Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка веб-платформи для соціальної мережі “Mesh” з використанням технологій React JS, NodeJS і MySQL

Виконав: студент IV курсу, групи СН-41
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Горин О.І.
(підпис) (прізвище та ініціали)

Керівник _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ _____
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«_27_» _червня_ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Горин Олександр Ігорович
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка вебплатформи для соціальної мережі "Mesh" з використанням технологій ReactJS, NodeJS I MySQL

Керівник роботи ст викл. кафедри КН Шимчук Григорій Валерійович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*07*» *травня* 2025 року № *4/7-444*

2. Термін подання студентом завершеної роботи 25 червня 2025 р.

3. Вихідні дані до роботи Літературні джерела щодо розробки веб-сервісів і розробки програмного забезпечення

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. Розділ. 1 Аналіз сучасних технологій та аналогічних рішень. 1.1 Аналіз сучасних веб -технологій для створення соц. мереж. 1.2 Огляд існуючих соціальних платформ. 1.3 Порівняння архітектурних рішень та стеків технологій. 1.4 Вибір інструментів для розробки. 1.5 Висновок до першого розділу. Розділ. 2 Проектування веб-платформи "Mesh". 2.1 Постановка задачі та функціональні вимоги до системи. 2.2 Нефункціональні вимоги. 2.3 Архітектура системи. 2.4 Проектування бази даних. 2.5 UX/UI проєктування інтерфейсу користувача. 2.6 Висновок до другого розділу. Розділ. 3 Реалізація веб-платформи. 3.1 Реалізація клієнтської частини на React JS. 3.2 Реалізація серверної частини на Node.js з використанням Express.js. 3.3 Взаємодія з базою даних MySQL. 3.4 Забезпечення авторизації, реєстрації та захисту даних. 3.5 Методологія тестування веб-платформи. 3.6 Висновок до третього розділу. 4. Безпека життєдіяльності, основи охорони праці. 4.1 Загальгі вимоги безпеки з охорони праці для користувачів ПК Загальні вимоги безпеки з охорони праці для користувачів ПК. 4.2 Критичні стани людини. Висновки. Перелік джерел. Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Сенчишин В.С.	27.06.2025	15.06.2025

7. Дата видачі завдання 27 січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	27.01.2025	*Виконано*
2.	Підбір джерел про Веб-розробку і технології	31.01.2025-03.02.2025	*Виконано*
3.	Опрацювання джерел по темі кваліфікаційної роботи	04.02.2025-06.02.2025	*Виконано*
4.	Оформлення розділу «Аналіз сучасних технологій та аналогічних рішень»	03.06.2025-05.06.2025	*Виконано*
5.	Оформлення розділу «Проектування веб-платформи “Mesh”»	06.06.2025-08.06.2025	*Виконано*
6.	Оформлення розділу «Реалізація веб-платформи»	09.06.2025-11.06.2025	*Виконано*
7.	Оформлення розділу «Безпека життєдіяльності, основи охорони праці»		*Виконано*
8.	Виконання завдання до підрозділу «Безпека життєдіяльності»	12.06.2025-13.06.2025	*Виконано*
9.	Виконання завдання до підрозділу «Основи охорони праці»	14.06.2025-15.06.2025	*Виконано*
10.	Оформлення кваліфікаційної роботи	16.06.2025-17.06.2025	*Виконано*
11.	Нормоконтроль	18.06.2025-19.06.2025	*Виконано*
12.	Перевірка на плагіат	20.06.2025	*Виконано*
13	Попередній захист кваліфікаційної роботи	21.06.2025	*Виконано*
14	Захист кваліфікаційної роботи	25.06.2025	*Виконано*

Студент _____ Горин О.І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

“Розробка веб-платформи для соціальної мережі “Mesh” з Використанням технологій React JS, NodeJS і MySQL”. // Кваліфікаційна робота освітнього рівня «Бакалавр» // Горин Олександр Ігорович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно-інформаційних систем і програмної інженерії, кафедра комп’ютерних наук, група СН-41 // Тернопіль, 2025 // С. – 53 , рис. – 11, табл. – 6, додат. – 23, презент. 14, бібліогр. – 32.

Ключові слова: Соціальна мережа, веб-платформа, React JS, Node.js, MySQL, фронтенд, бекенд, REST API.

Кваліфікаційна робота присвячена дослідженню процесу створення веб-платформи для соціальної мережі “Mesh” з використанням технологій React JS, Node.js та MySQL. Метою є розробка функціонального, безпечного і зручного для користувача веб-додатку.

У першому розділі описано сучасні технології розробки веб-застосунків і проведено аналіз аналогічних рішень. Висвітлено переваги використаних інструментів. Розглянуто можливості React, Node.js і MySQL у створенні соціальних платформ. Проаналізовано архітектурні підходи до побудови таких систем.

У другому розділі здійснено проектування веб-платформи. Досліджено структуру системи, побудову бази даних та архітектуру взаємодії клієнта з сервером. Подано схеми логіки застосунку та компонування інтерфейсу.

У третьому розділі описано реалізацію веб-платформи. Проаналізовано процес створення фронтенду і бекенду. Проведено інтеграцію компонентів та налаштування бази даних.

ANNOTATION

"Development of a Web Platform for the Social Network 'Mesh' Using React JS, NodeJS, and MySQL Technologies". // Bachelor's Qualification Thesis // Horyn Oleksandr Ihorovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, Group SN-41 // Ternopil, 2025 // p. – 53 , fig. – 11, tabl. – 6, app. – 23, present 14, annexes 14bibl. – 32.

Keywords: Social network, web platform, React JS, Node.js, MySQL, frontend, backend, REST API.

This bachelor's qualification thesis is devoted to the study of the process of developing a web platform for the social network "Mesh" using modern technologies such as React JS, Node.js, and MySQL. The main goal is to create a functional, secure, and user-friendly web application.

The first section describes modern web development technologies and analyzes similar existing solutions. The advantages of the selected tools are highlighted. The capabilities of React, Node.js, and MySQL in building social platforms are considered. Architectural approaches to the development of such systems are analyzed.

The second section presents the design of the web platform. The system structure, database design, and the architecture of client-server interaction are examined. Logic flowcharts and interface component structures are provided.

The third section outlines the implementation of the web platform. The process of developing the frontend and backend is analyzed. Integration of the system's components and database configuration is carried out.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (Application Programming Interface) – програмний інтерфейс, що надає набір готових функцій для взаємодії з програмним забезпеченням або сервісом.

БД – база даних, впорядкований набір структурованих даних, що зберігаються в електронному вигляді.

CRUD – абревіатура від англ. Create, Read, Update, Delete – основні операції з даними у веб-застосунках.

CSS (Cascading Style Sheets) – каскадні таблиці стилів, мова стилізації веб-сторінок.

DOM (Document Object Model) – об'єктна модель документа, що дозволяє скриптам динамічно змінювати вміст HTML-документів.

HTML (HyperText Markup Language) – мова розмітки гіпертексту, основа структури веб-сторінок.

JWT (JSON Web Token) – формат передачі даних для автентифікації та авторизації між клієнтом і сервером.

MySQL – система керування реляційними базами даних з відкритим вихідним кодом.

Node.js – серверне середовище виконання JavaScript, яке дозволяє створювати масштабовані веб-додатки.

NPM (Node Package Manager) – менеджер пакетів для JavaScript, що використовується для встановлення бібліотек у Node.js-проєктах.

React JS – JavaScript-бібліотека для побудови інтерфейсів користувача, розроблена компанією Meta.

REST API (Representational State Transfer) – архітектурний стиль веб-сервісів, що базується на стандартних HTTP-запитах.

SQL (Structured Query Language) – мова структурованих запитів, яка використовується для взаємодії з базами даних.

UI (User Interface) – інтерфейс користувача, спосіб взаємодії людини з програмним забезпеченням.

UX (User Experience) – досвід користувача, загальне враження від взаємодії з інтерфейсом чи системою.

Хешування – процес перетворення вхідних даних у фіксований рядок символів (хеш), який не дозволяє відновити початкову інформацію.

ЗМІСТ

ВСТУП

Актуальність теми. Внаслідок глобалізації та стрімкого розвитку цифрових технологій соціальні мережі стали невід'ємною частиною повсякденного життя мільйонів користувачів у всьому світі. Вони забезпечують ефективну комунікацію, обмін інформацією, формування спільнот за інтересами та підтримку соціальних зв'язків. У зв’язку з цим зростає потреба у створенні нових, гнучких та інноваційних рішень для побудови соціальних платформ.

Тому розробка веб-платформи для соціальної мережі з використанням сучасних технологій, таких як React JS, Node.js та MySQL, є актуальним напрямком сучасних досліджень у галузі веб-програмування та інженерії програмного забезпечення.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Бакалавр» є підвищення якості послуг у сфері соціальних платформ шляхом розробки ефективної, функціональної та безпечної веб-платформи для соціальної мережі “Mesh” з використанням сучасних веб-технологій.

Для досягнення поставленої мети потрібно виконати ряд завдань, зокрема:

– проаналізувати стан досліджень у сфері створення соціальних мереж і сучасних веб-технологій;

– дослідити можливості застосування React JS, Node.js і MySQL у веб-розробці;

– спроєктувати архітектуру веб-платформи;

– реалізувати функціональний прототип веб-додатку;

– провести тестування платформи та оцінити її ефективність;

– надати рекомендації щодо забезпечення безпеки життєдіяльності під час розробки програмного забезпечення.

Практичне значення одержаних результатів. Результати даної роботи можуть бути використані для створення реальних соціальних платформ, орієнтованих на швидку і безпечну взаємодію користувачів. Запропоноване

рішення може слугувати основою для подальшого розвитку програмних продуктів у сфері соціальних сервісів. Практична реалізація веб-платформи “Mesh” демонструє можливість ефективного застосування сучасних фреймворків і технологій у розробці повноцінних веб-застосунків.

РОЗДІЛ 1 АНАЛІЗ СУЧАСНИХ ТЕХНОЛОГІЙ ТА АНАЛОГІЧНИХ РІШЕНЬ

1.1 Аналіз сучасних веб-технологій для створення соціальних мереж

У сучасному світі соціальні мережі стали невід'ємною частиною повсякденного життя мільйонів користувачів. Розробка таких систем вимагає використання найновіших і найефективніших веб-технологій, здатних забезпечити високу продуктивність, масштабованість, безпеку та зручність використання. У цьому підрозділі розглянуто основні технології, які використовуються під час створення соціальних веб-платформ.

Для розробки клієнтської частини сучасних веб-додатків активно використовуються JavaScript-фреймворки, зокрема:

Для розробки клієнтської частини сучасних веб-застосунків часто обирають один із популярних фреймворків: розроблений Facebook *React.js*, що дозволяє створювати швидкі односторінкові застосунки; легкий та простий *Vue.js*, який чудово підходить для невеликих проєктів; або потужний *Angular* від Google для великих корпоративних рішень. Водночас серверна частина зазвичай реалізується за допомогою *Node.js* — середовища виконання JavaScript для побудови масштабованих веб-серверів, часто в поєднанні з мінімалістичним фреймворком *Express.js* для спрощення створення REST API.

Для зберігання даних використовуються:

Для зберігання структурованої інформації, такої як профілі користувачів, повідомлення чи налаштування, використовується реляційна СУБД MySQL, яка забезпечує надійність і цілісність даних. Водночас для роботи з неструктурованими даними або в реальногочасових функціях, як-от чати, можуть застосовуватись NoSQL-рішення, наприклад, MongoDB.

Комунікація між фронтендом і бекендом зазвичай реалізується за допомогою REST API або GraphQL, що дозволяє ефективно обмінюватися даними.

