МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
16 грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна технологія аналізу активності користувачів соціальних мереж»
здобувача групи ІН.м-25 Счастлівцева Сергія Миколайовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Сергій СЧАСТЛІВЦЕВ
(підпис)

Керівник,
в.о. завідувача кафедри,
кандидат технічних наук, доцент Ігор ШЕЛЕХОВ _____
(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН.м-25 Счастлівцева Сергія Миколайовича

1. Тема роботи: «Інформаційна технологія аналізу активності користувачів соціальних мереж»
затверджую наказом по СумДУ від «16» грудня 2023 р. № _____
2. Термін здачі здобувачем кваліфікаційної роботи до 16 грудня 2023 року
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій і методів, що будуть використані для розробки системи. 3) Розробка інформаційної системи для аналізу активної користувачів соціальних мереж. 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз, постановка й формування завдань дослідження*		
2	*Виділення основних проблем і пошук шляхів їх вирішення*		
3	*Розробка інформаційної системи для аналізу активної користувачів соціальних мереж*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 74 стор., 15 рис., 1 додаток, 30 джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена розв’язанню важливої практичної задачі аналітики активності користувачів соціальних мереж та надання інформації про аудиторію з метою поліпшення стану бізнесу, який базується на соціальних мережах.

Об’єкт дослідження — процес аналізу активності користувачів соціальних мереж.

Мета роботи —розробка інформаційного сервісу, який має аналізувати активність користувачів соціальних мереж та надавати відповідну звітність.

Методи дослідження — алгоритми аналізу отриманих даних і побудова статистики на їх основі.

Результати — розроблено інформаційну систему, яка надає можливість аналізувати активність користувачів соціальних мереж, оброблювати отримані дані, на їх основі будувати графіки та надавати інформацію у зручному вигляді про стан акаунту користувача. Проведено тестування на популярних соціальних мережах.

ІНФОРМАЦІЙНА СИСТЕМА, АНАЛІЗ ДАНИХ, ПОБУДОВА ГРАФІКІВ, TYPESCRIPT, REACT, NEST.JS, MONGODB.

ЗМІСТ

ВСТУП

Актуальність. Якщо говорити про актуальність цієї роботи, то можна виділити кілька найголовніших чинників.

По-перше, це звичайно, демонстрація знань і вмінь, здобутих під час навчання. Проектування і розробка подібного сервісу може продемонструвати всі вивчені вміння та навички.

По-друге, оскільки зараз соціальні мережі є також способом заробітку, ведення бізнесу та іншого, багатьом людям справді бракує зручного сервісу, який може в простій зручній формі запропонувати дані про ваш акаунт. Йдеться не про будь-які персональні дані, йдеться, звісно, про статистику.

Об’єкт дослідження. Процес аналізу активності користувачів соціальних мереж.

Предмет дослідження. Методи для дослідження аналізу активності користувачів соціальних мереж.

Гіпотеза. Якісного аналізу активності користувачів соціальних мереж можна досягти шляхом створення інформаційної системи, яка у зручному вигляді надає необхідні розрахунки на основі наданих даних.

Наукова новизна. На відміну від існуючих аналогів інформаційних систем, описане у даній роботі програмне рішення дозволить надавати якісну аналітику по різноманітним питанням, які стосуються активності аудиторії акаунту, надавати графіки та розрахунки у зручному вигляді.

Структура. Дане робота складається зі вступу, аналізу проблем та шляху їх вирішення, постановки задачі дослідження, вибір методики та інструментів для рішення поставленої проблеми, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз вимог до майбутнього сервісу

На цьому етапі найголовніше зрозуміти цільову аудиторію майбутнього продукту. У нашому випадку, це ті люди, які будуть займатись підтримкою соціальної мережі та безпосередньо самі користувачі, наприклад, власники інтернет-магазинів, які базуються на якійсь соціальній мережі. Наприклад, останніми роками, дуже велику популярність набули магазини у соціальній мережі Instagram. На разі, з’являються пропозиції на ринку праці з просування подібних акаунтів, їх підтримки та менеджменту. І як раз, у питанні просування акаунту, майбутній сервіс зможе допомогти.

Рисунок 1.1 – Приклад вакансії, яка містить роботу з акаунтом у Instagram [1]

Отже, першою вимогою до сервісу є надання інформації про найпопулярніші пости, теги та теми у стислому та лаконічному вигляді. Це повинно займати мінімальну кількість часу для ознайомлення та розуміння. Якщо, розглядається питання корисності, то за допомогою такої інформації користувач зможе зрозуміти, які позиції є найбільш актуальними і зможе коригувати свій бізнес план на основі отриманих даних.

Другою вимогою є аналіз аудиторії. Сюди можуть входити багато факторів, але найбільш потрібні нам це: місцезнаходження, вікові рамки аудиторії, час, який користувачі проводять на сторінці. Ці дані допоможуть, будувати більш підходящий контент для відповідної аудиторії. Очевидно, що якщо у користувача є інтернет-магазин, або він створює розважальний контент, то йому потрібно опиратись на подібні дані. Не може контент або продукція для людини похилого віку подобатись підліткам і навпаки. У будь-якому разі, у більшості випадків.[2]

Третьою вимогою є побудова графіків, які будуть корисні користувачу у його діяльності. Цей пункт є альтернативою до першого. Ми повинні брати до уваги різноманітні потреби користувачі, одні люди сприймають інформацію краще у вигляді тексту, інші у вигляді таблиць, інші – у вигляді графіків. Таким чином, ми зможемо покрити різні потреби клієнта і надавати якісні послуги у цій сфері.

