МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота бакалавра

ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СОЦІАЛЬНОЇ МЕРЕЖІ

Здобувач освіти гр. ІН – 82 Сергій СЧАСТЛІВЦЕВ

Науковий керівник,
кандидат фізико-математичних наук,
асистент кафедри комп'ютерних наук Ольга ШУТИЛЄВА

Завідувач кафедри
доктор технічних наук, професор Анатолій ДОВБИШ

СУМИ 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Затверджую_____
Зав. кафедрою Довбиш А.С.
"_____" _____2022 р.

ЗАВДАННЯ

до кваліфікаційної роботи

здобувача вищої освіти четвертого курсу, групи ІН-82 спеціальності «122 – Комп'ютерні науки» денної форми навчання Счастлівцева Сергія Миколайовича

Тема: «ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СОЦІАЛЬНОЇ МЕРЕЖІ»

Затверджена наказом по СумДУ
№_____ від _____2022 р.

Зміст пояснювальної записки: 1) літературний огляд за обраною тематикою роботи; 2) постановка завдання для розробки; 3) практична реалізація.

Дата видачі завдання «_____»_____ 2022 р.
Керівник роботи _____Ольга ШУТИЛЄВА
Завдання прийняв до виконання _____ Сергій СЧАСТЛІВЦЕВ

РЕФЕРАТ

Записка: 60 стор., 14 рис., 1 додаток, 22 джерел.

Об’єкт дослідження – технологія розробки веб-додатків.

Мета роботи – огляд сучасних методів для розробки веб-додатків. Створення додатку, який матиме функціонал соціальної мережі, такі як додавання постів, їх оцінка, підписка на інших користувачів, авторизація та реєстрація у додатку, повна клієнт-серверна взаємодія.

Результати – розроблено веб-додаток, за допомогою React, NodeJS, Koa.js, MongoDB. Додаток виконує всі поставлені перед ним задачі і коректно відображається на будь-яких пристроях.

ВЕБ-ДОДАТОК, REACT, NODEJS, KOA.JS, MONGODB

ЗМІСТ

ВСТУП

У сучасному світі соціальні мережі не є чимось новим та інноваційним. Люди вже давно звикли, що можуть у будь-який час написати будь-кому та дізнатись інформацію особисто від людини, навіть не зустрічаючись з нею особисто. Багато компаній та звичайних людей почали використовувати соціальні мережі як спосіб для заробітку. Якщо аудиторія велика, з'являється можливість продавати рекламу, а своє ім'я використовувати як бренд. Дехто, починає відкривати онлайн-магазини на платформі мережі. Це все має багато позитивних сторін, якими люди і користуються.

Актуальність роботи. Звісно, з одного боку це зовсім не інноваційна розробка, але зможе досить наглядно показати усі навички, які були здобуті під час навчання. Це не тільки сама розробка, це і пошук матеріалів, порівняння, збір статистики, окреслення проблематики та аналіз зібраних матеріалів.

Також, звісно немає можливості розробити мережу, яка буде конкурувати з провідними мережами, але написати свою реалізацію функціоналу, яким займаються одні з найкращих спеціалістів у світі та побачити різницю багато чого варте.

Мета роботи – розробити власний аналог соціальної мережі, проаналізувавши найпопулярніші розробки у цьому сегменті.

Взяти з кожного прикладу, який зараз є у вільному доступі для кожної людини, найкраще та реалізувати у своєму додатку.

Провести пошукову, аналітичну та порівняльну роботу, проаналізувати проблеми та плюси, які є у кожної мережі та спробувати не допустити подібного у своїй реалізації.

1. ЛІТЕРАТУРНИЙ ОГЛЯД ЗА ОБРАНОЮ ТЕМАТИКОЮ РОБОТИ

1.1 Аналіз принципів побудови та вимог, які частіше за все виставляються перед соціальною мережею

Призначення соціальної мережі полягає у тому, щоб дати людям можливість спілкуватися, обмінюватись даними, фото, відео в режимі реального часу. Власне, на основі цього і будуються вимоги.

Основний принцип – доступність додатку на усіх видах пристроїв. Людина повинна мати доступ у будь-який час і у будь-якому місце, де, звісно, є інтернет. Далі треба поставити вимогу, розробити інтерфейс, який буде зручно використовувати. І, звісно, також на усіх можливих пристроях.

Також, необхідним кроком є розроблення серверної частини та конфігурація бази даних таким чином, щоб вона могла зберігати велику кількість інформації про користувача та інформацію, яку користувач прив’язує до себе під час користування додатком. У нашому випадку, це буде стосуватись списку друзів, чатів, постів на головній сторінці.

Одним із важливих факторів успіху мережі є сама ідея, для чого вона була створена. Наприклад, у випадку з Twitter, була ціль створити проект, завдяки якому, люди можуть розміщувати у стислій формі свої міркування з того чи іншого приводу. Такі темі розподілені по розділам, часто прив’язані до подій у світі, але також дуже часто є місцем простого висловлення думок. Завдяки такому формату, Twitter нині став місцем, де можна знайти безліч висловів, які стали або скоро стануть «крилатими», тобто такими, які будуть на слуху серед людей.

1.2. Порівняння найпопулярніших соціальних мереж

Нині найпопулярнішими соціальними мережами у світі є Instagram, Facebook, Twitter, TikTok. Тому, саме їх було взято на розгляд для прикладу.

Необхідно зробити порівняльну характеристику і по кожній мережі винести короткий висновок.

Таблиця 1.1 – порівняння наявності функціоналу мережі

Функція/ соціальна мережа	Instagram	Twitter	Facebook	TikTok
Фото	+	+	+	-
Відео	+	+	+	+
Додавання постів	+	+	+	-
Чат	+	+	+	+
Аудіо повідомлення	+	-	+	-
Групи	-	-	+	-
Стрічка новин	+	+	+	+
Підписки	+	+	+	+
Друзі	-	-	+	+
Відео чати	+	-	+	-
Опитування	-	-	+	-

Отже, з таблиці 1.1 видно, що кожна мережа створювалась під окремі потреби і в деяких відсутній ряд функцій, які частіше за все повинні бути. Але це не заважає мережі бути популярною, основною причиною є те, що аналогів, які можуть скласти конкуренцію мало, а якщо вони і є, про них нікому не відомо [1].

1.3 Переваги та недоліки аналогів

Instagram – мережа, яка має майже повний спектр функцій, який зазвичай потрібен. Деякі з них мають своєрідну реалізацію, але є повністю

робочими і не призводять до проблем. Спочатку, Instagram створювався як мобільний додаток, але з часом була також додана версія під desktop. Саме ця версія і є проблемою мережі. У мобільної версії проблем майже немає, розробники якісно дбають про те, що мережа яку використовують мільйони користувачів працювала без перебоїв. Основна проблема комп'ютерної версії Instagram полягає у тому, що частина функції мобільної версії не було додана. Гарним прикладом слугують зникаючі повідомлення. Суть полягає у тому, що у особистому чаті можна відправити фото і коли співрозмовник його подивиться, фото знищиться. При спробі переглянути таке повідомлення на desktop версії, користувач побачить повідомлення про те, що дана функція працює лише на мобільній версії.

Отже, висновком може слугувати те, що необхідно розробити функціонал, який буде однаково працювати як на desktop, так і на mobile версіях додатку.

Twitter – додаток, який має урізаний функціонал порівняно з іншими мережами. Але водночас це і є основною темою додатку. Суть полягає у тому, що кожен може зареєструватись та у стислій формі висловити свою думку, про ту чи іншу подію. Twitter є однією з найпопулярніших мереж, але, на жаль, не отримало гідну оцінку серед користувачів України. Можливо через те, що має в основному англомовну аудиторію. Можливо причина в тому, що більшості вистачає мереж, які вони використовують і зараз, тому немає сенсу долучатися ще до одної, можливо комусь не подобається формат, який пропонує Twitter. Причин може бути багато, але все ж варто зазначити, що ця мережа є унікальною у своєму роді.

Висновком може слугувати як і те, що треба створити унікальну ідею, так і те, що необхідно робити кроки для інтегрування мережі серед різних країн.

Facebook – популярна соціальна мережа, яку підтримує компанія Meta. Вона поєднала у собі увесь функціонал, який має бути у еталонній соціальній мережі. Більше того, за роки свого існування вона стала мережею на яку стали

рівнятись у питаннях розробки. Але і тут не обійшлося без недоліків. Інтерфейс програми з першого погляду дуже нагромаджений, важко зрозуміти, для чого така кількість функціоналу, яким більшість людей не користується. Гарною ідеєю була б можливість налаштування відображення для окремого користувача. Зараз, Facebook популярний у більш старшої аудиторію, у крайньому разі, це правдиво для України.

До висновків винесемо те, що сторінка не повинна лякати користувача кількістю функціоналу, потрібно додати лише основні функції і зробити додаток максимально доступним.

TikTok – соціальна мережа, яка стрімко набрала популярність завдяки новому на ринку форматі коротких відео. Багато людей використовують його для запису коротких, розважливих відео, дехто створив навчальний майданчик на платформі TikTok. Він використовується для різних цілей, і, безумовно, виправдано займає одну з лідируючих позицій у світі. Серед мінусів, які я знайшов у додатку – недороблена система чатів та неінтуїтивний UI при спробі поділитись відео. Стосовно чатів, все досить просто, немає можливості записати голосове повідомлення, відправити фото. Також проблемою є те, що у більшості мереж, коли користувач починає писати повідомлення на мобільній версії, перша літера автоматично переходить у великий формат, у TikTok, цього немає. Стосовно UI при репості відео, це те, що при відправленні відео у іншу мережу, часто буває таке, що відео просто завантажується на телефон і лише тоді відправляється. Це незручно і з першого погляду незрозуміло, що буде відбуватись саме так.

Висновком є те, що у соціальній мережі, однією із найголовніших функцій є чат, тому варто гарно пропрацювати всі аспекти спілкування користувачів.

1.4 Інструменти, які можуть бути використані для розробки

У сучасній сфері розробки існує декілька способів розробки веб-додатків. Якщо брати до уваги клієнтську частину, то за останній час з'явилось безліч інструментів, який допомагають конвертувати будь-яку мову програмування у JavaScript – єдину мову, яку використовує браузер. Серверну частину можна написати на будь-яких із нині популярних інструментів. Комбінацій дуже багато, тому варто виділити декілька найпопулярніших.

Для клієнтської частини є три найголовніші технології, які із року в рік, залишаються на лідируючих позиціях – Angular, Vue, React. Перші дві технології є фреймворками, React, у свою чергу, лише бібліотека для реалізації UI. Різниця полягає у тому, що Vue та Angular вже несуть із собою всі необхідні для роботи інструменти, маршрутизація, типізація у випадку з Angular та інше. У випадку з React необхідно за допомогою пакетних менеджерів додавати необхідні модулі у додаток. Це є і плюсом, і мінусом одночасно. Гарна сторона полягає у тому, що частіше за все, готова збірка проекту має менший розмір, ніж у випадку фреймворків. Але за це доводиться платити зручністю розробки.

Для серверної частини є набагато більше варіантів. Можна використовувати добре всім відомі Java або C# для розробки та реляційні бази даних, які, на даний момент, використовують у більшості додатків.

Але є варіант піти по шляху створення мережі однією мовою програмування – JavaScript. Тобто, для розробки серверної частини можна використати Node.js, Koa та MongoDB. Таким чином, ми отримаємо обидві частини додатку, які будуть написані на одній мові програмування, тому для інтеграції нових розробників у проект піде менше часу.

1.4 Постановка задачі для розробки

Для успішного виконання поставленої мети кваліфікаційної роботи, необхідно реалізувати наступні задачі:

1. Розглянути найпопулярніші інструменти для розробки веб-додатків у сучасному світі IT та підібрати найбільш популярні та зручні інструменти, які водночас підійдуть для реалізації проекту;

2. Виконати проектування маршрутів користувача по додатку;

3. Розробити серверну частину додатку, яка буде відповідати усім вимогам, які ставляться перед соціальною мережею;

4. Спроектувати базу даних, для широкого кола користувачів. Також необхідно передбачити усі зв’язки одного користувача з інтерфейсом мережі та іншими користувачами;

5. Реалізувати наступний функціонал: реєстрація, авторизація, додавання постів, додавання користувачів у список друзів, система оцінки постів, фото, дописів.

6. Вдало з’єднати клієнтську та серверну частину, щоб вони працювали разом для виконання усіх операцій.

7. Реалізувати клієнт-серверну взаємодію у повному обсязі – CRUD.

8. Виконати тестування додатку на комп’ютерних, планшетних, мобільних пристроях.

9. Інтерфейс додатку має бути простим і зрозумілим, потрібно уникнути нагромадження контенту, а всі функції зробили легкодоступними та простими в користуванні;

10. При вдалому виконанні усіх задач, завантажити додаток на хостинг, щоб мати доступ через посилання.

2. МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

2.1 Стек технологій

MongoDB, Koa.js, React.js, Node.js. – основні технології, які були обрані для розробки мережі. Основною причиною для обрання саме такого набору інструментів стало те, що він дозволяє досить легко і у короткий строк розгорнути повноцінно функціонуючу систему, яка буде виконувати всі поставлені перед нею задачі.

Звісно, враховуючи специфічність технологій, разом із ними іде ряд додаткових бібліотек, які необхідні для повноцінної та зручної розробки [3]. У таблиці 2.1 представлені додаткові інструменти, які будуть використані при розробці клієнтської частини додатку. Bootstrap та Axios, звичайно, мають аналоги, але вибір було зроблено саме у їх сторону через зручність використання та чудову інтеграцію з React.

Таблиця 2.1 – Додаткові бібліотеки для розробки клієнтської частини

Bootsrap [4]	CSS-бібліотека, яка дозволить створити гнучкий інтерфейс, не витрачаючи всю увагу на стилі
Axios [5]	Бібліотека, яка полегшує написання запитів на сервер, підтримує усю повноту CRUD-операцій
React-router-dom [6]	Інструмент, який дозволяє створити SPA-додаток, за допомогою React
Font-awesome	Бібліотека іконок для використання у додатку
Jquery [7]	Необхідний плагін для роботи з Bootstrap
Js-md5 [8]	Проста хеш-функція для JavaScript, підтримує кодування UTF-8.
Jwt-decode [9]	Бібліотека для розшифрування jwt токенів
Popper.js [10]	Бібліотека для додавання спливаючих вікон додатку
React-js-pagination [11]	Бібліотека для простої реалізації пагінації
Redux	Інструмент для керування даними у додатку

У таблиці 2.2 представлені бібліотеки, що будуть використані при розробці серверної частини. Деякі з них замінюють функціонал, що представлений у Node.js з «коробки», але у свою чергу не сильно завантажують збірку проекту, але досить сильно полегшують розробку.