У розробці сучасних соціальних мереж важливе значення мають такі аспекти, як безпека (автентифікація, авторизація, захист від CSRF/XSS), адаптивність дизайну (mobile-first), швидке реагування інтерфейсу (SPA), а також інтеграція з зовнішніми сервісами (платіжні шлюзи, карти, аналітика).

1.2 Огляд існуючих соціальних платформ (Facebook, Twitter, LinkedIn тощо)

Для розробки ефективної та конкурентоспроможної соціальної мережі важливо врахувати досвід провідних платформ, які вже зайняли свою нішу на ринку. У цьому підрозділі буде розглянуто ключові функціональні, архітектурні та технологічні особливості таких платформ як Facebook, Twitter, LinkedIn та інших.

Facebook

Facebook є однією з наймасштабніших соціальних мереж у світі, з понад 2,9 мільярдами активних користувачів щомісяця (станом на 2024 рік). Основна архітектура системи побудована на мікросервісах та масштабованих хмарних інфраструктурах. Використовується GraphQL (розроблений Facebook) для зручної взаємодії клієнта з бекендом. Інтерфейс побудований на React.js, що забезпечує високий рівень динамічності та гнучкість UI.

Twitter

Twitter спеціалізується на коротких повідомленнях (твітах) і є прикладом високопродуктивної системи з обробкою великих обсягів даних у реальному часі. Його бекенд реалізовано з використанням Scala та Java, а для обміну повідомленнями застосовується Kafka. Twitter також активно використовує Redis і Memcached для кешування даних і зменшення навантаження на базу даних.

LinkedIn

LinkedIn орієнтований на професійні зв’язки, пошук роботи та рекрутингові сервіси. Його архітектура є прикладом масштабованої

корпоративної системи, побудованої з використанням Java, Kafka, Samza, Hadoop, а також спеціалізованої NoSQL-бази Espresso для обробки великої кількості зв’язків між користувачами.

Instagram ma TikTok

Ці платформи орієнтовані на мультимедійний контент — фото, відео, сторіс. Їх архітектура повинна забезпечувати швидке завантаження, обробку та доставку великого обсягу відеоданих, що реалізується через CDN, мікросервіси та масштабовані бекенди. Для користувацького інтерфейсу використовується React Native.

1.3 Порівняння архітектурних рішень та стеків технологій

Архітектура соціальної мережі має відповідати вимогам до масштабованості, надійності, безпеки та швидкодії. Розглянемо поширені архітектурні підходи та порівняємо технологічні стеки, що використовуються у найбільших платформах.

Моноліт vs Мікросервіси

Історично багато сервісів починали з монолітної архітектури (наприклад, Facebook на PHP), однак зі зростанням обсягу користувачів вони перейшли до мікросервісної архітектури. Це дозволяє кожному компоненту (авторизація, стрічка новин, повідомлення, профілі) функціонувати незалежно, масштабуватися окремо й розгортатися без зупинки всієї системи.

Технологічні підходи

Для побудови сучасних веб-додатків на фронтенді зазвичай використовують React, Vue або Angular з підтримкою SSR (Server-Side Rendering) для покращення SEO та продуктивності, тоді як на бекенді застосовують Node.js, Python (Django або Flask), Java чи Scala — залежно від складності системи. У якості баз даних використовуються як реляційні (MySQL, PostgreSQL), так і NoSQL-рішення (MongoDB, Cassandra, DynamoDB) для швидкого доступу до великих обсягів неструктурованих даних. Для кешування

популярних запитів і прискорення доступу до інформації застосовуються Redis або Memcached. Потокова обробка подій у реальному часі забезпечується за допомогою Kafka чи RabbitMQ, а для зберігання й розповсюдження статичного контенту використовуються хмарні сервіси на кшталт Amazon S3 або Google Cloud Storage через CDN.

Таблиця 1.1 – Порівняння платформ:

Платформа	**Frontend**	**Backend**	**DB**	**Інші сервіси**
Facebook	React	Hack, C++	MySQL, TAO	GraphQL, Memcached
Twitter	React	Scala, Java	MySQL	Kafka, Redis
LinkeIn	Ember	Java	Espress, Oracle	Kafka, Hadoop
TikTok	React Native	Python	MongoDB	CDN, AI алгоритм

Таблиця демонструє, які технології використовують великі соціальні платформи для ефективної роботи своїх сервісів. Кожна з них підбирає стек інструментів відповідно до своїх вимог до продуктивності, масштабованості та обробки даних у реальному часі.

1.4. Вибір інструментів для розробки (React, Node.js, MySQL): обґрунтування

Розробка соціальної мережі “Mesh” передбачає створення сучасного, швидкого та адаптивного веб-застосунку. На основі проведеного аналізу технологій та вимог до системи, обрано наступні інструменти:

React.js:

- Створення компонентної архітектури фронтенду.
- Підтримка SPA: швидкий інтерфейс без перезавантаження сторінок.
- Величезна екосистема (Redux, React Router, Material UI).

– Підтримка SSR (за потреби) для SEO-оптимізації.

Node.js + Express.js:

– Node.js дозволяє реалізувати сервер на JavaScript, що спрощує взаємодію між фронтендом і бекендом.

– Асинхронна модель вводу/виводу забезпечує обробку великої кількості запитів у реальному часі.

– Express.js — легкий фреймворк, що дозволяє швидко створювати REST API, управляти маршрутами та middleware.

MySQL:

– Реляційна СУБД, яка забезпечує структуроване зберігання даних (користувачі, пости, коментарі, лайки).

– Підтримка ACID-транзакцій, індексації, зовнішніх ключів.

– Добре масштабована за рахунок реплікації та шардингу.

Обраний стек дозволяє створити масштабовану архітектуру з чітким поділом на фронтенд, бекенд і базу даних, що спрощує розробку, тестування й підтримку системи.

1.5 Висновок до першого розділу

У результаті аналізу сучасних технологій та аналогічних платформ, зроблено висновок, що розробка соціальної мережі потребує:

– використання надійної, масштабованої та гнучкої архітектури;

– застосування сучасних фреймворків для фронтенду (React) та бекенду (Node.js);

– вибору СУБД, яка забезпечує збереження структурованої інформації (MySQL).

Було проаналізовано функціонал, інтерфейси та технічні характеристики таких провідних платформ, як Facebook, Twitter, LinkedIn. На основі цього сформовано обґрунтований вибір технологій для реалізації веб-платформи

“Mesh”, який базується на принципах продуктивності, безпеки та зручності користувача.

Цей розділ закладає концептуальну основу для наступного етапу – проектування архітектури та функціоналу власної системи, що буде розглянуто у другому розділі.

РОЗДІЛ 2 ПРОЕКТУВАННЯ ВЕБ-ПЛАТФОРМИ “MESH”

2.1 Постановка задачі та функціональні вимоги до системи

Соціальна мережа “Mesh” створюється як платформа, що забезпечує користувачам можливість реєстрації, створення профілю, публікації контенту, взаємодії між собою у формі підписок, лайків, коментарів і повідомлень.

Поставлена мета

Розробити функціональну веб-платформу, яка надає базовий та розширений функціонал соціальної мережі з сучасним, адаптивним і безпечним інтерфейсом.

Поставлені задачі:

– Створення клієнтської частини з використанням React.js.
– Реалізація серверної логіки на Node.js.
– Організація взаємодії через REST API.
– Розробка структури бази даних у MySQL.
– Забезпечення авторизації, безпеки та зберігання даних.

Функціональні вимоги

Функціональність соціальної платформи включає реєстрацію та авторизацію користувачів із валідацією даних, шифруванням паролів і підтвердженням через e-mail. Кожен користувач має особистий профіль із можливістю редагування, додавання аватара, біографії та перегляду кількості підписників.

У стрічці новин відображаються пости тих, на кого підписаний користувач, з можливістю додавання публікацій із текстом, зображеннями, датою створення та функцією видалення. Реалізована система лайків і коментарів для взаємодії з контентом, а також механізм підписки та відписки для керування соціальними зв’язками.

Є пошук користувачів за іменем або email, а за потреби – адміністративна панель для модерації постів і профілів.

2.2 Нефункціональні вимоги (продуктивність, масштабованість, безпека)

Нефункціональні вимоги визначають якісні характеристики системи, що забезпечують її стабільну роботу за умов реального навантаження.

Продуктивність

Система забезпечує швидкий час відгуку сервера на запити API – до 200 мс, із підтримкою не менше 1000 одночасних користувачів без втрати продуктивності. Для пришвидшення завантаження інтерфейсу застосовується асиметрична оптимізація рендерингу на фронтенді.

Масштабованість

Проєкт підтримує горизонтальне масштабування серверної частини на Node.js за допомогою контейнеризації (Docker), а також реплікацію бази даних для покращення продуктивності. Часті запити кешуються за допомогою Redis.

Безпека

Реалізовано авторизацію користувачів через JWT, шифрування паролів з використанням bcrypt, захист від XSS, CSRF та SQL-ін’єкцій. Передача даних здійснюється через захищене HTTPS-з’єднання, а введення перевіряється як на клієнті, так і на сервері.

Інше

Інтерфейс адаптується під різні типи пристроїв, реалізовано логування помилок і подій за допомогою інструментів на зразок Winston або LogRocket, а також впроваджено регулярне резервне копіювання бази даних.

2.3. Архітектура системи (клієнт-серверна модель)

Архітектура веб-платформи базується на класичній трирівневій клієнт-серверній моделі, що складається з трьох основних компонентів:

Клієнт (Front-end):

– Побудований на React.js.

– Спілкується із сервером через REST API.

– Реалізує SPA-підхід з використанням React Router, Axios, Redux.

Сервер (Back-end):

– Node.js + Express.js.

– Обробка HTTP-запитів, аутентифікація, логіка роботи з базою даних.

– REST API доступне через /api/v1/...

База даних (MySQL):

– Зберігає всі структуровані дані: користувачі, пости, лайки, підписки, повідомлення.

– Використання зовнішніх ключів, індексів для оптимізації запитів.

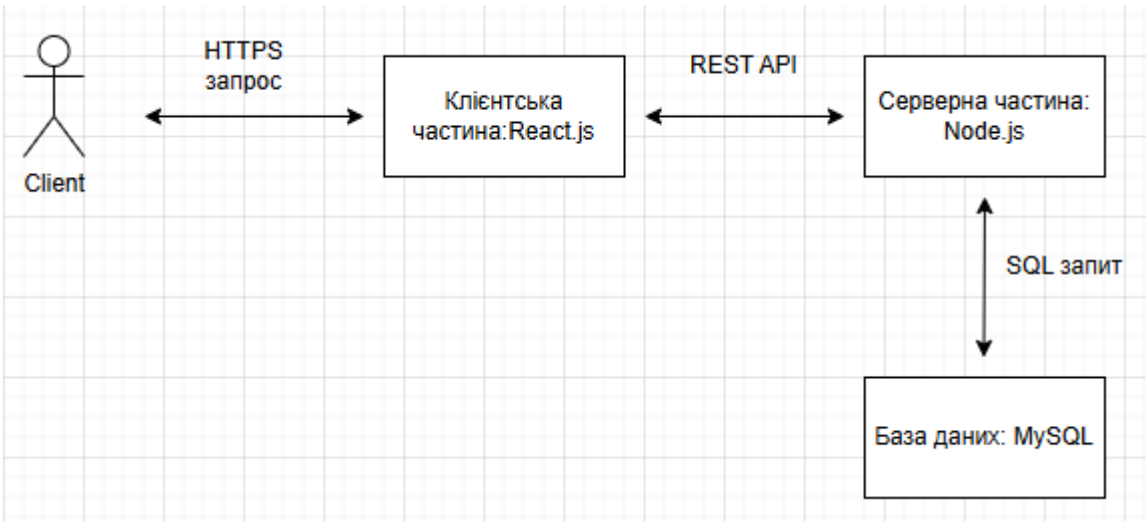

Рисунок 2.1 – Схема взаємодії клієнтської і серверної частини разом

На представленій схемі показано технологічний стек та взаємодію його компонентів. Frontend-частина, реалізована на React.js, комунікує з backend-сервером на Node.js через REST API. Для зберігання даних використовується реляційна система управління базами даних MySQL, взаємодія з якою відбувається на рівні сервера за допомогою SQL-запитів.