Четвертою вимогою є аналіз фідбеку користувачів. Сервіс повинен у процентному співвідношенні надавати загальний фідбек по акаунту користувача, фідбек по окремому посту та фідбек за якийсь проміжок часу. У будь-якого користувача соціальних мереж, який веде за їх допомогою бізнес, виникає проблема з тим, що багато часу витрачається на перегляд коментарів. І на разі, я не зустрічав, щоб соціальні мережі надавали статистику у форматі: «Позитивні: N%, негативні: K%». Це дозволить зберегти багато часу.

Останньою вимогою є надання статистики акаунту у певний проміжок часу. Сюди необхідно винести дані про зміни кількості підписників у той, чи інший період часу, найбільш затребуваний та нецікавий контент за цей час та посилання

на дані про місцезнаходження, вік та час проведений на сторінці нових користувачів.

1.2 Аналіз сучасних сервісів, які надають можливість аналізу діяльності користувачів соціальних мереж

На даний момент існує декілька основних сервісів, які допомагають користувачам аналізувати активність своєї аудиторії у соціальних мережах. Це Hootsuite, Buffer, Sprout Social, Google Analytics.

Отже, необхідно детально розібрати кожен інструмент, виділити плюси та мінуси.

Таблиця 1.1 – Виділення плюсів і мінусів сервісу Hootsuite

Hootsuite	
Плюси	Мінуси
Керування декількома соціальними мережами	Безкоштовна версія є обмеженою у функціоналі
Планування постів	Можливі затримки у оновленні даних
Командна взаємодія	Деякі соціальні мережі не підтримуються
Моніторинг повідомлень та відміток	Складний інтерфейс для нових користувачів

Отже, у таблиці 1.1 наведено коротка характеристика сервісу Hootsuite. Сервіс є доволі функціональним, там має багато цікавих рішень, що робить його одним із найкращих у своїй сфері. Але варто розуміти, що найбільшою їх проблемою є складність інтерфейсу для нових користувачів. Тому, є дуже важливо, розробити простий і доступний UI, щоб не повторити помилок Hootsuite.

Таблиця 1.2 – Виділення плюсів і мінусів сервісу Buffer

Buffer	
Плюси	Мінуси
Простота інтерфейсу	Безкоштовна версія є обмеженою у функціоналі
Планування постів	Функціонал значно менший ніж у конкурентів
Наявність плагіну для браузера	Деякі соціальні мережі не підтримуються
Можливість роботи у команді	-

Отже, у таблиці 1.2 наведена характеристика сервісу Buffer. У цьому випадку, сервіс має дійсно простий і зрозумілий інтерфейс, і тому набрав широку популярність. Але, суттєвим мінусом є менший функціонал. Більш просунуті користувачі, зможуть виділити час і розібратись із більш складним сервісом, якщо той, у свою чергу, надає більш детальну статистику.

Таблиця 1.3 – Виділення плюсів і мінусів сервісу Sprout Social

Sprout Social	
Плюси	Мінуси
Інтеграція з великою кількість соціальних мереж	Безкоштовна версія є обмеженою у функціоналі
Керування активністю соціальних мереж	Високий поріг входу
Детальна звітність по різноманітним показникам	Вартість
Можливість роботи у команді	Недостатність звітності у деяких соціальних мережах

Отже, у таблиці 1.3 представлено характеристику сервісу Sprout Social. Сервіс, дійсно, надає досить детальну звітність і має велику кількість функцій. Але, вартість та високий поріг входу на дають йому зайняти лідируючі позиції на ринку. Варто, враховувати ці показники при розробці власного сервісу.

Таблиця 1.4 – Виділення плюсів і мінусів сервісу Google Analytics

Google Analytics	
Плюси	Мінуси
Безкоштовність	Перше налаштування є складним
Широкий функціонал по різноманітним напрямам	Помилки у відстежуванні даних
Детальна звітність по різним показникам	Звітність з деяких питань є неповною
Гарна інтеграція з іншими сервісами Google	-

Отже, у таблиці 1.4 представлена характеристика Google Analytics. Сервіс набув популярності через повну інтеграцію з сервісами Google, бо це їх власна розробка і, також, він є безкоштовним, що важливо.

Підсумовуючи, можна сказати, що у всіх аналогів, є помилки одного і того роду. Тобто, в багатьох випадках, це складність, складний UI, недостатня кількість даних у звітності. Оперуючи цими даними, можна підійти до розробки сервісу з іншого боку, і не повторити помилок аналогів.

1.3 Проблеми інтеграції з соціальними мережами

У ході роботи, виникає розуміння наступної проблеми. Робота усіх нині популярних соціальних мереж є в основному конфіденційною. Тобто неможливо просто взяти і отримати дані про діяльність того чи іншого користувача.

Деякі соціальні мережі надають деякі рівні доступу. Наприклад, авторизація

на сторонніх сервісах за допомогою Facebook або Twitter. Але, знову ж таки, в цьому випадку соціальна мережа поверне дані про користувача, тільки у випадку вдалої авторизації. І, звичайно, це будуть дані самого користувача.

Отже, нашим найкращим варіантом, є використання офіційної API соціальних мереж. Наприклад, така API є у Instagram. Ми можемо отримувати дані про користувачів, кількість підписників, публікації та коментарі до них та багато чого іншого. Для початку роботи з цим інструментом, звичайно, необхідно ознайомитись з політикою конфіденційності компанії, з якою планується взаємодія.

Також, вірним рішенням, буде налаштувати авторизацію у нашому сервісі за допомогою обраної соціальної мережі, якщо вона дає таку можливість. Це дозволить отримати базові права доступу до даних.

