МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

НА ТЕМУ

«Інформаційна система вибору маршруту автотранспортного сполучення»

Галузь знань **12 «Інформаційні технології»**

Спеціальність **122 «Комп'ютерні науки»**

Освітня програма «**Прикладне програмування»**

Освітній рівень: бакалавр

Виконав: студент 4 курсу, групи ПП-41

_____Герасименко О.А._____
(прізвище та ініціали)

Керівник _____Зайцев Є.О._____
(прізвище та ініціали)

_____ д.т.н., с.н.с..._____
(науковий ступінь, звання)

Унікальність тексту 91%

Випускна кваліфікаційна робота бакалавра допущена до захисту
рішенням кафедри *прикладних інформаційних систем*
Протокол № 14 від 23 травня 2023р.
Зав. кафедри_____ Плескач В.Л.

Київ – 2023

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

№з/п	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	14.10.2022	виконано
2.	Видача завдання кваліфікаційної роботи бакалавра	24.10.2022	заява
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	31.10.2022	виконано
4.	Затвердження плану кваліфікаційної роботи бакалавра	01.11.2022	виконано
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	08.11.2022	виконано
6.	Підготовка і подання науковому керівнику першого варіанту I розділу роботи	21.12.2022	виконано
7.	Підготовка і подання науковому керівнику першого варіанту II розділу роботи	31.01.2023	виконано
8.	Підготовка і подання науковому керівнику першого варіанту III розділу роботи	30.03.2023	виконано
9.	Подання роботи у першому варіанті	28.04.2023	виконано
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	03.05.2023	виконано
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	22.05.2023	виконано
12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	26.05.2023	виконано
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	12.06.2023	виконано
14.	Захист кваліфікаційної роботи бакалавра	26.06.2023	виконано

Здобувач вищої освіти _____
(підпис)

Керівник _____
(підпис)

ВІДОМІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Складові частини кваліфікаційної роботи бакалавра	Обсяг, арк.
Титульний аркуш	1
Календарний план кваліфікаційної роботи бакалавра	1
Відомість кваліфікаційної роботи бакалавра	1
Анотація	1
Анотація (іноземною мовою-англійською)	1
Зміст	2
Перелік скорочень, умовних позначень, термінів	1
Вступ	3
1	9
2	21
3	14
Висновки	2
Перелік використаних джерел	3
Додатки	4

				ДП ХХХХ 00.000.00		
	ПІБ	Підп.	Дата			
Розробн.	Герасименко О.А.			Відомість кваліфікацій-ної роботи бакалавра	Лист	Листів
Керівн.	Зайцев Є.О.					
Н/контр.	Макаренко С.А.		26.05.2023			
Зав.каф.	Плескач В.Л.					

АНОТАЦІЯ

Дипломна робота: 64 с., 5 рис., 2 табл., 26 джерел, 4 дод.

Ця дипломна робота присвячена проектуванню та розробленню програмної системи вибору маршруту автотранспортного сполучення.

Метою дипломної роботи є розробка ефективної інформаційної системи вибору маршруту автотранспортного сполучення.

Для досягнення поставленої мети треба вирішити такі завдання:

- здійснити аналіз існуючих систем автоматизації вибору маршруту автотранспортного сполучення;
- здійснити аналіз програмно-технологічних рішень побудови інформаційних систем вибору маршруту автотранспортного сполучення;
- спроектувати, реалізувати, впровадити інформаційну систему вибору маршруту автотранспортного сполучення з урахуванням інженерії вимог;

Об’єкт дослідження.

Процеси вибору маршруту автотранспортного сполучення.

Предмет дослідження.

Програмно-технічні, принципи, підходи щодо побудови програмної інформаційної системи вибору маршруту автотранспортного сполучення.

Методи дослідження.

Аналіз аналогічних програмних систем та методології їх побудови, застосування їх переваг та вдалих рішень та. Із застосуванням інструментів сучасних технологій та використання їх у комплексі для розробки програмної системи, яка відповідатиме головним вимогам та потребам користувачів.

Ключові слова: інформаційна система, маршрут, мікросервіси, архітектура, алгоритм Дейкстри.

ABSTRACT

Thesis: 64 pages, 5 figures, 2 tables, 26 sources, 4 appendices.

This thesis is devoted to the design and development of a software system for choosing a road transport route.

The purpose of the thesis is to develop an effective information system for choosing a road transport route.

To achieve this goal you need to solve the following tasks:

- analyze the existing route selection automation systems road transport;
- carry out an analysis of software and technological construction solutions information systems for choosing a road transport route;
- design, implement, implement the information system of choice road transport route taking into account engineering requirements

Object of study.

Information system for choosing a road transport route.

Subject of study.

Software and technical, principles, approaches to the construction of a software information system for choosing a road transport route.

Research methods.

Analysis of similar software systems and their construction methodology, application of their advantages and successful solutions and. With the application of tools of modern technologies and their use in a complex to develop a software system that will meet the main requirements and needs of users.

Keywords: information system, route, microservices, architecture, Dijkstree's algorithm.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

ІТ – Інформаційні технології

БД – База даних

СУБД – Система управління базами даних

DAO – Data access object

DTO – Data Transfer Object

GPS – Global Positioning System

AI – Artificial intelligence

API – Application Programming Interface

JVM – Java Virtual Machine

ВСТУП

За останні роки великі міста стали сильно перенасиченими транспортними засобами, що призводить до затримок, перевантаження доріг та інших проблем. Також, пасажирам часто доводиться витрачати значну кількість часу на пошук необхідного маршруту та розкладу руху транспорту.

В розвинених країнах інформаційні системи вибору маршруту автотранспортного сполучення стали невід'ємною частиною щоденного життя. Вони дозволяють користувачам швидко та зручно знаходити найкоротший шлях, враховуючи розклад руху транспорту та можливість пересадок, а також отримувати актуальну інформацію про затримки та інші події, які можуть вплинути на рух транспорту.

Актуальність цієї теми зумовлено тим, що розробка інформаційної системи вибору маршруту автотранспортного сполучення може вирішити проблеми транспортного сполучення та покращити якість життя людей. Така система може стати важливим інструментом для організації транспортного руху та підвищення його ефективності. Отже, тема є досить актуальною та має значний потенціал для подальшого дослідження та розробки.

Метою кваліфікаційної роботи бакалавра є розробка алгоритму пошуку оптимального маршруту автотранспортного сполучення на базі веб-застосунку.

Завдання дослідження:

− здійснити аналіз існуючих систем автоматизації вибору маршруту автотранспортного сполучення;

− здійснити аналіз програмно-технологічних рішень побудови інформаційних систем вибору маршруту автотранспортного сполучення;

− спроектувати, реалізувати, впровадити інформаційну систему вибору маршруту автотранспортного сполучення з урахуванням інженерії вимог;

Об'єктом дослідження є процеси вибору маршруту автотранспортного сполучення.

Предметом дослідження кваліфікаційної роботи бакалавра є програмно-технічні, організаційні засади, принципи, підходи щодо побудови програмної інформаційної системи вибору маршруту автотранспортного сполучення.

Методи дослідження: аналіз аналогічних програмних систем та методології їх побудови, застосування їх переваг та вдалих рішень у власному проекті та, за можливістю, виправлення присутніх недоліків. Із застосуванням інструментів сучасних технологій та використання їх у комплексі для розробки програмної системи, яка відповідатиме головним вимогам та потребам користувачів.

Практичне значення одержаних результатів:

- Розробка інформаційної системи вибору маршруту автотранспортного сполучення, яка буде забезпечувати ефективний та оптимальний вибір маршруту автотранспортного сполучення для користувачів.
- Підвищення рівня задоволеності користувачів автотранспортним сполученням, що відбувається через підвищення комфорту, швидкості та ефективності вибору маршруту.
- Підвищення комфорту, швидкості та ефективності вибору маршруту, зменшення витрат користувачів на автотранспортне сполучення.
- Покращення екологічної ситуації, що відбувається через оптимальне розподілення пасажиропотоків та зменшення забруднення повітря.
- Розвиток сучасних інформаційних технологій та інформаційно-комунікаційних систем, що може забезпечити створення нових робочих місць та сприяти розвитку економіки.

Структура роботи:

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, розподілених на підрозділи та висновку.

РОЗДІЛ 1 АНАЛІЗ ІСНУЮЧИХ СИСТЕМ АВТОМАТИЗАЦІЇ ВИБОРУ МАРШРУТУ АВТОТРАНСПОРТНОГО СПОЛУЧЕННЯ

1.1 Огляд систем автоматизації вибору маршруту.

1.1.1 Початок розвитку систем автоматизації вибору маршруту

Системи автоматизації вибору маршруту розвивалися поступово протягом багатьох років. Початок цього розвитку можна пов'язати з винайденням комп'ютерів та розвитком технологій інформаційної обробки даних. Починаючи з 1960-х років компанія General Motors започаткувала проект по створенню системи навігації, що використовувала технології GPS та комп'ютеризованої картографії. Цей проект був запущений для використання військовими, але згодом він був перетворений на цивільний проект і названий NAVSTAR.

У 1990-х роках з'явилися перші навігаційні системи для автомобілів, такі як TomTom та Garmin. Вони використовували GPS технологію та дозволяли користувачам вибирати оптимальний маршрут на основі актуальної інформації про трафік.

Зараз системи автоматизації в вибору маршруту продовжують розвиватися швидко, завдяки новим технологіям та підходам, таким як штучний інтелект та машинне навчання. Сучасні системи можуть пропонувати різні маршрути залежно від пріоритетів користувача, таких як найшвидший маршрут, маршрут з найменшою кількістю пересадок або маршрут, що уникне дорожніх пригод або заторів. Є також інтерактивні системи, які можуть працювати з різними видами транспорту, такими як автомобілі, велосипеди, електричні самокати, громадський транспорт та пішоходи. Вони можуть підказувати оптимальний маршрут в залежності від обраного виду транспорту, а також враховувати фактори, такі як доступність транспорту для людей з інвалідністю або необхідність перевезення велосипеда.

Застосування сучасних технологій, таких як сенсорні пристрої, машинне навчання та аналіз даних, дозволяють системам автоматизації вибору маршруту стати все більш точними та ефективними. Наприклад, деякі системи можуть прогнозувати трафік на дорогах та побудувати маршрути з урахуванням цих прогнозів.

Все більша кількість компаній, що надають послуги транспорту, також використовують системи автоматизації вибору маршруту для підвищення ефективності свого бізнесу та поліпшення досвіду користувачів.

1.1.2 Сучасні системи автоматизації вибору маршруту

Сучасні системи автоматизації вибору маршруту базуються на різних технологіях та алгоритмах, що дозволяють враховувати багато факторів при побудові маршруту. Основними факторами, які враховуються при побудові маршруту, є час поїздки, відстань, кількість пересадок, наявність заторів, погода та інші фактори, які можуть вплинути на поїздку.

Штучний інтелект та машинне навчання дозволяють системам автоматизації вибору маршруту вчитися на основі великої кількості даних та робити прогнози щодо трафіку на дорогах, часу поїздки та інших параметрів. Це дозволяє системам враховувати не тільки поточну ситуацію на дорогах, але і передбачити майбутні зміни трафіку та побудувати оптимальний маршрут з урахуванням цих змін.

Крім того, системи автоматизації вибору маршруту можуть бути інтегровані з іншими додатками та сервісами, такими як системи онлайн-замовлення таксі, забезпечуючи зручну та швидку транспортну послугу для користувачів.

До найпопулярніших сучасних систем автоматизації вибору маршруту відносять:

- Google Maps - це додаток для мобільних пристроїв, який дозволяє користувачам знайти найкоротший та найшвидший маршрут для подорожі на

різних видів транспорту. Google Maps дозволяє знайти маршрут на автомобілі, велосипеді, громадському транспорті або пішки. Додаток показує трафік, можливість об'їзду та розраховує приблизний час прибуття. Google Maps також містить інформацію про наявність ресторанів, готелів та інших місць відпочинку в районах, де користувачі шукають маршрут.