2.4 Проектування бази даних (ER-діаграми, структури таблиць)

Структура бази даних має забезпечити логічну цілісність, зв’язність даних та їх оптимальне зберігання.

Основні таблиці:

- users – користувачі системи.
- posts –пости користувачів.
- comments –коментарі до постів.
- likes –лайки на пости.
- followers –підписки користувачів.

Таблиця 2.1 – Приклад структури users:

Поле	Тип	Опис
id	INT, PK	Унікальний ID користувача
email	VARCHAR	Логін(унікальний)
password	VARCHAR	Хешований пароль
username	VARCHAR	Ім’я користувача
avatar_url	TEXT	Посилання на аватар
created_at	TIMESTAMP	Дата створення профілю

Таблиця users містить основну інформацію про користувачів системи. Вона зберігає унікальний ідентифікатор, логін (email), хешований пароль, ім’я користувача, аватар та дату реєстрації.

Таблиця 2.2 – Приклад структури posts:

Поле	Тип	Опис
id	INT, PK	Унікальний ID поста
desc	VARCHAR	Форма для написання тексту поста
img	VARCHAR	Фото яке буде вставлене в пості
userId	INT	Унікальне ID користувача який створює пост
created_at	TIMESTAMP	Дата створення поста

Таблиця posts зберігає інформацію про пости користувачів. Містить унікальний ідентифікатор поста, текст опису, зображення та ID користувача, який створив пост.

Таблиця 2.3 – Приклад структури comments:

Поле	Тип	Опис
id	INT, PK	Унікальний ID користувача
desc	VARCHAR	Логін(унікальний)
created_at	TIMESTAMP	Дата створення профілю
userId	INT	Унікальний ID користувача який написав коментар під постом
postId	INT	Унікальний ID поста під яким залишили коментар

Таблиця comments зберігає коментарі користувачів до постів. Вона містить унікальний ID коментаря, текст коментаря, дату створення, а також посилання на користувача (userId) та пост (postId), до якого належить коментар.

Таблиця 2.4 – Приклад структури likes:

Поле	Тип	Опис
id	INT, PK	Унікальний ID лайка
userId	INT	Унікальне ID користувача який залишає лайк
postId	INT	Унікальне ID поста під яким саме постом залишили лайк

Таблиця likes відображає інформацію про лайки користувачів. Вона містить унікальний ID лайка, а також посилання на користувача (userId) та пост (postId), якому поставлено лайк.

Таблиця 2.4 – Приклад структури followers:

Поле	Тип	Опис
id	INT, PK	Унікальний ID запису
followerUserId	INT	ID користувача, який підписався на вас
followeredUserId	INT	ID користувача, на якого ви підписалися

Таблиця followers зберігає інформацію про підписки між користувачами. Містить унікальний ID запису, ID користувача, який підписався (followerUserId), та ID користувача, на якого підписалися (followeredUserId).

2.5 UX/UI проєктування інтерфейсу користувача

Зовнішній вигляд і зручність взаємодії з платформою напряму впливають на її успіх. Проєктування інтерфейсу ґрунтується на принципах адаптивності, простоти та інтуїтивності.

Основні принципи UX/UI:

– Mobile-first дизайн –оптимізація для смартфонів.

– Чітка навігація –логічне меню, стрічка новин, профіль.
– Мінімалістичний стиль –чистий інтерфейс без перевантаження.
– Темна/світла тема –для комфорту користувачів.
– Стан системи –відображення завантаження, помилок, підтверджень.

Основні екрани:

– Головна стрічка –список публікацій.
– Профіль користувача –фото, ім’я, пости, підписники.
– Редагування профілю –форма з полями.
– Форма публікації поста –textarea + upload фото.
– Коментарі до поста –розгортання/схлопування.
– Особисті повідомлення –чат у стилі месенджера.

Інтерфейс проєктувався з урахуванням Material Design та рекомендацій Nielsen Norman Group.

Дизайн був побудований із застосуванням основних принципів UX/UI-дизайну, що передбачає логічне розміщення навігаційних елементів, візуальну рівновагу та комфортне сприйняття інформації.

Основнною кольоровою палітрою сайту стали білосніжні відтінки (#F6F3F3), які асоціюються з чистотою і невиністтю, а також синій (#5271FF) в який застосовується у деяких компонентах, що додає контрасність.

У цій роботі використовувався один основний шрифт це Open Sans.

2.6 Висновок до другого розділу

У другому розділі було здійснено проектування архітектури, функціональності та структури веб-платформи “Mesh”. Було визначено функціональні та нефункціональні вимоги, обґрунтовано вибір клієнт-серверної моделі, розроблено структуру бази даних та описано підходи до UI/UX-дизайну.

Проєкт “Mesh” орієнтований на створення легко масштабованої, безпечної та привабливої для користувачів платформи, яка поєднує інтуїтивний інтерфейс

з потужною серверною логікою. У наступному розділі буде розглянуто безпосередню реалізацію системи, описано використані інструменти, середовище розробки та результати тестування.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ВЕБ-ПЛАТФОРМИ

3.1 Реалізація клієнтської частини на React JS

Веб-платформа "Mesh" реалізована на основі **React JS**, що дозволяє створити динамічний та швидкий інтерфейс для користувачів. Вибір React JS для розробки клієнтської частини був зумовлений кількома факторами: високою продуктивністю, можливістю масштабування, простотою компонування інтерфейсних елементів, а також зручністю у підтримці коду завдяки використанню компонентного підходу.

Основним аспектом реалізації клієнтської частини є створення адаптивного та інтуїтивно зрозумілого інтерфейсу користувача. Для цього використовуються компоненти, які відображають різні елементи інтерфейсу, такі як кнопки, форми, списки, елементи навігації тощо. Завдяки використанню React Router реалізовано зручну навігацію між сторінками без перезавантаження браузера, що дозволяє забезпечити безшовний досвід користувача.

3.1.1 Вибір технології React JS

React JS обрано для розробки клієнтської частини через її високу продуктивність, компонентний підхід та широку підтримку спільноти. React дозволяє ефективно оновлювати інтерфейс користувача завдяки віртуальному DOM, що зменшує навантаження на браузер та покращує взаємодію з користувачем.

Переваги React JS:

– Віртуальний DOM: React використовує віртуальний DOM для оптимізації оновлень інтерфейсу, що забезпечує високу продуктивність .

– Компонентний підхід: Розробка інтерфейсу за допомогою незалежних компонентів спрощує підтримку та розширення застосунку.

– Широка екосистема: React має велику кількість бібліотек та інструментів, що полегшують розробку та тестування.

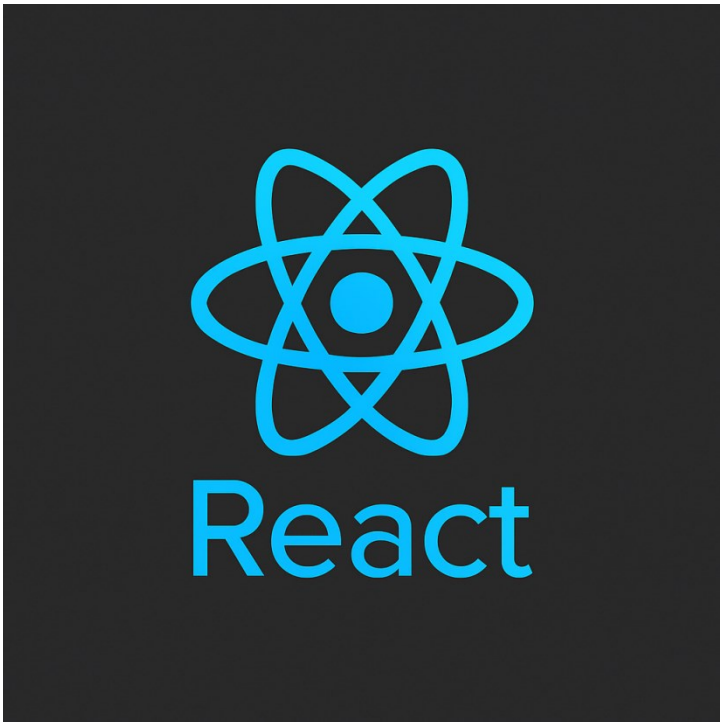

Рисунок 3.1 – Логотип фреймовка React JS

Ця фотографія демонструє офіційний логотип фреймворку React JS. Логотип виконаний у блакитних тонах на чорному тлі та являє собою стилізоване зображення, що нагадує атом або молекулу, з підписом "React" внизу. Це зображення є впізнаваним символом однієї з найпопулярніших технологій для розробки сучасних користувацьких інтерфейсів.

3.1.2 Структура клієнтського застосунку

У цьому проекті структура клієнтського застосунку побудована відповідно до сучасних принципів організації коду, що дозволяє легко підтримувати й масштабувати застосунок. Основними каталогами є components/ для універсальних компонентів, pages/ для сторінок застосунку, services/ для роботи з API, а також contexts/, які використовуються для управління глобальним станом.

Приклад структури проекту:

CSS

```
src/
├── components/
│   ├── Button.js
│   ├── Modal.js
│   └── ...
├── pages/
│   ├── Home.js
│   ├── Profile.js
│   └── ...
├── services/
│   ├── api.js
│   └── ...
├── contexts/
│   ├── AuthContext.js
│   └── ...
└── assets/
    ├── images/
    └── styles/
```

Також важливою частиною є використання структурованих компонентів, таких як форми, кнопки, списки тощо, що дозволяє розробникам швидко додавати нові функції, не впливаючи на вже існуючі.

3.1.3 Реалізація основних інтерфейсів користувача

Клієнтська частина включає кілька основних інтерфейсів, таких як головна сторінка, профіль користувача, налаштування і сторінки новин. Усі інтерфейси адаптивні, що забезпечує правильне відображення як на настільних комп’ютерах, так і на мобільних пристроях. Інтерфейс реалізований за допомогою компонентного підходу, що дозволяє повторно використовувати код і легко вносити зміни.

Також важливим аспектом є інтеграція з бібліотеками для управління формами та валідації, що спрощує обробку введених даних користувачем та підвищує зручність користування веб-платформою.