Таблиця 2.2 – Додаткові бібліотеки для розробки серверної частини

Passport-jwt[12]	Стратегія для аутентифікації користувача з jsonwebtoken
Bctypt [13]	Бібліотека, що дозволяє зберігати дані у БД у захешованому вигляді
Dotenv [14]	Необхідний для використання змінних з process.env
Mongoose [15]	Бібліотека, необхідна для коректної роботи Node.js/Express з MongoDB
Nodemon [16]	Бібліотека, що дозволяє запускати сервер у режимі розробки та автоматично перезапускає його після внесення будь-яких змін.
jsonwebtoken[17]	Бібліотека для створення токенів на серверній частині
koa[18]	Фреймворк для node.js, який спрощує роботу з ним
Mongoose-private-paths[19]	Для створення закритих шляхів у базі даних

2.2 Клієнтська частина веб-додатку

При розробці клієнтської частини необхідно звернути увагу на декілька необхідних елементів. По-перше, побудова структури проекту таким чином, щоб у майбутньому можна було легко ввести зміни та підтримувати проект.

Це досягається за рахунок того, що необхідно відділити бізнес логіку, стилі, компоненти відображення інформації, запити на сервер. Система import/export модулів, яка була додана у JavaScript з синтаксисом ES6 дозволяє створити чудову файлову структура проекту, так користуватись модулями у інших файлах зручно. Таким чином, файли проекту не великі та легко підтримуються.

По-друге, це написання стилів для додатку. React має багато способів стилізації свої компонентів. Аналогами обраного Bootstrap є Tailwind.css, Bootstrap, Ant Design та інші. Також, є можливість писати стилі за допомогою стандартних можливостей CSS або використовуючи препроцесори, наприклад SCSS. І останній варіант, це підхід CSS-in-JS. Тобто стилі пишуться у JS-файлах у виді спеціальних конструкції, а потім завдяки Webpack компілюються у зрозумілий браузеру CSS. Цей варіант був відкинутий через навантаження непотрібними конструкціями файли компонентів, а варіант з препроцесорами та звичайним CSS вимагає багато часу для розробки.

По-третє, невід'ємною частиною сучасних веб-додатків є концепція SPA(singe-page-application). Вона полягає у тому, що при першому завантаженні сторінки ми отримуємо пустий HTML-файл, у якому є лише мета теги та кореневий елемент, куди і буде додано наш додаток. З цим файлом до користувача приходить доволі велика кількість JS-коду, який і відповідає за відображення елементів на сторінці. У React такий синтаксис додає JSX. Це конструкція, яка дозволяє зручно писати розмітку всередині JS-файлів. А за роботу у форматі SPA відповідає модуль react-router-dom [20].

2.3 Серверна частина веб-додатку

У серверній частині головне завдання забезпечити зв'язок клієнтської частини з даними, які знаходяться в БД та написати ряд endpoint, на які клієнт буде відправляти запити для коректної роботи згідно вимог до додатку. Наприклад, це потрібно для реєстрації, авторизації, написання повідомлень, додавання постів користувачем. Для кожної такої дії, буде відправлятись запит на сервер, а у самому endpoint буде описана послідовність дій сервера, при отриманні запити по цій адресі.

Важливим фактором є те, що у серверній частині при взаємодії з БД є можливість налаштувати моделі, які будуть використовуватись у самій MongoDB. Завдяки правильно налаштованим моделям є можливість знайти у

базі необхідний елемент або масив елементів та повернути його на клієнтську частину. Також, на серверній частині буде описано правила для політики безпеки CORS. Якщо цього не зробити, клієнт не зможе відправляти запити на сервер, CORS буде блокувати кожен із них. Туди додаються правила, які необхідні для авторизації користувача та роботи всередині додатку. Враховуючи те, що у додатку буде підтримуватись повний набір CRUD операцій – налаштування політики безпеки є обов’язковим елементром [21].

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Інформаційна модель

Для вирішення поставлених задач, необхідно візуалізувати головні задачі, які потрібно буде вирішити. На рисунку 3.1 схематично представлено водночас вимоги та приблизні дії, які зможе виконати користувач.

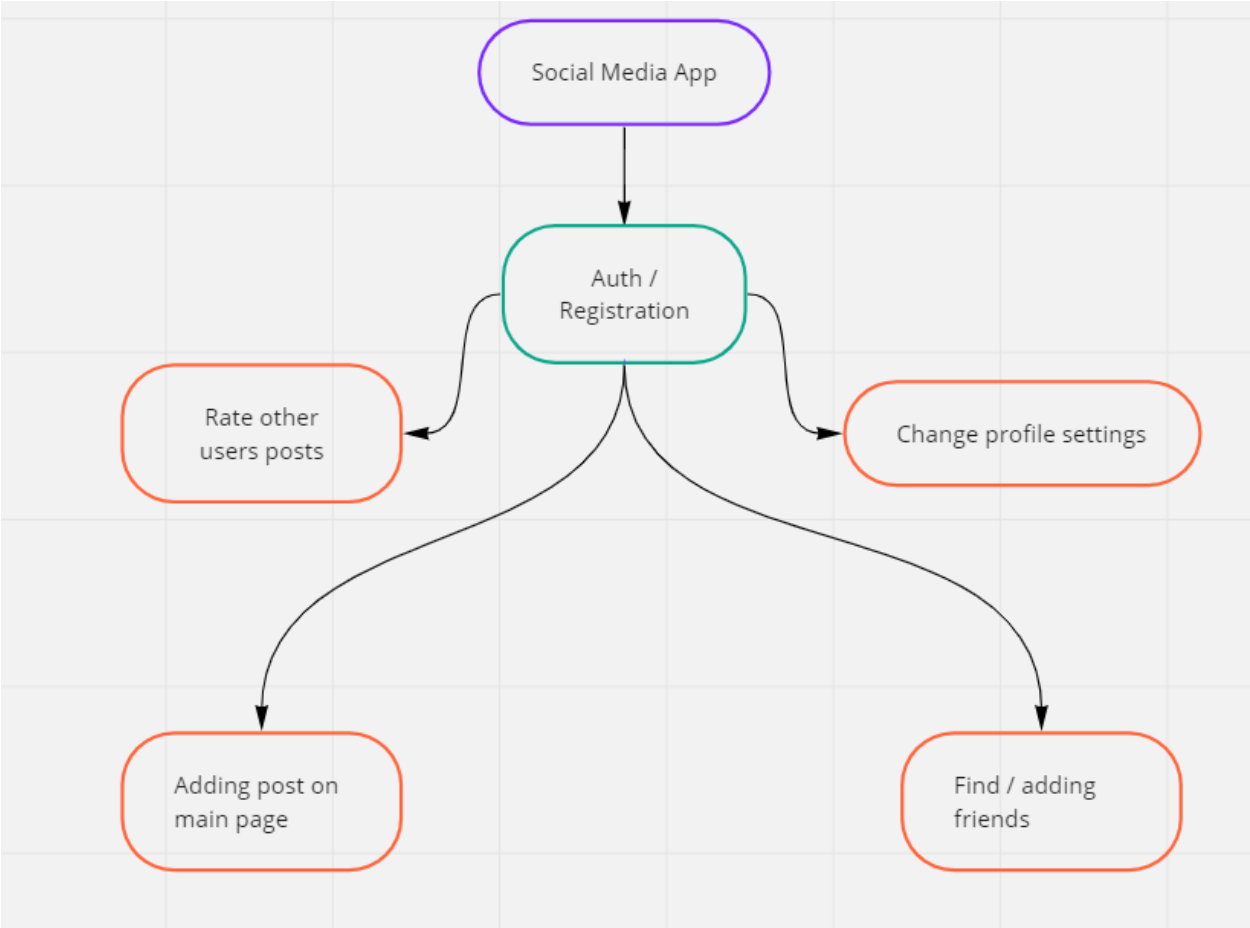

Рисунок 3.1 – можливості користувача у мережі

Варто зазначити, що усі дії у додатку, може виконати лише авторизований користувач. Таким чином, є можливість отримати дані про людину, для відображення їх для інших користувачів. На рисунку 3.1 такі дії позначені помаранчевим кольором. Етап авторизації у додатку – відповідно зеленим.

MongoDB – документ орієнтована база даних, інформація у ній представлена у вигляді JSON, що є дуже зручним. Для обробки кожної дії користувача, необхідно підготувати модель, за допомогою якої буду відстежуватись дані про нього.

У таблиці 3.1 представлено основні елементи, з якими буде взаємодіяти користувач. Кожен користувач має свої дані, у нашому випадку це ім’я, електронна пошта, пароль та дата, коли був створений акаунт.

Таблиця 3.1 – Поля моделі User для зберігання у MongoDB

name	Type: String, require: true
email	Type: String, require: true, unique: true
password	Type: String, require: true, private: true
createdDate	Type: Date, default: “Date now”

У таблиці 3.2 представлена конструкція постів, які зможе додавати користувач. При додаванні поста, обов’язковою умовою є авторизація, тому додано поле user. Також додатково, необов’язково можна додати фото та написати деякий текст посту. Звісно, додана можливість оцінки постів. У кожного поста буде відображена кількість вподобань, у БД вони будуть зберігатись у вигляді масиву.

Таблиця 3.2 – Поля моделі Post для зберігання у MongoDB

Body	Type: String, required: true
User	Type: Schema.Types.ObjectId, Ref: “users”, required: true

Likes	{
	User: {
	Type: Schema.Types.ObjectId,
	Ref: “users”,
	required: true
	},
	createdDate: {

	type: Date, default: Date.now } }

comments	{
	Body: {
	Type: String,
	Required: true
	}
	User: {
	Type: Schema.Types.ObjectId,
	Ref: “users”,
	required: true
	},
	createdDate: {
	type: Date,
	default: Date.now
	},
	createdDate: {
	type: Date,
	default: Date.now
	}
	}

createdDate	createdDate: {
	type: Date,
	default: Date.now
	}

У таблиці 3.3 представлена модель, яка буде реалізовувати підписку одного користувача на іншого. Додається поле Subscriber та Profile для того, щоб користувач міг бачити дані того, на кого збирається підписатись.

Таблиця 3.3 – Поля моделі Subscription для зберігання у MongoDB

Subscriber	type: Schema.Types.ObjectId, ref: 'users', required: true
Profile	type: Schema.Types.ObjectId, ref: 'users', required: true
createdDate	type: Date, default: Date.now

3.2 Програмна реалізація

На рисунку 3.2 показана стартова сторінка додатку. Після того, як користувач переходить у додаток, відображається header з навігаційними пунктами та пости, які були створені іншими користувачами раніше.