- Waze - це додаток, який дозволяє користувачам знайти найоптимальніший маршрут на автомобілі з урахуванням трафіку та інших факторів. Waze використовує дані, які надають інші користувачі додатку, щоб у режимі реального часу надавати інформацію про затори на дорогах, аварії та інші перешкоди на маршруті. Додаток також має функції повідомлення про радари, швидкість руху та інші корисні функції для водіїв.

- Citymapper - це додаток, який пропонує оптимальні маршрути на різних видів транспорту в більш ніж 40 містах світу. Citymapper використовує відкриті дані про розклади громадського транспорту, а також враховує інформацію про затори на дорогах та інші фактори, що можуть вплинути на час подорожі. Додаток також має функції для знайомства з місцевими транспортними системами, такі як поїздка на велосипеді або на електросамокаті.

- Rome2rio - це онлайн-сервіс, який допомагає користувачам знайти найоптимальніший транспортний засіб для подорожі між двома місцями. Rome2rio дозволяє знайти маршрути на автомобілі, громадському транспорті, літаком та іншими видами транспорту. Сервіс також показує вартість подорожі та дозволяє забронювати квитки на потрібний транспортний засіб.

- Moovit - це додаток, який дозволяє знайти найкоротший маршрут на громадському транспорті в більш ніж 3000 містах світу. Moovit використовує відкриті дані про розклади громадського транспорту та інші фактори, що можуть

вплинути на час подорожі. Додаток також надає користувачам повідомлення про зміни у розкладі громадського транспорту та інші корисні функції.

- TripIt - це додаток, який дозволяє користувачам організовувати свої подорожі та планувати маршрути. TripIt дозволяє додавати інформацію про бронювання готелів, квитки на літаки та інші транспортні засоби. Додаток автоматично складає розклад подорожі та надає користувачам інформацію про час вильоту/прибуття, транспортні засоби та місце перебування на кожному етапі подорожі.

Всі ці системи автоматизації вибору маршруту допомагають користувачам швидко та зручно знайти найкращий маршрут для подорожі, враховуючи різні фактори, такі як трафік, час подорожі та вартість. Кожен з цих додатків має свої унікальні функції та особливості, але вони всі спрямовані на полегшення подорожей та економію часу та коштів.

Наприклад, Google Maps і Waze мають велику базу користувачів та використовують дані про трафік у реальному часі для надання користувачам найкращих маршрутів. Це особливо корисно великим містам зі складним рухом транспорту. Citymapper спеціалізується на громадському транспорті та має детальні мапи транспортних мереж у різних містах світу, що дозволяє знайти найбільш ефективний маршрут з використанням різних видів громадського транспорту.

Система автоматизації вибору маршруту для велосипеда або електросамоката, наприклад, Citymapper, може бути особливо корисною для велосипедистів та електросамокатів. Вона може допомогти знайти найкращий маршрут з використанням велосипедних доріжок та велосипедних прокатів, а також надає інформацію про електрозаправки для електросамокатів.

Rome2rio мають більш широкий спектр транспортних засобів, що дозволяє знайти оптимальний маршрут для будь-якої подорожі. Окрім того, Rome2rio дозволяє забронювати квитки на потрібний транспортний засіб.

Moovit допомагає знайти найкоротший маршрут на громадському транспорті, а також надає корисну інформацію про розклади та зміни у розкладі громадського транспорту, що дозволяє користувачам бути в курсі подорожі.

TripIt дозволяє організувати та спланувати подорожі, включаючи додавання інформації про квитки на різні види транспорту, готелі, ресторани та інші місця для відвідування. Вона може автоматично імпортувати дані з електронної пошти та додатків для бронювання, щоб забезпечити повну інформацію про подорож.

Кожна з цих систем має свої переваги та особливості, тому вибір найкращої залежить від потреб користувача та його конкретної подорожі.

1.2 Аналіз переваг та недоліків існуючих систем

Кожна з систем автоматизації вибору маршруту має свої переваги та недоліки. Розглянемо їх більш детально.

Google Maps

Переваги:

- Велика база даних з картами та детальною інформацією про маршрути.
- Легкість використання та інтуїтивний інтерфейс.
- Автоматичне попередження про трафік та можливі затори.
- Широкий спектр функцій, включаючи пошук ресторанів, готелів, бензоколонок тощо.

Недоліки:

- Можливі неточності в даних про транспортні засоби, зокрема громадський транспорт.

- Не завжди оптимально вибирає маршрут, особливо в умовах заторів та дорожньої ремонту.
- Не завжди має актуальну інформацію про транспорт в режимі реального часу.

Waze

Переваги:

- Актуальна інформація про трафік та можливі затори в режимі реального часу.
- Повідомлення про патрульні поліцейські та радари.
- Інтеграція з соціальними мережами, що дозволяє ділитися інформацією про дорожні умови з іншими користувачами.

Недоліки:

- Не завжди має актуальну інформацію про громадський транспорт та його розклад.
- Може запропонувати неоптимальний маршрут з метою уникнення заторів, що може зайняти більше часу.

Citymapper

Переваги:

- Спеціалізована на громадському транспорті, тому надає детальну інформацію про розклади, маршрути та пересадки.
- Інтеграція з популярними сервісами для бронювання квитків на транспорт.

Недоліки:

- Недоступна у всіх містах світу

Moovit

Переваги:

- Спеціалізована на громадському транспорті, тому надає детальну інформацію про розклади, маршрути та пересадки.
- Актуальна інформація про трафік та можливі затори в режимі реального часу.
- Можливість додавати улюблені маршрути та станції.
- Інтеграція з популярними сервісами для бронювання квитків на транспорт.

Недоліки:

- Не має інформації про маршрути автомобільного транспорту.

Rome2rio

Переваги:

- Має широкий охоплюваний регіон та можливість вибору різних видів транспорту, включаючи автомобільний транспорт, поїзди, авіаційний транспорт та міський транспорт.
- Надає детальну інформацію про маршрут, включаючи пересадки, час в дорозі та вартість проїзду.
- Має можливість знайти альтернативні маршрути та порівняти їх вартість та час в дорозі.

Недоліки:

- Не завжди містить актуальну інформацію про розклади та ціни, тому варто перевіряти інформацію на офіційних сайтах перевізників.

- Не має можливості бронювання квитків на транспорт.

TripIt

Переваги:

- Має можливість автоматичного створення подорожі на основі електронних квитків та підтверджень бронювання.
- Інтегрується з іншими сервісами, такими як Airbnb, Uber та інші, для створення повної картини подорожі.
- Можливість створення додаткових нотаток та планів для кожної зупинки у подорожі.

Недоліки:

- Не надає детальної інформації про транспортні маршрути та можливості пересадок.
- Має певну вартість та обмеження у безкоштовному плані.

1.3 Висновки з аналізу існуючих систем

Після аналізування шести систем автоматизації вибору маршруту - Google Maps, Waze, Citymapper, Rome2rio, Moovit та TripIt, ми можемо зробити наступні висновки:

Google Maps та Waze - це в першу чергу системи навігації для водіїв з можливістю вибору маршруту, орієнтовані на користувачів із приватним транспортом. Вони забезпечують широкий вибір функцій, включаючи оновлення в режимі реального часу, інформацію про трафік та затори, та можуть працювати безкоштовно.

Citymapper та Moovit - це системи громадського транспорту, що дозволяють користувачам знаходити оптимальний маршрут з використанням громадського

транспорту, включаючи автобуси, трамваї та метро. Вони пропонують додаткові функції, такі як карти міст, відстеження руху транспорту та планування поїздок.

Rome2rio та TripIt - це системи, які допомагають користувачам знаходити оптимальний маршрут для подорожей. Rome2rio використовує різні види транспорту, включаючи повітряний, залізничний та автобусний, щоб знайти найкращий маршрут. TripIt допомагає користувачам планувати свої подорожі, додаючи резервації для перельотів, готелів та оренди автомобілів.

Всі ці системи досить важкі та оперують величезною кількістю даних. Але навіть так вони не завжди можуть надати інформацію про маршрут автотранспортного сполучення у маленьких містах чи селах. Саме тому моя система має можливість завантаження карт і датасету з інформацією про автобусні маршрути для маленьких міст, що дозволить людям зручно і швидко планувати свої маршрути. Також, через невелику кількість даних вона буде досить легка та при правильній реалізації буде дуже швидко працювати.

РОЗДІЛ 2 ПРОГРАМНО-ТЕХНОЛОГІЧНІ РІШЕННЯ ПОБУДОВИ ІНФОРМАЦІЙНИХ СИСТЕМ ВИБОРУ МАРШРУТУ АВТОТРАНСПОРТНОГО СПОЛУЧЕННЯ

2.1 Технологічні рішення для розробки інформаційних систем

2.1.1 Методології розробки програмного забезпечення

Методології розробки програмних продуктів є певний набір підходів, правил і принципів, які дозволяють організувати та провести процес розробки програмного продукту найбільш ефективним та результативним способом.

Існує безліч різних методологій розробки програмного забезпечення, кожна з яких має свої особливості та підходи до процесу розробки. Деякі з найбільш поширених методологій розробки програмного забезпечення включають:

• Водоспадна модель - це традиційний підхід до розробки програмного забезпечення, який передбачає послідовне виконання фаз проектування, розробки, тестування, впровадження та підтримки.

• Agile – це методологія, яка заснована на гнучкому підході до розробки, приділяючи велику увагу взаємодії та комунікації між учасниками команди розробки та замовником. Agile методології включають Scrum, Kanban і XP.

• DevOps - це методологія, яка поєднує розробку та експлуатацію програмного забезпечення в одну єдину команду, приділяючи велику увагу автоматизації та безперервній інтеграції та розгортанню.

• Lean - це методологія, яка сфокусована на мінімізації витрат та максимізації доданої вартості у процесі розробки програмного забезпечення.

Кожна з цих методологій має свої переваги та недоліки, і вибір методології залежить від багатьох факторів, включаючи характеристики проекту, потреби замовника, можливості команди тощо.

2.1.3 Хмарні технології

Cloud-технології стали необхідною частиною сучасних інформаційних систем. Їх використання може допомогти зменшити витрати на обладнання та програмне забезпечення, а також підвищити доступність та надійність інформаційних систем.

Однією з основних переваг використання cloud-технологій є можливість зберігання та обробки даних у хмарі. Це означає, що компанії можуть зберігати великі обсяги даних без необхідності утримувати великий обсяг обчислювальних ресурсів на місцевості. Крім того, вони можуть користуватися послугами аналізу даних та машинного навчання, які забезпечують провайдери cloud-технологій.

Cloud-технології також можуть бути використані для забезпечення доступу до інформаційних систем з будь-якого місця а та в будь-який час. Це дозволяє користувачам отримувати доступ до своїх даних та програмного забезпечення з будь-якого місця, де є Інтернет-підключення, що дуже зручно для дистанційної роботи або співпраці між віддаленими командами.

Інша перевага використання cloud-технологій полягає у зменшенні ризиків втрати даних. Оскільки дані зберігаються в хмарі, вони автоматично резервуються на віддалених серверах, що дозволяє зменшити ризики випадкової втрати даних через технічні збої або природні катастрофи.

Крім того, cloud-технології можуть бути використані для підвищення масштабованості інформаційної системи. Якщо компанія зростає та збільшується обсяг обробки даних, провайдери cloud-технологій можуть швидко надати додаткові ресурси для розширення функціональності системи.

Проте, використання cloud-технологій може мати і деякі недоліки, такі як залежність від стороннього провайдера, можливість порушення безпеки даних та зниження продуктивності при обробці великих обсягів даних. Тому, перед використанням cloud-технологій, необхідно ретельно оцінити всі переваги та

недоліки, які пов'язані з їх застосуванням, і вибрати провайдера з доброю репутацією та надійною системою захисту даних.