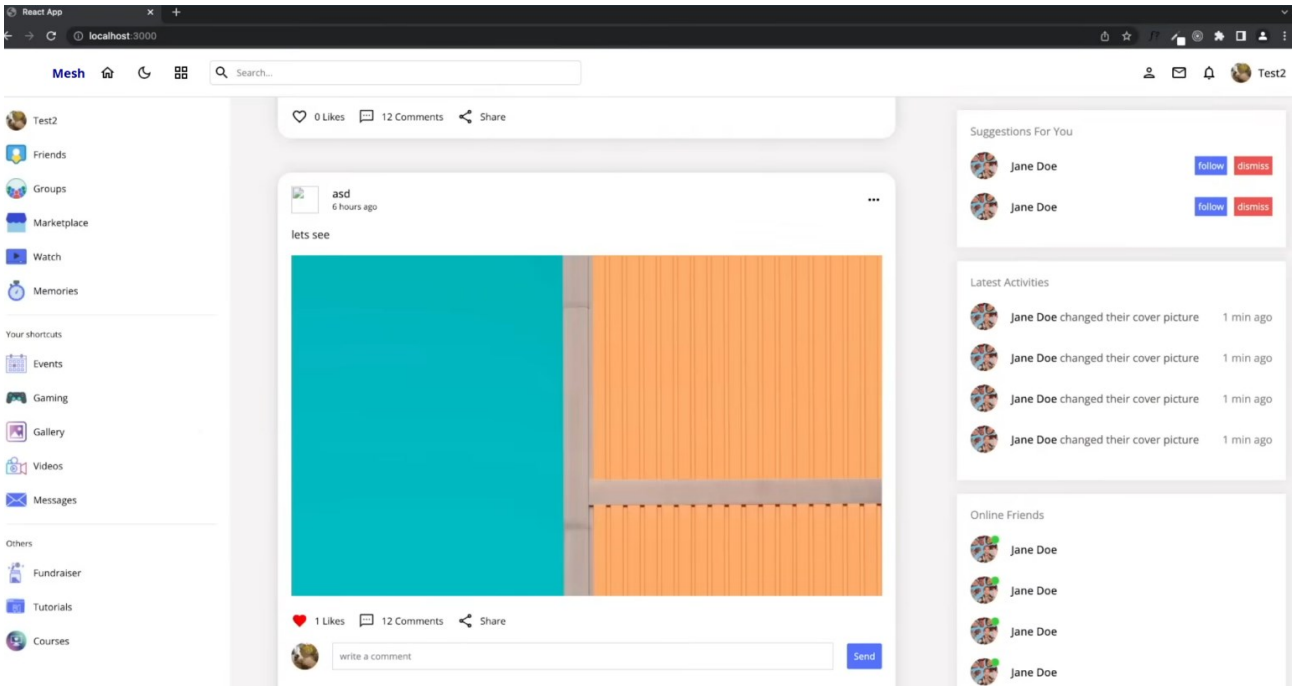

Рисунок 3.2 – Головна сторінка веб сайту

На рисунку 3.2 продемонстровано реалізацію головної сторінки веб-сайту. Вона містить основні компоненти, згадані вище: навігаційне меню, стрічку новин, а також інтерактивні елементи, як-от лайки, коментарі та список друзів. Структура сторінки є адаптивною та забезпечує зручне користування як на настільних, так і на мобільних пристроях.

3.1.4 Навігація та адаптивність дизайну

Навігація в застосунку здійснюється через React Router, що дозволяє переключатися між сторінками без необхідності перезавантаження сторінки. Це дає можливість створити швидкий і безперервний досвід користувача. Адаптивність інтерфейсу досягається завдяки використанню CSS Grid, Flexbox

та медіазапитів, що дозволяє ефективно оптимізувати дизайн для різних пристроїв.

Проект підтримує не лише базову адаптивність, а й специфічні адаптивні режими для різних розмірів екрану, що робить застосунок зручним для користувачів з різними типами пристроїв.

3.2 Реалізація серверної частини на Node.js з використанням Express.js

Для серверної частини платформи "Mesh" було обрано **Node JS** завдяки його високу продуктивність, асинхронний підхід і зручність розробки серверних додатків. Для побудови серверу був використаний фреймворк Express JS, який дозволяє швидко розробляти RESTful API.

Архітектура серверної частини побудована таким чином, що дозволяє ефективно обробляти запити від клієнтської частини та здійснювати взаємодію з базою даних MySQL. Система маршрутизації Express JS дає змогу створювати різні маршрути для обробки запитів від користувачів (наприклад, реєстрація, авторизація, створення та редагування публікацій, коментарів тощо).

3.2.1 Архітектура серверного застосунку

Серверна частина побудована на базі популярної платформи Node.js з використанням фреймворку Express.js. Вибір цієї технології обумовлений високою швидкістю роботи Node.js і його здатністю ефективно обробляти великі обсяги одночасних запитів. Архітектура серверної частини організована за принципом Model-View-Controller (MVC), що забезпечує чітке розділення між обробкою запитів, взаємодією з базою даних та формуванням відповіді.

Цей підхід дозволяє створювати масштабовані та легко підтримувані програми, де зміни в одному з компонентів не впливають на інші, а логіка застосунку залишається чітко структурованою.

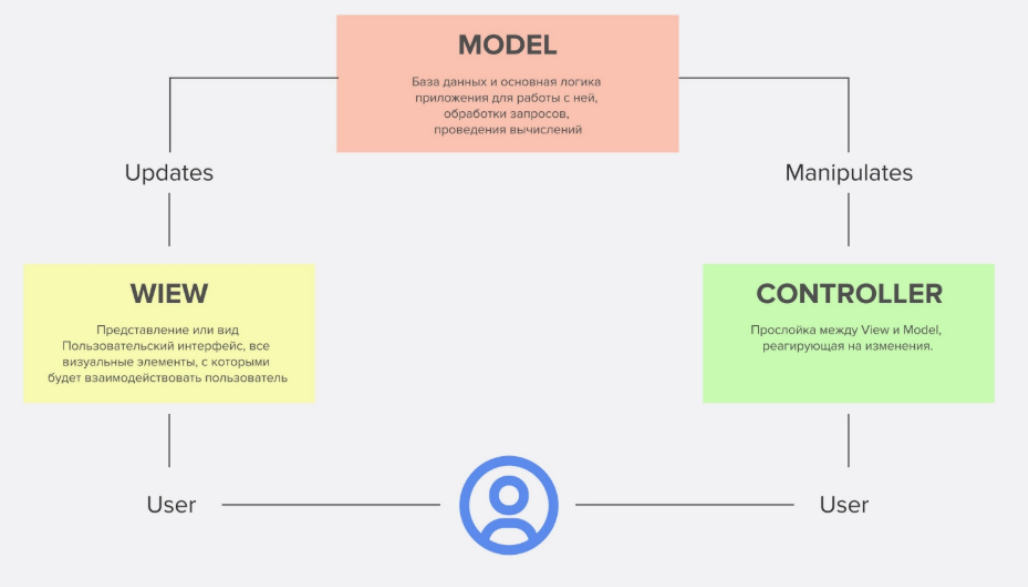

Рисунок 3.3 – Архітектура MVC

Рисунок 3.3 ілюструє взаємодію компонентів в архітектурі MVC. Користувач взаємодіє з *Представленням (View)*, а його дії обробляються *Контролером (Controller)*. Контролер, у свою чергу, маніпулює даними в *Моделі (Model)*. Після зміни даних Модель оновлює Представлення, завдяки чому користувач бачить актуальну інформацію. Цей цикл забезпечує організовану та структуровану роботу застосунку.

3.2.2 Створення REST API

Для забезпечення зв’язку між клієнтською та серверною частинами був реалізований REST API, що використовує стандартні HTTP методи (GET, POST, PUT, DELETE). Це дає змогу клієнтському застосунку отримувати, створювати, оновлювати й видаляти дані, забезпечуючи повну взаємодію з сервером через прості й ефективні запити.

Кожен маршрут API має чітко визначену функціональність, а параметри запитів перевіряються для уникнення помилок.

Розроблено REST API з використанням Express.js, що включає наступні маршрути:

– POST /auth/register – реєстрація користувача.
– POST /auth/login – вхід користувача.
– GET /posts – отримання публікацій.
– POST /posts – створення нової публікації.
– POST /messages – надсилання повідомлення.

API відповідає у форматі JSON та підтримує методи GET, POST, PUT, DELETE.

Рисунок 3.4 – Відповідь серверної частини в Insomnia

Ця фотографія показує скріншот з інтерфейсу Insomnia, інструменту для тестування API. На зображенні відображено успішний POST-запит до ендпоінту /api/auth/login на локальному сервері localhost:8800, про що свідчить статус відповіді 200 OK. Це демонструє, як відбувається тестування функціоналу входу користувача в розробленій REST API.

3.2.3 Реалізація основних маршрутів і логіки обробки запитів

Маршрути в серверній частині розроблені для обробки основних дій користувачів, таких як реєстрація, вхід, публікації та коментарі. Логіка обробки запитів забезпечує взаємодію з базою даних за допомогою ORM Sequelize, що дозволяє формувати запити до бази даних без необхідності написання сирих SQL запитів. Це спрощує код і підвищує безпеку застосунку.
Використовуються middleware для перевірки авторизації та обробки помилок.

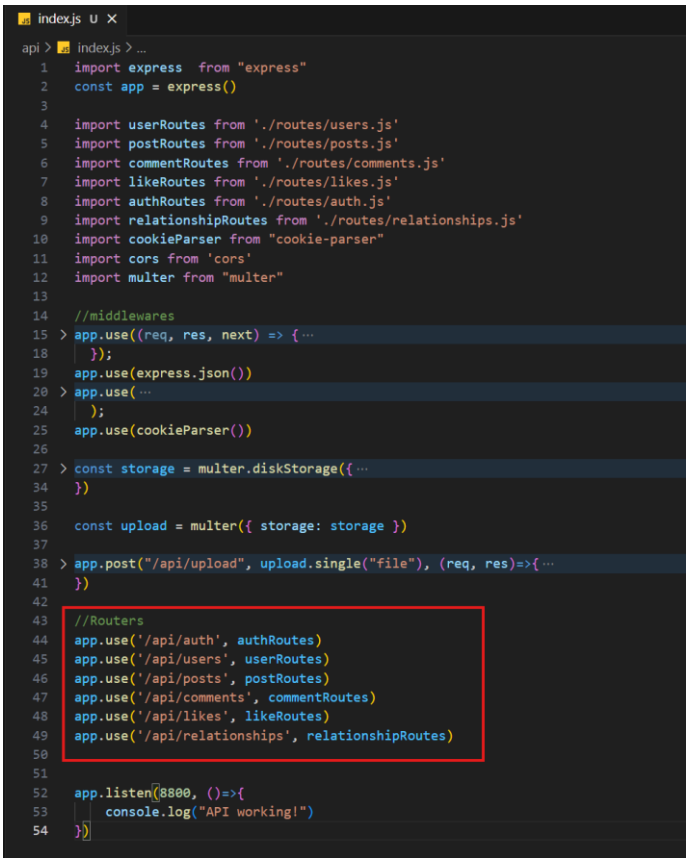

Рисунок 3.5 - Частина коду маршруту в Express.js

Це фото демонструє фрагмент коду з файлу index.js в Express.js. На ньому видно, як налаштовуються основні маршрути (routes) API для різних сутностей, таких як користувачі, публікації, коментарі, вподобання та відносини, а також застосування middleware для обробки запитів.

3.2.4 Взаємодія з клієнтською частиною

Сервер взаємодіє з клієнтом через HTTP-запити. Кожен запит перевіряється на наявність відповідного авторизаційного токена, що гарантує безпеку даних і захист від несанкціонованого доступу. Клієнтська частина застосунку передає запити на сервер, і сервер відповідає даними у форматі JSON, що дозволяє клієнту оновлювати інформацію без необхідності перезавантаження сторінки.

Процес взаємодії виглядає таким чином:

– Клієнт робить запит: Наприклад, запит на отримання публікацій користувача або відправку повідомлення.

– Сервер перевіряє JWT токен: За допомогою middleware на сервері, перевіряється валідність токену. Якщо токен правильний, обробляється запит.

– Відповідь від сервера: Сервер формує відповідь у форматі JSON і відправляє її клієнту.

Завдяки такій структурі запитів, комунікація між клієнтом і сервером є безпечною та ефективною.

