метод buildGraph() формує граф з усіх маршрутів бази даних: для кожної пари суміжних

станцій за stationOrder додаються двонаправлені ребра з пропорційними вагами. Метод findShortestPath() виконує Дейкстру з PriorityQueue та відновлює шлях через prevEdge і prevNode, формуючи ShortestPathDTO із сегментами PathSegmentDTO [11].

3.3 Реалізація клієнтської частини (фронтенду)

Фронтенд реалізований як SPA на React 19 в одному файлі App.js. Компонент App управляє навігацією через стан currentPage і currentResource (хук useState). Функція navigate(page, resource) оновлює стан без перезавантаження сторінки, реалізуючи клієнтську маршрутизацію без React Router.

Компонент AdminPanel відображає сітку з 10 карток для ресурсів: bookings, cities, passengers, routes, routeStations, seats, stations, trains, trips, tripSchedules. Кожна картка при натисканні відкриває AdminResourcePage для відповідного ресурсу.

Компонент AdminResourcePage реалізує CRUD–функціонал для будь–якого ресурсу. Хук useCallback мемоізує fetchData, хук useEffect ініціює завантаження при зміні resource. Методи handleCreate, handleUpdate і handleDelete викликають відповідні функції об'єкта api та оновлюють стан через fetchData. Пошук по таблиці реалізовано через стан searchTerm і фільтрацію даних у рендері.

Об'єкт api інкапсулює HTTP–запити через Fetch API: методи get, post, put, delete. Базовий URL http://localhost:8081/api вказує на бекенд. Заголовок Content–Type: application/json встановлюється для POST і PUT. Обробка помилок DELETE–запитів реалізована через перевірку res.ok.

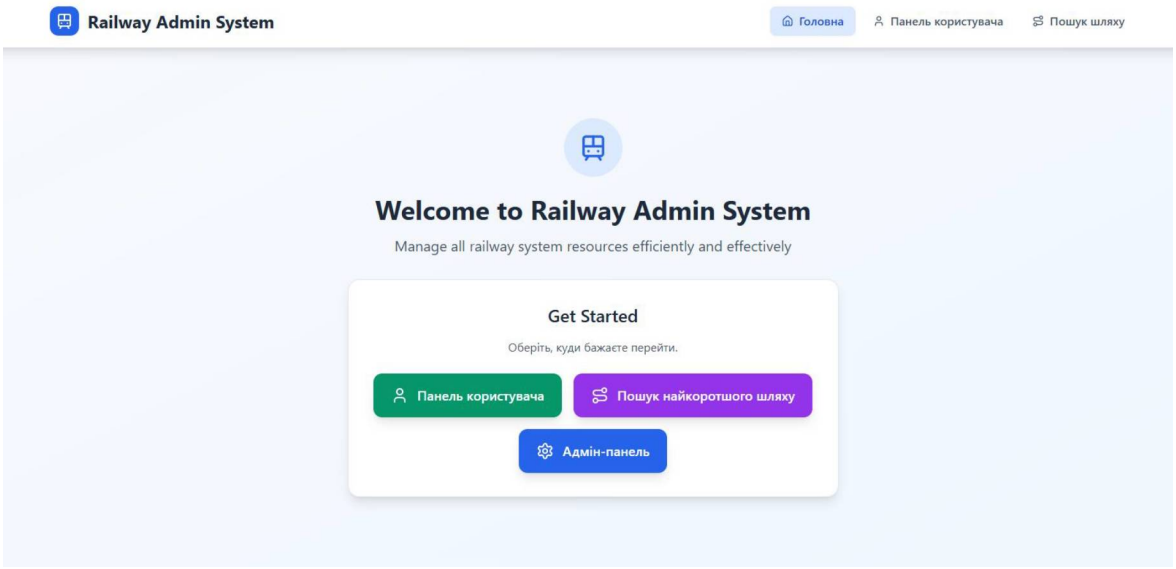


Рисунок 3.1 – Головна сторінка Railway Admin System.

Компонент ShortestPathPage реалізує форму вибору початкової і кінцевої станцій з динамічно завантаженими списками. При відправці форми виконується запит до GET /api/routes/shortest–path. Результат відображається у вигляді картки з переліком сегментів PathSegmentDTO, загальною тривалістю, відстанню та кількістю пересадок.

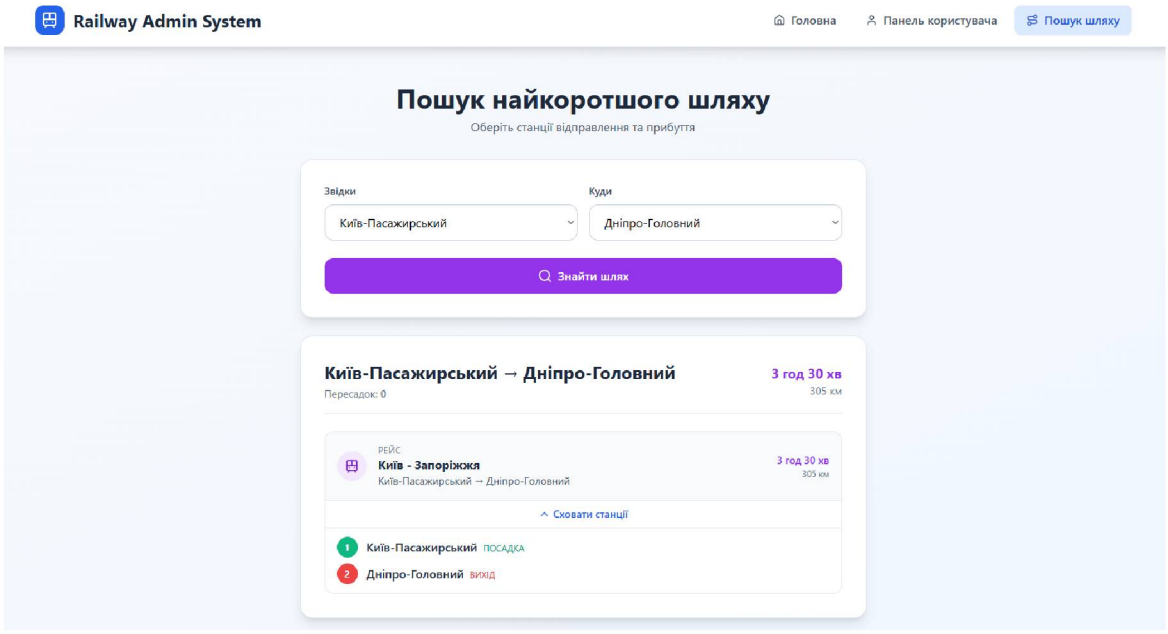


Рисунок 3.2 – Сторінка ShortestPathPage з результатом пошуку маршруту.

Стилізація виконана засобами Tailwind CSS 3.4 без окремих CSS–файлів. Утилітарні класи: bg–blue–600 (кольори), rounded–xl (заокруглення кутів), shadow–lg/hover:shadow–xl (тіні), hover:–translate–y–1 і transition–all duration–200 (анімації). Адаптивна верстка реалізована через брейкпоінти sm:, lg:, xl: у класах сітки grid.

Інтерфейс користувача поділено на дві основні частини: публічну панель користувача, призначену для пасажирів, і захищену адмін–панель, доступну адміністраторам системи. Обидві панелі реалізовано як набір функціональних React–компонентів, що повторно використовують спільні UI–елементи (кнопки, картки, поля форм, таблиці), оформлені утилітарними класами Tailwind CSS.

Панель користувача (UserPanel) є стартовою точкою застосунку та надає пасажиру доступ до основних сценаріїв роботи: перегляду списку доступних маршрутів, пошуку найкоротшого шляху між станціями, ознайомлення з розкладом рейсів та бронювання квитків. Компонент UserPanel побудований як сторінка–лендинг із навігаційним меню у верхній частині, блоком швидкого пошуку маршруту (форма з двома полями «Станція відправлення» і «Станція призначення») та секцією з популярними напрямками, оформленими у вигляді адаптивної сітки карток.

