Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

УДК 004.942

На правах рукопису

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: «Веб-додаток для моніторингу якості автодоріг»

Спеціальність – 121 «Інженерія програмного забезпечення»

ПОЯСНЮВАЛЬНА ЗАПИСКА

ВКБР.ІПЗ – 22.00.00.000 ІПЗ

Студент

ІПЗ-42 _____ /Михайло КАТЕРИНИЧ/

Науковий керівник

к.ф.-м.н.,доц._____/Оксана КОВТУН /

Консультант

з питань нормконтролю

_____/Тамара ЧАПОВСЬКА/

Допускається до захисту

Завідувач кафедри

д.т.н.,проф._____/Олексій БИЧКОВ/

Київ – 2021

Рішенням Екзаменаційної комісії

випускна кваліфікаційна робота студента

_____

захищена з оцінкою

_____

Голова Екзаменаційної комісії

професор, доктор техн. наук Бондарчук А.П.

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖЕНО:

Завідувач кафедри програмних систем і технологій

_____(Олексій БИЧКОВ)

« » _____20__ р.

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Катериничу Михайлу Михайловичу

1. **Тема бакалаврської роботи «Веб-додаток для моніторингу якості автодоріг»** керівник проекту (роботи) Ковтун Оксана Іванівна, к.ф.-м.н., доцент затверджена наказом вищого навчального закладу від «11»_листопада__2020 р. №6
2. **Строк здачі студентом закінченої роботи «___»_____2021 р.**
3. **Вхідні дані до проекту (роботи):** підручники, навчальні посібники, офіційна документація, статті зарубіжних авторів, інтернет-ресурси.
4. **Зміст пояснювальної записки (перелік питань, що їх належить розробити)**

1.Аналіз стану питання та постановка задач дослідження

2.Функціонал веб-додатку

3.Архітектура та структура серверної частини веб-додатку

4.Архітектура та структура клієнтської частини веб-додатку

5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень)

1. Інтерфейс сайту «Інтерактивна мапа Укравтодору» (рис. 1.2.1, ст. 17)
2. Інтерфейс телеграм-боту «E-transport» (рис. 1.2.2, ст. 18)
3. Модуль авторизації (рис. 2.1.1, ст. 21)
4. Головна сторінка (рис. 2.1.2, ст. 21)
5. Профіль автодороги (рис. 2.1.3, ст. 22)

6. Схема архітектури динамічного сайту (рис. 3.5.1, ст. 33)
7. Збережені дані користувачів у базі даних у вигляді документів(рис. 3.5.2, ст. 34)
8. Діаграма основного потоку даних у серверній частині додатку (рис. 3.6.1, ст. 34)
9. Структура файлів і папок проекту (рис. 3.7.1, ст. 40)
10. Контролери обробки запитів для роботи з автодорогами (рис. А.1, ст. 50)
11. Схема даних з критеріями оцінювання автодоріг (рис. А.2, ст. 51)
12. Контролери обробки запитів для реєстрації та авторизації користувачів (рис. А.3, ст. 52)
13. Компонент головної сторінки веб-додатку (рис. Б.1, ст. 54)
14. Компонент App додатку (рис. Б.2, ст. 55)

6. Консультанти з роботи із зазначенням розділів роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Розділ 1. Аналіз стану питання та постановка задач дослідження	Ковтун О. І.	15.11.2020	19.01.2021
Розділ 2. Функціонал веб-додатку	Ковтун О. І.	22.01.2021	01.03.2021
Розділ 3. Архітектура та структура серверної частини веб-додатку	Ковтун О. І.	02.03.2021	15.03.2021
Розділ 4. Архітектура та структура клієнтської частини веб-додатку	Ковтун О. І.	17.03.2021	20.04.2021

7.

Дата видачі завдання _____

Керівник _____ (Оксана КОВТУН)
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ (Михайло КАТЕРИНИЧ)
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів бакалаврської роботи	Термін виконання етапів роботи	Відмітка про виконання
1.	Уточнення постановки задачі	15.11.2020-01.12.2020	Виконано
2.	Аналіз літератури	02.12.2020 - 15.01.2021	Виконано
3.	Огляд та аналіз існуючих методів, концепцій та алгоритмів вирішення завдання	17.01.2021 - 27.01.2021	Виконано
4.	Побудова алгоритмічної моделі основних процесів	27.01.2021 - 06.03.2021	Виконано
5.	Розробка програмного забезпечення	06.03.2021 - 07.04.2021	Виконано
6.	Тестування розробленого програмного забезпечення	08.04.2021 - 16.05.2021	Виконано
7.	Оформлення і друк пояснювальної записки	20.05.2021 - 01.06.2021	Виконано
8.	Отримання рецензії	02.06.2021 - 05.06.2021	Виконано
9.	Оформлення презентації	05.06.2021-09.06.2021	Виконано

10.	Затвердження пояснювальної записки роботи виконувачем обов'язки завідувача кафедри	31.05.2021	Виконано
11.	Захист дипломної роботи	23.06.2020	

Студент _____ **(**Михайло КАТЕРИНИЧ**)**

(підпис)

Керівник роботи _____ **(**Оксана КОВТУН**)**

(підпис)

АНОТАЦІЯ

Випускна кваліфікаційна бакалаврська робота: 56 с., 9 рис., 2 дод., 20 джерел.

Тема: веб-додаток для контролю якості автодоріг.

Об’єкт дослідження: технології для організації громадського контролю за владою.

Мета роботи: розробка програмного забезпечення для контролю якості автомобільних доріг державного значення в Україні.

Предмет дослідження: технології розробки динамічних веб-додатків.

Результати дослідження: досліджено можливості застосування веб-технологій для розробки веб-додатку, що надає можливість громадській спільноті фіксувати та обговорювати виявлені недоліки щодо автодоріг, сповіщати про них інших користувачів шляхів та органи влади для їх усунення.

Висновок: у результаті виконання дипломної роботи було створено динамічний веб-додаток, що має функціонал для складання рейтингу автодоріг державного значення України, який дозволяє громадянам більш ефективно слідкувати за виконанням владою своїх обов’язків щодо забезпечення якості дорожньої інфраструктури.

АННОТАЦИЯ

Выпускная квалификационная бакалаврская работа: 56 с., 9 рис., 2 доп., 20 источников.

Тема: веб-приложение для контроля качества автодорог.

Объект исследования: технологии для организации общественного контроля за властью.

Цель работы: разработка программного обеспечения для контроля качества автомобильных дорог государственного значения в Украине.

Предмет исследования: технологии разработки динамических веб-приложений.

Результаты исследования: исследованы возможности применения веб-технологий для разработки веб-приложения, позволяет общественному сообществу фиксировать и обсуждать выявленные недостатки автодорог, сообщать о них другим пользователям путей и органам власти для их устранения.

Вывод: в результате выполнения дипломной работы было создано динамическое веб-приложение, которое содержит функционал для составления рейтинга автодорог государственного значения Украины, который позволяет гражданам более эффективно следить за выполнением властью своих обязанностей по обеспечению качества дорожной инфраструктуры.

ANNOTATION

Graduation qualification bachelor's thesis: 56 pp., 9 figs., 2 additions, 20 sources.

Topic: web application for control quality of roads.

Object of research: technologies for the organization of public control over power.

Purpose: development of software for quality control of state roads in Ukraine.

Subject of research: technologies for dynamic web applications developing.

The results of the study: explored the possibilities of using web technologies to develop a web application, allows the community to record and discuss the identified shortcomings, to report them to other road users and authorities to address them.

Conclusion: as a result of the thesis was created a dynamic web application, has the functionality to compile a rating of roads of state importance in Ukraine, which allows citizens to more effectively monitor the performance of their duties to ensure the quality of road infrastructure.

ЗМІСТ

Стр.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД - база даних

ПЗ - програмне забезпечення

ОС - операційна система

КІ - користувацький інтерфейс

ВД - веб-додаток

ВСТУП

Одним з головних напрямків Української держави є активна взаємодія всіх його органів з інститутами громадянського суспільства. У цьому процесі важливим є контроль з боку громадян за дотриманням виконання державними органами своїх обов'язків перед народом. В сучасних умовах її роль і значення громадського контролю багаторазово зростають в зв'язку з потребами модернізації державного управління, боротьби з корупцією, підвищення якості реалізації державних функцій і надання державних послуг.

Важко не погодитися з тим, що громадський контроль - це діяльність, перш за все, інститутів громадянського суспільства та окремих громадян.

З розвитком Інтернет-технологій з’являються можливості розробки зручних і масових ресурсів, що можуть спростити комунікацію між громадянами країни та владою. Велика зацікавленість до розробки та використання веб-ресурсів для громадського контролю обумовлена очікуванням підвищення ефективності діяльності влади.

