Міністерство освіти і науки України

Криворізький національний університет

Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Розробка системи з алгоритмом адаптивного формування тарифів пасажирських перевезень в інтелектуальній транспортній системі міста

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ-20-2

_____/Г. О. Лушов /

Керівник кваліфікаційної роботи _____ / Н. Н. Шаповалова /

Завідувач кафедри _____ / А. М. Стрюк /

Кривий Ріг 2024

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: бакалавр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А.М.Стрюк

«__» _____ 20__р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-20-2 Лушов Гліб Олександрович

1. Тема: Розробка системи з алгоритмом адаптивного формування тарифів пасажирських перевезень в інтелектуальній транспортній системі міста. Затверджено наказом по КНУ №__ від «__» _____ 2024р.
2. Термін подання студентом закінченої роботи : «__» _____ 2024р.
3. Вихідні дані по роботі: вихідні данні повинні виводити адаптивні тарифи автобусних перевозок.
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): проаналізувати існуючі системи формування тарифів, розглянути технології що можна використати для розробки, розробити програмне забезпечення для реалізації системи.
5. Перелік ілюстративного матеріалу: блок-схеми алгоритмів, знімки екрану з відображенням інтерфейсу, графи нейронних мереж, схеми взаємодії програмних модулів.

РЕФЕРАТ

АЛГОРИТМ АДАПТИВНОГО ФОРМУВАННЯ ТАРИФІВ, МАШИНЕ НАВЧАННЯ, ІНТЕЛЕКТУАЛЬНА СИСТЕМА МІСТА.

Пояснювальна записка: 66 с., 5 табл., 15 рис., 1 дод., 4 джерел.

За останні роки інтелектуальні транспортні системи (ІТС) стали важливим елементом сучасних міст, пропонуючи ефективні рішення для управління рухом та забезпечення комфортності для пасажирів. Однією з ключових складових таких систем є алгоритми машинного навчання, які дозволяють ефективно управляти транспортним навантаженням та забезпечувати сталість доходів.

Метою кваліфікаційної роботи є дослідження інтелектуальних транспортних систем з використанням алгоритмів машинного навчання. У ході дослідження буде проведено аналіз існуючих систем управління рухом та їх ефективності з метою вдосконалення. Також буде розроблено та впроваджено новий підхід до управління транспортним навантаженням, що базується на алгоритмах машинного навчання.

Практичне значення розробленої системи полягатиме в підвищенні ефективності управління транспортними потоками, зменшенні часу очікування для пасажирів та підвищенні загальної комфортності пересування у місті. Результати дослідження та впровадження нової системи можуть бути корисними для міських влад та транспортних компаній у вдосконаленні існуючих систем управління рухом та плануванні маршрутів.

Дослідження інтелектуальних транспортних систем є актуальним завданням, оскільки вони можуть допомогти вирішити проблеми перевантаження доріг, заторів та забруднення навколишнього середовища. Використання алгоритмів машинного навчання дозволяє створити системи, які адаптивно реагують на зміни в транспортних потоках та швидко оптимізують рух транспорту.

ABSTRACT

ALGORITHM OF ADAPTIVE TARIFF FORMATION, MACHINE LEARNING, INTELLIGENT CITY SYSTEM.

Explanatory note: 30 p., 5 tables, 15 images, 1 appendix, 5 sources.

In recent years, intelligent transport systems (ITS) have become an important element of modern cities, offering effective solutions for traffic management and passenger comfort. One of the key components of such systems is machine learning algorithms, which allow for efficient traffic management and ensure revenue sustainability.

The purpose of the qualification work is to study intelligent transport systems using machine learning algorithms.

The study will analyze existing traffic management systems and their effectiveness in order to improve them. A new approach to traffic management based on machine learning algorithms will also be developed and implemented.

The practical significance of the developed system will be to improve the efficiency of traffic management, reduce waiting time for passengers and increase the overall comfort of movement in the city. The results of the research and implementation of the new system can be useful for city authorities and transport companies in improving existing traffic management systems and route planning.

Researching intelligent transport systems is a pressing task, as they can help solve the problems of road congestion, traffic jams and environmental pollution. The use of machine learning algorithms allows us to create systems that adaptively respond to changes in traffic flows and quickly optimise traffic.

ЗМІСТ

Умовні позначення, скорочення

МН – Машинне навчання

ІТС - інтелектуальні транспортні системи

НМ – нейромережа

IDE - інтегроване середовище розробки

ВСТУП

За останні роки інтелектуальні транспортні системи (ІТС) стрімко набирають обертів, перетворюючись на невід'ємну частину сучасних міст. Ці комплексні системи, що об'єднують інформаційні технології, телекомунікації та транспортну інфраструктуру, пропонують широкий спектр інноваційних рішень для оптимізації руху, підвищення безпеки та комфорту пасажирів.

В основі ІТС лежать алгоритми машинного навчання, які здатні аналізувати величезні обсяги даних в режимі реального часу. Цей інтелект дозволяє системам гнучко реагувати на динамічні умови на дорогах, прогнозувати затори та пропонувати альтернативні маршрути, економячи час та ресурси водіїв та пасажирів.

Окрім покращення мобільності, ІТС роблять значний внесок у розвиток сталих міст. Завдяки оптимізації маршрутів та скороченню простою транспортних засобів, знижується рівень викидів парникових газів та забруднення повітря.

Також функцією ІТС є оптимізація маршрутів та графіків руху транспорту з метою мінімізації часу очікування для пасажирів. Алгоритми машинного навчання аналізують дані про транспортний потік, попередньо прогнозують його інтенсивність та виробляють оптимальні маршрути, що дозволяє пасажирам ефективно планувати свої поїздки та зменшує загальний час витрачений на переміщення.

Важливо також відзначити роль ІТС у забезпеченні стійкості доходів транспортних підприємств. Завдяки точному прогнозуванню пасажиропотоку та динамічному ціноутворенню, транспортні компанії можуть ефективніше планувати свою роботу, maximizуючи заповнення та рентабельність.

Впровадження ІТС відкриває нові можливості для розвитку транспортних систем майбутнього. Ці системи здатні не лише покращити життя мешканців міст, але й зробити значний внесок у вирішення глобальних проблем, таких як зміна клімату та урбанізація.

1 СУЧАСНИЙ СТАН І АКТУАЛЬНІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1 Актуальність теми кваліфікаційної роботи

Актуальність теми можливо обумовити низкою факторів:

- у сучасних містах зростає проблема мобільності через збільшення населення та урбанізацію, що призводить до перевантаження транспортних систем, заторів і забруднення навколишнього середовища. Традиційні методи управління рухом не в змозі ефективно реагувати на ці динамічні зміни;
- використання ІТС з алгоритмами машинного навчання може допомогти вирішити ці проблеми, адаптивно реагуючи на транспортні потоки та оптимізуючи рух. Це сприятиме зменшенню заторів, заощадженню часу та ресурсів, а також покращенню екологічної ситуації;
- потрібно вдосконалити системи формування тарифів, оскільки існуючі не завжди враховують реальний попит на транспорт та динаміку транспортних потоків, що може призвести до незручностей для пасажирів і нерентабельності транспортних підприємств;
- розвиток технологій машинного навчання дозволяє аналізувати великі обсяги даних та прогнозувати транспортний попит, що відкриває можливості для створення адаптивних систем формування тарифів, більш ефективних і зручних для користувачів;
- впровадження ІТС з адаптивними тарифами може мати значний економічний і соціальний вплив, сприяючи розвитку громадського транспорту, покращенню якості життя і зроблячи міста більш комфортними для проживання.

1.2 Цілі та завдання кваліфікаційної роботи

Ціллю кваліфікаційної роботи є розробка та впровадження системи з алгоритмом адаптивного формування тарифів. Ця система має на меті покращення мобільності мешканців міста шляхом оптимізації транспортного потоку, зменшення заторів, економії часу та ресурсів, а також покращення якості життя в місті.

Завданнями роботи є:

Дослідження потреб користувачів та визначення ключових факторів – перш за все, необхідно зрозуміти, які потреби мають користувачі транспортної системи та які основні фактори впливають на формування тарифів. Це може включати аналіз часу доби, днів тижня, сезонних коливань, спеціальних подій у місті та інших умов, які можуть впливати на попит.

Аналіз даних про пасажиропотік – важливо зібрати та проаналізувати дані про пасажиропотік, щоб виявити тенденції та закономірності у використанні транспортної системи. Це допоможе визначити, як попит змінюється в різний час та в різних умовах.

Розробка математичної моделі для прогнозування попиту – на основі зібраних даних потрібно створити математичну модель, яка дозволить прогнозувати попит на пасажирські перевезення. Ця модель буде використовуватися для адаптації тарифів відповідно до змін у попиті.

Програмування алгоритму адаптивного формування тарифів – полягає у розробці алгоритму, який може автоматично коригувати тарифи на основі даних, отриманих від математичної моделі. Алгоритм повинен бути інтегрований з інтелектуальною транспортною системою міста.

Тестування та валідація алгоритму – після розробки алгоритму необхідно провести його тестування, щоб переконатися у його ефективності та точності. Тестування може включати використання історичних даних та проведення пілотного проекту у реальних умовах.

1.3 Критичний аналіз літературних джерел за темою

Після аналізу великої кількості літературних джерел, зокрема наукових робіт і статей, я зробив кілька ключових висновків.

Можливо сказати, що ІТС - це передові програми, спрямовані на надання інноваційних послуг, пов'язаних з різними видами транспорту та управлінням дорожнім рухом. Вони дозволяють користувачам бути краще поінформованими та безпечніше, більш скоординовано і "розумніше" використовувати транспортні мережі. Ці системи інтегрують інформаційні та комунікаційні технології у сфері автомобільного транспорту, включаючи інфраструктуру, транспортні засоби та користувачів, а також управління дорожнім рухом та управління мобільністю [1,2].

Також було виявлено, що застосування інтелектуальних транспортних систем (ІТС) та методів машинного навчання може бути обіцяним напрямом для вирішення проблеми формування тарифів. ІТС можуть адаптивно реагувати на зміни в транспортних потоках та враховувати різні фактори, що впливають на ціни на перевезення [3].

Стратегія розвитку ІТС може бути зосереджена на інтеграції передових технологій, включаючи штучний інтелект (ШІ) та автомобільний зв'язок. Ця стратегія може передбачати створення більш ефективної, стійкої та взаємопов'язаної транспортної екосистеми. Вона також може передбачати покращення досвіду користувачів транспорту та результатів сталого розвитку [3].

Крім того, варто відзначити, що реалізація таких систем може потребувати значних зусиль і ресурсів на етапі впровадження. Це означає не лише фінансові витрати, але й мобілізацію кваліфікованих кадрів для розробки, тестування та налагодження системи [4].

Але, необхідно враховувати, що впровадження інтелектуальних систем потребує не лише технологічних, але й правових та організаційних змін, адже вони можуть впливати на існуючі правила та стандарти у галузі пасажирських перевезень [4].

2 РОЗРОБКА ФУНКЦІОНАЛЬНОЇ СХЕМИ І АЛГОРИТМУ

2.1 Розробка та докладний опис функціональної схеми програми

Структурна діаграма - це графічне зображення структури системи або програмного продукту, яке показує компоненти системи та їх взаємозв'язки. Ця діаграма допомагає візуалізувати організацію програмного забезпечення, його модулі та класи, а також співвідношення між ними.

Наступна діаграма сприяє у розумінні структури програми, визначенні послідовності виконання операцій та осмисленні взаємодії між її складовими.

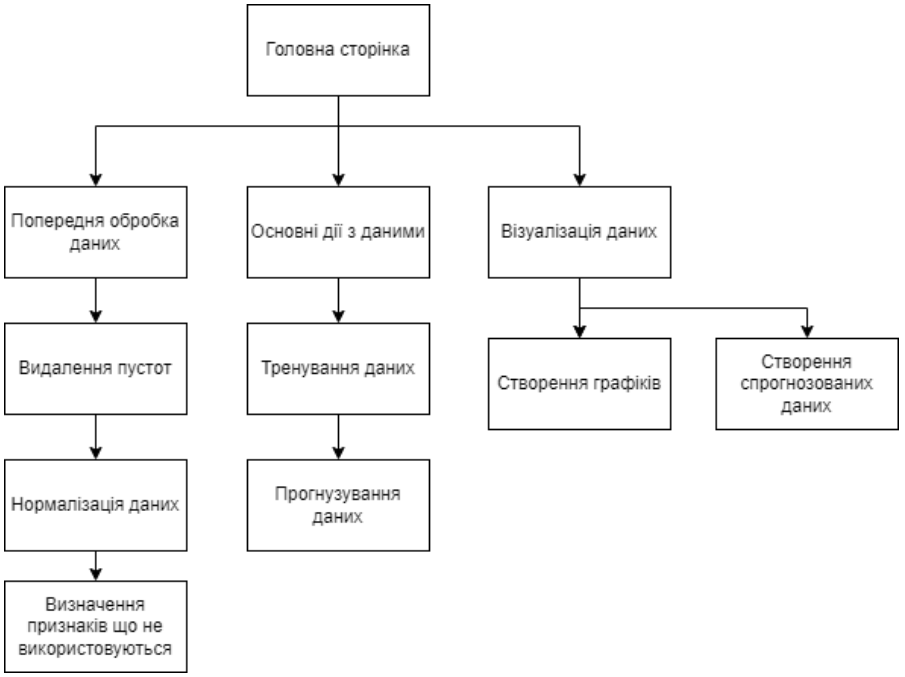

Рисунок 2.1 – Структурна діаграма

Тому на рисунку 2.1 показана структурна діаграма, що описує функціонал програми:

1. Попередня обробка даних. Це включає в себе очищення даних, нормалізацію даних, видалення пустот, визначення признаків що не використовуються.
2. Основні дані. Включає аналіз даних для виявлення закономірностей та тенденцій. Це включає в себе:

- Тренування даних. навчання моделі машинного навчання на даних.
- Прогнозування даних. використання моделі машинного навчання для прогнозування нових значень даних.

