РОЗРОБКА ВЕБСАЙТУ З ВИКОРИСТАННЯМ REACT ТА NODE.JX

Самченко С.О.
ІТІНФ-20-1

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Самченку Станіславу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка вебсайту з використанням react та node.js

затверджена наказом університету від 20 травня 2024 року № 464 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 03 червня 2024 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, бібліотека для розробки клієнтської частини з відкритим кодом React, середовище виконання JavaScript коду Node.js.

4. Перелік питань, що потрібно опрацювати в роботі
1. Розробка клієнтської частини на бібліотеці React.
2. Стилізувати вебзастосунок препроцесором SCSS та бібліотекою mui.
3. Розробка серверної частини за допомогою середовища виконання коду Node.js.
4. Підключити нереляційну БД MongoDB для зберігання даних до вебзастосунку.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Демонстрація прикладів існуючих вебзастомунків для обміну інформації, схематичні зображення принципу взаємодії складових вебзастосуну, тестові зображення.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	08.04.2024	
2	Аналіз завдання, підбір літератури	09.04.24-17.04.24	
3	Аналіз літератури з досліджуваної проблеми	18.04.24-21.04.24	
4	Аналіз технічних засобів	22.04.24-23.04.24	
5	Розробка методу	24.04.24-16.05.24	
6	Програмна реалізація	17.05.24-25.05.24	
7	Оформлення пояснювальної записки	26.05.24-29.05.24	
8	Перевірка на плагіат	30.05.24	
9	Рецензування	31.05.24	
10	Підготовка презентації та доповіді	28.05.24-02.06.24	
11	Занесення роботи в електронний архів	03.06.24	
12	Попередній захист кваліфікаційної роботи	12.06.24	

Дата видачі завдання 8 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ ст. викл. Путятіна О.Є.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 60 с., 2 табл., 16 рис., 5 дод., 30 джерел.

КЛІЄНТ-СЕРВЕР, ВЕБЗАСТОСУНОК, РОЗРОБКА ПОВНОЦІННОГО САЙТУ, РОЗРОБКА ВЕББЛОГУ, FULLSTACK, FRONTEND REACTJS, BACKEND NODEJS.

Об’єктом роботи є вебзастосунок, призначений для розміщення інформації яка цікавить певних читачів.

Метою роботи є повноцінна розробка(клієнтської та серверної частини) власного веб блогу, що дозволить людям публікувати інформацію.

Було проведено ретельне дослідження усіх можливих прикладів веб блогів для обміну інформації між їх користувачами , на підставі чого було виявлено їх основні переваги та недоліки. Була побудована інформаційна модель веб блогу для обміну інформації.

У результаті роботи здійснена програмна реалізація вебзастосунку для обміну інформації з великою аудиторією.

CLIENT-SERVER, WEB APLICATION, DEVELOPMENT OF A FULL SITE, DEVELOPMENT OF A WEBLOG, FULLSTACK, FRONTEND REACTJS, BACKEND NODEJS.

The object of the work is a web site used for posting information to attract previous readers.

The goal of this work is the complete development (client and server parts) of a powerful web blog to allow people to publish information.

A thorough investigation of all possible applications of web blogs for the exchange of information between their users was carried out, from which their main advantages and shortcomings were revealed. An information model was created to create a web blog for information exchange.

As a result of the work, a program implementation of a web site for exchanging information with a large audience was created.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

JS – мова програмування JavaScript

CSS – Cascading Style Sheets

HTML – HyperText Markup Language

СУБД – система управління базою даних

SQL – Structured Query Language

БД – база даних

JSON – JavaScript Object Notation

API – Application Programming Interface

ВСТУП

React та Node.js – це два популярні інструменти для розробки вебсайтів та вебзастосунків. Сьогодні мільйони розробників використовують їх для створення та підтримки своїх сайтів, інтернет магазинів, соціальних мереж, криптобірж та багато іншого.

В даній роботі ці 2 інструменти будуть використовуватися для створення повноцінного блогу та всіх його сучасних можливостей. Блог – це вебсайт або онлайн-платформа, на якій автори (блогери) публікують свої записи або статті у хронологічному порядку. Ці записи, відомі як «пости», зазвичай відображаються у зворотному хронологічному порядку, з найновішими постами вгорі сторінки. Нині стало досить популярно ділитися своїми новинами у різних мережах. Даний проєкт розроблений приблизно за таким же принципом як і будь-яка сучасний вебблог, але тільки в менших масштабах.

Блоги можуть бути орієнтовані на різні теми та призначені для різних аудиторій. Наприклад, блоги можуть бути присвячені технічним новинам та оглядам, особистому життю, моді, кулінарії, подорожам, музиці, літературі та багато іншого. У блозі, розробленому в цьому проєкті, не буде конкретної тематики і люди зможуть ділитися з читачами будь-якою інформацією.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВА ЗАДАЧІ

1.1 Огляд архітектури клієнт-сервер

Архітектура клієнт-сервер є однією з найпоширеніших моделей веброзробки, що використовується для побудови сучасних вебзастосунків. Вона полягає в тому, що клієнти (наприклад, веббраузери або мобільні застосунки) взаємодіють з серверами (наприклад, вебсерверами) для отримання доступу до різних ресурсів та послуг.

На клієнтській стороні розташована вся інтерактивна частина застосунка, з якою користувач безпосередньо взаємодіє. Це включає в себе графічний інтерфейс користувача (UI), який відображається у веббраузері або мобільному застосунку, логіку застосунка, що керує взаємодією з користувачем та обробляє введення, а також локальні дані, такі як кешовані дані, налаштування та інше. Клієнтська сторона також включає в себе бібліотеки та фреймворки, які використовуються для розробки інтерфейсу, такі як React, Angular або Vue.js.

У цій роботі як раз буде задіяний один із них (React). React – це JavaScript бібліотека, розроблена Facebook для створення інтерфейсів користувача. Вона дозволяє розробникам створювати динамічні та інтерактивні вебзастосунки, поділяючи інтерфейс користувача на компоненти. React використовує Virtual DOM для ефективного оновлення інтерфейсу користувача та забезпечення швидкого відтворення змін.

Серверна сторона відповідає за обробку запитів від клієнтів, виконання бізнес-логіки застосунка та надання необхідних даних та ресурсів. Вона містить в собі бізнес-логіку, яка керує основними функціями застосунка, зберігання даних у базі даних або інших сховищах, API для взаємодії з клієнтами, а також механізми аутентифікації та авторизації, які забезпечують безпеку та конфіденційність. Серверна частина може бути реалізована

такими мовами програмування як наприклад: JavaScript (Node.js), Python (Django), Java (Spring Boot), Ruby (Ruby on Rails), PHP (Laravel), Go (Go Kit), C# (ASP.NET Core), Kotlin (Ktor), Swift (Vapor).

У цій роботі за основу розробки серверної частини буде відповідати мова програмування JavaScript та спеціально створений для цих задач фреймворк Node.js. Node.js – це середовище виконання JavaScript, побудоване на двигуні V8 від Google Chrome. Вона дозволяє розробникам створювати серверні програми на JavaScript. Node.js має потужні інструменти для роботи з мережею, файловою системою та базами даних, що робить його ідеальним вибором для створення вебсерверів та API.

Отже, веб застосунки складаються з інтерфейсу виконаного у вигляді певної сторінки в інтернеті та набору механізмів, які необхідні для реалізації логіки програми. На рисунку 1.1 зображена схема взаємодії між цими компонентами.

Рисунок 1.1 – Схема взаємодії складових вебзастосунку

Основні протоколи, які використовуються для комунікації між клієнтською та серверною сторонами, це HTTP (Hypertext Transfer Protocol) та WebSocket. HTTP використовується для передачі запитів та відповідей між клієнтом та сервером, тоді як WebSocket дозволяє зберігати постійне

з’єднання між ними для обміну даними в реальному часі, що дозволяє реалізувати функціонал чату, онлайн-ігор та інших реактивних застосунків.

Безпека є важливим аспектом архітектури клієнт-сервер. Шифрування використовується для захисту конфіденційності даних під час їх передачі через мережу, а також інші заходи безпеки, такі як обробка введення, захист від атак типу CSRF (міжсайтовий міждоменний скриптинг) та XSS (міжсайтовий скриптинг), а також управління доступом.

У цілому, архітектура клієнт-сервер надає гнучку та масштабовану модель для розробки вебзастосунків, яка дозволяє ефективно розділяти відповідальність між клієнтською та серверною сторонами застосунка.

1.2 Пояснення терміну вебзастосунок

1.2.1 Призначення вебзастосунків та принцип їх роботи

Вебзастосунок – це програмне забезпечення, яке працює у веббраузері та надає користувачам можливість взаємодіяти з ним через Інтернет. Вебзастосунок не потрібно завантажувати на персональний комп’ютер, оскільки він доступний онлайн у режимі реального часу. Користувачі можуть отримати доступ до вебзастосунку через будь-який існуючий браузер. Браузер – це спеціальне програмне забезпечення, що використовується для перегляду вебсторінок та контенту з цих сторінок у Інтернеті. Найвідомішими на сьогодні браузерами є Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, Opera та інші.

Головна мета будь-якого вебзастосунку полягає в створенні зручного, ефективного та інтуїтивно зрозумілого інтерфейсу для користувачів. Вони можуть бути призначені для різноманітних цілей: інформаційні сторінки, калькулятори для розрахунків, інтернет-магазини, онлайн біржі, соціальні мережі, та інші.

Для повноцінного функціонування будь-кого вебзастосунку обов’зково потрібні: сторінка, яка відображає характер та можливості застосунку, сервер та база даних. Як було зазначено в пункті 1.1, на клієнтській стороні скрипт відповідає за можливості застосунку. При запуску вебзастосунку у браузері завантажується скрипт, який відображає усі графічні елементи, текст та медіафайли з серверу, які повинні бути на сторінці. Коли користувач починає взаємодіяти з такими елементами як авторизація або завантаження файлів до застосунку ця інформація оброблюється у серверній частині і передається до необхідних баз даних.

База даних – це організована колекція даних, яка зберігається та управляється системою, що забезпечує доступ до даних у різний час та для різних користувачів. База даних може бути реляційною (MySQL, PostgreSQL, Oracle) або нереляційною (MongoDB, Cassandra, Redis). Реляційні бази даних організовані у вигляді таблиць з рядками та стовпцями, де дані взаємозв’язані за допомогою ключів. Нереляційні бази даних не використовують табличну структуру і можуть зберігати дані у вигляді документів, графів, або ключ-значення пар.

Вебзастосунки в основному мають короткі цикли розробки та невеликі розробницькі команди. Більшість вебзастосунків розробляються на JavaScript, HTML та CSS для клієнтської частини, та мовах, таких як Python, Java або Ruby для серверної частини. Адміністратори відповідають за управління контентом та співпрацюють з розробниками для визначення стратегії та шляхів розвитку вебзастосунку.

У зв’язку з передовими технологіями сьогодення, розробка різноманітних програм і інструментів для повноцінних вебзастосунків стала досить доступною. Проте, цей процес вимагає від учасників проєкту високого рівня кваліфікації, оскільки робота з багатофункціональними та складними завданнями потребує глибоких знань та досвіду. Таким чином, розробка великих вебзастосунків передбачає залучення висококваліфікованих фахівців з відповідним досвідом і навичками.

В наш час існує велика кількість вебзастосунків у різних сферах життя. Деякі з них включають соціальні мережі, наприклад, Facebook, Instagram, Twitter; електронні комерційні платформи, такі як Amazon, eBay, Alibaba; різноманітні сервіси з відео та мультимедійним контентом, наприклад, YouTube, Netflix, Spotify; а також інструменти для співпраці та роботи, такі як Google Docs, Microsoft Office Online, Trello. Крім цього, існують вебзастосунки для навчання, ведення особистого бюджету, подорожей, здоров'я та фітнесу, новин та багато інших. Вебзастосунки стали невід'ємною частиною нашого повсякденного життя та діяльності.

1.2.2 Чим відрізняється вебсайт від вебзастосунку

Вебсайт і вебзастосунок мають суттєві відмінності в своїй природі та призначенні.

Вебсайт зазвичай є колекцією вебсторінок, які можуть містити інформацію про певну компанію, організацію, продукт або послугу. Основна мета вебсайту полягає в тому, щоб надати користувачам інформацію чи розважальний контент, а також можливість зв'язку з власником сайту через контактні форми чи інші засоби комунікації. Вебсайти можуть бути статичними, коли їхній вміст не змінюється часто, або динамічними, коли вони використовують скрипти або бази даних для генерації контенту на льоту.