Актуальною проблемою сучасної України є незадовільний стан її автодоріг. У результаті користування автомобільних шляхів з неякісним покриттям стрімко підвищується кількість дорожньо-транспортних пригод, що часто призводить до травмування чи смерті громадян. До того ж, через низьку якість дорожньої інфраструктури значно знижується зручність пересування автошляхами і збільшується час, витрачений на подолання маршруту.

Тому, було вирішено дослідити можливість розробки веб-ресурсу для громадського моніторингу та контролю якості покриття та інфраструктури автошляхів України державного значення, що дозволить громадській спільноті фіксувати та обговорювати виявлені недоліки, сповіщати про них інших користувачів шляхів та органи влади для їх усунення.

Засобами розробки були актуальні методи динамічної розробки веб-

додатків з використанням мови програмування JavaScript, БД MongoDB, та відповідних фреймворків для роботи з нею.

РОЗДІЛ 1

АНАЛІЗ СТАНУ ПИТАННЯ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз предметної області

Сьогодні існує значна кількість громадських організацій, що слідкують за виконанням органами влади своїх обов'язків. В умовах розвитку інформаційних технологій з'явилася можливість автоматизації та оптимізації цього процесу за допомогою створення програмного забезпечення.

Веб-сервіси для громадського контролю дозволяють користувачам оперативно доносити зауваження та пропозиції до відповідних органів влади, активні громадяни можуть комунікувати та консолідуватися задля швидкого вирішення проблеми.

Процес розробки веб-ресурсів складається з:

- Визначення технічного завдання на розробку веб-додатку;
- визначення структурної схеми веб-додатку – проектування взаємного розташування розділів додатку;
- веб-дизайн – створення інтерфейсу користувача;
- розробку ПЗ.
- тестування і впровадження.

Розглянемо такі поняття як: громадський контроль, веб-додаток, веб-портал.

Громадський контроль - це спостереження за діяльністю органів державної і муніципальної влади, інших державних органів та посадових осіб, оцінка законності та ефективності цієї діяльності, а також прийняття правових заходів по припиненню виявлених порушень прав і свобод людини з боку зазначених органів і посадових осіб. Цивільний контроль як система практик є однією з основних функцій громадянського суспільства.

Веб-додаток - це прикладне програмне забезпечення, яке працює на веб-сервері, на відміну від програм на базі комп'ютерних програм, які запускаються локально в ОС пристрою. Доступ до веб-додатків здійснюється користувачем через веб-браузер з активним підключенням до мережі. Ці програми програмуються за допомогою змодельованої структури «клієнт-сервер» - користувач («клієнт») взаємодіє з додатком через зовнішній сервер[1].

Веб-портал - це веб-платформа, яка збирає інформацію з різних джерел в єдиний КІ і представляє користувачам найбільш релевантну інформацію для їхнього контексту.

Отже, мета веб-додатку для громадського контролю полягає у наданні користувачу можливості ділитися та отримувати інформацію про стан доріг, взаємодіяти з іншими користувачами, доносити інформацію до відповідних органів влади.

1.2 Порівняльний аналіз аналогів

На сьогоднішній день існує декілька реалізацій сервісів для громадського контролю.

Проаналізуемо деякі з них.

Розглянемо веб-сайт «інтерактивна мапа Укравтодору». Сайт представлений у вигляді мапи, на якій відмічені місця виникнення проблем на автодорозі. При натисканні на мітку можемо дізнатися про статус виконання робіт.

Інтерфейс сайту доволі зручний, простий і зрозумілий через малу кількість елементів. Відсутнє нагромадження кольорів.

На головній сторінці присутня мапа України з можливістю обрати область у якій користувач бажає залишити мітку зі зверненням. Ця функція продубльована випадаючим списком з областями у верхній частині сайту.

Також у хедері можна обрати пункти «звернення», «ремонти доріг», «утримання доріг», присутня кнопка «створити звернення» та іконка профілю користувача.

Із недоліків можна виділити відсутність можливості пошуку дороги чи адреси за назвою. Також користувачі не можуть залишати свої коментарі під зверненнями, що не дає можливості більш якісно контролювати виконання робіт.

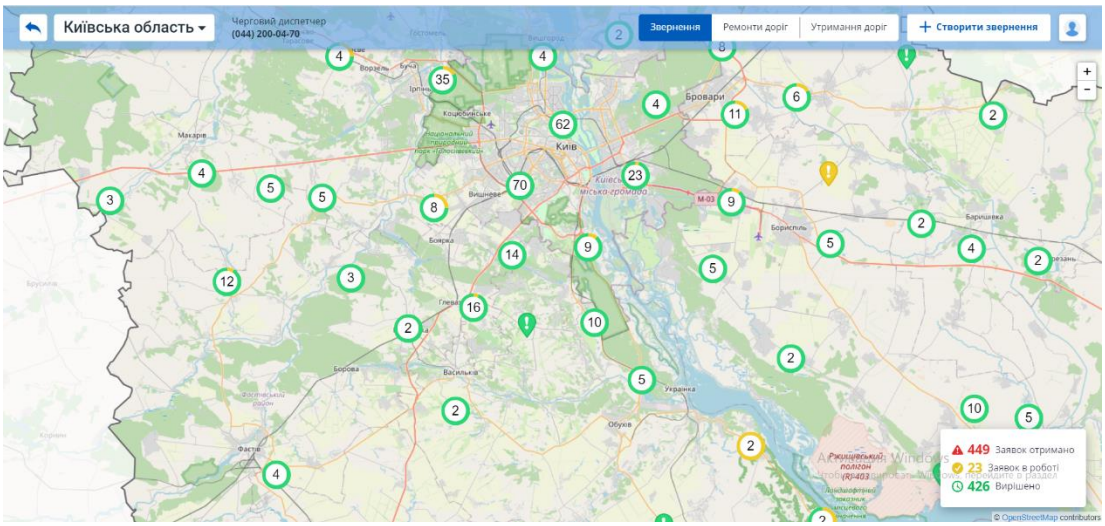

Рис. 1.2.1. Інтерфейс сайту «Інтерактивна мапа Укравтодору»

Наступним аналогом є телеграм-бот «E-transport» від Транспортного порталу електронних послуг. За допомогою цього сервісу користувач має змогу отримати інформацію щодо ремонтних робіт на дорогах а також подати заяву на виконання ремонтних робіт на обраній ділянці дороги.

Із переваг можна виділити зручність користування, можливість пошуку ділянок автодоріг за кодом дороги, областю та геолокацією, додавання заяв на ремонт дороги за місцезнаходженням користувача.

Недоліками даного сервісу є відсутність можливості подавати заявки «заднім числом», тобто коли користувач не знаходиться поряд з проблемною ділянкою та неможливість комунікації користувачів задля більш якісного контролю за виконанням робіт.

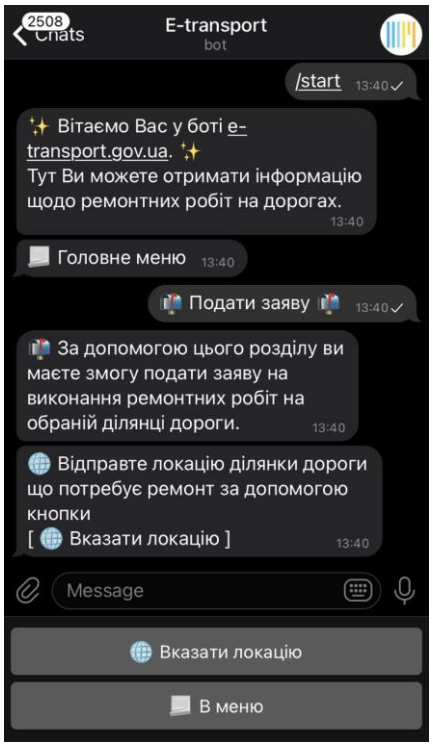

Рис. 1.2.2. Інтерфейс телеграм-боту «E-transport»

1.3 Постановка задач розробки

Виходячи з того, що завданням роботи є створення веб-додатку для контролю якості автодоріг, можна виділити такі задачі для виконання:

- Проаналізувати можливість практичного застосування веб-додатку в рамках предметної області;
- Спроектувати логічну та функціональну структури сайту;
- Порівняти можливості мов програмування і обрати найбільш зручну та ефективну для реалізації серверної та клієнтської частини проекту;
- Обрати базу даних, що відповідає умовам функціональності ресурсу;
- Розробити серверну частину веб-додатку;
- Розробити інтерфейс користувача;
- Провести тестування готового програмного продукту.

1.4 Висновки до розділу

В процесі роботи над першим розділом було проведено огляд та аналіз предметної області та визначено, що з розвитком інформаційних технологій почали активно розвиватися сервіси для громадського контролю. В свою чергу з’являється можливість для розробки веб-додатку для контролю якості автодоріг.

Було проаналізовано аналоги та сформульовано задачі розробки.