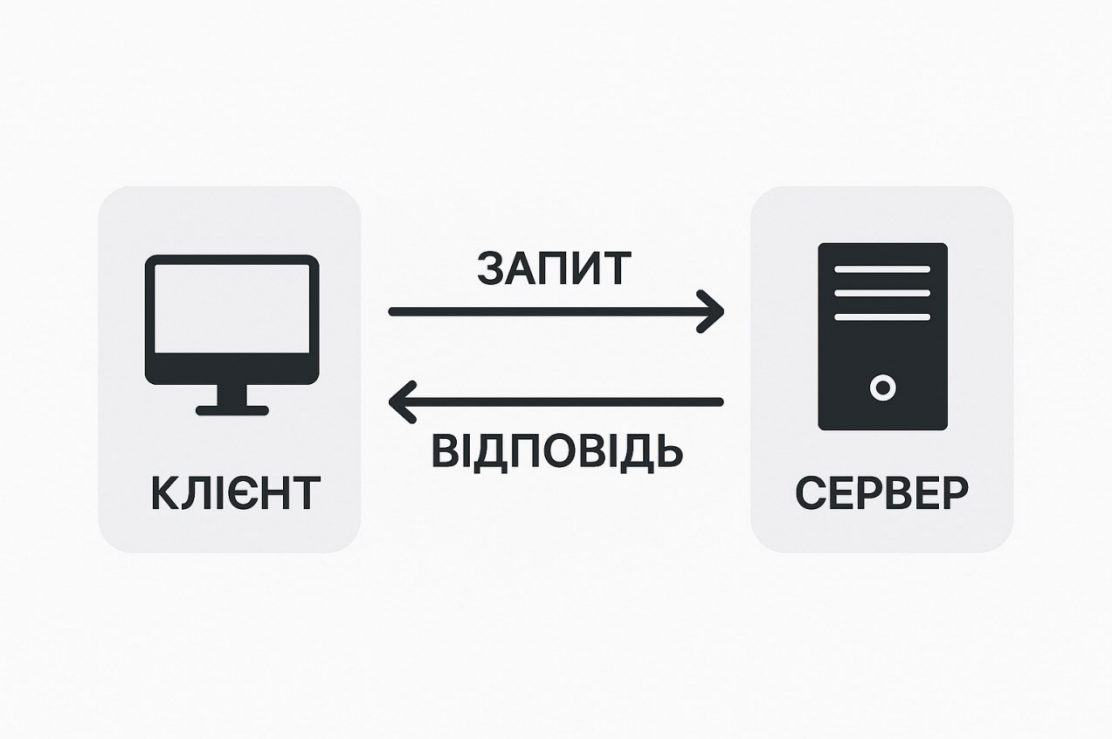

Рисунок 3.6 – Взаємодія між клієнтом і сервером

Ця фотографія ілюструє спрощену схему взаємодії між клієнтом та сервером. Вона показує, як клієнт надсилає "ЗАПИТ" до сервера, а сервер у відповідь надсилає "ВІДПОВІДЬ", демонструючи двосторонній зв'язок у мережевій архітектурі.

3.3 Взаємодія з базою даних MySQL

Для зберігання та обробки даних платформи було вибрано реляційну базу даних MySQL. Цей вибір обумовлений необхідністю зберігати структуровані дані, такі як профілі користувачів, пости, коментарі та інші елементи платформи. Крім того, MySQL забезпечує високу продуктивність при роботі з великими обсягами даних, що є важливим аспектом для веб-платформи, яка повинна обробляти запити від численних користувачів.

3.3.1 Структура бази даних

Для зберігання даних використовується реляційна база даних MySQL. Структура бази організована таким чином, що забезпечує максимальну ефективність і нормалізацію даних. Вона містить кілька основних таблиць, таких як users, posts, comments, messages, і friends. Кожна таблиця містить основні атрибути для зберігання даних, зокрема, особисту інформацію користувачів, публікації та повідомлення. Основні таблиці, які входять до структури бази даних:

– users – містить інформацію про користувачів, зокрема їхні імена, електронні адреси та паролі.

– posts – публікації користувачів, включаючи текст публікації, дату та час створення.

– comments – коментарі до публікацій користувачів.

– messages – повідомлення між користувачами.

– friends – зв'язки між користувачами, які вказують на їхні стосунки як друзі.

Структура таблиці users:

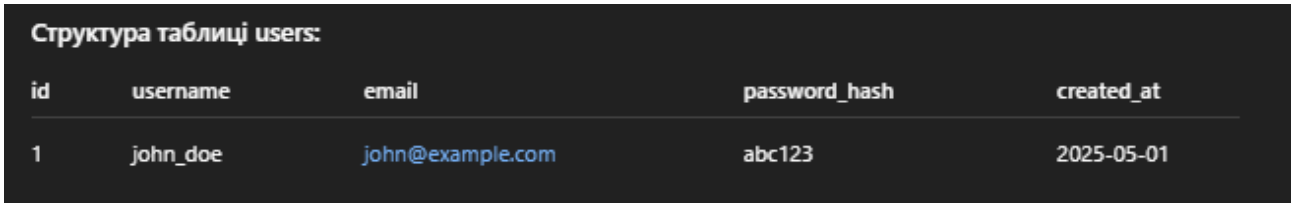

Рисунок 3.7 – Приклад структури таблиці в MySQL

На цій фотографії зображено приклад структури таблиці users у MySQL, що демонструє її поля (id, username, email, password_hash, created_at) та один тестовий запис для ілюстрації.

3.3.2 Зв’язки між таблицями

Між таблицями встановлено такі зв'язки:

– Один до багатьох: Один користувач може мати багато публікацій. Для цього в таблиці posts є зовнішній ключ user_id, який посилається на користувача в таблиці users.

– Багато до багатьох: Користувачі можуть бути друзями один з одним. Це реалізовано через таблицю friends, де зберігаються пари ID користувачів, що є друзями.

Приклад зв’язку між таблицями users та posts:

– Один користувач може створити багато публікацій, тому в таблиці posts є колонка user_id, яка є зовнішнім ключем для таблиці users.

3.3.3 Оптимізація запитів та індексація

Для покращення продуктивності роботи з базою даних застосовано індексацію найчастіше використовуваних полів, таких як user_id в таблиці posts, що прискорює виконання запитів на отримання публікацій.

Також були використані складені індекси для полів, які часто використовуються у запитах з фільтрацією за кількома параметрами (наприклад, дата створення публікації та статус).

3.4 Забезпечення авторизації, реєстрації та захисту даних

Одним з основних аспектів безпеки веб-платформи "Mesh" є правильна реалізація процесу авторизації та реєстрації користувачів. Під час реєстрації користувач вводить свій email, пароль, а також іншу особисту інформацію, яка зберігається в зашифрованому вигляді. Для цього використовувався алгоритм *bcrypt*, що дозволяє надійно захищати паролі.

Для авторизації користувачів було впроваджено механізм *JWT (JSON Web Token)*, який дозволяє створювати токени доступу для користувачів після

успішної авторизації. Токен зберігається на клієнтській стороні (як правило, в локальному сховищі браузера) і додається до кожного запиту на сервер для перевірки доступу до приватних маршрутів.

Захист даних реалізовано за допомогою сучасних методів безпеки, таких як:

– Шифрування паролів та інших чутливих даних.

– Перевірка на наявність SQL-ін'єкцій за допомогою параметризованих запитів.

– Вбудована захист від XSS та CSRF атак за допомогою додаткових бібліотек та middleware.

3.4.1 Загальгі вимоги безпеки з охорони праці для користувачів ПК Реєстрація та шифрування паролів

Для забезпечення безпеки зберігання паролів використовується bcrypt.js, що дозволяє шифрувати паролі перед їх збереженням у базі даних. Це забезпечує захист від крадіжки паролів, навіть у разі компрометації бази даних.

Лістинг 3.1 – Програмний код для шифрування пароля

```
const bcrypt = require('bcrypt');
async function hashPassword(password) {
  const salt = await bcrypt.genSalt(10);
  const hashedPassword = await bcrypt.hash(password, salt);
  return hashedPassword;
}
```

Після реєстрації користувача, пароль шифрується, і лише хеш зберігається в базі даних.

3.4.2 Авторизація з використанням JWT

JSON Web Token (JWT) дозволяє забезпечити безпечну авторизацію користувачів. Після успішного входу в систему, користувач отримує токен, який передається у заголовках для доступу до захищених ресурсів.

Лістинг 3.2 – Приклад створення JWT Token

```
const jwt = require('jsonwebtoken');

function generateToken(user) {
  return jwt.sign({ id: user.id }, 'secretKey', { expiresIn: '1h' });
}
```

Токен містить інформацію про користувача та має обмежений термін дії, що підвищує безпеку системи.

3.4.3 Захист від типових атак (XSS, CSRF, SQL-ін’єкції)

Для захисту від типових атак, таких як XSS, CSRF, SQL-ін'єкції, вживаються різні заходи безпеки, зокрема, використання бібліотек для захисту від XSS та CSRF атак, а також параметризація SQL-запитів для запобігання ін'єкціям.

Рисунок 3.8 – фото захисту від кібер атак

На цій фотографії зображено схему захисту від кібератак, яка фокусується на запобіганні трьом поширеним загрозам: XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery) та SQL-ін'єкціям. Центральний елемент "ЗАХИСТ ВІД АТАК" пов'язаний з кожним з цих векторів атак, підкреслюючи важливість комплексного підходу до безпеки.

3.4.4 Захист API та приватних маршрутів

API захищене за допомогою перевірки авторизаційних токенів для доступу до приватних маршрутів.

Лістинг 3.3 – Приклад middleware для перевірки токену

```
function authenticateToken(req, res, next) {
  const token = req.header('Authorization');
  if (!token) return res.status(403).send('Access denied');
```

```
  jwt.verify(token, 'secretKey', (err, user) => {
    if (err) return res.status(403).send('Access denied');
    req.user = user;
    next();
  });
}
```

Кожен запит, що потребує доступу до приватних даних, проходить перевірку на наявність правильного JWT-токена в заголовку запиту.

3.5 Методологія тестування веб-платформи

Тестування веб-платформи є невід'ємною частиною процесу розробки, оскільки дозволяє забезпечити правильну роботу всіх функціональних компонентів та уникнути помилок у коді. Для тестування використовувалися як ручні, так і автоматизовані методи.

Ручне тестування дозволяє перевіряти інтерфейс та взаємодію з користувачем, виявляючи проблеми в UI/UX та допомагаючи в налаштуванні інтерфейсу під різні пристрої. Автоматизовані тести допомагають перевірити серверну частину та базу даних, включаючи перевірку API-запитів, правильність виконання SQL-запитів, а також взаємодію між клієнтом і сервером.

3.5.1 Підходи до тестування: ручне та автоматизоване

У процесі тестування використовувалися як ручні, так і автоматизовані підходи. Ручне тестування допомогло виявити проблеми в інтерфейсі та функціональності, які могли б бути пропущені автоматизованими тестами. Автоматизовані тести використовувалися для перевірки функцій на серверній стороні, таких як авторизація, реєстрація та взаємодія з базою даних.

3.5.2 Інструменти тестування (Postman, Insomnia, React Testing Library)

Тестування є невід'ємною частиною розробки веб-платформи, оскільки воно дозволяє виявляти помилки, перевіряти функціональність та забезпечувати високу якість програмного продукту. Для тестування застосовувались різноманітні інструменти, що дозволяють перевіряти як клієнтську, так і серверну частини системи.

React Testing Library

Одним із основних інструментів для тестування клієнтської частини є React Testing Library (RTL). Ця бібліотека спрямована на те, щоб перевіряти компоненти таким чином, як це робить користувач, тобто через взаємодію з інтерфейсом користувача. Вона дозволяє перевіряти рендеринг компонентів, їх поведінку при взаємодії та коректність відображення UI-елементів.