1.4 Обробка даних отриманих від соціальних мереж

Наступним питанням, яке постає під час проектування сервісу є яким чином, ми будемо обробляти дані, які отримуємо від API соціальної мережі.

Проблема є у тому, що якщо цільовий акаунт є популярним, від API будуть приходити величезні об’єми даних. Наша мета, оброблювати їх таким чином, щоб користувачу не доводилось чекати декілька хвилин після кожного запиту.

Логічним рішенням, у такому випадку є відображення результату частинами. Тобто, після того, як нам прийшли дані, ми одразу починаємо динамічно відображати дані, не чекаючи, поки отримаємо всю колекцію. Це дозволить створить зручний інтерфейс.

Але, такий підхід не є гарним, коли мова заходить про графіки та діаграми. Проблема у тому, що для них необхідно мати повний об’єм даних, щоб відобразити коректну статистику. Вирішенням цього питання є надання побудові графіків динамічності. Так би мовити, графік буде будуватись на очах користувача. Таким чином, ми надаємо сервісу динамічності, що є важливим у сучасному світі, та вирішимо нашу проблему. [3]

1.5 Вирішення питання з аналізом фідбеку

Насправді, це питання є найскладнішим у цій роботі. Але, я виділив декілька шляхів її вирішення, від простих до більш просунутих.

Найбільш очевидними варіантами є:

- Створення окремого сервісу для аналізу найбільш вживаних фраз у коментарях, які можуть позитивно або негативно описувати якусь сутність;
- Аналіз постів за вподобаннями та переглядами;
- Використання AI-технологій.

Нижче, буде наведено таблицю, де детально буде описано, плюси та мінуси можливих підходів для вирішення проблеми:

Таблиця 1.5 – Виділення плюсів і мінусів використання AI-технологій для аналізу фідбеку користувачів

Метод	Використання AI-технологій
Плюси	Швидкість, аналіз великих обсягів даних, можливість делегувати рутинний процес
Мінуси	Нестабільність, можлива недостатня прозорість, не завжди зможе зрозуміти те, що людина мала на увазі, можлива відсутність API для взаємодії
Обґрунтування	Штучний інтелект, частіше за все, видає вірну відповідь, робить це швидко і, як мінімум, дає фундамент, на якому вже можна побудувати щось більше. Він дійсно може перейняти на себе важкі процеси, але ризиків досить багато. Варто враховувати, що штучний інтелект писали люди, тому, звісно, він не є панацеєю. Він може помилитись, дати невірні дані, щось сплутати. Також постає проблема, як використати його відповідь для побудови графіку або діаграми, яка чітко відобразить ситуацію для користувача.

Таблиця 1.6 – Виділення плюсів і мінусів створення окремого сервісу для аналізу фідбеку користувачів

Метод	Окремий сервіс для аналізу
Плюси	Повна передбачуваність та контроль над сервісом, можливість написати тести для всього функціоналу
Мінуси	Високі затрати у часі, неможливо покрити усі кейси
Обґрунтування	З одного боку, написання власного сервісу є гарною ідеєю. Ми отримуємо повний контроль над розробкою і отримуємо повністю передбачуваний продукт. Ми можемо написати unit-тести до нього і уникнути помилок. Але це не є гарною задумкою, коли часу для реалізації недостатньо. Призначення сервісу досить складне, бо потрібно покрити неймовірно широку кількість варіантів, до того ж адаптувати це на різні мови.

Таблиця 1.7 – Виділення плюсів і мінусів аналізу постів за вподобаннями і переглядами для надання аналізу фідбеку користувачів

Метод	Аналіз постів за вподобаннями і переглядами
Плюси	Не потребує складних розрахунків та витрат часу, інструментарій надає API соціальної мережі
Мінуси	Важко судити про фідбек користувачів, використовуючи лише ці дані, можливі деякі неточності.
Обґрунтування	В цілому, цей варіант є найбільш релевантним для нас. Звичайно, якщо брати до уваги те, що ми не просто виводимо окремі цифри вподобань і переглядів, а і проводимо розрахунки. Ми можемо виділити з переглядів відсоток людей, який вподобали пост. На основі цих даних сформувати загальний фідбек. Можна додатково виділити границі вимірювання. Наприклад: - 5-15% - фідбек негативний; - 20-30% - фідбек змішаний; - 40+% - фідбек позитивний. Відсотки варто вимірювати нижче 50%, бо як показує практика, більшість людей просто переглядають пости.

1.6 Можливі технології та сервіси для роботи:

Перш за все, це API соціальної мережі. Можна використати API компанії Meta, яке надає інформацію Facebook та Instagram. Також можна використати власну реалізацію. Невеличкої соціальної мережі для тестування необхідного функціоналу буде достатньо.

Далі необхідно попрацювати над серверною реалізацією. Тут буде використано Node.js або Nest.js як фреймворк для роботи з сервером та MongoDB як база даних. Оскільки, ми не маємо на меті створення інформаційної технології для розгортання та отримання прибутку, нереляційної MongoDB буде достатньо для реалізації наших потреб.

На клієнтській частині додатку буде використано React + TypeScript. Можливості JavaScript, який є основою цієї зв'язки буде достатньо для реалізації. TypeScript, у свою чергу додасть нашому коду передбачуваності та збереже від небажаних помилок.

