Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота

за освітнім ступенем «бакалавр»

на тему:

ІНФОРМАЦІЙНА СИСТЕМА НА ТРАНСПОРТНУ ТЕМАТИКУ, ПОБУДОВАНА З ВИКОРИСТАННЯМ ХМАРНОЇ БАЗИ ДАНИХ

Виконав:
здобувач IV курсу
групи КН-41
спеціальності 122 «Комп’ютерні науки»
Король Максим Миколайович

Науковий керівник:
к.т.н. Шевцова Н.В.

Рівне – 2024

ЗМІСТ

ВСТУП

Актуальність дослідження. В сучасних умовах, практично в усіх сферах людської діяльності використовуються інформаційні системи, що ґрунтуються на використанні найновіших інформаційних технологій, зокрема хмарних обчисленнях. Використання хмарних сервісів для роботи з базами даних забезпечує масштабованість та високу доступність БД, при цьому провайдер бере на себе значну частину функцій по її обслуговуванню, що спрощує розробку та обслуговування інформаційної системи.

Сучасні транспортні системи є складними та багатогранними структурами, які потребують ефективного управління інформацією для забезпечення безперебійного функціонування. Зростання обсягів перевезень, розвиток логістики та необхідність в оперативному прийнятті рішень вимагають впровадження новітніх технологій для обробки даних. Хмарні бази даних є одним із перспективних напрямків у цій галузі, оскільки вони дозволяють зберігати та обробляти великі обсяги інформації з високою швидкістю та надійністю. Використання хмарних технологій у транспортній системі може значно підвищити її ефективність, зменшити витрати та покращити якість обслуговування клієнтів. Таким чином, дослідження та розробка інформаційної системи на транспортну тематику з використанням хмарної бази даних є актуальною та важливою задачею.

Мета дослідження: розробка інформаційної системи для управління транспортними процесами з використанням хмарної бази даних, що забезпечить ефективне зберігання, обробку та аналіз даних, необхідних для функціонування транспортної системи.

Для досягнення поставленої мети необхідно вирішити наступні **завдання**:

1. Провести аналіз існуючих рішень в галузі інформаційних систем для транспорту.
2. Вибрати оптимального хмарного провайдера серед таких як Google Cloud, Amazon Web Services (AWS) та Microsoft Azure.

3. Розробити архітектуру інформаційної системи на транспортну тематику з використанням хмарної бази даних.
4. Реалізувати вебдодаток для управління транспортними процесами на основі Node.js та MongoDB.
5. Провести тестування та оцінку ефективності розробленої системи.

Об’єкт дослідження: інформаційні системи для управління транспортними процесами.

Предмет дослідження: використання хмарних баз даних для зберігання та обробки інформації в транспортних системах.

У процесі дослідження були використані наступні **методи**:

1. Аналіз літературних джерел та існуючих рішень.
2. Порівняння та вибір хмарних провайдерів.
3. Проектування та розробка інформаційної системи.
4. Програмування вебдодатку на основі Node.js та MongoDB.
5. Тестування та оцінка ефективності системи.

Практичне значення дослідження. Розроблена інформаційна система може бути використана підприємствами для управління транспортними процесами, що дозволить підвищити ефективність роботи, зменшити витрати та покращити якість обслуговування клієнтів. Система забезпечить оперативне прийняття рішень на основі аналізу великого обсягу даних, зберігання яких здійснюється у хмарі.

Структура роботи. складається з вступу, трьох розділів, висновків, списку використаних джерел. Перший розділ присвячений вибору хмарного провайдера та обґрунтуванню вибору. Другий розділ містить опис архітектури системи та вебдодатку. Третій розділ містить опис розробки додатку . Висновки містять основні результати дослідження та рекомендації щодо подальшого вдосконалення системи.

РОЗДІЛ 1
ПРОВАЙДЕРИ ХМАРНИХ ПОСЛУГ

1.1. Хмарні сервіси баз даних різних провайдерів

1.1.1. Хмарний сервіс від Microsoft Azure SQL Databases

База даних SQL Azure доступна як одиночна база даних із власним набором ресурсів, керована за допомогою логічного сервера, або як база даних в еластичному пулі із загальним набором ресурсів, для управління якою також використовується логічний сервер. Еластичні пули зазвичай призначені для типових моделей додатків SaaS (програмне забезпечення як послуга), в яких передбачена одна база даних на замовника або клієнт. У разі використання пулів ви керуєте загальною продуктивністю, а масштаб бази даних збільшується або зменшується автоматично [4].

Щоб створити власну базу данних спочатку потрібно пройти реєстрацію на сайті Microsoft (рис. 1.1), вводимо пошту ім’я та прізвище а також пароль і усі інші необхідні данні. Після цього створюємо нову базу данних (рис. 1.2).

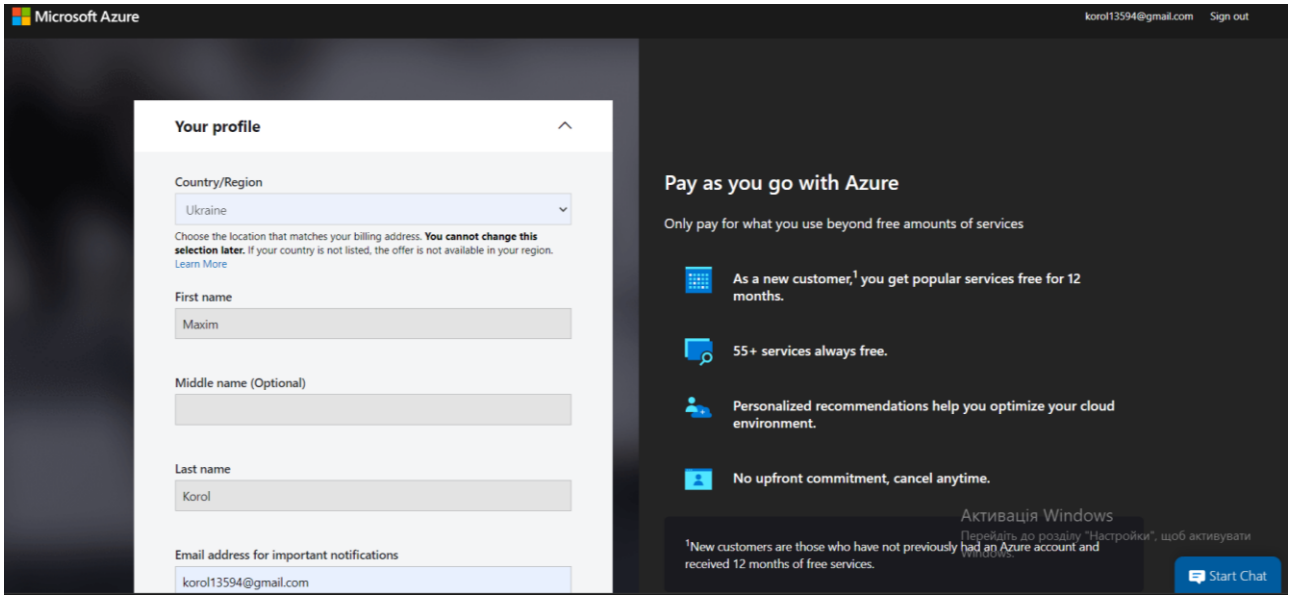

Рисунок 1.1. Сторінка для реєстрації на Microsoft Azure

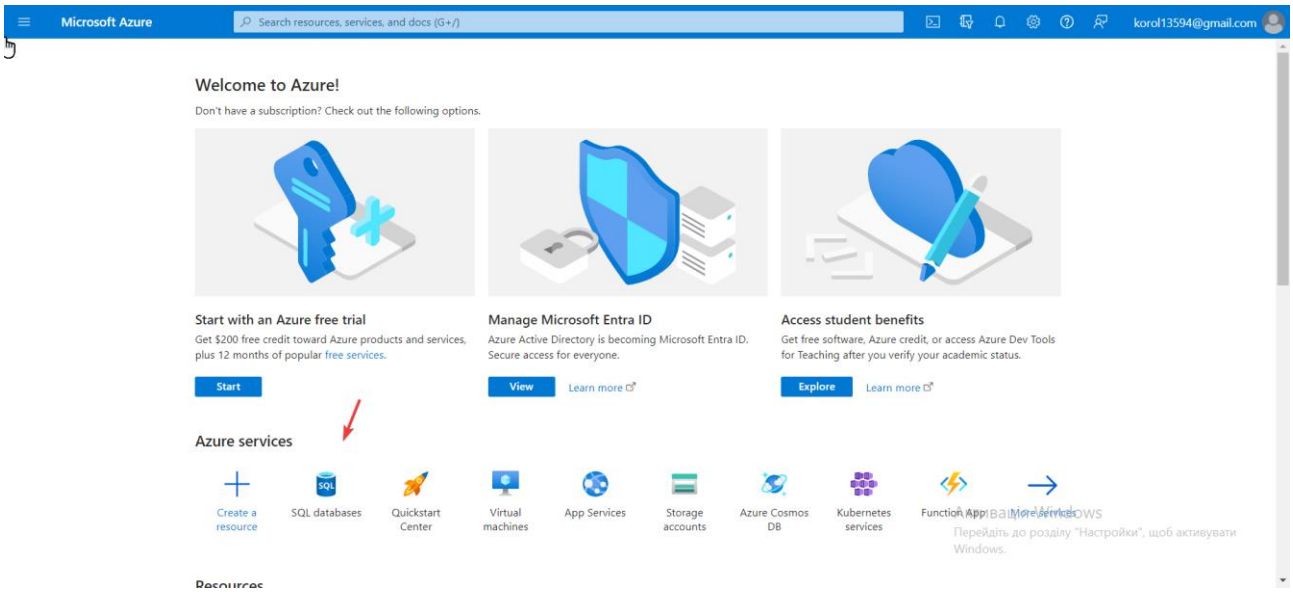

Рисунок 1.2. Головна сторінка Microsoft Azure

Перед створенням необхідно виконати базові налаштування (рис. 1.3).