У той час як вебзастосунок – це програмне забезпечення, що працює у веббраузері і надає інтерактивний інтерфейс для користувача. Вебзастосунки можуть мати різноманітні функції та можуть бути спрямовані на вирішення різних завдань, від обробки даних і взаємодії з користувачем до забезпечення конкретних послуг або функціональності. Наприклад, онлайн-магазин, система керування відносинами з клієнтами (CRM), соціальна мережа або вебплатформа для навчання - це приклади вебзастосунків.

Отже, основна відмінність між вебсайтом і вебзастосунком полягає у їхньому призначенні та функціональності. Вебсайт зазвичай надає інформацію та контент, тоді як вебзастосунок надає інтерактивні можливості та функціональність для виконання конкретних завдань.

1.3 Огляд вебзастосунків, призначених для обміну інфориації

1.3.1 Пояснення терміну вебблог

Вебблог – це вебзастосунок, завдяки якому автор чи група авторів регулярно публікують записи стосовно свого життя або конкретної тематики, часто у хронологічному порядку. Ці записи можуть бути короткими думками, оглядами, рецензіями, новинами, особистими історіями чи будь-якою іншою інформацією, яка цікава аудиторії. Блоги можуть охоплювати різноманітні теми, такі як мода, подорожі, технології, кулінарія, мистецтво, політика, бізнес тощо.

Блоги надають можливість авторам взаємодіяти зі своєю аудиторією через коментарі та соціальні мережі, забезпечуючи двосторонню комунікацію. Крім того, вони можуть використовуватися для розвитку особистого бренду, просування бізнесу або надання експертної інформації у певній галузі.

Блоги можна класифікувати за різними критеріями, такими як тематика, формат подачі або цільова аудиторія. Наприклад, за тематикою можна виділити особисті блоги, де автор ділиться своїми думками, досвідом і щоденними подіями в стилі онлайн-щоденника, блоги про подорожі, які містять розповіді, поради та фотографії для планування подорожей, а також технічні блоги, присвячені оглядам гаджетів, навчальним матеріалам з програмування та новинам у сфері IT. Серед інших популярних тематик – кулінарні блоги з рецептами і порадами для приготування їжі, блоги про моду та красу з оглядами трендів у стилі та косметиці, політичні блоги з

аналізом політичних процесів і державної політики, блоги про фінанси та бізнес, що пропонують поради з особистих фінансів, стратегії розвитку бізнесу та інтерв'ю з підприємцями, а також блоги про мистецтво і дизайн, які висвітлюють тренди у мистецтві та поради для дизайнерів. Спортивні блоги зазвичай аналізують спортивні події, обговорюють матчі та дають поради з фітнесу і тренувань.

Формат подачі контенту у блогах також варіюється. Текстові блоги містять переважно текстові публікації з додаванням графічного контенту, фотоблоги зосереджені на фотографіях з мінімальним текстовим супроводом, а відеоблоги (влоги) надають інформацію у відеоформаті. Подкасти є аудіоблогами з розмовами або монологами, а мікроблоги пропонують короткі пости, на зразок тих, що публікуються у Twitter, де інформація подається лаконічно.

Крім того, блоги можуть бути орієнтовані на конкретну аудиторію. Блоги для професіоналів приваблюють людей певної професії і пропонують спеціалізований контент. Освітні блоги призначені для навчання та надання інструкцій, а розважальні створені для розваг, зосереджуючись на курйозах, жартах та новинах у сфері шоу-бізнесу.

1.3.2 Приклади існуючих вебблогів

Найпростішим та найпопулярнішим прикладом персонального блогу є досить відома площадка відеохостингу YouTube від компанії Google (рис. 1.2). Сьогодні кожна людина може створити свій власний блог (канал) на цій платформі та ділитися із аудиторією будь-якою інформацією (у рамках правил YouTube) шляхом публікації відео, звичайних текстових постів та проведенням прямих трансляцій (стрими).

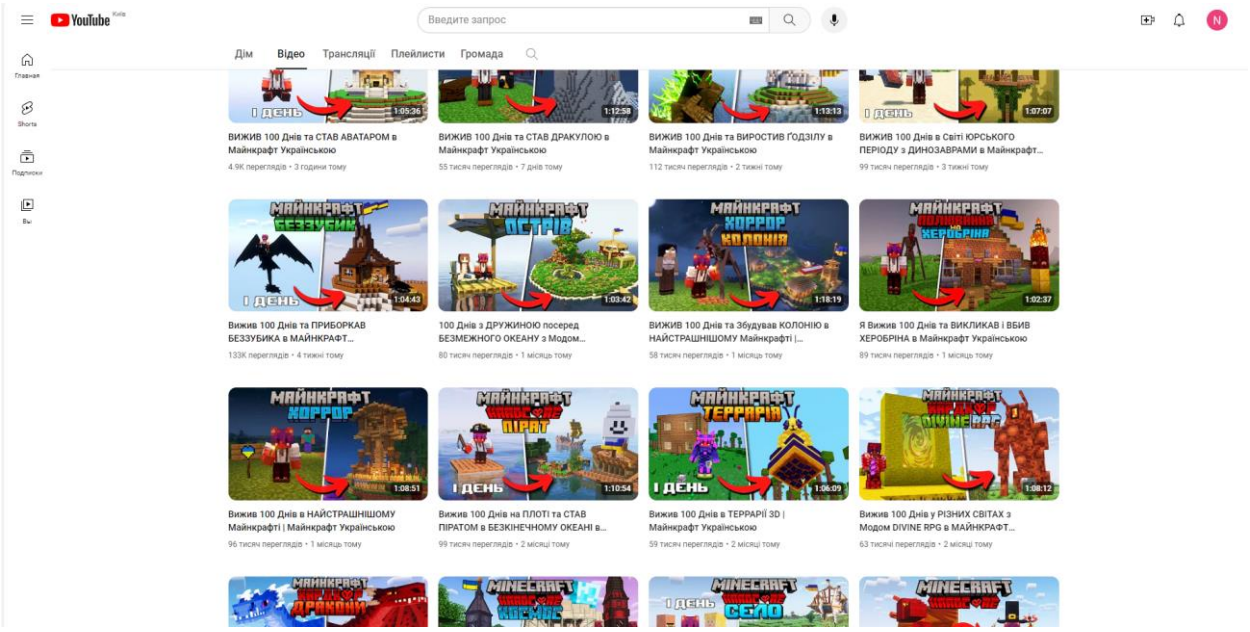

Рисунок 1.2 – Представлення вигляду примітивного каналу на платформі YouTube

Схожим на YouTube, але більш вузькоспрямованим на прямі ефіри, є стримінговий сервіс під назвою Twitch (рис. 1.3). Так само як і на YouTube, будь-яка людина може створити власний блог (канал) та почати публікувати інформацією для інших користувачів площадки. На цій платформі блогери в основному діляться своїм контентом та новинами з аудиторією через прямі трансляції. Такий формат блогу є головною задумкою та особливістю цього веб застосунку. І хоча існує багато подібних веб ресурсів (у тому ж числі і YouTube), переважна більшість блогерів та представників аудиторії, яка зацікавлена саме у такому форматі подачі та сприйняття інформації, віддають перевагу саме Twitch, через зручність платформи та велику кількість способів інтерактивної взаємодії з аудиторією.

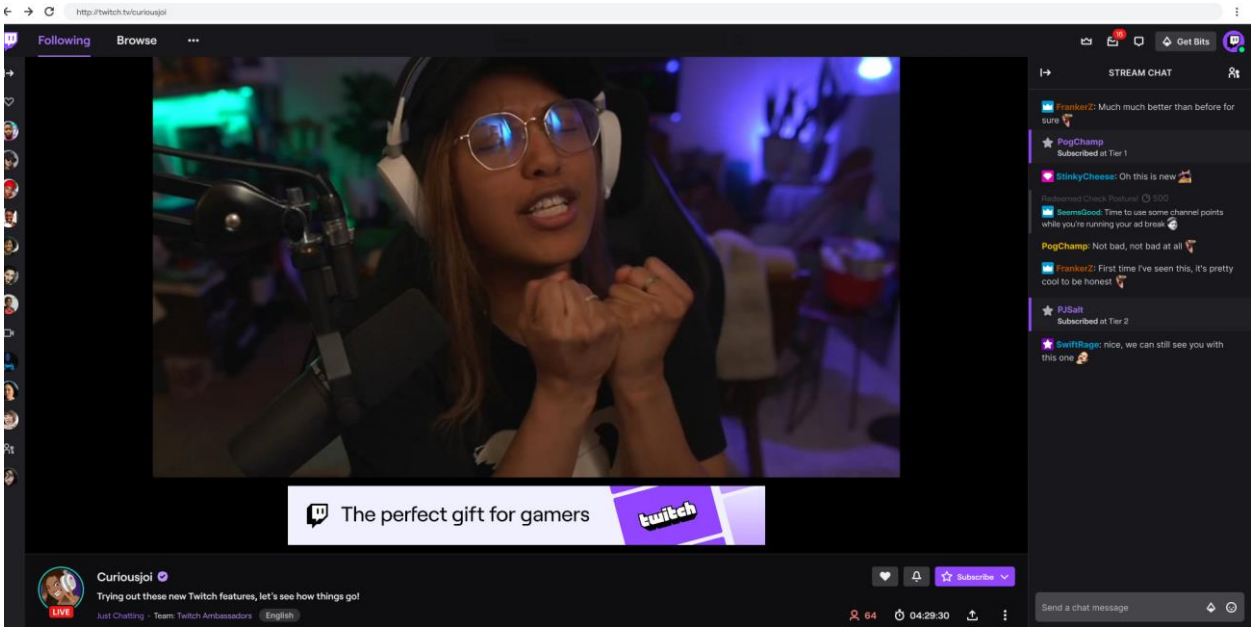

Рисунок 1.3 – Наглядний приклад того, як влаштований обмін контентом з аудиторією на платформі Twitch

Окрім YouTube та Twitch існує безліч відомих подібних веб застосунків, які надають своїм юзерам можливість ділитися корисною або розважальною інформацією з іншими. Така можливість може бути як основною ідеєю та функцією платформи (як наприклад YouTube, Twitch, TikTok, Instagram) так і додатковою можливістю соцмережі або месенджеру (як наприклад канал новин у Telegram або група, присвячена певній тематиці у Facebook).

Але не всіх людей влаштовують принципи, алгоритми, правила, функціонал або просто зовнішній вигляд популярних веб блогів. Через це деякі блогери створюють власні подібні вебзастосунки, за своїми перевагами та смаками, на яких вони можуть розміщувати будь який контент у рамках закону з повним налаштуванням зовнішнього вигляду, алгоритмів публікації та обробки їх персонального вебблогу. На рисунку 1.4 приведено приклад одного із таких вебзастосунків.