Для завантаження довідкових даних (міст, станцій, розкладу) використовується хук useEffect разом із модулем api, описаним вище. Отримані списки кешуються у локальному стані компонента, що дозволяє уникнути повторних HTTP–запитів під час зміни значень у полях форм. Поля вибору станцій реалізовані як компоненти <select> з опціями, що формуються динамічно за даними, отриманими від ендпоінта GET /api/stations.

Після відправки форми пошуку користувач перенаправляється на сторінку ShortestPathPage, де відображається детальна інформація про

знайдений маршрут: послідовність сегментів, проміжні станції, час відправлення та прибуття, тривалість кожного сегмента, загальна відстань і кількість пересадок. Результати подані у вигляді вертикальної стрічки карток, де кожна картка відповідає одному сегменту PathSegmentDTO і містить іконки поїзда та стрілки напрямку руху.

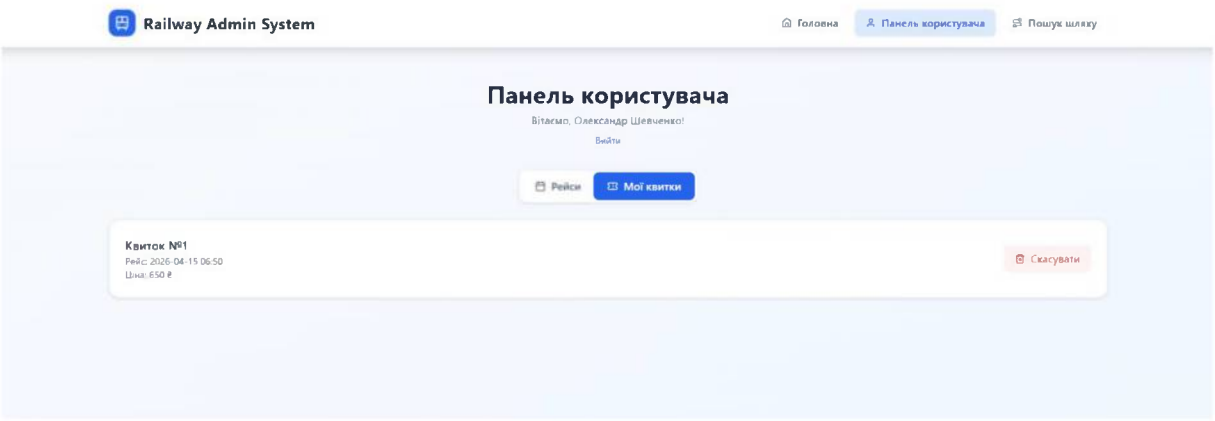


Рисунок 3.3 – Сторінка з квитками в панелі користувача.

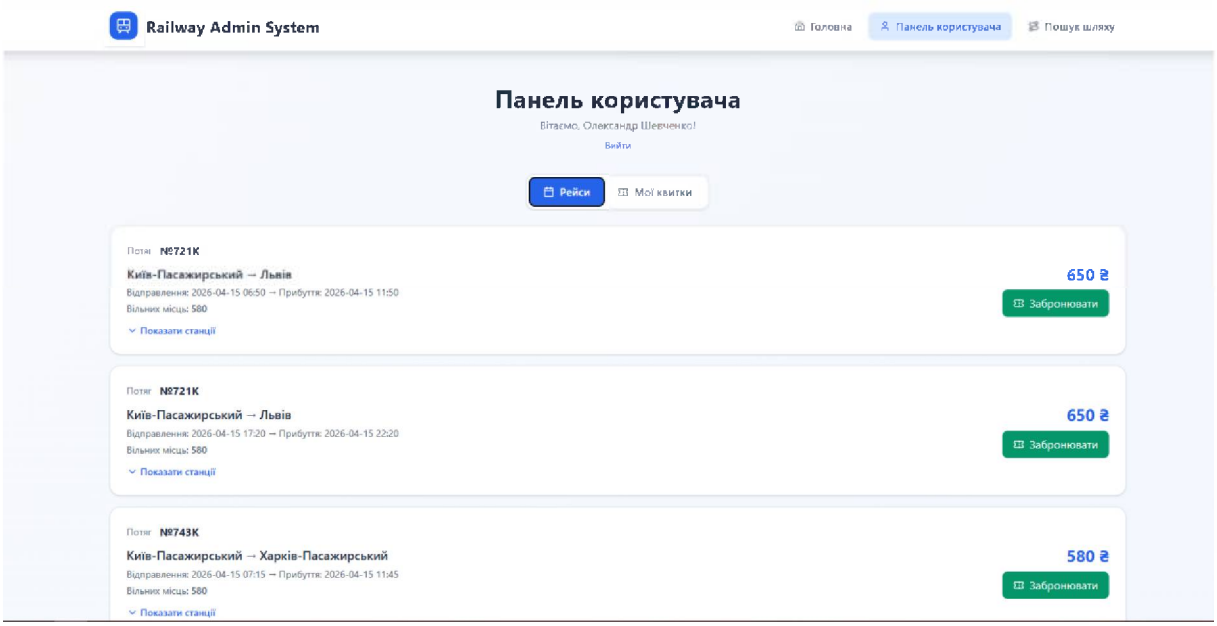


Рисунок 3.4 – Сторінка з рейсами в панелі користувача.

Адмін–панель (AdminPanel) доступна після переходу за відповідним пунктом меню та реалізує повний набір CRUD–операцій над усіма десятьма ресурсами системи. Головний екран адмін–панелі являє собою сітку з карток, де кожна картка відповідає окремій сутності (bookings, cities, passengers, routes, routeStations, seats, stations, trains, trips, tripSchedules) і супроводжується піктограмою та стислим описом призначення ресурсу. Сітка адаптується під розмір екрана завдяки брейкпоінтам.

При натисканні на картку відкривається сторінка AdminResourcePage, що містить заголовок з назвою ресурсу, поле пошуку по таблиці, кнопку «Додати новий запис», а також основну таблицю з даними. Рядки таблиці підсвічуються при наведенні, а у правій колонці кожного рядка розташовані кнопки «Редагувати» та «Видалити». Форма створення і редагування запису реалізована як модальне вікно, поля якого формуються динамічно на основі схеми конкретного ресурсу.

Для зручності роботи адміністратора реалізовано клієнтську фільтрацію записів за рядковим запитом (стан searchTerm), підтвердження видалення через діалогове вікно window.confirm, а також індикатори завантаження та повідомлення про помилки, що з’являються у верхній частині сторінки після невдалих запитів до REST API.