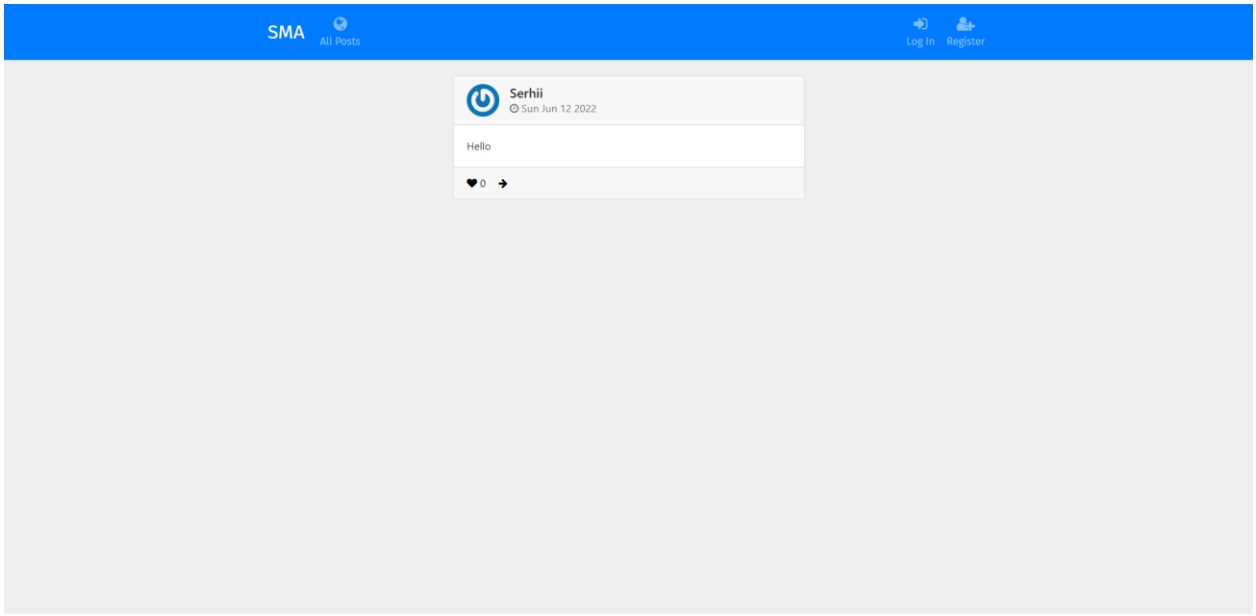

Рисунок 3.2 – Стартова сторінка додатку

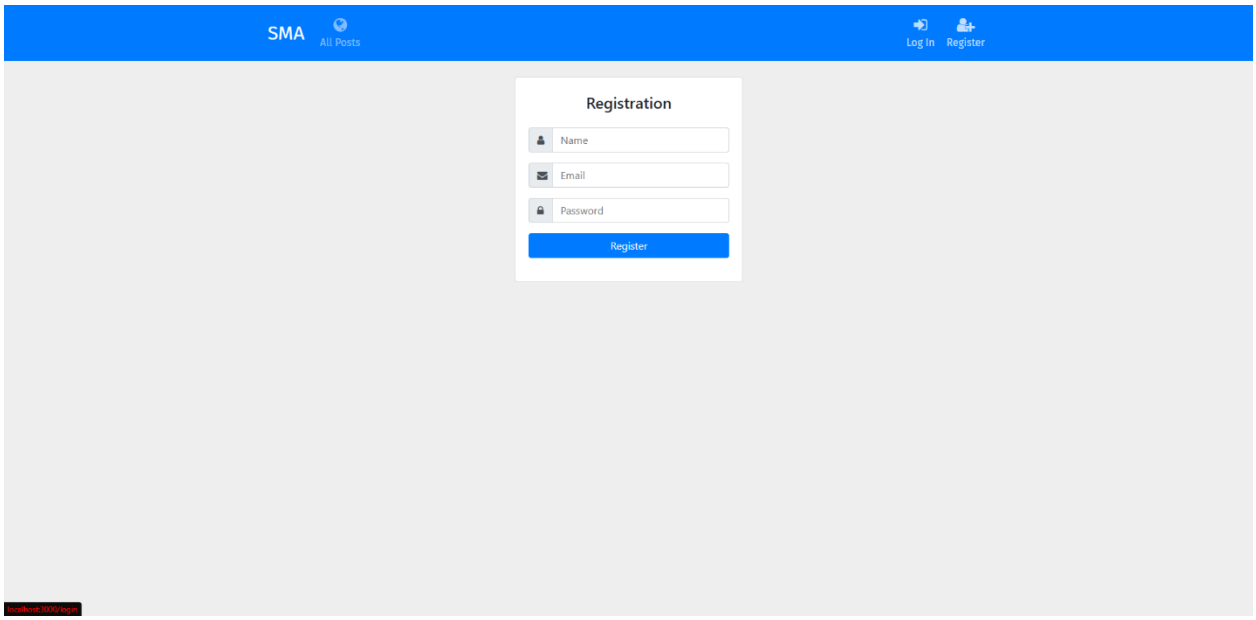

Рисунок 3.3 – Сторінка реєстрації користувача

На сторінці реєстрації, яка показана на рисунку 3.3, клієнт може створити новий акаунт. Для реєстрації необхідно ввести ім’я, електронну пошту та придумати пароль.

```
router.post('/register', async (ctx) => {
  const { name, email, password } = ctx.request.body
  const user = await User.findOne({ email })
  if (user) {
    ctx.throw(400, 'Email already exists')
  }
  const salt = await bcrypt.genSalt(10)
  const hash = await bcrypt.hash(password, salt)
  await new User({ email, name, password: hash }).save()
  ctx.status = 201
})
```

У прикладі коду показана реєстрація користувача на серверній стороні. Після того, як користувач натискає кнопку “Register” дані, які були введені в поля будуть відправлені на сервер. Далі, за допомогою findOne сервер перевірить, чи є користувач з такою електронною поштою у базі. Якщо є, то повертаємо помилку, якщо немає то починаємо процес хешування паролю та збереження даних користувача у БД.

Після того, як користувач пройшов процедуру реєстрації, відбувається перехід на сторінку авторизації, щоб перейти на головну сторінку додатку, де необхідно ввести електронну пошту та пароль.

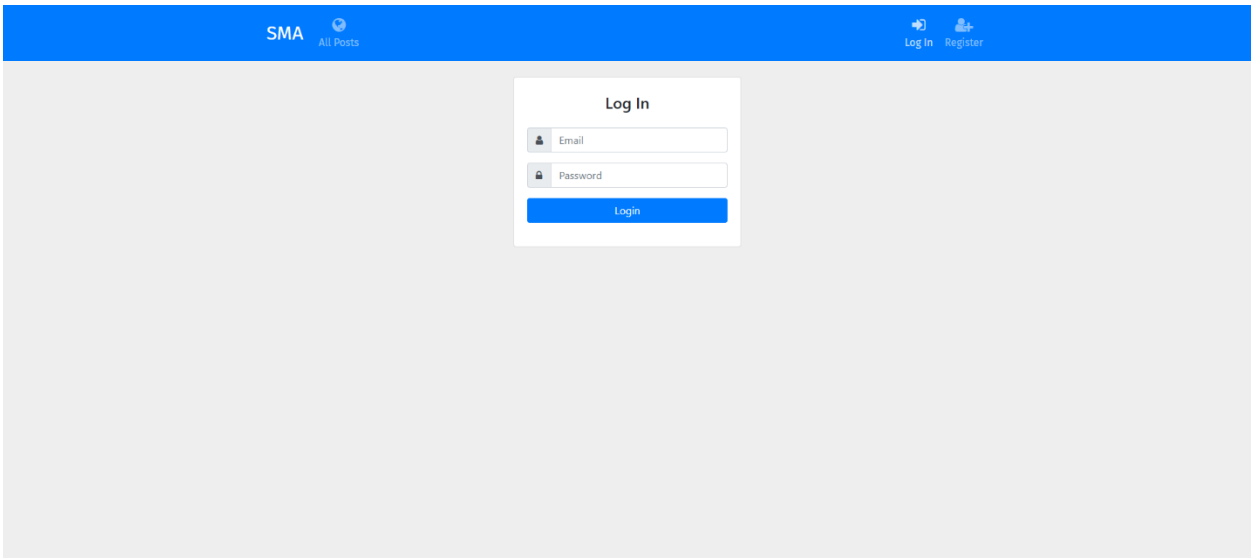

Рисунок 3.4 – Сторінка авторизації користувача

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

Після введення користувачем своїх даних, вони приходять на сторону сервера. Приклад процесу авторизації користувача на стороні сервера:

```
        const { email, password } = ctx.request.body
        const user = await User.findOne({ email })
        if (!user) {
          ctx.throw(400, 'User with this email does not exist')
        }
        const isMatch = await bcrypt.compare(password, user.password)
        if (isMatch) {
          const payload = {
            id: user.id,
            name: user.name,
            email: user.email
          }
          const token = jwt.sign(payload, config.secret, { expiresIn: 3600 * 24
})
          ctx.body = { token: `Bearer ${token}` }
        } else {
          ctx.throw(400, 'Password incorrect')
        }
      })
```

За допомогою функції findOne відбувається пошук користувача з такою електронною поштою. Якщо користувача не було знайдено, повертаємо помилку. Якщо пошук пройшов успішно, у моделі користувача забираємо пароль, для того, щоб його розхешувати за допомогою bcrypt.compare. Далі проводиться порівняння паролю, який ввів користувач з паролем у БД, якщо порівняння повертає true, то надаємо користувачу токен, з яким він зможе перейти на основну сторінку додатку.

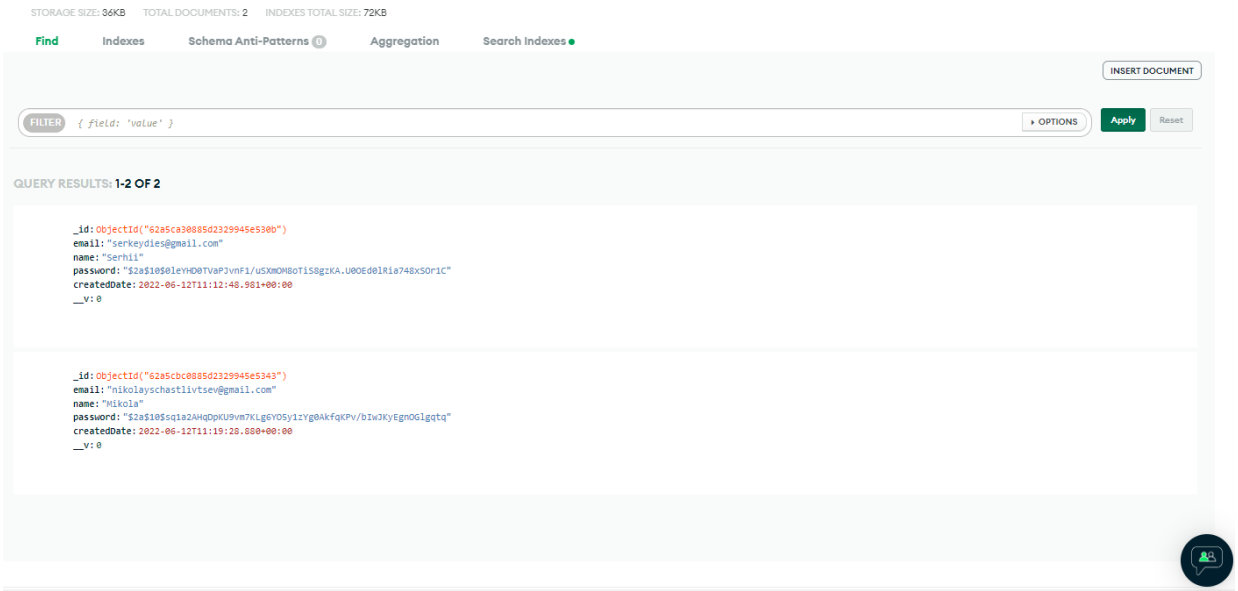

Рисунок 3.5 – Документ бази даних для моделі User

На рисунку 3.5 показано документ бази даних, який зберігає дані про користувачів у додатку. Варто звернути увагу, що хешування паролю проходить вдало, він зберігається у базі у зміненому вигляді, що позитивно впливає на безпеку даних користувача.

На головній сторінці додатку (рис.3.6) також виводиться список всіх постів, які раніше залишили користувачі додатку. Присутня форма для додавання нових постів користувачем. Додана можливість додавати посилання на відео та картинки у пості.

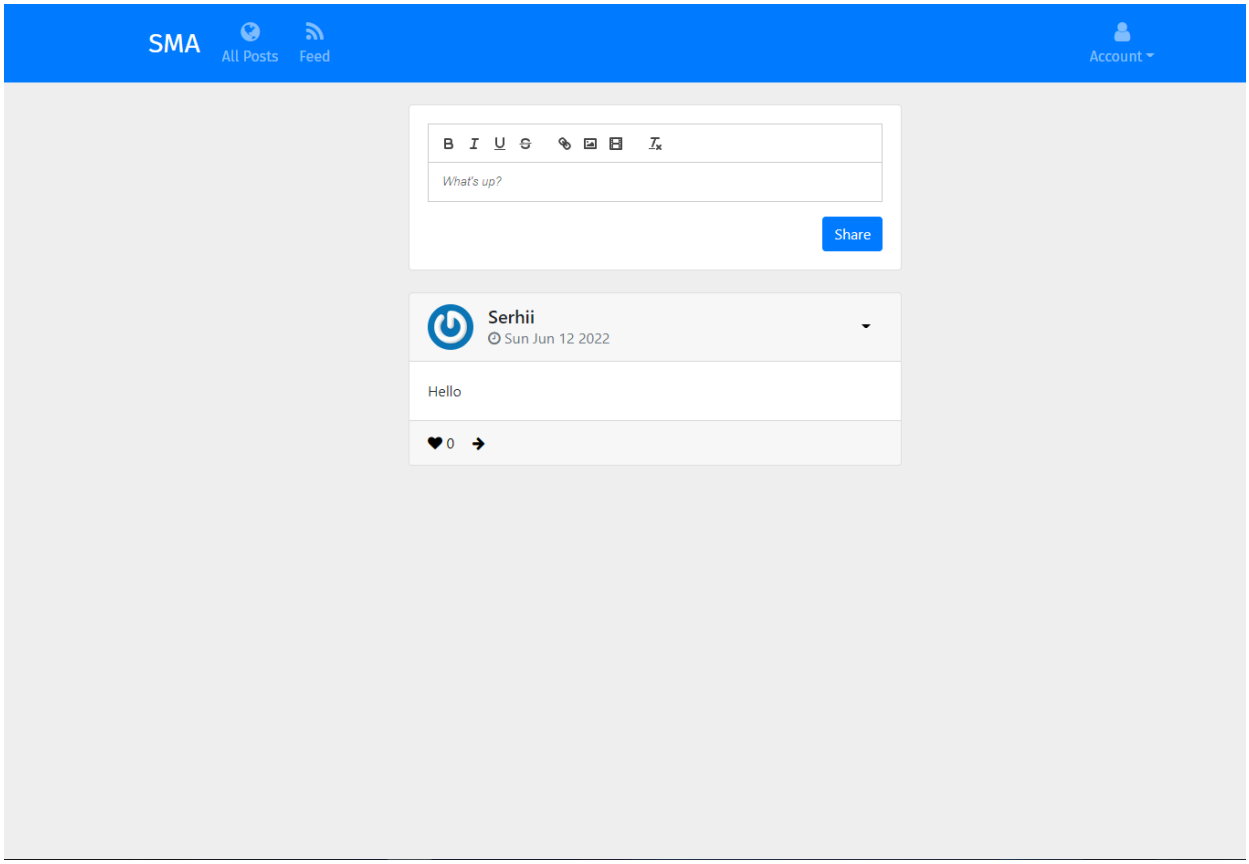

Рисунок 3.6 – Головна сторінка додатку

.

На рисунку 3.7 показано можливість додавання постів у з картинкою. Пости зберігаються у базі даних. Додана можливість оцінки постів та видалення їх. У кожного поста також вказується автор та час, коли він був доданий.