2.1.4 AI та машинне навчання

Штучний інтелект (AI) та машинне навчання є двома важливими аспектами інформаційних систем. AI відноситься до розумних алгоритмів, які дозволяють розпізнавати, аналізувати та вирішувати проблеми на основі великої кількості даних. Машинне навчання - це один з методів реалізації AI, який використовує статистичні алгоритми для пошуку закономірностей та побудови прогностичних моделей на основі даних.

AI та машинне навчання можуть бути використані в інформаційних системах для різних цілей. Давайте розглянемо кілька способів, як ці технології можуть бути застосовані:

- Обробка даних - AI та машинне навчання можуть бути використані для аналізу великих обсягів даних, включаючи структуровані та неструктуровані дані. Можна використовувати алгоритми машинного навчання, щоб знайти патерни та залежності в даних, які можуть бути важливі для прийняття рішень.

- Пошук інформації - AI може бути використаний для поліпшення пошуку та обробки інформації в системах. Наприклад, система може використовувати алгоритми навчання з підкріпленням (reinforcement learning) для відповіді на запити користувачів, або для прогнозування наслідків прийняття рішень.

- Автоматичне прийняття рішень - AI може бути використаний для прийняття рішень на основі аналізу даних. Наприклад, система може використовувати алгоритми навчання з підкріпленням для автоматичного прийняття рішень в складних бізнес-ситуаціях.

- Аналіз ризиків - AI може бути використаний для аналізу ризиків у різних бізнес-сценаріях. Наприклад, система може використовувати алгоритми машинного навчання для прогнозування ризиків при прийнятті рішень, або для визначення найкращих стратегій управління ризиками.

2.2 Архітектура системи

2.2.1 Опис архітектури системи вибору маршруту автотранспортного сполучення.

Основою архітектури системи вибору маршруту автотранспортного сполучення найчастіше є клієнт-серверна архітектура.

Клієнт-серверна архітектура - це модель розподіленої системи, що передбачає наявність двох основних компонентів: клієнта та сервера. Клієнтська частина взаємодіє з користувачем та здійснює запити до серверної частини, а серверна частина відповідає на запити та забезпечує роботу всієї системи.

У цій архітектурі, сервер зазвичай виконує більш складні функції, тоді як клієнт забезпечує взаємодію з користувачем та відображення даних, що надходять від сервера. Клієнтська частина може бути реалізована у вигляді програмного забезпечення на комп'ютері або мобільному пристрої, веб-браузері або будь-якій іншій формі інтерфейсу користувача.

Серверна частина може бути фізично розміщена в будь-якому місці і забезпечує функціональні можливості, необхідні для роботи системи. Вона зазвичай здатна обробляти одночасно багато запитів від клієнтів та забезпечує захист від несанкціонованого доступу до даних.

Клієнт взаємодіє з сервером за допомогою API. API (Application Programming Interface) - це набір протоколів, структур даних та інструкцій, які визначають, як різні програми можуть взаємодіяти між собою. API визначає які

операції можуть бути виконані, які дані можуть бути передані та які формати даних повинні бути використані для комунікації між програмами. API зазвичай використовується через мережеві протоколи, такі як TCP/IP або HTTP, що дозволяє передавати дані та команди між клієнтом та сервером.

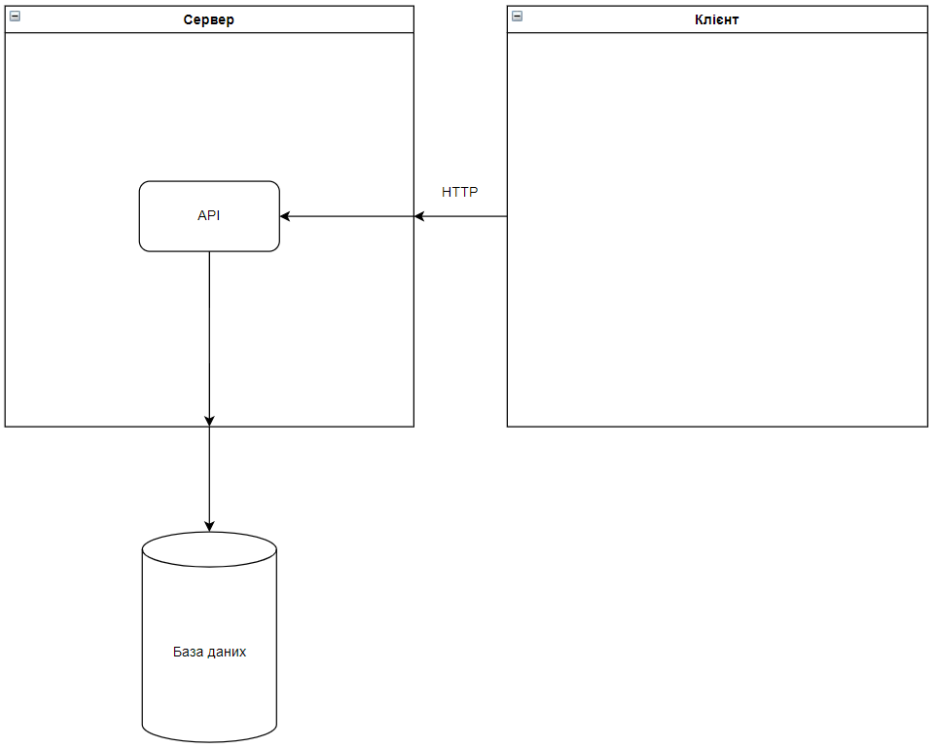

Рисунок 2.1 - Схема клієнт-серверної архітектури

Однак, архітектура системи вибору маршруту автотранспортного сполучення може бути складною та мати багато підмодулів.

2.2.2 Використання мікросервісної архітектури

Мікросервісна архітектура - це підхід до розробки програмного забезпечення, в якому програма складається з невеликих і незалежних компонентів, які називаються мікросервісами. Кожен мікросервіс виконує конкретну функцію та має свій власний інтерфейс та базу даних. Мікросервісна архітектура дозволяє швидко масштабувати та змінювати програмне

забезпечення, зменшує залежність між компонентами та полегшує розробку та тестування окремих частин системи.

У випадку системи вибору маршруту автотранспортного сполучення, мікросервісна архітектура може бути корисною для декількох компонентів системи, зокрема:

Клієнтські:

- Модуль взаємодії з користувачем: цей компонент взаємодіє з користувачем та отримує від нього вхідні дані, такі як місце відправлення та призначення, бажаний час відправлення та інші параметри. Крім того, цей модуль може надавати користувачеві інформацію про розклади та маршрути транспорту, а також результати пошуку оптимального маршруту.

- Модуль оплати: цей компонент відповідає за всі процедури оплати. Він може включати в себе інтеграцію з платіжними системами та іншими сервісами для зручності користувачів.

Серверні:

- Модуль вибору оптимального маршруту: цей компонент відповідає за аналіз доступних маршрутів та вибір оптимального, враховуючи різні фактори, такі як час поїздки, кількість пересадок та вартість квитка. Для цього можуть використовуватися різні алгоритми.

- База даних маршрутів та розкладів руху транспорту: цей компонент зберігає інформацію про всі доступні маршрути транспортного сполучення, їх тривалість та розклад руху. Ці дані можуть оновлюватися в режимі реального часу, щоб система завжди мала актуальну інформацію.

- Модуль оновлення даних: цей компонент відповідає за оновлення бази даних маршрутів та розкладів руху транспорту в режимі реального часу. Він може

отримувати дані про затримки, розклади руху та інші оновлення від постачальників інформації.

- Модуль моніторингу та аналітики: цей компонент відповідає за моніторинг роботи системи та збір аналітичних даних. Це може включати в себе моніторинг часу роботи системи, кількість користувачів та їх поведінку, відгуки та інші метрики, які можуть бути використані для покращення роботи системи.

- Модуль безпеки: цей компонент відповідає за забезпечення безпеки та захисту даних користувачів. Це може включати в себе захист від хакерських атак, захист від шахрайства та інших видів злочинної діяльності, а також захист конфіденційної інформації користувачів.

- Модуль інтеграції з іншими системами: цей компонент відповідає за інтеграцію системи вибору маршруту автотранспортного сполучення з іншими сервісами та системами, такими як системи міської інфраструктури, такі як метро, трамваї та автобуси, або сервіси таксі та оренди авто.

Ці компоненти розташовані на різних рівнях архітектури системи та взаємодіють між собою за допомогою різних інтерфейсів та протоколів. У деяких випадках, система вибору маршруту автотранспортного сполучення може бути інтегрована з іншими сервісами, такими як додатки для мобільних пристроїв, щоб забезпечити більш зручний та простий доступ для користувачів.

Схема взаємодії модулів мікросервісної архітектури інформаційної системи вибору маршруту автотранспортного сполучення:

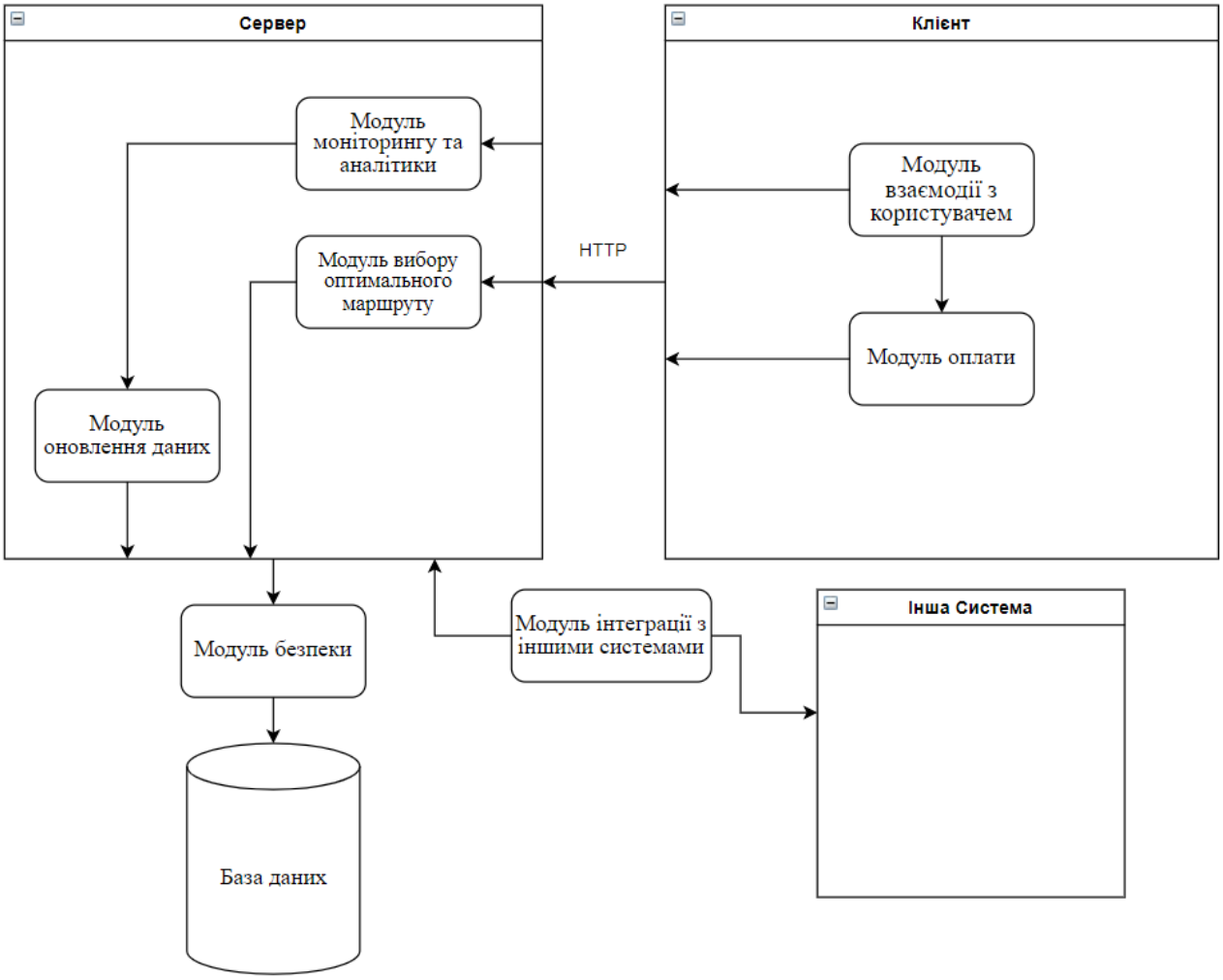

Рисунок 2.2 - Модулі мікросервісної архітектури

2.3 Вибір баз даних та інструментів розробки

2.3.1 Вибір СУБД для зберігання даних системи вибору маршруту

СУБД (Система управління базами даних) - це програмне забезпечення, яке дозволяє зберігати, організовувати та керувати доступом до бази даних. СУБД надає можливість створення, зміни та видалення даних з бази даних, а також роботу з запитами до даних та забезпечує контроль доступу до даних з боку користувачів. СУБД можуть бути реляційними, коли дані зберігаються у вигляді

таблиць зі зв'язками між ними, або нереляційними, коли дані зберігаються у вигляді наборів документів або ключ-значення пар.