РОЗДІЛ 2

ФУНКЦІОНАЛ ВЕБ-ДОДАТКУ

Сьогодні в Глобальній мережі на будь-яку тематику знайдеться як мінімум кілька сотень веб-проектів, тому власники Інтернет-ресурсів так активно борються за частку на найбільшій маркетингової майданчику - Всесвітній Павутині. Головний інструмент суперництва - якість і зручність веб-сторінок. Ось чому в ході розробки сайту так важливо не шкодувати часу і сил на підбір і реалізацію відповідного функціоналу ресурсу, що вдається далеко не кожному.

Веб-ресурси, які підтримують лише візуальну взаємодію, без технічної можливості виконання більш складних інтерактивних дій не є конкурентно спроможними. Тому навіть найпримітивніші сайти-візитки створюються з використанням програмних засобів, що дозволяють користувачам комунікувати між собою (соціальні мережі, форуми), доносити свою думку до інших користувачів.

У сучасних технічно складних проектах, таких як веб-порталах, обов’язково створюється функціонал для розміщення новин, пошуку по сайту комунікації користувачів за допомогою коментарів тощо.

Таким чином, функціональний веб-ресурс очима користувача - це сайт з підтримкою технічних засобів, що дозволить вирішити задачу відвідувача швидко, ефективно і зручно[2].

2.1 Функціонал веб-додатку для моніторингу якості автодоріг

Після аналізу веб-ресурсів зі схожою тематикою було вирішено реалізувати функціонал, що повністю вирішує потреби користувача.

Функціональні модулі, що використовуються у веб-додатку для моніторингу якості автодоріг:

- **Реєстрація та авторизація**

Модуль надає можливість реєстрації відвідувачів на сайті з подальшою можливістю авторизації. При цьому створюється база даних з профілями користувачів.

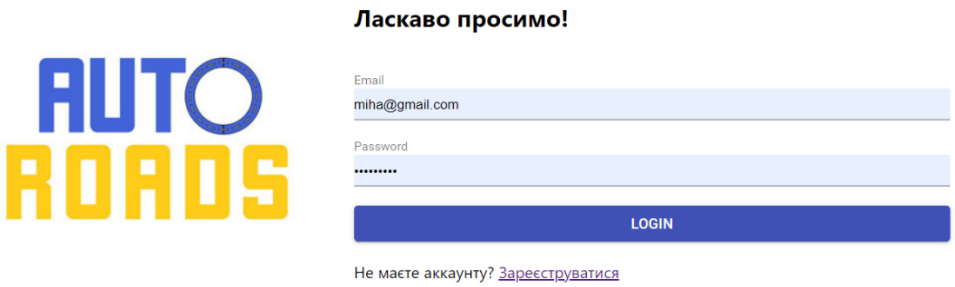

Рис. 2.1.1. Модуль авторизації

- **Головна сторінка**

Модуль головної сторінки містить навігаційну панель, надає можливість пошуку автодороги за назвою. Містить кнопки виходу з аккаунту та кнопку переходу в профіль користувача. Дороги відображаються у вигляді списку.

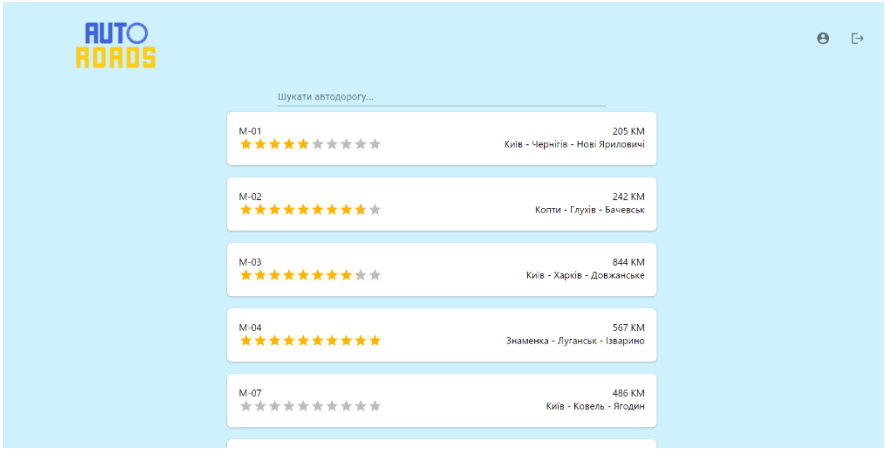

Рис. 2.1.2. Головна сторінка

- **Профіль автодороги**

Модуль надає можливість користувачу переглянути профіль дороги, який містить основну інформацію про неї. Є можливість переглянути середній

рейтинговий бал дороги та оцінити її за критеріями.

Рис. 2.1.3. Профіль автодороги

- **Профіль користувача**

  Модуль містить в собі профіль користувача з анкетними даними.

2.2 Висновки до розділу

В процесі виконання другого розділу дипломної роботи було спроектовано основні функціональні модулі веб-додатку для контролю якості автодоріг. Було описано модулі реєстрації та авторизації користувачів, основну головну сторінку сайту та профілі користувача та дороги.

Ці функціональні модулі ефективно і зручно вирішують задачі, що повинен вирішувати даний веб-додаток.

РОЗДІЛ 3

АРХІТЕКТУРА ТА СТРУКТУРА СЕРВЕРНОЇ ЧАСТИНИ ВЕБ-ДОДАТКУ

Сереверна частина веб-додатку, зазвичай, складається з двох рівнів: прикладного та рівня даних[3].

На прикладному рівні обробляється логіка, яка забезпечує функціонування веб-додатка. Тут виконуються будь-які операції і алгоритми, що забезпечують роботу бізнес-функцій програми.

Рівень даних зберігає, організує і управляє доступом до даних додатка з використанням бази даних. Реалізується в коді на стороні сервера.

3.1 Обрання мови для серверної частини веб – додатку

Сьогодні існує безліч мов для програмування серверної частини веб додатків. Розглянемо деякі з них:

- **PHP**

На сьогоднішній день PHP - це найчастіше використовувана скриптова серверна мова програмування. Трохи більше 80% веб-сайтів працюють на PHP. Це була перша мова програмування, розроблена спеціально для Інтернету, і це призвело до її домінування в епоху Web 2.0 (блогінг, створення контенту) 2000-х. Крім того, PHP є основою Wordpress, на якій працює 25% веб-сайтів сьогодні, включаючи найбільш популярні блоги та веб-сайти з новинами.

- **Java**

Java - ще одна популярна мова, яку використовують для великих веб-сайтів. Однак вона є занадто громіздка для багатьох невеликих додатків, де

можна досягти однакового результату чимось простішим. Багато великих корпоративних програм, таких як банківські та страхові, використовують його для спілкування з іншими системами, наприклад, мейнфреймами. Він потужний з точки зору масштабованості.

- **Ruby**

Ruby в основному популярний для невеликих додатків, оскільки він підходить для швидкої веб-розробки. Rails - найпопулярніший фреймворк для роботи з Ruby. Сьогодні його використовують у багатьох стартапів, які хочуть швидко розробляти програмне забезпечення та виходити на ринок.

- **Python**

Python являє собою інтерпретовану, об'єктивно-орієнтовану мову програмування. Вона, звичайно, проста і містить не велику кількість ключових слів, разом із тим вона дуже гнучка і виразна. Ця мова більш високого рівня, ніж Pascal, C ++ і, звичайно, C, що досягається, в основному, завдяки вбудованим високорівневим структурам даних.

- **Node.js(JavaScript)**

Node.js - найновіший у списку (випущений у 2009 році) та найшвидший, що зростає сьогодні. Він надає можливість запускати JavaScript-код на сервері. Велика перевага його в тому, що вам не доведеться вивчати нову мову для back-end-програмування. Ви можете використовувати JavaScript для front-end рендерингу, а потім повторно використовувати його в back-end, що є однією з ключових переваг написання JavaScript. Одне, що Node.js дуже добре вирізняє з-поміж більшості інших мов, це додатки в реальному часі. Тому, якщо ви розробляєте щось, для чого потрібно працювати в режимі реального часу, наприклад, чати або ігри, то ця мова вам відмінно підійде. Також, оскільки він новіший, ніж інші мови, він оснащений деякими зручними та сучасними функціями, яких не вистачає у старих мовах програмування, які роблять

розробку більш зручною. Наприклад, менеджер пакунків NPM, який поставляється разом з ним за вмовчуванням[4].

3.2 Чому було обрано саме NodeJS?

Для створення серверних інструментів веб-додатку для моніторингу якості автодоріг було обрано програмну платформу NodeJS, що модифікує мову JavaScript в мову загального призначення. Платформа призначена для використання поза контекстом браузера (тобто працює безпосередньо на ОС комп'ютера або сервера). Таким чином, середовище упускає специфічні для браузера API і додає підтримку більш традиційних API-програм ОС, включаючи HTTP та бібліотеки файлової системи[5].