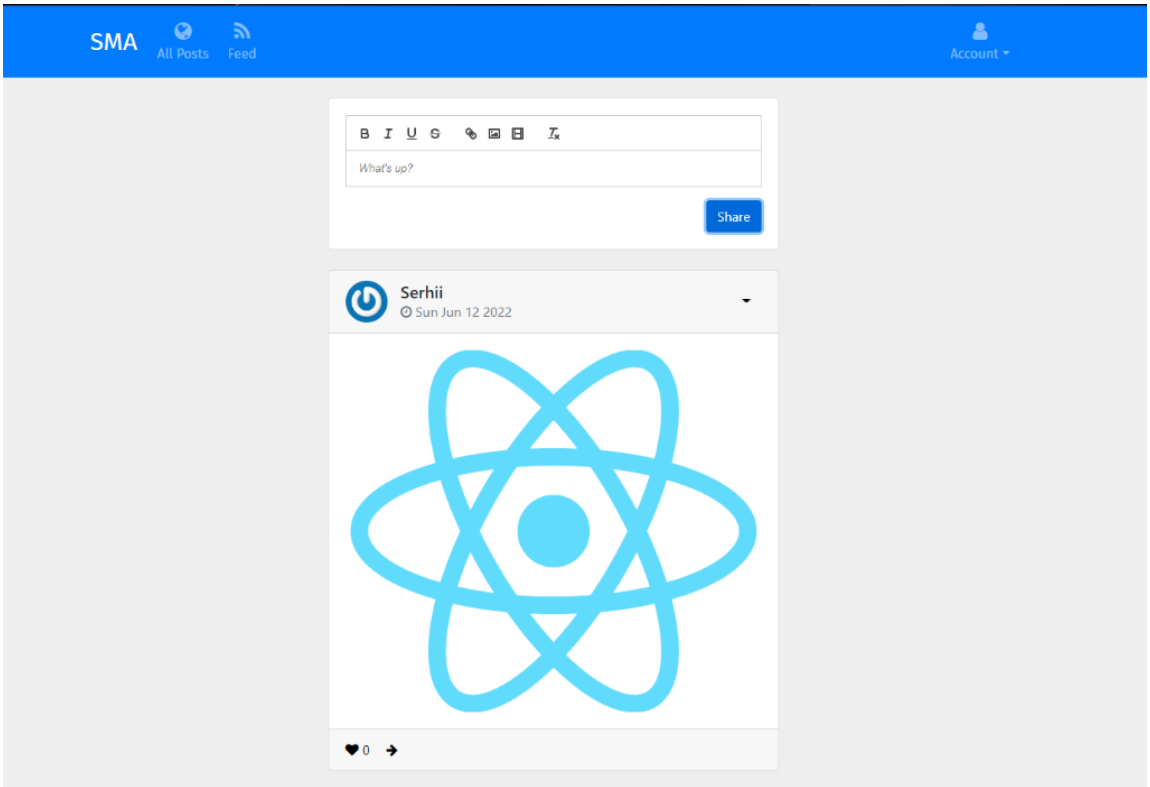

Рисунок 3.7 – Доданий пост із картинкою

Після отримання запита post на відповідний route, відбувається процес створення поста, що показано у прикладі нижче. З тіла запиту забираємо дані про користувача, створюємо новий поста за допомогою new Post та зберігаємо у базі даних.

```
      router.post('/', passport.authenticate('jwt', { session: false }), async
(ctx) => {
        const { body } = ctx.request.body
        const user = ctx.state.user._id
        ctx.body = await new Post({ body, user }).save()
        ctx.status = 201
      })
```

На рисунку 3.8 показано документ бази даних для збереження моделі постів. До кожного посту додана можливість оцінки поста. Список вподобано зберігається у БД у полі likes у вигляді масиву об’єктів. Вказується id користувача та дата.

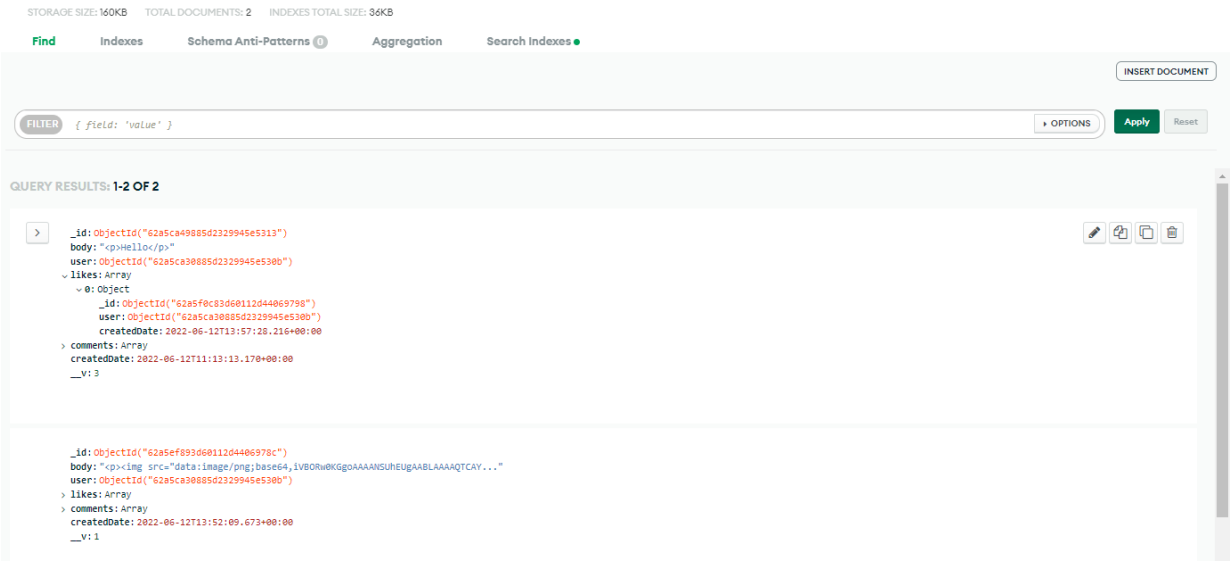

Рисунок 3.8 – Документ бази даних, який зберігає пости.

На рисунку 3.9 показано реалізацію додавання поста із відео. У поле введення, яке з’являється після натискання на відповідну кнопку, необхідно ввести посилання на відео, щоб завантажити його.

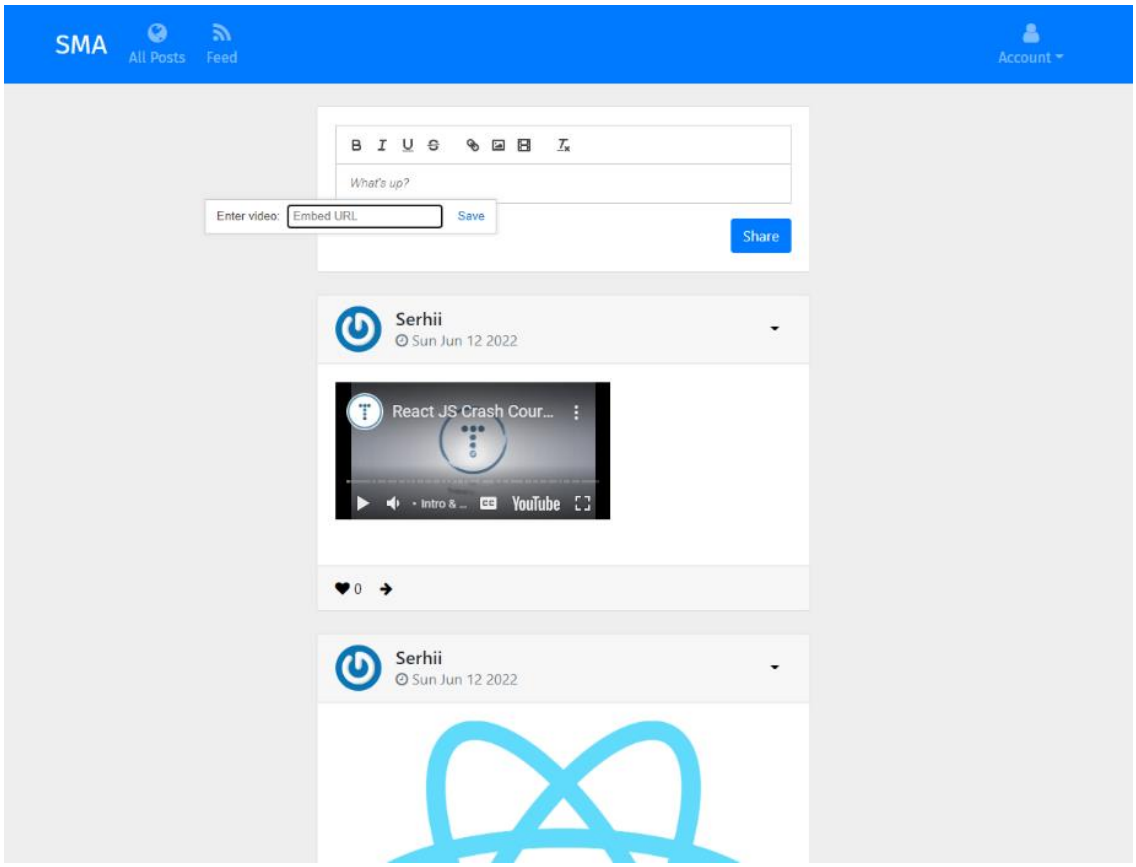

Рисунок 3.9 – Додавання поста із відео

На сторінці профілю, який показаний на рисунку 3.10, є можливість побачити тільки ті пости, які додавав сам користувач. Також можна прямо з цього інтерфейсу додавати нові пости.

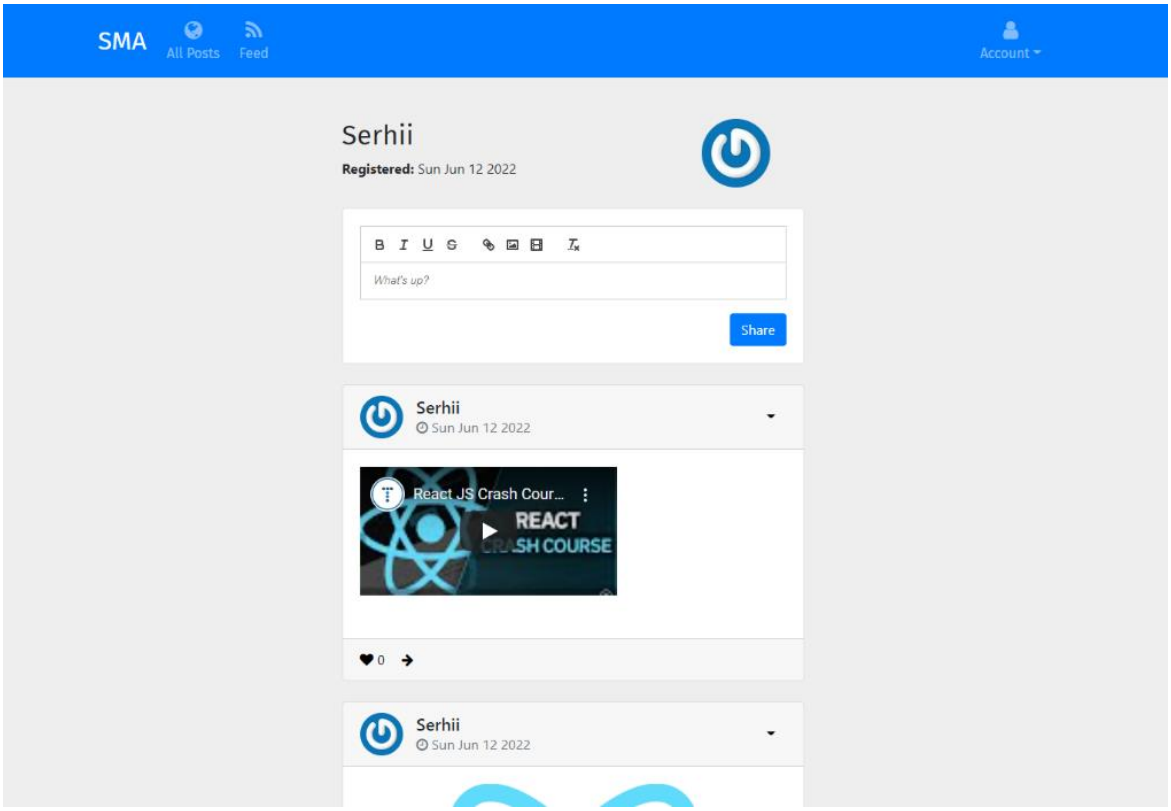

Рисунок 3.10 – Сторінка профілю користувача

```
export const getUserById = (id, history) => (dispatch) => {
  dispatch(setUserLoading(true))
  axios
    .get(`/api/users/${id}`)
    .then((res) => dispatch({
      type: GET_USER,
      payload: res.data
    }))
    .catch(() => {
      dispatch(setUserLoading(false))
      history.push('/404')
    })
}
```

У прикладі представлений reducer, який виконує логіку по додаванню даних про користувача, що прийшли з сервера у глобальне сховище даних на клієнтській стороні додатку. Потім, після збереження даних, у нас є можливість використати ці дані та показати користувачу дані про його профіль.

Інший користувач, який заходить на сторінку користувача, пост якого знайшов, потрапить на сторінку його профілю (рис.3.11). Також додана кнопка для підписки.

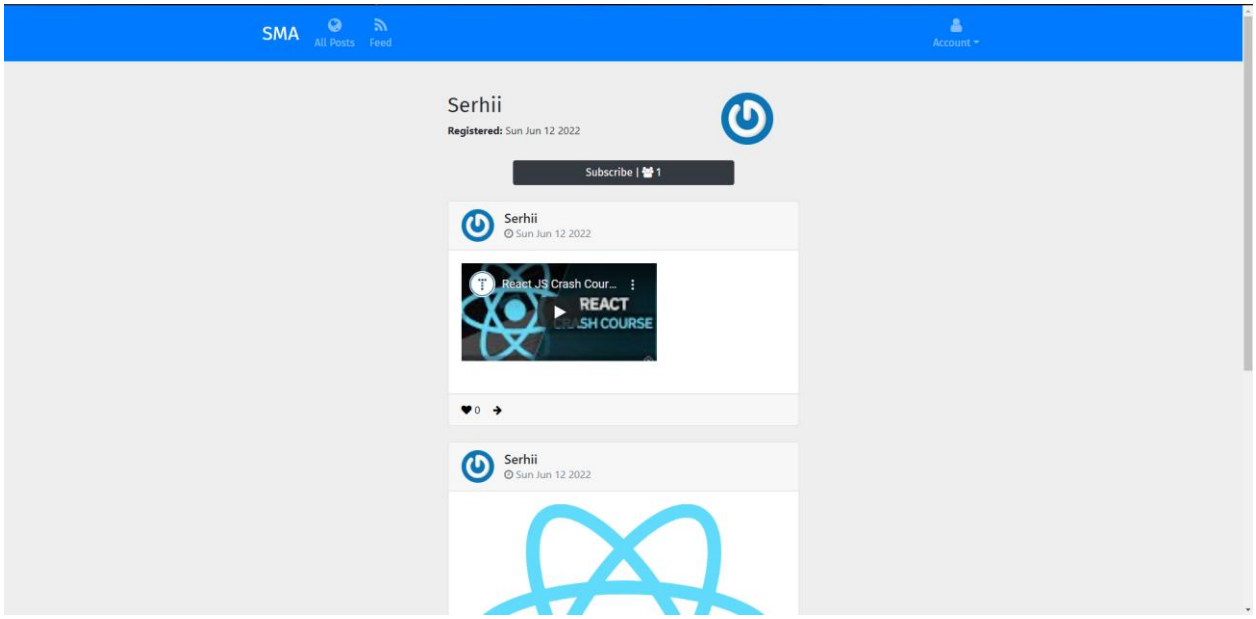

Рисунок 3.11 – Сторінка користувача з точки зору іншого користувача

```
      router.post('/', passport.authenticate('jwt', {
        session: false
      }), async (ctx) => {
        const { profile } = ctx.request.body
        const subscriber = ctx.state.user._id
        const checkSubscription = await Subscription.findOne({ subscriber,
profile })
        if (checkSubscription) {
          ctx.throw(400, 'You have alredy subscribed')
        }
        ctx.body = await new Subscription({ subscriber, profile }).save()
        ctx.status = 201
      })
```

У прикладі вище продемонстрована можливість підписки. З клієнтської частини на серверну приходять дані про двох користувачів. Далі йде перевірка, чи не підписаний уже користувач на цього клієнта. У раз існуючої підписки повертається помилка, якщо не – створюємо нову підписку.