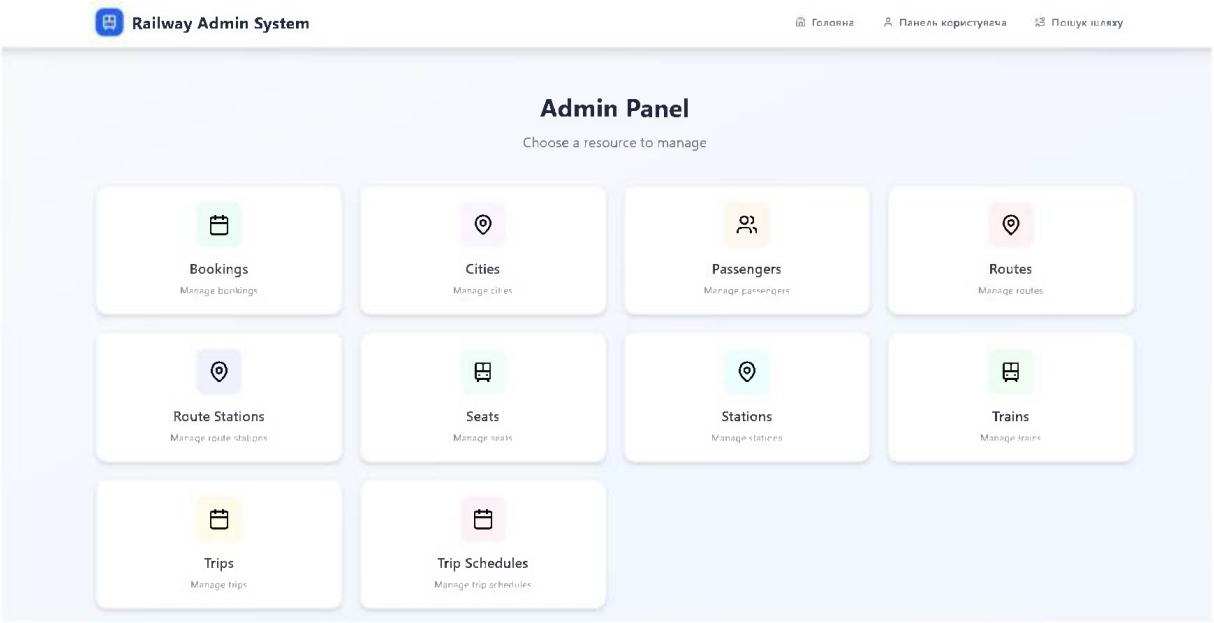


Рисунок 3.5 – Головна сторінка адмін–панелі зі списком доступних ресурсів.

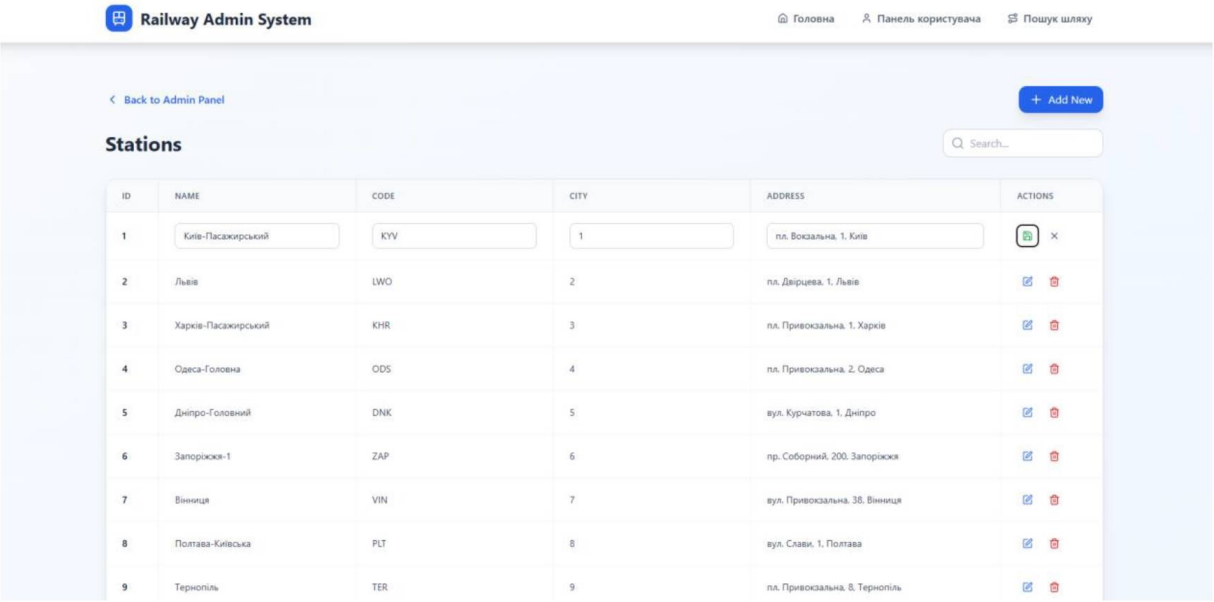


Рисунок 3.6 – Сторінка AdminResourcePage з таблицею записів та формою редагування.

Таким чином, розділення інтерфейсу на панель користувача та адмін–панель дозволяє одночасно забезпечити простоту використання для

кінцевих пасажирів і повноцінний інструментарій для адміністраторів, які керують довідниковими даними та розкладом рейсів. Обидві панелі працюють поверх єдиного REST API бекенду, використовують спільний набір UI–компонентів та стилів Tailwind CSS, що спрощує підтримку та подальший розвиток застосунку.

3.4 Тестування та перевірка функціональності

3.4.1 Тестування CRUD–операцій адмін–панелі

Функціональне тестування CRUD–операцій проводилося через інтерфейс Swagger UI та безпосередньо через адмін–панель застосунку.

Тестування створення маршруту (POST /api/routes): надсилався JSON із назвою, ідентифікаторами станцій відправлення і прибуття, відстанню і тривалістю. Очікувана відповідь – 201 Created з RouteDTO із присвоєним id.

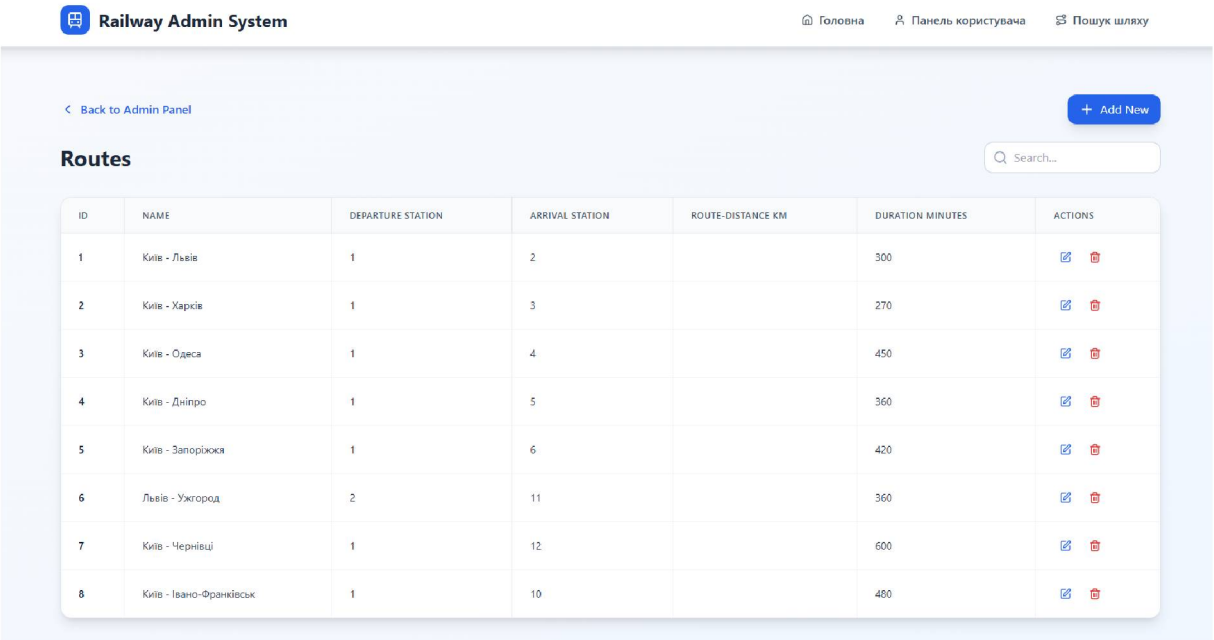


Рисунок 3.7 – Адмін–панель застосунку – сторінка управління маршрутами (Routes) з таблицею наявних маршрутів та активними кнопками редагування і видалення.