3. Візуалізація даних.
   - Створення графіків. Візуалізація закономірностей та тенденцій в даних.
   - Створення таблиць. Візуалізація даних у табличному форматі.

Діаграма потоків даних (DFD) - це графічний засіб для представлення потоків даних у системі. Вона відображає рух даних в системі та взаємозв'язки між різними частинами цієї системи.

На рисунку 2.2 показана діаграма потоків даних, де описано взаємодію системи з користувачем та сервером. Користувач передає свої дані в систему. Система обробляє дані, генерує прогнозовані дані та візуалізує їх.

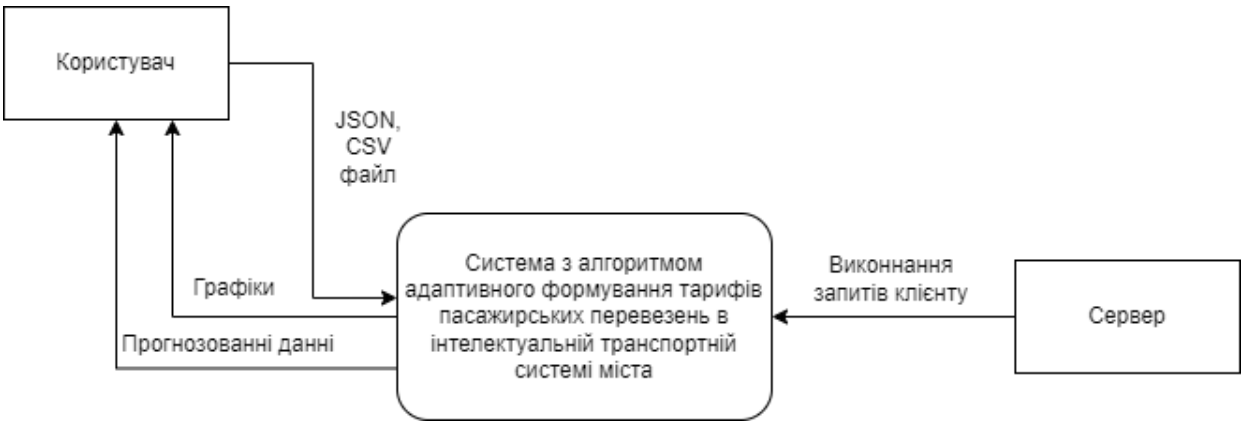

Рисунок 2.2 – Контекстна діаграма потоків даних

Загальна діаграма повністю ілюструє процес передачі даних у системі, що представлений на рисунку 2.2. Ця діаграма відображає всі етапи взаємодії між користувачем та системою, включаючи збір та введення даних, їх обробку та аналіз, а також вивід результатів користувачеві.

На рисунку 2.3 показана діаграма потоків даних. Де:

1. Файл з даними. Система використовує файл з даними що зберігається

на сервері.

2. Вхід на сайт. Користувач входить на сайт та переглядає можливі дії
3. Вибір взаємодії. Користувач вибирає можливу дію на клієнтській частині сайту
4. Виконання функції. Функція виконується на сервері обробляючи данні користувача отримані з клієнту
5. Формування результатів. Система формує результат та відсилає його клієнту
6. Результати. Дані виводяться на клієнті та показуються користувачу

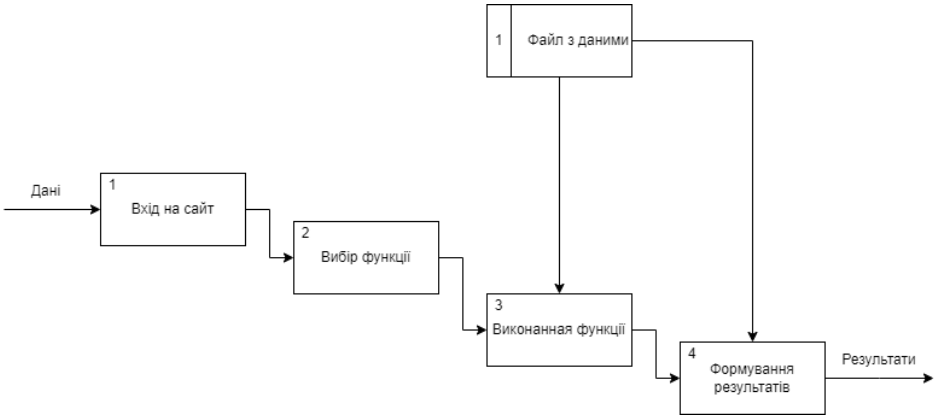

Рисунок 2.3 – Загальна діаграма потоків даних

На наступній діаграмі розглянуто більш детально пункт 3 з попередньої діаграми на рисунку 2.3. Тут можуть відбуватися різноманітні операції, такі як обробка, передбачення, тренування або інші дії з даними, залежно від вимог системи.

На рисунку 2.4 зображена діаграма що описує функцію яка виконує обробку даних та складається з наступних кроків:

- Перевірка даних. Перевірка даних на відповідність їх структури
- Обробка даних. Різні дії з даними наприклад їх передбачення, тренування
- Повернення помилки. Повернення помилки та подальше відображення її на стороні клієнту

Рисунок 2.4 – Детальна діаграма потоків даних

Функціональна діаграма є інструментом для візуалізації функціональності системи або програмного продукту, а також для встановлення зв'язків між цими функціями. Ця діаграма допомагає розуміти, як система працює та які функції вона виконує.

На рисунку 2.5 подана загальна діаграма, що ілюструє обробку та аналіз даних у системі. Вхідні дані можуть бути у форматі JSON або CSV файлу. Система обробляє ці дані та надає користувачеві різноманітні вихідні результати, такі як розклад автобусів, передбачення для розкладу автобусів та графіки.

Перетворюючи вхідні дані у корисний вихідний результат, система забезпечує зручність та користь для користувачів, дозволяючи їм ефективно використовувати надану інформацію.

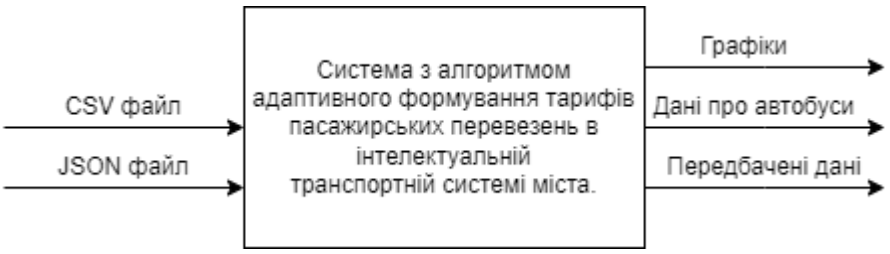

Рисунок 2.5 – Загальна функціональна діаграма

Діаграма 2.6 показує функціональну модель тренувальної програми. Вона описує основні етапи процесу тренування, починаючи з підготовки даних і

закінчуючи візуалізацією результатів.

На рисунку 2.6 зображено

- Попередня обробка даних: На цьому етапі дані готуються до подальшого аналізу моделі. Це може включати такі завдання, як видалення пустот, нормалізація даних, визначення та видалення непотрібних ознак.
- Внесення даних: Підготовлені дані завантажуються в систему.
- Дії з даними: Дані можуть бути трансформовані або очищені перед використанням для тренування моделі.
- Тренування даних: На цьому етапі модель машинного навчання тренується на підготовлених даних. Модель вчиться розпізнавати закономірності в даних і робити прогнози або приймати рішення.
- Передбачення даних: Натренована модель використовується для прогнозування нових даних.
- Візуалізація даних: Результати тренування моделі можуть бути візуалізовані за допомогою таблиць або графіків.

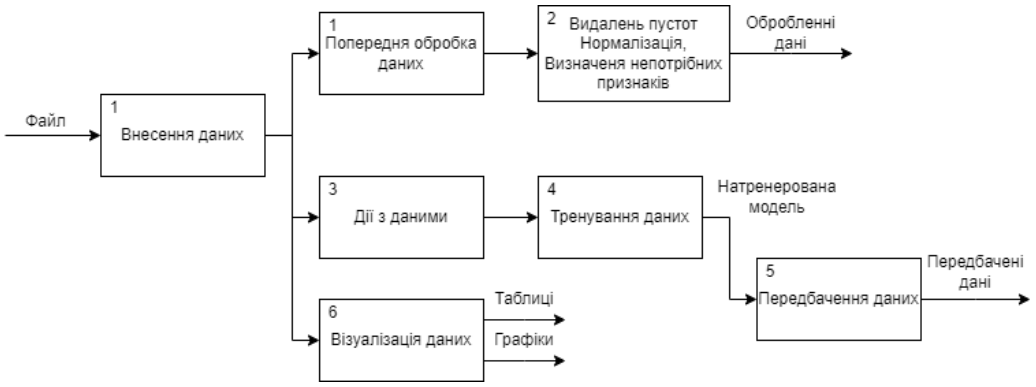

Рисунок 2.6 – Функціональна діаграма

2.2 Розробка та докладний опис роботи програми

2.2.1 Опис вимог до системи

Функціональні вимоги визначають конкретні функції, які система повинна виконувати та яким чином вона має взаємодіяти з користувачем та іншими системами.

Ці вимоги можуть описувати функції в системі, такі як обробка даних, взаємодія з базою даних, генерація звітів, а також способи взаємодії з користувачем через інтерфейс користувача або інші зовнішні системи.

Таблиця 2.1 – Використані технології

№	Вимога
1	Система повинна отримувати дані про пасажирські перевезення в місті, що може включаючи час подорожі, маршрути, кількість пасажирів тощо. Дані повинні аналізуватися для виявлення патернів та тенденцій.
2	Система повинна використовувати алгоритм чи нейромережу для адаптивного формування тарифів на основі зібраних та проаналізованих даних. Тарифи повинні змінюватися в залежності від попиту, часу доби тощо.
3	Система має обчислювати вартість поїздки для пасажирів на основі залежності від часу подорожі та актуального тарифу.
4	Система повинна мати зручний інтерфейс користувача, який дозволяє пасажирам отримувати інформацію про актуальні тарифи, планувати поїздки та оплачувати проїзд.
5	Генерація графіків для візуалізації структури тексту, наприклад, частота появи термінів у тексті.

Не функціональні вимоги описують характеристики системи, такі як надійність, швидкодія, безпека, доступність та інші якості, які вона повинна мати, але не визначають конкретних функцій, які система повинна виконувати.

Ці вимоги можуть включати такі аспекти, як час відновлення після збою, швидкодія відгуку системи на дії користувача, рівень захисту даних від несанкціонованого доступу та інші характеристики, які впливають на ефективність та ефективність системи в цілому.

Таблиця 2.2 – Нефункціональні вимоги до системи

№	Вимога
1	Система повинна забезпечувати швидку реакцію на зміни в попиті та інших умовах, забезпечуючи мінімальні затримки у розрахунках та формуванні тарифів.
2	Система має працювати стабільно та надійно, мінімізуючи можливість виникнення помилок або відмов.
3	Система повинна бути масштабованою, щоб її можна було швидко розширити на нові маршрути, види транспорту та кількість користувачів.
4	Система повинна бути простою та легкою у використанні для забезпечення максимальної доступності для користувачів будь-якого рівня технічної підготовки.

2.2.2 Технології та інструменти для реалізації

Під час розробки системи з алгоритмом адаптивного формування тарифів пасажирських перевезень буде використано мову JavaScript, клієнт-серверну архітектуру та фреймворки і бібліотеки

Ці технолігії розглянуто в таблиці 2.3:

Таблиця 2.3 – Використані при розробці технології

Технологія/Інструмент	Опис
Клієнт Бібліотеки	React – бібліотека для створення динамічних інтерфейсів користувача, що дозволяє створювати динамічні та інтерактивні веб-сторінки.
	Redux – бібліотека для управління станом інтерфейсу користувача. Вона використовується для зберігання та оновлення даних, що відображаються в інтерфейсі.
	Tailwind – бібліотека для стилізації інтерфейсу користувача. Вона використовується для створення візуально привабливого та зручного інтерфейсу.
Сервер Фреймворк	Node.js - середовище виконання JavaScript, яке дозволяє запускати JavaScript-код на сервері. Вона використовується для розробки динамічних веб-серверів та інших серверних застосунків.