На рисунку 3.12 показано документ бази даних, який зберігає дані про підписки. У документі створено поля для профілю користувача та його підписника. Після підписки, у полі Feed будуть відображатись лише пости користувачів, на яких є підписка.

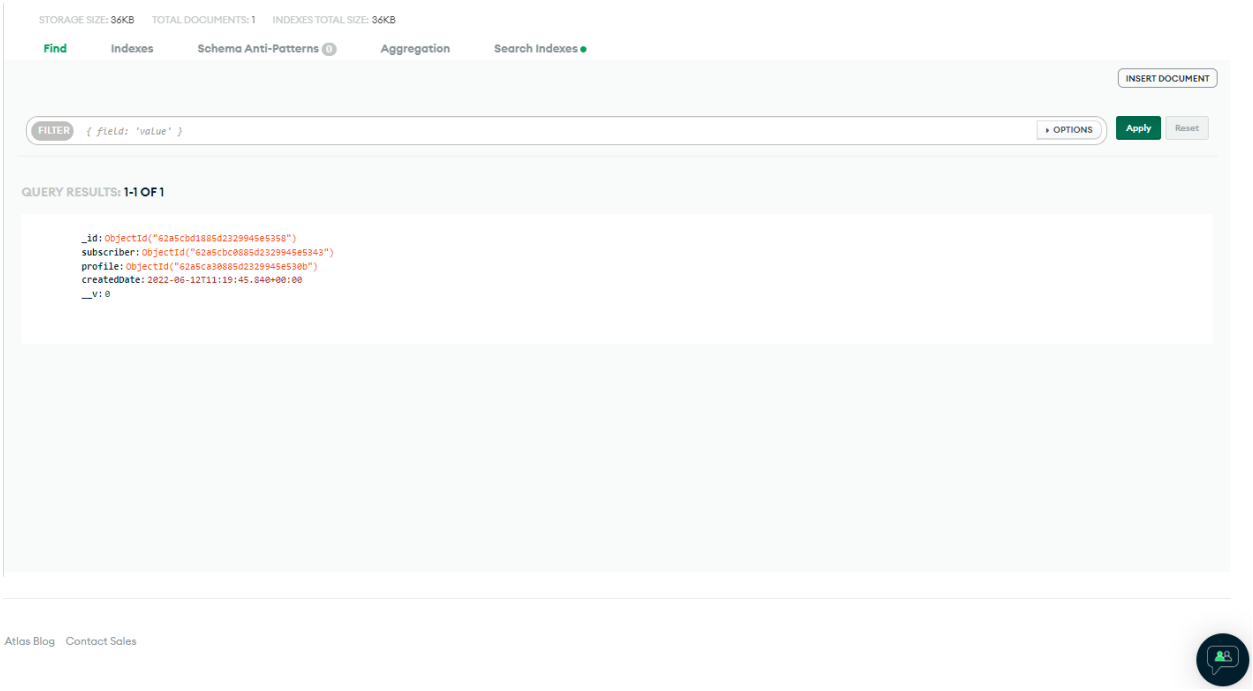

Рисунок 3.12 – Документ бази даних з користувацькими підписками

На рисунку 3.13 показано сторінку Feed. На сторінці відображено пости лише тих людей, на яких підписаний активний користувач.

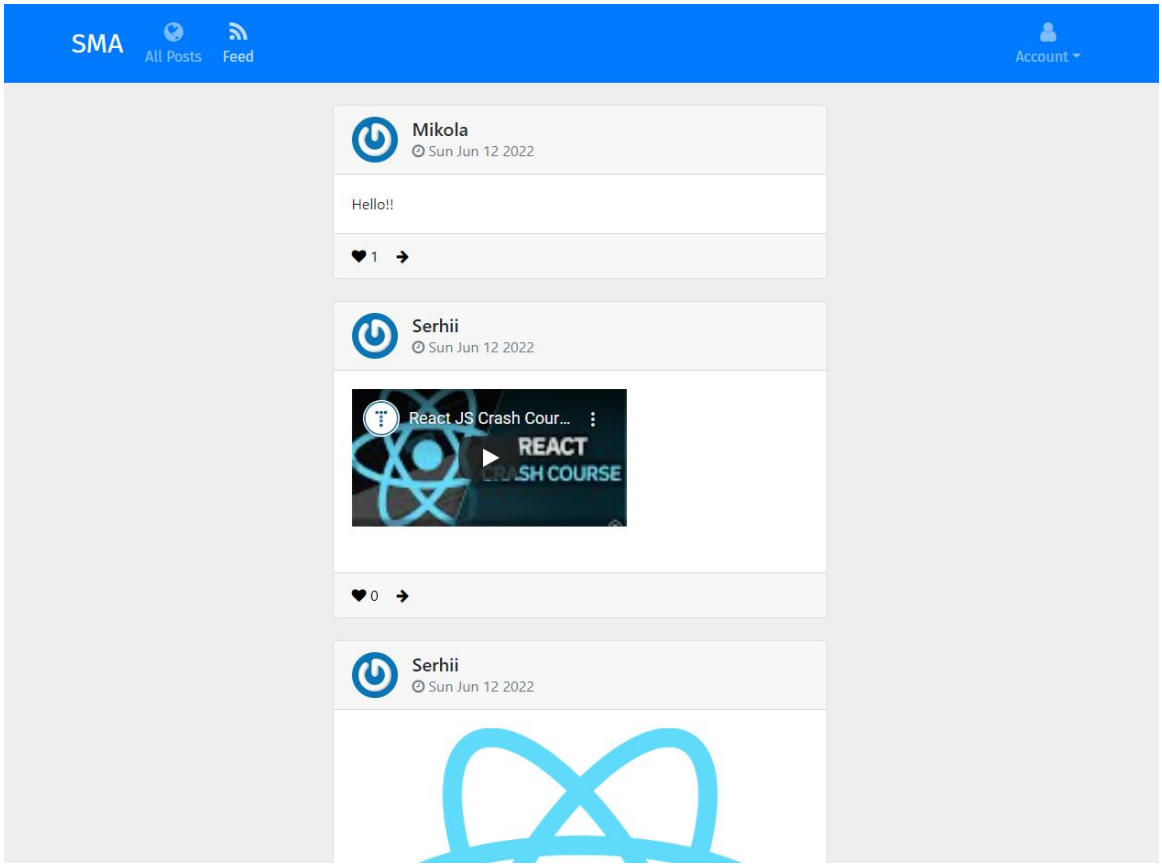

Рисунок 3.13 – Сторінка Feed додатку

```
const Feed = ({
  getAllSubscriptions, auth, subscription: { isLoading, subscriptions }
}) => {
  useEffect(() => getAllSubscriptions({ subscriber: auth.user.id }), [])
  return !isLoading ? (
    <div className="row mt-4">
      <div className="col-md-6 mx-auto">
        {subscriptions.length !== 0 ? (
          <Posts queryParams={{
            users: subscriptions.map((s) => s.profile).join(',')
          }} />
        ) : (
          <div className="text-center">
            <h2>You have no subscriptions</h2>
          </div>
        )}
      </div>
    </div>
  ) : <Loader />
}
```

У прикладі коду продемонстровано отримання постів, на яких оформлена підписка. Запит на сервер відбувається у useEffect з пустим масивом залежностей. Такий синтаксис гарантує, що запит відправиться тільки тоді, коли користувач вперше перейде на сторінку Feed для отримання оновлених даних. Далі за допомогою методу масива map додаємо пости на сторінку.

У додаток доданий адаптивний інтерфейс під різні пристрої. На рисунку 3.14 продемонстрована робота додатку на екрані ноутбука.

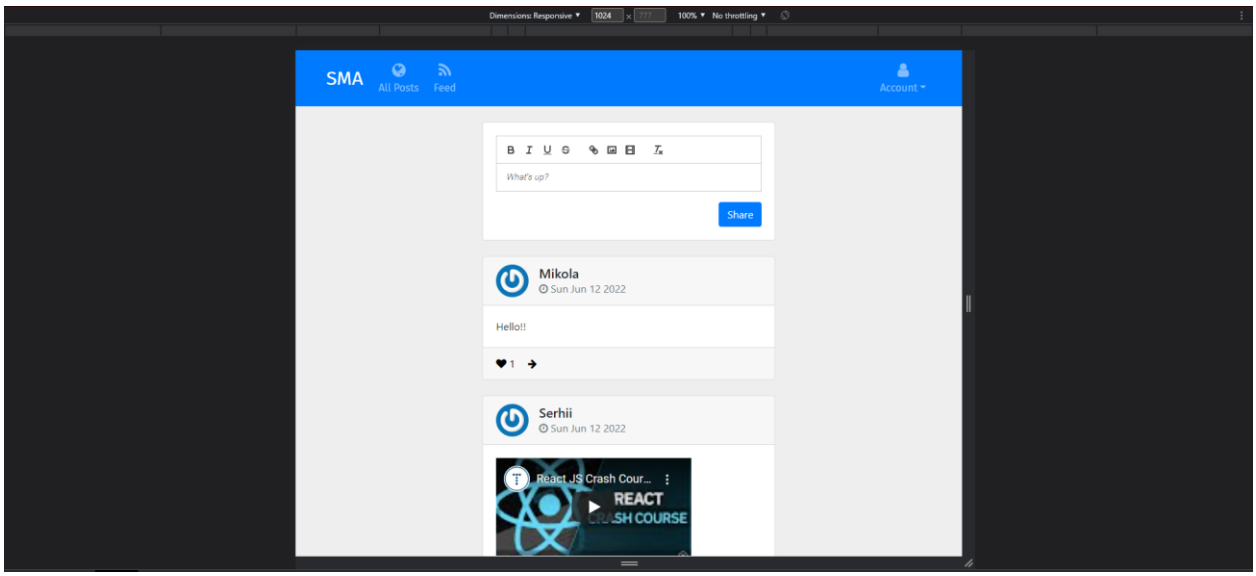

Рисунок 3.14 – Додаток на екрані ноутбука

На планшетній ширині (рис.3.15) замість звичайного меню з’являється адаптивне меню, яке відкривається по кліку на нього.

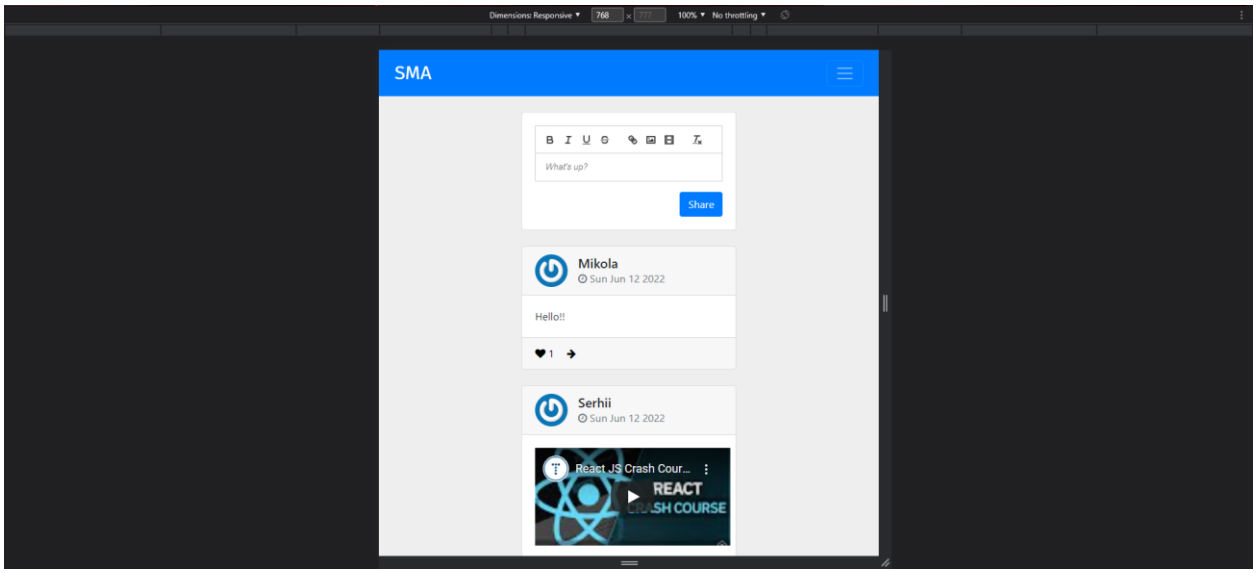

Рисунок 3.15 – Додаток на екрані планшету

На рисунку 3.16 показано додаток на екрані телефона з відкритим меню. Додаток працює та даній ширині та виконує основні функції. Користуватись ним зручно і не викликає жодних проблем.