Тестування бронювання: при спробі забронювати вже зайняте місце (is_available = false) система повертає 500 з повідомленням «Seat X is already booked». Після скасування бронювання місце знову стає доступним (is_available = true), і повторне бронювання проходить успішно.

Тестування перегляду пасажирів (GET /api/passengers) – повертає список пасажирів з кодом 200 OK. Тестування видалення рейсу з пов'язаними бронюваннями – каскадне видалення CascadeType.ALL забезпечує коректне видалення всіх пов'язаних записів Booking.

3.4.2 Тестування алгоритму пошуку маршрутів

Тестування RouteFinderService здійснювалося на тестовому наборі: мережа з кількох міст, станцій і маршрутів між ними.

Сценарій 1 – пряме з'єднання: для двох станцій, з'єднаних одним маршрутом, алгоритм коректно повертає один PathSegmentDTO, transfers = 0, а тривалість відповідає пропорційній вазі сегмента.

Сценарій 2 – маршрут із пересадкою: для станцій, з'єднаних через транзитну, алгоритм повертає два сегменти з різними routeId, transfers = 1, загальна тривалість = сума тривалостей обох сегментів.

Сценарій 3 – однакові станції: при fromId == toId метод findShortestPath кидає IllegalArgumentException «Станції відправлення та прибуття співпадають», що забезпечує коректну обробку граничного випадку.

Сценарій 4 – відсутній шлях: для станцій, між якими немає маршрутів у базі, алгоритм кидає IllegalStateException «Шлях між станціями не знайдено», що відображається як помилка у фронтенді.

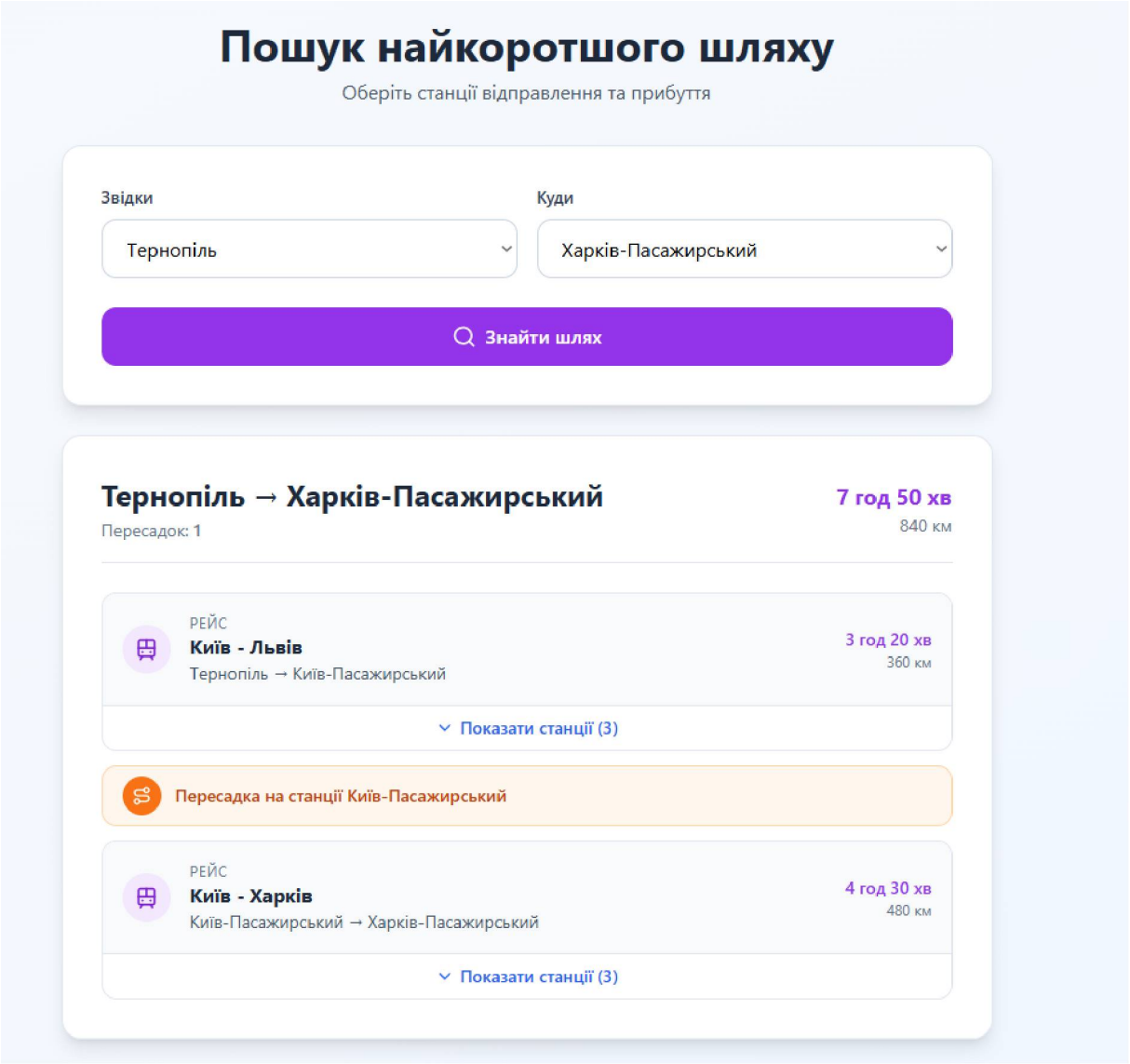


Рисунок 3.8 – Сторінка ShortestPathPage з тестовим результатом для маршруту з пересадкою.

Висновки до розділу 3

У третьому розділі описано практичну реалізацію веб–застосунку railway–web–app. Розгорнуто середовище на Docker Compose з трьома контейнерами (PostgreSQL, Spring Boot, React/Nginx). Реалізовано бекенд: 10 REST–контролерів, сервіси з бізнес–логікою та репозиторії Spring Data JPA. Особливу увагу приділено BookingService з транзакційним управлінням і перевіркою доступності місць, а також RouteFinderService з алгоритмом Дейкстри. Реалізовано фронтенд як React 19 SPA з

компонентами AdminPanel, AdminResourcePage, UserPanel і ShortestPathPage, стилізованими засобами Tailwind CSS. Проведено функціональне тестування CRUD–операцій та алгоритму пошуку маршрутів за чотирма сценаріями, підтверджено коректність роботи всіх ключових функцій системи.

ВИСНОВКИ

У кваліфікаційній роботі було розглянуто розробку веб–застосунку для керування залізничними рейсами, орієнтованого на підвищення ефективності планування маршрутів, формування розкладу руху поїздів та обслуговування пасажирів в умовах зростання обсягів залізничних перевезень.

Актуальність роботи полягає у необхідності автоматизації процесів керування залізничними перевезеннями, зменшенні впливу людського фактора, підвищенні точності планування та забезпеченні гнучкості системи в умовах зростання пасажиропотоку і ускладнення транспортних мереж. Використання сучасних веб–технологій у поєднанні з алгоритмами оптимізації маршрутів дозволяє створити ефективний інструмент, що відповідає вимогам до сучасних інформаційних систем залізничного транспорту.

У роботі застосовано підхід триланцюгової архітектури, що об'єднує сучасні технології: React 19 з Tailwind CSS на фронтенді, Spring Boot 3.2 з Java 21 на бекенді та PostgreSQL як СУБД, розгорнуті за допомогою Docker Compose. Для розв'язання задачі пошуку оптимальних маршрутів обрано алгоритм Дейкстри, адаптований до специфіки залізничних графів із невід'ємними вагами. Практичне значення одержаних результатів полягає у створенні повнофункціонального веб–застосунку railway–web–app, що забезпечує керування довідниками (міста, станції, поїзди, маршрути, проміжні станції), формування і моніторинг рейсів, бронювання місць пасажирами та побудову оптимальних маршрутів, включаючи маршрути з пересадками.