З точки зору розробки веб-сервера, Node має такі переваги:

- Гарна пропускна здатність. Node був створений для підвищення пропускної здатності та оптимізації веб-додатків і є хорошим рішенням для багатьох поширених проблем веб-розробки.
- Код написано "простим старим JavaScript", що означає, що менше часу витрачається на "зміщення контексту" між мовами, коли ви пишете як на стороні клієнта, так і на стороні сервера.
- JavaScript є відносно новою мовою програмування і отримує переваги від вдосконалення дизайну мови порівняно з іншими традиційними мовами веб-сервера (наприклад, Python, PHP тощо). Багато інших нових і популярних мов компілюються / перетворюються на JavaScript, тому, ви можете також використовувати ClojureScript, CoffeeScript, LiveScript, TypeScript, Scala тощо.
- Менеджер пакетів вузлів (NPM) надає доступ до великої кількості пакетів для багаторазового використання.
- Node.js портативний. Він доступний в майже усіх операційних системах. Крім того, він добре підтримується багатьма постачальниками веб-

хостингів, які часто мають в собі документацію та інфраструктуру для розміщення веб-ресурсів з використанням Node.

- Вона має дуже активну сторонню екосистему та спільноту розробників, які завжди готові допомогти.
- Середовище Node.js з відкритим кодом так само надає можливість кешування обраних модулів. Кожного разу, коли надходить запит на перший модуль, він залишається у cash-пам'яті програми.
- Розробникам не доведеться заново виконувати коди, оскільки кешування дозволяє програмам швидше завантажувати веб-сторінки та швидше реагувати на запити користувача.

JavaScript, який є основою NodeJS використовуєть понад 80% розробників та 95% веб-сайтів[6].

3.3 Обрання веб-фреймворку для роботи з Node.js

Де-факто, стандартним фреймворком для Node.js є Express, тому саме його і було обрано. Він дозволяє додати специфічну обробку для різних HTTP-запитів, наприклад, GET, POST, DELETE тощо. Також він дає можливість окремо обробляти запити на різних шляхах URL-адрес ("routes"), обслуговувати статичні файли або використовувати шаблони для динамічного створення відповіді на запити[7].

Express – найпопулярніший веб-фреймворк Node і є базовою бібліотекою для ряду інших популярних фреймворків Node[8]. Він забезпечує механізми:

- Писати обробники для запитів з різними HTTP-запитами в різні URL-адреси (routes).

- Встановлення загальних параметрів ВД такі як порт, який буде використовуватися для з'єднання, та розташування шаблонів, які використовуються для рендерингу відповіді.
- Додавання додаткового "middleware" – функції проміжної обробки, в будь-якій точці в рамках конвеєра обробки запитів.

Хоча сам Express досить мінімалістичний, розробники створили сумісні пакети програмного забезпечення для вирішення майже будь-якої проблеми веб-розробки. Є бібліотеки для роботи з файлами cookie, сеансами, входами користувачів, параметрами URL-адреси, даними POST, заголовками безпеки та багато іншого.

Дуже важливим фактором є те, що цей фреймворк, виходячи з кількості гучних компаній, які використовують Express, кількість людей, що беруть участь у кодовій базі, та кількості людей, які надають як безкоштовну, так і платну підтримку є досить популярним, оскільки це є показником того, що він і далі буде існувати і підтримуватися.

3.4 Обрання бази даних для роботи з Node.js

Express-додаток може використовувати будь-які бази даних, підтримувані Node (сам по собі Express не визначає будь-яких конкретних додаткових властивостей і вимог для управління базами даних). Є багато популярних варіантів - PostgreSQL, MySQL, Redis, SQLite, і MongoDB[9]. При виборі бази даних слід враховувати такі фактори як час розробки, час навчання, простота реплікації і копіювання, витрати, підтримка спільноти і т. д.

Існує два підходи при роботі з базою даних:

- Використання рідної мови запитів баз даних (тобто SQL).

- Використання об'єктної моделі даних (ODM) або об'єктно-реляційної моделі (ORM). ODM / ORM представлють дані веб-сайту як об'єкти JavaScript, які потім передаються до бази даних. Деякі ORM прив'язані до певної бази даних, тоді як інші не залежать від конкретної.

Різниця між реляційними та нереляційними базами даних:

Реляційна модель має на увазі логічну структуру даних: таблиці, уявлення і індекси. Логічна структура відрізняється від фізичної структури зберігання. Такий поділ дає можливість адміністраторам керувати фізичної системою зберігання, не змінюючи даних, що містяться в логічній структурі. Наприклад, зміна імені файлу бази даних не вплине на таблиці, що в ній зберігаються .

Нереляційні бази даних, навпаки, мають гнучкими схемами для неструктурованих даних. Вони можуть зберігатися по-різному: в колонках, документах, графах або у вигляді сховища «ключ-значення». Ця гнучкість дозволяє:

- Можна створювати документи, не визначаючи їх структуру заздалегідь;
- Кожен документ може мати власну унікальну структуру;
- Синтаксис може відрізнятися в різних базах даних;
- В процесі роботи можна додавати нові поля.

Реляційна модель має тільки одну конструкцією для представлення даних і зв'язків між даними - відношенням. Наприклад, для представлення зв'язку "багато до багатьох" між двома сутностями A і B необхідно створити три відносини: два для подання сутностей A і B, а третє - для подання зв'язку. При цьому не існує ніякого механізму встановлення відмінностей між сутностями і зв'язками або між різними типами зв'язків, заданими між сутностями. Наприклад, зв'язок "один до багатьох" може мати різне значення: Has (має). Owns (володіє), Manages (управляє) і т.д. Якби була можливість

відобразити подібні відмінності в схемі, то операціям можна було надати певний сенс. Через відсутність подібних можливостей кажуть, що реляційна модель семантично перевантажена.

У порівнянні з ієрархічною і мережевою моделями реляційна модель має більш низьку швидкість доступу і вимагає більшого обсягу зовнішньої пам'яті. В даний час цей фактор не є критичним внаслідок багаторазового збільшення швидкодії комп'ютерів і такого ж зростання обсягу дискової пам'яті.

SQL БД мають форму таблиць, а в NoSQL БД дані подаються у вигляді документів, пар «ключ-значення», графів або сховищ wide-column

Нижче представлені сильні сторони MySQL:

- Дає можливість користувачам отримувати доступ до даних в системах керування базами даних.
- Дає можливість користувачам описувати дані.
- Дає можливість користувачам визначати дані в базі даних і управляти ними**.**
- Дає можливість вбудовування в інші мови з використанням модулів SQL, бібліотек і пре-компіляторів.
- Дає можливість користувачам створювати в базі даних представлення, збережені процедури, функції.

Нижче представлені сильні сторони MongoDB[10]:

- Дана БД заснована на колекціях різних документів. Кількість полів, зміст і розмір цих документів може відрізнятися. Тобто різні сутності не повинні бути ідентичні за структурою.
- Вкрай зрозуміла структура кожного об'єкта.
- Для зберігання використовуваних в даний момент даних використовується внутрішня пам'ять, що дозволяє отримувати більш швидкий доступ.

- Дані зберігаються у вигляді JSON документів.
- MongoDB підтримує динамічні запити документів (document-based query)

Реляційні БД ідеальні для роботи зі структурованими даними, структура яких не схильна до частих змін.

MongoDB, навпаки, підійде для бізнесів з швидким зростанням або для баз даних, в яких не використовуються певні схеми. Бази даних NoSQL підходять для зберігання великих обсягів неструктурованої інформації, а також хороші для швидкої розробки та тестування гіпотез.

Найкращу продуктивність можна отримати за допомогою SQL або іншої мови запитів, підтримуваною базою даних. Об'єктні моделі (ODM) часто повільніше, тому що вимагають переведення об'єктів в формат бази даних, при цьому не обов'язково будуть використані найбільш ефективні запити до бази даних (особливо, якщо ODM призначена для різних баз даних і повинна йти на великі компроміси в сенсі підтримки тих чи інших функцій бази даних).

Перевага застосування ORM полягає в тому, що програмісти можуть зосередитися на об'єктах JavaScript, а не на семантиці бази даних - особливо, якщо потрібно працювати з різними базами даних (на одному або різних веб-сайтах). Вони також дають очевидне місце для валідації і перевірки даних.

Оскільки застосування ODM / ORM часто призводить до зниження витрат на розробку та обслуговування, було обрано саме цей варіант.

Існує безліч популярних доступних ODM/ORM рішень, які можна використовувати.

Ось деякі з них:

- **Mongoose**

Це засіб моделювання об'єктів бази даних MongoDB, призначений для асинхронної роботи.

- **Waterline**

ORM фреймворку Sails (заснований на Express). Вона надає єдиний API для доступу до безлічі баз даних, в тому числі Redis, mySQL, LDAP, MongoDB, і Postgre.

- **Bookshelf**

Підтримує як promise- так і традиційні callback- інтерфейси, підтримка транзакцій, eager / nested-eager relation loading, поліморфні асоціації, і підтримка, «один до одного», «один до багатьох», і «багатьох до багатьох» зв’язків. Працює з PostgreSQL, MySQL, і SQLite3.