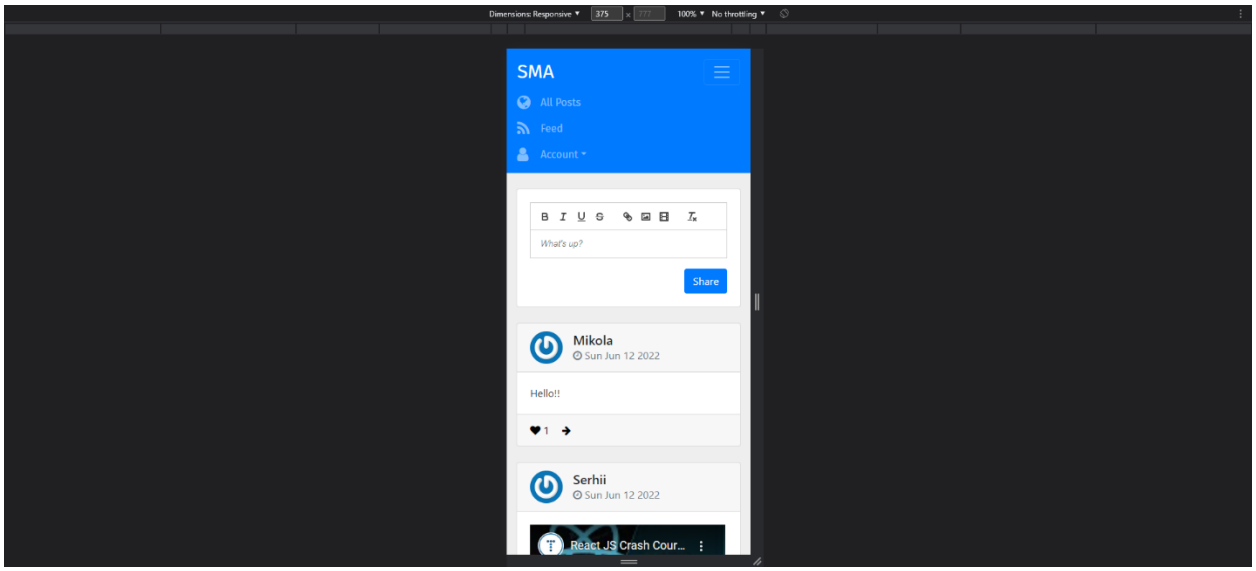

Рисунок 3.16 – Додаток на екрані телефона з відкритим меню

```
@media (min-width: 768px) {
      .navbar-icon-top.navbar-expand-md .navbar-nav .nav-link {
            text-align: center;
            display: table-cell;
            height: 70px;
```

```
                vertical-align: middle;
                padding-top: 0;
                padding-bottom: 0;
            }

            .navbar-icon-top.navbar-expand-md .navbar-nav .nav-link > .fa {
                display: block;
                width: 48px;
                margin: 2px auto 4px auto;
                top: 0;
                line-height: 24px;
            }

            .navbar-icon-top.navbar-expand-md .navbar-nav .nav-link > .fa > .badge {
                top: -7px;
            }
        }
```

Адаптування в основному відбувається за рахунок зміни сітки, зменшення шрифтів та розмірів основних елементів.

ВИСНОВКИ

У ході виконання кваліфікаційної бакалаврської роботи було виконано:

1) огляд та порівняння методик розробки соціальних мереж;

2) порівняння найпопулярніших додатків у цій сфері, виділення основних плюсів та мінусів;

3) огляд основних інструментів для розробки додатків у сучасній IT-сфері та обґрунтування обраного стеку технологій;

4) постановка вимог до кінцевого додатку;

5) опис дій при розробці клієнтської та серверної частини;

6) представлено конфігурацію основних моделей бази даних, які будуть використані при розробці.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Стаття з порівнянням соціальних мереж [Електронний ресурс] – Режим доступу до ресурсу: https://cyberleninka.ru/article/n/sravnitelnyy-analiz-sotsialnyh-setey/viewer
2. Інстумент для повудови таблиць та графіків [Електронний ресурс] – режим доступу до ресурсу: https://miro.com
3. Посібник: знаймоство з стеком MERN [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/mern-stack
4. Посібник: знаймоство з Bootstrap [Електронний ресурс] – Режим доступу до ресурсу: https://bootstrap-4.ru/docs/4.0/getting-started/introduction/
5. Посібник: знаймоство з Axios [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/axios
6. Посібник: знаймоство з React-router-dom [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/react-router-dom
7. Посібник: знаймоство з JQuery [Електронний ресурс] – Режим доступу до ресурсу: https://plugins.jquery.com/
8. Посібник: знаймоство з Js-md5 [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/js-md5
9. Посібник: знаймоство з Jwt-decode [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/jwt-decode
10. Посібник: знаймоство з Popper.js [Електронний ресурс] – Режим доступу до ресурсу: https://popper.js.org/docs/v2/
11. Посібник: знаймоство з React-js-pagination [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/react-js-pagination
12. Посібник: знаймоство з Passport-jwt [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/passport-jwt
13. Посібник: знаймоство з Bctypt [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/bcryptjs

14. Посібник: знаймоство з Dotenv [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/dotenv
15. Посібник: знаймоство з Mongoose [Електронний ресурс] – Режим доступу до ресурсу: https://mongoosejs.com/docs/guide.html
16. Посібник: знаймоство з Nodemon [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/nodemon
17. Посібник: знаймоство з jsonwebtoken [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/jsonwebtoken
18. Посібник: знаймоство з koa [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/koa
19. Посібник: знаймоство з Mongoose-private-paths [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/mongoose-private-paths
20. Посібник: знаймоство з React [Електронний ресурс] – Режим доступу до ресурсу: https://ru.reactjs.org/
21. Посібник: знаймоство з NodeJS [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/dist/latest-v17.x/docs/api/
22. Посібник: знаймоство з MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/docs/

ДОДАТОК А

Файл auth.js

```
import axios from 'axios'
import jwtDecode from 'jwt-decode'

import { SET_CURRENT_USER } from './types'
import setAuthToken from '../utils/setAuthToken'

export const register = (userData, history) => () => {
  axios
    .post('/api/auth/register', userData)
    .then(() => history.push('/login'))
}

export const login = (userData) => (dispatch) => {
  axios
    .post('/api/auth/login', userData)
    .then((res) => {
      const { token } = res.data
      localStorage.setItem('access_token', token)
      setAuthToken(token)
      const decoded = jwtDecode(token)
      dispatch(setCurrentUser(decoded))
    })
}

export const logout = () => (dispatch) => {
  localStorage.removeItem('access_token')
  setAuthToken(false)
  dispatch(setCurrentUser({}))
}

export const setCurrentUser = (user) => ({
  type: SET_CURRENT_USER,
  payload: user
})
```

Файл post.js

```
import axios from 'axios'

import {
  POST_LOADING,
  ADD_POST,
  GET_POSTS,
  GET_POST,
  DELETE_POST,
  CLEAR_POSTS,
  UPDATE_POST
} from './types'

export const create = (post) => (dispatch) => {
  axios
    .post('/api/posts', post)
    .then((res) => dispatch({
      type: ADD_POST,
      payload: res.data
    }))
}

export const getAll = (params) => (dispatch) => {
```

```
  dispatch(setPostLoading(true))
  axios
    .get('/api/posts', { params })
    .then((res) => dispatch({
      type: GET_POSTS,
      payload: {
        posts: res.data,
        totalCount: +res.headers['x-total-count']
      }
    }))
    .catch(() => {
      dispatch(setPostLoading(false))
      dispatch(clearPosts())
    })
}

export const getById = (id, history) => (dispatch) => {
  dispatch(setPostLoading(true))
  axios
    .get(`/api/posts/${id}`)
    .then((res) => dispatch({
      type: GET_POST,
      payload: res.data
    }))
    .catch(() => {
      dispatch(setPostLoading(false))
      history.push('/404')
    })
}

export const remove = (id) => (dispatch) => {
  axios
    .delete(`/api/posts/${id}`)
    .then(() => dispatch({
      type: DELETE_POST,
      payload: id
    }))
}

export const createLike = (postId, TYPE) => (dispatch) => {
  axios
    .post(`/api/posts/${postId}/likes`)
    .then((res) => dispatch({
      type: TYPE,
      payload: res.data
    }))
}

export const removeLike = (postId, likeId, TYPE) => (dispatch) => {
  axios
    .delete(`/api/posts/${postId}/likes/${likeId}`)
    .then((res) => dispatch({
      type: TYPE,
      payload: res.data
    }))
}

export const createComment = (postId, comment) => (dispatch) => {
  axios
    .post(`/api/posts/${postId}/comments`, comment)
    .then((res) => dispatch({
      type: UPDATE_POST,
      payload: res.data
    }))
```

```
}

export const removeComment = (postId, commentId) => (dispatch) => {
  axios
    .delete(`/api/posts/${postId}/comments/${commentId}`)
    .then((res) => dispatch({
      type: UPDATE_POST,
      payload: res.data
    }))
}

const clearPosts = () => ({
  type: CLEAR_POSTS
})

const setPostLoading = (isLoading) => ({
  type: POST_LOADING,
  payload: isLoading
})
```

Файл subscription.js

```
import axios from 'axios'

import {
  SUBSCRIPTION_LOADING,
  ADD_SUBSCRIPTION,
  GET_SUBSCRIPTIONS,
  DELETE_SUBSCRIPTION
} from './types'

export const create = (like) => (dispatch) => {
  axios
    .post('/api/subscriptions', like)
    .then((res) => dispatch({
      type: ADD_SUBSCRIPTION,
      payload: res.data
    }))
}

export const getAll = (params = {}) => (dispatch) => {
  dispatch(setSubscriptionLoading(true))
  axios
    .get('/api/subscriptions', { params })
    .then((res) => dispatch({
      type: GET_SUBSCRIPTIONS,
      payload: res.data
    }))
    .catch(() => dispatch(setSubscriptionLoading(false)))
}

export const remove = (id) => (dispatch) => {
  axios
    .delete(`/api/subscriptions/${id}`)
    .then(() => dispatch({
      type: DELETE_SUBSCRIPTION,
      payload: id
    }))
}

export const setSubscriptionLoading = (isLoading) => ({
  type: SUBSCRIPTION_LOADING,
```

```
  payload: isLoading
})
```

Файл types.js

```
export const SET_CURRENT_USER = 'SET_CURRENT_USER'

export const POST_LOADING = 'POST_LOADING'
export const ADD_POST = 'ADD_POST'
export const UPDATE_POST = 'UPDATE_POST'
export const UPDATE_POSTS = 'UPDATE_POSTS'
export const GET_POSTS = 'GET_POSTS'
export const GET_POST = 'GET_POST'
export const DELETE_POST = 'DELETE_POST'
export const CLEAR_POSTS = 'CLEAR_POSTS'

export const SUBSCRIPTION_LOADING = 'SUBSCRIPTION_LOADING'
export const ADD_SUBSCRIPTION = 'ADD_SUBSCRIPTION'
export const GET_SUBSCRIPTIONS = 'GET_SUBSCRIPTIONS'
export const DELETE_SUBSCRIPTION = 'DELETE_SUBSCRIPTION'

export const USER_LOADING = 'USER_LOADING'
export const GET_USER = 'GET_USER'
```

Файл user.js

```
import axios from 'axios'

import { GET_USER, USER_LOADING } from './types'

export const getUserById = (id, history) => (dispatch) => {
  dispatch(setUserLoading(true))
  axios
    .get(`/api/users/${id}`)
    .then((res) => dispatch({
      type: GET_USER,
      payload: res.data
    }))
    .catch(() => {
      dispatch(setUserLoading(false))
      history.push('/404')
    })
}

const setUserLoading = (isLoading) => ({
  type: USER_LOADING,
  payload: isLoading
})
```

Файл AllPosts.js

```
import React from 'react'
import PropTypes from 'prop-types'

import { connect } from '../../store'

import PostForm from '../shared/PostForm'
import Posts from '../shared/Posts'

const AllPosts = ({ auth }) => (
  <div className="row mt-4">
    <div className="col-md-6 mx-auto">
      {auth.isAuthenticated && <PostForm />}
      <Posts queryParams={{}} />
    </div>
  </div>
)
```

```
AllPosts.propTypes = {
  auth: PropTypes.object.isRequired
}

const mapStateToProps = (state) => ({
  auth: state.auth
})

export default connect(mapStateToProps)(AllPosts)
```

Файл Login.js

```
import React, { useState, useEffect } from 'react';
import PropTypes from 'prop-types';

import { connect } from '../../store';
import { login } from '../../actions/auth';

const Login = ({ history, auth, login }) => {
      const [email, setEmail] = useState('');
      const [password, setPassword] = useState('');

      useEffect(() => {
            if (auth.isAuthenticated) {
                  history.push('/');
            }
      }, [auth]);

      const onSubmit = e => {
            e.preventDefault();
            login({ email, password });
      };

      return (
            <div className='row mt-4'>
                  <div className='col-4 mx-auto'>
                        <div className='card'>
                              <article className='card-body'>
                                    <h4 className='card-title text-center mb-4
mt-1'>
                                          Log In
                                    </h4>
                                    <form onSubmit={onSubmit}>
                                          <div className='form-group'>
                                                <div className='input-group'>
                                                      <div className='input-
group-prepend'>
                                                            <span
className='input-group-text'>
                                                                  <i
className='fa fa-user'></i>
                                                            </span>
                                                      </div>
                                                      <input
                                                            className='form-
control'

      placeholder='Email'
                                                            type='email'
                                                            name='email'
```

```
                                                value={email}
                                                onChange={e =>
setEmail(e.target.value)}
                                                pattern='.{5,30}'
                                                required
                                          />
                                    </div>
                              </div>
                              <div className='form-group'>
                                    <div className='input-group'>
                                          <div className='input-
group-prepend'>
                                                <span
className='input-group-text'>
                                                      <i
className='fa fa-lock'></i>
                                                </span>
                                          </div>
                                          <input
                                                className='form-
control'

      placeholder='Password'
                                                type='password'
                                                name='password'
                                                value={password}
                                                onChange={e =>

      setPassword(e.target.value)
                                                }
                                                pattern='.{6,30}'
                                                required
                                          />
                                    </div>
                              </div>
                              <div className='form-group'>
                                    <button
                                          type='submit'
                                          className='btn btn-
primary btn-block'
                                    >
                                          Login
                                    </button>
                              </div>
                        </form>
                  </article>
            </div>
      </div>
</div>
      );
};

Login.propTypes = {
      login: PropTypes.func.isRequired,
      auth: PropTypes.object.isRequired
};

const mapStateToProps = state => ({ auth: state.auth });
```

```
export default connect(mapStateToProps, { login })(Login);
```