Продовження Таблиці 2.3

Технологія/Інструмент	**Опис**
	Express – фреймворк для створення веб-серверів. Він використовується для обробки HTTP-запитів та відповідей, а також для маршрутизації запитів до відповідних функцій.
Сервер Бібліотеки	TensorFlow – бібліотека з відкритим кодом для машинного навчання. Вона використовується для створення та навчання моделей машинного навчання, які можуть використовуватися для аналізу даних та прогнозування.
Система контролю версій	Git – система контролю версій, яка використовується для відстеження змін у коді.
Інтегроване середовище	Visual Studio Code – безкоштовний редактор коду з відкритим кодом, який використовується для розробки програмного забезпечення. Він має багато функцій, які полегшують написання та відстеження коду.

2.3 Опис та порівняння моделей та нейромереж

Алгоритми машинного навчання - це комп'ютерні алгоритми, які дозволяють системі вчитися з даних без явного програмування. Вони використовуються для створення моделей, які можуть робити прогнози, знаходити закономірності у даних та робити висновки.

Нейромережа - це математична модель, яка імітує роботу нервової системи людини. Вона складається зі штучних нейронів, які зв'язані між собою та організовані у вигляді мережі. Нейрони обробляють вхідні сигнали, передають їх через внутрішній обчислювальний процес і видають вихідні сигнали.

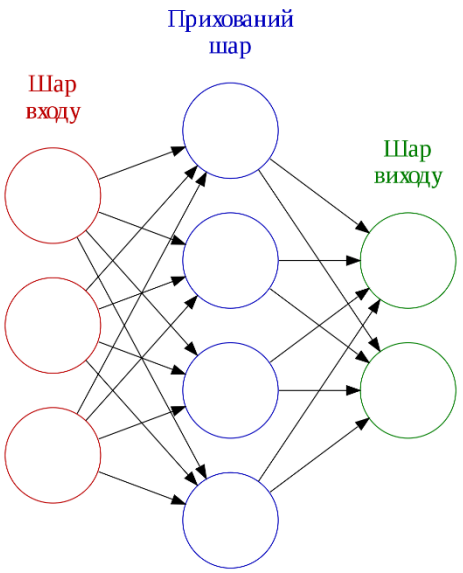

Рисунок 2.7 – Приклад одношарового персептрону нейромережі

Лінійна регресія: Це проста модель, яка використовується для прогнозування значення однієї залежної змінної на основі однієї або декількох незалежних змінних. Лінійна регресія може бути ефективною, якщо зв'язок між тарифами та факторами, що їх впливають, є лінійним.

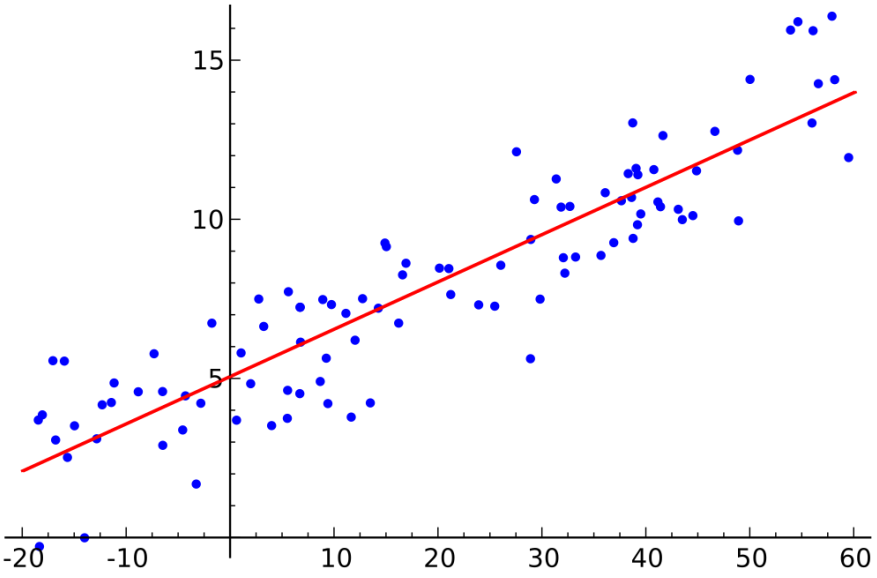

Рисунок 2.8 – Приклад лінійної регреції

Логістична регресія: Ця модель використовується для прогнозування ймовірності події, наприклад, чи буде пасажир користуватися громадським транспортом. Логістична регресія може бути корисною для прогнозування попиту на транспортні послуги.

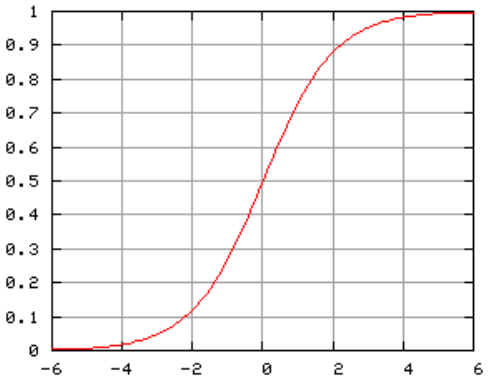

Рисунок 2.9 – Приклад логістичної регреції

K-найближчі сусіди (KNN): Ця модель прогнозує значення нового даного пункту на основі значень найближчих до нього пунктів у наборі даних. Вона може бути корисною для прогнозування тарифів, якщо існує чітка залежність між тарифами для подібних маршрутів або в подібні години.

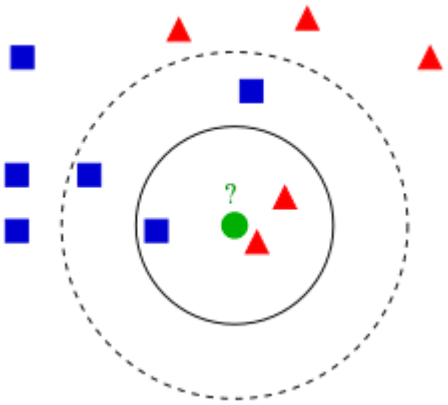

Рисунок 2.10 – K-найближчі сусіди (KNN)

Зразок, позначений зеленим колом, потрібно класифікувати як синій квадрат або червоний трикутник. Якщо кількість найближчих сусідів (k) дорівнює 3, то він буде класифікуватися як червоний трикутник, оскільки всередині зеленого кола є 2 трикутника та лише 1 квадрат. У випадку, коли k = 5, зразок буде класифікуватися як синій квадрат, оскільки всередині більшого кола знаходиться 3 квадрата проти 2 трикутників.

Згорткові нейронні мережі (CNN): CNN - це тип нейронних мереж, які спеціально розроблені для обробки просторових даних, таких як зображення. CNN можуть бути корисними для аналізу даних про пасажиропотік та маршрути.

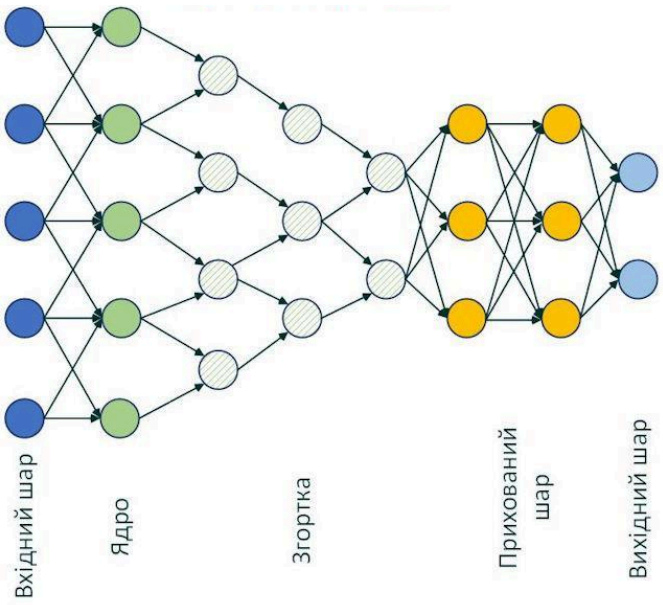

Рисунок 2.11 – Згорткова нейронна мережа (CNN)

Операція згортки в нейронних мережах, зокрема в згорткових нейронних мережах (CNN), використовує спільні параметри шарів, що дозволяє обробляти різні частини вхідного зображення за допомогою однакових вагових коефіцієнтів. Це означає, що ваги фільтрів, які застосовуються до зображення, однакові для різних частин зображення. Ця особливість дозволяє зменшити кількість параметрів, які потрібно навчати в нейронній мережі, що полегшує процес навчання і робить модель менш вимогливою до обчислювальних ресурсів.

Рекурентні нейронні мережі (RNN): RNN - це тип нейронних мереж, які спеціально розроблені для обробки послідовних даних, таких як час. RNN можуть бути корисними для прогнозування попиту на транспортні послуги в динамічному середовищі.

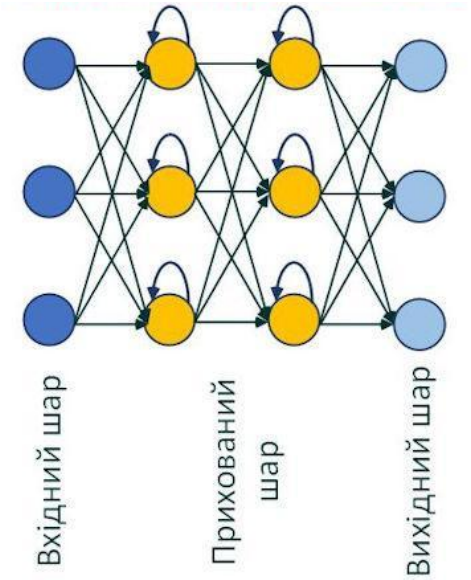

Рисунок 2.12 – Рекурентні нейронні мережі (RNN)

Основна концепція рекурентних нейронних мереж (RNN) полягає в тому, що вони здатні зберігати інформацію про попередні кроки і використовувати її для аналізу в більш складних контекстах. Ці мережі дозволяють створювати моделі, що мають вбудовану пам'ять для зберігання даних і виявлення короткочасних залежностей. Вони також широко використовуються у

прогнозуванні часових рядів, щоб виявити кореляції між даними та встановити закономірності.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕСПЕЧЕННЯ

3.1 Аналіз середовища програмування

При обранні технологій для розробки програмного забезпечення було проаналізовано ряд факторів, що підкреслюють розумність вибору мови програмування JavaScript та середовища розробки Visual Studio Code. JavaScript, як мова, користується широкою популярністю у веб-розробці та в інших застосуваннях, що охоплюють як фронтенд, так і бекенд. Маючи велику спільноту розробників та розмаїття інструментів та бібліотек, JavaScript полегшує розробку програм та дозволяє швидко створювати якісні продукти.

Плюси:

Універсальність: JavaScript можна використовувати як на клієнтській стороні (у браузері), так і на серверній стороні (за допомогою Node.js), що робить його дуже гнучким для створення повномасштабних додатків.

Широкі можливості: JavaScript має величезну екосистему бібліотек і фреймворків, таких як React.js, Angular, Vue.js, що спрощує розробку та підтримку великих проектів.

Асинхронне програмування: JavaScript підтримує асинхронне програмування, що дозволяє створювати ефективні та відзивчиві додатки, особливо в мережевому середовищі.

Мінуси:

Динамічна типізація: JavaScript має динамічну типізацію, що може призвести до помилок в час виконання програми через неочікувані типи даних.

Відсутність компіляції: JavaScript є інтерпретованою мовою, що може призвести до менш ефективної роботи порівняно з компільованими мовами, такими як Java або C++

Рисунок 3.1 – Visual Studio Code

Visual Studio Code — це інтегроване середовище розробки (IDE), розроблене компанією Microsoft. Це потужний інструмент для створення різноманітних програмних продуктів, включаючи веб-додатки, мобільні додатки, десктопні програми, ігри, сервіси та інші рішення.

Visual Studio Code надає широкий набір функцій та інструментів, які допомагають розробникам в усіх етапах розробки програмного забезпечення, від написання коду до тестування та налагодження.

Основна область IDE відображає редактор коду з кодом на мові JavaScript, який включає імпортовані заявки, визначення функцій та конфігурації викликів API.

Зліва в IDE є бічна панель з кількома вкладками, такими як “Explorer”, “Search”, “Source Control” та “Extensions”. Вибрано вкладку “Explorer”, яка показує структуру каталогів з папками, такими як “node_modules”, “public”, та файлами, наприклад, “package.json”.

Внизу лівого кута є значки для управління обліковим записом та налаштуваннями. У верхній частині IDE видно меню з опціями “File”, “Edit”

та “View”

3.2 Створення сховища даних

Під час створення цього сховища даних я використовував інформацію у форматі JSON. В файлі цього формату містяться основні дані про маршрути та транспортні засоби, а також інші необхідні додаткові відомості, що включалися для більш повного розуміння контексту та характеристик транспортного руху.