Тип СУБД вибирають по запитам системи, але можна відокремити два найпопулярніші типи:

- Реляційні бази даних
- Нереляційні бази даних

2.3.1.1 Реляційні бази даних

Реляційні бази даних — це тип бази даних, яка організовує дані в таблиці з рядками та стовпцями. Кожна таблиця в реляційній базі даних представляє певний тип сутності або концепції, наприклад клієнтів, замовлень або продуктів. Стовпці в таблиці представляють атрибути або властивості цієї сутності, наприклад назву, адресу або ціну.

Реляційні бази даних використовують структуровану мову запитів (SQL) для керування та обробки даних. SQL дозволяє користувачам створювати, читати, оновлювати та видаляти дані в базі даних. Це також дозволяє користувачам встановлювати зв’язки між таблицями, які можна використовувати для зв’язування пов’язаних даних і забезпечення узгодженості даних.

У реляційній базі даних таблиці можуть бути пов’язані одна з одною за допомогою первинних і зовнішніх ключів. Первинний ключ — це стовпець або набір стовпців, які однозначно ідентифікують кожен рядок у таблиці. Зовнішній ключ — це стовпець або набір стовпців, які посилаються на первинний ключ в іншій таблиці. Цей зв’язок між таблицями використовується для встановлення посилальної цілісності, яка гарантує послідовність і точність даних у пов’язаних таблицях.

Реляційні бази даних широко використовуються в різноманітних програмах, включаючи фінансові системи, управління запасами та управління взаємовідносинами з клієнтами. Вони мають високу масштабованість і можуть ефективно обробляти великі обсяги даних. Однак їх може бути складно розробити та підтримувати, а для ефективного використання можуть знадобитися спеціальні навички та знання.

2.3.1.2 Нереляційні бази даних

Нереляційні бази даних (NoSQL) — це тип бази даних, призначений для обробки неструктурованих і напівструктурованих даних. На відміну від реляційних баз даних, які організовують дані в таблиці з рядками та стовпцями, бази даних NoSQL використовують різноманітні моделі даних для зберігання та отримання даних. Деякі поширені моделі даних NoSQL включають ключ-значення, документ, сімейство стовпців і графік.

Бази даних ключ-значення зберігають дані як набір пар ключ-значення, де кожен ключ є унікальним і відображається на значення. Ця модель даних проста та ефективна, що робить її корисною для кешування та зберігання даних, до яких часто звертаються.

Бази даних документів зберігають дані як напівструктуровані документи, наприклад JSON або XML. Кожен документ містить набір пар ключ-значення, і документи в колекції можуть мати різні структури. Ця модель даних корисна для обробки даних зі складною змінною структурою, як-от канали соціальних мереж або дані датчиків.

Бази даних сімейства стовпців зберігають дані у вигляді стовпців, а не рядків, і групують стовпці в сімейства стовпців на основі моделей їх використання. Ця модель даних корисна для обробки великих обсягів даних із високою пропускною здатністю запису, наприклад даних журналу.

Бази даних графів зберігають дані як вузли та ребра, де вузли представляють сутності, а ребра представляють зв’язки між сутностями. Ця модель даних корисна для обробки складних зв’язків між даними, такими як соціальні мережі чи системи рекомендацій.

Бази даних NoSQL мають високу масштабованість і можуть ефективно обробляти великі обсяги неструктурованих даних. Вони зазвичай використовуються у веб-додатках, де дані потрібно отримати швидко та з низькою затримкою.

Інформаційна система вибору маршруту автотранспортного сполучення потребую зберігання великої кількості даних про всі зупинки, їх розташування на карті, автобуси та маршрути, затори та ремонти на дорогах.

Оскільки цих даних багато, їх можна структурувати та масштабувати, кращим рішенням буде використовувати реляційну базу даних. За допомогою неї можна буде швидко обробляти великі обсяги даних не втрачаючи зручності використання.

2.3.2 Вибір мов програмування

Вибір мов програмування залежить від вимог до системи, технічних обмежень та відповідності бізнес-потребам. Основними мовами програмування, які можуть бути використані для розробки інформаційної системи, є:

- Java
- C++
- JS
- Python

- Swift

2.3.2.1 Java

Java —об’єктно-орієнтована мова, розроблена таким чином, щоб бути незалежною від платформи, тобто програми Java можуть працювати на будь-якому комп’ютері чи пристрої, який має віртуальну машину Java (JVM).

Java відома своєю філософією «напиши один раз, запусти будь-де», яка дозволяє розробникам написати код один раз і розгорнути його на кількох платформах без змін. Це стало можливим завдяки JVM, яка діє як рівень абстракції між кодом Java і основним обладнанням та операційною системою.

Java використовується для широкого спектру програм, включаючи веб-розробку, розробку мобільних додатків і розробку корпоративного програмного забезпечення. Це універсальна мова, яка відома своєю міцністю, надійністю та функціями безпеки.

Деякі ключові функції Java включають:

- Об'єктно-орієнтована модель програмування
- Збірка сміття, яка автоматично керує виділенням і звільненням пам’яті
- Незалежність від платформи
- Багатопотоковість, яка дозволяє паралельне програмування
- Обробка винятків, яка допомагає запобігти помилкам і збоям у програмах

Java має велике та активне співтовариство розробників, що призвело до створення великої екосистеми бібліотек, фреймворків та інструментів для розробки Java.

2.3.2.2 C++

C++ — це об’єктно-орієнтована мова, яка надає такі функції, як класи, успадкування, інкапсуляція та поліморфізм, що робить її добре придатною для великомасштабної розробки програмного забезпечення.

C++ використовується в широкому діапазоні програм, включаючи системне програмне забезпечення, вбудовані системи, ігри та наукові обчислення. Він відомий своєю ефективністю, продуктивністю та можливістю прямого доступу до обладнання, що робить його популярним вибором для ресурсомістких програм.

Деякі ключові особливості C++ включають:

- Об'єктно-орієнтована модель програмування
- Маніпуляції з пам'яттю низького рівня
- Шаблони, які дозволяють загальне програмування
- Обробка винятків, яка допомагає запобігти помилкам і збоям у програмах
- Стандартна бібліотека шаблонів (STL), яка надає колекцію загальних алгоритмів і структур даних

C++ має велике та активне співтовариство розробників, що призвело до величезної екосистеми бібліотек, фреймворків та інструментів для розробки. Однак C++ може бути складнішим і важчим для вивчення, ніж деякі інші мови програмування, і вимагає хорошого розуміння концепцій програмування та основ інформатики.

2.3.2.3 Python

Python — це інтерпретована мова програмування високого рівня. Вона відома своєю простотою, легкістю використання та читабельністю, що робить її популярним вибором як для початківців, так і для досвідчених розробників.

Також, це об’єктно-орієнтована мова, яка підтримує різні парадигми програмування, включаючи процедурне, функціональне та об’єктно-орієнтоване програмування. Він надає багатий набір вбудованих модулів і бібліотек, а також величезну екосистему сторонніх модулів, які можна легко встановити за допомогою індексу пакетів Python (PyPI).

Деякі ключові функції Python включають:

- Легко читається, з чітким і лаконічним синтаксисом
- Динамічний тип, що дозволяє швидко створювати прототипи та розробляти
- Автоматичне керування пам’яттю, що усуває потребу в ручному розподілі та звільненні пам’яті
- Підтримка між платформами, доступні версії для Windows, macOS, Linux та інших операційних систем
- Інтерпретований, що дозволяє швидко отримувати відгуки та налагодження під час розробки

Python використовується в широкому діапазоні програм, включаючи веб-розробку, аналіз даних, машинне навчання, наукові обчислення тощо. Він має велике й активне співтовариство розробників, результатом якого є величезна екосистема бібліотек, фреймворків та інструментів для розробки Python.

Простота використання та багата екосистема Python роблять його популярним вибором для розробників, які хочуть швидко створювати прототипи

та створювати програми, а його потужні функції та гнучкість роблять його придатним для створення складних і складних програм.

2.3.2.4 JavaScript

JavaScript – це мова програмування високого рівня, яка використовується переважно для створення інтерактивних веб-додатків. JavaScript — це мова на стороні клієнта, яка виконується веб-браузерами, що забезпечує динамічний вміст та інтерактивність на веб-сторінках.

JavaScript є об’єктно-орієнтованою мовою, яка підтримує різні парадигми програмування, включаючи функціональне та імперативне програмування. Він надає багатий набір вбудованих API і бібліотек, а також величезну екосистему сторонніх модулів і бібліотек, які можна легко встановити за допомогою менеджерів пакунків, таких як NPM.

Деякі ключові функції JavaScript включають:

- Модель програмування, керована подіями, яка дозволяє обробляти дані користувача та інші асинхронні події
- Динамічний тип, який забезпечує гнучкість і швидкий розвиток
- Спадкування прототипів, що дозволяє гнучко створювати та повторно використовувати об’єкти
- Закриття, які дозволяють інкапсуляцію та приватні дані у функціях

JavaScript використовується в широкому діапазоні програм, включаючи веб-розробку, серверне програмування, настільні програми, мобільні програми тощо. Його гнучкість і універсальність роблять його популярним вибором для розробників, які хочуть створювати складні та складні програми.

Окрім клієнтського JavaScript, існує також Node.js, яке є середовищем виконання JavaScript на стороні сервера, яке дозволяє розробникам створювати

програми на стороні сервера за допомогою JavaScript. Node.js забезпечує керовану подіями неблокуючу модель вводу-виводу, що робить її добре придатною для створення масштабованих мережевих програм.

2.3.2.5 Висновки

Інформаційна система вибору маршруту автотранспортного сполучення має клієнт-серверну архітектуру, тому треба вибрати мови на ці два архітектурних рівня.

Серверна частина потребує розробку швидкого алгоритму який буде обробляти велику кількості даних щоб правильно генерувати маршрути між двома точками. Досить складно зробити оптимальний алгоритм використовуючи більш високорівневі мови програмування, тому краще за все використовувати C++ для написання сервера. C++ дуже швидкий, може маніпулювати пам'яттю низького рівня та має безліч сучасних бібліотек та фреймворків для написання швидкого та відмовостійкого сервера.

Мова клієнтської частини залежить від платформи на якій вона повинна працювати. Якщо треба веб-застосунок, краще за все використовувати JavaScript, бо на ньому можна дуже швидко на просто написати гарний інтерфейс користувача.

Для мобільних додатків можна використовувати Swift для написання на IOS та Java або Kotlin на Android. Якщо клієнт повинен бути на десктопі, можна розглянути Python.