У першому розділі проведено аналіз предметної області залізничних перевезень та виявлено основні сутності системи: маршрут, рейс, поїзд, станція, місце, пасажир, бронювання і розклад. Виконано огляд існуючих

систем управління залізничними маршрутами, зокрема UZ Online, RailEurope і DB Navigator, та визначено їхні переваги і обмеження з точки зору функціональних можливостей, зручності використання та адаптивності. Здійснено порівняльний аналіз алгоритмів пошуку найкоротших шляхів, за результатами якого обрано алгоритм Дейкстри як найбільш відповідний задачі пошуку оптимального маршруту за тривалістю в графах із невід'ємними вагами. Обґрунтовано вибір технологічного стеку проекту railway–web–app, який забезпечує поєднання продуктивності, масштабованості та зручності розробки.

У другому розділі сформульовано функціональні та нефункціональні вимоги до системи управління залізничними маршрутами і спроектовано триланцюгову архітектуру застосунку: React SPA–фронтенд, Spring Boot REST API бекенд та PostgreSQL база даних, розгорнуті у Docker–контейнерах. Розроблено ER–діаграму бази даних у схемі railway, що включає 10 взаємопов'язаних таблиць. Детально описано структуру пакетів бекенду та призначення кожного шару: entities, dto, mapper, repository, service, controller, що відповідає принципам чистої архітектури та забезпечує розмежування відповідальностей. Розглянуто алгоритм Дейкстри, реалізований у RouteFinderService, та його адаптацію до задачі пошуку залізничних маршрутів з урахуванням часу стоянок на проміжних станціях і можливості формування маршрутів із пересадками. Спроектовано REST API з понад 50 ендпоінтами, CORS–захистом та валідацією вхідних даних.

У третьому розділі описано практичну реалізацію веб–застосунку railway–web–app. Розгорнуто середовище розробки на базі Docker Compose з трьома контейнерами: PostgreSQL, Spring Boot та React/Nginx, що забезпечує відтворюваність середовища запуску. Реалізовано бекенд: 10 REST–контролерів, сервіси з бізнес–логікою та репозиторії Spring Data JPA. Особливу увагу приділено BookingService з транзакційним управлінням і

перевіркою доступності місць, а також RouteFinderService з алгоритмом Дейкстри для побудови оптимальних маршрутів. Реалізовано фронтенд як React 19 SPA з компонентами AdminPanel, AdminResourcePage, UserPanel і ShortestPathPage, стилізованими засобами Tailwind CSS. Проведено функціональне тестування CRUD–операцій адмін–панелі та алгоритму пошуку маршрутів за чотирма сценаріями, що підтвердило коректність роботи всіх ключових функцій системи.

Аналіз результатів показав, що розроблений веб–застосунок railway–web–app успішно розв'язує поставлену задачу підвищення ефективності керування залізничними рейсами. Використання алгоритму Дейкстри забезпечує швидкий та оптимальний пошук маршрутів, а триланцюгова архітектура на базі Spring Boot і React гарантує модульність, зручність супроводу та можливість подальшого масштабування системи. Розроблена система має значні переваги перед розглянутими аналогами завдяки універсальності інтерфейсу, підтримці пошуку маршрутів з пересадками, транзакційному обслуговуванню бронювань і зручному адмініструванню довідників, а розгортання у Docker–контейнерах спрощує впровадження системи у виробничих умовах та забезпечує її готовність до експлуатації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Закон України «Про залізничний транспорт» від 04.07.1996 № 273/96–ВР [Електронний ресурс]. – Режим доступу: URL: https://zakon.rada.gov.ua/laws/show/273/96–вр (дата звернення: 10.04.2026).

2. Іщенко М. І., Польова Л. В. Інформаційні системи на транспорті: навч. посіб. Київ: НТУ, 2020. 248 с.

3. International Union of Railways (UIC). Digital Railway: Strategic Roadmap 2030. Paris: UIC, 2022. 86 p.

4. Evans E. Domain–Driven Design: Tackling Complexity in the Heart of Software. Boston: Addison–Wesley, 2003. 560 p.

5. Date C. J. An Introduction to Database Systems. 8th ed. Boston: Addison–Wesley, 2003. 1024 p.

6. Walls C. Spring Boot in Action. 2nd ed. Shelter Island: Manning Publications, 2022. 320 p.

7. Richardson C. Microservices Patterns: With Examples in Java. Shelter Island: Manning Publications, 2018. 520 p.

8. Deutsche Bahn. DB Navigator – Technology Overview [Електронний ресурс]. – Режим доступу: URL: https://www.bahn.de/service/mobile/db–navigator (дата звернення: 15.10.2025).

9. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison–Wesley, 2002. 560 p.

10. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. 4th ed. Cambridge: MIT Press, 2022. 1312 p.

11. Dijkstra E. W. A note on two problems in connexion with graphs. Numerische Mathematik. 1959. Vol. 1. P. 269–271.

12. Hart P. E., Nilsson N. J., Raphael B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics. 1968. Vol. 4, No. 2. P. 100–107.

13. Bellman R. On a Routing Problem. Quarterly of Applied Mathematics. 1958. Vol. 16, No. 1. P. 87–90.

14. Floyd R. W. Algorithm 97: Shortest Path. Communications of the ACM. 1962. Vol. 5, No. 9. P. 345.

15. Sedgewick R., Wayne K. Algorithms. 4th ed. Boston: Addison–Wesley, 2011. 976 p.

16. Pivotal Software. Spring Boot Reference Documentation 3.2.x [Електронний ресурс]. – Режим доступу: URL: https://docs.spring.io/spring–boot/docs/3.2.0/reference/html/ (дата звернення: 24.10.2025).

17. King G., Bauer C. Java Persistence with Hibernate. 3rd ed. Shelter Island: Manning Publications, 2023. 600 p.

18. PostgreSQL Global Development Group. PostgreSQL 16 Documentation [Електронний ресурс]. – Режим доступу: URL: https://www.postgresql.org/docs/16/ (дата звернення: 14.07.2026).

19. Project Lombok. Lombok Documentation [Електронний ресурс]. – Режим доступу: URL: https://projectlombok.org/features/ (дата звернення: 20.07.2026).

20. MapStruct Authors. MapStruct 1.5 Reference Guide [Електронний ресурс]. – Режим доступу: URL: https://mapstruct.org/documentation/stable/reference/html/ (дата звернення: 19.08.2026).

21. Dodds K., Banks A. Learning React. 2nd ed. Sebastopol: O'Reilly Media, 2020. 350 p.

22. Docker Inc. Docker Compose Documentation [Електронний ресурс]. – Режим доступу: URL: https://docs.docker.com/compose/ (дата звернення: 13.03.2026).

23. Webber J., Parastatidis S., Robinson I. REST in Practice. Sebastopol: O'Reilly Media, 2010. 448 p.

24. Fielding R. T. Architectural Styles and the Design of Network–based Software Architectures: PhD Thesis. UC Irvine, 2000. 162 p.