Сюди входять:

- Тип підписки
- Регіон сервера
- Метод авторизації
- Тип процесора та кількість пам’яті

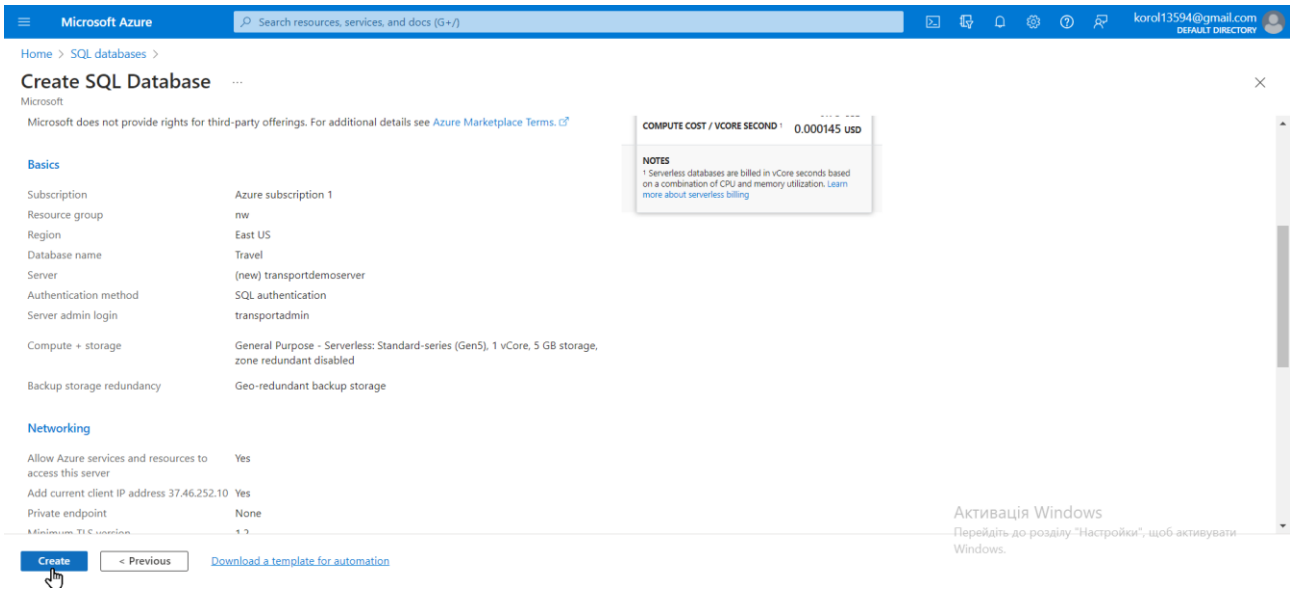

Рисунок 1.3.Сторінка створення БД

Наступний крок – це створення таблиці: додати поля, назвати поля, заповнити даними (рис. 1.4).

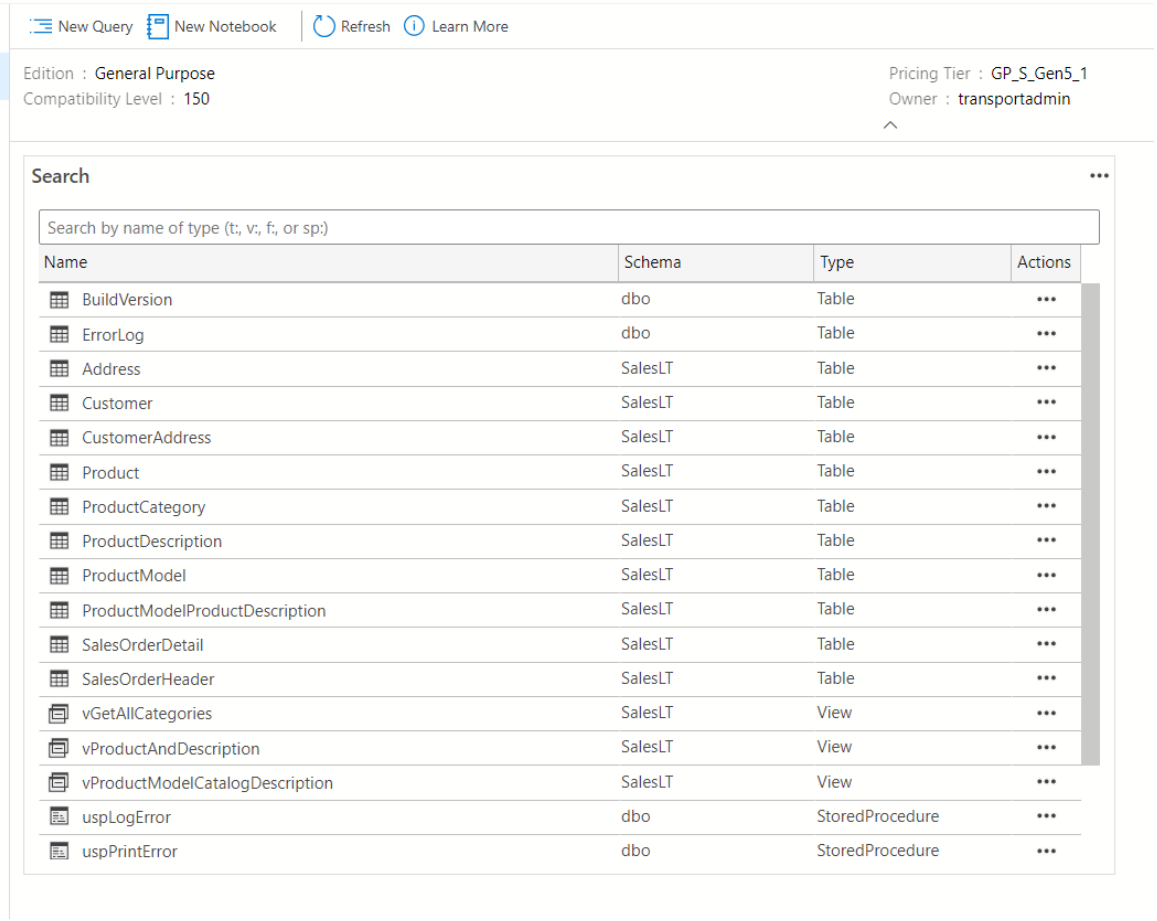

Рисунок 1.4.Список таблиць

1.1.2. Google cloud

Вперше Google Cloud був анонсований у 2008 році і тоді називався App Engine. Метою сервісу було спростити роботу та полегшити запуск вебдодатків. Основним завданням було відкрити нові можливості для роботи на хмарній платформі та почати масштабування. На той час у платформи вже був конкурент від Microsoft. Перший реліз сервісу також був призначений для тестування платформи, тому GCP (Google Cloud Platform, тоді App Engine) був доступний для більш ніж 10 000 розробників. Вони могли використовувати 500 МБ пам'яті, 200 мільйонів мегациклів процесора і 10 ГБ пропускної здатності на день. Випробування сервісу закінчилося в 2011 році, після чого Google запустив GCP як офіційний продукт. Сьогодні хмарними сервісами Google користуються такі

компанії, як Nintendo, eBay, UPS, Home Depot, Etsy, PayPal, 20th Century Fox і Twitter. Наразі існує 187 периферійних локацій GCP у понад 200 країнах і територіях, і ця мережа швидко розширюється [9].

Для створення бази данних за допомогою Google Cloud потрібно також пройти реєстрацію і мати обліковий запис Google. Так само перейти на сторінку сервісу Google Cloud і почати створювати базу даних (рис. 1.5).

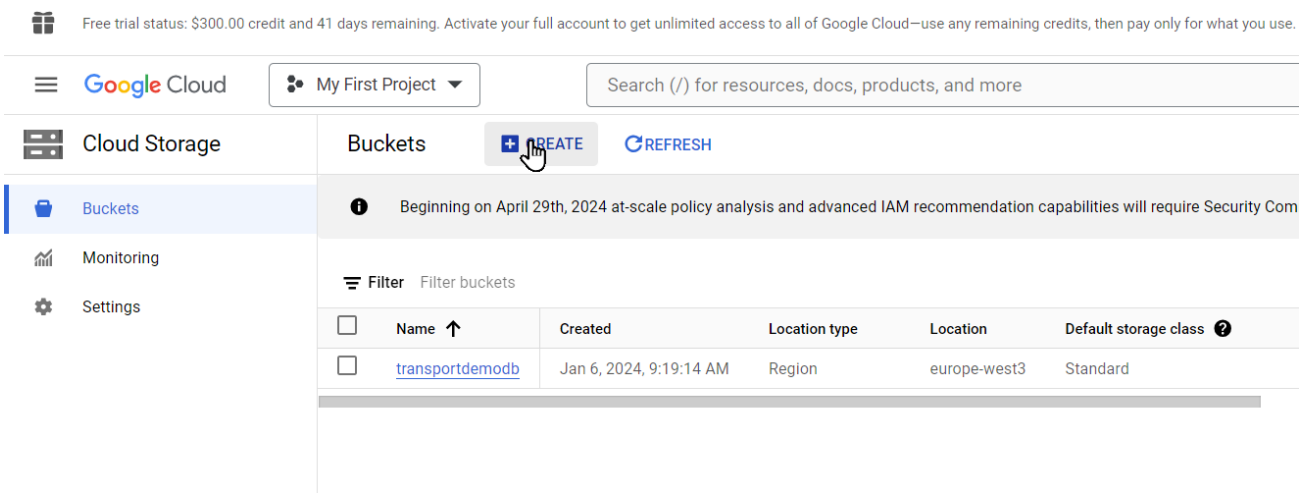

Рисунок 1.5. Створення Cloud Storage

Називаємо і налаштовуємо БД (рис. 1.6).

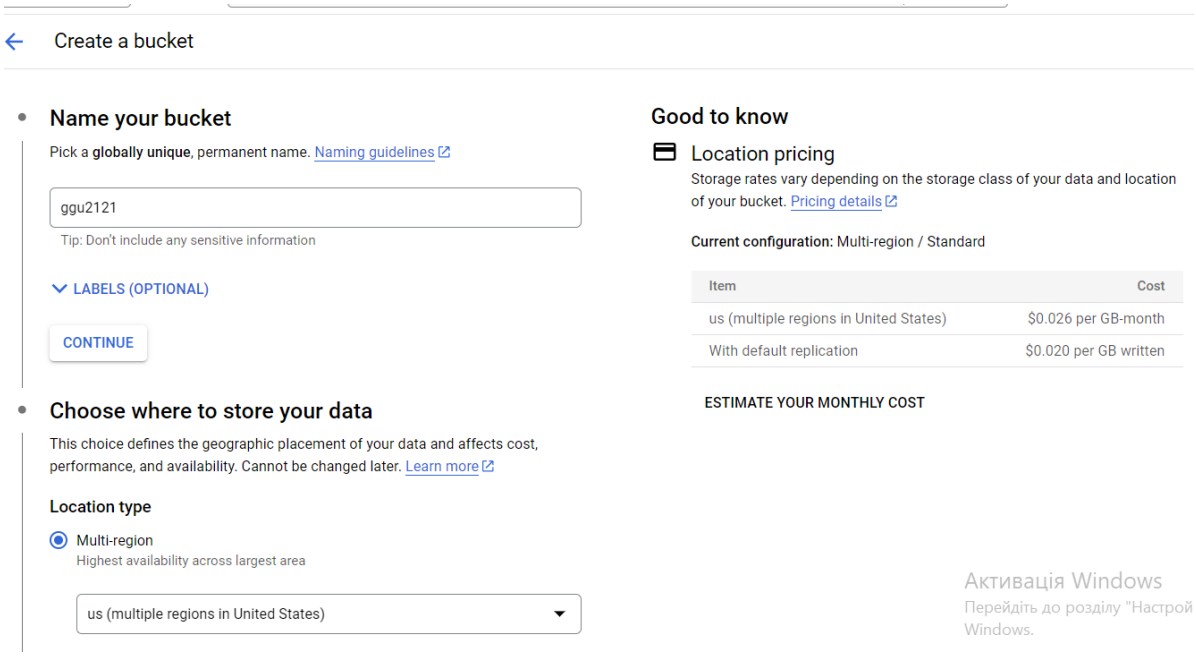

Рисунок 1.6. Налаштування БД

Бачимо безкоштовний обсях данних в 400 Gb якого нам вистачить приблизно на 40 днів. Берем також до уваги ціну за Gb в 0.026$ для подальшого порівняння з іншими ресурсами (рис. 1.7).