2.4 Опис алгоритмів і моделей

2.4.1 Алгоритм Дейкстри

Алгоритм Дейкстри — це алгоритм обходу графа, який використовується для пошуку найкоротшого шляху між двома вершинами в графі з невід'ємними вагами ребер. Він був запропонований голландським комп'ютерним науковцем Едсгером В. Дейкстрою в 1956 році і є одним із найбільш часто використовуваних алгоритмів для пошуку найкоротших шляхів у графах.

Алгоритм працює, починаючи з вихідного вузла та ітеративно досліджуючи його сусідні вузли, вибираючи вузол із найменшою відстанню від джерела як наступний вузол для відвідування. Алгоритм підтримує набір відвіданих вузлів і набір невідвіданих вузлів. Спочатку всі вузли є невідвіданими, а їх відстань від вихідного вузла встановлено на нескінченність, за винятком самого вихідного вузла, відстань якого дорівнює 0.

Алгоритм далі виглядає наступним чином:

Нехай вузол, з якого ми починаємо, буде називатися початковим вузлом. Нехай відстань вузла Y буде відстанню від початкового вузла до Y. Алгоритм Дейкстри спочатку почне з нескінченних відстаней і намагатиметься покращити їх крок за кроком.

1. Позначити всі вузли як невідвідані. Створіть набір усіх невідвіданих вузлів, який називається невідвіданим набором.

2. Призначте кожному вузлу умовне значення відстані: встановіть його рівним нулю для нашого початкового вузла та нескінченністю для всіх інших вузлів. Під час роботи алгоритму орієнтовна відстань вузла v — це довжина найкоротшого шляху, знайденого на даний момент, між вузлом v і початковим вузлом. Оскільки спочатку не відомий жоден шлях до будь-якої іншої вершини, крім самого джерела (який є шляхом нульової довжини), усі інші попередні відстані спочатку встановлені на нескінченність. Встановіть початковий вузол як поточний.

3. Для поточного вузла розгляньте всіх його невідвіданих сусідів і обчисліть їхні орієнтовні відстані через поточний вузол. Порівняйте щойно розраховану орієнтовну відстань із тією, яка зараз призначена сусідові, і призначте їй меншу. Наприклад, якщо поточний вузол A позначено відстанню 6, а ребро, що з'єднує його з сусіднім B, має довжину 2, то відстань до B через A буде 6 + 2 = 8. Якщо B раніше було позначено відстань більше 8, потім змініть його на 8. В іншому випадку поточне значення буде збережено.

4. Коли ми закінчимо з розглядом усіх невідвіданих сусідів поточного вузла, позначте поточний вузол як відвіданий і видаліть його з невідвіданого набору. Відвіданий вузол більше ніколи не перевірятиметься (це є дійсним і оптимальним у зв'язку з поведінкою на кроці 6: наступні відвідувані вузли завжди будуть у порядку «спочатку найменша відстань від початкового вузла», тому будь-які відвідування після цього будуть мають більшу відстань).

5. Якщо вузол призначення було позначено як відвіданий (під час планування маршруту між двома певними вузлами) або якщо найменша орієнтовна відстань між вузлами в невідвіданому наборі дорівнює нескінченності (під час планування повного обходу; відбувається, коли немає зв'язку між початковим вузлом і залишилися невідвіданими вузлами), а потім зупиниться. Алгоритм завершено.

6. В іншому випадку виберіть невідвіданий вузол, позначений найменшою орієнтовною відстанню, установіть його як новий поточний вузол і поверніться до кроку 3.

Під час планування маршруту насправді немає необхідності чекати, поки вузол призначення буде «відвіданий», як описано вище: алгоритм може зупинитися, коли вузол призначення матиме найменшу орієнтовну відстань серед усіх «невідвіданих» вузлів (і, таким чином, може бути обраний як наступний «поточний»).

Часова складність алгоритму Дейкстри становить O(|E| + |V| log |V|), де |E| – кількість ребер і |V| кількість вершин у графі. Просторова складність дорівнює O(|V|), яка використовується для зберігання відстаней від вихідного вузла до кожного вузла на графі. Псевдокод алгоритму наданий у Додатку А.

2.4.2 Алгоритм A*

Алгоритм A* - це алгоритм пошуку шляху в графі з вагованими ребрами, який використовує евристичну функцію для швидкого пошуку найкоротшого шляху між двома вузлами. Алгоритм A* може бути використаний для пошуку найбільш ефективного маршруту між двома точками з урахуванням обмежень на шляху, таких як дорожні роботи, пробки та інші.

На кожній ітерації свого основного циклу A* має визначити, який із своїх шляхів продовжити. Це робиться на основі вартості шляху та оцінки вартості, необхідної для продовження шляху до мети. Зокрема, A* вибирає шлях, який мінімізує

$$f(n) = g(n) + h(n), \quad (1)$$

де n — наступний вузол на шляху, g(n) — вартість шляху від початкового вузла до n, а h(n) — евристична функція, яка оцінює вартість найдешевшого шляху від n до мети. A* завершується, коли шлях, який він вибирає для продовження, є шляхом від початку до цілі або якщо немає шляхів, які можна продовжити. Евристична функція є проблемною. Якщо евристична функція допустима, тобто вона ніколи не переоцінює фактичну вартість досягнення мети, A* гарантовано повертає шлях із найменшими витратами від початку до мети.

У типових реалізаціях A* використовується пріоритетна черга для повторного вибору мінімальних (приблизних) вузлів вартості для розширення. Ця

пріоритетна черга відома як відкритий набір, край або межа. На кожному кроці алгоритму вузол із найменшим значенням f(x) видаляється з черги, значення f і g його сусідів відповідно оновлюються, і ці сусіди додаються до черги. Алгоритм продовжується до тих пір, поки вилучений вузол (тобто вузол із найменшим значенням f з усіх периферійних вузлів) не стане цільовим вузлом. [b] Значення f цієї цілі тоді також є вартістю найкоротшого шляху, оскільки h на мета дорівнює нулю в допустимій евристики.

Алгоритм, описаний досі, дає нам лише довжину найкоротшого шляху. Щоб знайти фактичну послідовність кроків, алгоритм можна легко переглянути, щоб кожен вузол на шляху відстежував свій попередник. Після виконання цього алгоритму кінцевий вузол вказуватиме на свого попередника і так далі, поки попередник якогось вузла не стане початковим вузлом.

Якщо евристика h задовольняє додаткову умову $h(x) \leq d(x, y) + h(y)$ для кожного ребра (x, y) графа (де d позначає довжину цього ребра), то h називається монотонні, або послідовні. За допомогою узгодженої евристики A* гарантовано знайде оптимальний шлях без обробки будь-якого вузла більше одного разу, а A* еквівалентно запуску алгоритму Дейкстри зі зниженою вартістю $d'(x, y) = d(x, y) + h(y) - h(x)$

2.4.3 Модель маршрутної оптимізації

Модель маршрутної оптимізації - це математична модель, яка дозволяє знайти найбільш оптимальний маршрут для пересування з одного місця в інше на основі врахування різних обмежень та критеріїв ефективності.

Ця модель може бути використана для оптимізації маршрутів автотранспорту, зокрема для зменшення витрат палива, скорочення часу поїздки та покращення якості обслуговування. Основними етапами моделі маршрутної оптимізації є:

1. Визначення параметрів маршруту - на цьому етапі враховуються різні фактори, такі як відстань, швидкість руху, трафік, наявність перехресть та інші обмеження. Для цього можуть використовуватися дані з датчиків та інших джерел.

2. Створення матриці відстаней та часу - на основі визначених параметрів створюється матриця відстаней та часу між всіма пунктами маршруту.

3. Побудова графу маршруту - на основі матриці відстаней та часу створюється граф маршруту з вузлами і зв'язками між ними.

4. Використання алгоритмів пошуку шляхів - на цьому етапі використовуються алгоритми пошуку шляху, такі як алгоритм Дейкстри або алгоритм A* для знаходження найкоротшого маршруту з урахуванням обмежень.

5. Визначення оптимального маршруту - на основі знайденого маршруту та обмежень визначається оптимальний маршрут для пересування з одного пункту в інший.

6. Аналіз ефективності - на цьому етапі проводиться аналіз ефективності знайденого маршруту та можливих варіантів маршрутів. Цей етап може включати аналіз показників ефективності, таких як витрати палива, час поїздки, кількість пересадок, ризик заторів та інші.

Модель маршрутної оптимізації може бути доповнена різними алгоритмами та методами для забезпечення оптимальної ефективності. Наприклад, можна використовувати методи машинного навчання та аналізу даних для покращення точності прогнозування трафіку, а також для виявлення та управління ризиками заторів та інших непередбачених ситуацій.

Додатково, модель маршрутної оптимізації може бути доповнена моделлю розміщення зупинок автобусів та орієнтування на розклади руху транспорту для досягнення максимальної ефективності та забезпечення пасажиропотоку відповідно до попиту.

Усі ці етапи та методи можуть бути реалізовані в системі вибору маршруту автотранспортного сполучення для забезпечення максимальної ефективності та задоволення потреб користувачів.

2.5 Висновки

Дуже важливо правильно вибрати програмно-технологічні рішень для побудови ефективної інформаційної системи автоматизації вибору маршруту автотранспортного сполучення. Зазвичай подібна система має складну архітектуру з великою кількістю модулів тому краще за все використовувати мікросервісну архітектуру яку можна просто масштабувати та підтримувати. Також зберігання і обробка великої кількості даних потребує структуровану та швидку СУБД. Тому реляційна база даних може бути найкращим вибором.

Крім того, побудова маршруту повинна бути швидкою та точною і для цього гарно підходять алгоритми пошуку пошуку найкоротшого шляху між двома точками: алгоритм Дейкстри та A*, які можна написати на досить складний але ефективній мові програмування C++.

РОЗДІЛ 3 ПРОЕКТУВАННЯ, РОЗРОБЛЕННЯ, РЕАЛІЗАЦІЯ СИСТЕМИ АВТОМАТИЗАЦІЇ ВИБОРУ МАРШРУТУ АВТОТРАНСПОРТНОГО СПОЛУЧЕННЯ

3.1 Інженерія вимог до програмної системи вибору маршруту автотранспортного сполучення

3.1.1 Постановка задачі

Провівши аналіз існуючих систем та вибору маршруту автотранспортного сполучення та програмно-технологічні рішень до побудови подібних систем, виокремивши для себе основні пункти, яких варто дотримуватися та недоліки, яких варто уникати, можемо сформулювати основне завдання цієї роботи: спроектувати та розробити прототип програмної системи автоматизації вибору маршруту автотранспортного сполучення. Систему побудувати за принципом клієнт-серверної та мікросерверної архітектури.

Для виконання поставленої задачі необхідно послідовно вирішити наступні завдання:

- Розробити модуль маршрутизації (сервер побудови маршрутів);
- Розробити модуль доступу до даних (сервер БД) з використанням реляційних баз даних;
- Розробити алгоритм пошуку найкоротшого маршруту між двома точками (алгоритм Дейкстри) та модель маршрутної оптимізації;
- Розробити зручний клієнт-орієнтований інтерфейс взаємодії з користувачами;
- Розробити функціонал загрузки користувальницьких карт.
- Забезпечити на ресурсі головні вимоги до системи автоматизації вибору маршруту та впровадити на сайті необхідний функціонал.

При побудові системи дотримуватися основних вимог та рекомендацій, наведених в попередніх розділах роботи, щоб забезпечити максимально ефективну та зручну систему.

3.1.2 Функціональні можливості проектованої системи

Проектована програмна система передбачає реалізацію фундаментального функціоналу, який є затребуваним на платформах автоматизації вибору маршруту автотранспортних сполучень. Для виокремлення функціональних можливостей, які треба було реалізувати на ресурсі було проаналізовано декілька платформ зі схожою тематикою, такі як GoogleMaps та Waze. На основі аналізу наявних в них можливостей, а також знайдених недоліків, було сформульовано перелік функцій, які варто передбачити, та наведено у Таблиці 3.1.