25. Sommerville I. Software Engineering. 10th ed. Boston: Pearson, 2016. 816 p.

ДОДАТКИ

Додаток А

Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних
систем та робототехніки
Максим ХРУСЛОВ
_____
(підпис)
«13» жовтня 2025року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувача <u>першого (бакалаврського)</u> рівня вищої освіти
(першого (бакалаврського) / другого (магістерського))

<u>ДВОРЕЦЬКОГО ДАНІЇЛА ІВАНОВИЧА</u>
(прізвище, ім'я, по батькові студента)

Спеціальність: 123 - Комп'ютерна інженерія
Освітня програма «Комп'ютерна інженерія»
1. Тема роботи **<u>СИСТЕМА МОНІТОРИНГУ ТА АНАЛІЗУ ДАНИХ ТРАНСПОРТНОЇ ЛОГІСТИКИ НА БАЗІ ВЕБ-ТЕХНОЛОГІЙ</u>**
керівник роботи <u>Чуб Ольга Ігорівна, к.е.н., доцент</u>
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від ***№ 4101-5/1245 від 28.05.2026 року.***
2. Строк подання студентом роботи ***30 травня 2026 року***
3. Вихідні дані до роботи:
   1) сучасні веб-технології для розробки інформаційних систем транспортної логістики;
   2) програмні засоби для створення клієнтської та серверної частини веб-застосунку;
   3) системи керування базами даних для зберігання інформації про рейси, розклад і користувачів;
   4) вимоги до стабільної роботи, безпеки даних та зручності користування;
   5) засоби тестування, моніторингу та аналізу ефективності роботи системи;
   6) персональний комп'ютер для розробки, тестування та розгортання веб-застосунку.
4. Перелік питань, які потрібно розробити:
   1) Аналіз предметної області та існуючих інформаційних систем керування залізничними рейсами, їх функціональних можливостей, переваг і недоліків.
   2) Проєктування архітектури веб-застосунку для керування залізничними рейсами, включно з моделлю даних, ролями користувачів та бізнес-логікою.
   3) Розробка веб-застосунку для керування залізничними рейсами, реалізація основних функцій (управління рейсами, розкладом, станами, користувачами тощо).
   4) Тестування та оцінка працездатності й ефективності розробленого веб-застосунку, перевірка коректності роботи та зручності використання.
   5) Аналіз можливостей подальшого розвитку та масштабування веб-застосунку (інтеграція з іншими системами, аналітика, автоматизація).

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1.	Затвердження теми кваліфікаційної роботи з науковим керівником.	13.10.2025 - 20.10.2025
2.	Пошук та аналіз наукової літератури за темою кваліфікаційної роботи.	21.10.2025 - 02.11.2025
3.	Аналіз існуючих аналогів веб-сервісів керування залізничними рейсами.	04.11.2025 - 19.11.2025
4.	Проєктування архітектури клієнт-серверного веб-застосунку.	22.11.2025 - 13.12.2025
5.	Проєктування структури бази даних та налаштування доступу застосунку до бази даних.	15.12.2025 - 28.12.2025
6.	Розробка серверної частини веб-застосунку (backend).	01.01.2026 - 16.01.2026
7.	Реалізація REST API для забезпечення взаємодії клієнтської та серверної частин.	18.01.2026 - 02.02.2026
8.	Тестування REST API з використанням спеціалізованих інструментів.	04.02.2026 - 17.02.2026
9.	Розробка клієнтської частини веб-застосунку (frontend).	19.02.2026 - 14.03.2026
10.	Реалізація користувацького інтерфейсу веб-застосунку.	16.03.2026 - 03.04.2026
11.	Проведення інтеграційного тестування клієнтської та серверної частин застосунку.	05.04.2026 - 21.04.2026
12.	Аналіз можливостей подальшого розвитку та масштабування веб-застосунку.	24.04.2026 - 03.05.2026
13.	Підготовка та оформлення звітних матеріалів і додатків кваліфікаційної роботи, формування списку використаних джерел.	07.05.2026 - 11.05.2026
14.	Оформлення пояснювальної записки кваліфікаційної роботи відповідно до вимог до звітів про НДР.	13.05.2026 - 20.05.2026
15.	Оформлення звіту про переддипломну практику.	22.05.2026 - 28.05.2026
16.	Представлення кваліфікаційної роботи науковому керівнику та рецензенту.	28.05.2026 - 30.05.2026

5. Дата видачі завдання ***13 жовтня 2025 року.***

Здобувач вищої освіти Д.І. Дворецький _____
ініціали, прізвище підпис

Керівник роботи О.І. Чуб _____

Додаток Б

Затверджую

_____

«_____»_____2026 р.

Технічне завдання

на розробку програмного виробу

«СИСТЕМА МОНІТОРИНГУ ТА АНАЛІЗУ ДАНИХ ТРАНСПОРТНОЇ ЛОГІСТИКИ НА БАЗІ ВЕБ–ТЕХНОЛОГІЙ»

Назва розділу	Назва та зміст підрозділу
1. Вступ	1.1. Назва програмного виробу: «СИСТЕМА МОНІТОРИНГУ ТА АНАЛІЗУ ДАНИХ ТРАНСПОРТНОЇ ЛОГІСТИКИ НА БАЗІ ВЕБ–ТЕХНОЛОГІЙ» 1.2. Галузь застосування: Веб–застосунок для керування залізничними рейсами застосовується в сфері транспортної логістики та управління перевезеннями. Він може використовуватися для моніторингу руху поїздів у режимі реального часу, планування та оптимізації розкладу, управління маршрутами, а також контролю завантаженості та своєчасності рейсів. Крім того, система дозволяє оперативно реагувати на затримки, зміну умов руху або інші відхилення, що сприяє підвищенню ефективності перевезень і покращенню якості обслуговування пасажирів. Такі веб–застосунки активно впроваджуються в залізничній галузі для автоматизації процесів, зменшення людського фактору та забезпечення надійності транспортної інфраструктури.
2. Підстава для розробки	2.1. Навчальний план за спеціальністю: 123 – Комп'ютерна інженерія, Харківський національний університет імені В. Н. Каразіна, Навчально–науковий інститут комп'ютерних наук та штучного інтелекту. 2.2. Завдання на кваліфікаційну роботу: №4101–5/1245___ від 28.05.2026р. ___ Розробка веб–застосунку для керування залізничними рейсами.

3. Призначення розробки	3.1. Мета розробки програмного виробу: Метою дослідження є підвищення ефективності керування залізничними перевезеннями за допомогою розробки веб–застосунку з використанням методів оптимізації маршрутів. Для досягнення сформульованої мети були виділені такі завдання дослідження: - виконати аналіз предметної області та існуючих інформаційних систем керування залізничними рейсами, їх функціональних можливостей, переваг і недоліків; - спроєктувати архітектуру веб–застосунку, включаючи модель даних, ролі користувачів та бізнес–логіку; - розробити веб–застосунок для керування залізничними рейсами з реалізацією основних функцій (управління рейсами, розкладом, станами та користувачами); - провести тестування та оцінку ефективності роботи застосунку, перевірити коректність функціонування та зручність використання; - проаналізувати можливості подальшого розвитку та масштабування системи, зокрема інтеграцію з іншими сервісами та впровадження аналітичних інструментів. 3.2. Призначення програмного виробу: Цей програмний виріб призначений для керування залізничними рейсами з метою підвищення ефективності перевезень, оптимізації розкладу руху поїздів та покращення якості обслуговування пасажирів. 3.3. Вихідні дані для розробки : - Використання сучасних веб–технологій та фреймворків (наприклад, React, Angular або Vue для фронтенду та Spring Boot або Node.js для бекенду) з метою створення ефективного та масштабованого веб–застосунку. - Для функціонування системи використовується база даних, що містить інформацію про залізничні рейси, маршрути, розклад руху