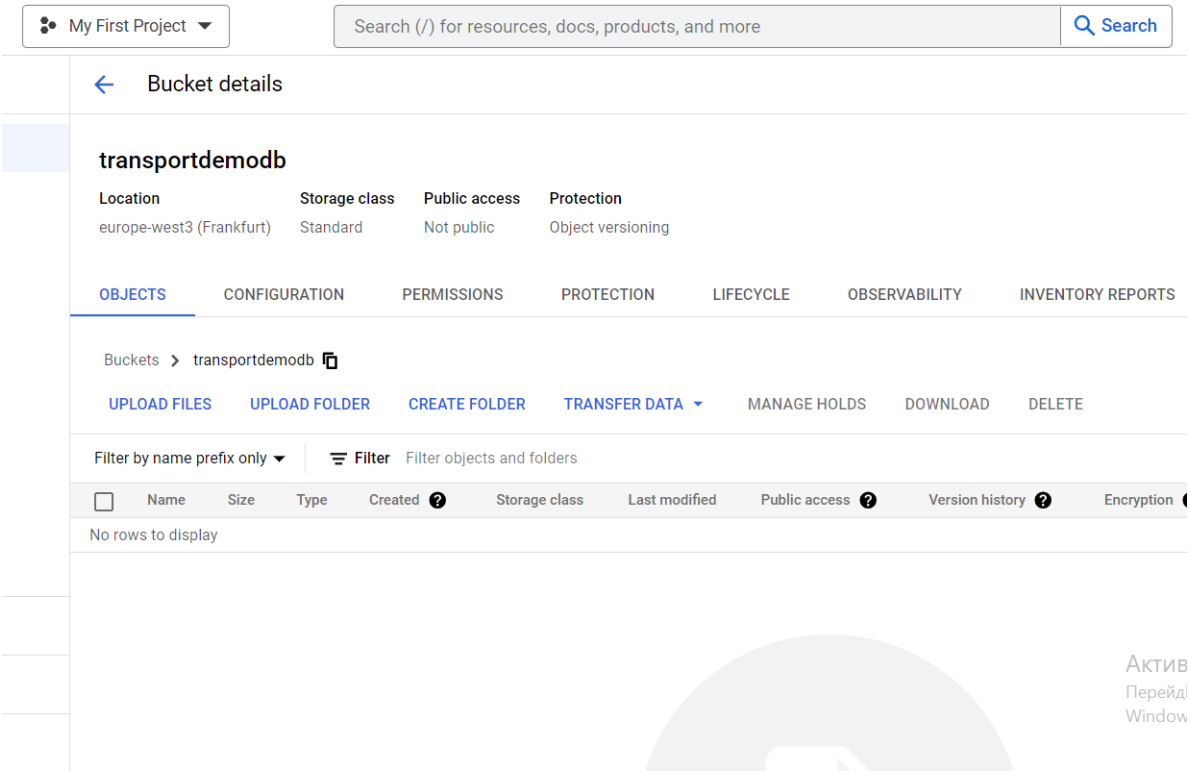

Рисунок 1.7. Меню БД

Після створення заповнюємо полями на транспортну тематику і зберігаємо результат (рис. 1.8).

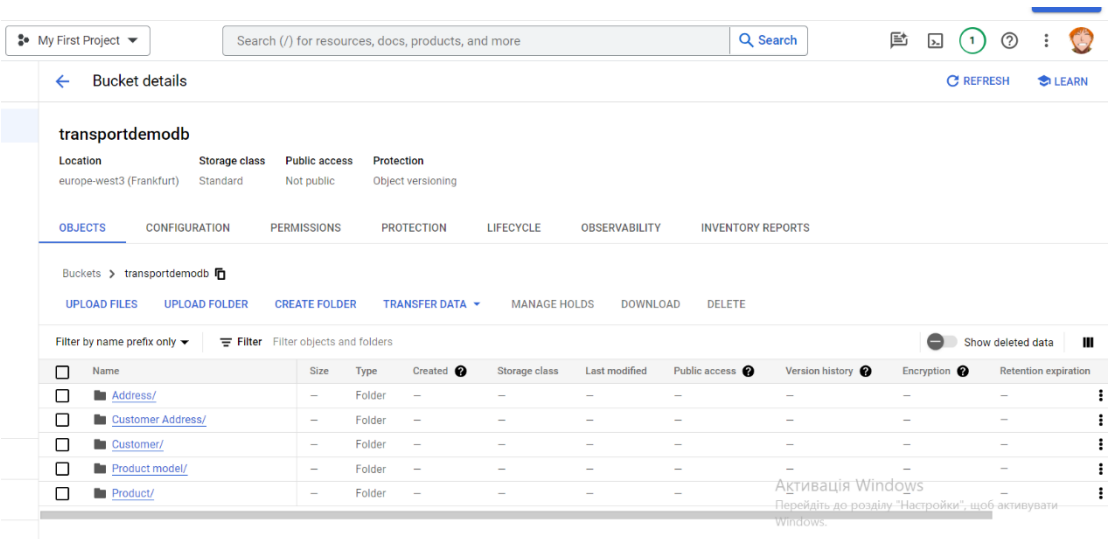

Рисунок 1.8. Заповнена БД

1.1.3. Хмарний сервіс від AWS: MongoDB

MongoDB – це система управління базами даних, яка використовує документи у форматі BSON (бінарна форма JSON) для зберігання даних. Основні особливості та характеристики MongoDB включають [2]:

- Документоорієнтована база даних. Замість традиційних таблиць, як у реляційних базах даних, MongoDB використовує колекції документів, що забезпечує більшу гнучкість і зручність у роботі з даними.
- Гнучка схема. У MongoDB немає фіксованої схеми, що дозволяє зберігати документи з різними структурами в одній колекції. Це особливо корисно при роботі з нерегулярними або часто змінюваними даними.
- Масштабованість. MongoDB підтримує горизонтальне масштабування через розподіл даних по кількох серверах (шардінг). Це дозволяє ефективно обробляти великі обсяги даних і високі навантаження.
- Висока продуктивність. MongoDB оптимізована для високошвидкісних операцій читання та запису, що робить її придатною для додатків з високими вимогами до продуктивності.
- Індексація. Підтримка складних індексів для прискорення запитів, включаючи індекси на поля всередині документів.
- Агрегаційні фреймворки. Наявність потужних інструментів для агрегації та обробки даних безпосередньо на рівні бази даних.
- Реплікація. Підтримка реплікації даних для підвищення доступності і надійності через реплікаційні набори.

MongoDB широко використовується в різних галузях, включаючи веброзробку, аналітику даних, IoT (інтернет речей) та багато інших додатків, де важлива гнучкість структури даних і висока продуктивність.

Після реєстрації на сайті переходимо до вибору кластера (рис. 1.9) і обираємо навчальний безкоштовний.

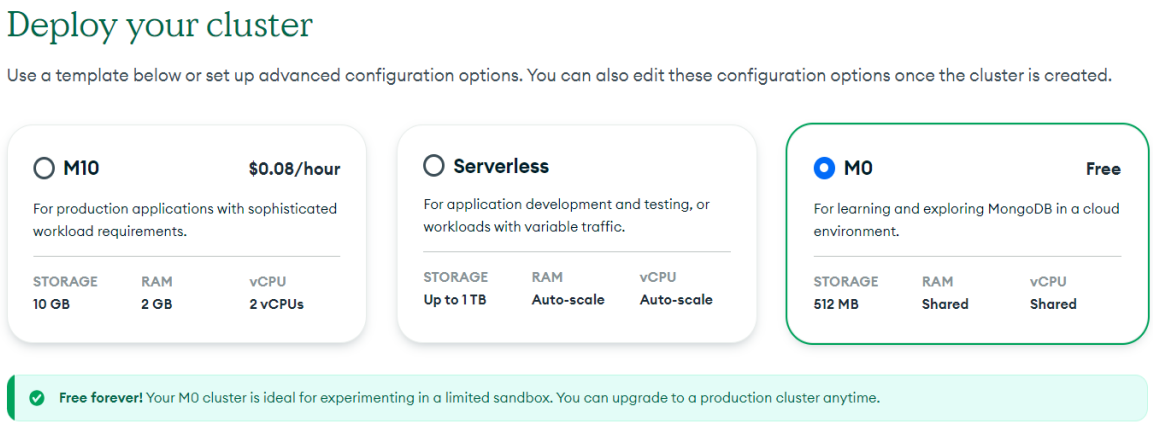

Рисунок 1.9. Варіанти тарифів

Обираємо провайдера AWS та перходимо до створення БД (рис. 1.10).

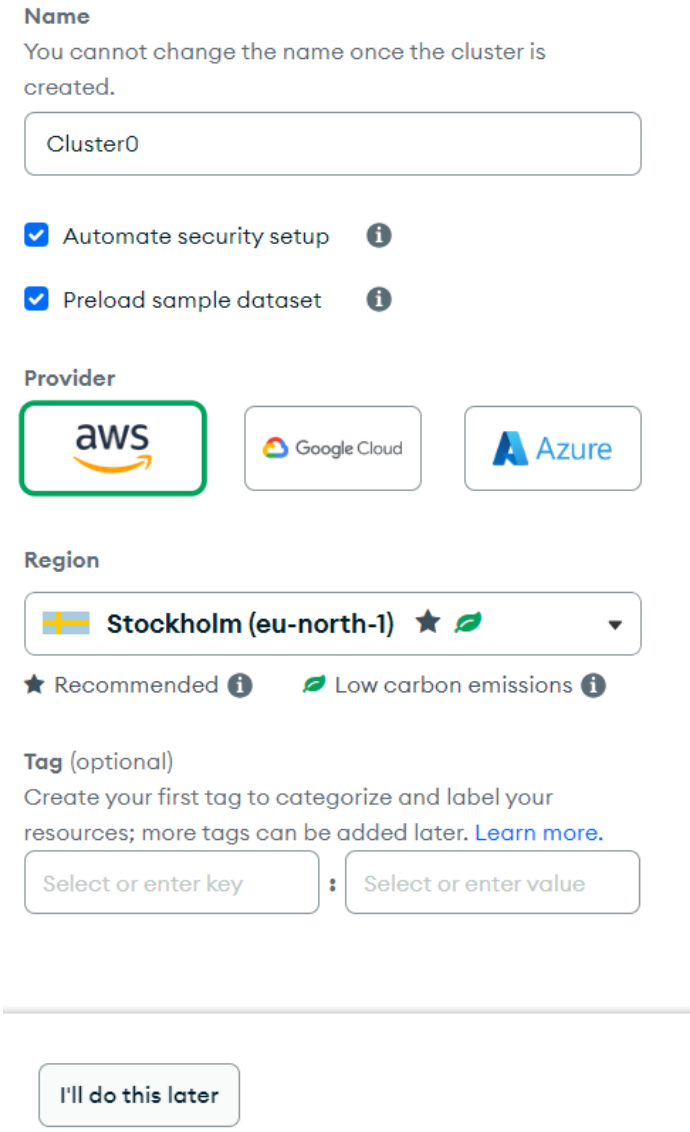

Рисунок 1.10. Створення кластера

Виконуємо необхідні кроки для з’єднання з БД. Створюємо користувача БД (рис. 1.11).