Масив про транспортні засоби містить дані про різні види транспорту і включає в себе атрибути, такі як їхні місткості, споживання палива, податки компанії та інші.

id	name	capacity	fuel_consumption	company_tax	vehicle_type	fuel_type	distance_km	maintenance_cost
1	Yolande Rackstraw	43	79.25	35.42	bus	diesel	73.52	822.62
2	Carlin Luddy	18	54.89	28.44	bus	gasoline	79.42	60.16
3	Kendra Wattam	45	99.88	24.78	bus	gasoline	59.54	626.35
4	Nichole McInnes	16	77.93	27.09	bus	diesel	27.89	530.85
5	Weider Elphey	6	51.12	10.81	bus	gasoline	9.74	177.04
6	Sander Head	42	56.61	0.01	bus	gasoline	33.91	656.75
7	Nellie Karpenya	4	69.08	16.55	bus	gasoline	93.54	907.41
8	Camey Casassa	40	93.71	29.8	bus	gasoline	1.76	708.48
9	Clemmy MacConchie	13	74.33	32.29	bus	gasoline	33.54	600.52

Рисунок 3.2 – Приклад транспортних засобів

Таблиця 3.1 – Таблиця транспорту

Технологія/Інструмент	**Опис**
name	Назва транспорту
capacity	Вмістимість транспорту
fuel_consumption	Споживання палива (на км)
company_tax	Податок компанії за транспорт
route_id	Ідентифікатор маршруту
vehicle_type	Тип транспортного засобу
fuel_type	Тип палива
transport_type	Тип транспорту
maintenance_cost	Витрати на обслуговування

Масив про маршрути містить інформацію про різні маршрути, включаючи дату, місцезнаходження початку та кінця маршруту, час руху, погодні умови, кількість пасажирів та інші параметри.

route_id	date	start_location	end_location	start_time	end_time	day_of_week	day_type	passenger_count	fare_amount_curency	weather_condition	traffic_condition	route_distance	fuel_price_per_litter	transport_id	fare_amount
1	7/15/2022	094 Morrow Pass	64 Carberry Way	10:08 PM	5:40 PM	Tuesday	Weekend	86	Hryvnia	Snowy	Light	47.36	55.97	4	0
2	8/14/2022	637 Brown Hill	31444 Eagan Center	10:16 PM	4:20 PM	Tuesday	Weekend	59	Dollar	Snowy	Light	42.61	24.06	20	0
3	10/21/2022	7 Cardinal Trail	06 Fulton Plaza	10:50 PM	7:47 PM	Sunday	Weekday	74	Koruna	Cloudy	Moderate	31.56	31.27	6	0
4	5/3/2022	20895 Eggendart Lane	86 Di Loreto Drive	9:40 AM	4:17 PM	Tuesday	Weekday	24	Krona	Snowy	Moderate	27.73	11.84	7	0
5	11/27/2022	330 Division Point	59 Crest Line Terrace	5:52 PM	7:09 PM	Friday	Weekday	53	Peso	Sunny	Light	21.98	54.06	5	0
6	1/28/2022	67789 Nobel Junction	2044 Grover Circle	1:01 PM	4:20 AM	Friday	Holiday	73	Peso	Cloudy	Light	23.89	26.98	25	0
7	2/8/2022	00 Hollow Ridge Point	93694 Scott Terrace	5:11 AM	6:24 PM	Sunday	Weekend	55	Krona	Cloudy	Moderate	26.11	20.59	12	0

Рисунок 3.3 – Приклад маршрутів

Таблиця 3.2 – Таблиця маршрутів

Технологія/Інструмент	**Опис**
route_id	Ідентифікатор маршруту
date	Дата маршруту
start_location	Початкове місце розташування
end_location	Кінцеве місце розташування
start_time	Початковий час відправлення
end_time	Кінцевий час прибуття
day_of_week	День тижня
day_type	Тип дня (робочий, вихідний, свято)
passenger_count	Кількість пасажирів
fare_amount	Сума тарифу
fare_amount_curency	Валюта суми тарифу
weather_condition	Погодні умови
traffic_condition	Умови дорожнього руху

route_distance	Відстань маршруту у кілометрах
fuel_price_per_litter	Вартість пального за літр

3.3 Огляд основних функцій

Програмна архітектура дослідження організована як клієнт-серверний веб-сайт, де частина відповідальна за передбачення зосереджена на серверному боці. Ця частина коду розділена на три основні функції:

```
const createMlpModel = () => {
  const model = tf.sequential();
  model.add(tf.layers.dense({ units: 64, activation: 'relu', inputShape:
  [5] }));
  model.add(tf.layers.dense({ units: 64, activation: 'relu' }));
  model.add(tf.layers.dense({ units: 1 }));
  model.compile({ optimizer: 'adam', loss: 'meanSquaredError' });
  return model;
};
```

```
const createCnnModel = () => {
  const model = tf.sequential();
  model.add(tf.layers.reshape({ targetShape: [5, 1], inputShape: [5] }));
  model.add(tf.layers.conv1d({ filters: 32, kernelSize: 3, activation:
  'relu' }));
  model.add(tf.layers.maxPooling1d({ poolSize: 2 }));
  model.add(tf.layers.flatten());
  model.add(tf.layers.dense({ units: 64, activation: 'relu' }));
  model.add(tf.layers.dense({ units: 1 }));
  model.compile({ optimizer: 'adam', loss: 'meanSquaredError' });
  return model;
};
```

```
const createRnnModel = () => {
  const model = tf.sequential();
  model.add(tf.layers.reshape({ targetShape: [1, 5], inputShape: [5] }));
  model.add(tf.layers.lstm({ units: 64, returnSequences: true }));
  model.add(tf.layers.lstm({ units: 64 }));
  model.add(tf.layers.dense({ units: 1 }));
  model.compile({ optimizer: 'adam', loss: 'meanSquaredError' });
  return model;
};
```

Рисунок 3.4 – Код createNeuralModel

createNeuralModel: у Neural показанотип нейромережі (MLP, CNN або RNN) і кожна з функцій створює відповідну модель з відповідними параметрами, такими як кількість шарів, кількість нейронів у кожному шарі,

активаційні функції та оптимізатор. Після цього модель компілюється з вказаним оптимізатором та функцією втрат для готовності до тренування.

```
const determineBestModel = (
  mlpPredictions,
  cnnPredictions,
  rnnPredictions,
  y_test
) => {
  const mlpMSE = meanSquaredError(mlpPredictions, y_test);
  const cnnMSE = meanSquaredError(cnnPredictions, y_test);
  const rnnMSE = meanSquaredError(rnnPredictions, y_test);

  console.log(`MLP Model MSE: ${mlpMSE}`);
  console.log(`CNN Model MSE: ${cnnMSE}`);
  console.log(`RNN Model MSE: ${rnnMSE}`);

  const bestModel = Math.min(mlpMSE, cnnMSE, rnnMSE);
  if (bestModel === mlpMSE) {
    return 'MLP';
  } else if (bestModel === cnnMSE) {
    return 'CNN';
  } else {
    return 'RNN';
  }
};
```

Рисунок 3.5 – Код determine_best_model

determine_best_model: Функція determine_best_model реалізує процес визначення найкращої моделі нейронної мережі на основі найменшої середньо-квадратичної помилки (MSE). Дана функція дозволяє автоматизувати процес вибору найкращої моделі для конкретного завдання машинного навчання на основі емпіричного оцінювання на тестових даних.

Рисунок 3.6 – Код preprocess_data

preprocess_data: функція виконує попередню обробку вхідних даних перед їх подальшим використанням у тренуванні моделі чи проведенні передбачень. Основна мета передпрацювання - підготувати дані для подальшої обробки таким чином, щоб вони були придатні для моделювання.

Рисунок 3.7 – Код trainModel

trainModel: функція призначена для тренування моделі машинного навчання на вхідних даних після їх попередньої обробки. Основна мета цієї функції - навчити модель на даних з метою подальшого використання її для проведення передбачень на нових або тестових даних.

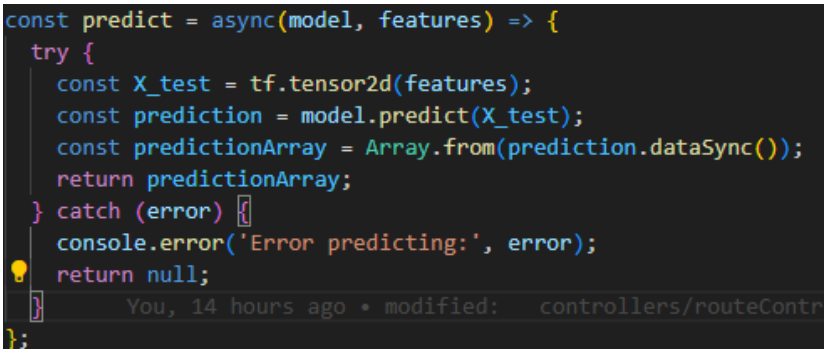

Рисунок 3.8 – Код predicts

predicts: функція використовує навчену модель для проведення передбачень на нових даних. Основна мета цієї функції - застосувати модель, яка була навчена на попередньому етапі, до нових даних і отримати передбачені значення цільової змінної чи класу

ВИСНОВОК

Розробка системи з алгоритмом адаптивного формування тарифів пасажирських перевезень в інтелектуальній транспортній системі міста відображає важливий напрямок розвитку сучасних міських транспортних сервісів. Цей підхід дозволяє створювати більш ефективні та конкурентоспроможні системи перевезень, що відповідають потребам користувачів. Шляхом аналізу даних про використання та попит на транспортні послуги, алгоритм може динамічно адаптувати тарифи залежно від часу доби, обсягу пасажиропотоку, а також інших факторів, що впливають на попит.

JavaScript є потужним інструментом для розробки веб-додатків та серверних застосувань, і використання його в контексті розробки інтелектуальних транспортних систем може забезпечити гнучкість та швидкість розгортання таких рішень. Використання TensorFlow, відкритої бібліотеки машинного навчання, може допомогти в розробці складних алгоритмів для аналізу даних та прогнозування пасажирських потоків, що є

ключовим аспектом у реалізації адаптивних тарифних систем.

На основі зазначених технологій можна створити систему, яка здатна адаптувати тарифи на пасажирські перевезення в реальному часі, враховуючи різноманітні фактори, такі як час доби, популярність маршруту, зміни в транспортній інфраструктурі тощо. Це дозволить оптимізувати використання ресурсів та забезпечити ефективне функціонування системи пасажирських перевезень в місті.

ПЕРЕЛІК ПОСИЛАНЬ

1. United Nations Economic Commission for Europe (UNECE) (2012). Intelligent Transport Systems (ITS) for sustainable mobility. – 123 с. Режим доступу https://unece.org/transport/publications/intelligent-transport-systems-its-sustainable-mobility
2. The secretariat of the Economic and Social Commission for Asia and the Pacific (ESCAP) (2015). Intelligent Transportation Systems for Sustainable Development in Asia and the Pacific. – 38 с. Режим доступу https://www.unescap.org/sites/default/files/ITS.pdf
3. Linköping University. Faculty of Science and Engineering (2024). Master's Programme in Intelligent Transport Systems and Logistics. Режим доступу https://studieinfo.liu.se/en/program/6mtsl/#syllabus
4. Chenyang Gao, Jianfeng Ma, Teng Li & Yulong Shen (2023). Hybrid swarm intelligent algorithm for multi-UAV formation reconfiguration. – С. 1929-1962. Режим доступу https://link.springer.com/article/10.1007/s40747-022-00891-7