Jest

Для написання юніт-тестів на сервері та клієнтській частині використовувався Insomnia, популярний тестовий фреймворк для JavaScript. Insomnia дозволяє виконувати як одиничні тести, так і інтеграційні, а також має підтримку асинхронного тестування, що є необхідним для перевірки API та серверної логіки.

Postman

Для тестування серверної частини використовувався Postman, який дозволяє перевіряти REST API. За допомогою цього інструменту можна створювати запити (GET, POST, PUT, DELETE) до серверних маршрутів, перевіряти правильність відповіді, статуси HTTP, а також перевіряти формат даних, що повертаються.

На зображені буде прилад застосування Insomnia.

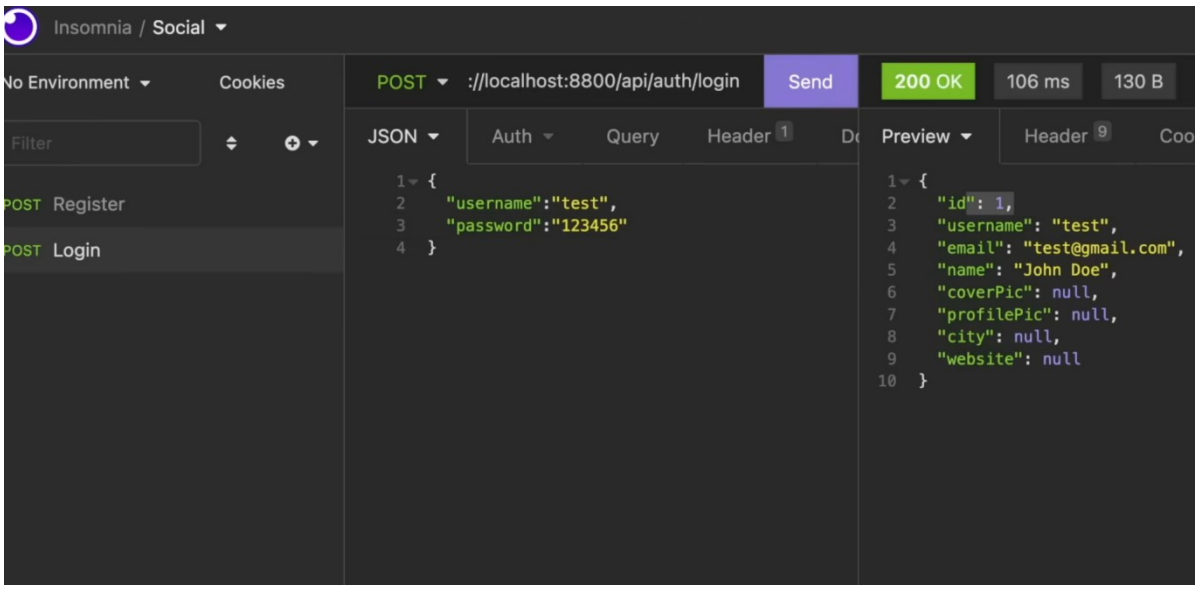

Рисунок 3.9 – Тестування за допомогою Insomnia

На фотографії представлений скріншот з програми Insomnia, де демонструється тестування API: видно POST-запит на /api/auth/login з даними для входу та успішну відповідь від сервера зі статусом 200 OK і даними про користувача.

3.6 Висновок до третього розділу

У цьому розділі було детально розглянуто процес розробки веб-платформи "Mesh" на всіх основних етапах її реалізації. Була проаналізована клієнтська частина, побудована за допомогою React JS, що забезпечує швидкий, гнучкий та адаптивний інтерфейс користувача. Застосування компонентного підходу, організована структура коду, а також ефективна навігація значно поліпшили взаємодію користувача з платформою.

Серверна частина, реалізована з використанням Node.js та Express.js, забезпечує надійний обробник запитів, гнучке API та ефективну взаємодію з клієнтською частиною. REST API дозволяє здійснювати повноцінну комунікацію між клієнтом і сервером, включаючи операції над публікаціями, повідомленнями, автентифікацією тощо.

Взаємодія з реляційною базою даних MySQL за допомогою ORM Sequelize забезпечує збереження структурованих даних, реалізацію складних зв’язків між таблицями, а також дозволяє підтримувати продуктивність системи завдяки оптимізації запитів та індексації.

Окрему увагу було приділено реалізації авторизації, реєстрації та захисту персональних даних користувачів. Інтеграція JWT, шифрування паролів через bcrypt, а також захист від типових атак (XSS, CSRF, SQL-ін’єкції) дозволяють гарантувати високий рівень безпеки веб-платформи.

Завершальним етапом стала побудова системи тестування, яка охоплює як клієнтську, так і серверну частини застосунку. Використання інструментів Postman, Jest і React Testing Library дозволило перевірити функціональність системи, її стабільність і готовність до реального використання.

Загалом, розроблена веб-платформа "Mesh" відповідає сучасним вимогам до веб-технологій, поєднує гнучкість, масштабованість, безпеку та зручність у використанні, що створює передумови для її подальшого розвитку та впровадження в реальні проєкти.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Загальгі вимоги безпеки з охорони праці для користувачів ПК Загальгі вимоги безпеки з охорони праці для користувачів ПК

Техніка безпеки є важливою складовою охорони праці, яка розглядає як технічні так й організаційні методи гарантування безпеки праці. Основною метою заходів з техніки безпеки є профілактика травматизму, тобто запобігання нещасним випадкам на виробництві.

Робота з комп'ютером характеризується значною розумовою напругою і нервово-емоційним навантаженням операторів, високою напруженістю зорової роботи і достатньо великим навантаженням на м'язи рук при роботі з клавіатурою .

У процесі роботи з комп'ютером необхідно дотримувати правильний режим праці та відпочинку. В іншому випадку у персоналу наголошуються значна напруга зорового апарату з появою скарг на незадоволеність роботою, головні болі, дратівливість, порушення сну, втому і хворобливі відчуття в очах, в поясниці, в області шиї і руках.

Необхідно дотримуватись певних вимог, а саме:

– система гігієнічних вимог – бо тривала робота з комп'ютером призводить до розладів стану здоров'я. Робота за комп'ютером з грубими порушенням гігієнічних норм і правил, призводить до підвищеного стомлення. Шкідливий вплив комп'ютерної системи на організм людини є комплексним. Обладнання робочого місця впливає на органи опорно-рухової системи;

– вимоги до робочого місця, оскільки, кідливими можуть бути всі частини комп’ютера (не тільки монітор). Процесор генерує промені, що поширюються у просторі у вигляді електромагнітних хвиль, часто несучи дезінформацію електромагнітного поля людини. Через нагрівання материнської плати і корпусу відбувається де іонізація повітря та виділення в навколишнього середовища шкідливих речовин. Саме тому необхідно провітрювати середовище, де

встановлене комп’ютерне обладнання. Робочі місця слід розташовувати так, щоб уникнути попадання в очі прямого природного або штучного світла. Джерела освітлення рекомендується розташовувати з обох боків екрану паралельно напрямку погляду. Робочі місця мають бути розташовані між собою на відстані не менше 1,5 м, на відстані не менше 1,5 м від стіни з вікнами, від інших стін на відстані 1м. Відносно вікон робоче місце доцільно розташовувати таким чином, щоб природне світло падало на нього збоку (переважно зліва);

– вимоги до розташування монітору, бо він виділяє небезпечні випромінювання і концентрація їх дії залежить від того, як монітор розташований до користувача.

Передня сторона монітора має захисне покриття, а от задня стінка і бічні поверхні не захищені.

Монітор встановлюється прямо перед користувачем і не вимагає повороту голови або корпусу тіла (на рисунку 4.1 зображено розташування монітора відносно користувача), а також робочий стіл і посадочне місце має висоту, щоб рівень очей користувача знаходився трохи вище центру монітора.

Рисунок 4.1 – Розташування монітора відносно користувача

Клавіатура розташовується на такій висоті, щоб пальці рук розташовувалися на ній вільно, без напруги, а кут між плечем і передпліччям становив 100 ° - 110 °.

Для роботи рекомендується використовувати спеціальні комп'ютерні столи, що мають висувні полички для клавіатури. Дотримання техніки безпеки

при розробці веб-сайту є важливим аспектом, оскільки правильне положення за робочим столом, освітлення та дотримання правильного режиму та відпочинку є головним фактором, що впливає на стан здоров’я розробника.

4.2 Критичні стани людини

До критичних станів відносять непритомність, шок, стенокардію, гіпертонічний криз. В даному розділі розглянути критичні стани, які можуть виникнути в розробників/користувачів ПК при тривалій і стресовій роботі за комп’ютером.

Критичні стани:

1. Непритомність (зімління) - найбільш поширений і легкий прояв гострої судинної недостатності внаслідок раптового короткочасного малокрів`я головного мозку.

Виникає при крововтратах, травмах, при хвилюванні, духоті, голодуванні, перевтомленні, захворюванні серцево-судинної системи.

Перша допомога:

– горизонтальне положення, доступ свіжого повітря;

– звільнення від тісного одягу (комір, ремінь - послабити), підняти ноги вище тулуба;

– скропити холодною водою, дати понюхати ватку, змочену нашатирним спиртом; напоїти гарячим чаєм, кавою, розтерти руки, ноги, дати грілку.

2. Шок – це комплекс грізних симптомів, які супроводжуються різким порушенням нервової регуляції життєво важливих функцій органів і систем. При шоку передусім страждає ЦНС.

Перша допомога:

– усунути джерело патологічної дії на потерпілого,

– покласти, упевнитись у прохідності дихальних шляхів,

– зупинити кровотечу, ввести знеболювальне, іммобілізувати травмовані кінцівки.

3. Стенокардія - гостра ішемія міокарду, зумовлена погіршенням його кровопостачання з наступним швидким відновленням кровообігу в зоні ішемії.

Перша допомога:

– заспокоїти, забезпечити доступ свіжого повітря, сидяче положення;

– валідол або нітрогліцерин під язик, серцеві краплі;

– на груди, ділянку серця - гірчичники, для рук і ніг — гарячу ванну.

4. Гіпертонічний криз. У здорової людини в нормі артеріальний тиск = 120/80 мм.рт.ст. Максимально допустимий рівень АТ=140/90. Підвищення АТ понад максимальну норму називається гіпертонічною хворобою. Загострення гіпертонічної хвороби спричиняється тривалим хвилюванням, нервовим пере навантаженням і призводить до піднесення АТ до 170/95 - 200/120, що називається гіпертонічним кризом.

Перша допомога:

– хворого покласти і створити повний фізичний і психічний спокій;

– забезпечити доступ свіжого повітря, масаж шиї і потиличної ділянки;

– гірчичники на шию. потилицю і литкові м'язи;

– холодний компрес до голови;

– транспортування напівсидячи.

Розробка веб-сайту – це досить стресова і виснажлива робота. При постійній напрузі та в схвильованому стані, можуть виникати критичні стани і надання першої домедичної допомоги може бути життєво необхідним навиком при виникненні такого стану.

ВИСНОВКИ

У межах цієї кваліфікаційної роботи освітнього рівня «Бакалавр» було успішно реалізовано проєкт зі створення веб-платформи для соціальної мережі “Mesh”, що базується на сучасних веб-технологіях React JS для фронтенд-частини, Node.js для бекенд-логіки та MySQL як системи керування базами даних. Основною метою було спроєктувати та розробити зручну, функціональну й масштабовану систему, яка відповідатиме актуальним вимогам користувачів до сучасних соціальних мереж, забезпечуючи високу швидкодію, інтерактивність та безпеку.