1.7 Постановка задачі

Метою роботи є розробка інформаційного сервісу, який має аналізувати активність користувачів соціальних мереж та надавати відповідну звітність. Для досягнення поставленої мети необхідно реалізувати наступні задачі:

1. Підібрати API соціальної мережі, яке покриє наші потреби.
2. Реалізувати повну серверну взаємодію між API соціальної мережі та нашим сервером
3. Розробити мінімалістичний інтерфейс клієнтської частини додатку
4. Створити функціонал динамічного створення графіків та діаграм на основі даних від API
5. Відображати дані динамічно з мінімальними затримками користувача
6. Покрити код клієнтської та серверної частини unit-тестами, необхідно мінімізувати кількість непередбачуваних подій

7. На основі вподобань і переглядів постів формувати процент позитивності фідбеку
8. Надавати користувачу усю необхідну інформацію про аудиторію акаунту

2 ВИБІР МЕТОДУ РІШЕННЯ

Для вирішення поставлених задач необхідна візуалізація процесу:

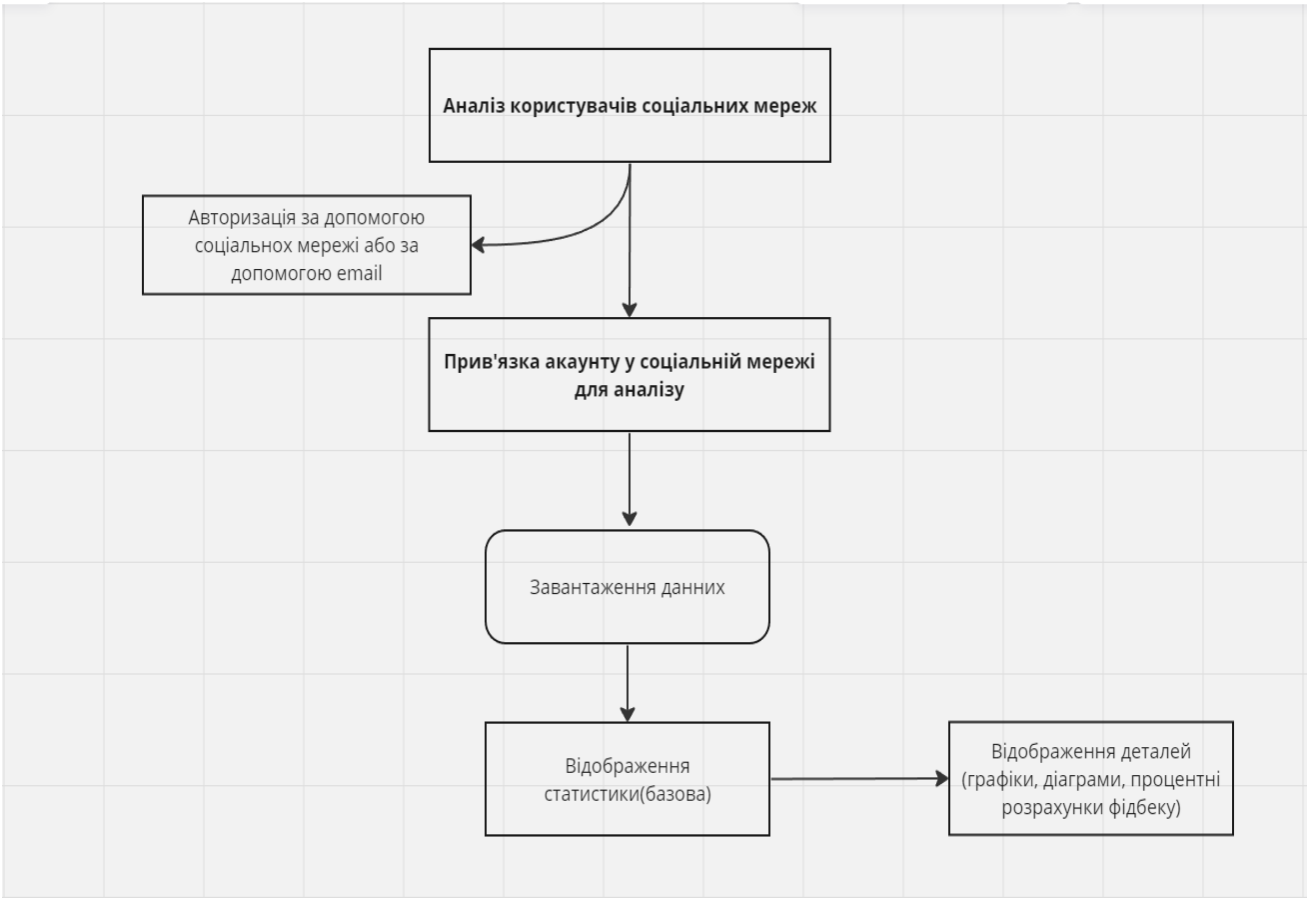


Рисунок 2.1 – Базові можливості сервісу[4]

На рисунку 2.1 відображені базові можливості користувача. Для початку, необхідно пройти авторизацію. Це можна зробити шляхом прив’язки акаунту соціальної мережі або через електронну пошту.

Варто зазначити, що якщо вибрана електронна пошта, то потім додатково необхідно обрати, з якою саме соціальною мережею буде вестись робота. Після цього йде етап завантаження даних, підрахунки та побудова графіків, діаграм та процентні розрахунки фідбеку.

Це можна робити у фоновому режимі і показати на екрані, лише тоді, коли користувач натисне відповідну кнопку.

2.1 Остаточний стек технологій для реалізації

Клієнтська частина:

React, TypeScript, Axios, react-router-dom, Redux-Toolkit – основні технології для розробки клієнтської частини. Також необхідно враховувати, що це лише база, тобто найважливіші інструменти, які будуть використовуватись. Поряд із ними будуть також більш специфічні бібліотеки, які надають можливість швидко та ефективно вирішити ту чи іншу проблему.