Додаток А

Сервер

app.js

```
const express = require('express');
const cors = require('cors');
const transportsRoutes = require('./routes/transport.routes');
const routeRoutes = require('./routes/route.routes');
const utilRoutes=require('./routes/utils.routes')
require('dotenv').config();
const app = express();
const port = process.env.PORT || 3001;
app.use(cors());
app.use(express.json());
app.use('/transports', transportsRoutes);
app.use('/routes', routeRoutes);
app.use('/utils',utilRoutes)
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ message: 'Internal Server Error' });
});
app.listen(port, (error) => {
  if (!error) {console.log("Server started on port " + port);
  } else {console.log("Error occurred, server can't start", error);}
});
```

util.routes.js

```
const express =require('express')
const router=express.Router()
const utilsController=require('../controllers/utilController')
const utils1Controller=require('../controllers/util1Controller')
router.get('/',utilsController.getStandartUtil)
router.get('/structure',utils1Controller.getDataStructure)
router.get('/train',utils1Controller.postTrain)
router.post('/predict',utils1Controller.postPredict)
module.exports=router
```

transport.routes.js

```
const express =require('express')
const router=express.Router()
const transportController=require('../controllers/transportController')
router.get('/',transportController.getAllTransports)
module.exports=router
```

route.routes.js

```
const express =require('express')
const router=express.Router()
const routeController=require('../controllers/routeController')
router.get('/',routeController.getAllRoutes)
router.get('/pasang',routeController.getAllRoutesWithBusName)
router.get('/calculate',routeController.getCalculateAll)
module.exports=router
```

util1Controller.js

```
const utilService =require('../services/util1Service')
const getDataStructure=async(req,res)=>{
 try{
   const dataStructure = await utilService.sendStructure();

   res.status(200).json(dataStructure);
 }catch (error) {
   console.error(error);
   res.status(500).json({ message: 'Internal Server Error' });
 }
}
const postTrain = async (req, res) => {
        try {
                const data=await utilService.loadData()
                const { features, labels } = utilService.preprocessData(data);
                console.log('Work 1')
                const mlpModel = utilService.createMlpModel();
                const mlpModelTrained = await utilService.trainModel(mlpModel, features,
labels);
                await utilService.saveModel(mlpModelTrained, 'mlp_model');
                /*const rnnModel = utilService.createRnnModel();
                await utilService.trainModel(rnnModel, features, labels);
                await utilService.saveModel(rnnModel, 'rnn_model');
                const cnnModel = utilService.createCnnModel();
                await utilService.trainModel(cnnModel, features, labels);
                await utilService.saveModel(cnnModel, 'cnn_model');*/
                res.status(200).json({ message: 'Models trained and saved successfully' });
} catch (error) {res.status(500).json({ error: error.message });}
};
const postPredict = async (req, res) => {
        try {
                const data = req.body;
                if(data!=null||data!=''){
                const {features} =utilService.preprocessData(data);
                const loadedMlpModel = await utilService.loadModel('mlp_model');
                /*const loadedCnnModel = await utilService.loadModel('cnn_model');
                const loadedRnnModel = await utilService.loadModel('rnn_model');*/
                const mlpPredictions = await utilService.predict(loadedMlpModel, features);
                /*const cnnPredictions = utilService.predict(loadedCnnModel, features);
                const rnnPredictions = utilService.predict(loadedRnnModel, features);
                const bestModel = utilService.determineBestModel(mlpPredictions,
cnnPredictions, rnnPredictions, labels);*/
                res.status(200).json(mlpPredictions);
}
                else{res.status(200).json({message:"The data do not match the sample"});
} catch (error) {res.status(500).json({ error: error.message });
}};
const sendData=mlpPredictions.map((item)=>utilService.calc(item,2))
module.exports = { postTrain, postPredict ,getDataStructure};
```

transportController.js

```
const transportService =require('../services/transportService')
const getAllTransports = async (req, res) => {
 try {
  const { id, start, end } = req.query;
  if (id) {
   const transport = await transportService.getTransportById(parseInt(id, 10));
   if (!transport) {
    return res.status(404).json({ message: 'Transport not found' });
   }
   const routes = await transportService.getRoutesByTransportId(parseInt(id, 10));
   return res.status(200).json({ transport, routes });
  }
  const transports = await transportService.getAllTransports({
   start: start ? parseInt(start, 10) : undefined,
   end: end ? parseInt(end, 10) : undefined,
  });
  return res.status(200).json(transports);
 } catch (error) {
  console.error(error);
  return res.status(500).json({ message: 'Internal Server Error' });
 }
};
const getTransportsWithRoutes=async(req,res)=>{
 try{
  const data=await transportService.transportsWithRoutes()
  res.status(200).json(data)
 }
 catch(error){res.status(500).json({message:''})}
}
module.exports={getAllTransports,getTransportsWithRoutes}
```

routeController.js

```
const routeService =require('../services/routesService')
const getAllRoutes = async (req, res) => {
 try {
  const start = req.query.start ? parseInt(req.query.start) : undefined;
  const end = req.query.end ? parseInt(req.query.end) : undefined;
  const routes = await routeService.getRoutes({ start, end });
  res.status(200).json(routes);
 } catch (error) {
  console.error(error);
  res.status(500).json({ message: 'Internal Server Error' });}
};
const getAllRoutesWithBusName = async (req, res) => {
 try {
  const data = await routeService.getRoutesWithTransportName();
  res.status(200).json(data);
 } catch (error) {
  console.error(error);
  res.status(500).json({ message: 'Internal Server Error' });
 }
```

```
};
const getCalculateAll = async (req, res) => {
  try {
    await routeService.calculateAll();
    res.status(200).json({ message: 'Calculate Good' });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Internal Server Error' });}
};
module.exports={getAllRoutes,getAllRoutesWithBusName,getCalculateAll}
```

util1Service.js

```
const fs = require('fs').promises;
const path = require('path');
const dataFilePath = path.join(__dirname, '../data/data.json');
const modelsPath = path.join(__dirname, '../models');
const tf = require('@tensorflow/tfjs');
require('tfjs-node-save');
const preprocessData = (data) => {
  const weatherConditions = { Snowy: 3, Rain: 1, Clear: 0, Fog: 2 };
  const trafficConditions = { High: 2, Light: 0, Medium: 1 };
  const defaultWeatherCondition = 0;
  const defaultTrafficCondition = 0;
  const features = [];
  const labels = [];
  data.routes.forEach((route) => {
    const featureVector = [
      weatherConditions[route.weather_condition] ?? defaultWeatherCondition,
      trafficConditions[route.traffic_condition] ?? defaultTrafficCondition,
      route.fuel_price_per_litter,
      route.passenger_count,
      route.route_distance,
    ];
    const label = route.fare_amount;
    features.push(featureVector);
    labels.push(label);
  });
  return { features, labels };
};
const createMlpModel = () => {
  const model = tf.sequential();
  model.add(tf.layers.dense({ units: 64, activation: 'relu', inputShape: [5] }));
  model.add(tf.layers.dense({ units: 64, activation: 'relu' }));
  model.add(tf.layers.dense({ units: 1 }));
  model.compile({ optimizer: 'adam', loss: 'meanSquaredError' });
  return model;
};
const createCnnModel = () => {
  const model = tf.sequential();
  model.add(tf.layers.reshape({ targetShape: [5, 1], inputShape: [5] }));
  model.add(tf.layers.conv1d({ filters: 32, kernelSize: 3, activation: 'relu' }));
  model.add(tf.layers.maxPooling1d({ poolSize: 2 }));
```

```
  model.add(tf.layers.flatten());
  model.add(tf.layers.dense({ units: 64, activation: 'relu' }));
  model.add(tf.layers.dense({ units: 1 }));
  model.compile({ optimizer: 'adam', loss: 'meanSquaredError' });
  return model;
};

const createRnnModel = () => {
  const model = tf.sequential();
  model.add(tf.layers.reshape({ targetShape: [1, 5], inputShape: [5] }));
  model.add(tf.layers.lstm({ units: 64, returnSequences: true }));
  model.add(tf.layers.lstm({ units: 64 }));
  model.add(tf.layers.dense({ units: 1 }));
  model.compile({ optimizer: 'adam', loss: 'meanSquaredError' });
  return model;
};
const meanSquaredError = (predictions, actuals) => {
  const errors = predictions.map((pred, index) =>
    Math.pow(pred[0] - actuals[index], 2)
  );
  return errors.reduce((a, b) => a + b, 0) / errors.length;
};
const determineBestModel = (
  mlpPredictions,
  cnnPredictions,
  rnnPredictions,
  y_test
) => {
  const mlpMSE = meanSquaredError(mlpPredictions, y_test);
  const cnnMSE = meanSquaredError(cnnPredictions, y_test);
  const rnnMSE = meanSquaredError(rnnPredictions, y_test);
  console.log(`MLP Model MSE: ${mlpMSE}`);
  console.log(`CNN Model MSE: ${cnnMSE}`);
  console.log(`RNN Model MSE: ${rnnMSE}`);
  const bestModel = Math.min(mlpMSE, cnnMSE, rnnMSE);
  if (bestModel === mlpMSE) {
    return 'MLP';
  } else if (bestModel === cnnMSE) {
    return 'CNN';
  } else {
    return 'RNN';
  }
};
const trainModel = async (model, features, labels) => {
  const X = tf.tensor2d(features);
  const y = tf.tensor2d(labels, [labels.length, 1]);
  await model.fit(X, y, {
    epochs: 100,
    batchSize: 32,
    validationSplit: 0.2,
  });
  return model;
```

```
};
const saveModel = async (model, modelName) => {
  const modelSavePath = path.resolve(modelsPath, modelName);
  console.log(modelSavePath)
  try {
    await model.save(`file://${modelSavePath}`);
    console.log(`Model ${modelName} saved successfully.`);
    return
  } catch (error) {
    console.error(`Error saving model ${modelName}: ${error}`);
    throw error;
  }
};
const loadModel = async (modelName) => {
  const modelLoadPath = path.join(modelsPath, modelName, 'model.json');
  console.log(modelLoadPath)
  return await tf.loadLayersModel(`file://${modelLoadPath}`);
};
const predict = async(model, features) => {
  try {
    const X_test = tf.tensor2d(features);
    const prediction = model.predict(X_test);
    const predictionArray = Array.from(prediction.dataSync());
    return predictionArray;
  } catch (error) {
    console.error('Error predicting:', error);
    return null;
  }
};
const getType = (value) => {
  if (Array.isArray(value)) {
    return 'array';
  } else if (typeof value === 'object' && value !== null) {
    return 'object';
  } else {
    return typeof value;
  }
};
const capitalize = (str) => str.charAt(0).toUpperCase() + str.slice(1);
const getStructure = (obj, typeDefinitions = {}, parentKey = 'Root') => {
  const typeName = capitalize(parentKey);
  if (!typeDefinitions[typeName]) {
    typeDefinitions[typeName] = {};
  }
  for (const key in obj) {
    const value = obj[key];
    const type = getType(value);

    if (type === 'object') {
      const nestedTypeName = `${typeName}${capitalize(key)}`;
      getStructure(value, typeDefinitions, `${typeName}${capitalize(key)}`);
      typeDefinitions[typeName][key] = nestedTypeName;
```

```
    } else if (type === 'array') {
      if (value.length > 0) {
        const arrayType = getType(value[0]);
        if (arrayType === 'object') {
          const nestedTypeName = `${typeName}${capitalize(key)}Item`;
          getStructure(
            value[0],
            typeDefinitions,
            `${typeName}${capitalize(key)}Item`
          );
          typeDefinitions[typeName][key] = `${nestedTypeName}[]`;
        } else {
          typeDefinitions[typeName][key] = `${arrayType}[]`;
        }
      } else {
        typeDefinitions[typeName][key] = 'any[]';
      }
    } else {
      typeDefinitions[typeName][key] = type;
    }
  }
  return typeDefinitions;
};
const loadData = async () => {
  try {
    const rawData = await fs.readFile(dataFilePath);
    return JSON.parse(rawData);
  } catch (error) {
    console.error('Error loading data:', error);
    return null;
  }
};
const sendStructure = async () => {
  try {
    const rawData = await fs.readFile(dataFilePath, 'utf8');
    const data = JSON.parse(rawData);
    const typeDefinitions = getStructure(data);
    let result = '';
    for (const [typeName, fields] of Object.entries(typeDefinitions)) {
      result += `interface ${typeName} {`;
      for (const [field, type] of Object.entries(fields)) {
        result += `  ${field}: ${type};`;
      }
      result += '}';
    }
    return result;
  } catch (error) {
    console.error(`Error reading or processing data: ${error}`);}
};
const calc=(n,c)=>{ return Number(n.toFixed(c))  }
module.exports = {
  loadModel, predict, trainModel, saveModel, createMlpModel, createCnnModel,
```

```
  createRnnModel, determineBestModel, preprocessData, sendStructure, loadData,calc};
```

transportService.js

```
const fs=require('fs').promises
const path =require('path')
const tf = require('@tensorflow/tfjs');
const dataFilePath=path.join(__dirname,'../data/data.json')
const getAllTransports = async (filter = {}) => {
  try {
    const data = await fs.readFile(dataFilePath, 'utf-8');
    let transports = JSON.parse(data).transports;
    if (filter.start !== undefined && filter.end !== undefined) {
      transports = transports.filter(
        (transport) =>
          transport.id >= filter.start && transport.id <= filter.end
      );
    }
    return transports;
  } catch (error) {
    console.error('Error reading bus data:', error);
    throw new Error('Failed to retrieve bus data');
  }
};
const transportsWithRoutes = async () => {
  try {
    const data = await fs.readFile(dataFilePath, 'utf-8');
    const parsedData = JSON.parse(data);
    const transports = parsedData.transports;
    const routes = parsedData.routes;
    const result = transports.map(transport => {
      const associatedRoutes = routes.filter(route => route.transport_id === transport.id);
      return {
        ...transport,
        routes: associatedRoutes
      };
    });
    return result;
  } catch (error) {
    console.error('Error retrieving transports with routes:', error);
    throw new Error('Failed to retrieve transports with routes');
  }
};
const getTransportById = async (id) => {
  try {
    const data = await fs.readFile(dataFilePath, 'utf-8');
    const transports = JSON.parse(data).transports;
    return transports.find((transport) => transport.id === id);
  } catch (error) {
    console.error('Error reading transport data:', error);
    throw new Error('Failed to retrieve transport data');
  }
};
const getRoutesByTransportId = async (id) => {
```

```
  try {
    const data = await fs.readFile(dataFilePath, 'utf-8');
    const routes = JSON.parse(data).routes;
    return routes.filter((route) => route.transport_id === parseInt(id));
  } catch (error) {
    console.error('Error reading route data:', error);
    throw new Error('Failed to retrieve route data');
  }
};
module.exports = {
  getAllTransports,
  transportsWithRoutes,
  getTransportById,
  getRoutesByTransportId
};
```