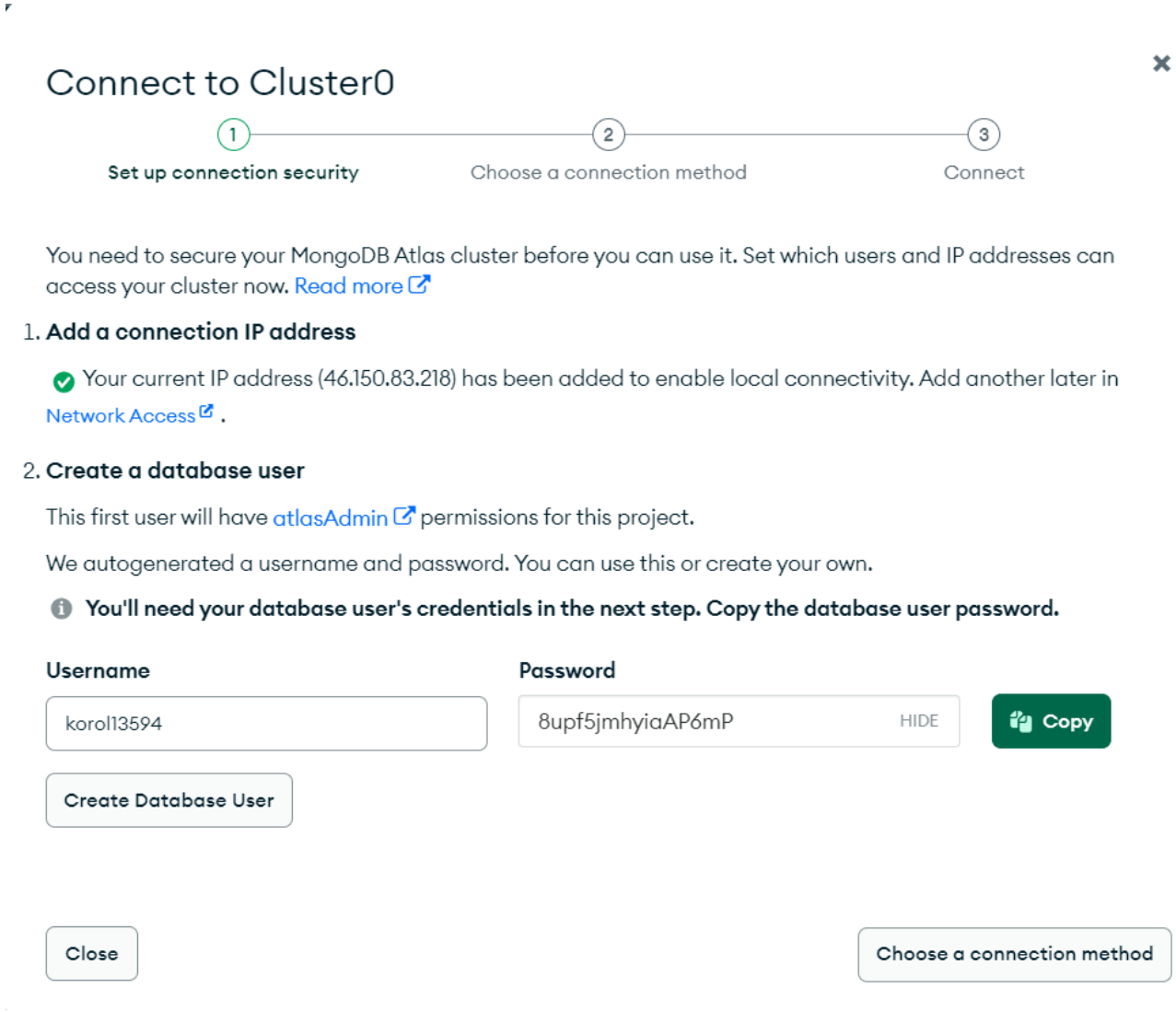

Рисунок 1.11 Створення користувача

Обираємо драйвер та копіюємо фрагмент коду для з’єднання з БД (рис. 1.12).

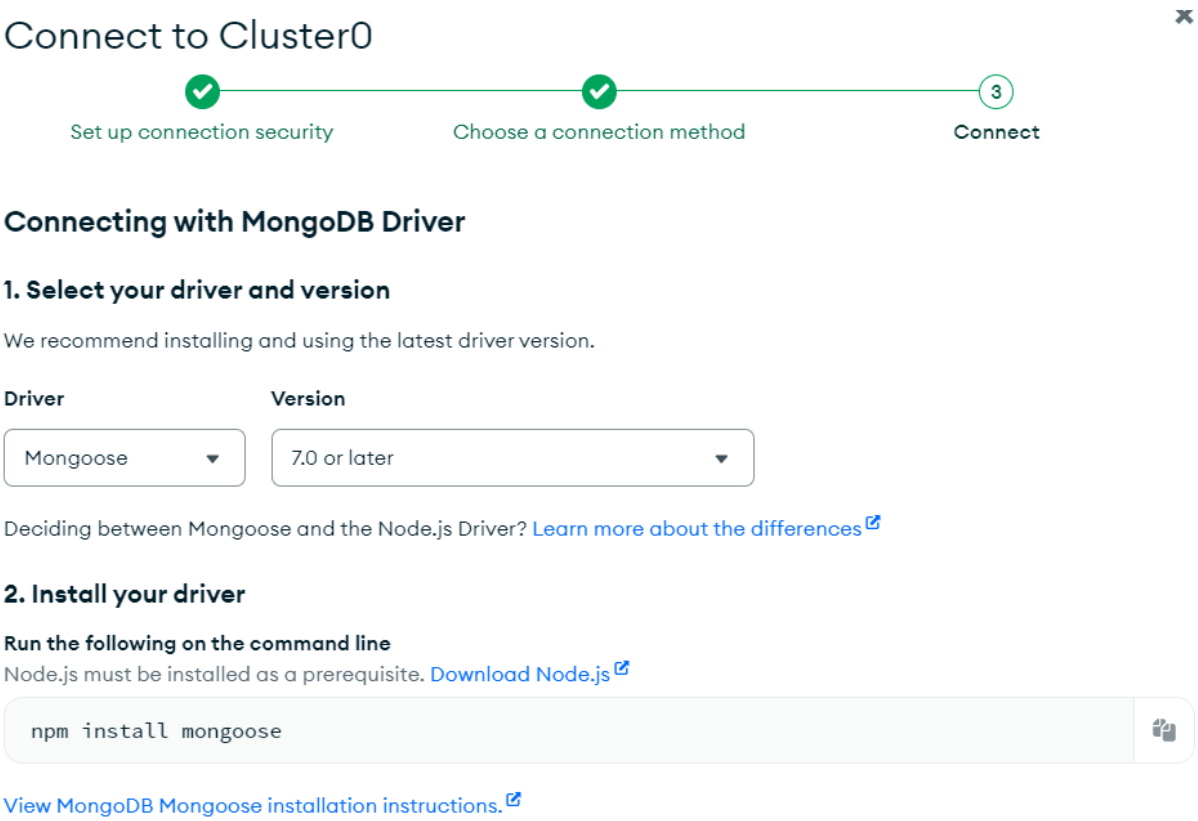

Рисунок 1.12. Установка драйвера БД

1.2. Порівняльна характеристика хмарних баз даних

Порівняння ціни було проведено суб'єктивно від лиця одного користувача (таблиця 1.1) [8].

Таблиця 1.1

Вартість користування хмарною БД

Cloud provider	Storage (GB/Month)
Amazon MongoDB	$0.023
Azure	$0.021
Google Cloud Platform	$0.026

Звісно є спосіб отримати деякі знижки та зменшити рахунок за хмару – скористатися потужністю, яка зараз ніким не використовується. Хмарні провайдери продають надлишок потужностей з неймовірними знижками. Екземпляри AWS Spot пропонують знижку до 90% від ставок On-Demand, Preemptible VM у Google можуть бути навіть на 80% дешевшими за звичайні [7]:

-AWS пропонує чудову ціну на машини загального призначення, хоча знижка не найвища;

-Azure пропонує найбільші знижки для обох інстанцій, оптимізованих для обчислень. Ціна оптимізованого для обчислень F4s v2 дуже приваблива;

- Google Cloud Platform також можна непогано зекономити хота и не настільки як в Azure.

Порівняємо поширення та популярність хмарних БД від різних компаній (таблиця 1.2) [8].

Таблиця 1.2.

Категорія	**AWS**	**Microsoft Azure**	**Google Cloud**
Випуск	Березень 2006	Лютий 2010	Листопад 2011
Регіони	32	47	39
Зони доступності	102	92	118
Частка ринку	22%	10%	33%

Microsoft Azure може похвалитися найбільшою кількістю регіонів, але справжня доступність означає наявність зон доступності. Google Cloud є лідером у цьому важливому відношенні, пропонуючи найбільшу кількість зон по всьому світу. Кожен регіон пропонує щонайменше дві або три зони доступності, що збільшує економічно ефективну географічну доступність. Як бачимо, вражаюча деталізація та молодість Google Cloud доповнюються винятковою доступністю, що зміцнює його позиції порівняно з іншими хмарними провайдерами.

Безпека та відповідність вимогам:

- AWS: пропонує надійні функції безпеки, сертифікацію відповідності та моделі спільної відповідальності.
- Azure: фокусується на високих рівнях безпеки та відповідності, особливо для таких галузей, як охорона здоров'я та фінанси.
- GCP: забезпечує безпечну інфраструктуру з такими функціями, як управління ідентифікацією та доступом (IAM) і шифрування; GCP єдина хмарна платформа, яка ніколи не зазнавала хакерських атак і нещодавно витримала найбільшу DDoS-атаку в історії, при цьому якість обслуговування клієнтів не постраждала. Якість обслуговування не постраждала.

Підтримка та документація [9]:

- AWS: велика спільнота, обширна документація та низка планів підтримки. Преміум-підтримка надає доступ до інженерів хмарної підтримки 24/7.
- Azure: потужна підтримка та широкий вибір планів. Добре задокументована та використовує переваги екосистеми підтримки Microsoft.
- GCP: пропонує низку планів підтримки, а документація проста та добре організована.

1.3. Переваги хмарних баз даних

Переваги хмарних баз даних у порівнянні із стаціонарними [10]:

- *Збільшення обсягу зберігання даних*. Хмарне сховище не обмежується ємністю фізичного пристрою. Ви можете зберігати більше даних, не турбуючись про оновлення пам'яті.
- *Підвищена масштабованість*. Деякі компанії мають мінливі потреби в ресурсах. У хмарі вони можуть гнучко збільшувати або зменшувати обсяг оперативної пам'яті, віртуальних машин і дисків за потреби.

- *Віддалена організація роботи*. Співробітники можуть підключатися до сервісу з будь-якої точки світу і з будь-якого пристрою, підвищуючи взаємодію між віддаленими командами.
- *Економія коштів*. Оренда хмарних ресурсів позбавляє від необхідності купувати обладнання та зменшує капітальні витрати. Ці кошти можна спрямувати на розвиток бізнесу або освоєння нових ринків.
- *Резервне копіювання та аварійне відновлення*. Рішення для резервного копіювання та відновлення можна впровадити економічно ефективно і без зайвих клопотів.

Майбутнє хмарних обчислень у 2024 році сповнене перспектив та інновацій. Прийняття змін і використання потенціалу хмарних технологій буде ключовим фактором для того, щоб орієнтуватися в цих тенденціях і залишатися попереду в цифровому середовищі.