У процесі розробки платформи були реалізовані основні функції, притаманні більшості соціальних мереж: реєстрація та авторизація користувачів, створення та редагування профілю, публікація контенту (текстових дописів, зображень), взаємодія між користувачами у вигляді вподобань, коментарів та підписок. Завдяки використанню React JS інтерфейс вийшов динамічним, інтуїтивно зрозумілим та адаптивним до різних розмірів екрану. Важливою перевагою цієї технології стало компонувальне програмування, що дало змогу спростити структуру проєкту та покращити підтримку й масштабування коду в майбутньому.

Node.js забезпечив ефективну роботу серверної частини, дозволяючи обробляти численні запити одночасно завдяки неблокуючій архітектурі. Це дало змогу досягти високої продуктивності системи без суттєвих витрат на ресурси. Серверна логіка була організована за принципами REST-архітектури, що забезпечує гнучкість, модульність та можливість інтеграції з іншими сервісами у подальшому.

MySQL було обрано як перевірену, надійну СКБД, що добре масштабується та підтримує складні реляційні запити. Було розроблено та оптимізовано структуру бази даних, яка дозволяє ефективно зберігати та обробляти інформацію про користувачів, публікації, взаємодії між ними.

Особливу увагу приділено забезпеченню цілісності даних та оптимізації запитів для підвищення швидкодії.

Окремо варто відзначити важливість аспектів безпеки, які були враховані під час розробки. Зокрема, реалізовано хешування паролів за допомогою бібліотеки bcrypt, використано JSON Web Tokens (JWT) для авторизації, а також вжито заходів для запобігання поширеним атакам, таким як SQL-ін’єкції та XSS.

У ході реалізації проєкту було здобуто практичний досвід роботи з повним стеком веб-розробки (Full Stack Development), організації архітектури веб-застосунку, роботи з API, базами даних, а також з управлінням станом на фронтенді. У результаті розроблена платформа є повноцінною базою для подальшого розвитку проєкту “Mesh” — додавання нових функцій, масштабування системи, впровадження мобільної версії тощо.

Підсумовуючи, можна стверджувати, що поставлені у вступі цілі та завдання дипломної роботи були досягнуті повною мірою. Розроблена система відповідає сучасним стандартам розробки веб-застосунків, має зрозумілу та гнучку архітектуру, є ефективною з технічної точки зору та зручною для кінцевого користувача. У майбутньому проєкт може бути розширений з урахуванням зворотного зв’язку від користувачів та новітніх тенденцій у сфері цифрових соціальних сервісів.

ПЕРЕЛІК ДЖЕРЕЛ

1 Banks A., Porcello E. Learning React: Functional Web Development with React and Redux. – 2nd ed. – O'Reilly Media, 2020. – 350 p.

2 Tilkov S., Vinoski S. Node.js: Using JavaScript to Build High-Performance Network Programs // IEEE Internet Computing. – 2010. – № 14(6). – С. 80–83.

3 Murach J. Murach's MySQL. – Mike Murach & Associates, 2019. – 630 p.

4 Grinberg M. Flask Web Development: Developing Web Applications with Python and Flask. – O'Reilly Media, 2018. – (використовувалось для порівняння бекенд-архітектур).

5 MDN Web Docs. React documentation – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide – Дата звернення: 10.04.2025.

6 MySQL Documentation. MySQL 8.0 Reference Manual – https://dev.mysql.com/doc/refman/8.0/en/ – Дата звернення: 12.04.2025.

7 JWT.io. Introduction to JSON Web Tokens – https://jwt.io/introduction/ – Дата звернення: 09.04.2025.

8 Leshchyshyn, Y., Scherbak, L., Nazarevych, O., Gotovych, V., Tymkiv, P., & Shymchuk, G. (2019, May). Multicomponent Model of the Heart Rate Variability Change-point. In 2019 IEEE XVth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH) (pp. 110-113). IEEE.

9 Lytvynenko, I., Lupenko, S., Nazarevych, O., Shymchuk, G., & Hotovych, V. (2021, September). Mathematical model of gas consumption process in the form of cyclic random process. In 2021 IEEE 16th International Conference on Computer Sciences and Information Technologies (CSIT) (Vol. 1, pp. 232-235). IEEE.

10 Kozlovskyi, V., Balanyuk, Y., Martyniuk, H., Nazarevych, O., Scherbak, L., & Shymchuk, G. (2022, April). Information Technology for Estimating City Gas Consumption During the Year. In 2022 International Conference on Smart Information Systems and Technologies (SIST) (pp. 1-4). IEEE.

11 Lytvynenko, I., Lupenko, S., Kunanets, N., Nazarevych, O., Shymchuk, G., & Hotovych, V. (2021, November). Simulation of gas consumption process based on the mathematical model in the form of cyclic random process considering the scale factors. In 1st International Workshop on Information Technologies: Theoretical and Applied Problems, ITTAP 2021.

12 Kunanets, N., Pasichnyk, V., Bodnarchuk, I., Martsenko, S., Matsiuk, O., Matsiuk, A., ... & Shymchuk, H. (2019). Information system for visual analyzer disease diagnostics. In CEUR Workshop Proceedings (pp. 43-56).

13 Lupenko, S., Lytvynenko, I., Nazarevych, O., Shymchuk, G., & Hotovych, V. (2021, December). Approach to gas consumption process forecasting on the basis of a mathematical model in the form of a random cyclic process. In Proceedings of the International Conference „Advanced applied energy and information technologies 2021", 2021 (pp. 213-219). TNTU, Zhytomyr «Publishing house „Book-Druk "» LLC.

14 Lytvynenko, I., Lupenko, S., Nazarevych, O., Shymchuk, H., & Hotovych, V. (2022). Additive mathematical model of gas consumption process. Вісник Тернопільського національного технічного університету, 104(4), 87-97.

15 Nazarevych, O., Leshchyshyn, Y., Lupenko, S., Hotovych, V., Shymchuk, G., & Shabliy, N. (2020, September). Method of Gas Consumption Change-point Detection Based on Seasonally Multicomponent Model. In 2020 10th International Conference on Advanced Computer Information Technologies (ACIT) (pp. 152-155). IEEE.

16 Palianytsia, Y., Lytvynenko, I., Menoub, A., Shymchuk, H., & Dubchak, A. (2024). Development of an algorithm for identification of damage types on the surface of sheet metal.

17 Nazarevych, O., Gotovych, V., & Shymchuk, G. (2016). Information Technology for Monitoring of Municipal Gas Consumption, Based on Additive Model and Correlated for Weather Factors. Journal of Information and Computing Science, 11(3), 180-187.

18 Shymchuk, G., Lytvynenko, I., Hromyak, R., Lytvynenko, S., & Hotovych, V. (2023). Gas Consumption Forecasting Using Machine Learning Methods and Taking Into Account Climatic Indicators. In CITI (pp. 156-163).

19 Leschyshyn, Y. Z., Nazarevych, O. B., Shymchuk, G. V., Revutskyi, E. A., & Shcherbak, L. M. (2016, September). The Methods of Change Point Detection and Statistical Estimating of Dynamic of the Noise Stochastic Signals Characteristics. In THE SEVENTH WORLD CONGRESS "AVIATION IN THE XXI-st CENTURY" Safety in Aviation and Space Technologies September 19-21, NATIONAL AVIATION UNIVERSITY. Kyiv: NAU.

20 Шимчук, Г. В., Маєвський, О. В., & Назаревич, О. Б. (2016). Конспект лекцій з дисципліни Комп'ютерна графіка для студентів освітнього рівня «бакалавр» спеціальності 125 «Кібербезпека».

21 Шимчук, Г. В., Маєвський, О. В., & Назаревич, О. Б. (2016). Конспект лекцій з дисципліни «Розподілені системи моніторингу та керування».

22 Шимчук, Г. В., Маєвський, О. В., Назаревич, О. Б., & Стадник, М. А. (2016). Конспект лекцій з дисципліни «Грід-системи та технології хмарних обчислень» для студентів освітніх рівнів «спеціаліст»,«магістр» 122 «Комп'ютерні науки та інформаційні технології».

23 ШИМЧУК, Г., ШЕВЧЕНКО, Н., ШВИРЛО, К., & ГАРМАТЮК, Н. (2025). СИСТЕМА ВІДНОВЛЕННЯ ДАНИХ У БЕЗДРОТОВИХ СЕНСОРНИХ МЕРЕЖАХ НА ОСНОВІ МАШИННОГО НАВЧАННЯ. Herald of Khmelnytskyi National University. Technical sciences, 353(3.2), 246-250.

24 Шимчук, Г. В., Маєвський, О. В., & Назаревич, О. Б. (2016). Методичні вказівки до самостійної роботи студентів та модульного контролю знань з дисципліни «Розподілені системи моніторингу та керування» для студентів освітнього рівня «бакалавр» спеціальності 125–«Кібербезпека».

25 Mardan A. Express.js Guide: The Comprehensive Book on Express.js. – Leanpub, 2014. – 236 с.

26 Duckett J. JavaScript and JQuery: Interactive Front-End Web Development. – Wiley, 2014. – 640 с.

27 Nielsen J., Loranger H. Prioritizing Web Usability. – New Riders, 2006. – 456 с.

28 Beighley L. Learning MySQL and MariaDB. – O’Reilly Media, 2015. – 608 с.

29 Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. – O’Reilly Media, 2020. – 350 с.

30 Cantelon M. Node.js in Action. – Manning Publications, 2017. – 360 с.

31 Freeman E., Robson E. Head First HTML and CSS. – O’Reilly Media, 2012. – 768 с.

32 Flanagan D. JavaScript: The Definitive Guide. – O'Reilly Media, 2020. – 706 с.

ДОДАТКИ

Додаток А

Програмний код лістингу index.js

```
import express  from "express"
const app = express()

import userRoutes from './routes/users.js'
import postRoutes from './routes/posts.js'
import commentRoutes from './routes/comments.js'
import likeRoutes from './routes/likes.js'
import authRoutes from './routes/auth.js'
import relationshipRoutes from './routes/relationships.js'
import cookieParser from "cookie-parser"
import cors from 'cors'
import multer from "multer"

//middlewares
app.use((req, res, next) => {
    res.header("Access-Control-Allow-Credentials", true);
    next();
  });
app.use(express.json())
app.use(
    cors({
      origin: "http://localhost:3000",
    })
  );
app.use(cookieParser())
```

```
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, '../client/public/upload')
  },
  filename: function (req, file, cb) {
    cb(null, Date.now() + file.originalname)
  }
})

const upload = multer({ storage: storage })

app.post("/api/upload", upload.single("file"), (req, res)=>{
  const file = req.file;
  res.status(200).json(file.filename)
})

//Routers
app.use('/api/auth', authRoutes)
app.use('/api/users', userRoutes)
app.use('/api/posts', postRoutes)
app.use('/api/comments', commentRoutes)
app.use('/api/likes', likeRoutes)
app.use('/api/relationships', relationshipRoutes)

app.listen(8800, ()=>{
   console.log("API working!")
})
```

Додаток Б

Програмний код лістингу connect.js

```
import mysql from 'mysql'

export const db = mysql.createConnection({
    host: "localhost",
    user:"root",
    password:"194$56%JuMManJiNO1",
    database: "social"
})
```

Додаток В

Програмний код лістингу router/auth.js

```
import express  from "express"
import { login, register, logout } from "../controllers/auth.js"

const router = express.Router()

router.post('/login', login)
router.post('/register', register)
router.post('/logout', logout)

export default router
```

Додаток Г

Програмний код лістингу router/posts.js

```
import express  from "express"
import { getPosts, addPost, deletePost } from "../controllers/post.js"

const router = express.Router()

router.get("/", getPosts)
router.post("/", addPost)
router.delete("/:id", deletePost);

export default router
```