routeService.js

```
  const fs = require('fs').promises;
  const path = require('path');
  const tf = require('@tensorflow/tfjs');
  const dataFilePath = path.join(__dirname, '../data/data.json');
  const calculateFuelPricePerKm = (fuelConsumptionPer100Km, pricePerLiter) => {
    return (fuelConsumptionPer100Km / 100) * pricePerLiter;
  };
  const calculateBaseTicketPrice = (
    distance,
    fuelPricePerKm,
    companyProfit,
    additionalExpenses
  ) => {
    const fuelCost = distance * fuelPricePerKm;
    const basePrice = fuelCost + additionalExpenses;
    const finalPrice = basePrice + basePrice * (companyProfit / 100);
    return finalPrice;
  };
  const getRoutes = async (filter = {}) => {
    try {
      const data = await fs.readFile(dataFilePath, 'utf-8');
      let routes = JSON.parse(data).routes;

      if (filter.start !== undefined && filter.end !== undefined) {
        routes = routes.filter(
          (route) =>
            route.route_id >= filter.start && route.route_id <= filter.end
        );
      }
      return routes;
    } catch (error) {
      console.error('Error reading route data:', error);
      throw new Error('Failed to retrieve route data');
    }
  };
```

```
const getRoutesWithTransportName = async (filter = {}) => {
  try {
    const data = await fs.readFile(dataFilePath, 'utf-8');
    const dataJson = JSON.parse(data);
    const routesWithDetails = dataJson.routes.map((route) => {
      const transport = dataJson.transports.find(
        (t) => t.id === route.transport_id
      );
      return {
        route_id: route.route_id,
        date: route.date,
        start_location: route.start_location,
        end_location: route.end_location,
        transport_name: transport ? transport.name : 'Unknown',
        start_time: route.start_time,
        end_time: route.end_time,
        fare_amount: route.fare_amount,
        fare_amount_currency: route.fare_amount_curency,
        bus_id: route.transport_id,
      };
    });
    return routesWithDetails;
  } catch (error) {
    console.error('Error retrieving routes with details:', error);
    throw new Error('Failed to retrieve routes with details');
  }
};
const parseFilterString = (filterString) => {
  const filterObj = {};
  const pairs = filterString.split('&');
  pairs.forEach((pair) => {
    const [key, value] = pair.split('=');
    filterObj[key] = value;
  });
  return filterObj;
};
const addRoute = async (route) => {
  try {
    const data = await fs.readFile(dataFilePath, 'utf-8');
    const routes = JSON.parse(data).routes;
    routes.push(route);
    await fs.writeFile(dataFilePath, JSON.stringify({ routes }, null, 2));
    console.log('Route added successfully');
  } catch (error) {
    console.error('Error adding route:', error);
    throw new Error('Failed to add route');
  }};
const updateRoute = async (id, route) => {
  const data = await fs.readFile(dataFilePath, 'utf-8');
  const routes = JSON.parse(data).routes;
  const index = routes.findIndex((r) => r.id === id);
  if (index !== -1) {
```

```
    routes[index] = { ...routes[index], ...route };
    await fs.writeFile(dataFilePath, JSON.stringify({ routes }, null, 2));
    console.log('Route updated successfully');
  } else {
    console.error('Route not found');
  }};
const deleteRoute = async (id) => {
  try {
    const data = await fs.readFile(dataFilePath, 'utf-8');
    let routes = JSON.parse(data).routes;
    routes = routes.filter((route) => route.id !== id);
    await fs.writeFile(dataFilePath, JSON.stringify({ routes }, null, 2));
    console.log('Route deleted successfully');
  } catch (error) {
    console.error('Error deleting route:', error);
    throw new Error('Failed to delete route');
 }};
const calculateAll = async () => {
  try {
    const data = await fs.readFile(dataFilePath, 'utf-8');
    const parsedBusData = JSON.parse(data);
    const routesData = parsedBusData.routes;
    for (let i = 0; i < routesData.length; i++) {
      const route = routesData[i];
      const distance = route.route_distance;
      const transport = parsedBusData.transports.find(
        (t) => t.id === route.transport_id
      );
      if (transport) {
        const fuelPricePerKm = calculateFuelPricePerKm(
          transport.fuel_consumption,
          route.fuel_price_per_litter
        );
        let baseTicketPrice = calculateBaseTicketPrice(
          distance,
          fuelPricePerKm,
          transport.company_tax,
          transport.maintenance_cost
        );
        baseTicketPrice = baseTicketPrice / 30;
        route.fare_amount = Math.floor(baseTicketPrice * 100) / 100;
      }
    }
    const resultFilePath = path.join(__dirname, '../data/calc.json');
    await fs.writeFile(resultFilePath, JSON.stringify(parsedBusData, null, 2));
  } catch (error) {
    console.error('Error calculating prices:', error);
    throw new Error('Failed to calculate prices');
  }
};
module.exports = {getRoutes, addRoute, updateRoute, deleteRoute, calculateAll,
getRoutesWithTransportName,};
```

Клієнт

App.jsx

```
import { Provider } from 'react-redux';

import store from './store/store';
import { RouterProvider } from 'react-router-dom';
import router from './routes/routings';
const App=()=> {

  return (<>
    <Provider store={store}>
      <RouterProvider router={router}/>
    </Provider>
  </>);
}

export default App;
```

routings.jsx

```
import { createBrowserRouter } from 'react-router-dom';
import Home from '../pages/Home';
import ErrorComponent from '../layouts/Layout1/ErrorComponent2';
import TransportsPage from '../pages/TransportsPage';
import PredictPage from '../pages/PredictPage';
import RoutePage from '../pages/RoutePage';
import TransportPage from '../pages/TransportPage';
const router = createBrowserRouter([
  {
    path: '/',
    element: <Home />,
    index: true,
    errorElement: <ErrorComponent />,
  },
  {
    path: '/routes',
    element: <RoutePage />,
  },
  {
    path: '/transports',
    element: <TransportsPage />,
  },
  {
    path: '/transports/:id',
    element: <TransportPage />,
  },
  {
    path: '/predict',
    element: <PredictPage />,
  },
]);
export default router;
```

store.js

```
import { configureStore } from "@reduxjs/toolkit";
import transportReducer from "./reducers/transportReducer";
import routeReducer from "./reducers/routeReducer";
import utilReducer from "./reducers/utilReducer";
const store=configureStore({
        reducer:{
                transports:transportReducer,
                routes:routeReducer,
                utils:utilReducer
        }
})
export default store
```

routes.actions.js

```
import axios from 'axios';
const url = import.meta.env.VITE_API_URL;
export const ADD_ROUTE = 'ADD_ROUTE::ROUTE';
export const UPDATE_ROUTE = 'UPDATE_ROUTE::ROUTE';
export const DELETE_ROUTE = 'DELETE_ROUTE::ROUTE';
export const SET_ROUTES = 'SET_DATA::ROUTE';
export const SET_DATA = 'SET_DATA::ROUTENET';
export const SET_LOADING = 'SET_LOADING::ROUTENET';
export const SET_ERROR = 'SET_ERROR::ROUTENET';
export const addRoute = (route) => ({
  type: ADD_ROUTE,
  payload: route,
});
export const updateRoute = (route) => ({
  type: UPDATE_ROUTE,
  payload: route,
});
export const deleteRoute = (route) => ({
  type: DELETE_ROUTE,
  payload: route,
});
export const setLoading = (loading) => ({
  type: SET_LOADING,
  payload: loading,
});
export const setData = (data) => ({
  type: SET_DATA,
  payload: data,
});
export const setRoutes = (data) => ({
  type: SET_ROUTES,
  payload: data,
});
export const setError = (error) => ({
  type: SET_ERROR,
  payload: error,
});
```

```
export const fetchRoutes = (start, end) => async (dispatch) => {
  dispatch(setLoading(true));
  let urlWithParams = `${url}/routes`;

  if (start !== undefined && end !== undefined) {
    urlWithParams += `?start=${start}&end=${end}`;
  }
  try {
    const response = await axios.get(urlWithParams);
    dispatch(setData(response.data));
    dispatch(setRoutes(response.data));
  } catch (error) {
    dispatch(setError(error.message));
  } finally {
    dispatch(setLoading(false));
  }
};
export const fetchRoutesPasang = () => async (dispatch) => {
  dispatch(setLoading(true));
  try {
    console.log(url);
    console.log(`${url}/routes`);
    const response = await axios.get(`${url}/routes/pasang`);
    dispatch(setData(response.data));
    dispatch(setRoutes(response.data));
  } catch (error) {
    dispatch(setError(error.message));
  } finally {
    dispatch(setLoading(false));
  }
};
```

transports.actions.js

```
import axios from 'axios';
const url = import.meta.env.VITE_API_URL;
export const ADD_TRANSPORT = 'ADD_TRANSPORT::TRANSPORT';
export const UPDATE_TRANSPORT = 'UPDATE_TRANSPORT::TRANSPORT';
export const DELETE_TRANSPORT = 'DELETE_TRANSPORT::TRANSPORT';
export const SET_DATA = 'SET_DATA::TRANSPORT';
export const SET_LOADING = 'SET_LOADING::TRANSPORT';
export const SET_ERROR = 'SET_ERROR::TRANSPORT';
export const addTransport = (transport) => ({
  type: ADD_TRANSPORT,
  payload: transport,
});
export const updateTransport = (transport) => ({
  type: UPDATE_TRANSPORT,
  payload: transport,
});
export const deleteTransport = (transport) => ({
  type: DELETE_TRANSPORT,
  payload: transport,
```

```
});
export const setLoading = (loading) => ({
  type: SET_LOADING,
  payload: loading,
});
export const setData = (data) => ({
  type: SET_DATA,
  payload: data,
});
export const setError = (error) => ({
  type: SET_ERROR,
  payload: error,
});
export const fetchTransports = (start, end) => async (dispatch) => {
  dispatch(setLoading(true));
  let urlWithParams = `${url}/transports`;
  if (start !== undefined && end !== undefined) {
    urlWithParams += `?start=${start}&end=${end}`;
  }
  try {
    const response = await axios.get(urlWithParams);
    dispatch(setData(response.data));
  } catch (error) {
    dispatch(setError(error.message));
  } finally {
    dispatch(setLoading(false));
  }
};
export const fetchTransportDetails = (id) => async (dispatch) => {
  dispatch(setLoading(true));
  try {
    console.log(id);
    const response = await axios.get(`${url}/transports?id=${id}`);
    console.log(response.data);
    dispatch(setData(response.data));
  } catch (error) {
    dispatch(setError(error.message));
  } finally {
    dispatch(setLoading(false));
  }
};
```

utils.actions.js

```
import axios from 'axios'
const url=import.meta.env.VITE_API_URL
export const SET_LOADING = 'SET_LOADING::UTILS';
export const SET_ERROR = 'SET_ERROR::UTILS';
export const SET_DATA_STRUCTURE = 'SET_DATA_STRUCTURE::UTILS';
export const SET_TRAIN = 'SET_TRAIN::UTILS';
export const SET_PREDICTIONS = 'SET_PREDICTIONS::UTILS';
export const SET_FILE_DATA = 'SET_FILE_DATA::UTILS';
export const setDataStructure = (dataStructure) => ({
```

```
  type: SET_DATA_STRUCTURE,
  payload: dataStructure
});
export const setPredictions = (predictions) => ({
  type: SET_PREDICTIONS,
  payload: predictions
});
export const setTrainState = (trainingData) => ({
  type: SET_TRAIN,
  payload: trainingData
});
export const setStoreData=(fileData)=>({
  type:SET_FILE_DATA,
  payload: fileData
})
export const setLoading=(loading)=>({
        type:SET_LOADING,
        payload:loading
})
export const setError=(error)=>({
        type:SET_ERROR,
        payload:error
})
export const fetchDataStructure = () => async (dispatch) => {
  dispatch(setLoading(true));
  try {
    const response = await axios.get(`${url}/utils/structure`);
    dispatch(setDataStructure(response.data));
  } catch (error) {
    dispatch(setError(error));
  } finally {
    dispatch(setLoading(false));
  }
};
export const fetchSetAndTrainData = (data) => async (dispatch) => {
  dispatch(setLoading(true));
  try {
    const response = await axios.post(`${url}/utils/train`, data, {
      headers: {
        'Content-Type': 'application/json'
      }
    });
    dispatch(setTrainState(response.data));
    //Data has been successfully trained
    //An error occurred while training data
  } catch (error) {
    dispatch(setError(error));
  } finally {
    dispatch(setLoading(false));
  }
};
export const fetchReTrainData=()=>async (dispatch)=>{
```

```
  dispatch(setLoading(true))
  try{
    const response = await axios.get(`${url}/utils/train`)
    dispatch(setTrainState(response.data))
    //Data has been successfully retrained
    //An error occurred while retraining data
  }catch(error){
    dispatch(setError(error))
  }finally{
    dispatch(false)
  }
}
export const fetchPredict = (data) => async (dispatch) => {
  console.log(data)
  dispatch(setLoading(true));
  try {
    const response = await axios.post(`${url}/utils/predict`, data, {
      headers: {
        'Content-Type': 'application/json'
      }
    });
    dispatch(setPredictions(response.data));
    //Data that was predicted
    //Error predicting data
  } catch (error) {
    dispatch(setError(error.message));
  } finally {
    dispatch(setLoading(false));
  }
};
```