РОЗДІЛ 2
СТРУКТУРА ІНФОРМАЦІЙНОЇ СИСТЕМИ ТА ТЕХНОЛОГІЇ ДЛЯ ЇЇ РЕАЛІЗАЦІЇ

2.1. Огляд технологій для створення додатку

2.1.1. Особливості створення вебдодатку

Вебдодатки використовуються в різних сферах, включаючи електронну комерцію, банківські послуги, соціальні мережі, онлайн-освіту, управління проектами, системи бронювання та багато інших. Вони забезпечують зручний спосіб взаємодії з користувачами та дозволяють швидко і ефективно обробляти великі обсяги даних.

Вебдодатки – це програмне забезпечення, яке працює на вебсервері і доступне користувачам через веббраузер. Вони поєднують функціональність настільних додатків з перевагами роботи через Інтернет.

Однією з головних характеристик вебдодатків є їх доступність через браузер, що означає, що користувачі не потребують встановлення додаткового програмного забезпечення на свої пристрої. Це також забезпечує кросплатформність, оскільки вебдодатки можуть працювати на будь-якій операційній системі, будь то Windows, macOS, Linux, Android чи iOS [11].

Вебдодатки мають централізоване управління, що означає, що всі оновлення і зміни здійснюються на сервері. Завдяки цьому користувачі завжди працюють з актуальною версією додатку, без необхідності вручну встановлювати оновлення.

Інтерактивність є ще однією ключовою характеристикою вебдодатків. Сучасні вебдодатки використовують технології, такі як JavaScript, AJAX, та різні фреймворки (наприклад, React, Vue.js або Angular), щоб забезпечити динамічну і інтерактивну взаємодію користувачів з додатком. Важливою складовою вебдодатків є робота з базами даних. Вебдодатки часто використовують серверні

бази даних для зберігання та обробки даних, що дозволяє зберігати великі обсяги інформації та забезпечувати швидкий доступ до неї [15].

Клієнтська частина (frontend) відповідає за відображення інтерфейсу користувача та обробку взаємодії з ним, використовуючи HTML для структури, CSS для стилізації та JavaScript для додавання динамічних елементів.

Серверна частина (backend) відповідає за обробку запитів від клієнта, взаємодію з базами даних, бізнес-логіку та управління сеансами користувачів.

Бази даних зберігають дані, які використовуються вебдодатком, і це можуть бути як реляційні, так і нереляційні бази даних. API (інтерфейс програмування додатків) забезпечує стандартизований обмін даними між клієнтом та сервером, використовуючи RESTful або GraphQL API.

Вебдодатки мають багато переваг, таких як доступність з будь-якого місця і пристрою, легкість в оновленні, безпека даних, які зберігаються на сервері, та масштабованість, що дозволяє легко адаптуватися до навантаження та кількості користувачів [1].

Створення вебдодатку складається з кількох важливих етапів, кожен з яких включає в себе певні завдання та процеси, що забезпечують успішну реалізацію проекту [9]:

Перший етап – це визначення вимог. На цьому етапі розробники разом із замовниками або зацікавленими сторонами збирають і аналізують вимоги до вебдодатку. Це включає в себе розуміння цільової аудиторії, визначення основних функціональних можливостей, встановлення технічних обмежень та створення документації з вимогами.

Другий етап – планування проекту. Після того, як вимоги визначені, команда розробників створює план проекту. Це включає розробку графіку робіт, розподіл завдань серед членів команди, оцінку ресурсів та визначення ключових етапів та мільйостей проекту. Планування також охоплює вибір технологій та інструментів, які будуть використовуватися під час розробки.

Третій етап – дизайн. На цьому етапі створюються макети та прототипи користувацького інтерфейсу. Дизайнери працюють над створенням візуального

стилю, розташуванням елементів на сторінках та розробкою користувацьких сценаріїв. Важливо забезпечити зручність користування, естетичність та відповідність дизайну вимогам замовника.

Четвертий етап – розробка. Це основний етап, на якому програмісти пишуть код додатку. Спочатку створюється структура проекту та базові компоненти, а потім реалізуються функціональні можливості. Розробка може бути поділена на frontend та backend частини. Frontend-розробники створюють інтерфейс користувача, використовуючи HTML, CSS та JavaScript, тоді як backend-розробники працюють над серверною логікою, базами даних та API.

П'ятий етап – тестування. Після того, як основна частина коду написана, додаток проходить етап тестування. Це включає в себе модульне тестування, інтеграційне тестування та системне тестування. Мета тестування – виявити та виправити помилки, забезпечити стабільну роботу додатку та перевірити відповідність вимогам.

Шостий етап – деплоймент. Після успішного тестування вебдодаток готується до розгортання на сервері. Це включає налаштування серверного оточення, баз даних, сертифікатів безпеки та інших компонентів. Додаток розгортається на продакшн сервері, де він стає доступним для кінцевих користувачів.

Сьомий етап – моніторинг та підтримка. Після розгортання додаток потребує постійного моніторингу для забезпечення його стабільної роботи. Це включає в себе моніторинг продуктивності, безпеки та обробку зворотного зв'язку від користувачів. Підтримка також включає в себе виправлення помилок, оновлення додатку та впровадження нових функцій відповідно до потреб користувачів.

Кожен з цих етапів є критично важливим для успішної реалізації вебдодатку. Від визначення вимог до підтримки готового продукту, всі етапи потребують ретельного планування.

2.1.2. Огляд сервісів баз даних для вебдодатку

Реляційні бази даних використовують таблиці для зберігання даних Вони базуються на реляційній моделі, запропонованій Едгаром Коддом і дозволяють зберігати дані у вигляді рядків і стовпців Основні характеристики реляційних баз даних включають таблиці, первинні ключі, зовнішні ключі та SQL (Structured Query Language). Приклади реляційних баз даних: MySQL, PostgreSQL, Microsoft SQL Server, Oracle Database [11].

Нереляційні бази даних (NoSQL) відходять від традиційної табличної моделі і можуть використовувати різні структури даних для зберігання та маніпуляції інформацією. Види нереляційних баз даних включають документо-орієнтовані бази даних, колонкові бази даних, графові бази даних та бази даних «ключ-значення». Приклади: MongoDB, Apache Cassandra, Neo4j, Redis [1].

Структуровані та неструктуровані бази даних розрізняються за організацією та типом даних, які вони зберігають. Структуровані дані це дані, що організовані у визначену модель або формат, як-от таблиці в реляційних базах даних. Неструктуровані дані це дані, що не піддаються легкій категоризації або зберігаються у форматі, який не має певної моделі, наприклад текстові файли, мультимедіа, документи JSON [9].

Моделі зберігання даних визначають спосіб організації та доступу до даних/ Основні моделі включають [10]:

- Реляційна модель організовує дані у вигляді таблиць з рядками та стовпцями, де кожна таблиця має назву та схему, що визначає типи даних для кожного стовпця і відносини між таблицями. Вона використовує мову структурованих запитів, наприклад, SQL, для маніпуляції даними.
- Документна модель представляє дані у вигляді документів, що можуть бути у форматі JSON, XML або іншому подібному форматі. Кожен документ містить пари ключ-значення і може мати вкладені

структури, що робить цю модель гнучкою та зручною для зберігання складних і різнорідних даних.

- Графова модель використовує графи для представлення даних, де вузли відображають об'єкти, а ребра - відносини між ними. Ця модель особливо корисна для вираження складних взаємозв'язків між об'єктами, таких як соціальні мережі або рекомендаційні системи.
- Модель «ключ-значення» зберігає дані у вигляді пар ключ-значення, де кожен ключ унікальний і відповідає певному значенню. Ця модель проста і ефективна для швидкого доступу до даних за ключем, але менш гнучка для складних запитів.
- Колонкова модель зберігає дані у вигляді таблиць, але дані розділяються по колонках, а не рядках. Це дозволяє ефективно виконувати операції, що вимагають доступу до певних колонок, такі як агрегація або аналіз великих обсягів даних.

MySQL відомий своєю стабільністю, широкою підтримкою та розповсюдженістю. Він має потужні можливості оптимізації та високий рівень безпеки. Крім того, MySQL підтримує транзакції, які важливі для додатків, що потребують синхронізації даних. Головним недоліком є обмежена масштабованість для великих обсягів даних

PostgreSQL відомий своєю потужною підтримкою ACID (атомарність, консистентність, ізольованість, довершеність), що робить його популярним для великих підприємств та додатків, що вимагають високої надійності. Він також має розширюваність та широкий набір функцій, таких як географічні запити і текстовий пошук. Але високий рівень вимог до ресурсів, так як PostgreSQL володіє вищим рівнем складності та вимагає більше обчислювальних ресурсів порівняно з деякими іншими системами управління базами даних.

Apache Cassandra відомий своєю горизонтальною масштабованістю та надійністю. Він добре підходить для розподілених систем, що обробляють великі

обсяги даних, таких як мережі соціальних мереж, IoT (Інтернет речей) та аналітика в реальному часі. Крім того, Cassandra пропонує гарну підтримку резервного копіювання даних і автоматичне відновлення в разі збоїв. Його головним недоліком є складність у налаштуванні та адмініструванні, особливо для менш досвідчених користувачів.

MongoDB відома своєю гнучкістю схеми даних, швидкістю розгортання та горизонтальним масштабуванням. MongoDB – це нереляційна, документо-орієнтована база даних Вона використовує документи у форматі BSON (бінарне представлення JSON) для зберігання даних, що дозволяє зберігати дані з гнучкою та змінною структурою. Це робить MongoDB ідеальною для роботи з неструктурованими або частково структурованими даними. Її основні характеристики [16]:

- Тип бази даних: Нереляційна (NoSQL);
- Модель зберігання даних: Документна модель;
- Формат даних: BSON (бінарний формат JSON);
- Призначення: Ефективна для роботи з великими обсягами неструктурованих або частково структурованих даних, що потребують гнучкості у схемі;
- Структуровані/Неструктуровані дані: Переважно працює з неструктурованими даними, але може зберігати і структуровані дані у вигляді документів.

Зважаючи на переваги MongoDB, саме цю БД будемо використовувати в дипломному проєкті.

Щоб створити базу даних у MongoDB, потрібно завантажити та встановити MongoDB з офіційного сайту. Після цього потрібно запустити сервер MongoDB командою `mongod`. Потім відкрити MongoDB Shell, використовуючи команду `mongo`, і створити нову базу даних командою `use mydatabase`. Після цього потрібно додати перший документ до нової колекції за допомогою команди `db.mycollection.insertOne({ name: "example" })`.

Щоб підключити MongoDB до проєкту на Node.js, спершу потрібно ініціалізувати новий проєкт командою `npm init -y`. Далі потрібно встановити бібліотеку `mongoose` командою `npm install mongoose`.

У файлі основного додатка, наприклад `app.js`, потрібно імпортувати бібліотеку `mongoose` і створити підключення до вашої бази даних за допомогою `mongoose.connect`, вказавши URI вашої бази даних, наприклад, `mongodb://localhost:27017/mydatabase`, і налаштувати параметри з'єднання.