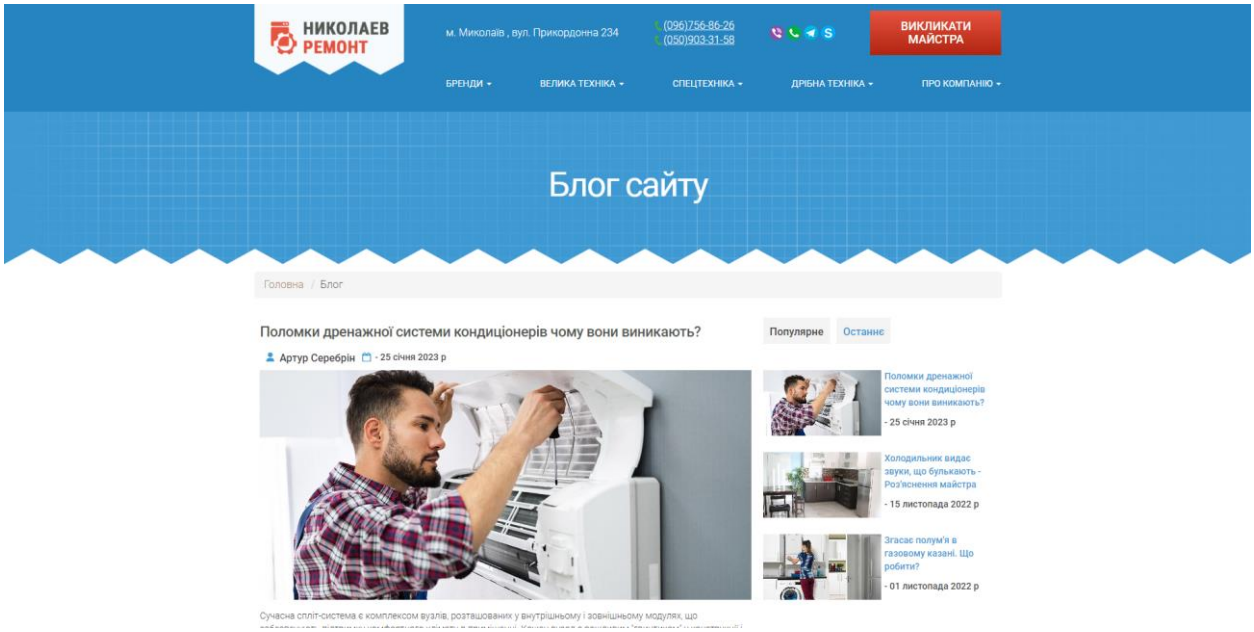

Рисунок 1.4 – Приклад власно-розробленого вебблогу

Цей приклад є досить детальною демонстрацію того, як автор може налаштувати кожний елемент свого веб застосунку. Тематикою даного блогу є ремонт несправної побутової техніки. Окрім того, що автор розповідає своїм читачам про свій досвід роботи, чому відбуваються випадки поломки, як можна самому вирішити подібну проблему та як такої проблеми запобігти – він також надає людям свої послуги у питанні відновлення зламаних пристроїв. Окрім цього помітно, що блог має унікальний дизайн, зручну навігацію, контактну інформацію та незвичайну сітку зі статтями, присвяченій головній тематиці блогу. У сукупності такі елементи сприяють гарному враженню, бажанню ознайомитися з інформацією та викликають довіру відвідувачів ресурсу, що безпосередньо є гарним результатом.

1.3.3 Основні характеристикии вебблогу

Виходячи з даних, отриманих при огляді та аналізі існуючих вебблогів до їх основних характеристик можна віднести такі аспекти як:

– тематика, яка цікавить певну групу людей;

– має набір публікацій, які зазвичай включають статті, пости, рецензії, інструкції або інші типи контенту;

– може бути ведений однією особою або групою авторів;

– вебблоги часто мають можливість коментування, що дозволяє читачам висловлювати свої думки та відгуки щодо публікацій;

– блог можна вести як на готовому ресурсі(але погоджуватися із правилами, які встановлені власниками) так і розробити власний з нуля(де правила також встановлюються нами і регулюються тільки законом);

– дизайн блогу може бути різноманітним і відображати його тематику, стиль та індивідуальність автора.

1.4 Постановка задачі

Таким чином, розробка власного веб блогу є актуальною задачею, тому після проведення аналізу предметної області, необхідно поставити конкретні завдання для розробки вебзастосунку, який надає можливість ділитися з читачами певною інформацією.

Об’єктом роботи є вебзастосунок, призначений для розміщення інформації яка цікавить певних читачів.

Метою роботи є повноцінна розробка (клієнтської та серверної частини) власного вебблогу, що дозволить людям публікувати інформацію.

Для досягнення мети необхідно реалізувати наступні завдання:

– ознайомитися з існуючими на сьогодні веб блогами та з принципом їх роботи;

– розробити концепцію створення вебзастосунку;

– опанувати необхідні для розробки технології;

– створити логіку здійснення публікацій та реалізувати зворотній зв’язку;

- розробити клієнтську частину застосунку;
- створити серверну частину;
- реалізувати зв’язок застосунку із БД.

2 МОДЕЛЮВАННЯ ВЕБЗАСТОСУНКУ

2.1 Огляд складових частин для розробки вебзастосунку

2.1.1 Моделювання клієнтської частини вебблогу

Фронтенд, або клієнтська частина вебзастосунка, є тим, що користувач бачить та з чим він активно взаємодіє через вебзастосунок. Основні складові структури фронтенду включають в себе HTML, CSS та JavaScript (рис. 2.1).

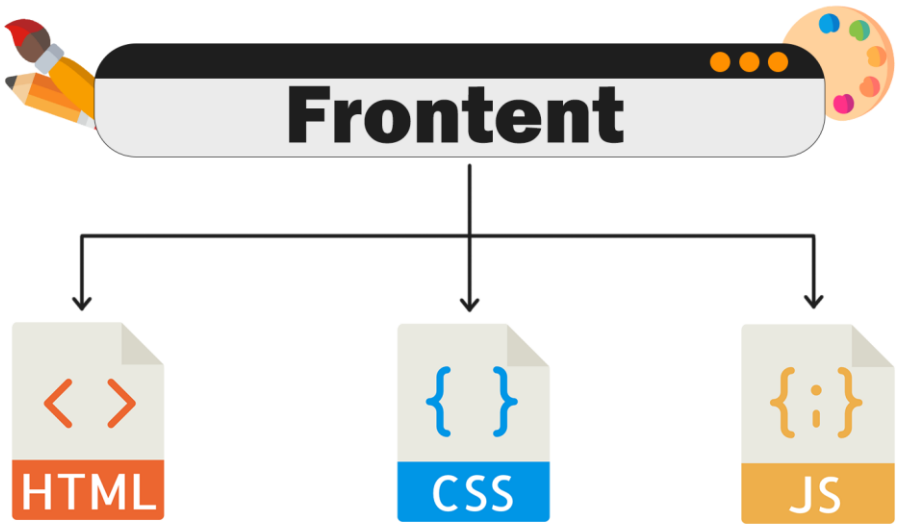

Рисунок 2.1 – Структурні складові фронтенд розробки

Проте, у сучасній веброзробці, дуже рідко можна зустріти сайт, який був створений саме на чистих HTML, CSS та JavaScript. Сьогодні, верстальники віддають перевагу більш сучасним та зручним інструментам для цих задач, які називаються фреймворки та бібліотеки. Вони можуть відрізнятися синтаксисом, структурою, логікою, побудовою, але ціль у них одна – спростити роботу розробникам. Найвідомішими фреймворками та

бібліотеками у сфері фронтент є React, Vue та Angular. У цій роботі буде використовуватися та описуватися React.

React – це потужна бібліотека JavaScript, яка дозволяє розробляти вебзастосунки з високою швидкодією та динамічними інтерфейсами. Основними концепціями React є компонентний підхід, віртуальний DOM та односторонній потік даних (рис. 2.2).

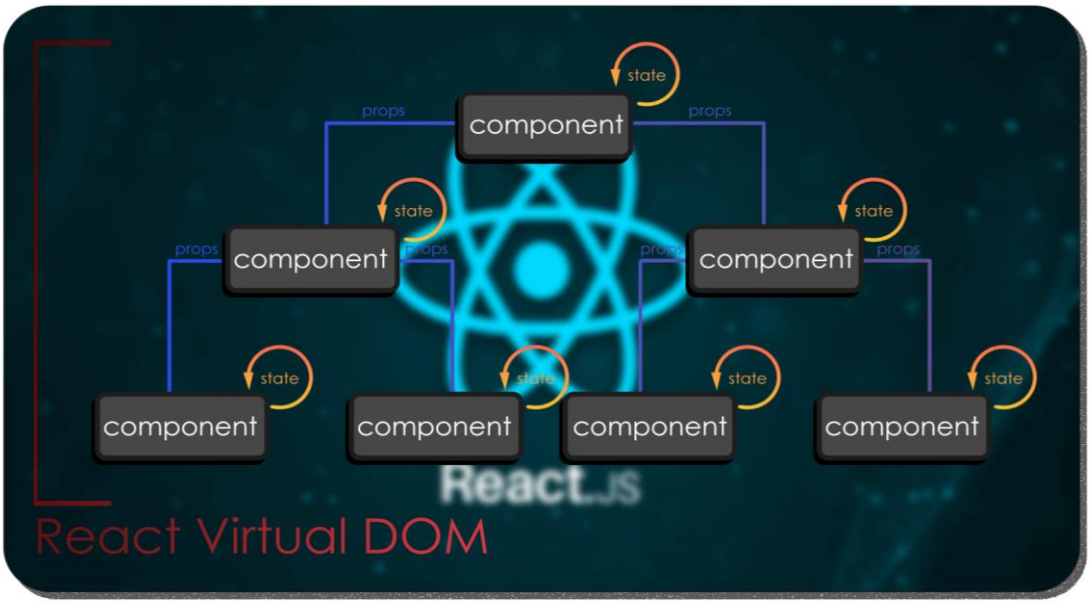

Рисунок 2.2 – Логіка React застосунку

У React, все обертається навколо компонентів. Компоненти – це незалежні, перевикористовувані блоки, які складаються з HTML-подібної розмітки та логіки JavaScript. Кожен компонент може приймати вхідні дані (props) і повертати елементи візуалізації, які React потім рендерить в браузер.

У відмінності від традиційного DOM-моделі, React використовує віртуальний DOM, який є копією реального DOM. При зміні стану компонентів, React вносить зміни в віртуальний DOM, а потім порівнює його з реальним DOM та виконує мінімальну кількість маніпуляцій для оновлення сторінки. Це робить процес рендерингу ефективнішим та швидшим.

Структура React додатку зазвичай складається з кореневого компонента, який рендериться в DOM, та його дочірніх компонентів. Компоненти можуть вкладатися один в одного, утворюючи дерево компонентів. Кожен компонент може мати власний стан та властивості, які визначають його поведінку та вигляд.

Поняття стану (state) в React використовується для зберігання даних, які можуть змінюватися в процесі роботи додатку. Стан може бути оновлений зовні (за допомогою подій) або внутрішньо (відповідно до логіки компоненту). Властивості (props) використовуються для передачі даних від батьківського компонента до дочірніх.

Крім того, React надає можливість використовувати хуки (hooks), які дозволяють використовувати стан та інші функціональності React у функціональних компонентах. Це робить React більш гнучким та потужним інструментом для розробки вебзастосунків.

У таблиці 2.1 приведено декілька вагомих критерій для порівняння React зі звичайною веброзробкою.

Таблиця 2.1 – Порівняння React з простою верстkою

Критерій	**React**	**Звичайна верстка**
1	**2**	**3**
Продуктивність	Використання віртуального DOM	Відсутність віртуального DOM
Масштабованість	Легке масштабування	Складніше масштабування
Підтримка	Велика та активна спільнота	Менш активна спільнота
Функціональність	Компонентна архітектура, JSX, управління станом та пропсами	Відсутність таких можливостей

Продовження таблиці 2.1

1	2	3
Швидкість розробки	Швидше розробка завдяки компонентній архітектурі та інструментам для розробки	Повільніша розробка
Спільнота	Активна спільнота розробників, яка забезпечує підтримку та розвиток	Менш активна спільнота, менше ресурсів для підтримки
Навчання	Велика кількість документації, туторіалів та онлайн-курсів	Менш доступних ресурсів для навчання, менше інструкцій
Продуктивність	Швидкіше виконання завдань завдяки оптимізації та ефективному управлінню станом	Повільніше виконання завдань, менша оптимізація
Гнучкість	Більша гнучкість у побудові складних інтерфейсів та компонентів	Обмежена гнучкість у побудові складних інтерфейсів
Тестування	Зручніше тестування компонентів та стану за допомогою інструментів, таких як Jest або Enzyme	Обмежені можливості тестування, менше інструментів

Але за допомогою React створюється лише оболонку сайту, щоб стилізувати сторінку та примінити до неї власний дизайн використовується CSS. Проте для CSS також існують свої вдосконалення, наприклад SCSS, який також використовується у даній роботі.

SCSS – це препроцесор CSS, який розширює функціонал та можливості до звичайного CSS. Основна перевага SCSS полягає в його розширеній синтаксичній можливості, яка включає змінні, вкладені правила, міксини, спадковість та інші функції, що дозволяють писати CSS більш організовано і зручно.

Змінні дозволяють зберігати значення, такі як кольори чи розміри, в одному місці і використовувати їх потім в усій таблиці стилів, що робить код більш читабельним і легко змінюваним. Вкладені правила дозволяють вкладати одні стилі всередині інших, що полегшує організацію та структуру коду. Міксини дозволяють створювати набір стилів, які можна використовувати повторно для різних елементів.