routeReducer.js

```
import {
  ADD_ROUTE,
  DELETE_ROUTE,
  UPDATE_ROUTE,
  SET_DATA,
  SET_LOADING,
  SET_ERROR,
} from '../actions/routes';
const initialState = {
  routes: [],
  loading: false,
  error: null,
};
const routeReducer = (state = initialState, action) => {
  switch (action.type) {
    case ADD_ROUTE:
      return {
        ...state,
        routes: [...state.routes, action.payload],
      };
```

```
    case UPDATE_ROUTE:
      return {
        ...state,
        routes: state.routes.map((route) =>
          route.id === action.payload.id ? action.payload : route
        ),
      };
    case DELETE_ROUTE:
      return {
        ...state,
        routes: state.routes.filter((route) => route.id !== action.payload),
      };
    case SET_DATA:
      return {
        ...state,
        routes: action.payload,
        loading: false,
        error: null,
      };
    case SET_LOADING:
      return {
        ...state,
        loading: action.payload,
        error: null,
      };
    case SET_ERROR:
      return {
        ...state,
        loading: false,
        error: action.payload,
      };
    default:
      return state;
  }
};
export default routeReducer;
```

transportReducer.js

```
import {
  ADD_TRANSPORT,
  DELETE_TRANSPORT,
  SET_DATA,
  SET_ERROR,
  SET_LOADING,
  UPDATE_TRANSPORT,
} from '../actions/transports';
const initialState = {
  transports: [],
  data: null,
  loading: false,
  error: null,
};
```

```
const transportReducer = (state = initialState, action) => {
  switch (action.type) {
    case ADD_TRANSPORT:
      return {
        ...state,
        transports: [...state.transports, action.payload],
      };
    case UPDATE_TRANSPORT:
      return {
        ...state,
        transports: state.transports.map((transport) =>
          transport.id === action.payload.id ? action.payload : transport
        ),
      };
    case DELETE_TRANSPORT:
      return {
        ...state,
        transports: state.transports.filter(
          (transport) => transport.id !== action.payload
        ),
      };
    case SET_DATA:
      return {
        ...state,
        data: action.payload,
        loading: false,
        error: null,
      };
    case SET_LOADING:
      return {
        ...state,
        loading: action.payload,
        error: null,
      };
    case SET_ERROR:
      return {
        ...state,
        loading: false,
        error: action.payload,
      };
    default:
      return state;
  }
};
export default transportReducer;
```

utilReducer.jsx

```
import {
  SET_ERROR,
  SET_LOADING,
  SET_PREDICTIONS,
  SET_DATA_STRUCTURE,
```

```
  SET_TRAIN,
  SET_FILE_DATA,
} from '../actions/utils';
const initialState = {
  trainData: null,
  predictions: null,
  dataStructure: null,
  uploadResult: null,
  data: null,
  isLoading: false,
  error: null,
};
const utilReducer = (state = initialState, action) => {
  switch (action.type) {
    case SET_LOADING:
      return {
        ...state,
        isLoading: action.payload,
      };
    case SET_ERROR:
      return {
        ...state,
        error: action.payload,
      };
    case SET_PREDICTIONS:
      return {
        ...state,
        predictions: action.payload,
      };
    case SET_DATA_STRUCTURE:
      return {
        ...state,
        dataStructure: action.payload,
      };
    case SET_TRAIN:
      return {
        ...state,
        trainData: action.payload,
      };
    case SET_FILE_DATA:
      return {
        ...state,
        uploadResult: action.payload,
      };
    default:
      return state;
  }
};
export default utilReducer;
```

RouteList.jsx

```
import { memo } from 'react';
```

```
import { useSelector } from 'react-redux';
import RouteItem from './RouteItem';
const RouteSecondList = () => {
  const { routes, error, isLoading } = useSelector((state) => state.routes);
  return (
    <div className='p-4'>
      <h1 className='text-2xl font-bold mb-4 text-lime-500'>Routes and Their Transports</h1>
      <div className='grid grid-cols-8 gap-4 bg-green-800 py-2 px-4 rounded-t-lg text-white'>
        <div className='font-bold'>Route ID</div>
        <div className='font-bold'>Transport Name</div>
        <div className='font-bold'>Date</div>
        <div className='font-bold'>Start Location</div>
        <div className='font-bold'>End Location</div>
        <div className='font-bold'>Departure Time</div>
        <div className='font-bold'>Arrival Time</div>
        <div className='font-bold'>Fare Amount</div>
      </div>
      {routes && routes.length > 0 ? (
        <div>
          {routes.map((route) => (
            <RouteItem key={route.route_id} route={route} />
          ))}
        </div>
      ) : (
        <div>No data available</div>
      )}
    </div>
  );
};
export default memo(RouteSecondList);
```

RouteItem.jsx

```
const RouteItem = ({ route }) => {
  const
{route_id,transport_name,date,start_location,end_location,start_time,end_time,fare_amount,far
e_amount_currency}= route
  return (
    <div className="grid grid-cols-8 gap-4 py-2 px-4 border-b">
      <div>{route_id}</div>
      <div>{transport_name}</div>
      <div>{date}</div>
      <div>{start_location}</div>
      <div>{end_location}</div>
      <div>{start_time}</div>
      <div>{end_time}</div>
      <div>
      {fare_amount_currency}{fare_amount}
      </div>
    </div>
  );
};
export default RouteItem ;
```

RouteSecondList.jsx

```
import { memo } from 'react';
import { useSelector } from 'react-redux';
import RouteItem from './RouteItem';
const RouteSecondList = () => {
  const { routes, error, isLoading } = useSelector((state) => state.routes);
  return (
    <div className="">
      <div className='grid grid-cols-8 gap-4 bg-green-800 py-2 px-4 rounded-t-lg text-white'>
        <div className='font-bold'>Route ID</div>
        <div className='font-bold'>Date</div>
        <div className='font-bold'>Start Location</div>
        <div className='font-bold'>End Location</div>
        <div className='font-bold'>Departure Time</div>
        <div className='font-bold'>Arrival Time</div>
        <div className='font-bold'>Fare Amount</div>
        <div className='font-bold'>Transport Id</div>
      </div>
      {routes && routes.length > 0 ? (
        <div>
          {routes.map((route) => (
            <RouteItem key={route.route_id} route={route} />
          ))}
        </div>
      ) : (
        <div>No data available</div>
      )}
    </div>
  );
};
export default memo(RouteSecondList);
```

RouteItem.jsx

```
const RouteItem = ({ route }) => {
  const
{route_id,transport_id,date,start_location,end_location,start_time,end_time,fare_amount,fare_a
mount_curency}= route
  return (
    <div className="grid grid-cols-8 gap-4 py-2 px-4 border-b">
      <div>{route_id}</div>
      <div>{date}</div>
      <div>{start_location}</div>
      <div>{end_location}</div>
      <div>{start_time}</div>
      <div>{end_time}</div>
      <div>
      {fare_amount_curency}{fare_amount}
      </div>
      <div>{transport_id}</div>
    </div>
);};
export default RouteItem ;
```

TransportList.jsx

```
import { memo, useEffect, useState } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import ErrorComponent from '../../layouts/Layout1/ErrorComponent';
import { addTransport, fetchTransports } from '../../store/actions/transports';
import TransportCard from './TransportCard';
import Button from '../UI/Button';
const TransportList = () => {
  const dispatch = useDispatch();
  const { data, error, isLoading } = useSelector((state) => state.transports);
  console.log(useSelector((state) => state));
  console.log(data);
  useEffect(() => {
    dispatch(fetchTransports());
  }, [dispatch]);
  return (
    <div className='p-4'>
      <div className='flex justify-between items-center'>
        <h1 className='text-2xl font-bold text-lime-500'>Transports</h1>
        {/* <Button text='Add Transport' onClick={dispatch(addTransport)} />*/}
      </div>
      <hr className=' my-2' />
      {isLoading && <div>Loading...</div>}
      {error && <div>Error: {error}</div>}
      <div className='grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-4'>
        {data && data.length > 0 ? (
          data.map((transport) => (
            <TransportCard key={transport.id} transport={transport} />
          ))
        ) : (
          <div>No data available</div>
        )}
      </div>
    </div>
  );
};
export default memo(TransportList);
```

TransportCard.jsx

```
import { useDispatch } from "react-redux";
import { deleteTransport, updateTransport } from "../../../store/actions/transports";
import Button from "../../UI/Button";
import { useNavigate } from "react-router-dom";
const TransportCard=({ transport })=>{
        const {id, name, capacity, fuel_consumption, company_tax, vehicle_type, fuel_type,
maintenance_cost } = transport;
        /*const dispatch=useDispatch()*/
        const navigate = useNavigate();
        const handleCardClick=()=>{
                navigate(`/transports/${id}`)
        }
        return(
```

```
                <div className="bg-white shadow-md rounded-lg p-4 m-2 transition transform
hover:scale-105 cursor-pointer" onClick={handleCardClick}
   >
     <h2 className="text-xl font-bold text-green-400">{name}</h2>
                        <hr className=" my-2"/>
     <p ><strong>Vehicle Type:</strong> {vehicle_type}</p>
     <p><strong>Fuel Type:</strong> {fuel_type}</p>
     <p><strong>Capacity:</strong> {capacity}</p>
     <p><strong>Fuel Consumption:</strong> {fuel_consumption} L/100km</p>
     <p><strong>Company Tax:</strong> {company_tax}</p>
     <p><strong>Maintenance Cost:</strong> {maintenance_cost}</p>
                        <hr className=" my-2"/>
                        {/* <div  className="mt-4 flex space-x-2"><Button text='Update'
onClick={dispatch(updateTransport)} /><Button text='Delete'
onClick={dispatch(deleteTransport)} className={'bg-red-500 hover:bg-red-600'}/></div> */}
   </div>
       )
}
export default TransportCard
```

TransportSecondList.jsx

```
import { memo, useEffect } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { fetchTransports } from '../../store/actions/transports';
import TransportItem from './TransportItem';
const TransportList = () => {
  const dispatch = useDispatch();
  const { data, error, isLoading } = useSelector((state) => state.transports);
  useEffect(() => {
    dispatch(fetchTransports(1, 10));
  }, [dispatch]);
  return (
    <div className="">
      <div className='grid grid-cols-5 bg-green-800 py-4 px-4 rounded-t-lg text-white'>
        <div className='font-bold'>Transport ID</div>
        <div className='font-bold'>Name</div>
        <div className='font-bold'>Capacity</div>
        <div className='font-bold'>Fuel Consumption</div>
        <div className='font-bold'>Vehicle Type</div>
      </div>
      {isLoading ? (
        <div>Loading...</div>
      ) : error ? (
        <div>Error: {error}</div>
      ) : data && data.length > 0 ? (
        <div>
          {data.map((transport) => (<TransportItem key={transport.id} transport={transport} />))}
        </div>
      ) : (
        <div>No data available</div>)}
</div>);};
export default memo(TransportList);
```

TransportItem.jsx

```
const TransportItem = ({ transport }) => {
const { id, name, capacity, fuel_consumption, vehicle_type } = transport;
  return (
    <div className="grid grid-cols-5 gap-4 py-2 px-4 bg-white border-b border-gray-200">
      <div>{id}</div>
      <div>{name}</div>
      <div>{capacity}</div>
      <div>{fuel_consumption}</div>
      <div>{vehicle_type}</div>
    </div>
  );
};
export default TransportItem;
```

Button.jsx

```
const Button = ({ onClick, text, className = '' }) => {
  return (
    <button
      className={`bg-green-500 hover:bg-green-600 text-white font-bold py-2 px-4 rounded
transition duration-300 ease-in-out ${className}`}
      onClick={(e) => {
        e.stopPropagation();
        onClick();
      }}
    >
      {text}
    </button>
  );
};
export default Button;
```

LoadDataComponent.jsx

```
import { useState } from 'react';
import { useDispatch } from 'react-redux';
import {
  fetchSetAndTrainData,
  fetchPredict,
} from '../../../store/actions/utils';

const LoadFileComponent = ({ isVisible, onClose, onAct }) => {
  const dispatch = useDispatch();
  const [file, setSelectedFile] = useState(null);
  const handleFileChange = (e) => {
    setSelectedFile(e.target.files[0]);
  };
  const fileToData = async (file) => {
    const fileExtension = file.name.split('.').pop().toLowerCase();
    switch (fileExtension) {
      case 'json':
        return file.text().then(JSON.parse);
```

```
      case 'csv':
        return file.text().then((text) => {
          const [headers, ...rows] = text
            .split('\n')
            .map((row) => row.split(','));
          return rows.map((row) =>
            Object.fromEntries(row.map((cell, index) => [headers[index], cell]))
          );
        });
      case 'txt':
        return file.text().then((text) => {
          return text.split('\n').map((line) => ({ line }));
        });
      default:
        return { error: 'File not acceptable format: json, txt, csv' };
    }
  };
  const handleUpload = async () => {
    if (!file) {
      console.error('No file selected');
      return;
    }
    const data = await fileToData(file);
    if (data.error) {
      alert(data.error);
      return;
    }
    if (onAct === 'train') {
      dispatch(fetchSetAndTrainData(data));
    } else if (onAct === 'predict') {
      dispatch(fetchPredict(data));
    }
    onClose();
  };
  if (!isVisible) return null;
  return (
    <div className='fixed inset-0 flex items-center justify-center bg-black bg-opacity-50'>
      <div className='bg-white p-6 rounded-lg shadow-lg w-1/2'>
        <h2 className='text-2xl font-bold mb-4'>Load File</h2>
        <input type='file' className='mb-4' onChange={handleFileChange} />
        <div className='flex justify-end'>
          <button
            className='bg-red-500 hover:bg-red-600 text-white font-bold py-2 px-4 rounded mr-2'
            onClick={onClose}>
            Close
          </button>
          <button
            className='bg-green-500 hover:bg-green-600 text-white font-bold py-2 px-4 rounded'
            onClick={handleUpload}>
            Upload
          </button></div></div></div>);};
export default LoadFileComponent;
```