Після успішного підключення потрібно визначити схему для вашої колекції, використовуючи `mongoose.Schema`, і створити модель на її основі. Щоб перевірити підключення та роботу з базою даних, потрібно створити новий документ, використовуючи модель, і зберегти його в базі даних за допомогою методу `save`.

2.1.3. Середовище виконання та вибір технологій

Для розробки додатку використовуватимемо Node.js в комбінації з MongoDB оскільки разом вони мають низку переваг, які роблять її популярною серед розробників:

Переваги Node.js [14]:

- *Асинхронність та неблокуючий ввід/вивід*: Node.js використовує неблокуючу модель вводу/виводу, що дозволяє обробляти велику кількість одночасних запитів без створення нових потоків для кожного з них. Це робить Node.js дуже ефективним для обробки подій у реальному часі та великої кількості одночасних з'єднань.
- *Єдина мова програмування*: З Node.js можна використовувати JavaScript як на стороні клієнта, так і на стороні сервера, що спрощує процес розробки та дозволяє використовувати один стек технологій для всього додатку.
- *Швидкість*: Node.js побудований на V8, швидкому JavaScript рушії від Google, що забезпечує високу продуктивність і швидке виконання коду.

- *Велика екосистема*: Node.js має величезну екосистему пакетів npm (Node Package Manager), що дозволяє легко знаходити та використовувати бібліотеки для майже будь-якої задачі.
- *Підтримка спільноти*: Велика і активна спільнота розробників забезпечує постійну підтримку, оновлення та поліпшення платформи.

Переваги MongoDB [16]:

- *Гнучкість структури даних*: MongoDB є документоорієнтованою базою даних, яка використовує BSON (Binary JSON). Це дозволяє зберігати документи з різною структурою без необхідності жорсткого визначення схеми, що особливо корисно для швидко змінюваних або нерегулярних даних.
- *Масштабованість*: MongoDB легко масштабується як вертикально, так і горизонтально. Це означає, що можна збільшувати продуктивність бази даних, додаючи нові ресурси або розподіляючи дані на кілька серверів.
- *Висока доступність*: Завдяки вбудованим механізмам реплікації та автоматичного відновлення, MongoDB забезпечує високу доступність і надійність даних.
- *Продуктивність*: MongoDB оптимізований для високої швидкості читання та запису даних, що робить його придатним для додатків, які потребують швидкого доступу до великих обсягів даних.

Переваги комбінації Node.js + MongoDB [12]:

- *JSON* (*JavaScript Object Notation*): Як Node.js, так і MongoDB працюють з JSON-документами. Це означає, що ви можете зберігати, запитувати і обробляти дані у форматі JSON на всіх рівнях вашого додатку, що спрощує процес розробки.
- *Швидкість розробки*: Використовуючи JavaScript на всіх рівнях додатку, можна значно скоротити час розробки та налагодження коду. Велика кількість готових бібліотек та інструментів також сприяє швидкому розвитку.

- *Реалізація реального часу*: Поєднання неблокуючої моделі Node.js і високої швидкості MongoDB ідеально підходить для додатків реального часу, таких як чати, системи обміну повідомленнями та онлайн-ігри.
- *Масштабованість та гнучкість*: Node.js легко масштабується, як і MongoDB, що дозволяє обробляти великі навантаження та динамічні дані без значних змін в архітектурі додатку.

Використання Node.js і MongoDB разом створює потужний і гнучкий стек для сучасних вебдодатків, забезпечуючи високу продуктивність, масштабованість та швидкість розробки.

Фреймворки відіграють важливу роль у розробці сучасного програмного забезпечення. Вони забезпечують структуру, яка полегшує та прискорює процес розробки, надаючи готові компоненти, шаблони та інструменти. Фреймворки допомагають підтримувати чітку структуру проекту, що полегшує розуміння та підтримку коду. Завдяки готовим рішенням та компонентам, розробники можуть зосередитися на бізнес-логіці, не витрачаючи час на розробку базових функцій. Використання фреймворків забезпечує дотримання стандартів розробки, що полегшує інтеграцію з іншими системами та бібліотеками. Фреймворки часто включають вбудовані механізми захисту від поширених вразливостей, що підвищує загальну безпеку додатку. Популярні фреймворки мають великі спільноти розробників, що забезпечує доступ до документації, прикладів та підтримки.

При реалізації дипломного проєкту будемо використовувати Vue Component Framework.

Vue.js є одним із найбільш популярних фреймворків для створення інтерфейсів користувача, завдяки своїм численним перевагам та можливостям. Vue.js має низький поріг входження. Його легко освоїти навіть розробникам-початківцям, завдяки простій та зрозумілій документації.

Vue.js можна використовувати як для створення невеликих компонентів, так і для розробки масштабних додатків. Це дозволяє легко масштабувати проект

відповідно до потреб бізнесу. У Vue.js все побудовано на компонентах, що дозволяє створювати багаторазові блоки коду. Це сприяє кращій організації та повторному використанню коду, що знижує витрати на підтримку та розвиток додатка [13].

Vue.js забезпечує високу продуктивність завдяки реактивному механізму, який автоматично оновлює DOM при зміні стану додатка. Це дозволяє створювати швидкі та ефективні інтерфейси користувача. Vue.js легко інтегрується з іншими бібліотеками та існуючими проектами. Це дозволяє поступово впроваджувати Vue.js у великі проекти без необхідності переписувати весь код. Vue.js має розвинену екосистему, включаючи офіційні бібліотеки для управління станом (Vuex), роутінгу (Vue Router), та інструменти для створення та налаштування проектів (Vue CLI). Це забезпечує розробників усіма необхідними інструментами для повноцінної розробки [12].

Vue.js має велику та активну спільноту розробників, які постійно вдосконалюють фреймворк та створюють нові плагіни та розширення. Це забезпечує надійну підтримку та швидке вирішення проблем, які можуть виникнути в процесі розробки.

Vue.js оптимізований для високої продуктивності. Його розмір менший порівняно з іншими популярними фреймворками, що забезпечує швидше завантаження та відгук додатків. Vue.js добре працює з серверним рендерингом за допомогою Nuxt.js, що дозволяє створювати SEO-оптимізовані додатки, які краще індексуються пошуковими системами. Таким чином, використання Vue Component Framework дозволяє створювати ефективні, масштабовані та легко підтримувані вебдодатки. Це робить його ідеальним вибором для проєктів різного масштабу, від простих вебсайтів до складних вебдодатків [13].

2.2. Архітектура інформаційної системи

Розроблена в дипломній роботі інформаційна система є вебдодатком на транспортну тематику, спроєктованим для надання користувачам можливості замовлення доставки пакунків з одного місця до іншого. Додаток вимагає від

користувачів заповнення різних полів, таких як орієнтовна вартість, вага пакунку, адреса доставки та адреса відправки.

Компоненти системи:

- *Клієнтська частина*: вебінтерфейс, який користувачі використовують для замовлення доставки;
- *Серверна частина*: логіка додатку та взаємодія з базою даних;
- *База даних*: MongoDB, яка використовується для зберігання інформації про замовлення та користувачів;
- *Інтерфейс користувача*: Реалізований у вигляді веб-додатку з використанням фреймворка;
- *Взаємозв'язки між компонентами*: Клієнтська частина взаємодіє з серверною частиною через HTTP протокол, висилаючи запити на сервер. Серверна частина обробляє ці запити, виконує необхідні дії (наприклад, збереження замовлення у базі даних) та надсилає відповідь. Для зберігання та отримання даних використовується MongoDB.

Для реалізації інформаційної системи обрано архітектурні рішення:

- Використання Node.js для створення серверної частини додатку, оскільки він дозволяє реалізувати асинхронний та швидкий код;
- Використання MongoDB як бази даних, оскільки вона забезпечує гнучкість схеми та хорошу масштабованість.

При розробці інформаційної системи будуть реалізовані програмні механізми:

- *Асинхронне програмування*: використання асинхронних операцій та колбеків для ефективної обробки запитів;
- *Кешування*: застосування механізмів кешування для збереження проміжних результатів та підвищення продуктивності;
- *Автентифікація та авторизація*: впровадження механізмів автентифікації та авторизації користувачів для захисту конфіденційності та безпеки даних.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ НА ТРАНСПОРТНУ ТЕМАТИКУ

3.1. Огляд вебдодатку, побудованого з використанням хмарної бази даних

3.1.1. Опис серверної (backend) частини вебдодатку

Налаштування з’єднання з базою даних реалізовано наступним чином:

```
// Make function to connect to database
const mongoose = require('mongoose')
const connectDatabase = async () => {
  await mongoose.connect(process.env.DB_URL,
  {useNewUrlParser: true, useUnifiedTopology: true})
    .then(() => {
      console.log('mongoose connect')
})
}
module.exports = connectDatabase
//use this function in the root file of the project
const app = require('./app')
const connectDataBase = require('./config/database')
connectDataBase()
const PORT = process.env.PORT || 4000
const server = app.listen(PORT, () => {
  console.log(`Server is working on http://localhost:${PORT}`)
})
```

У цій частині коду створюється функція для підключення до бази даних MongoDB за допомогою Mongoose. Функція виконує підключення до бази даних

за URL, збереженим у змінній середовища, та виводить повідомлення про успішне підключення. Потім ця функція експортується для використання в інших частинах програми.

У кореневому файлі проекту імпортується Express-додаток та функція підключення до бази даних. Функція підключення викликається для встановлення з'єднання з базою даних. Далі налаштовується сервер, який прослуховує порт, визначений у змінній середовища або за замовчуванням 4000, та виводить повідомлення про успішний запуск сервера з вказанням URL.

Запуск сервера:

```
"scripts": {
"start": "node server.js",
"dev": "nodemon server.js"
 },
```

Налаштування обробки помилок:

```
const ApiError = require('../exception/api.error');
module.exports = function (err, req, res, next) {
  console.log(err);
    if (err instanceof ApiError) {
    return res.status(err.status).json
   ({ message: err.message, errors: err.errors });
  }
  return res.status(500).json({ message: 'Something went wrong' });
};
module.exports = class ApiError extends Error {
  status;
  errors;
  constructor(status, message, errors = []) {
    super(message);
```

```
        this.status = status;
        this.errors = errors;
    }
    static UnauthorizedError() {
        return new ApiError(401, 'Користувач не авторизований');
    }
    static BadRequest(message, errors = []) {
        return new ApiError(400, message, errors);
    }
};
```

Це дозволяє додатку централізовано обробляти помилки, забезпечуючи єдиний формат відповіді для різних типів помилок, що можуть виникнути під час роботи серверу. Ця частина коду визначає механізм обробки помилок у вебдодатку. Спочатку створюється функція, яка використовується як middleware для перехоплення і обробки помилок. Якщо перехоплена помилка є екземпляром класу ApiError, то клієнту відправляється відповідь з статусом помилки і повідомленням, які визначені в об'єкті ApiError. В іншому випадку, клієнту повертається загальне повідомлення про помилку з статусом 500.

Також визначається клас ApiError, що розширює стандартний клас Error. Цей клас додає додаткові властивості status і errors, які використовуються для більш точного опису помилок. Клас містить статичні методи UnauthorizedError і BadRequest для створення типових помилок, які можна використовувати для більш зручного управління помилками в додатку. Це дозволяє розробнику створювати і відправляти спеціалізовані помилки, які містять додаткову інформацію про причину помилки і її статус.