Крім цього, SCSS підтримує оператори, функції, умовні конструкції та інші функціональні можливості, які розширюють можливості CSS і полегшують розробку та обслуговування стилів. Після написання коду на SCSS, його можна компілювати в звичайний CSS, який використовується для відображення на вебсторінці. У таблиці 2.2 приведено декілька вагомих критерій для порівняння SCSS та CSS.

Таблиця 2.2 – Порівняння SCSS та CSS

Критерій	**CSS**	**SCSS**
1	**2**	**3**
Синтаксис	Звичайний CSS	Розширений синтаксис SCSS
Змінні	Немає підтримки змінних	Підтримка змінних
Вкладеність	Обмежена можливість вкладеності	Безліч вкладених правил

Продовження таблиці 2.2

1	2	3
Міксини	Відсутність можливості використання міксинів	Можливість використовувати міксини
Імпорт	Обмежена підтримка імпорту CSS файлів	Повна підтримка імпорту
Наслідування	Обмежена підтримка наслідування	Повна підтримка наслідування
Підтримка	Підтримується всіма браузерами	Підтримується всіма браузерами
Розширення	Можливість використовувати розширені функції та селектори	Підтримка розширених функцій та селекторів

Також для стилізації використовуються шаблони з бібліотеки Material-UI. Material-UI (або MUI) – це бібліотека компонентів інтерфейсу користувача для React, яка базується на дизайні Google Material Design. Вона надає набір готових до використання компонентів і стилів, що допомагає розробникам швидко створювати стильні та сучасні вебзастосунки.

Material-UI пропонує широкий вибір різноманітних компонентів, таких як кнопки, форми, таблиці, модальні вікна, навігаційні панелі та багато інших. Всі вони мають чистий та консистентний дизайн, який дотримується принципів Material Design.

Однією з переваг Material-UI є його простота використання. Завдяки прописаній документації та демонстраційним прикладам, розробники можуть швидко оволодіти бібліотекою та почати створювати додатки. Крім того,

Material-UI підтримує адаптивний дизайн, що дозволяє створювати додатки, які працюють на різних пристроях і розмірах екрану.

Бібліотека також надає можливість налаштування стилів компонентів за допомогою тем і стилів. Це дозволяє розробникам легко змінювати вигляд та поведінку компонентів, відповідно до потреб їх додатка.

Узагальнюючи, Material-UI є потужною та зручною бібліотекою компонентів для React, яка допомагає розробникам створювати красиві та функціональні вебзастосунки швидко та ефективно.

Таким чином, для розробки структури вебзастосунку буде використовуватися бібліотека React: вибір обумовлений її широким функціональним спектром, що спрощує розробку та підтримку клієнтської частини програми. Для стилизації застосунку буде використовуватися SCSS, тому що цей препроцесор пришвидшує та значно спрощує процес стилізації вебсайту та бібліотека Material-UI, в якій є всі необхідні шаблони для даного проєкту.

2.1.2 Моделювання серверної частини вебблогу

Бекенд – це та частина програмного забезпечення, яка забезпечує обробку даних та виконання логіки на сервері. В контексті веброзробки, бекенд відповідає за всі серверні операції, що стосуються обробки запитів, зберігання та управління даними, а також взаємодію з іншими системами (рис. 2.3).

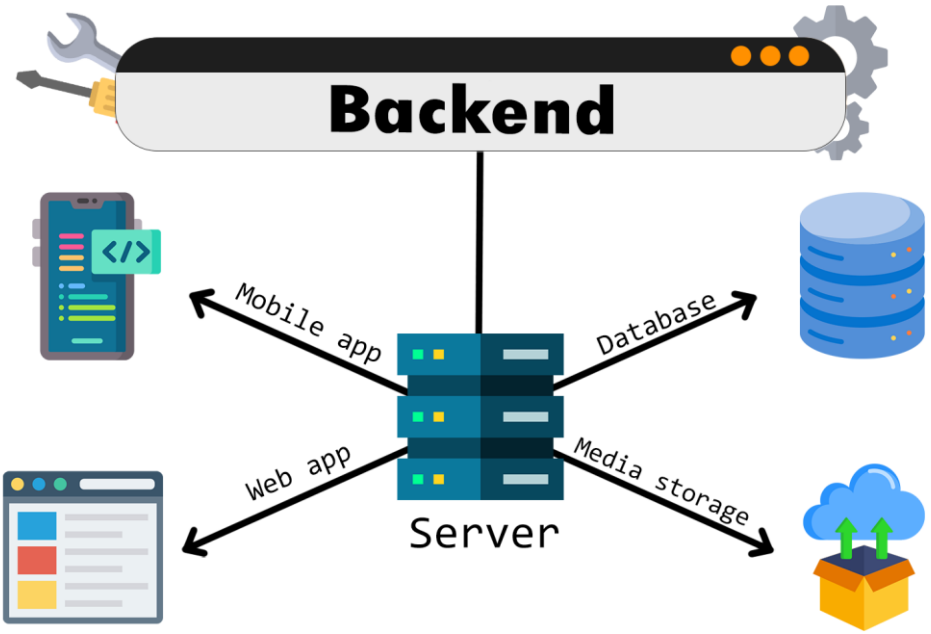

Рисунок 2.3 – Візуальне представлення принципу роботи бекенду

Основні функції бекенду включають управління базами даних, реалізацію бізнес-логіки додатку, обробку та перевірку даних, аутентифікацію та авторизацію користувачів, обробку запитів від клієнтів та відправлення їм відповідей. Крім того, бекенд може включати в себе такі компоненти, як кешування, розподілений обчислювальний потік, обробку файлів, роботу з мережевими службами та інші.

Технології, що використовуються для створення бекенду, можуть включати мови програмування, такі як JavaScript (за допомогою Node.js), Python, Ruby, Java, PHP, а також фреймворки та інструменти для розробки вебзастосунків, такі як Express.js для Node.js, Django для Python, Ruby on Rails для Ruby, Spring для Java та Laravel для PHP. Бекенд інтегрується з фронтендом (клієнтською частиною) для створення повноцінного вебзастосунку, що надає користувачеві доступ до всіх необхідних функціональних можливостей.

В контексті розробки проєкту було обрано мову програмування, яка в змозі забезпечити реалізацію поставлених питань, а саме – Node.js.

Node.js є платформою, яка дозволяє виконувати JavaScript на серверному боці. Це означає, що розробники можуть створювати вебзастосунки, які працюють на сервері, використовуючи JavaScript. Однією з ключових особливостей Node.js є можливість використання JavaScript для програмування як на клієнтській, так і на серверній стороні, що спрощує розробку та збільшує переносимість коду між фронтендом і бекендом.

Node.js використовує асинхронний та подійно-орієнтований підхід до програмування. Це означає, що код Node.js виконується неблокуючим способом, що дозволяє обробляти багато запитів та подій одночасно. Це робить його особливо ефективним для створення високонавантажених вебзастосунків, таких як вебсервери та API.

Однією з ключових переваг Node.js є його ефективність та масштабованість. Вбудована підтримка механізмів масштабування, таких як кластеризація та багатопоточність, дозволяє легко розгортати та масштабувати додатки для обробки великого обсягу запитів.

Крім того, Node.js має широкий вибір бібліотек та фреймворків, які полегшують розробку вебзастосунків. Наприклад, Express.js - це популярний фреймворк для створення вебсерверів та API з використанням Node.js.

Загалом, Node.js є потужною та універсальною платформою для створення різноманітних вебзастосунків, яка дозволяє розробникам швидко та ефективно створювати сучасні та масштабовані додатки за допомогою JavaScript.

2.2 Огляд та моделювання бази даних

2.2.1 MongoDB як система управління БД

Для розробки бази даних вебзастосунку було обрано MongoDB – це документ-орієнтована база даних, яка використовує JSON-подібні документи для зберігання даних. Основна особливість MongoDB полягає в тому, що вона дозволяє зберігати дані у вигляді документів, які можуть бути

складними структурами даних, а не просто таблицями зі стовпцями, як у традиційних реляційних базах даних.

У MongoDB дані зберігаються у вигляді колекцій, які містять документи. Кожен документ може мати різну структуру, але зазвичай вони містять ключі та значення у форматі JSON. Це дозволяє зберігати дані більш гнучко і при цьому забезпечує швидкий доступ до них.

Однією з переваг MongoDB є гнучкість схеми даних. У відміну від реляційних баз даних, де схема даних повинна бути заздалегідь визначена, в MongoDB можна зберігати документи з різними структурами у тій же колекції. Це робить її ідеальним вибором для проектів, де структура даних може змінюватися з часом.

Ще однією перевагою MongoDB є горизонтальне масштабування. MongoDB дозволяє розподіляти дані по кількох серверах (кластеризація), щоб забезпечити підтримку великих обсягів даних та високу доступність системи.

В цілому, MongoDB є потужною та гнучкою базою даних, яка підходить для різноманітних проектів, особливо тих, де важливо швидко реагувати на зміни в структурі даних та масштабувати систему з ростом обсягів інформації.

У базі будуть зберігатися конфіденційні дані користувачів. І хоч MongoDB – це надійний вибір, обов’язково потрібне додаткове страхування у вигляді шифрування. Для цього ідеально підходить JWT (JSON Web Token).

JWT – це відкритий стандарт для безпечної передачі інформації між сторонами у вигляді JSON-об’єкта. JWT зазвичай використовуються для аутентифікації та авторизації в вебзастосунках.

JWT складається з трьох частин: заголовка, корисного навантаження і підпису. Заголовок містить тип токена і алгоритм підпису. Корисне навантаження включає заяви (claims) про користувача або інші метадані. Підпис використовується для перевірки цілісності токена.

Процес створення JWT починається з формування заголовка і корисного навантаження, які кодуються в Base64Url. Потім створюється підпис з використанням алгоритму, визначеного в заголовку. Підпис додається до закодованих заголовка і корисного навантаження.

Переваги JWT включають самодостатність (вся інформація зберігається в токені), легкість передачі через URL, POST-параметри або заголовки HTTP, і підвищену швидкість обробки запитів. Типове використання JWT виглядає так: користувач вводить облікові дані, сервер генерує JWT і відправляє його клієнту, який використовує токен для доступу до захищених ресурсів.

Недоліки JWT включають необхідність механізмів відкликання токенів або коротких термінів дії для відображення змін у користувацьких даних.

Загалом, JWT є ефективним інструментом для аутентифікації та авторизації в сучасних веб-застосунках.

2.2.2 Моделювання БД вебзастосунку

У застосунку передбачається збереження нових користувачів для авторизації та звірка зареєстрованих для авторизації, тому виникає необхідність у відповідній для цього таблиці, яка називається «Users».

Для реєстрації та авторизаціє користувачу необхідно надати наступні данні:

– fullName: це поле призначене для створення імені кожного користувача. Людина може внести як своє справжнє ім’я, так і використати псевдонім, яким наіменовує себе у інтернеті. Дане поле є строковим(string) та має ключ «required: true», що робить поле обов’язковим для заповнення;

– email: поле, яке відповідно використовується для вводу електронної адреси користувача. Також являє собою строковий тип даних та є обов’язковим для заповнення. Окрім цього має флаг «unique: true», що

означає, що поле повинно бути унікальним, так як два та більше юзера не можуть мати однакову пошту;

– passwordHash: поле, яке зберігає згенерований користувачем пароль від його аканту. Також являє собою строковий тип даних та є обов’язковим для заповнення;

– avatarUrl: так як об’єктом даної роботи виступає вебблог, у користувачів має бути можливість встановити зображення для свого профілю. Саме для реалізації цієї можливості існує дане поле. Незважаючи на своє призначення, воно є строковим, і зберігає наіменування файлу, розташованого на бекенді. Воно також є необов’язковим, так як не кожна людина може захотіти встановити собі аватар;

– id: поле ідентифікатора є автоматичним, та створюється самою БД. Воно є унікальним, строковим та ключовим (PRIMARY KEY);

Таким чином, у таблиці «Users», надана користувачами інформація буде зберігатися у полях fullName, email, passwordHash, avatarUrl та id.

Скрипт відношення представлений у лістингу 2.1.