Таблиця 2.1 – Опис технологій для розробки клієнтської частини

React [5]	Бібліотека для створення користувацьких інтерфейсів. Дозволяє створювати швидкі та ефективні додатки не займаючи фінальний проект великою кількістю непотрібних інструментів
Axios [6]	Бібліотека, яка полегшує написання запитів на сервер, підтримує усю повноту CRUD-операцій
React-router-dom [7]	Інструмент, який дозволяє створити SPA-додаток, за допомогою React
TypeScript[8]	Надбудова над JavaScript, яка додає типізацію. Без нього сучасна розробка неможлива, бо завжди необхідно забезпечити уникання великої кількості непередбачуваних помилок
Redux Toolkit[9]	Його можна назвати сховком для даних на клієнтській частині. Він дозволяє ефективно маніпулювати великими об’ємами даних дуже зручно
Styled Components[10]	Зручна бібліотека для написання стилів на клієнтській частині додатку
React-redux[11]	Провайдер, який забезпечує сумісну роботу React та Redux Toolkit
React Testing[12] Library	Бібліотека для тестування React компонентів. Забезпечує написання тестів для збереження відображення компонентів, при зміні коду
Jest[13]	Бібліотека для написання unit-тестів
TS-Jest [14]	Бібліотека, що полегшує роботу з TypeScript і Jest

У таблиці 2.1 наведено список основних технологій, які будуть використані при розробці клієнтської частини. Також наведено невеличкий опис технологій, щоб розуміти роль кожної у процесі.

Серверна частина:

Таблиця 2.2 – Опис технологій для розробки серверної частини

Nest.js[15]	Фреймворк для створення серверної частини, має досить великі можливості і дає змогу створювати додатки під будь-які задачі
MongoDB [16]	Нереляційна база даних, яка надає дані у схожому на JSON форматі
Passport-jwt[17]	Необхідний для аутентифікації користувача з jsonwebtoken
Bctypt [18]	Дозволяє зберігати дані у БД у захешованому вигляді
Dotenv [19]	Необхідний для використання змінних з process.env
jsonwebtoken[20]	Бібліотека для створення токенів на серверній частині
Mongoose[21]	Бібліотека для запитів до бази даних MongoDB

У таблиці 2.2 наведено список основних технологій, які будуть використані при розробці серверної частини. Також наведено невеличкий опис технологій, щоб розуміти роль кожної у процесі.

Загальна частина:

Далі буде наведено список обов’язкових бібліотек, що будуть використані і у клієнтській частині, так і в серверній. Вони, у більшості, слугують для підтримки style-guides для коду, щоб він був однаковий на всіх пристроях. Також, частина із них відповідає за якість коду.

Таблиця 2.3 – Опис технологій для розробки серверної частини

Eslint [22]	Бібліотека для підтримання єдиних правил коду у всьому сервісі
Prettier [23]	Бібліотека для підтримання єдиного стилю коду у всьому сервісі
Husky[24]	Пре-комміт хук, який відпрацьовує перед комітом і перевіряє, щоб зміни у коді були згідно правил
Eslint-plugin-prettier [25]	Бібліотека, яка нормалізує взаємодію Eslint та Prettier. Зменшує можливість конфліктів правил
eslint-plugin-simple-import-sort [26]	Бібліотека для автоматичного сортування імпортів при збереженні файлу
eslint-import-resolver-typescript [27]	Бібліотека, яка надає є базовим налаштуванням Eslint та TypeScript для імпортів.
eslint-plugin-import [28]	Бібліотека, яка надає змогу розробити правила для імпортів, щоб вони були у єдиному стилі

У таблиці 2.3 наведено кінцевий список загальних інструментів, який буде використано при розробці.

2.2 Конфігурація бази даних для роботи з сервісом

Таблиця 2.4 – 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 2.4 представлена модель користувача. При авторизації або реєстрації ми будем зберігати ім’я, пошту та пароль користувача у захешованому вигляді, для безпеки.

Таблиця 2.5 – Поля моделі UserSocialMedia для зберігання у MongoDB

name	Type: String, require: true
connectionDate	Type: Date, default: “Date now”
url	Type: String, require: true, private: true

У таблиці 2.5 представлена модель соціальної мережі, до якої підключений користувач. Тобто, аналіз якої мережі буде проводитись.

Нам необхідні назва мережі, дата підключення та посилання, яке буде фігурувати у всіх майбутніх запитах до API.

Таблиця 2.6 – Поля моделі Post для зберігання у MongoDB

PostTitle	Type: String, require: true
createdDate	Type: Date, default: “Date now”
url	Type: String, require: true, private: true
usersLikes	Type: number, require: true
usersViews	Type: number, require: true
author	Type: Schema.Types.ObjectId, Ref: “users”, required: true

У таблиці 2.6 представлена модель для зберігання поста у базі даних. Об’єкти подібного типу будуть поміщені у масив, щоб мати змогу отримати цілу

колекцію постів користувача для аналізу.

У кожного поста необхідними для нас полями є заголовок, автор, посилання, кількість вподобань та переглядів для аналізу.

2.2 Остаточний вибір API соціальної мережі

Для наших задач нам ідеально підходить API, яке надає компанія Meta – Meta for Developers[29].

До переваг можна віднести, що нам будуть доступні авторизація за допомогою Facebook, яку буде працювати на будь-яких пристроях. Також, у нас буде доступ до постів, користувачів, вподобань та інших необхідних нам параметрів як із Facebook, так і з Instagram.