	поїздів, станції, а також дані про затримки та зміни у графіку руху.
4. Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: - Моніторинг рейсів: Автоматичне отримання даних (телеметрії) про стан залізничних рейсів у режимі реального часу. - Інтелектуальний аналіз: Виявлення дефектів полотна або відхилень у графіку руху за допомогою попередньо навченої моделі нейронної мережі на стороні сервера. - Інтерактивна візуалізація: Відображення стану рейсу на цифровій карті з підсвічуванням ділянок, де виявлено критичні аномалії чи дефекти. - Централізоване сховище: Збереження результатів аналізу, логів руху та звітів про дефекти у масштабованій хмарній базі даних. - Продуктивність: Час обробки та візуалізації стану одного рейсу на сервері не повинен перевищувати 2 секунд при одночасній роботі до 50 користувачів. 4.2. Вимоги до надійності: - Стійкість моделей: Алгоритми обробки повинні коректно інтерпретувати дані за різних погодних умов (освітленість, опади) та стану сенсорів. - Валідація даних: Система повинна здійснювати перевірку цілісності та формату вхідних даних від датчиків. При отриманні некоректних пакетів – генерується запис про помилку, а кадр (дані) відкидається без зупинки системи. - Безперервність: Застосунок повинен забезпечувати цілодобову роботу (24/7) з автоматичним перезапуском сервісів у разі критичних помилок. - Цілісність БД: Забезпечення атомарності операцій запису для запобігання втраті даних про стан рейсів під час технічних збоїв. 4.3. Вимоги до умов експлуатації: - Мережевий доступ: Веб–застосунок розгортається на сервері та доступний

	користувачам через сучасні веб–браузери (Chrome, Firefox, Edge) без необхідності встановлення додаткового ПЗ на клієнтські машини. - Інтерфейс: Користувач отримує доступ до панелі керування через веб–інтерфейс, що забезпечує відображення актуального стану мережі залізничних рейсів. - Інфраструктура: Застосунок має бути адаптований для роботи в корпоративній мережі залізниці з підключенням до захищеного сервера. - Документація: Наявність вбудованої довідкової системи для адміністраторів та операторів диспетчерської служби. 4.4. Вимоги до складу параметрів технічних засобів: - Серверна частина: Програмний виріб розгортається на сервері під керуванням ОС Windows Server або Linux (наприклад, Ubuntu 20.04+). Необхідна наявність встановленого середовища виконання Java Runtime Environment (JRE) версії 17 або вище. - Мінімальні апаратні вимоги: * Процесор із тактовою частотою від 2.5 ГГц (2 або 4 ядра); 8 ГБ оперативної пам'яті (RAM); 5 ГБ вільного місця на диску для стабільної роботи бази даних та логів. - Графічне прискорення: Завдяки оптимізованій архітектурі, обробка зображень може здійснюватися силами центрального процесора (CPU). Використання дискретної відеокарти не є обов'язковим, що дозволяє запускати систему на стандартному офісному обладнанні або недорогих хмарних інстансах. 4.5. Вимоги до інформаційної та програмної сумісності - Стек технологій: Бекенд–частина реалізована на мові Java (з використанням фреймворку Spring Boot), що забезпечує високу стабільність та кросплатформеність. Фронтенд адаптований під роботу у будь–яких сучасних браузерах (Chrome, Firefox, Safari). - Вхідні та вихідні дані: у форматі JSON.

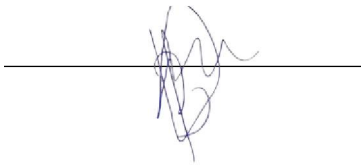
	- Інтеграція: Завдяки використанню Java, застосунок легко інтегрується з іншими корпоративними системами через стандартні REST–інтерфейси. - Захист даних: Оскільки продукт призначено для локального навчального та дослідницького використання, спеціальні механізми шифрування та захисту даних не реалізовані (використовуються базові налаштування безпеки веб–сервера). 4.6. Вимоги до маркування та упаковки Вимоги до маркування та упаковки не висуваються, оскільки програмний продукт є цифровим рішенням і не потребує фізичної упаковки для використання в навчальних, дослідницьких або виробничих умовах. 4.7. Вимоги до транспортування та зберігання Вимоги до транспортування та зберігання відсутні, оскільки продукт є програмним забезпеченням, яке може бути передане в електронному вигляді та не потребує спеціальних умов зберігання. 4.8. Спеціальні вимоги Спеціальні вимоги до програмного продукту не передбачені. Система розроблена як універсальне рішення і не вимагає додаткового апаратного або програмного забезпечення, окрім стандартних компонентів, зазначених у вимогах до експлуатації.
5. Вимоги до програмної документації.	Склад програмної документації до виробу «Веб–застосунок для керування залізничними рейсами»: 1. Це Технічне завдання на розробку програмного виробу (представити у вигляді Додатка Б до пояснювальної записки до кваліфікаційної роботи). 2. Програму та методику випробувань розробленого програмного виробу (представити у вигляді Додатка В до пояснювальної записки до кваліфікаційної роботи). 3. Опис програмного виробу (представити в розділі пояснювальної записки до кваліфікаційної роботи). 4. Текст програми (представити в Додатку Г до пояснювальної записки до кваліфікаційної роботи).
6. Техніко–економічні показники	1. Планується досягти високої точності обробки та актуальності даних про залізничні рейси (на рівні 95–99%), що забезпечить своєчасне оновлення розкладу,

	коректне відображення змін у русі поїздів та зменшення кількості помилок при управлінні перевезеннями. 2. Розробка веб–застосунку триватиме орієнтовно 2–4 місяці. Для реалізації будуть використовуватися переважно безкоштовні або відкриті програмні засоби. Основні витрати можуть бути пов'язані з розгортанням серверної інфраструктури, хостингом, підтримкою бази даних та забезпеченням стабільної роботи системи при високому навантаженні. 3. Очікується, що розроблений веб–застосунок підвищить ефективність управління залізничними перевезеннями, зменшить вплив людського фактору та оптимізує процес планування рейсів. Система буде легко масштабуватися, інтегруватися з існуючими інформаційними системами та може бути використана як у реальних транспортних компаніях, так і в навчальних або дослідницьких цілях, що забезпечує її економічну доцільність і практичну цінність.
7. Стадії та етапи розробки	Розробка моделі класифікації архітектурних споруд проходить у три основні стадії відповідно до ДЕСТ 19.102–77: Стадії: 1. Розробка технічного завдання– формування вимог, постановка задачі, погодження з керівником. 2. Робоче проектування– включає програмування, налагодження, розробку документації та випробування. 3. Впровадження розробленого виробу– підготовка, передача програми та документації до експлуатації. Етапи: 1. Розробка веб–застосунку (проєктування архітектури системи, розробка інтерфейсу користувача, реалізація серверної логіки, створення та налаштування бази даних, інтеграція модулів для роботи з розкладом, маршрутами та моніторингом рейсів, тестування та налагодження системи). 2. Розробка програмної документації (створення пояснювальної записки, опис функціональних можливостей системи, підготовка інструкцій користувача та адміністратора, візуалізація роботи веб–застосунку).

	3. Впровадження (випробування) програми (розгортання веб–застосунку на сервері, тестування в реальних або наближених до реальних умовах, передача системи для практичного або навчального використання).
8. Порядок контролю та приймання	1. Перевірка ходу розроблення заданих проектних документів керівнику кваліфікаційної роботи здійснювати 1 раз на 3 тижні. 2. Випробування програмного виробу. 3. Захист розробленого програмного виробу провести на засіданні ЕК. 4. Розроблені проектні документи подати в електронному вигляді.

Виконавець

Дворецький Даніїл Іванович



Замовник

Чуб Ольга Ігорівна



Додаток В

Програма та методика випробувань програмного виробу «СИСТЕМА МОНІТОРИНГУ ТА АНАЛІЗУ ДАНИХ ТРАНСПОРТНОЇ ЛОГІСТИКИ НА БАЗІ ВЕБ–ТЕХНОЛОГІЙ»

1. Об'єкт випробувань

1.1 Найменування програмного виробу:

Веб–застосунок для керування залізничними рейсами.

1.2 Область застосування:

Веб–застосунок для керування залізничними рейсами застосовується у сфері транспортної логістики та управління залізничними перевезеннями. Він може використовуватися для моніторингу руху поїздів у режимі реального часу, формування та оптимізації розкладу, управління маршрутами, а також контролю своєчасності виконання рейсів. Крім того, система дозволяє оперативно реагувати на затримки, зміни у графіку руху чи інші позаштатні ситуації, що сприяє підвищенню ефективності роботи транспортної інфраструктури.