Лістинг 2.1 Реалізація таблиці «Users»:

```
const UserSchema = new mongoose.Schema(
 {
  fullName: {
   type: String,
   required: true,
  },
  email: {
   type: String,
   required: true,
   unique: true,
  },
```

```
    passwordHash: {
      type: String,
      required: true,
    },
    avatarUrl: String,
  },
  {
    timestamps: true,
  },
);
export default mongoose.model('User', UserSchema);
```

Оскільки вебблог передбачає перед собою мету ділитися якоюсь інформацією з аудиторією, виникає необхідність десь зберігати данні, стосовно кожного такого посту. Тому доцільно для реалізації такого функціоналу створити окрему таблицю «Posts», яка буде зберігати всю необхідну для цього інформацію. У ній будуть знаходитися наступні поля:

– title: поле, яке зберігає в собі інформацію про заголовок публікації. Воно є строковим та обов'язковим;

– text: поле, у якому знаходиться основний контент публікації – інформація. Воно є строковим, обов'язковим та унікальним, щоб юзери не цупили один у одного контент та не публікували одну й та саму інформацію на блозі;

– tags: щоб користувачам блогу було легше та зручніше знаходити інформацію, в якій вони зацікавлені, функціонал вебзастосунку надає можливість авторам використовувати теги(ключові слова). Вони є не обов'язковим атрибутом і представляють собою масив, перерахованих строкових значень;

– viewsCount: для статистики та розуміння, являється опублікована інформація цікавою та корисною, кожен пост, має лічильник переглядів цілісного типу, значення якого по замовчуванню дорівнює нулю;

– user: поле, яке є зовнішнім ключем (FOREIGN KEY), та зсилається на данні (id) автора публікації, які знаходяться у таблиці «Users». Звичайно, поле є обов'язковим;

– imageUrl: необов'язкове поле, яке існує на випадок, якщо автор захоче виділитися гучним заголовком. По аналогії з полем «avatarUrl» із таблиці «Users» – також є строковим і зберігає в собі найменування картинки;

– поле «createdAt» містить дату та час створення документа, а «updatedAt» містить дату та час останнього оновлення документа. Ці поля можуть бути корисні для відстеження часу створення та оновлення даних в базі даних, і вони додаються автоматично без потреби вручну встановлювати значення для них;

– слід зазначити, що дане відношення також має ключ «timestamps: true». У MongoDB, параметр «timestamps: true» в схемі документа вказує, що колекція, до якої відноситься документ, буде автоматично додавати два додаткові поля: «createdAt» і «updatedAt». Ці поля автоматично оновлюються MongoDB при створенні нового документа (додавання в колекцію) і при оновленні існуючого документа (зміни в колекції).

Таким чином, у таблиці «Posts», інформація про публікації авторів буде зберігатися у полях title, text, tags, viewsCount, user та imageUrl.

Скрипт відношення представлений у лістингу 2.2.

Лістинг 2.2 Реалізація таблиці «Posts»:

```
const PostSchema = new mongoose.Schema(
  {
    title: {
```

```
    type: String,
    required: true,
  },
  text: {
    type: String,
    required: true,
    unique: true,
  },
  tags: {
    type: Array,
    default: [],
  },
  viewsCount: {
    type: Number,
    default: 0,
  },
  user: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
  imageUrl: String,
 },
 {
  timestamps: true,
 },
);
export default mongoose.model('Post', PostSchema);
```

Таким чином, на підставі створених відношень буде створена наступна БД вебзастосунку (рис. 2.4).

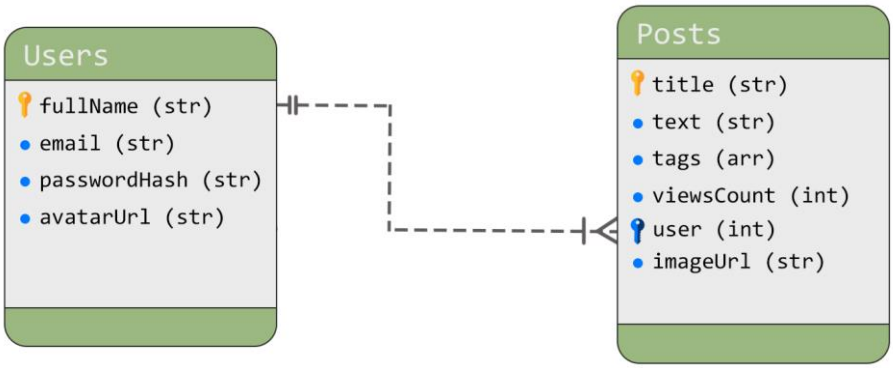

Рисунок 2.4 – Візуальне представлення моделі БД

2.3 Обґрунтування вибору середовища розробки

Для досягнення більшої ефективності та максимально комфортної роботи потрібно вибрати відповідне середовище, яке буде повністю влаштовувати програміста. Ідеальна IDE повинна мати зручний і інтуєтивно зрозумілий інтерфейс, що дозволяє розробникам ефективно працювати з кодом. Вона має включати широкий набір функцій, таких як автодоповнення коду, вбудовану документацію, інструменти для налагодження та відлагодження коду, а також можливості рефакторингу. Саме таким серидовищем можна вважати Visual Studio Code (рис. 2.5), якому віддають перевагу велика кількість розробників (особливо на ОС Windows).

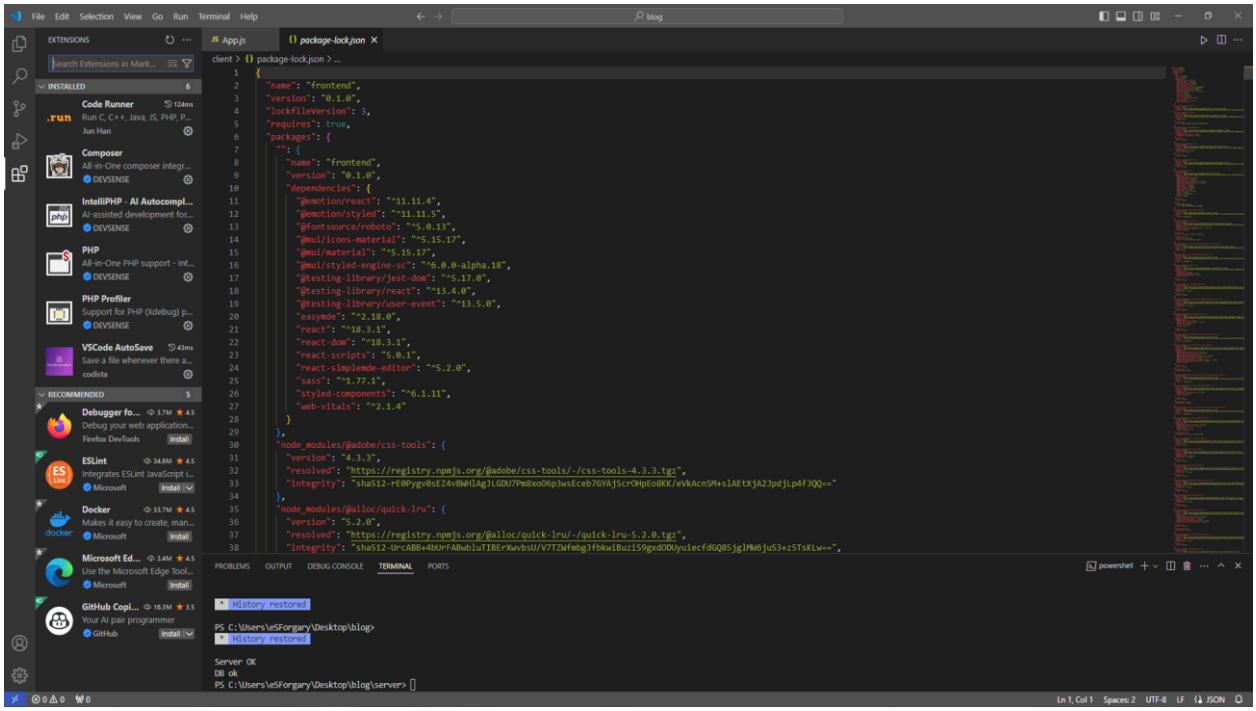

Рисунок 2.5 – Інтерфейс та можливості Visual Studio Code

VS Code – це потужний текстовий редактор, що розроблений компанією Microsoft. Він має широкий набір функцій і можливостей, які роблять його популярним серед розробників усіх рівнів.

VS Code є відкритим проектом і розповсюджується безкоштовно, що означає, що будь-який розробник може переглядати, редагувати та вносити вклад у його розвиток. Це створює сприятливе середовище для спільної роботи та співтворчості.

Редактор має велику кількість розширень та плагінів, які дозволяють розширити його функціональність та значно спростити процес розробки. Є можливість встановлювати плагіни для роботи з різними мовами програмування, інтеграції з різними сервісами та фреймворками, а також для вдосконалення робочого процесу.

VS Code надає зручні інструменти для швидкого переміщення між файлами, функціями та рядками коду. Це забезпечує швидкий доступ до необхідних ресурсів і сприяє підвищенню продуктивності розробника.

Редактор має функцію автодоповнення, яка пропонує можливі варіанти завершення коду на основі контексту та типу даних. Це значно полегшує написання коду, запобігає допущенню помилок і прискорює процес розробки.

VS Code ідеально поєднується з різними системами контролю версій, такими як Git, що дозволяє зручно вести роботу зі сховищами коду, використовуючи всі основні функції контролю версій. Інтеграція з Git дозволяє легко відслідковувати зміни в коді, створювати гілки і злиття, а також виконувати інші операції керування версіями.

У VS Code вбудована можливість взаємодії з командним рядком (терміналом) безпосередньо в редакторі, що дозволяє виконувати команди, запускати скрипти та встановлювати залежності без переходу до зовнішнього терміналу. Це забезпечує зручний та безперервний робочий процес.

VS Code надає підтримку для багатьох мов програмування, включаючи JavaScript, TypeScript, Python, Java, C#, HTML/CSS, і багато інших. Кожна мова має свої вбудовані функції та розширення для поліпшення робочого процесу, що робить VS Code універсальним інструментом для розробки різноманітних проектів.

Редактор підтримує спільну роботу над проектами за допомогою вбудованих інструментів для обміну кодом та комунікації з іншими членами команди. Це дозволяє легко об’єднувати зусилля та спільно працювати над проектами в реальному часі, спрощуючи командну роботу і підвищуючи ефективність розробки.

Таким чином, враховуючи усі можливості редактора Visual Studio Code можна вважати його ідеальним вибором для реалізації поставлених задач у розробці клієнтської та серверної частини вебзастосунку.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ДЕМОНСТРАЦІЯ РОБОТИ ВЕББЛОГУ

3.1 Програмна реалізація

У рамках кваліфікаційної роботи був розроблений вебзастосунок, який дозволяє користувачам публічно ділитися інформацією з іншими.

Програмна реалізація була повністю організована на мові програмування JavaScript. За клієнтську частину відповідає бібліотека React та препроцесор SCSS, а за серверну частину – Node.JS. Зберігання даних відбувається у нереляційній базі даних MongoDB.

Для початку розглянемо програмну реалізацію серверної частини, яка починається з лістингу 3.1

Лістинг 3.1 Реалізація серверної части вебзастосунку:

```
mongoose
  .connect(process.env.MONGODB_URI)
  .then(() => console.log('DB is work'))
  .catch((err) => console.log('DB error', err));

const app = express();

const storage = multer.diskStorage({
   destination: (_, __, cb) => {
    if (!fs.existsSync('uploads')) {
      fs.mkdirSync('uploads');
    }
    cb(null, 'uploads');
  },
  filename: (_, file, cb) => {
```

```
    cb(null, file.originalname);
  },
});

const upload = multer({ storage });

app.use(express.json());
app.use(cors());
app.use('/uploads', express.static('uploads'));

app.post('/auth/login', loginValidation, handleValidationErrors, UserController.login);
app.post('/auth/register', registerValidation, handleValidationErrors, UserController.register);
app.get('/auth/me', checkAuth, UserController.getMe);

app.post('/upload', checkAuth, upload.single('image'), (req, res) => {
  res.json({
    url: `/uploads/${req.file.originalname}`,
  });
});

app.get('/tags', PostController.getLastTags);

app.get('/posts', PostController.getAll);
app.get('/posts/tags', PostController.getLastTags);
app.get('/posts/:id', PostController.getOne);
app.post('/posts', checkAuth, postCreateValidation, handleValidationErrors, PostController.create);
app.delete('/posts/:id', checkAuth, PostController.remove);
```

```
app.patch(
  '/posts/:id',
  checkAuth,
  postCreateValidation,
  handleValidationErrors,
  PostController.update,
);

app.listen(process.env.PORT || 4444, (err) => {
  if (err) {
    return console.log(err);
  }

  console.log('Server is work');
});
```

Цей код являє собою сервер на Node.js, який використовує Express.js та Mongoose для підключення до бази даних MongoDB.