Також, Meta надає зручні інструменти для роботи JavaScript з їх API, що ідеально для нас підходить. Оскільки компанія є всесвітньо відомою, можна не сумніватись у якості наданого API. Це гарантує, що проблем зі сторони API не буде, що є дуже важливим показником.

У більшості випадків, нам буде достатньо цього пункту у документації:

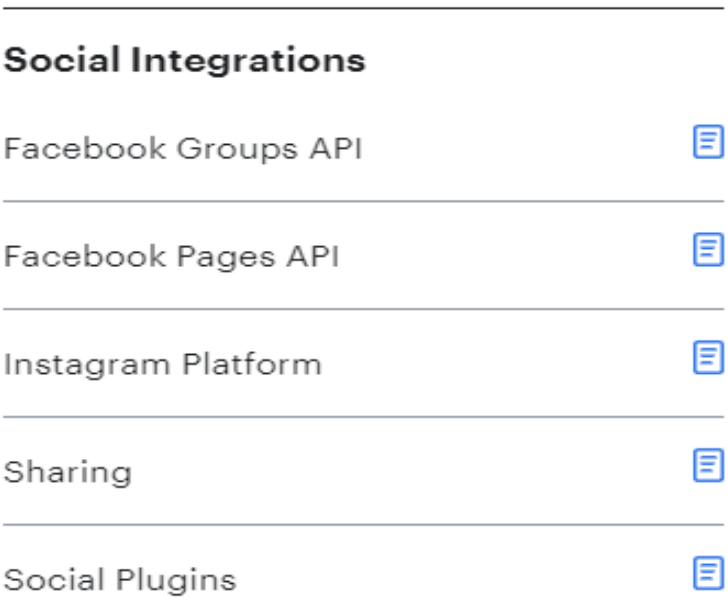
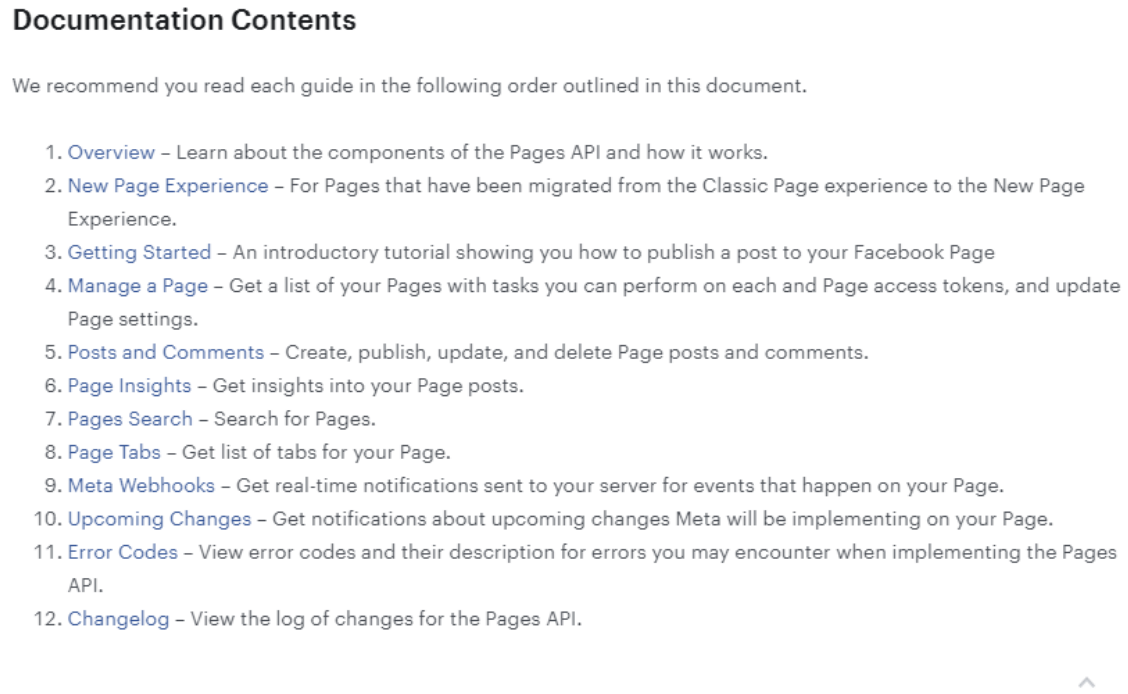


Рисунок 0.2 – Основні необхідні пункти документації Meta for Developers[29]

На рисунку 2.2 показано, що ми матимемо доступ до платформи Instagram та Facebook.



Documentation Contents

We recommend you read each guide in the following order outlined in this document.

1. Overview – Learn about the components of the Pages API and how it works.
2. New Page Experience – For Pages that have been migrated from the Classic Page experience to the New Page Experience.
3. Getting Started – An introductory tutorial showing you how to publish a post to your Facebook Page
4. Manage a Page – Get a list of your Pages with tasks you can perform on each and Page access tokens, and update Page settings.
5. Posts and Comments – Create, publish, update, and delete Page posts and comments.
6. Page Insights – Get insights into your Page posts.
7. Pages Search – Search for Pages.
8. Page Tabs – Get list of tabs for your Page.
9. Meta Webhooks – Get real-time notifications sent to your server for events that happen on your Page.
10. Upcoming Changes – Get notifications about upcoming changes Meta will be implementing on your Page.
11. Error Codes – View error codes and their description for errors you may encounter when implementing the Pages API.
12. Changelog – View the log of changes for the Pages API.