- **Objection**

Робить настільки легким, наскільки можливо, використання всієї потужності SQL і рушія бази даних (підтримує SQLite3, Postgres, і MySQL).

- **Sequelize**

Заснована на промісах ORM для Node.js і io.js. Підтримує діалекти PostgreSQL, MySQL, MariaDB, SQLite і MSSQL, має надійну підтримку транзакцій, відносин, читання копій і т.д.

- **Node ORM2**

Це OR менеджер для NodeJS. Підтримує MySQL, SQLite і Progress, допомагає працювати з БД, використовуючи об'єктний підхід.

- **JugglingDB**

Це крос-ДБ ORM для NodeJS, що забезпечує загальний інтерфейс для доступу до найбільш популярним форматам БД. Підтримує MySQL, SQLite3, Postgre, MongoDB, Redis і зберігання даних в пам'яті js (власний рушій, тільки для тестування).

Як правило, при виборі рішення слід враховувати як функції, що надаються, так і "діяльність спільноти" (завантаження, внесок, звіти про помилки, якість документації, і т.д.). На момент створення веб-додатку Mongoose була найпопулярнішою ORM, яка повністю виконувала наші вимоги, тому її разом із базою даних MongoDB і було обрано в якості сховища даних для нашого веб-додатку.

Mongoose є інтерфейсом для MongoDB, NoSQL-бази даних з відкритим вихідним кодом, в якій використана документо-орієнтована модель даних. У MongoDB «колекції» і «документи» - це аналоги «таблиць» і «рядків» в реляційних БД[10].

Це поєднання ODM і БД вельми популярне в співтоваристві Node, частково тому, що система зберігання документів і запитів дуже схожа на JSON і тому знайома розробникам JavaScript.

3.5 Структура серверної частини веб-додатку

Сервер в веб-додатку прослуховує запити, що надходять від клієнта. Під час налаштування HTTP-сервера він повинен прослуховувати конкретний номер порту. Номер порту завжди пов'язаний з IP-адресою комп'ютера[11].

Ви можете розглядати порти як окремі канали на кожному комп'ютері, які можна використовувати для виконання різних завдань: один порт може бути використаний для відвідування одного веб-ресурсу, в той час як через інший отримуєте електронну пошту. Це можливо, оскільки кожне з додатків (веб-браузер і клієнт електронної пошти) використовує різні номери портів.

Після налаштування HTTP-сервера для прослуховування певного порту сервер очікує клієнтські запитів, що надходять на цей порт, виконує всі дії, зазначені в запиті, і відправляє всі запитані дані через HTTP-відповідь.

Веб-додаток для моніторингу якості автодоріг є динамічним.

Динамічні веб-додатки, в свою чергу, мають змінювані сторінки, адаптуються під конкретного користувача. Такі сторінки не розміщені на сервері в готовому вигляді, а збираються заново по кожному новому запитом. Спочатку сервер знаходить потрібний документ і відправляє його інтерпретатора, який виконує код з HTML-документа і звіряється з файлами і базою даних. Після цього документ повертається на сервер і потім відображається в браузері.

На відміну від незмінних статичних сайтів, динамічні є більш гнучкі, що стосується управління. Складові сайту точно такі ж - мова розмітки, текст, графіка, однак сторінки цього типу можна створювати попутно з мінімальними витратами часу.

Також, варто відзначити, що більша частина коду для підтримки динамічного веб-додатку має виконуватися на сервері.

Схема нижче показує типову архітектуру динамічного веб-додатку. Браузер надсилають HTTP-запити на веб-сервер. Він, в свою чергу, обробляє їх і повертає потрібні відповіді HTTP[12].

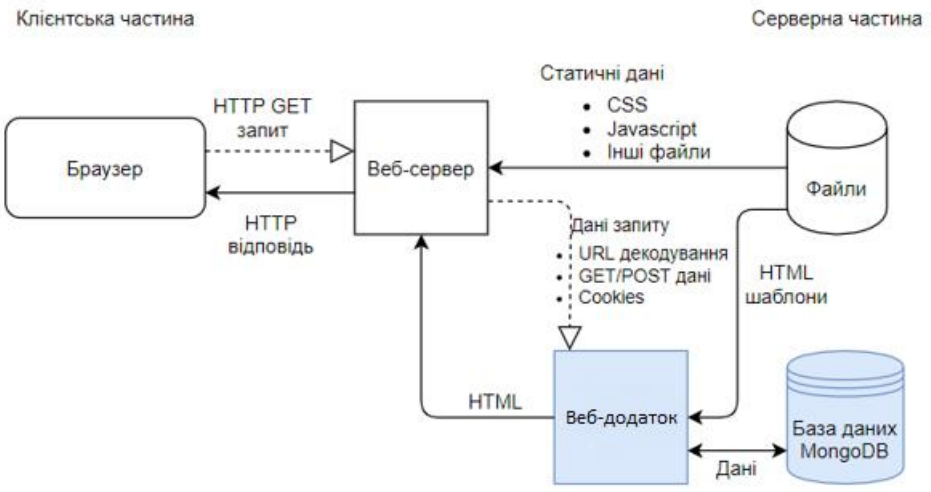

Рис. 3.5.1. Схема архітектури динамічного сайту

Запити з динамічними даними надсилаються до серверної частини. Для «динамічних запитів» сервер обробляє запит, отримує потрібні дані з БД, комбінує взяті дані з шаблонами HTML і надсилає відповідь, що має в собі згенеровану HTML-сторінку.

Програмування серверної частини додатку надає можливість розміщати дані в БД і динамічно створювати та повертати HTML та файли інших типів. Також є можливість просто повернути дані (JSON, XML, і т.д.) для відображення використовуючи відповідний фреймворк клієнтської частини (це зменшує завантаження процесора на сервері і кількість переданих даних.

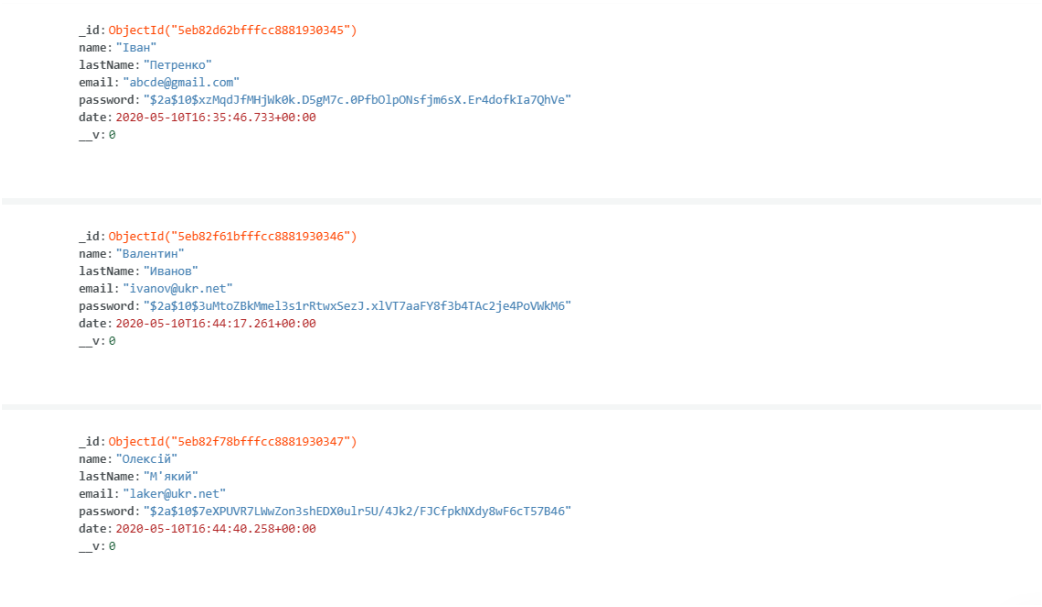

Рис. 3.5.2. Збережені дані користувачів у базі даних у вигляді документів

3.6 Структура Node/Express-додатку для моніторингу якості автодоріг

Наведена нижче діаграма зображує основний потоці даних і про елементи, які були реалізовані при обробці HTTP-запиту / відповіді.