Додаток Д

Програмний код лістингу router/comments.js

```
import express  from "express"
import { getComments, addComments } from "../controllers/comment.js"

const router = express.Router()

router.get('/', getComments)
router.post('/', addComments)

export default router
```

Додаток Ж

Програмний код лістингу router/likes.js

```
import express  from "express"
import { getLikes, addLikes, deleteLikes } from "../controllers/like.js"

const router = express.Router()

router.get('/', getLikes)
router.post('/', addLikes)
router.delete('/', deleteLikes)

export default router
```

Додаток К

Програмний код лістингу router/relationship.js

```
import express from "express";
import { getRelationships, addRelationship, deleteRelationship } from "../controllers/relationship.js";

const router = express.Router()

router.get("/", getRelationships)
router.post("/", addRelationship)
router.delete("/", deleteRelationship)


export default router
```

Додаток М

Програмний код лістингу router/user.js

```
import express  from "express"
import { getUser, updateUser } from "../controllers/user.js"

const router = express.Router()

router.get("/find/:userId", getUser)
router.put("/", updateUser)

export default router
```

Додаток Н

Програмний код лістингу controllers/auth.js

```
import { db } from "../connect.js";
import bcrypt from "bcryptjs";
import jwt from "jsonwebtoken";

export const register = (req, res) => {
  const q = "SELECT * FROM users WHERE username = ?";

  db.query(q, [req.body.username], (err, data) => {
    if (err) return res.status(500).json(err);
    if (data.length) return res.status(409).json("User already exists!");
    //CREATE A NEW USER
    //Hash the password
    const salt = bcrypt.genSaltSync(10);
    const hashedPassword = bcrypt.hashSync(req.body.password, salt);

    const q =
      "INSERT INTO users (`username`,`email`,`password`,`name`) VALUE (?)";

    const values = [
      req.body.username,
      req.body.email,
      hashedPassword,
      req.body.name,
    ];

    db.query(q, [values], (err, data) => {
```

```
 if (err) return res.status(500).json(err);
     return res.status(200).json("User has been created.");
   });
  });
};

export const login = (req, res) => {
  const q = "SELECT * FROM users WHERE username = ?";

  db.query(q, [req.body.username], (err, data) => {
    if (err) return res.status(500).json(err);
    if (data.length === 0) return res.status(404).json("User not found!");

    const checkPassword = bcrypt.compareSync(
      //Тут використовується функція bcrypt.compareSync для порівняння паролю, введеного користувачем, з захешованим паролем, який був збережений у базі даних.
      req.body.password,
      data[0].password
    );

    if (!checkPassword)
      return res.status(400).json("Wrong password or username!");

    const token = jwt.sign({ id: data[0].id }, "secretkey"); //Тут створюється JWT-токен з ідентифікатором користувача (id), який був отриманий з бази даних. Токен підписується за допомогою секретного ключа "secretkey".
```

const { password, ...others } = data[0];//Ця конструкція ES6 деструктуризує об'єкт data[0], вилучаючи з нього поле password та залишаючи всі інші поля, які потім будуть відправлені у відповіді.

```
   res
     .cookie("accessToken", token, {
      //У цій частині коду встановлюється кукі з ім'ям "accessToken", яка містить JWT-токен,
      //створений раніше. Відповідь містить статус 200 (OK) та дані користувача (за винятком паролю) у форматі JSON.
       httpOnly: true,
     })
     .status(200)
     .json(others);
  });
};

export const logout = (req, res) => {
  res.clearCookie("accessToken",{
    secure:true,
    sameSite:"none"
  }).status(200).json("User has been logged out.")
};
```

Додаток О

Програмний код лістингу controllers/comments.js

```
import { db } from "../connect.js";
import jwt from 'jsonwebtoken';
import moment from 'moment';
export const getComments = (req, res) => {
   const q = `SELECT c.*, u.id AS userId, name, profilePic FROM comments AS c
JOIN users AS u ON (u.id = c.userId)
   WHERE c.postId=? ORDER BY c.createdAt DESC`;

   db.query(q, [req.query.postId], (err, data)=>{
      if(err) return res.status(500).json(err);
      return res.status(200).json(data);
   })

}
export const addComments = (req, res)=>{
   const token = req.cookies.accessToken;
   if(!token) return res.status(401).json("Not logged in!");

   jwt.verify(token, "secretkey", (err, userInfo) => {
      if(err) return res.status(403).json("token is not valid!")

      const q = "INSERT INTO comments (`desc`, `createdAt`, `userId`, `postId`)
VALUES (?)";

      const values = [
         req.body.desc,
         moment(Date.now()).format("YYYY-MM-DD HH:mm:ss"),
         userInfo.id,
```

```
    req.body.postId
]

db.query(q, [values], (err, data)=>{
    if(err) return res.status(500).json(err);
    return res.status(200).json("Comment has been created");
})
```

Додаток П

Програмний код лістингу controllers/like.js

```
import { db } from "../connect.js";
import jwt from 'jsonwebtoken';

export const getLikes = (req, res) => {
    const q = "SELECT userId FROM likes WHERE postId = ?";

    db.query(q, [req.query.postId], (err, data)=>{
        if(err) return res.status(500).json(err);
        return res.status(200).json(data.map(like=>like.userId));
    })
}
export const addLikes = (req, res)=>{
    const token = req.cookies.accessToken;
    if(!token) return res.status(401).json("Not logged in!");
    jwt.verify(token, "secretkey", (err, userInfo) => {
        if(err) return res.status(403).json("token is not valid!")
        const q = "INSERT INTO likes (`userId`, `postId`) VALUE (?)";
        const values = [
            userInfo.id,
            req.body.postId
        ]

        db.query(q, [values], (err, data)=>{
            if(err) return res.status(500).json(err);
            return res.status(200).json("Post has been liked");
        }
```

```
        })
}

export const deleteLikes = (req, res)=>{
    const token = req.cookies.accessToken;
    if(!token) return res.status(401).json("Not logged in!");

    jwt.verify(token, "secretkey", (err, userInfo) => {
        if(err) return res.status(403).json("token is not valid!")

        const q = "DELETE FROM likes WHERE `userId` = ? AND `postId` = ?";

        db.query(q, [userInfo.id, req.query.postId], (err, data)=>{
            if(err) return res.status(500).json(err);
            return res.status(200).json("Post has been disliked");
        })
    })
}
```

Додаток Р

Програмний код лістингу controllers/post.js

```
import moment from 'moment';
import { db } from '../connect.js'
import jwt from 'jsonwebtoken';

export const getPosts = (req, res)=>{

    const userId = req.query.userId
    const token = req.cookies.accessToken;
    if(!token) return res.status(401).json("Not logged in!");

    jwt.verify(token, "secretkey", (err, userInfo) => {
        if(err) return res.status(403).json("token is not valid!")

        console.log(userId)

        const q = userId !== "undefined"
        ? `SELECT p.*, u.id AS userId, name, profilePic FROM posts AS p JOIN users
AS u ON (u.id = p.userId) WHERE p.userId = ? ORDER BY p.createdAt DESC`
        : `SELECT p.*, u.id AS userId, name, profilePic FROM posts AS p JOIN users
AS u ON (u.id = p.userId)
        LEFT JOIN relationship AS r ON (p.userId = r.followedUserId) WHERE
r.followerUserId = ? OR p.userId = ?
        ORDER BY p.createdAt DESC`;

        const values = userId !== "undefined" ? [userId] : [userInfo.id, userInfo.id]

        db.query(q, values, (err, data)=>{
            if(err) return res.status(500).json(err);
            return res.status(200).json(data);
```

```
        })
    })
}

export const addPost = (req, res)=>{
    const token = req.cookies.accessToken;

   if(!token) return res.status(401).json("Not logged in!");

  jwt.verify(token, "secretkey", (err, userInfo) => {
        if(err) return res.status(403).json("token is not valid!")

        const q = "INSERT INTO posts (`desc`, `img`, `createdAt`, `userId`) VALUES
(?)";

        const values = [
            req.body.desc,
            req.body.img,
            moment(Date.now()).format("YYYY-MM-DD HH:mm:ss"),
            userInfo.id
        ]

        db.query(q, [values], (err, data)=>{
            if(err) return res.status(500).json(err);
            return res.status(200).json("Post has been created");
        })
    })
}

export const deletePost = (req, res) => {
    const token = req.cookies.accessToken;
```

```
  if (!token) return res.status(401).json("Not logged in!");

  jwt.verify(token, "secretkey", (err, userInfo) => {
    if (err) return res.status(403).json("Token is not valid!");

    const q =
      "DELETE FROM posts WHERE `id`=? AND `userId` = ?";

    db.query(q, [req.params.id, userInfo.id], (err, data) => {
      if (err) return res.status(500).json(err);
      if(data.affectedRows>0) return res.status(200).json("Post has been deleted.");
      return res.status(403).json("You can delete only your post")
    });
  });
};
```

Додаток С

Програмний код лістингу controllers/relationship.js

```
import { db } from "../connect.js";
import jwt from "jsonwebtoken";

export const getRelationships = (req,res)=>{
   const q = "SELECT followerUserId FROM relationship WHERE followedUserId
= ?";

   db.query(q, [req.query.followedUserId], (err, data) => {
     if (err) return res.status(500).json(err);
     return res.status(200).json(data.map(relationship=>relationship.followerUserId));
   });
}

export const addRelationship = (req, res) => {
  const token = req.cookies.accessToken;
  if (!token) return res.status(401).json("Not logged in!");

  jwt.verify(token, "secretkey", (err, userInfo) => {
    if (err) return res.status(403).json("Token is not valid!");

    const q = "INSERT INTO relationship (`followerUserId`,`followedUserId`)
VALUES (?)";
    const values = [
      userInfo.id,
      req.body.userId
    ];

    db.query(q, [values], (err, data) => {
      if (err) return res.status(500).json(err);
```

```
      return res.status(200).json("Following");
    });
  });
};

export const deleteRelationship = (req, res) => {
const token = req.cookies.accessToken;
  if (!token) return res.status(401).json("Not logged in!");

  jwt.verify(token, "secretkey", (err, userInfo) => {
    if (err) return res.status(403).json("Token is not valid!");

    const q = "DELETE FROM relationship WHERE `followerUserId` = ? AND
`followedUserId` = ?";

    db.query(q, [userInfo.id, req.query.userId], (err, data) => {
      if (err) return res.status(500).json(err);
      return res.status(200).json("Unfollow");
    });
  });
};
```

Додаток Т

Програмний код лістингу controllers/user.js

```
import { db } from '../connect.js'
import jwt from 'jsonwebtoken';

export const getUser = (req,res)=> {
  const userId = req.params.userId;
  const q = "SELECT * FROM users WHERE id=?";

  db.query(q, [userId], (err, data) => {
    if (err) return res.status(500).json(err);
    const { password, ...info } = data[0];
    return res.json(info);
  });
}

export const updateUser = (req, res) => {
  const token = req.cookies.accessToken;
  if (!token) return res.status(401).json("Not authenticated!");

  jwt.verify(token, "secretkey", (err, userInfo) => {
    if (err) return res.status(403).json("Token is not valid!");

    const q =
      "UPDATE users SET `name`=?,`city`=?,`website`=?,`profilePic`=?,`coverPic`=?
WHERE id=? ";

    db.query(
```

```
q,
    [
      req.body.name,
      req.body.city,
      req.body.website,
      req.body.coverPic,
      req.body.profilePic,
      userInfo.id,
    ],
    (err, data) => {
      if (err) res.status(500).json(err);
      if (data.affectedRows > 0) return res.json("Updated!");
      return res.status(403).json("You can update only your post!");
    }
  );
 });
};
```