Рисунок 0.3 – Доступний контент для Facebook[30]

На рисунку 2.3 показано доступний контент, який надає Meta для Facebook. Деякі можливості недоступні, але тому нам знадобляться одразу 2 соціальні мережі, щоб показати у роботі весь можливий функціонал.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Створення оточення для роботи на серверній частині

Для початку, необхідно підготувати все необхідне для роботи на сервері. Налаштувати лінтери, підключити базу даних, створити базовий шаблон, який буде працювати.

Отже, після створення базового шаблона проекту, маємо таку структуру:

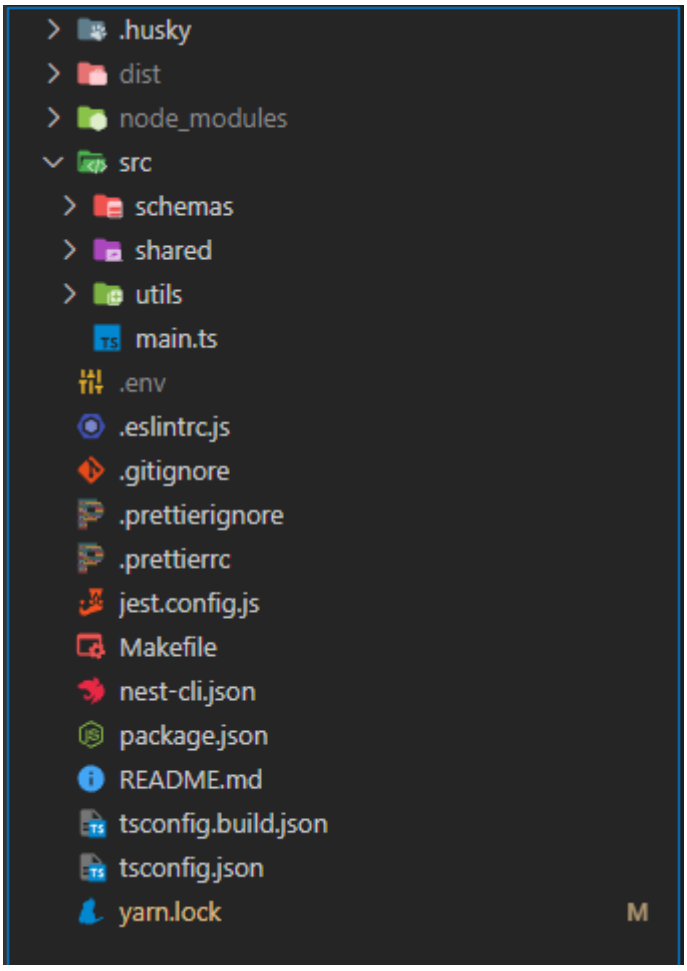


Рисунок 3.1 – Структура проекту серверної частини

Отже, на рисунку 3.1 зображена структура проекту серверної частини. Це базова структура, яку рекомендує документація Nest.js.

Далі необхідно налаштувати правила для Eslint, Prettier та Husky.

```
{
  "tabWidth": 4,
  "singleQuote": true,
  "trailingComma": "es5",
  "requirePragma": false,
  "arrowParens": "always",
  "printWidth": 160
}
```

Такі налаштування для були обрані для Prettier. Це дозволить зберегти код серверної частини у єдиному стилі.

```
module.exports = {
  plugins: ['@typescript-eslint', 'prettier', 'import'],
  extends: [
    'eslint:recommended',
    'plugin:@typescript-eslint/recommended',
    'plugin:prettier/recommended',
    'plugin:import/recommended',
    'plugin:import/typescript',
  ],
  settings: {
    'import/resolver': {
      typescript: {
        project: './tsconfig.json',
      },
    },
  },
  rules: {
    '@typescript-eslint/no-var-requires': 0,
    'import/no-unresolved': 'warn',
    'import/order': [
      'error',
      {
        groups: ['builtin', 'external', 'internal', ['sibling', 'parent'], 'index', 'unknown'],
        'newlines-between': 'always',
        alphabetize: {
          order: 'asc',
          caseInsensitive: true,
        },
      },
    ],
  },
```

Вище наведено налаштування для Eslint. Це дасть можливість

використовувати єдині правила для написання коду на серверній частині.

3.2 Створення оточення для роботи на клієнтській частині

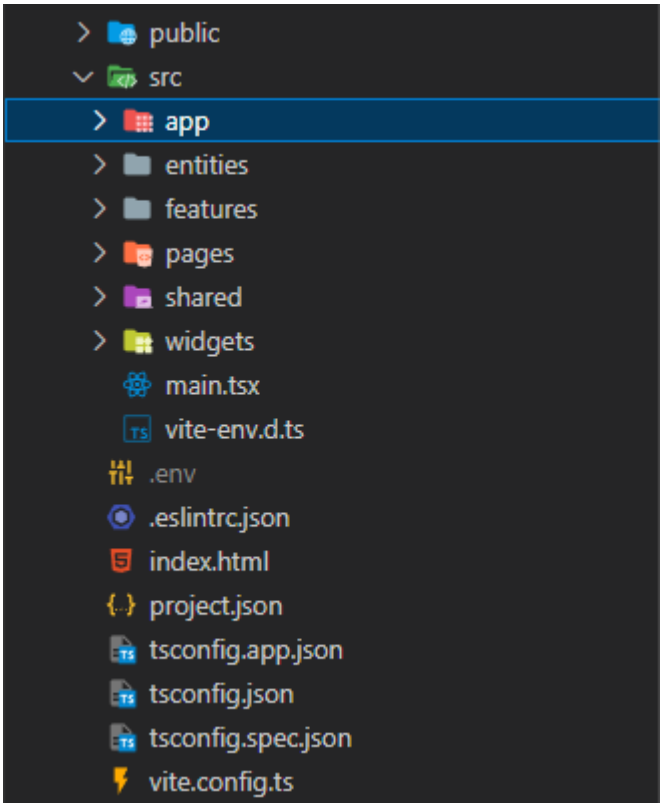


Рисунок 3.2 – Структура проекту клієнтської частини

Отже, на рисунку 3.2 продемонстровано структуру проекту клієнтської частини. Для формування було використано методологію feature-slices-design. Вона дозволяє створювати проекти різноманітних розмірів та не хвилюватись, що буде втрачено контроль над кодом проекту.