ErrorComponent.jsx

```
const ErrorComponent = (props) => {
  const{error}=props
  return (
    <div className='bg-gray-100 text-center flex h-full flex-col justify-center items-center'>
       <p className='text-8xl font-extrabold text-green-500'>{error.code}</p>
       <p className="text-4xl font-medium text-gray-800">{error.message}</p>
       <a href="/" className="mt-4 text-xl text-green-600 hover:underline">Go back home</a>
    </div>);};
export default ErrorComponent;
```

ErrorComponent2.jsx

```
import { useRouteError } from "react-router-dom";
const ErrorComponent = () => {
  const error = useRouteError();
  const { status, statusText, data, message } = error;
  return (
    <div className="bg-gray-100 text-center flex h-full flex-col justify-center items-center
p-6">
      <div className="max-w-md">
        <p className="text-8xl font-extrabold text-green-500">{status || 'Error'}</p>
        <p className="text-4xl font-medium text-gray-800 mt-4">{statusText || 'Something went
wrong'}</p>
         {message && <p className="text-lg text-gray-600 mt-2">{message}</p>}
         {data && <p className="text-lg text-gray-600 mt-2">{data}</p>}
        <a href="/" className="mt-4 text-xl text-green-600 hover:underline">Go back home</a>
      </div>
    </div>
  );
};
export default ErrorComponent;
```

Footer.jsx

```
const Footer = () => {
  return (
    <div className=' w-full flex flex-col items-center bg-zinc-30 text-center text-surface
bg-emerald-600 dark:text-white shadow-lg'>
      <div className='w-full bg-black/5 p-4 text-center '>
        @2024 Copyright: Created by Timurka
      </div>
    </div>
  );
};
export default Footer;
```

Header.jsx

```
const Header = () => {
  return (
    <header className="bg-emerald-500 text-white p-4 shadow-md">
      <div className="container mx-auto flex justify-between">
        <h1 className="text-3xl font-bold">Transports Routes</h1>
```

```
        <nav>
          <ul className="flex space-x-4 font-medium">
            <li><a href="/" className="block py-2 px-3 hover:text-gray-300">Home</a></li>
            <li><a href="/transports" className="block py-2 px-3
hover:text-gray-300">Transports</a></li>
            <li><a href="/routes" className="block py-2 px-3
hover:text-gray-300">Routes</a></li>
            <li><a href="/predict" className="block py-2 px-3 hover:text-gray-300">Train and
Predict</a></li>
          </ul>
        </nav>
      </div>
    </header>
  );
};
export default Header;
```

LoadComponent.jsx

```
const LoadComponent = () => {
  return (
    <div>
      <div className='bg-gray-200 w-full min-h-screen flex justify-center items-center'>
        <div className='flex min-h-screen w-full items-center justify-center bg-gray-200'>
          <div className='flex items-center justify-center w-16 h-16 rounded-full animate-spin
border-y-2 border-solid border-green-500 border-t-transparent'>
          </div>
        </div>
      </div>
    </div>
  );
};
export default LoadComponent;
```

Layout1.jsx

```
import Footer from './Footer';
import Header from './Header';
const Layout_1 = ({ children }) => {
  return (
    <div className='flex flex-col h-screen'>
      <Header />
        <div className='flex-1'>{children}</div>
      <Footer />
    </div>
  );
};
export default Layout_1;
```

Home.jsx

```
import Layout from '../../layouts/Layout1';
import backgroundImage from '../../assets/1.jpeg';
import { Link } from 'react-router-dom';
```

```
const HomePage = () => {
  return (
    <Layout>
      <div
        className='flex flex-col items-center justify-center h-screen bg-cover bg-center
opacity-90'
        style={{
          backgroundImage: `url(${backgroundImage})`,
        }}>
        <div className='text-center animate-fadeIn'>
          <h1 className='text-5xl font-bold text-white mb-8'>
            Welcome to Our Website
          </h1>
          <p className='text-lg text-white mb-4'>
            Explore our transport and more
          </p>
          <Link
            to='/transports'
            className='px-6 py-3 bg-green-500 text-white rounded-md hover:bg-green-600
transition duration-300'>
            View Transports
          </Link>
        </div>
      </div>
    </Layout>
  );
};
export default HomePage;
```

PredictPage.jsx

```
import { useEffect, useState } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import TransportSecondList from '../../components/TransportSecondList';
import Button from '../../components/UI/Button';
import LoadFileComponent from '../../components/UI/LoadDataComponent';
import Layout_1 from '../../layouts/Layout1';
import { fetchRoutes } from '../../store/actions/routes';
import {
  fetchDataStructure,
  fetchPredict,
  fetchReTrainData,
  fetchSetAndTrainData,
} from '../../store/actions/utils';
import RouteSecondList from '../../components/RouteSecondList';
const PredictPage = () => {
  const { predictions, dataStructure } = useSelector((state) => state.utils);
  const [isModalVisible, setIsModalVisible] = useState(false);
  const [action, setAction] = useState(null);
  const dispatch = useDispatch();
  useEffect(() => {
    dispatch(fetchDataStructure());
    dispatch(fetchRoutes(1, 5));
```

```
}, [dispatch]);
const handleFetchDataStructure = () => {
  dispatch(fetchDataStructure());
};
const handleReTrainData = () => {
  dispatch(fetchReTrainData());
};
const handleSaveAndTrainDataFile = () => {
  setAction('train');
  setIsModalVisible(true);
};
const handlePredictData = () => {
  setAction('predict');
  setIsModalVisible(true);
};
const closeModal = () => {
  setIsModalVisible(false);
};
const handleCopyStructure = () => {
  navigator.clipboard.writeText(JSON.stringify(dataStructure, null, 2));
};
return (
  <Layout_1>
    <div className='p-4'>
      <hr className='my-4' />
      <h2 className='text-2xl font-bold mb-4 text-lime-500'>
        Training values{' '}
        <div className='text-base float-right'>
          <Button
            text='ReTrain Data'
            onClick={handleReTrainData}
            className='mx-2'/>
          <Button
            text='Save and Train Data File'
            onClick={handleSaveAndTrainDataFile}
            className='mx-2'/>
        </div>
      </h2>
      <hr className='my-2' />
      <div className='mb-4 p-2 bg-white shadow rounded'>
        <h3 className='text-xl font-semibold mb-2 text-emerald-400'>
          Example of route data
        </h3>
        <div className='overflow-x-auto'>
          <RouteSecondList />
        </div>
      </div>
      <div className='mb-4 p-2 bg-white shadow rounded'>
        <h3 className='text-xl font-semibold mb-2 text-emerald-400'>
          Example of transport data
        </h3>
        <div className='overflow-x-auto'>
```

```
          <TransportSecondList />
        </div>
      </div>
      <div className='mb-4 p-4 bg-white shadow rounded'>
        <h3 className='text-xl font-semibold mb-2 text-emerald-400'>
          Predicted Values{' '}
          <div className='text-base float-right'>
            <Button
              text='Predict Data'
              onClick={handlePredictData}
              className='mx-2'/>
          </div>
        </h3>
        <div className='overflow-x-auto'>
          {predictions
            ? predictions.map((item, index) => <div key={index}>{item}</div>)
            : 'No Predictions Available'}
        </div>
      </div>
      <div className='mb-4 p-4 bg-white shadow rounded'>
        <div className='flex justify-between items-center mb-2'>
          <h3 className='text-xl font-semibold text-emerald-400'>
            Data Structure
          </h3>
          <div>
            <Button
              text='Get Data Structure'
              className='mx-2'
              onClick={handleFetchDataStructure}/>
            <Button
              text='Copy Structure'
              className='mx-2'
              onClick={handleCopyStructure}/>
          </div>
        </div>
        <div className='overflow-x-auto'>
          {dataStructure ? (
            <pre>{JSON.stringify(dataStructure, null, 2)}</pre>
          ) : (
            'No Data Structure Available'
          )}
        </div>
      </div>
      {isModalVisible && (
        <LoadFileComponent
          isVisible={isModalVisible}
          onClose={closeModal}
          onAct={action}/>
      )}
    </div>
  </Layout_1>);};
export default PredictPage;
```

RoutePage.jsx

```
import { useDispatch } from 'react-redux';
import Layout_1 from '../../layouts/Layout1';
import { fetchRoutesPasang } from '../../store/actions/routes';
import { useEffect } from 'react';
import RouteList from '../../components/RouteList';
const RoutePage = () => {
  const dispatch = useDispatch();
  useEffect(() => {
    dispatch(fetchRoutesPasang());
  }, [dispatch]);
  return (
    <Layout_1>
      <RouteList/>
    </Layout_1>
  );
};
export default RoutePage;
```

TransportPage.jsx

```
import { useEffect } from "react";
import { useDispatch, useSelector } from "react-redux";
import { useParams } from "react-router-dom";
import { fetchTransportDetails } from "../../store/actions/transports";
import Layout_1 from "../../layouts/Layout1";
import RouteItem from "./RouteItem";
const TransportPage = () => {
  const { id } = useParams();
  const dispatch = useDispatch();
  const { data, loading, error } = useSelector((state) => state.transports);
  useEffect(() => {
    dispatch(fetchTransportDetails(id));
  }, [dispatch, id]);
  if (loading) {
    return <div>Loading...</div>;
  }
  if (error) {
    return <div>Error: {error}</div>;
  }
  if (!data || !data.transport) {
    return <div>No transport data available</div>;
  }
  const { transport, routes } = data;
  return (
    <Layout_1>
      <div className="p-2">
      <hr className="my-2" />
      <h2 className="text-2xl font-bold text-green-400 text-center">{transport.name} </h2>
      <hr className="my-2" />
        <div className="bg-white shadow-md rounded-lg p-2 m-2 text-center">
          <hr className="my-2" />
          <div className="grid grid-cols-2">
```

```
        <p><strong>Vehicle Type:</strong> {transport.vehicle_type}</p>
        <p><strong>Fuel Consumption:</strong> {transport.fuel_consumption} L/100km</p>
        <p><strong>Fuel Type:</strong> {transport.fuel_type}</p>
        <p><strong>Company Tax:</strong> {transport.company_tax}</p>
        <p><strong>Capacity:</strong> {transport.capacity}</p>
        <p><strong>Maintenance Cost:</strong> {transport.maintenance_cost}</p>
        </div>
        <hr className="my-2" />
       </div>
       <hr className="my-2" />
       <h2 className="text-2xl font-bold my-4 text-emerald-300 text-center">Transport
Routes</h2>
       <hr className="my-2" />
       <div className="grid grid-cols-7 gap-4 bg-green-800 py-4 px-4 rounded-t-lg
text-white">
        <div className="font-bold">Route ID</div>
        <div className="font-bold">Date</div>
        <div className="font-bold">Start Location</div>
        <div className="font-bold">End Location</div>
        <div className="font-bold">Start Time</div>
        <div className="font-bold">End Time</div>
        <div className="font-bold">Fare Amount</div>
       </div>
       {routes && routes.length > 0 ? (
        routes.map((route) => (
          <RouteItem key={route.route_id} route={route} />
        ))
       ) : (
        <div>No routes available</div>
       )}
     </div>
   </Layout_1>
  );
};
export default TransportPage;
```

RouteItem.jsx

```
const RouteItem = ({ route }) => {
  const
{route_id,date,start_location,end_location,start_time,end_time,fare_amount,fare_amount_cure
ncy}= route
  return (
   <div className="grid grid-cols-7 gap-4 py-2 px-4 border-b">
    <div>{route_id}</div>
    <div>{date}</div>
    <div>{start_location}</div>
    <div>{end_location}</div>
    <div>{start_time}</div>
    <div>{end_time}</div>
    <div>
    {fare_amount_curency}{fare_amount}
    </div>
```

```
    </div>
  );
};
export default RouteItem ;
```

TransportsPage.jsx

```
import { useDispatch } from 'react-redux';
import Layout_1 from '../../layouts/Layout1';
import {
  fetchTransports
} from '../../store/actions/transports';
import { useEffect } from 'react';
import TransportList from '../../components/TransportList';
const TransportsPage = () => {
  const dispatch = useDispatch();
  useEffect(() => {
    dispatch(fetchTransports());
  }, [dispatch]);
  return (
    <Layout_1>
      <TransportList/>
    </Layout_1>
  );
};
export default TransportsPage;
```

main.jsx

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import './index.css'
ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite + React</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```