Таблиця 3.1 – Перелік та опис функцій системи

Карта маршрутів	Відображення карти всіх автобусних маршрутів. Кожен автобусний маршрут позначений відповідним кольором та має назву автобусу.
Побудова маршруту	Побудова маршруту з однієї зупинки до іншої. Перелік всіх зупинок відображається в інтерфейсі.

Продовження таблиці 3.1

Деталі маршруту	Перегляд деталей побудованого маршруту, таких як: номери автобусів, на яких потрібно буде їхати, послідовний список всіх зупинок і пересадок, час очікування автобуса, час в дорозі до пересадки, загальний час поїздки, інтерактивний степер з прогресу поїздки
Пошук зупинок	Зручний пошук зупинок за ім'ям.
Автобуси	Перегляд детальної інформації про всі автобуси та їх маршрути.
Зупинки	Перегляд інформації про всі автобусні зупинки
Режим Гість	Дозволяє користуватися системою не маючи облікового запису.
Реєстрація	Додавання користувача за логіном та паролем.
Логін	Вхід користувача в систему за логіном та паролем.
Перегляд попередніх маршрутів	Ця функція доступна тільки зареєстрованим користувачам. Після побудови маршруту він зберігається та його можна знову переглянути.

Усі ці функції є необхідними при реалізації інформаційної системи автоматизації вибору маршруту автотранспортного сполучення.

Під час проектування системи були враховані переваги характеристик з аналізу споріднених за характером та тематикою систем.

3.2 Архітектурні рішення програмної системи вибору маршруту автотранспортного сполучення

3.2.1 Тип архітектури

Після аналізу існуючих систем вибору маршруту автотранспортного сполучення було зазначено, що клієнт-серверна архітектура, яка поділена на мікросервіси є найкращим варіантом через зручність розробки та великий потенціал масштабованості.

Цей тип архітектури характеризується наявністю проміжного програмного забезпечення між клієнтською машиною та сервером даних. Вся логіка даних і бізнес-логіка зберігається в проміжному програмному забезпеченні. Використання проміжного програмного забезпечення підвищує гнучкість і продуктивність розробленої системи: трирівнева архітектура розділена на три рівні - рівень представлення (клієнтський), рівень додатку (бізнес-логіка) і рівень бази даних (дані), як показано на діаграмі розгортання системи (рис. 3.1).

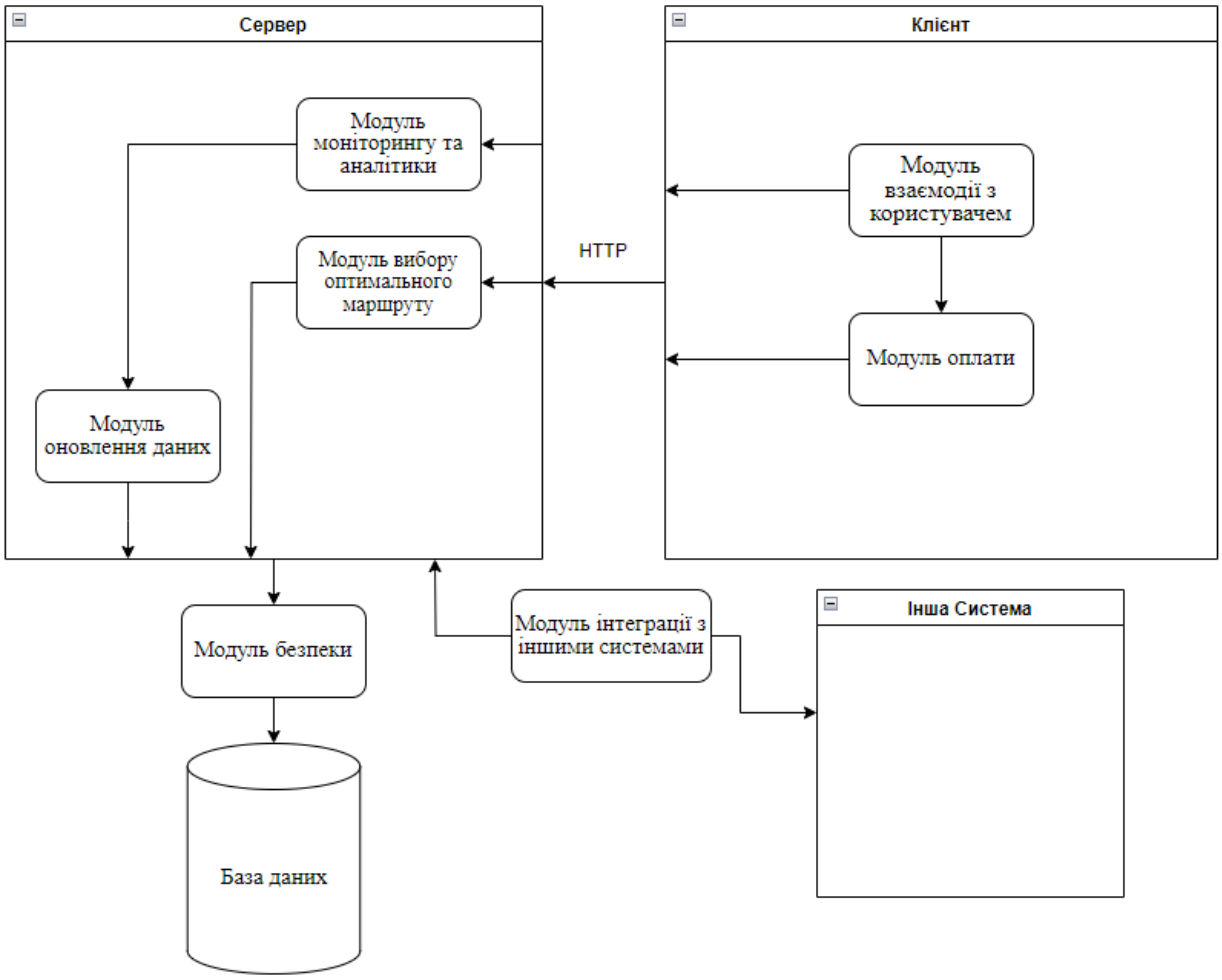

Рисунок 3.1 - Схема модулів клієнт-серверної архітектури

Кожен рівень має декілька модулів, які спілкуються один з одним через різні інтерфейси та виконуюсь усі функції рівня. Клієнти системи можуть отримати доступ до центрального (проміжного) рівня програми шляхом підключення через приватну мережу. Центральним елементом розробленої клієнт-серверної архітектури є веб-сервер, який являє собою високопродуктивну комп'ютерну систему, здатну розміщувати декілька веб-ресурсів. На сервері встановлено програмне забезпечення типу веб-сервера Microsoft IIS, яке забезпечує доступ до розміщених в Інтернеті веб-ресурсів. Сервер підключений до Інтернету за допомогою високошвидкісного з'єднання, що забезпечує високу

швидкість передачі даних. Прикладний рівень взаємодіє з серверною машиною, яка представляє рівень бази даних, за допомогою протоколу передачі даних, надсилаючи запити та отримуючи відповіді, які інтерпретуються проміжним рівнем та надсилаються клієнту. Рівень даних розробленої системи представлений компонентом бази даних MySQL.

3.2.2 Опис структури програми

Структура програми складається з п’яти самостійних модулів, які представлені окремими папками. В кожній папці може буди декілька файлів, в яких реалізований функціонал модуля. Кожен модуль має свій інтерфейс для взаємодії та може працювати з іншими компонентами системи. Опис і перелік всіх модулів зазначений у Таблиці 3.2.

Таблиця 3.2 - Перелік та опис модулів системи

Назва модуля	Опис
Database	Містить набір інтерфейсів взаємодії з базою даних. Перетворює дані з БД в відповідне представлення структури на мові C++.
Engine	Головний модуль сервера, який виконую усю бізнес-логіку. Містить реалізацію побудови графа та карти маршрутів, пошук оптимального маршруту між двома зупинками та має функції обробки всіх серверних запитів.

Продовження таблиці 3.2

Render	Модуль, який відповідає за генерацію SVG зображень, базуючись на даних графа представлень автобусних зупинок та маршрутів. На вихідних SVG файлах може бути зображена карта маршрутів та зупинок або конкретний побудований маршрут.
Utils	Цей модуль реалізовую додатковий функціонал, який використовують всі інші модулі.
Tests	Модуль тестів, які покривають більшість функціоналу сервера. Він допомагає додавати новий функціонал та бачити, якщо старий при цьому зламався.

3.2.3 Архітектура серверної частини

Сервер побудований на базі C++ фреймворку Qt 6.5 з використанням модулів core та httpserver.

Qt - це кросплатформний фреймворк для розробки додатків, який дозволяє створювати графічні інтерфейси користувача (GUI) та виконувати інші завдання, пов'язані з програмуванням, такі як робота з базами даних, мережна взаємодія, багатопоточність та інше.

Qt надає набір інструментів та бібліотек, які спрощують та прискорюють процес розробки програмного забезпечення. Фреймворк дозволяє розробникам писати код один раз і компілювати його для різних операційних систем, таких як Windows, MacOS, Linux, Android та iOS, що робить Qt одним з найбільш популярних виборів для створення кросплатформових додатків.

Для розробки сервера використовувався модуль QHttpServer, який надає можливості для створення HTTP-серверів. Він забезпечує простий і гнучкий спосіб реалізації сервера через класи та методи для обробки вхідних HTTP-запитів та надсилання відповідей на них.

Уся обробка запитів знаходиться в функції main файлу main.cpp та обробляються за допомогою функції route, яка приймає шлях запиту та функцію обробки. Усі вхідні дані потрапляють до класу RouteManager, який через рівень абстракції надає ці дані модулю Engine. Цей рівень абстракції є шаблоном проектування DAO, який використовується для відокремлення логіки доступу до даних від решти програми та надає абстракцію над базою даних або іншим джерелом даних, таким чином, що зміни в самому джерелі даних не впливають на решту програми. Також він використовується для забезпечення простоти та масштабованості програмного забезпечення.

Структура DAO включає наступні елементи:

- Interface/Абстракція DAO: Це інтерфейс або абстрактний клас, який визначає методи, доступні для взаємодії з джерелом даних. Оскільки в мене HTTP сервер, то і реалізовано 4 найпопулярніші функції обробки HTTP запитів: POST, GET, PUT, DELETE.

- Конкретна реалізація DAO: Це клас, який реалізує методи, визначені в інтерфейсі DAO. Він містить логіку доступу до джерела даних (SQL-запити до бази даних). В коді це конкретні унікальні класи, які виконують бізнес-логіку та взаємодіють з модулем Engine.

- Об'єкти моделей(DTO): Це класи, що представляють структуру даних, які можуть бути збережені або витягнуті з джерела даних. Вони використовуються як контейнери для передачі даних між DAO та іншими частинами програми. В моєму випадку це уявлення для зупинок, автобусів, карти та маршруту.

Тобто мій сервер має такий ланцюг перетворення даних с БД уявлення до відображення на клієнтській частині:

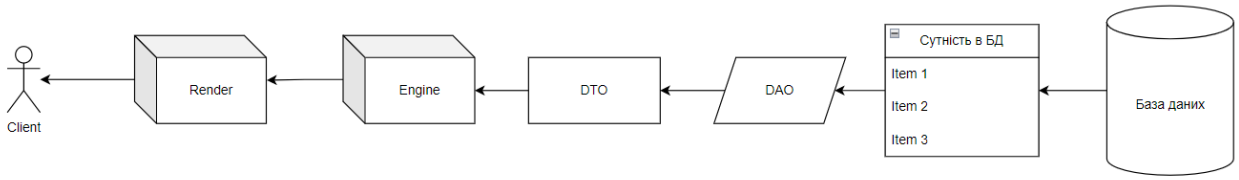

Рисунок 3.2 - Схема перетворення даних

Усі дані, які були перетворені с БД уявлення до кодового зберігаються в стандартних контейнерах класу RouteDatabase. Цей клас містить велику кількість зв’язків між контейнерами та дуже швидко обробляю усі запити до нього. Майже кожна операція має складність O(1) або O(log). Зберігання даних в зручному уявленні в оперативній пам’яті дає можливість дуже швидко та зручно отримувати інформацію. Але такий підхід має мінус – це неможливість оперувати величезною кількістю даних.