У перших рядках (`mongoose.connect`) налаштовується підключення до MongoDB за допомогою Mongoose. Якщо підключення успішне, в консоль виводиться повідомлення «DB is work», а в разі помилки – «DB error» з деталями помилки.

Далі створюється Express-застосунок (`const app = express()`). Налаштовується сховище для файлів з використанням `multer.diskStorage`. Цей блок коду створює папку `uploads`, якщо вона не існує, і зберігає файли з їхніми оригінальними іменами.

Ініціалізується `multer` з налаштованим сховищем (`const upload = multer({ storage })`).

Встановлюються кілька середніх обробників (`app.use`). `express.json()` дозволяє працювати з JSON-форматом, `cors()` вмикає підтримку CORS

(Cross-Origin Resource Sharing), а `express.static` дозволяє обслуговувати статичні файли з папки `uploads`.

Далі налаштовуються маршрути для обробки запитів. `app.post('/auth/login')` та `app.post('/auth/register')` використовуються для входу та реєстрації користувачів відповідно, з перевіркою вхідних даних та обробкою помилок.

Маршрут `app.get('/auth/me')` отримує інформацію про поточного авторизованого користувача за допомогою середнього обробника `checkAuth`.

Маршрут `app.post('/upload')` дозволяє авторизованим користувачам завантажувати файли. Завантажений файл зберігається в папці `uploads`, а відповідь містить URL до цього файлу.

Маршрут `app.get('/tags')` повертає останні теги, які використовуються в публікаціях.

Маршрути, що починаються з `/posts`, обробляють запити, пов'язані з публікаціями: отримання всіх публікацій, отримання останніх тегів, отримання окремої публікації за ідентифікатором, створення нової публікації, видалення та оновлення публікацій. Деякі з цих маршрутів використовують середні обробники `checkAuth`, щоб переконатися, що користувач авторизований.

Останній блок коду (`app.listen`) запускає сервер на вказаному порті (за замовчуванням 4444), виводячи в консоль повідомлення «Server is work» або помилку, якщо сервер не вдалося запустити.

Також слід розглянути процес шифрування, реалізований за допомогою JWT токену, який також відбувається на бекенді. Його подано у лістингу 3.2.

Лістин 3.2 Представлення реалізації механіки шифрування:

```
export default (req, res, next) => {
    const token = (req.headers.authorization || '').replace(/Bearer\s?/, '');
```

```
if (token) {
    try {
     const decoded = jwt.verify(token, ‘secret123’);

     req.userId = decoded._id;
     next();
    } catch (e) {
     return res.status(403).json({
       message: ‘Нет доступа’,
        });
     }
} else {
  return res.status(403).json({
     message: ‘Нет доступа’,
       });
     }
};
```

JSON Web Token (JWT) – це стандарт для безпечної передачі інформації між сторонами у вигляді JSON-об’єкта. Він працює наступним чином: клієнт автентифікується на сервері, після чого сервер створює токен, який містить закодовану інформацію про користувача та підписаний секретним ключем. Цей токен передається клієнту, який зберігає його (наприклад, у локальному сховищі або кукі). При наступних запитах клієнт додає цей токен до заголовка авторизації запиту. Сервер отримує запит, перевіряє токен за допомогою секретного ключа. Якщо токен валідний, сервер ідентифікує користувача і обробляє запит, інакше відмовляє у доступі.

Остання робота, яку здійснює сервер додатку відповідає за валідацію. Фрагмент коду реалізації представлено у лістингу 3.3.

Лістинг 3.3 Представлення валідатора:

```
export default (req, res, next) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json(errors.array());
  }
  next();
};
```

Цей код імпортує validationResult з бібліотеки express-validator і експортує проміжну функцію для обробки помилок валідації у запитах. Коли запит надходить до сервера, функція перевіряє наявність помилок валідації за допомогою validationResult(req). Якщо помилки є, функція повертає відповідь з кодом статусу 400 і масивом помилок у форматі JSON. Якщо ж помилок немає, викликається функція next(), яка передає обробку запиту до наступного проміжного програмного забезпечення або маршруту.

Розглянемо програмну реалізацію клієнтської частини. Вона складається з головної сторінки, розгорнутої публікації, сторінки, на якій автор пише пост, сторінки авторизаціє та сторінки регестрації. Усі елементи фронтенду створені за однаковим принципом та логікою де структура складається з React компенентів а стилі задані SCSS-ом. Таким чином, для прикладу можна розглянути всього один із них, замість того, щоб звертати увагу на кожен окремо. Тому, розглянемо програмну реалізацію сторінки розгорнутої публікації. Вона вказана у лістингу 3.4.

Лістинг 3.4 Програмна реалізація структури сторінки посту:

```
import React from ‘react’;
import clsx from ‘clsx’;
import IconButton from ‘@mui/material/IconButton’;
```

```
import DeleteIcon from '@mui/icons-material/Clear';
import EditIcon from '@mui/icons-material/Edit';
import EyeIcon from '@mui/icons-material/RemoveRedEyeOutlined';
import CommentIcon from '@mui/icons-material/ChatBubbleOutlineOutlined';

import styles from './Post.module.scss';
import { UserInfo } from '../UserInfo';
import { PostSkeleton } from './Skeleton';

export const Post = ({
    _id,
    title,
    createdAt,
  imageUrl,
  user,
    viewsCount,
 commentsCount,
    tags,
  children,
  isFullPost,
  isLoading,
    isEditable,
}) => {
    if (isLoading) {
      return <PostSkeleton />;
    }

   const onClickRemove = () => {};
```

```
return (
<div className={clsx(styles.root, { [styles.rootFull]: isFullPost })}>
{isEditable && (
  <div className={styles.editButtons}>
      <a href={`/posts/${_id}/edit`}>
      <IconButton color="primary">
            <EditIcon />
       </IconButton>
      </a>
      <IconButton onClick={onClickRemove} color="secondary">
      <DeleteIcon />
      </IconButton>
       </div>
      )}
      {imageUrl && (
          <img
      className={clsx(styles.image, { [styles.imageFull]: isFullPost })}
       src={imageUrl}
      alt={title}
     />
)}
<div className={styles.wrapper}>
  <UserInfo {...user} additionalText={createdAt} />
  <div className={styles.indention}>
    <h2 className={clsx(styles.title, { [styles.titleFull]: isFullPost })}>
     {isFullPost ? title : <a href={`/posts/${_id}`}>{title}</a>}
    </h2>
    <ul className={styles.tags}>
     {tags.map((name) => (
       <li key={name}>
```

```
            <a href={`/tag/${name}`}>#{name}</a>
          </li>
        ))}
        </ul>
        {children && <div className={styles.content}>{children}</div>}
        <ul className={styles.postDetails}>
        <li>
          <EyeIcon />
          <span>{viewsCount}</span>
              </li>
              <li>
              <CommentIcon />
              <span>{commentsCount}</span>
             </li>
            </ul>
      </div>
     </div>
    </div>
   );
};
```

Цей приклад не просто показує як влаштований React, а й наглядно демонструє як будуть виглядати усі компоненти у даному вебзастосунку. Компонент Post виконує роль відображення окремого поста на вебсторінці. Цей компонент отримує на вхід різноманітні дані про пост, такі як його заголовок, дату створення, зображення, інформацію про автора, кількість переглядів та коментарів, а також список тегів, які визначають тематику поста.

При передачі параметрів компонент може відображати пост у повному або скороченому вигляді, залежно від того, яку інформацію користувач хоче

побачити. Наприклад, у повному вигляді можуть бути показані всі дані про пост, включаючи зображення та інші додаткові елементи, а в скороченому вигляді – лише заголовок та основна інформація.

Крім того, якщо переданий відповідний параметр, компонент може надати користувачу можливість редагувати або видаляти пост. Це важливо для адміністраторів або авторів постів, які можуть потребувати управління своїм контентом.

У випадку, якщо дані про пост ще не завантажені, компонент може відображати скелетон (заготовку) поста, що дає користувачу зрозуміти, що контент ще завантажується. Це дозволяє створити більш зручний інтерфейс для користувача, який отримує повідомлення про стан завантаження контенту.

Внутрішня структура компонента Post складається з різних елементів, таких як зображення, інформація про автора, заголовок та теги поста. Також, компонент містить кнопки для редагування та видалення поста, якщо користувач має відповідні права.

За більшість стилів компонента Post відповідають готові стилі із бібліотеки mui, але окрім них також використовуються додаткові стилі SCSS. Приклад SCSS стилів приводиться у лістингу 3.5.

Лістинг 3.5 Приклад SCSS стилів для компоненту Post:

```
$blue: #4361ee;

.root {
  background-color: #fff;
  border: 1px solid #dedede;
  border-radius: 6px;
  overflow: hidden;
  margin-bottom: 15px;
  position: relative;
```

```
  &:hover {
    border: 1px solid $blue;
    box-shadow: 0 0 0 1px $blue;

    .editButtons {
      opacity: 1;
    }
  }

  &Full {
    &:hover {
      background-color: #fff;
      border: 1px solid #dedede;
      box-shadow: none;
    }
  }
}

.image {
  width: 100%;
  height: 300px;
  object-fit: cover;

  &Full {
    min-height: 300px;
    height: 100%;
  }
}
```

```
.wrapper {
  padding: 10px 20px 20px;
}

.content {
  margin: 30px 0 50px;

  p {
    font-size: 22px;
    line-height: 36px;
  }
}

.indention {
  padding-left: 40px;
}

.title {
  font-size: 28px;
  margin: 0;

  a {
    text-decoration: none;
    color: #000;

    &:hover {
      color: $blue;
    }
  }
```

```
  &Full {
    font-size: 42px;
    font-weight: 900;
  }
}

.tags {
  list-style: none;
  padding: 0;
  margin: 5px 0 0 0;

  li {
    display: inline-block;
    font-size: 14px;
    margin-right: 15px;
    opacity: 0.5;

    &:hover {
      opacity: 1;
    }

    a {
      text-decoration: none;
      color: #000;
    }
  }
}
...
```

У даному прикладі для демонстрації приводиться лише половина усього файлу стилів, так як він дуже об’ємний, але однотипний. Цього достатньо, щоб ознайомитися з принципом стилізації препроцесора SCSS.

У даному коді об’єктам, таким як «p», «a» або «.tags» призначаються різноманітні параметри, такі як «color», «text-decoration», «display», «padding», «margin» та інші. Параметри приймають ціле значення в процентах, пікселях та інших одиницях виміру або в якості ключових слів, які визначають ступінь дії відповідного параметру. З назви параметру стає зрозуміло на що саме він націлений, а з його значення – на скільки.

Логіка SCSS схожа на звичайний CSS, зокрема швидкодії та можливості вкладання одного тегу в інший. Через такі можливості SCSS зазнає більшої популярності та частіше використовується у масштабних проєктах.

3.2 Інструкція користувача

При запуску вебзастосунок користувач потрапляє на головну сторінку (рис. 3.1).