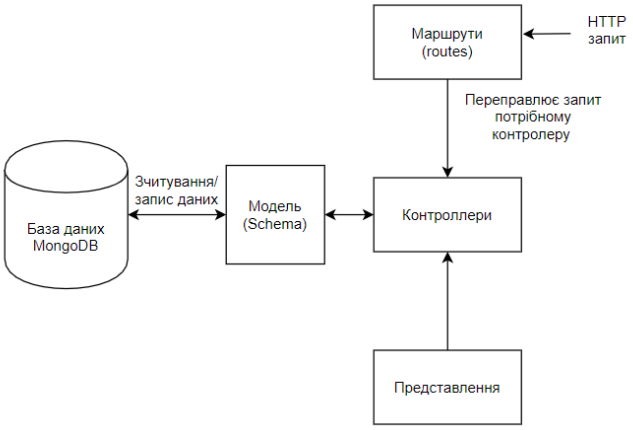

Рис. 3.6.1. Діаграма основного потоку даних у серверній частині додатку

Серверна частина додатку для контролю якості автодоріг складається з таких частин:

- **Головного файлу проекту index.js**

У файлі index.js міститься підключення основних фреймворків, імпортування маршрутів (routes), з’єднання з базою даних, визначення проміжних функцій для цих маршрутів та функцію прослуховування додатком обраного порту.

- **Моделей Mongoose**

Моделі були створені за допомогою інтерфейсу Schema. Schema дозволила вказати поля в кожному документі, значення полів за замовчуванням і вимоги валідації.

Схеми "компілюються" в остаточну модель методом mongoose.model (). Після створення моделі її можна використовувати для пошуку, створення, оновлення та видалення об'єктів даного типу[13].

- **User**

Містить поля, необхідні для реєстрації та авторизації користувачів: name (ім’я користувача), lastName (прізвище), email (електронна пошта), password (пароль) та date (дата реєстрації користувача).

- **Profile**

Містить поля із детальною інформацію для заповнення профіля користувача: photo (фотографія користувача), phone (телефон), та посилання на соціальні мережі.

- **Feedbacks**

Містить поля для складання відгуку: userId (унікальний ідентифікатор користувача, якому залишають відгук), from (ідентифікатор користувача, який залишає відгук), text (текст відгука), createdAt (дата та час залишання відгуку).

- **Ratings**

Містить поля критеріїв для складання рейтингу автодоріг: roadId (ідентифікатор дороги), ratedBy (ідентифікатор користувача, що оцінює дорогу), roadSurface (дорожне покриття), technicalMeans (технічні засоби), engineeringArrangement (інженерне облаштування), serviceObjects(об’єкти дорожнього сервісу), sanitaryElements(елементи санітарного облаштування), artificialConstructions(штучні споруди), mean(середнє значення оцінок по всім критеріям).

- **RoadProfile**

Містить поля з інформацією про автодороги: roadName (назва дороги), region (регіон місцезнаходження), direction (напрям), length (довжина).

- **Маршрути (routes)**

Маршрутизація визначає, як додаток відповідає на клієнтський запит до конкретної адреси (URL). Метод router() є похідним від одного з методів HTTP і приєднується до примірника класу express.

Express підтримує перераховані далі методи маршрутизації, які відповідають методам HTTP: get, post, put, head, delete, options, trace, copy, lock, mkcol, move, purge, propfind, proppatch, unlock, report, mkactivity, checkout, merge, m -search, notify, subscribe, unsubscribe, patch, search і connect[14].

Шляхи маршрутів, в поєднанні з методом запиту, визначають конкретні адреси (кінцеві точки), в яких можуть бути створені запити. Шляхи маршрутів можуть являти собою рядки, шаблони рядків або регулярні вирази.

Для обробки запиту можна вказати кілька функцій зворотного виклику, подібних middleware. Єдиним винятком є те, що ці зворотні виклики можуть ініціювати next ('route') для обходу інших зворотних викликів маршруту. За допомогою цього механізму можна включити в маршрут попередні умови, а потім передати управління подальшим маршрутами, якщо продовжувати роботу з поточним маршрутом не потрібно.

Маршрути, створені для додатку:

- **auth**

Маршрут призначений для аутентифікації користувача – реєстрації та подальшого входу.

Він містить методи:

- router.post()

Метод відповіді на запит POST зі шляхом «/register»: перевіряє дані на валідність, перевіряє, чи не існує користувача с такими даними, хешує пароль, введений користувачем. Після цього створює новий екземпляр схеми User і передає у відповідні поля дані, введені користувачем. Далі за допомогою методу save() дані зберігаються і додаються в колекцію Users бази даних.

- router.post()

Метод відповіді на запит POST зі шляхом «/login»: перевіряє введені дані на коректність, шукає, чи існує введена електронна пошта, звіряє

введений пароль з існуючими в базі. Якщо всі дані збігаються, створюється JWT (JSON Web Token), який засвідчує авторизацію.

- router.get()

Метод відповіді на запит GET зі шляхом «/all»: повертає дані всіх користувачів з колекції Users у форматі JSON.

- **profile**

Маршрут призначений для заповнення особистих даних користувачів.

Він містить методи:

- router.post()

Метод відповіді на запит POST зі шляхом «/profile»: створює новий екземпляр схеми Profile і додає у відповідні поля інформацію, введену користувачем. Після цього, за допомогою методу save(), дані зберігаються і додаються в колекцію Profiles бази даних.

- router.get()

Метод відповіді на запит GET зі шляхом «/profile»: перевіряє наданий UserId, шукає відповідного користувача у базі даних та у разі відповідності повертає дані користувача з колекції Profiles разом з полями з колекції Users.

router.get()

Метод відповіді на запит GET зі шляхом «/profile/all»: повертає всі дані з колекції Profiles у форматі JSON.

- **roadProf**

Маршрут призначений для додавання профілей автодоріг з основною інформацією про них, отримання профілей автодоріг з бази даних та пошуку автодоріг за назвою та регіоном.

Він містить методи:

- router.post()

Метод відповіді на запит POST зі шляхом «/road»: створює новий екземпляр схеми RoadProfile і додає у відповідні поля інформацію про автодорогу. Після цього, за допомогою методу save(), дані зберігаються і додаються в колекцію roadprofiles бази даних у форматі JSON.

- router.get()

Метод відповіді на запит GET зі шляхом «/road/all»: повертає всі профілі автодоріг з колекції roadprofiles.

- router.get()

Метод відповіді на запит GET зі шляхом «/searchroad»: метод, що реалізує пошук автодороги за назвою та регіоном за допомогою пошукової строки.

- **rating**

Маршрут призначений для обрахування рейтингу автодороги. Користувач оцінює автодорогу за такими критеріями як дорожнє покриття, технічні засоби (знаки, світлофори тощо), інженерне облаштування(освітлення, зважування, засоби примусового зниження швидкості руху), об’єкти дорожнього сервісу (стоянки, АЗС, готелі, мийки тощо), елементи санітарного облаштування (туалети, смітники тощо), штучні споруди (мости, шляхопроводи, тунелі тощо), після чого обраховується середня оцінка і присвоюється дорозі.

Він містить методи:

- mean**()**

Метод розрахунку середньої оцінки дороги.

- router.post()

Метод відповіді на запит POST зі шляхом «/rate»: створює новий екземпляр схеми Ratings, додає оцінки, введені користувачем, та середнє

значення рейтингу у відповідні поля та зберігає і додає документ у базу даних в колекцію ratings.

- **verifyToken**

Маршрут verifyToken містить функцію, яка перевіряє вебтокен на валідність.

3.7 Структура файлів та папок серверної частини проекту

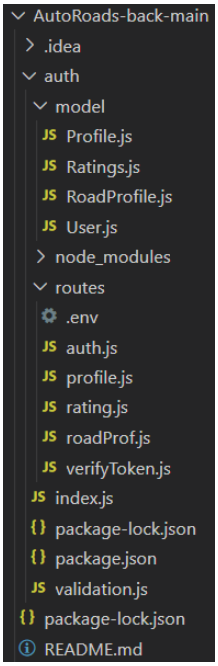

Рис. 3.7.1. Структура файлів і папок проекту

Коренева папка містить папку model, в якій містяться Mongoose схеми даних, routes, що містить маршрути (routes), файл index.js та інші службові папки та файли.

3.7 Висновки до розділу

Отже, після аналізу ряду мов програмування було обрано JavaScript як мову для написання серверної частини веб-додатку для контролю якості автодоріг. Відповідно, для реалізації серверних інструментів додатку було вирішено використовувати програмну платформу NodeJS. У якості бази даних було обрано нереляційну БД MongoDB.

У третьому розділі роботи було створено структуру та архітектуру веб-додатку для моніторингу якості доріг. Були розроблені моделі даних для додавання та приймання інформації про користувачів та автодороги з бази даних та відповідні контролери для обробки запитів користувача та виконання необхідних дій з даними.

РОЗДІЛ 4

АРХІТЕКТУРА ТА СТРУКТУРА КЛІЄНТСЬКОЇ ЧАСТИНИ ВЕБ-ДОДАТКУ

Клієнтська частина веб-додатку складається з таких частин: HTML – мова гіпертекстової розмітки, що відповідає за зміст веб-додатку, CSS – мова, що відповідає за відображення HTML-документу та JavaScript – мова програмування, що призначена для «оживлення» додатку. Також використовуються різноманітні фреймворки та бібліотеки, що призначені для скорочення необхідності виконання задач, що повторюються.