Такі веб–застосунки активно впроваджуються у залізничних компаніях, диспетчерських центрах, логістичних організаціях, а також можуть використовуватися в навчальних і дослідницьких цілях, де важливими є точність планування, швидкість обробки інформації та зменшення впливу людського фактора.

1.3 Умовне призначення розробки (за потреби).

Цей програмний виріб призначений для автоматизації процесів керування залізничними рейсами, включаючи планування розкладу руху, моніторинг поїздів та управління маршрутами. Він розроблений для використання в умовах транспортної інфраструктури, зокрема в залізничних

компаніях та диспетчерських службах, а також може бути застосований у навчальних та дослідницьких цілях у сфері інформаційних технологій і транспортної логістики.

2. Мета випробувань

Метою випробувань є перевірка відповідності функціональних характеристик розробленого веб–застосунку вимогам, зазначеним у Технічному завданні. Зокрема, це підтвердження здатності системи ефективно здійснювати керування залізничними рейсами, включаючи формування та оновлення розкладу руху, моніторинг поїздів у режимі реального часу, а також обробку змін і відхилень у графіку.

Випробування мають також на меті оцінку ефективності використання інформаційних технологій для підвищення швидкості обробки даних, зменшення часу реагування на затримки чи позаштатні ситуації та забезпечення стабільної роботи системи при різних навантаженнях. Це дозволяє зменшити вплив людського фактору, підвищити точність планування перевезень і забезпечити надійний рівень управління для подальшого аналізу та оптимізації роботи залізничної інфраструктури.

3. Загальні положення

3.1 Підстави щодо випробувань

Підставою для проведення випробувань є: наказ ХНУ №4101–5/1245 *від* 28.05.2026р. *про затвердження тем кваліфікаційних робіт бакалаврів.*

3.2 Місце та тривалість випробувань

Приймальні випробування проводяться на базі комп'ютерного класу кафедри.

3.3. Обсяг випробувань

Приймальні випробування програмного виробу проводяться в обсязі відповідно до цієї Програми та методики випробувань.

3.4 Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться за участю Замовника та Виконавця..

4. Вимоги до програмного виробу

- Програмний виріб повинен забезпечувати автоматизоване керування залізничними рейсами, включаючи створення, редагування та відображення розкладу руху поїздів.
- Програмний виріб має підтримувати обробку та зберігання даних про рейси, маршрути, станції та зміни у графіку руху.
- Програмний виріб повинен мати можливість відображення інформації у веб–інтерфейсі, а також забезпечувати збереження даних у базі даних для подальшого аналізу чи архівації.
- Користувач повинен мати можливість взаємодіяти з системою через веб–інтерфейс: переглядати розклад, додавати нові рейси, редагувати існуючі записи та відстежувати статус поїздів.

5. Вимоги до програмної документації

Програмна документація до виробу "Веб–застосунок для керування залізничними рейсами":

1. Це Технічне завдання на розробку програмного виробу (представити у вигляді Додатка Б до пояснювальної записки до кваліфікаційної роботи).

2. Програму та методику випробувань розробленого веб–застосунку (представити у вигляді Додатка В до пояснювальної записки до кваліфікаційної роботи).

3. Опис програмного виробу (представити в розділі пояснювальної записки до кваліфікаційної роботи).

6. Засоби та порядок випробувань

6.1 Засоби випробувань

Для проведення випробувань буде використовуватись наступне технічне та програмне забезпечення:

Технічні засоби:

- Сервери або робочі станції з достатньою кількістю оперативної пам’яті та обчислювальних ресурсів для забезпечення стабільної роботи веб–застосунку.
- Клієнтські пристрої (персональні комп’ютери, ноутбуки) для доступу до веб–інтерфейсу.
- Мережеве обладнання для забезпечення стабільного з’єднання та передачі даних.

Програмні засоби:

- JavaScript – для розробки клієнтської частини веб–застосунку.
- React – фреймворки для створення інтерфейсу користувача.
- Spring Boot – для реалізації серверної частини.
- СУБД (PostgreSQL) – для зберігання даних про рейси, маршрути та розклад.
- Інструменти тестування (Postman, Swagger) – для перевірки API та взаємодії між компонентами системи.
- Браузери (Google Chrome, Mozilla Firefox) – для тестування веб–інтерфейсу.

Та інші.

6.2 Порядок проведення випробувань

1–й підетап. Перевірка комплектності програмної документації

Перевіряється наявність усіх необхідних документів згідно з Технічним завданням. Перевірка комплектності технічних та програмних засобів здійснюється за критерієм відповідності їх складу вимогам середовища виконання.

2–й підетап. Перевірка якості програмної документації

Якість програмної документації відповідає вимогам ДЕСТ 19.301–79 ЄСПД.

3–й етап – функціональне випробування програмного виробу

На цьому етапі перевіряється працездатність веб–застосунку та ступінь виконання функціональних вимог: коректність відображення розкладу, можливість керування рейсами, обробка змін у реальному часі та стабільність роботи системи при різних навантаженнях.

1. Перевірка працездатності програмного виробу

Методика проведення Тесту 1:

- Запуск веб–застосунку здійснюється у Docker контейнери.
- Перевірка основного функціоналу веб–застосунку щодо відображення та керування залізничними рейсами. Користувач входить у систему через веб–інтерфейс, переглядає наявний розклад руху поїздів, додає новий рейс із зазначенням маршруту, часу відправлення та прибуття, після чого зберігає зміни. Далі здійснюється перевірка коректності відображення доданого рейсу у загальному списку та його збереження в базі даних.

Очікуваний результат: новий рейс успішно додається до системи, коректно відображається у розкладі, всі введені дані зберігаються без помилок, а інтерфейс користувача оновлюється без затримок.

Фактичний результат: новий рейс було успішно додано до системи, дані збережено у базі даних, відображення у розкладі

відповідає введеним параметрам, помилок у роботі застосунку не виявлено.

Висновок: у результаті виконання Тесту 1 функціональна вимога щодо керування залізничними рейсами виконана.

Методика проведення Тесту 2:

- Запуск тестування моделі здійснюється у середовищі PyCharm.
- Перевірка функціоналу оновлення та редагування інформації про залізничний рейс. Користувач відкриває існуючий рейс у веб–застосунку, змінює параметри (час відправлення, маршрут або статус рейсу), після чого зберігає зміни. Далі виконується перевірка коректності оновлення даних у загальному розкладі та їх збереження у базі даних без втрати інформації.

Очікуваний результат:
інформація про рейс успішно оновлюється, зміни коректно відображаються у розкладі руху поїздів, усі оновлені дані зберігаються в базі даних, інтерфейс працює стабільно без помилок.

Фактичний результат: інформацію про рейс було успішно змінено та збережено, оновлені дані коректно відображаються у веб–інтерфейсі, помилок при виконанні операції не виявлено.

Висновок: у результаті виконання Тесту 2 функціональна вимога щодо редагування та оновлення даних рейсів виконана. Також підтверджено стабільність роботи системи при внесенні змін до існуючих записів у розкладі.

Висновок за результатами проведених випробувань

Програмний виріб вважається таким, що пройшов випробування в цілому, якщо успішно виконані всі Тести 1 і 2. Загалом, тестування показало, що веб–застосунок працює коректно та забезпечує ефективне керування залізничними рейсами, включаючи додавання, редагування та відображення розкладу руху поїздів. Це дозволяє зробити висновок про його готовність до використання в реальних умовах або для подальшого доопрацювання.

Рекомендується проводити подальше тестування для оцінки роботи системи при підвищеному навантаженні, інтеграції з іншими інформаційними системами та розширенні функціональних можливостей.

Виконавець: Дворецький Даніїл Іванович 