3.1.2. Опис клієнтської (frontend) частини вебдодатку

Розробка даного коду починалася з визначення структури даних, необхідної для зберігання та обробки інформації у вебдодатку. Для реалізації

цього використовувався Mongoose, об'єктно-документний моделізатор для MongoDB, який дозволяє визначати схеми для документів і створювати моделі на основі цих схем.

Перша частина коду стосується створення користувача. Було створено схему UserSchema, яка визначає структуру документа користувача. Поля включають email, password, isActivated, activationLink, first_name, та last_name. Кожне поле має свої обмеження, такі як унікальність для email та обов'язковість для всіх полів. Потім схема використовується для створення моделі User за допомогою функції model. Далі, для управління токенами було створено схему TokenSchema. Вона містить посилання на користувача за допомогою поля user, яке є об'єктом типу ObjectId і посилається на модель User, та поле refreshToken, яке є обов'язковим для зберігання токену оновлення. Після визначення схеми, створюється модель Token.

Остання частина коду відповідає за створення замовлення. Схема OrderSchema була визначена для зберігання інформації про замовлення. Ця схема включає вкладені структури для зберігання адреси та інформації про пакунок. Поле address містить дві обов'язкові строки: pickup_address та delivery_address. Поле package включає вкладені поля для ваги, типу матеріалу, розмірів (довжина, ширина, висота), ціни та коментарів, кожне з яких є обов'язковим. Також до схеми було додано поля для загальної ціни, імені та прізвища замовника, номера телефону, дати створення замовлення (з значенням за замовчуванням, яке відповідає поточному часу), та статусу замовлення (з значенням за замовчуванням "in-progress"). На основі цієї схеми створюється модель Order.

Для кожної схеми використовувалася функція model з Mongoose, щоб створити моделі, які можуть бути використані для створення, читання, оновлення та видалення документів у колекціях MongoDB, відповідно до визначених схем. Цей підхід забезпечує структурованість даних і дозволяє легко керувати даними у вебдодатку, забезпечуючи при цьому збереження цілісності та валідності даних.

Для побудови контролера пакунків було створено файл PackageController.js, який імпортує необхідні сервіси та модулі для обробки запитів. Код починається з імпорту userService, packageService, бібліотеки express-validator для валідації вхідних даних, а також класу ApiError для обробки помилок.

Клас PackageController містить кілька асинхронних методів, кожен з яких відповідає за обробку певного типу запиту до сервера.

Перший метод, createPackage, обробляє запит на створення нового пакунку. Спочатку він виконує валідацію вхідних даних за допомогою функції validationResult. Якщо валідація не пройшла успішно, повертається помилка з відповідним повідомленням та масивом помилок. Якщо валідація пройшла успішно, отримується тіло запиту і передається в метод createPackage сервісу packageService. Після створення пакунку результат повертається у форматі JSON.

Метод getAllPackages відповідає за обробку запитів на отримання всіх пакунків. Він отримує параметри запиту, передає їх у метод getAllPackages сервісу packageService та повертає отриманий список пакунків у форматі JSON.

Метод getPackage обробляє запит на отримання конкретного пакунку за його ідентифікатором. Ідентифікатор отримується з параметрів запиту та передається в метод getPackage сервісу packageService. Отриманий пакунок повертається у форматі JSON.

Метод updatePackage відповідає за оновлення даних пакунку. Тіло запиту передається у метод updatePackage сервісу packageService, після чого оновлені дані пакунку повертаються у форматі JSON.

Метод deletePackage обробляє запити на видалення пакунку за його ідентифікатором. Ідентифікатор пакунку отримується з параметрів запиту та передається у метод deletePackage сервісу packageService. Після видалення пакунку результат операції повертається у форматі JSON.

Кожен з методів містить блок try-catch для обробки можливих помилок. У випадку помилки вона передається в наступний обробник за допомогою функції next(e).

Завершується код експортом екземпляру класу PackageController, що дозволяє використовувати його методи у інших частинах програми для обробки відповідних запитів.

3.2. Тестування функціоналу інформаційної системи

На рисунку 3.1 представлений інтерфейс вебдодатку для створення нового пакунку. В лівій частині екрану знаходиться бокове меню з розділами "Home", "Користувачі", "Замовлення" та "Товари", де активний розділ "Товари" виділений. У верхній частині екрану розташоване поле для пошуку, а також є кнопка для налаштувань, позначена значком шестерні, і профіль користувача у верхньому правому куті.

У центральній частині екрану міститься форма для введення даних про новий пакунок, яка включає такі поля:

- "Вага", де користувач може ввести вагу пакунку.
- "Тип матеріалу", призначене для введення типу матеріалу, з якого зроблений пакунок.
- "Розміри", де користувач може вказати розміри пакунку.
- "Орієнтовна вартість", поле для введення приблизної вартості пакунку.
- "Додаткові відомості", поле для введення додаткової інформації про пакунок. Під формою розташована кнопка "Створити", яка, очевидно, використовується для збереження введених даних і створення нового пакунку.

На рисунку 3.2 представлений інтерфейс вебдодатку для перегляду списку замовлень. Ця частина містить таблицю зі списком замовлень. Таблиця має кілька колонок: "Вага", "Тип матеріалу", "Розміри", "Орієнтовна вартість" та "Додаткові відомості".

В рядках таблиці представлені дані про конкретні замовлення. Наприклад, перший рядок містить інформацію про товар з вагою 28 кг, типом матеріалу "Метал", розмірами 30x10x40, орієнтовною вартістю 400 грн та додатковими відомостями "Будівельні матеріали стійкі до ударів". Другий рядок містить дані про товар з вагою 0.2 кг, типом матеріалу "Вата", розмірами 10x20x10, орієнтовною вартістю 100 грн та додатковими відомостями "Маленька дитяча іграшка". Крім цього, кожен рядок таблиці має кнопки для редагування та видалення відповідного запису. У верхньому правому куті екрану розташований профіль користувача.

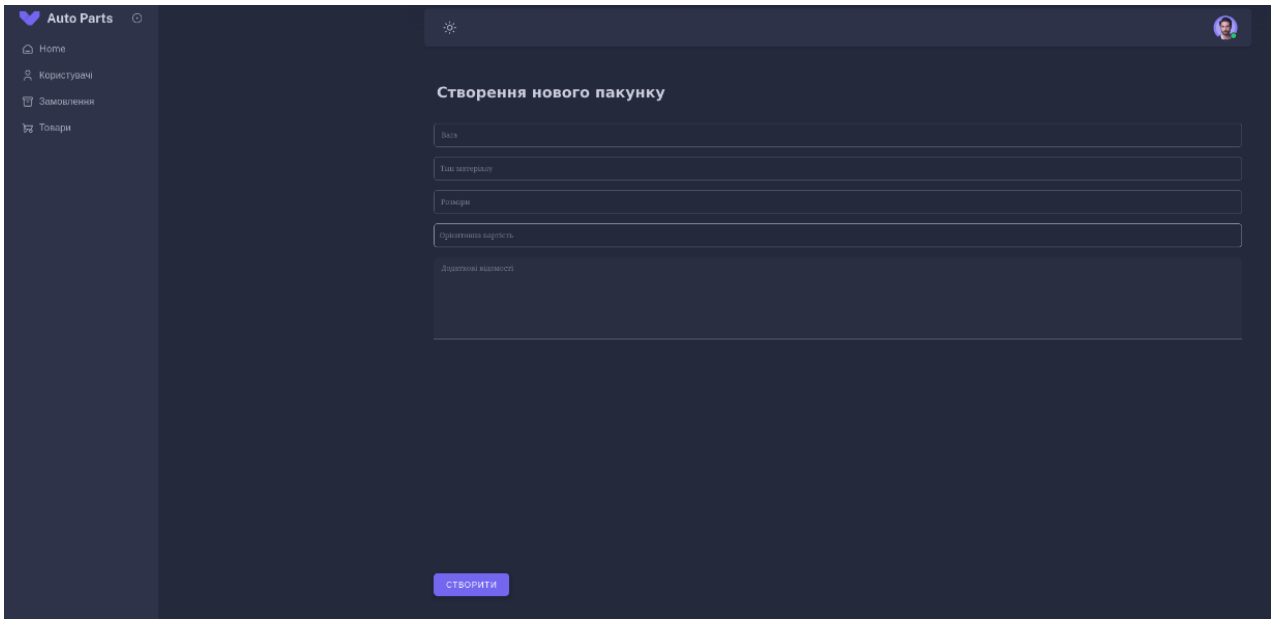

Рисунок 3.1. Створення замовлення

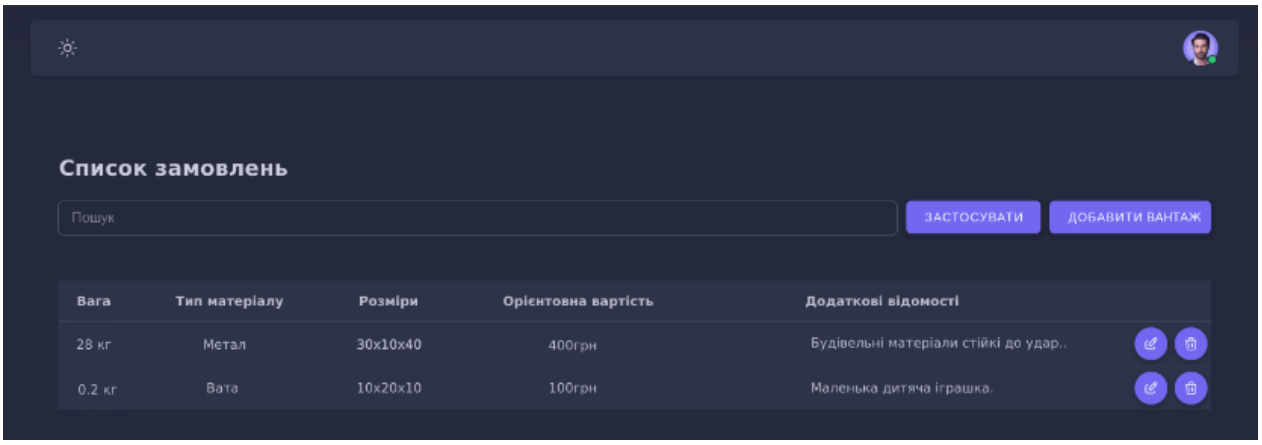

Рисунок 3.2. Список замовлень

На рисунку 3.3 представлений інтерфейс сторінки вебдодатку для управління замовленнями. В цьому меню є кілька розділів.

В центрі сторінки розміщений основний заголовок "Список доставок". Під ним знаходиться пошуковий рядок, де можна ввести номер телефону для пошуку замовлень, а поруч є кнопка "ЗАСТОСУВАТИ" для запуску пошуку.

Нижче розташована таблиця із замовленнями, яка містить кілька колонок. У першій колонці зазначено ім'я замовника, наприклад, "Максим". Друга колонка відображає номер телефона замовника, наприклад, "380964426617". У третій колонці вказана сума до сплати, наприклад, "800₴". Четверта колонка містить дату замовлення, наприклад, "2024-05-06". Остання колонка показує адресу доставки але не повністю, наприклад, "м. Рівне, вул. Академіка Грушевського 1".