4.1 Обрання фреймворків та бібліотек JavaScript для створення інтерфейсу користувача

Для створення веб-додатку були обрані наступні фреймворки:

- **React**

React - це JavaScript бібліотека GUI (англ) з відкритим вихідним кодом, зосереджена на одній конкретній меті - ефективному виконання завдань в рамках розробки призначеного для користувача інтерфейсу. Його можна віднести до категорії "V" в архітектурному шаблоні MVC (модель-вид-контролер)[15].

Компонент React створити простіше, оскільки він використовує JSX (англ), опціональне розширення синтаксису JavaScript, яке дозволяє комбінувати HTML з JavaScript.

JSX - це відмінна суміш JavaScript і HTML (англ). Воно робить весь процес написання структури сайту зрозумілішим. Крім того, розширення також значно спрощує рендеринг декількох функцій[16].

Хоча JSX може бути не найпопулярнішим розширенням синтаксису, воно довело свою ефективність при розробці спеціальних компонентів або

додатків великого обсягу.

Високий відсоток перевикористання коду підвищує покриття тестами, що, в свою чергу, призводить до більш високого рівня контролю якості. Використовуючи React Native мобільні додатки для Android і iOS, використовуючи досвід JavaScript і React розробки.

Користь, яку від React може отримати користувач[17]:

- Virtual DOM може підвищити продуктивність високонавантажених додатків, що може знизити ймовірність виникнення можливих незручностей і покращує користувальницький досвід;
- Використання ізоморфного підходу допомагає робити рендеринг сторінок швидше, тим самим дозволяючи користувачам відчувати себе більш комфортно під час роботи з вашим додатком. Пошукові системи індексують такі сторінки краще. Оскільки один і той же код може бути використаний як в клієнтській, так і в серверній частині програми, немає необхідності в дублюванні одного і того ж функціоналу. В результаті час розробки і витрати знижуються;
- Завдяки повторному використанню коду стало набагато простіше створювати мобільні додатки. Код, який був написаний під час створення сайту, може бути знову використаний для створення мобільного застосунку. Якщо ви плануєте використовувати не тільки сайт, але і мобільний додаток, немає необхідності наймати дві великі команди розробників.

- **Redux**

Redux це інструмент для управління станом даних і інтерфейсом користувача в додатках JavaScript, передбачуване сховище стану додатків. Він ідеальний для односторінкових додатків, в яких управління станом згодом може стати складним[18].

Redux майстерно справляється зі складними взаємодіями станів, які важко передати за допомогою стану компонента React. По суті, це система передачі повідомлень, яка зустрічається і в об'єктно-орієнтованому програмуванні, але вона не вбудована безпосередньо в мову, а реалізована у вигляді бібліотеки. Подібно ООП, Redux переводить контроль від об'єкта, що викликає, до одержувача - інтерфейс не керує станом безпосередньо, а передає йому повідомлення для обробки.

Переваги, що дає Redux[19]:

Завдяки інверсії контролю зникає необхідність оновлювати призначений для користувача інтерфейс, як тільки змінюється імплементація зміни станів. Додавати такі складні функції, як логуваннння, скасування дії або навіть time-travel debugging, стає простіше простого. Інтеграційні тести зводяться лише до того, щоб перевірити, відправляється чи правильна дія, а для всього іншого досить юніт-тестів.

Стан компонентів в React занадто громіздкий для роботи з наскрізною функціональністю, яка зачіпає багато модулів програми, як, наприклад, інформація про користувача або сповіщення. Якраз для цього в Redux є дерево станів, незалежне від призначеного для користувача інтерфейсу. До того ж, при обробці стану поза інтерфейсом легше підтримувати сталість, адже серіалізація в localStorage або URL проводиться в єдиному місці.

Редюсери дають більшу свободу в роботі з діями, які можна поєднувати, відправляти одночасно і навіть обробляти в стилі method_missing.

- **Redux persist**

Redux-persist гарантує, що стан буде зберігатися між сеансами додатку. Він ініціалізує ваше сховище Redux з будь-яким раніше збереженим станом під час запуску програми і буде підтримувати цей постійний стан додатку в

синхронізації з поточним станом додатку. Без такого роду збереження стану ваш додаток буде скидати налаштування сховища за замовчуванням після кожного виходу з програми.

- **Redux Saga**

Redux-saga - це бібліотека, яка покликана спростити і покращити виконання сайд-ефектів (тобто таких дій, як асинхронні операції, типу завантаження даних і "брудних" дій, типу доступу до браузерних кешу) в React / Redux додатках.

Можна представити Saga як окремий потік в вашому додатку, який відповідає за сайд-ефекти. redux-saga - це redux middleware, що означає, що цей потік може запускатися, зупинятися і скасовуватися з основного додатка за допомогою звичайних redux екшенів, воно має доступ до повного стану redux додатку[20].

- **Styled-components**

Styled Components - це бібліотека для React і React Native для написання та управління CSS. Це рішення «CSS-in-JS», тобто ви пишете CSS в файлах Javascript (зокрема, в компонентах, які є файлами Javascript).

Переваги:

Це простий CSS. Ви пишете CSS в файлі JS, але синтаксис CSS не змінюється;

Вендорні префікси автоматично додаються при використанні стилізованих компонентів, підвищуючи продуктивність в браузерах;

Весь невикористаний CSS і стилі автоматично видаляються;

Ви взагалі не пишете ніяких імен класів. Імена класів генеруються автоматично, тому немає необхідності управляти методологією іменування класів CSS.

4.2 Структура клієнтської частини веб-додатку для контролю якості автодоріг

Клієнтська частина додатку складається з таких компонентів:

- **App**

Головний компонент додатку, що відповідає за рендеринг усіх компонентів програми.

- **Header**

Компонент, що відповідає за відображення «шапки» сайту. Він містить в собі навігаційну панель та логотип сайту.

- **AuthenticationPage**

Компонент, що відповідає за аутентифікацію користувача. Відображає поля для вводу електронної адреси та паролю.

- **RegistrationPage**

Компонент, який відповідає за реєстрацію користувача. Містить поля для вводу ім’я, електронної пошти та паролю.

- **MainPage**

Компонент, що відповідає за рендеринг головної сторінки. Містить в собі пошук та список автодоріг.

- **RoadCard**

Компонент, що містить в собі профіль дороги з основною інформацією про неї. Містить поля roadName (назва дороги), region (регіон), direction(напрям), length(довжина) та rating(рейтинг).

- **UserProfilePage**

Компонент, що відображає профіль користувача з персональною інформацією.

4.3 Висновки до розділу

У ході виконання четвертого розділу дипломної роботи було проаналізовано основні технології створення клієнтської частини веб-додатку, обрано необхідні бібліотеки та фреймворки.

Було спроектовано та розроблено компоненти клієнтської частини веб-додатку для контролю якості автодоріг, що забезпечують ресурс необхідним зручним та ефективним користувацьким функціоналом.

ВИСНОВКИ

У результаті виконання бакалаврської роботи було розроблено веб-додаток для контролю якості автодоріг.

Було проаналізовано предметну область, аналоги та сформульовано задачі розробки.

Порівнявши доступні сучасні методи розробки для створення серверної частини було вирішено використовувати такі технології як мову програмування JavaScript, програмну платформу NodeJS, ODM-бібліотеку Mongoose та базу даних MongoDB. Для розробки клієнтської частини додатку обрано такі технології: мову програмування JavaScript, фреймворк React, бібліотеки Redux та Styled-Components.

У серверній частині веб додатку для контролю якості автодоріг було розроблено функціонал для приймання та обробки запитів користувача, інструменти для обробки, збереження та отримання інформації про користувачів та автодороги з бази даних.

Для клієнтської частини додатку було створено зовнішній інтерфейс, реалізовано форми для реєстрації та авторизації користувачів, створено візуальне оформлення списку та профілей доріг з необхідною інформацією та профіля користувача.

Отже, у результаті виконання бакалаврської дипломної роботи, використовуючи практичні навички розробки програмного забезпечення, було створено функціонально корисний веб-додаток, що дозволяє користувачам контролювати роботу органів влади щодо забезпечення якості дорожньої інфраструктури.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Брэд Дейли, Брендан Дейли, Калеб Дейли. Разработка веб-приложений с помощью Node.js, MongoDB и Angular: исчерпывающее руководство по использованию стека MEAN: навч. пос. Диалектика-Вильямс, 2020. - 656 с.

2. Дронов Владимир. JavaScript и AJAX в Web-дизайне/ БХВ-Петербург - М., 2015. - 736 с.

3. Макфарланд Дэвид. JavaScript. Подробное руководство / Эксмо - М., 2015. - 608 с.

4. М. Кантелон, М. Хартер, Т. Головайчук, Н. Райлих. Node.js в действии. / - СПб.: Питер, 2014. – 548 с.

5. Сайт фреймворку Express [Електронний ресурс] / Режим доступу: https://expressjs.com/.