Файл Register.js

```
import React, { useState, useEffect } from 'react';
import PropTypes from 'prop-types';

import { connect } from '../../store';
import { register } from '../../actions/auth';

const Register = ({ auth, history, register }) => {
      const [name, setName] = useState('');
      const [email, setEmail] = useState('');
      const [password, setPassword] = useState('');

      useEffect(() => {
            if (auth.isAuthenticated) {
                  history.push('/');
            }
      }, []);

      const onSubmit = e => {
            e.preventDefault();
            register({ name, email, password }, history);
      };

      return (
            <div className='row mt-4'>
                  <div className='col-4 mx-auto'>
                        <div className='card'>
                              <article className='card-body'>
                                    <h4 className='card-title text-center mb-4
mt-1'>
                                          Registration
                                    </h4>
                                    <form onSubmit={onSubmit}>
                                          <div className='form-group'>
                                                <div className='input-group'>
                                                      <div className='input-
group-prepend'>
                                                            <span
className='input-group-text'>
                                                                  <i
className='fa fa-user'></i>
                                                            </span>
                                                      </div>
                                                      <input
                                                            className='form-
control'
                                                            placeholder='Name'
                                                            type='text'
                                                            name='name'
                                                            value={name}
                                                            onChange={e =>
setName(e.target.value)}
                                                            pattern='.{3,20}'
                                                            required
                                                      />
                                                </div>
                                          </div>
                                          <div className='form-group'>
```

```
                                    <div className='input-group'>
                                          <div className='input-
group-prepend'>
                                                <span
className='input-group-text'>
                                                      <i
className='fa fa-envelope'></i>
                                                </span>
                                          </div>
                                          <input
                                                className='form-
control'
      placeholder='Email'
                                                type='email'
                                                name='email'
                                                value={email}
                                                onChange={e =>
setEmail(e.target.value)}
                                                pattern='.{5,30}'
                                                required
                                          />
                                    </div>
                              </div>
                              <div className='form-group'>
                                    <div className='input-group'>
                                          <div className='input-
group-prepend'>
                                                <span
className='input-group-text'>
                                                      <i
className='fa fa-lock'></i>
                                                </span>
                                          </div>
                                          <input
                                                className='form-
control'
      placeholder='Password'
                                                type='password'
                                                name='password'
                                                value={password}
                                                onChange={e =>
      setPassword(e.target.value)
                                                }
                                                pattern='.{6,30}'
                                          />
                                    </div>
                              </div>
                              <div className='form-group'>
                                    <button
                                          type='submit'
                                          className='btn btn-
primary btn-block'
                                    >
                                          Register
                                    </button>
                              </div>
```

```
                                    </form>
                                </article>
                            </div>
                        </div>
                    </div>
            );
};

Register.propTypes = {
      register: PropTypes.func.isRequired,
      auth: PropTypes.object.isRequired
};

const mapStateToProps = state => ({ auth: state.auth });

export default connect(mapStateToProps, { register })(Register);
```

Файл Feed.js

```
import React, { useEffect } from 'react'
import PropTypes from 'prop-types'

import { connect } from '../../store'
import { getAll as getAllSubscriptions } from '../../actions/subscription'
import Loader from '../shared/Loader'
import Posts from '../shared/Posts'

const Feed = ({
  getAllSubscriptions, auth, subscription: { isLoading, subscriptions }
}) => {
  useEffect(() => getAllSubscriptions({ subscriber: auth.user.id }), [])
  return !isLoading ? (
    <div className="row mt-4">
      <div className="col-md-6 mx-auto">
        {subscriptions.length !== 0 ? (
          <Posts queryParams={{
            users: subscriptions.map((s) => s.profile).join(',')
          }} />
        ) : (
          <div className="text-center">
            <h2>You have no subscriptions</h2>
          </div>
        )}
      </div>
    </div>
  ) : <Loader />
}

Feed.propTypes = {
  getAllSubscriptions: PropTypes.func.isRequired,
  subscription: PropTypes.object.isRequired,
  auth: PropTypes.object.isRequired
}

const mapStateToProps = (state) => ({
  subscription: state.subscription,
  auth: state.auth
})

export default connect(mapStateToProps, { getAllSubscriptions })(Feed)
```

Файл Header.js

```
import React from 'react';
import { Link } from 'react-router-dom';
import PropTypes from 'prop-types';

import { connect } from '../../store';
import { logout } from '../../actions/auth';

const Header = ({ logout, auth: { isAuthenticated, user } }) => {
	const onLogout = e => {
		e.preventDefault();
		logout();
	};
	let links;
	if (isAuthenticated) {
		links = (
			<li className='nav-item dropdown'>
				<a
					className='nav-link dropdown-toggle'
					href='/#'
					id='navbarDropdown'
					role='button'
					data-toggle='dropdown'
					aria-haspopup='true'
					aria-expanded='false'
				>
					<i className='fa fa-user'></i>
					Account
				</a>
				<div className='dropdown-menu' aria-
labelledby='navbarDropdown'>
					<Link className='dropdown-item' to={'/user/' +
user.id}>
						My Profile
					</Link>
					<div className='dropdown-divider'></div>
					<a className='dropdown-item' href='/#'
onClick={onLogout}>
						Log Out
					</a>
				</div>
			</li>
		);
	} else {
		links = (
			<React.Fragment>
				<li className='nav-item'>
					<Link className='nav-link' to='/login'>
						<i className='fa fa-sign-in'></i>
						Log In
					</Link>
				</li>
				<li className='nav-item'>
					<Link className='nav-link' to='/register'>
						<i className='fa fa-user-plus'></i>
						Register
					</Link>
				</li>
			</React.Fragment>
```

```
            );
      }
      return (
            <nav className='navbar navbar-icon-top navbar-expand-lg navbar-
dark bg-primary'>
                  <div className='container'>
                        <Link className='navbar-brand' to='/'>
                              SMA
                        </Link>
                        <button
                              className='navbar-toggler'
                              type='button'
                              data-toggle='collapse'
                              data-target='#navbarSupportedContent'
                              aria-controls='navbarSupportedContent'
                              aria-expanded='false'
                              aria-label='Toggle navigation'
                        >
                              <span className='navbar-toggler-icon'></span>
                        </button>
                        <div
                              className='collapse navbar-collapse'
                              id='navbarSupportedContent'
                        >
                              <ul className='navbar-nav mr-auto'>
                                    <li className='nav-item'>
                                          <Link className='nav-link' to='/'>
                                                <i className='fa fa-
globe'></i>
                                                All Posts
                                          </Link>
                                    </li>
                                    {isAuthenticated && (
                                          <li className='nav-item'>
                                                <Link className='nav-link'
to='/feed'>
                                                      <i className='fa fa-
rss'></i>
                                                      Feed
                                                </Link>
                                          </li>
                                    )}
                              </ul>
                              <ul className='navbar-nav'>{links}</ul>
                        </div>
                  </div>
            </nav>
      );
};

Header.propTypes = {
      logout: PropTypes.func.isRequired,
      auth: PropTypes.object.isRequired
};

const mapStateToProps = state => ({ auth: state.auth });

export default connect(mapStateToProps, { logout })(Header);
```

Файл NotFound.js

import React from 'react'

```
export default () => (
  <div className="text-center mt-5">
    <h1>Page Not Found</h1>
  </div>
)
```

Файл auth.js

```
import { SET_CURRENT_USER } from '../actions/types'

export default (state, action) => {
  switch (action.type) {
    case SET_CURRENT_USER:
      return {
        ...state,
        isAuthenticated: Object.keys(action.payload).length !== 0,
        user: action.payload
      }
    default:
      return state
  }
}
```

Файл index.js

```
import { combineReducers } from '../store'

import auth from './auth'
import post from './post'
import subscription from './subscription'
import user from './user'

export default combineReducers({ auth, post, subscription, user })
```

Файл post.js

```
import {
  POST_LOADING,
  ADD_POST,
  UPDATE_POST,
  UPDATE_POSTS,
  GET_POSTS,
  GET_POST,
  DELETE_POST,
  CLEAR_POSTS
} from '../actions/types'

export default (state, action) => {
  switch (action.type) {
    case POST_LOADING:
      return {
        ...state,
        isLoading: action.payload
      }
    case CLEAR_POSTS:
      return {
        ...state,
        posts: [],
        totalCount: 0
      }
    case GET_POSTS:
      return {
```

```
        ...state,
        posts: action.payload.posts,
        totalCount: action.payload.totalCount,
        isLoading: false
      }
    case GET_POST:
      return {
        ...state,
        post: action.payload,
        isLoading: false
      }
    case ADD_POST:
      return {
        ...state,
        posts: [action.payload, ...state.posts]
      }
    case UPDATE_POSTS:
      return {
        ...state,
        posts: state.posts.map((p) => p._id === action.payload._id ?
action.payload : p)
      }
    case UPDATE_POST:
      return {
        ...state,
        post: action.payload
      }
    case DELETE_POST:
      return {
        ...state,
        posts: state.posts.filter((post) => post._id !== action.payload)
      }
    default:
      return state
  }
}
```

Файл subscription.js

```
import {
  SUBSCRIPTION_LOADING,
  ADD_SUBSCRIPTION,
  GET_SUBSCRIPTIONS,
  DELETE_SUBSCRIPTION
} from '../actions/types'

export default (state, action) => {
  switch (action.type) {
    case SUBSCRIPTION_LOADING:
      return {
        ...state,
        isLoading: action.payload
      }
    case GET_SUBSCRIPTIONS:
      return {
        ...state,
        subscriptions: action.payload,
        isLoading: false
      }
    case ADD_SUBSCRIPTION:
      return {
```

```
        ...state,
        subscriptions: [action.payload, ...state.subscriptions]
      }
    case DELETE_SUBSCRIPTION:
      return {
        ...state,
        subscriptions: state.subscriptions.filter((s) => s._id !==
action.payload)
      }
    default:
      return state
  }
}
```

Файл index.js

```
import React from 'react'

const Context = React.createContext()

export const Provider = ({ initialState, reducer, children }) => {
  const [state, dispatch] = React.useReducer(reducer, initialState)
  return (
    <Context.Provider value={{ state, dispatch }}>
      {children}
    </Context.Provider>
  )
}

const wrapActions = (actions, dispatch) => {
  const result = {}
  for (const key in actions) {
    result[key] = (...args) => actions[key](...args)(dispatch)
  }
  return result
}

export const connect = (mapStateToProps, actions) => (Component) => (props)
=> (
  <Context.Consumer>
    {({ state, dispatch }) => (
      <Component
        {...props}
        {...mapStateToProps(state)}
        {...wrapActions(actions, dispatch)}
      />
    )}
  </Context.Consumer>
)

export const combineReducers = (reducers) => (state, action) => {
  const nextState = {}
  for (const key in reducers) {
    const previousStateForKey = state[key]
    const nextStateForKey = reducers[key](previousStateForKey, action)
    nextState[key] = nextStateForKey
  }
  return nextState
}
```

Файл initialState.js

```
import jwtDecode from 'jwt-decode'

import setAuthToken from '../utils/setAuthToken'

const initialState = {
  auth: {
    isAuthenticated: false,
    user: {}
  },
  post: {
    posts: [],
    totalCount: 0,
    post: null,
    isLoading: false
  },
  subscription: {
    subscriptions: [],
    isLoading: false
  },
  user: {
    user: null,
    isLoading: false
  }
}

if (localStorage.access_token) {
  const { access_token } = localStorage
  setAuthToken(access_token)
  const decoded = jwtDecode(access_token)
  initialState.auth.user = decoded
  initialState.auth.isAuthenticated = true
  const currentTime = Date.now() / 1000
  if (decoded.exp < currentTime) {
    localStorage.removeItem('access_token')
    setAuthToken(false)
    initialState.auth.user = {}
    initialState.auth.isAuthenticated = false
    window.location.href = '/login'
  }
}

export default initialState
```

Файл setAuthToken.js

```
import axios from 'axios'

export default (token) => {
  if (token) {
    axios.defaults.headers.common['Authorization'] = token
  } else {
    delete axios.defaults.headers.common['Authorization']
  }
}
```

Файл App.js

```
import React from 'react'
import { BrowserRouter, Route, Switch } from 'react-router-dom'

import { Provider } from './store'
import rootReducer from './reducers'
```

```
import initialState from './store/initialState'

import PrivateRoute from './components/shared/PrivateRoute'
import Header from './components/layout/Header'
import Footer from './components/layout/Footer'
import Login from './components/auth/Login'
import Register from './components/auth/Register'
import AllPosts from './components/all-posts/AllPosts'
import SinglePost from './components/single-post/SinglePost'
import UserProfile from './components/user-profile/UserProifle'
import Feed from './components/feed/Feed'
import NotFound from './components/not-found/NotFound'

function App() {
  return (
    <Provider reducer={rootReducer} initialState={initialState}>
      <BrowserRouter>
        <React.Fragment>
          <Header />
          <div className="container">
            <Route path="/register" component={Register} />
            <Route path="/login" component={Login} />
            <Route exact path="/" component={AllPosts} />
            <Route path="/post/:id" component={SinglePost} />
            <Route path="/user/:id" component={UserProfile} />
            <Switch>
              <PrivateRoute path="/feed" component={Feed} />
            </Switch>
            <Route path="/404" component={NotFound} />
          </div>
          <Footer />
        </React.Fragment>
      </BrowserRouter>
    </Provider>
  )
}

export default App
```