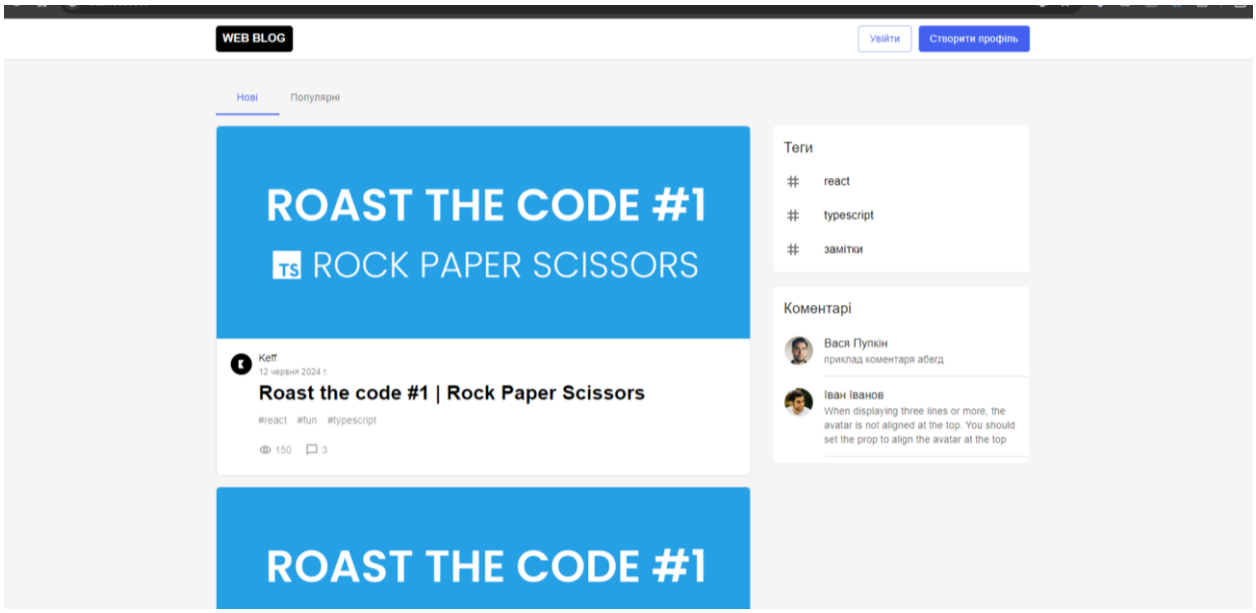

Рисунок 3.1 – Головна сторінка сайту

У шапці вебзастосунку знаходяться 3 кнопки-посилання: «Web Blog», «Увійти» та «Створити профіль». Кнопка : «Web Blog» повертає користувача на головну сторінку і завжди знаходиться у лівому верхньому кутку. Кнопка «Увійти» існує для вже зареєстрованих юзерів, та відповідно перенаправляє користувача на задану сторінку (рис. 3.2).

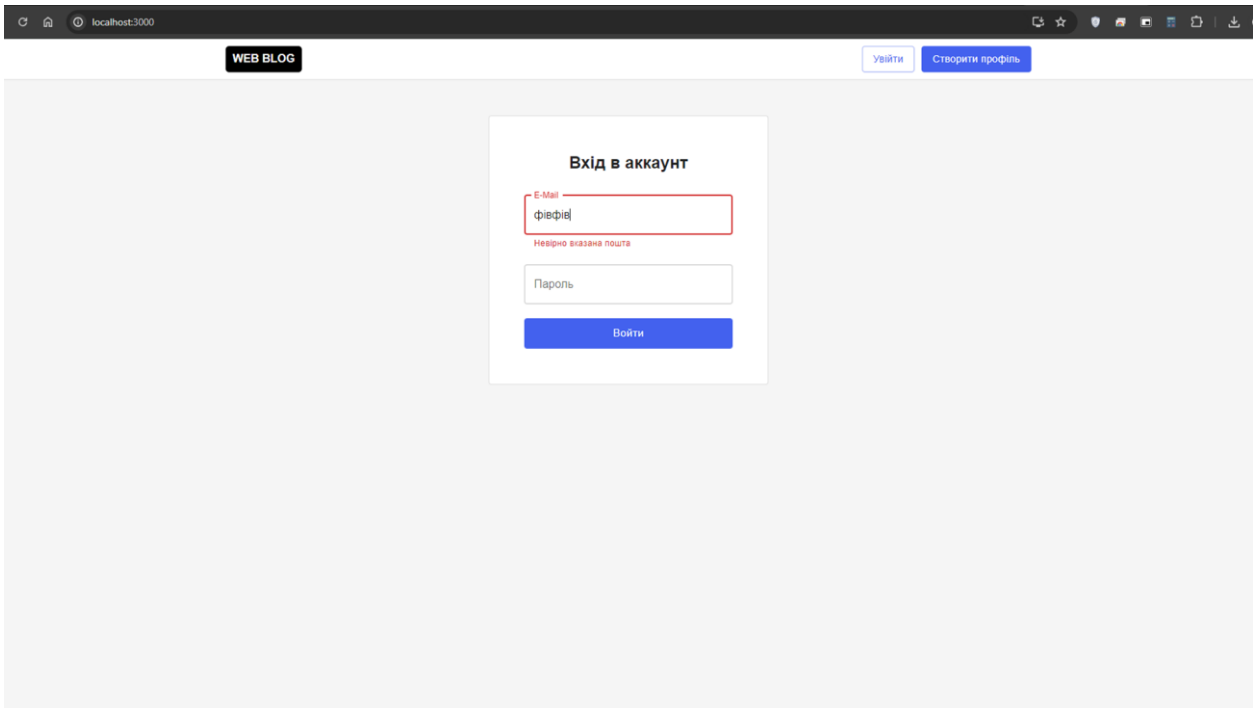

Рисунок 3.2 – Сторінка авторизації

Кнопка «Створити профіль» перенаправляє людину на сторінку реєстрації (рис. 3.3). На цій сторінці нові користувачі можуть приєднатися до вебблогу та заснувати свою персональну сторінку на ресурсі, після чого матимуть змогу авторизуватися. Цей процес обов’язковий для тих, хто хоче стати автором і почати робити публікації.

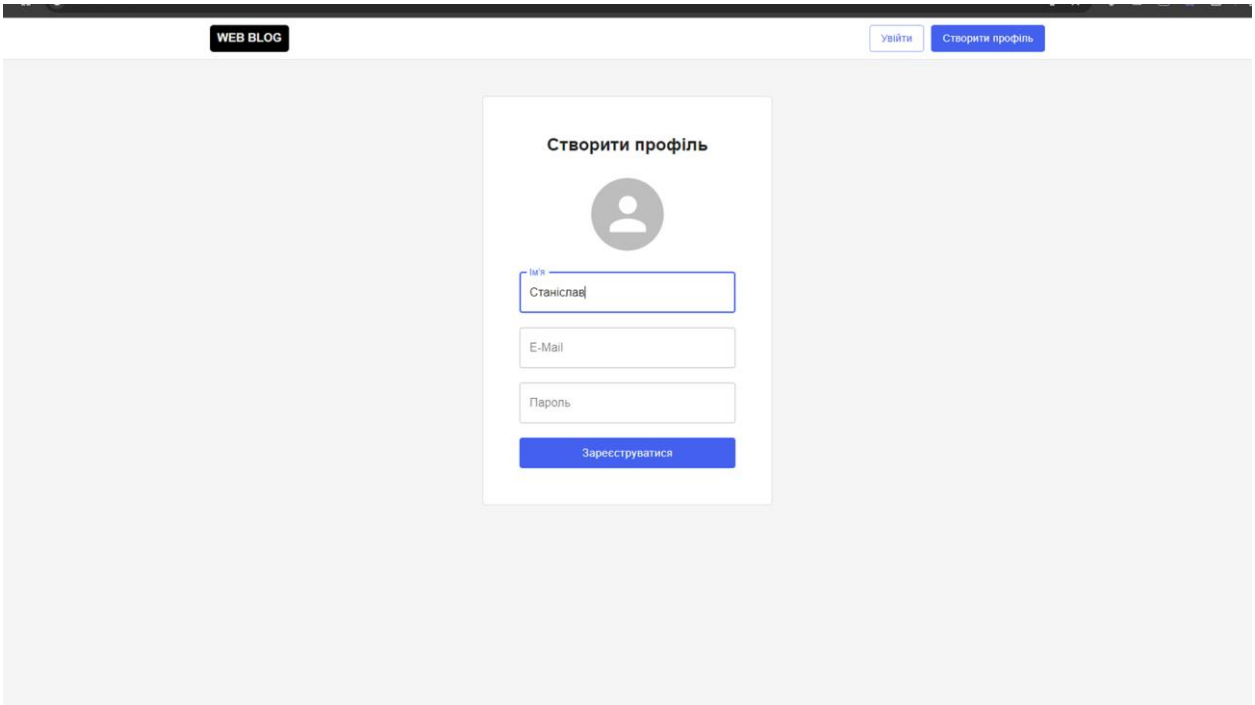

Рисунок 3.3 – Сторінка реєстрації

Після авторизації шапка користувача змінюється (рис 3.4) і тепер з’являється можливість створювати публікації або вийти з профілю.

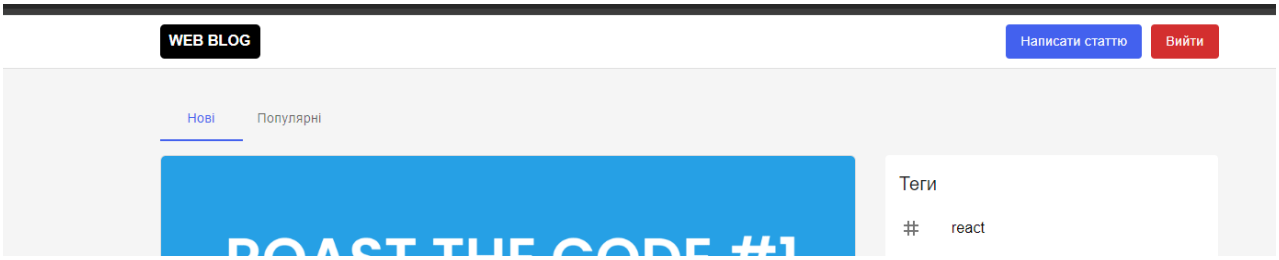

Рисунок 3.4 – Сторінка реєстрації

Відповідно, якщо натиснути на кнопку «Вийти» – користувач вийде зі свого профілю і матиме можливість тільки переглядати контент вебблогу. Для створення публікації необхідно натиснути на кнопку «Написати статтю», після чого він потрапляє на сторінку для створення посту (рис. 3.5).

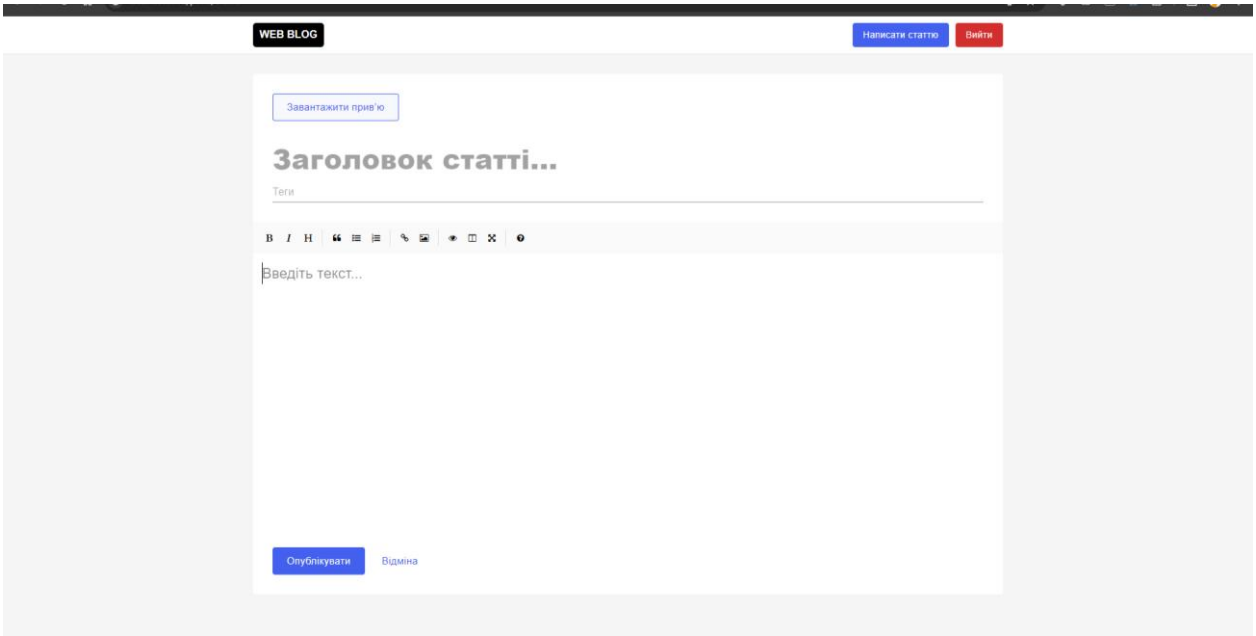

Рисунок 3.5 – Сторінка створення публікації

На даній сторінці користувач може написати заголовок для своєї майбутньої публікації, реалізувати CEO оптимізацію для свого посту за допомогою тегів, безпосередньо, написати саму статтю та додати зображення до свого посту, натиснув на кнопку «Завантажити прив’ю».

Коли користувач буде впевнений, що його публікація повністю готова для розміщення, залишиться лише натиснути на кнопку «Опублікувати».