6. Сайт питань та відповідей програмістів Stack Overflow [Електронний ресурс] / Режим доступу: https://ru.stackoverflow.com

7. IT-блог Medium [Електронний ресурс] / Режим доступу: https://medium.com

8. Бэнкс Алекс, Порселло Ева. React и Redux: функциональная веб-разработка. /– СПб.: Питер, 2018. – 336 с.:

9. Хортон А., Вайс Р., Разработка веб-приложений в ReactJS: пер. с англ. Рагимова Р. Н. / М.: ДМК Пресс, 2016. – 254 с.:

10. Кайл Бэнкер. MongoDB в действии. / М.: ДМК Пресс, 2012. -394 с.:

11. Браун Итан. Веб-разработка с применением Node и Express. Полноценное использование стека JavaScript. / СПб.: Питер, 2017. – 336 с.:

12. Васильев, Алексей Николаевич. JavaScript в примерах и задачах/ Москва: Издательство «Э», 2017. -720 с.

13. Минник, Крис, Холланд, Ева. JavaScript для чайников. / Пер. с англ. – М. : ООО «И. Д. Вильямс», 2017. – 320 с.

14. Фримен Э., Робсон Э. Изучаем программирование на JavaScript/ СПб.: Питер, 2015 – 640 с.

15. Сухов К. К. Node.js. Путеводитель по технологии. / М.: ДМК Пресс, 2015. – 416 с.

16. Браун, Этан. Изучаем JavaScript: руководство по созданию современных веб-сайтов/ СпБ.: ООО «Альфа-книга», 2017. – 368 с.

17. Каскиаро М., Маммино Л. Шаблоны проектирования Node.js / М.: ДМК Пресс, 2017. – 396 с.

18. Резиг, Джон, Фергюсон, Расс, Пакстон, Джон. JavaScript для профессионалов / М.: ООО «И. Д. Вильямс», 2016. – 240 с.

19. Чиннатамби, Кирупа. Изучаем React / Москва: Эксмо, 2019. – 368 с.

20. Роббинс Дж. HTML5, CSS3 и JavaScript. Исчерпывающее руководство/М.: Эксмо, 2014. – 528 с.

ДОДАТКИ

Додаток А

Вихідний код серверної частини додатку.

А.1

Контролери обробки запитів для роботи з автодорогами.

```
const router = require('express').Router();
const RoadProfile = require('../model/RoadProfile');

router.post('/road', async (req, res) => {
    const roadProfile = new RoadProfile({
        roadName: req.body.roadName,
        region: req.body.region,
        direction: req.body.direction,
        length: req.body.length,

    });
    try {
        const savedRoadProfile = await roadProfile.save();
        res.send(savedRoadProfile);
    } catch (err) {
        res.status(400).send(err);
    }
});

router.get('/road/all', function (req, res) {
    RoadProfile.find({}, function (err, data) {
        if (err) {
            res.send("ERROR");
            next();
        }
        res.json(data);
    })
});
     const Ratings = require('../model/Ratings');

     router.get('/searchroad', (req, res, next) => {
         const searchedField = req.query.roadName;


         RoadProfile.aggregate([
             {
                 $match: {
                     roadName: {
                         $regex: searchedField,
                         $options: '$i'
                     }



                 }
             },
             {
                 $lookup: {
                     from: "ratings",
                     localField: "_id",
                     foreignField: "roadId",
                     as: "ratings"
                 },
             },
         ]).exec((err, data) => {
             if (err) {
                 console.log(err);
             }
             console.log(data);
             res.send(data);
         });
     });

     module.exports = router;
```

А.2
Схема даних з критеріями оцінювання автодоріг

```
const mongoose = require('mongoose');

const ratingsSchema = new mongoose.Schema({
  roadId: {
    type: mongoose.Types.ObjectId,
    required: true
  },
  ratedBy: {
    type: mongoose.Types.ObjectId,
    required: true
  },
  roadSurface: {
    type: Number,
    required: true,
    max: 10,
    min: 1,
  },
  technicalMeans: {
    type: Number,
    required: true,
    max: 10,
    min: 1,
  },
  engineeringArrangement: {
    type: Number,
    required: true,
    max: 10,
    min: 1,
  },
  serviceObjects: {
    type: Number,
    required: true,
    max: 10,
    min: 1,
  },
  sanitaryElements: {
    type: Number,
    required: true,
    max: 10,
    min: 1,
  },
  artificialConstructions: {
    type: Number,
    required: true,
    max: 10,
    min: 1,
  },
  mean: {
    type: Number,
    required: true,
    max: 10,
    min: 1,
  },
});

module.exports = mongoose.model('Ratings', ratingsSchema);
```

А.3
Контролери обробки запитів для реєстрації та авторизації користувачів

```
const router = require('express').Router();
const User = require('../model/User');
const jwt = require('jsonwebtoken');
const {registerValidation, loginValidation} = require('../validation');
const bcrypt = require('bcryptjs');
const TOKEN_SECRET = 'ihfhaksjfaskhfsajkfsa';


const cors = require('cors');
const express = require('express');
let app = express();
app.use(cors());
app.options('*', cors());

router.post('/register', async (req,res) => {



    const{error} = registerValidation(req.body);

    if(error) return res.status(400).send(error.details[0].message);


    const emailExist = await User.findOne({email: req.body.email});
    if(emailExist) return res.status(400).send('Email already exists!');


    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(req.body.password, salt);


    const user = new User({
        name: req.body.name,
        email: req.body.email,
        password: hashedPassword
    });
    try {
        const savedUser = await user.save();
        res.send({ userId: user._id });
    }catch (err) {
        res.status(400).send(err);
    }
});
```

```
router.post('/login', async (req, res) => {

    const{error} = loginValidation(req.body);
    if(error) return res.status(400).send(error.details[0].message);


      const user = await User.findOne({email: req.body.email});
      if(!user) return res.status(400).send('Эл. почта или пароль неверные!');


      const validPass = await bcrypt.compare(req.body.password, user.password);
      if(!validPass) return res.status(400).send('Эл. почта или пароль неверные!');


      const token = jwt.sign({
        email: user.email,
        name: user.name,
        userId: user._id
    }, 'secretkey', {expiresIn: '20s'});

    res.header('auth-token', token).send(user);
});

router.get('/all', function (req, res) {
    User.find({}, function (err, users) {
        if (err) {
            res.send("ERROR");
            next();
        }
        res.json(users);
    })
});




module.exports = router;
```

Додаток Б
Вихідний код клієнтської частини додатку
Б.1
Компонент головної сторінки веб-додатку

```
import React, { useEffect } from 'react';
import styled from 'styled-components';
import { Header, RoadCard } from '../components';
import { TextField } from '@material-ui/core';
import { useDispatch, useSelector } from 'react-redux';
import { searchRoads } from '../store/actions/creators';
import { roadsSelector } from '../store/selectors/roads';
import { useHistory } from 'react-router';
import { Routes } from '../constants';



const MainPage = () => {
  const history = useHistory();
  const dispatch = useDispatch();
  const roads = useSelector(roadsSelector);

  useEffect(() => {
    dispatch(searchRoads({ query: '' }));
  }, []);

  const handleSearch = ({ target }) => {
    dispatch(searchRoads({ query: target.value }));
  };

  const handleRoadClick = (id) => {
    history.push(`${Routes.ROAD_PROFILE}/${id}`);
  }

  return (
    <Container>
      <Header />
      <RoadsContainer>
      <TextField label="Search..." halfSize type="text" onChange={handleSearch} />
        {roads.map((road) => (
          <RoadCard {...road} key={road._id} onClick={handleRoadClick} />
        ))}
      </RoadsContainer>
    </Container>
  );
};

const Container = styled.div`

`;



const RoadsContainer = styled.div`
  width: 100%;
  display: flex;
  flex-direction: column;
  align-items: center;

`;
```

Б.2
Компонент App додатку

```
import { BrowserRouter, Route } from 'react-router-dom';
import { Routes } from './constants';
import { AuthenticationPage, MainPage, RegistrationPage, RoadProfilePage } from './pages';
import { PrivateRoute } from './components';
import { useSelector } from 'react-redux';
import { authenticatedSelector } from './store/selectors/user';
import UserProfilePage from './pages/UserProfilePage';

function App() {
  const authenticated = useSelector(authenticatedSelector);

  return (
   <BrowserRouter>
     <Route exact path={Routes.LOGIN} component={AuthenticationPage} />
     <Route exact path={Routes.REGISTRATION} component={RegistrationPage} />
     <PrivateRoute exact path={Routes.HOME} component={MainPage} authenticated={authenticated} />
     <PrivateRoute exact path={Routes.USER_PROFILE} component={UserProfilePage} authenticated={authenticated} />
     <PrivateRoute exact path={`${Routes.ROAD_PROFILE}/:id`} component={RoadProfilePage} authenticated={authenticated} />
   </BrowserRouter>
  );
}

export default App;
```