Файл index.css

```
body {
      height: 100vh;
      background: #eee;
}

.navbar-brand {
      font-family: 'Fira Sans', sans-serif;
}

.navbar-collapse,
.profile-username,
.subscribe-btn {
      font-family: 'Fira Sans', sans-serif;
}

.card-link,
.btn-link {
      color: #000;
}
```

```
.card-body p img {
	width: 100%;
}

p {
	margin-bottom: 0;
}

.input-group-text {
	width: 37px;
}

.navbar-brand {
	font-size: 30px;
}

.navbar-icon-top .navbar-nav .nav-link > .fa {
	position: relative;
	width: 36px;
	font-size: 24px;
}

.navbar-icon-top .navbar-nav .nav-link > .fa > .badge {
	font-size: 0.75rem;
	position: absolute;
	right: 0;
	font-family: sans-serif;
}

.navbar-icon-top .navbar-nav .nav-link > .fa {
	top: 3px;
	line-height: 12px;
}

.navbar-icon-top .navbar-nav .nav-link > .fa > .badge {
	top: -10px;
}

@media (min-width: 576px) {
	.navbar-icon-top.navbar-expand-sm .navbar-nav .nav-link {
		text-align: center;
		display: table-cell;
		height: 70px;
		vertical-align: middle;
		padding-top: 0;
		padding-bottom: 0;
	}

	.navbar-icon-top.navbar-expand-sm .navbar-nav .nav-link > .fa {
		display: block;
		width: 48px;
		margin: 2px auto 4px auto;
		top: 0;
		line-height: 24px;
	}

	.navbar-icon-top.navbar-expand-sm .navbar-nav .nav-link > .fa > .badge {
		top: -7px;
	}
```

```
}

@media (min-width: 768px) {
	.navbar-icon-top.navbar-expand-md .navbar-nav .nav-link {
		text-align: center;
		display: table-cell;
		height: 70px;
		vertical-align: middle;
		padding-top: 0;
		padding-bottom: 0;
	}

	.navbar-icon-top.navbar-expand-md .navbar-nav .nav-link > .fa {
		display: block;
		width: 48px;
		margin: 2px auto 4px auto;
		top: 0;
		line-height: 24px;
	}

	.navbar-icon-top.navbar-expand-md .navbar-nav .nav-link > .fa > .badge {
		top: -7px;
	}
}

@media (min-width: 992px) {
	.navbar-icon-top.navbar-expand-lg .navbar-nav .nav-link {
		text-align: center;
		display: table-cell;
		height: 70px;
		vertical-align: middle;
		padding-top: 0;
		padding-bottom: 0;
	}

	.navbar-icon-top.navbar-expand-lg .navbar-nav .nav-link > .fa {
		display: block;
		width: 48px;
		margin: 2px auto 4px auto;
		top: 0;
		line-height: 24px;
	}

	.navbar-icon-top.navbar-expand-lg .navbar-nav .nav-link > .fa > .badge {
		top: -7px;
	}
}

@media (min-width: 1200px) {
	.navbar-icon-top.navbar-expand-xl .navbar-nav .nav-link {
		text-align: center;
		display: table-cell;
		height: 70px;
		vertical-align: middle;
		padding-top: 0;
		padding-bottom: 0;
	}

	.navbar-icon-top.navbar-expand-xl .navbar-nav .nav-link > .fa {
```

```
            display: block;
            width: 48px;
            margin: 2px auto 4px auto;
            top: 0;
            line-height: 24px;
      }

      .navbar-icon-top.navbar-expand-xl .navbar-nav .nav-link > .fa > .badge {
            top: -7px;
      }
}
```

Файл auth.js

```
const Router = require('koa-router')
const bcrypt = require('bcryptjs')
const jwt = require('jsonwebtoken')

const User = require('../models/User')
const config = require('../lib/config')

const router = new Router().prefix('/auth')

router.post('/register', async (ctx) => {
  const { name, email, password } = ctx.request.body
  const user = await User.findOne({ email })
  if (user) {
    ctx.throw(400, 'Email already exists')
  }
  const salt = await bcrypt.genSalt(10)
  const hash = await bcrypt.hash(password, salt)
  await new User({ email, name, password: hash }).save()
  ctx.status = 201
})

router.post('/login', async (ctx) => {
  const { email, password } = ctx.request.body
  const user = await User.findOne({ email })
  if (!user) {
    ctx.throw(400, 'User with this email does not exist')
  }
  const isMatch = await bcrypt.compare(password, user.password)
  if (isMatch) {
    const payload = {
      id: user.id,
      name: user.name,
      email: user.email
    }
    const token = jwt.sign(payload, config.secret, { expiresIn: 3600 * 24 })
    ctx.body = { token: `Bearer ${token}` }
  } else {
    ctx.throw(400, 'Password incorrect')
  }
})

module.exports = router.routes()
```

Файл index.js

```
const Router = require('koa-router')

const auth = require('./auth')
```

```
const posts = require('./posts')
const postsComments = require('./posts-comments')
const postsLikes = require('./posts-likes')
const subscriptions = require('./subscriptions')
const users = require('./users')

const router = new Router().prefix('/api')

router.use(auth, posts, postsComments, postsLikes, subscriptions, users)

module.exports = router
```

Файл posts-comments.js

```
const Router = require('koa-router')
const passport = require('koa-passport')

const Post = require('../models/Post')

const router = new Router().prefix('/posts/:postId/comments')

router.post('/', passport.authenticate('jwt', { session: false }), async
(ctx) => {
  const post = await Post.findById(ctx.params.postId)
  if (!post) {
    ctx.throw(404, 'Post has not been found')
  }
  const { body } = ctx.request.body
  post.comments.unshift({ body, user: ctx.state.user._id })
  ctx.body = await post.save()
})

router.delete('/:commentId', passport.authenticate('jwt', {
  session: false
}), async (ctx) => {
  const post = await Post.findById(ctx.params.postId)
  if (!post) {
    ctx.throw(404, 'Post has not been found')
  }
  const commentIndex = post.comments
    .findIndex((c) => c._id.toString() === ctx.params.commentId)
  if (commentIndex < 0) {
    ctx.throw(404, 'Comment has not been found')
  }
  post.comments.splice(commentIndex, 1)
  ctx.body = await post.save()
})

module.exports = router.routes()
```

Файл posts-likes.js

```
const Router = require('koa-router')
const passport = require('koa-passport')

const Post = require('../models/Post')

const router = new Router().prefix('/posts/:postId/likes')

router.post('/', passport.authenticate('jwt', { session: false }), async
(ctx) => {
  const post = await Post.findById(ctx.params.postId)
```

```
  if (!post) {
    ctx.throw(404, 'Post has not been found')
  }
  const user = ctx.state.user._id
  if (post.likes.find((l) => l.user.toString() === user.toString())) {
    ctx.throw(400, 'User already liked this post')
  }
  post.likes.unshift({ user })
  ctx.body = await post.save()
})

router.delete('/:likeId', passport.authenticate('jwt', { session: false }),
async (ctx) => {
  const post = await Post.findById(ctx.params.postId)
  if (!post) {
    ctx.throw(404, 'Post has not been found')
  }
  const likeIndex = post.likes.findIndex((l) => l._id.toString() ===
ctx.params.likeId)
  if (likeIndex < 0) {
    ctx.throw(404, 'Like has not been found')
  }
  post.likes.splice(likeIndex, 1)
  ctx.body = await post.save()
})

module.exports = router.routes()
```

Файл posts.js

```
const Router = require('koa-router')
const passport = require('koa-passport')

const Post = require('../models/Post')

const router = new Router().prefix('/posts')

router.post('/', passport.authenticate('jwt', { session: false }), async
(ctx) => {
  const { body } = ctx.request.body
  const user = ctx.state.user._id
  ctx.body = await new Post({ body, user }).save()
  ctx.status = 201
})

router.get('/', async (ctx) => {
  const { query } = ctx
  const { skip, limit } = query
  delete query.skip
  delete query.limit
  const q = 'users' in query ?
    { user: { $in: query.users.split(',') } } : query
  ctx.set('x-total-count', await Post.countDocuments(q))
  ctx.body = await Post
    .find(q)
    .sort({ createdDate: -1 })
    .skip(+skip)
    .limit(+limit)
})

router.get('/:id', async (ctx) => {
```

```
  const post = await Post.findById(ctx.params.id)
  if (post) {
    ctx.body = post
  } else {
    ctx.throw(404, 'Post has not been found')
  }
})

router.put('/', passport.authenticate('jwt', { session: false }), async (ctx)
=> {
  const { _id, body } = ctx.request.body
  const user = ctx.state.user._id
  ctx.body = await Post.findOneAndUpdate(
    { _id, user },
    { $set: { body } },
    { new: true }
  )
})

router.delete('/:_id', passport.authenticate('jwt', { session: false }),
async (ctx) => {
  await Post.findOneAndRemove({
    _id: ctx.params._id,
    user: ctx.state.user._id
  })
  ctx.body = { message: 'Post has been deleted' }
})

module.exports = router.routes()
```

Файлsubscriptions.js

```
const Router = require('koa-router')
const passport = require('koa-passport')

const Subscription = require('../models/Subscription')

const router = new Router().prefix('/subscriptions')

router.post('/', passport.authenticate('jwt', {
  session: false
}), async (ctx) => {
  const { profile } = ctx.request.body
  const subscriber = ctx.state.user._id
  const checkSubscription = await Subscription.findOne({ subscriber, profile
})
  if (checkSubscription) {
    ctx.throw(400, 'You have alredy subscribed')
  }
  ctx.body = await new Subscription({ subscriber, profile }).save()
  ctx.status = 201
})

router.get('/', async (ctx) => {
  ctx.body = await Subscription.find(ctx.query)
})

router.delete('/:_id', passport.authenticate('jwt', {
  session: false
}), async (ctx) => {
  await Subscription.findOneAndDelete({
```

```
    _id: ctx.params._id,
    subscriber: ctx.state.user._id
  })
  ctx.body = { message: 'You was unsubscribed' }
})

module.exports = router.routes()
```

Файл users.js

```
const Router = require('koa-router')

const User = require('../models/User')

const router = new Router().prefix('/users')

router.get('/:_id', async (ctx) => {
  const user = await User.findById(ctx.params._id)
  if (user) {
    ctx.body = user
  } else {
    ctx.throw(404)
  }
})

module.exports = router.routes()
```

Файл body-parser.js

```
const bodyParser = require('koa-bodyparser')

module.exports = bodyParser()
```

Файл catch-mongoose-errors.js

```
const MongooseError = require('mongoose').Error

module.exports = async (ctx, next) => {
  try {
    await next()
  } catch(e) {
    if (e instanceof MongooseError) {
      ctx.throw(400, 'Bad credentials')
    } else {
      ctx.throw(e)
    }
  }
}
```

Файл errors.js

```
module.exports = async (ctx, next) => {
  try {
    await next()
  } catch(e) {
    ctx.status = e.status || 500
    ctx.body = { error: e.message || 'Internal Server Error' }
  }
}
```

Файл index.js

```
const bodyParser = require('./body-parser')
const errors = require('./errors')
const catchMongooseErrors = require('./catch-mongoose-errors')
const passportInit = require('./passport-init')
const static = require('./static')
```

```
module.exports = [
  bodyParser,
  errors,
  catchMongooseErrors,
  passportInit,
  static
]
```

Файл passport-init.js

```
const passport = require('koa-passport')
const passportConfig = require('../lib/passport-config')

passportConfig(passport)

module.exports = passport.initialize()
```

Файл static.js

```
const static = require('koa-static')

module.exports = process.env.NODE_ENV === 'production' ?
  static('client/build') : (ctx, next) => next()
```

Файл config.js

```
module.exports = {
  port: process.env.PORT || 8080,
  mongoUri: process.env.MONGO_URI,
  secret: process.env.SECRET || 'secret'
}
```

Файлmongoose-config.js

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

const config = require('./config');

module.exports = () => {
      mongoose
            .connect(config.mongoUri, {
                  useNewUrlParser: true,
                  useUnifiedTopology: true
            })
            .then(() => console.log('MongoDB has been connected'))
            .catch(e => console.log(e));
};
```

Файл passport-config.js

```
const { Strategy, ExtractJwt }  = require('passport-jwt')

const config = require('./config')
const User = require('../models/User')

const opts = {
  jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
  secretOrKey: config.secret
}

module.exports = (passport) => {
  passport.use(new Strategy(opts, async (payload, done) => {
    const user = await User.findById(payload.id)
    if (user) {
      done(null, user)
    } else {
```

```
      done(null, false)
    }
  }))
}
```

Файл Post.js

```
const mongoose = require('mongoose')
const Schema = mongoose.Schema

const postSchema = new Schema({
  body: {
    type: String,
    required: true
  },
  user: {
    type: Schema.Types.ObjectId,
    ref: 'users',
    required: true
  },
  likes: [
    {
      user: {
        type: Schema.Types.ObjectId,
        ref: 'users',
        required: true
      },
      createdDate: {
        type: Date,
        default: Date.now
      }
    }
  ],
  comments: [
    {
      body: {
        type: String,
        required: true
      },
      user: {
        type: Schema.Types.ObjectId,
        ref: 'users',
        required: true
      },
      createdDate: {
        type: Date,
        default: Date.now
      }
    }
  ],
  createdDate: {
    type: Date,
    default: Date.now
  }
})

const populationFields = 'user comments.user'

postSchema.post('save', async (doc) => {
  await doc.populate(populationFields).execPopulate()
})

function populateFields() {
  this.populate(populationFields)
}
```

```
postSchema.pre('find', populateFields)
postSchema.pre('findOne', populateFields)
postSchema.pre('findOneAndUpdate', populateFields)

module.exports = mongoose.model('posts', postSchema)
```

Файл Subscription.js

```
const mongoose = require('mongoose')
const Schema = mongoose.Schema

const subscriptionSchema = new Schema({
  subscriber: {
    type: Schema.Types.ObjectId,
    ref: 'users',
    required: true
  },
  profile: {
    type: Schema.Types.ObjectId,
    ref: 'users',
    required: true
  },
  createdDate: {
    type: Date,
    default: Date.now
  }
})

module.exports = mongoose.model('subscriptions', subscriptionSchema)
```

Файл User.js

```
const mongoose = require('mongoose')
const Schema = mongoose.Schema
const privatePaths = require('mongoose-private-paths')

const userSchema = new Schema({
  name: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true,
    unique: true
  },
  password: {
    type: String,
    required: true,
    private: true
  },
  createdDate: {
    type: Date,
    default: Date.now
  }
})

userSchema.plugin(privatePaths)

module.exports = mongoose.model('users', userSchema)
```

Файл app.js

```
const Koa = require('koa')

const config = require('./lib/config')
const handlers = require('./handlers')
const controllers = require('./controllers')
const mongooseConfig = require('./lib/mongoose-config')

const app = new Koa()

handlers.forEach((h) => app.use(h))

app.use(controllers.routes())
app.use(controllers.allowedMethods())

module.exports = (callback) => {
  mongooseConfig()
  app.listen(config.port, callback)
  return app
}
```

Файл index.js

```
require('dotenv').config()

const app = require('./app')
const config = require('./lib/config')

app(() => console.log(`Server has been started ${config.port}`))
```