Якщо користувач передумає робити пост – він легко може відмінити цей процес і натиснути на кнопку «Відміна», але в такому разі вся інформація, яку він розмістив у відповідних полях та зображення будуть стерті.

Також кожен користувач може подивитися на публікацію ум розгорнутому вигляді (рис 3.6).

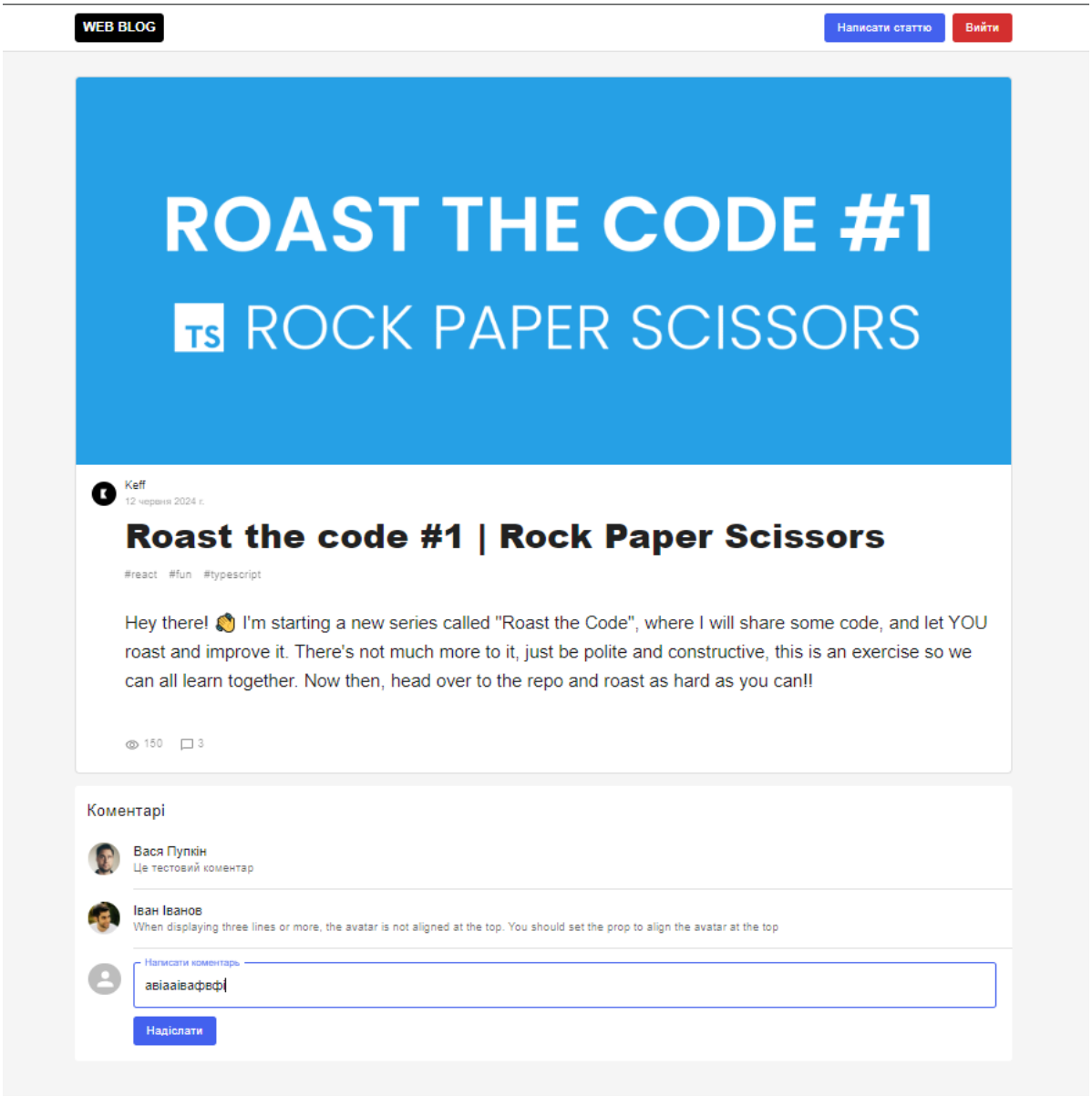

Рисунок 3.6 – Сторінка розгорнутої публікації

На розгорнутій сторінці статті можна побачити всю інформацію: тематика, текст посту, зображення, автора, дату створення публікації, кількість переглядів, залишені коментарі, та залишити коментар(якщо користувач авторизований).

Також, при необхідності, користувач може видалити або відредагувати свою публікацію. Для цього йому необхідно навести на свій пост мишкою, після чого в правому верхньому кутку з’явиться кнопки з хрестиком та олівцем (рис. 3.7).

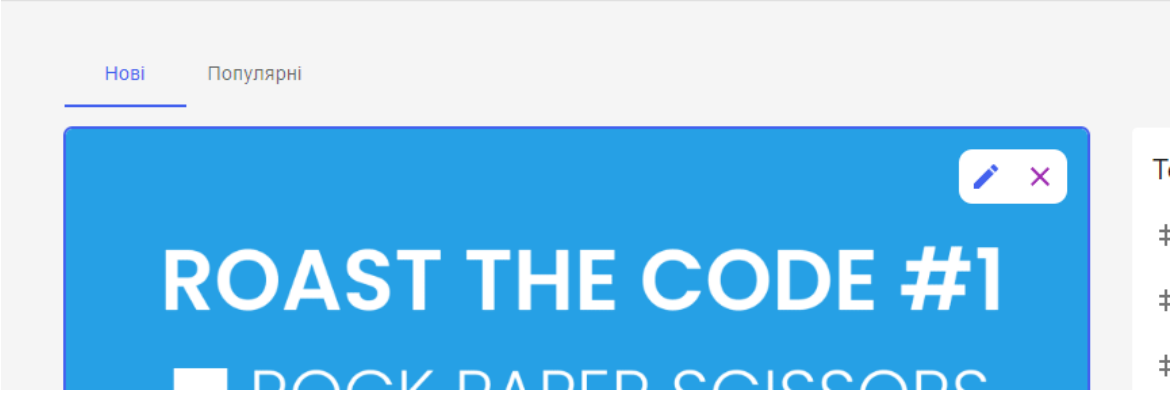

Рисунок 3.7 – Можливість взаємодіяти з існуючими публікаціями

Відповідно, якщо натиснути на хрестик – користувач видалить свій пост. При натиснути на олівець користувач побачить сторінку з публікацією як на (рис. 3.5), але з всією набраною інформацією, і матиме змогу змінити її.

ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблений вебзастосунок, завдяки якому будь-яка зареєстрована на сайті людина, має змогу поділитися з іншими користувачами певною інформацією.

Для реалізації поставлених задач у розробці вебсайту необхідно було розробити клієнтську частину для користувачів (фронтенд) та серверну частину для зберігання контенту і даних (бекенд).

Фронтенд був реалізований за допомогою мови програмування JacaScript, а саме бібліотеки React, яка ідеально підходить для створення швидкого та надійного інтерфейсу користувача. За стилізацію зовнішнього вигляду відповідає препроцесор SCSS та React бібліотека mui, завдяки яким вдалось уникнути написання зайвого коду і забезпечити його охайність та пришвидшити в декілька разів процес його написання.

Бекенд розроблено також за допомогою мови програмування JacaScript, але з використанням Node.js – сучасної платформи, перевагу котрій віддають розробники по всьому світу. Було створено БД для зберігання даних користувача, які надійно захищені токен шифрування JWT.

Результати роботи апробовано у вигляді тез доповідей під час Міжнародного молодіжного форуму «РАДІОЕЛЕКТРОНІКА І МОЛОДЬ У XXI СТОЛІТТІ», онлайн конференції «КОМП’ЮТЕРНИЙ ЗІР, СИСТЕМНИЙ АНАЛІЗ ТА МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Web Applications Types – Examples of Web Apps + Benefits and Challenges. URL: https://www.intelivita.com/blog/types-of-web-applications/ (дата звернення 14.04.2024)

2. Як створити веб-додаток: типи, переваги, принцип роботи. URL: https://wezom.com.ua/ua/blog/kak-sozdat-veb-prilozhenie (дата звернення 15.04.2024)

3. Загальні відомості про веб-застосунок PowerPoint Web App. URL: https://support.microsoft.com/uk-ua (дата звернення 02.05.2024)

4. Розробка з боку Front end – що це таке і чим відрізняється від Back end? URL: https://dan-it.com.ua/uk/blog/rozrobka-z-boku-front-end-shho-ce-take-i-chim-vidriznjaietsja-vid-back-end/ © Dan-it.com.ua (дата звернення 14.04.2024)

5. Хто такий frontend developer і що має вміти фронтенд-розробник. URL: https://itvdn.com/ua/specialities/frontend-developer (дата звернення 24.04.2024)

6. React Вікіпедія. URL: https://uk.wikipedia.org/wiki/React. (дата звернення 01.05.2024)

7. Wikipedia React (JavaScript library). URL: https://en.wikipedia.org/wiki/React_(JavaScript_library). (дата звернення 14.04.2024)

8. React – The library for web and native user interfaces. URL: https://react.dev/. (дата звернення 01.05.2024)

9. Flaticon – Access 15.5M+ vector icons & stickers. URL: https://www.flaticon.com/. (дата звернення 02.05.2024)

10. React – Getting Started URL: https://legacy.reactjs.org/docs/getting-started.html. (дата звернення 12.05.2024)

11. Tvoroshenko, I., Ahmad, M. A., Mustafa, S. K., Lyashenko, V., & Alharbi, A. R. (2020). Modification of models intensive development ontologies by fuzzy logic.

12. JSON Web Token. URL: https://en.wikipedia.org /wiki/ JSON_Web_Token (дата звернення 02.04.2024)

13. Visual Studio Code. URL: https://code.visualstudio.com/ (дата звернення 23.04.2024)

14. Хто такий Backend розробник? URL: https:// cases.media/en/article/khto-takii-backend-rozrobnik (дата звернення 02.05.2024)

15. Що таке back-end розробка і чим займається back-end розробник? URL: https://icstudio.online/post/shcho-take-back-end-rozrobka-i-chim-zajmayet sya-back-end-rozrobnik (дата звернення 08.05.2024)

16. Frontend, Backend і Fullstack: що це таке та в чому відмінності URL: https://goit.global/ua/articles/frontend-backend-i-fullstack-shcho-tse-take-ta-v-cho mu-vidminnosti/ (дата звернення 21.04.2024)

17. Node.js – Run JavaScript Everywhere. URL: https://nodejs.org/en (дата звернення 22.05.2024)

18. Розробка вебсайтів TECHNOLOGIES. URL: https://brander.ua /technologies/nodejs (дата звернення 12.05.2024)

19. Що таке Node.js? Основи серверної розробки на JavaScript. URL: https://devzone.org.ua/post/shcho-take-nodejs-osnovy-servernoyi-rozrobky-na-javascript (дата звернення 16.05.2024)

20. Що таке NodeJS/What is NodeJS? URL: https://nodejs.tutorial. in.ua/what-is-nodejs/ (дата звернення 02.05.2024)

21. MongoDB: The Developer Data Platform | MongoDB. URL: https://www.mongodb.com/ (дата звернення 02.05.2024)

22. Install MongoDB. URL: https://www.mongodb.com/ docs/manual/installation/ (дата звернення 02.05.2024)

23. MongoDB Tutorial. URL: https://www.w3schools.com/mongodb/ (дата звернення 30.05.2024)

24. Was ist MongoDB? URL: https://www.purestorage.com/de/knowledge/what-is-mongodb.html (дата звернення 27.04.2024)

25. Mongo. URL: https://github.com/mongodb/mongo (дата звернення 12.05.2024)

26. JavaScript Tutorial. URL: https://www.w3schools.com/js/ (дата звернення 30.05.2024)

27. The Modern JavaScript Tutorial. URL: https://javascript.info/ (дата звернення 25.04.2024)

28. JavaScript/Tutorials. URL: https://wiki.selfhtml.org/wiki/JavaScript/Tutorials (дата звернення 13.05.2024)

29. JavaScript basics. URL: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics (дата звернення 02.05.2024)

30. Тітов, С. В., & Тітова, О. В. (2015). Оцінка юзабіліті освітніх сайтів: методи і технології. Вісник Харківської державної академії культури. Серія: Соціальні комунікації, (47), 127-134.