Основну роботу сервера виконує клас RouteManager який має реалізацію усіх функцій, які використовуються в DAO. Після завантаження даних цей клас ініціює побудову зваженого графа зупинок, вершини якого є зупинками, а ребра дороги до зупинки. Вагою ребра э відстань від однієї зупинки до іншої. За уявлення графа відповідає клас DirectedWeightedGraph, який містить всю потрібну інформацію про вершини та ребра. З цим класом через композицію взаємодіє клас

Router, який займається побудовою маршруту, видалення його та отриманням усіх вершин графа, які беруть участь у маршруті. Побудова маршруту виконується через реалізацію алгоритму Дейкстра в функції BuildRoute. Усі маршрути, які були побудовані для пошуку оптимального знаходяться в кеші expanded_routes_cache_.

Наступним рівнем є клас Map, який містить дві найважливіші функції. По-перше він будує граф в вигляді карти маршрутів всіх автобусів, який потім можна зручно конвертувати в більш людське уявлення карти для передачі по API. По-друге, він працює з класами DirectedWeightedGraph та Router, щоб побудувати маршрут, перевести його в більш зручне уявлення та додати до нього додаткову інформацію по типу загального часу в дорозі. Після цього він видаляє маршрут з кеша усіх маршрутів для економії пам'яті.

Далі йде рівень реалізації запитів, який використовую класи Map та RouteDatabase. Перелік класів цього рівня: QueryStop, QueryBus, QueryRoute та QueryMap. Усі вони реалізують один інтерфейс класу Query, який містить усього два методи: Process та GetResult. Метод Process виконує усю логіку, а метод GetResult займається серіалізацією результату для передачі по API.

Схема архітектури наведена на Рисунку 3.3.

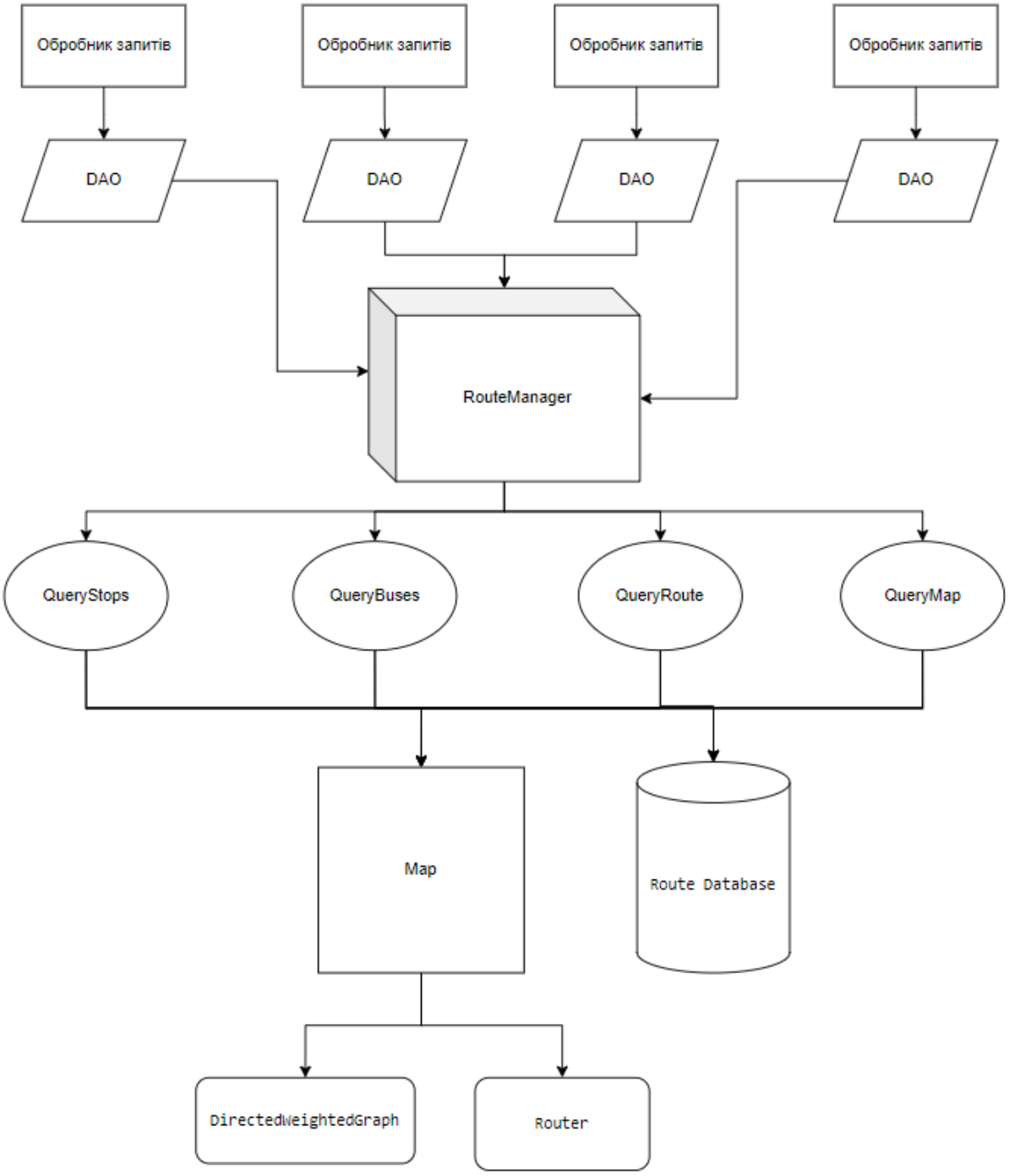

Рисунок 3.3 – Схема архітектури сервера

3.3 Проектування, кодування, реалізація інформаційної системи вибору маршруту автотранспортного сполучення.

3.3.1 Структура візуального представництва системи

Інтерфейс користувача реалізований в браузерному середовищі, тому має вигляд веб-сторінки.

Інтерфейс складається з двох частин: панелі управління та відображення. Панель управління це висувне меню на якому відображено весь основний функціонал системи: побудова маршрутів з можливістю зручного вибору та пошуку потрібних зупинок, кнопки відображення карти автобусних маршрутів та зупинок, перегляд всіх наявних у системі автобусів та інформації про них. Також на ньому розташована історія маршрутів, які будував користувач.

При побудові маршруту на панелі відображення малюється зображення карти, а праворуч від неї інтерактивний степер, який є повною інструкцією з побудованого маршруту.

3.3.2 Технології розробки інтерфейсу системи

Розробка користувальницького дизайну системи здійснена з використанням передового та сучасного стеку фронтенд технологій: TypeScript, Node.js, Next.js, MaterialUI, CSS, HTML.

Next.js є фреймворком для розробки веб-додатків на базі React. Він дозволяє створювати універсальні (ізоморфні) додатки, які працюють як на стороні клієнта, так і на стороні сервера. Next.js надає потужні інструменти для побудови сучасних веб-додатків з функціями, такими як серверний рендерінг (server-side rendering), статичний генерування (static generation), а також можливість побудови односторінкових додатків (SPA).

Інтерфейс був написаний на мові програмування TypeScrip, яка розширює JavaScript, додаючи підтримку статичного типізації. Реалізація головного компонента сторінки наведена в Додатку В.

Покращення вигляду стандартних компонент HTML до стиля Material Design було виконано за допомогою бібліотеки Material-UI, яка використовує плоскість, тіні, анімацію та живі кольори для створення модернізованого та стильного вигляду інтерфейсу. Також вона надає великий набір готових компонентів, таких як кнопки, картки, меню, форми, таблиці, діалогові вікна та багато інших. Ці компоненти мають налаштовані стилі, що відповідають принципам Material Design, і легко інтегруються в проекти з використанням популярних фреймворків.

В інтерфейсі реалізовано стилі на функціонал для таких компонентів: AppBar, Drawer, StopsComboBox, RouteStepper, BusesModalDialog, StopsModalDialog. Увесь функціонал та взаємодія компонентів були побудовані на використанні пропсів та стану, який надає фреймворк React.js. Для спілкування з сервером використовується функція fetch, яка повертає дані в JSON. Ці дані передаються в компоненти за допомогою функцій useState та useEffect. Отже структура клієнтської частини досить проста.

3.3.3 Структура бази даних

В інформаційній системі вибору маршрута автотранспортного сполучення використовується реляційна база даних. Основною концепцією реляційних баз даних є таблиці, що складаються з рядків і стовпців. Кожна таблиця представляє собою колекцію записів або рядків, де кожен рядок містить дані для конкретного об'єкта або елемента. Стовпці таблиці визначають типи даних, які можуть бути збережені в цих рядках. Реляційні бази даних використовують мову структурованих запитів (SQL) для взаємодії з даними. SQL дозволяє виконувати

операції зчитування, запису, оновлення та видалення даних у таблицях. Він також дозволяє визначати структуру таблиць, встановлювати зв'язки між ними та виконувати складні операції об'єднання та агрегування даних.

Для зберігання даних таблиць був обраний програмний додаток MySQL та MySQL Workbench– студія управління доступом до баз даних. Структура таблиць наведена в Додатку Г.

ВИСНОВОК

У результаті виконання кваліфікаційної роботи бакалавра було розроблено інформаційну систему вибору маршруту автотранспортного сполучення, зокрема:

− здійснено аналіз існуючих систем автоматизації вибору маршруту автотранспортного сполучення;

− здійснено аналіз програмно-технологічних рішень побудови інформаційних систем вибору маршруту автотранспортного сполучення;

− спроектовано, реалізовано та впроваджено інформаційну систему вибору маршруту автотранспортного сполучення з урахуванням інженерії вимог;

А саме, було детально проаналізовано шість подібних систем: Google Maps, Waze, Citymapper, Rome2rio, Moovit та TripIt. На основі аналізу дізналися про всі переваги на недоліки цих систем, їх функціонал на точність побудови маршруту. А також, що не існує єдиної універсальної системи, яка би мала весь потрібний функціонал та всі функції виконувала бездоганно.

Також здійснено аналіз програмно-технологічних рішень, в результаті якого маємо, що подібна система зазвичай має складну архітектуру з великою кількістю модулів тому краще за все використовувати клієнт-серверну архітектуру, яка буде розділена на мікросервіси. Також, що треба швидко обробляти велику кількість структурованих даних і для цього ідеально підійде реляційна БД. Написання ефективного алгоритму Дейкстра або A* на швидкому C++ теж є невід’ємною частиною програмно-технологічних рішень.