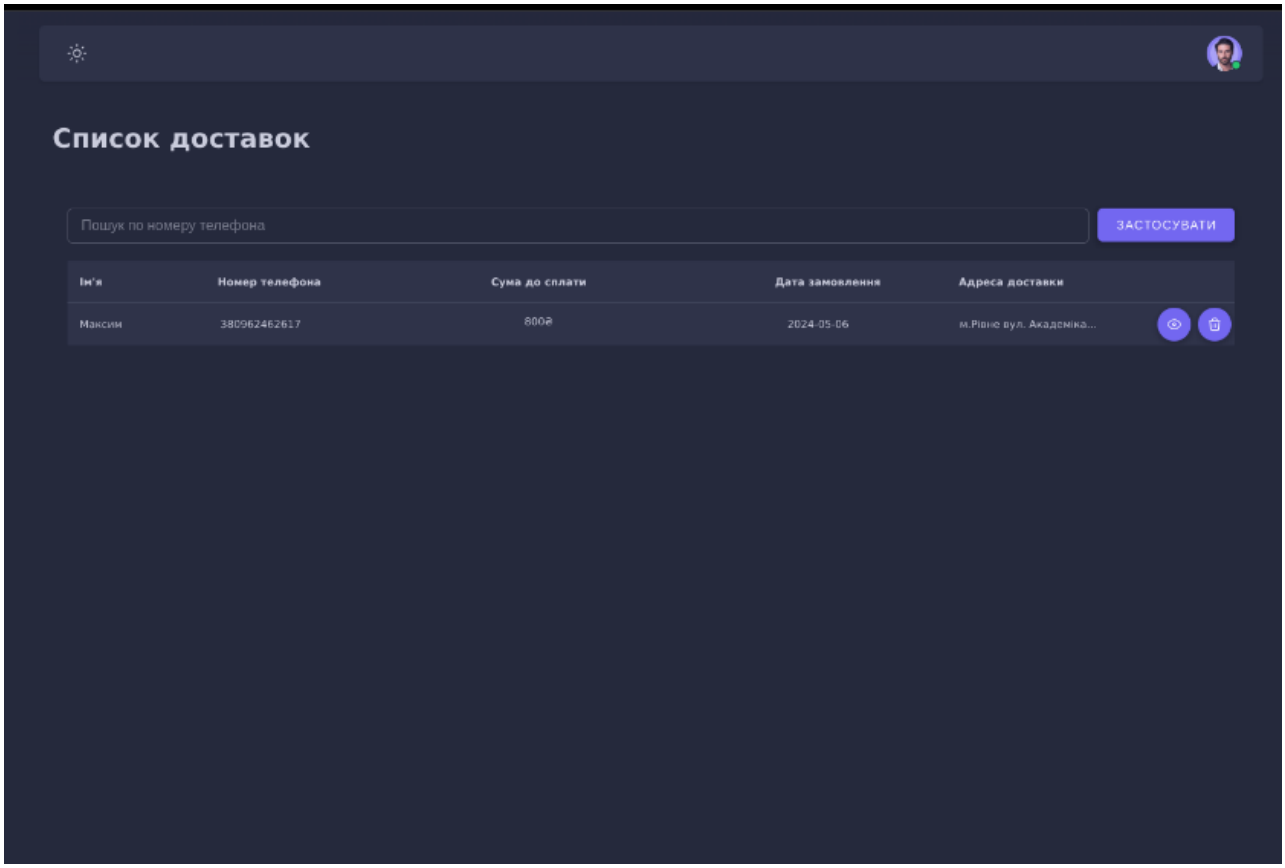

Рисунок 3.3. Список невиконаних доставок

3.3. Можливі варіанти розвитку додатку

Для покращення вебдодатку варто здійснити наступні кроки:

- *Аналіз потреб користувачів*. Провести дослідження потреб та вимог цільової аудиторії щодо функціоналу та можливостей додатку. Зібрати відгуки та пропозиції від поточних користувачів для покращення та розширення функціоналу.
- *Планування розробки*. Визначити пріоритети нового функціоналу на основі аналізу потреб користувачів та стратегії розвитку додатку. Розробити часовий графік для впровадження нових функцій та виправлення помилок.
- *Розширення функціоналу*. Додати можливість створення та зберігання шаблонів доставок для подальшого використання. Розглянути можливість інтеграції з іншими системами, такими як системи управління запасами або платформи для замовлення транспорту. Додати опції для підтримки різних видів доставок та вибору оптимальних маршрутів.
- *Мобільний додаток для водіїв та кур'єрів*. Розробити мобільний додаток, який дозволить водіям та кур'єрам отримувати та оновлювати інформацію про замовлення. Забезпечити можливість виконання доставки та спілкування з клієнтами через мобільний додаток.
- *Аналітика та звітність*. Розробити модуль для аналізу даних та створення звітів щодо ефективності логістичних операцій. Забезпечити можливість користувачам отримувати статистику та аналітичні звіти для прийняття стратегічних рішень.
- *Глобальна експансія*. Розглянути можливість розширення додатку на міжнародному рівні та надання послуг доставки вантажів у різних країнах. Провести аналіз міжнародного ринку та визначити стратегію входу на нього.

Розширення функціоналу веб-додатку може значно підвищити його корисність та привабливість для користувачів. Ось кілька ідей для нових функцій, які можна додати:

- *Реальний час відстеження вантажів*: можливість відстеження місцезнаходження вантажу в реальному часі за допомогою GPS. Це дозволить користувачам бачити поточне місце знаходження їхнього пакунка на карті.
- *Сповіщення та оповіщення*: впровадження системи сповіщень, яка буде інформувати користувачів про статус їхніх доставок через SMS, email або push-повідомлення.
- *Розширена аналітика*: можливість перегляду статистики та звітів для користувачів та адміністрації. Це може включати в себе аналіз часу доставки, кількості відправлень, популярних маршрутів тощо.
- *Інтеграція з платіжними системами*: інтеграція з популярними платіжними системами, такими як PayPal, Stripe або Google Pay, щоб користувачі могли оплачувати послуги доставки безпосередньо через додаток.
- *Мобільний додаток*: розробка мобільної версії вебдодатку для iOS та Android, щоб користувачі могли користуватися послугами в смартфоні.
- *Чат підтримки*: впровадження чат підтримки для надання допомоги користувачам у режимі реального часу.
- *Система лояльності*: програма лояльності для постійних клієнтів, яка буде пропонувати знижки або бонуси за часте користування послугами.
- *Рекомендаційна система*: алгоритм рекомендацій, який на основі попередніх замовлень користувача пропонуватиме оптимальні варіанти доставки або додаткові послуги.
- *Мульти-користувацький доступ*: можливість створення корпоративних облікових записів, які дозволять кільком користувачам управляти замовленнями та переглядати статистику.

- *Автоматизація маршрутизації*: впровадження системи автоматичного планування маршрутів для оптимізації часу та вартості доставки.
- *Підтримка декількох мов*: підтримка кількох мов, щоб ваш додаток міг використовуватися користувачами з різних країн.
- *Оцінки та відгуки*: можливість залишати оцінки та відгуки про сервіс доставки, що допоможе покращити якість обслуговування.

Ці функції допоможуть зробити вебдодаток більш привабливим та функціональним, забезпечуючи користувачів додатковими можливостями та покращуючи їхній досвід користування ним.

ВИСНОВКИ

У процесі виконання даної дипломної роботи було досліджено та розглянуто можливості різних хмарних провайдерів, таких як Google Cloud, Amazon Web Services (AWS) та Microsoft Azure. На основі аналізу їх характеристик, функціональності та вартості було обрано найкращий варіант для реалізації інформаційної системи на транспортну тематику.

Результатом роботи стало створення вебдодатка, який забезпечує ефективне управління транспортними даними. Вебдодаток був розроблений на основі платформи Node.js, що забезпечило високу продуктивність та масштабованість системи. Для зберігання та обробки даних була використана база даних MongoDB, яка дозволяє гнучко та ефективно працювати з великими обсягами даних.

Розроблена інформаційна система має низку переваг:

- Забезпечує зручний доступ до транспортних даних з будь-якої точки світу завдяки використанню хмарних технологій.
- Підвищує ефективність управління транспортними процесами за рахунок автоматизації та інтеграції даних.
- Дозволяє швидко масштабувати систему відповідно до потреб користувачів завдяки використанню хмарної інфраструктури.
- Гарантує високу надійність та безпеку зберігання даних завдяки сучасним методам захисту, що пропонують хмарні провайдери.

Таким чином, мету роботи було досягнуто: розроблено інформаційну систему на транспортну тематику, яка базується на використанні хмарної бази даних, що дозволяє ефективно управляти транспортними потоками та логістичними операціями. Даний вебдодаток може бути використаний для покращення роботи транспортних компаній, зниження витрат та підвищення рівня обслуговування клієнтів. Розробка також відкриває перспективи для подальшого розвитку та вдосконалення системи з використанням новітніх технологій та інноваційних рішень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вікіпедія. URL: https://uk.wikipedia.org/ (дата звернення 25.04.2024)
2. Amazon Web Services (AWS). URL: https://aws.amazon.com/ (дата звернення 18.04.2024)
3. Google Cloud documentation. URL: https://cloud.google.com/docs (дата звернення 25.04.2024)
4. Azure documentation. URL: https://learn.microsoft.com/en-us/azure/ (дата звернення 18.04.2024)
5. ItdeuCenter: Топ 5 хмарних провайдерів: плюси та мінуси. URL: https://blog.iteducenter.ua/sysadministration/top-5-cloud-providers-advantages-and-disadvantages/ (дата звернення 09.02.2024)
6. Azure vs. Google Cloud (2023): всебічне порівняння. URL: http://surl.li/pqoeg (дата звернення 17.02.2023)
7. Основні переваги хмарної платформи Google Cloud і чому Ви повинні вибрати її. URL: https://pingvin.pro/blogy/software-blogy/osnovni-perevagy-hmarnoyi-platformy-google-cloud-i-chomu-vy-povynni-vybraty-yiyi.html (дата звернення 17.03.2024)
8. Cloud Pricing Comparison: AWS vs. Azure vs. Google Cloud Platform in 2024. URL: https://cast.ai/blog/cloud-pricing-comparison-aws-vs-azure-vs-google-cloud-platform/ (дата звернення 17.03.2024)
9. Cloudfresh blog. URL: https://cloudfresh.com/ua/cloud-blog/ (дата звернення 18.04.2024)
10. Зінченко О.В., Іщеряков С.М., Прокопов С.В., Сєрих С.О., Василенко В.В. Хмарні технології. Навчальний посібник. К: ФОП Гуляєва В.М., 2020. 74 с.
11. Гайдаржи В., Ізварін І. Бази даних в інформаційних системах. К.: Університет "Україна", 2018. 418 с.
12. Vuetify – A Vue Component Framework. URL: https://vuetifyjs.com/en/ (дата звернення 17.03.2024)

13. Vue.js – The Progressive JavaScript Framework. URL: https://vuejs.org/ (дата звернення 17.03.2024)
14. Node.js v22.2.0 documentation. URL: https://nodejs.org/docs/latest/api/ (дата звернення 24.04.2024)
15. Express4.19.2. Fast, unopinionated, minimalist web framework for Node.js. URL:https://expressjs.com/ (дата звернення 24.04.2024)
16. MongoDB Documentation. URL: https://www.mongodb.com/docs/ (дата звернення 17.03.2024)