Для проектування та розробки системи був застосований великий стек технологій. На серверній стороні використовувався C++ як основна мова програмування. Qt з модулем QHttpServer як інструмент для написання саме HTTP сервера та базу даних MySql для зберігання даних. З клієнтського боку

використовувався TypeScript з фреймворком Next.js для написання веб-застосунка та Material-UI для покращення відображення стандартних компонентів HTML.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Проектування Інформаційних систем. *КПІ ім.Ігоря Сікорського*: веб-сайт. URL: https://ela.kpi.ua/bitstream/123456789/33651/1/PIS_KL.pdf (дата звернення: 04.03.2023).
2. Проектування Інформаційних систем. *Черкаський Національний Університет імені Богдана Хмельницького*: веб-сайт. URL: http://eprints.cdu.edu.ua/1481/1/pro.pdf (дата звернення: 06.03.2023).
3. Waze. *Waze*: веб-сайт. URL: https://www.waze.com/ (дата звернення: 07.03.2023).
4. Citymapper. *Citymapper*: веб-сайт. URL: https://citymapper.com/ (дата звернення: 08.03.2023).
5. Rome2rio. *Rome2rio*: веб-сайт. URL: https://www.rome2rio (дата звернення: 09.03.2023).
6. Moovit. *Moovit*: веб-сайт. URL: https://moovitapp.com/ (дата звернення: 10.03.2023).
7. TripIt. *TripIt*: веб-сайт. URL: https://www.tripit.com/web (дата звернення: 11.03.2023).
8. Основні методології розробки ПЗ. *Agile*: веб-сайт. URL: https://agile.yakubovsky.com/ua/2015/10/9-osnovnykh-metodolohiy-rozrobky-prohramnoho-zabezpechennya-agile/ (дата звернення: 12.03.2023).
9. Методології розробки ПЗ. *Харківський національний університет радіоелектроніки*: веб-сайт. URL: https://openarchive.nure.ua/server/api/core/bitstreams/394b3208-65b5-434f-b17d-be063d46f2c6/content (дата звернення: 12.03.2023).
10. Хмарні технології. *edin*: веб-сайт. URL: https://edin.ua/shho-take-xmarni-texnologi%D1%97-i-navishho-voni-potribni/ (дата звернення: 14.03.2023).

11. Хмарні технології. *Дія*: веб-сайт. URL: https://business.diia.gov.ua/cases/tehnologii/so-take-hmarni-tehnologii-i-ak-voni-mozut-dopomogti-vasomu-pidpriemstvu (дата звернення: 14.03.2023).
12. Хмарні обчислення. *Вікіпедія*: веб-сайт. URL: https://uk.wikipedia.org/wiki/%D0%A5%D0%BC%D0%B0%D1%80%D0%BD%D1%96_%D0%BE%D0%B1%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F (дата звернення: 14.03.2023).
13. AI. *IBM*: веб-сайт. URL: https://www.ibm.com/topics/artificial-intelligence (дата звернення: 18.03.2023).
14. AI. *Wikipedia*: веб-сайт. URL: https://en.wikipedia.org/wiki/Artificial_intelligence (дата звернення: 18.03.2023).
15. Архітектура інформаційних систем. *Pidru4niki*: веб-сайт. URL: https://pidru4niki.com/10561127/bankivska_sprava/arhitektura_informatsiynih_sistem (дата звернення: 19.03.2023).
16. Web APIs. *Mozilla*: веб-сайт. URL: https://developer.mozilla.org/ru/docs/Learn/JavaScript/Client-side_web_APIs/Introduction (дата звернення: 19.03.2023).
17. RESTfull API. *Amazon*: веб-сайт. URL: https://aws.amazon.com/ru/what-is/restful-api/ (дата звернення: 21.03.2023).
18. Microservices. *Wikipedia*: веб-сайт. URL: https://en.wikipedia.org/wiki/Microservices (дата звернення: 23.03.2023).
19. Database Management Systems. *Cisco*: веб-сайт. URL: https://www.appdynamics.com/topics/database-management-systems (дата звернення: 24.03.2023).
20. Реляційні бази даних. *Харьковский национальный экономический университет*: веб-сайт. URL: https://pns.hneu.edu.ua/pluginfile.php/601514/mod_resource/content/1/%D0%9B%D0%B5%D0%BA%D1%86%D1%96%D1%8F%203%20%D0%A0%D0%B5%D0%

BB%D1%8F%D1%86%D1%96%D0%B9%D0%BD%D1%96%20%D0%B1%D0%B0%D0%B7%D0%B8%20%D0%B4%D0%B0%D0%BD%D0%B8%D1%85.pdf (дата звернення: 03.04.2023).

21. Dijkstra's Algorithm. *Geeksforgeeks*: веб-сайт. URL: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/ (дата звернення: 06.04.2023).
22. A* Search Algorithm. *Geeksforgeeks*: веб-сайт. URL: https://www.geeksforgeeks.org/a-search-algorithm/ (дата звернення: 10.04.2023).
23. Qt HTTP Server. *Qt Docs*: веб-сайт. URL: https://doc.qt.io/qt-6/qthttpserver-index.html (дата звернення: 12.04.2023).
24. DAO and DTO. *Analyticsvidhya*: веб-сайт. URL: https://www.analyticsvidhya.com/blog/2023/02/what-are-data-access-object-and-data-transfer-object-in-python/ (дата звернення: 16.04.2023).
25. Next.js Docs. *Next.js*: веб-сайт. URL: https://nextjs.org/docs (дата звернення: 20.04.2023).
26. Material UI Docs. *Material UI*: веб-сайт. URL: https://mui.com/material-ui/getting-started/overview/ (дата звернення: 20.04.2023).

ДОДАТКИ

ДОДАТОК А

```
function dijkstra(G, S)
    for each vertex V in G
        distance[V] <- infinite
        previous[V] <- NULL
        If V != S, add V to Priority Queue Q
    distance[S] <- 0

    while Q IS NOT EMPTY
        U <- Extract MIN from Q
        for each unvisited neighbour V of U
            tempDistance <- distance[U] + edge_weight(U, V)
            if tempDistance < distance[V]
                distance[V] <- tempDistance
                previous[V] <- U
    return distance[], previous[]
```

ДОДАТОК Б

```
QHttpServer server;

server.afterRequest([] (QHttpServerResponse &&response) {
    response.setHeader("Access-Control-Allow-Origin", "*");
    response.setHeader("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE, OPTIONS");
    response.setHeader("Access-Control-Allow-Headers", "Content-Type, Authorization");
    return std::move(response);
});

server.route("/upload", QHttpServerRequest::Method::Post, [&manager](const QHttpServerRequest& request) {
     std::istringstream in(request.body().toStdString());
     manager.uploadDatabase(in);
     return QHttpServerResponder::StatusCode::Created;
 });

server.route("/map", [&manager](const QHttpServerRequest&) {
    auto result = QString::fromStdString(manager.getMap());
    auto response = QHttpServerResponse{result};
    response.setHeader("Content-Type", "application/json");
    return response;
});

server.route("/route", [&manager](const QHttpServerRequest &request) {
    if (!request.query().hasQueryItem("from") || !request.query().hasQueryItem("to")) {
        return QHttpServerResponse{QHttpServerResponder::StatusCode::BadRequest};
    }

    const auto from = request.query().queryItemValue("from").toStdString();
    const auto to = request.query().queryItemValue("to").toStdString();

    auto result = QString::fromStdString(manager.getRoute(from, to));

    auto response = QHttpServerResponse{result};
    response.setHeader("Content-Type", "application/json");
    return response;
});
```

Рисунок Б1 – Реалізація HTTP сервера

ДОДАТОК В

```
return (
    <Box sx={{ maxWidth: 400 }}>
        <Stepper activeStep={activeStep} orientation="vertical">
            {props.route.map((step: any, index: Number) => (
                <Step key={step.type == 'Wait' ? `Stop: ${step.stop_name}` : `${step.span_count} stops on bus ${step.bus}`}>
                    <StepLabel
                        optional={<Typography variant="caption">{step.type == 'Wait' ? `Estimated bus waiting time: ${step.time}min.` : `Travel time: ${step.time}min.`}</Typo
                    >
                        {step.type == 'Wait' ? `Stop: ${step.stop_name}` : `${step.span_count} stops on bus ${step.bus}`}
                    </StepLabel>
                    <StepContent>
                        <Box sx={{ mb: 2 }}>
                            <div>
                                <Button
                                    variant="contained"
                                    onClick={handleNext}
                                    sx={{ mt: 1, mr: 1 }}
                                >
                                    {index === props.route.length - 1 ? 'Finish' : 'Continue'}
                                </Button>
                                <Button
                                    disabled={index === 0}
                                    onClick={handleBack}
                                    sx={{ mt: 1, mr: 1 }}
                                >
                                    Back
                                </Button>
                            </div>
                        </Box>
                    </StepContent>
                </Step>
            ))}
        </Stepper>
        {activeStep === props.route.length && (
            <Paper square elevation={0} sx={{ p: 3 }}>
                <Typography>All steps completed - you&apos;re finished</Typography>
                <Button onClick={handleReset} sx={{ mt: 1, mr: 1 }}>
                    Reset
                </Button>
            </Paper>
        )}
    </Box>
);
}
```

Рисунок В1 – Реалізація головного компонента інтерфейсу

ДОДАТОК Г

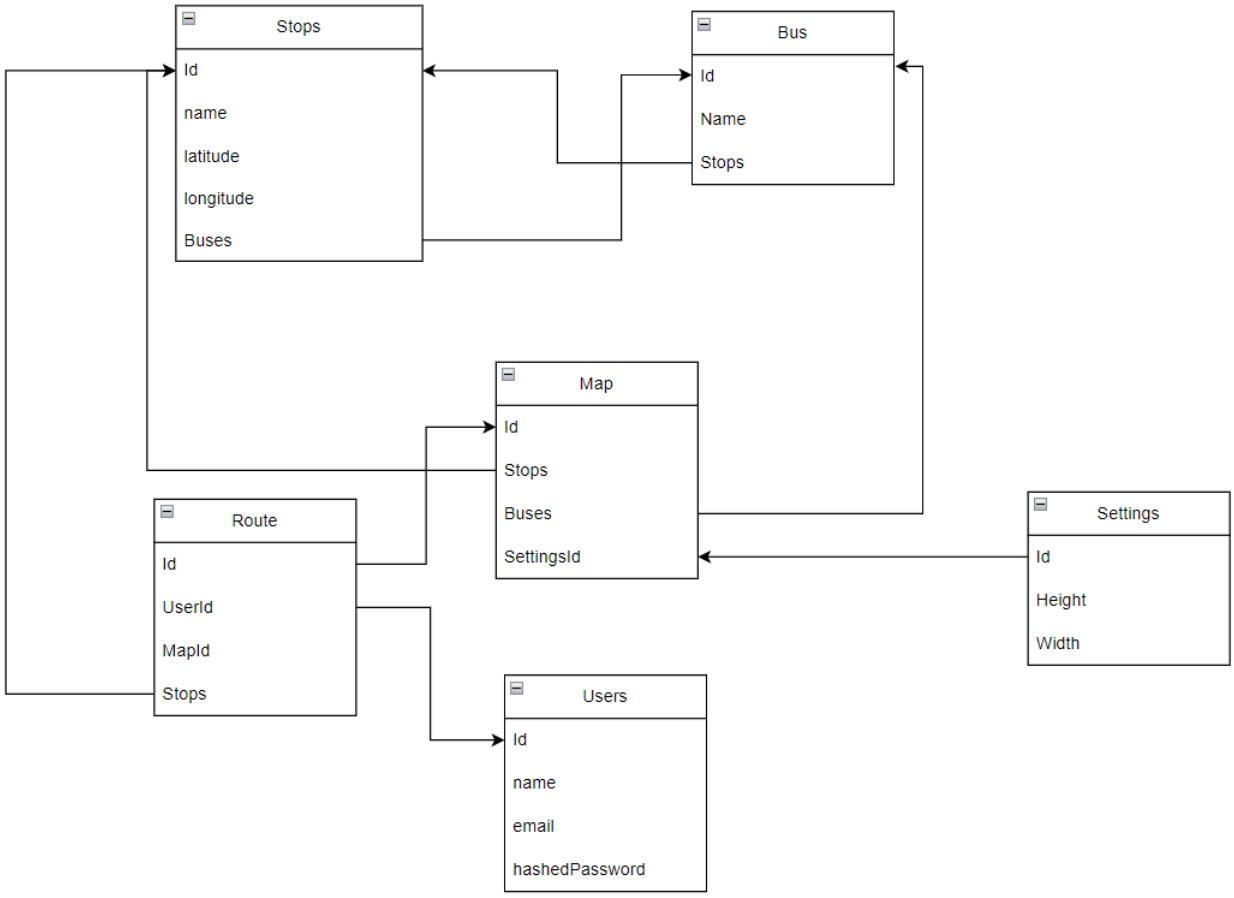

Рисунок Г1 – Схема таблиць БД