Структура Eslint та Prettier така ж, як і на серверній частині.

3.3 Конфігурація бази даних MongoDB

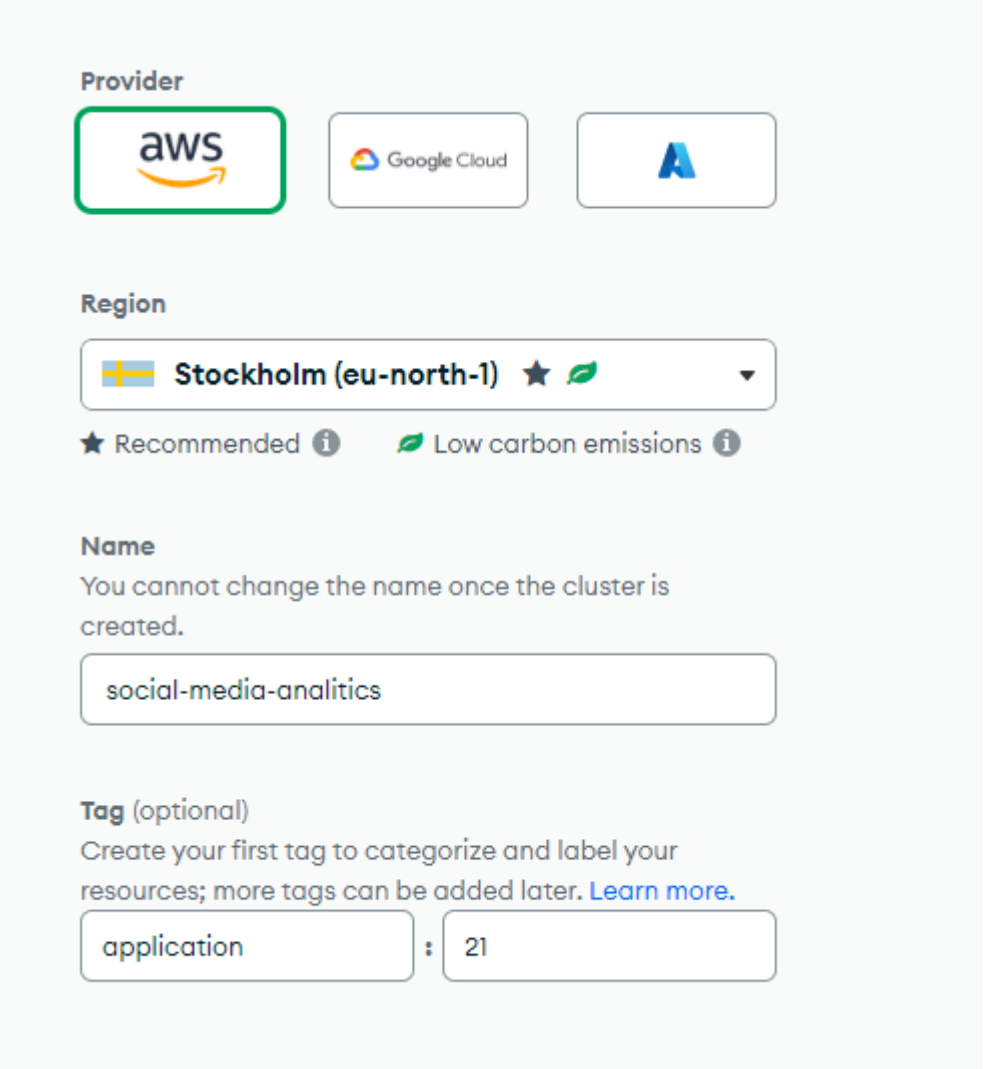


Рисунок 3.3 – Створення кластеру MongoDB

Отже, на рисунку 3.3 продемонстровано створення кластеру MongoDB. Необхідно вибрати регіон, який найбільш близько знаходиться до України, щоб уникнути зависань бази даних.

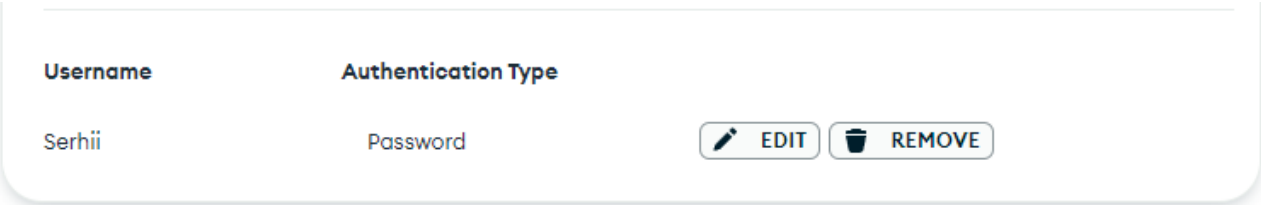


Рисунок 3.4 – Додавання користувача

На рисунку 3.4 показано доданого користувача, який буде використовуватись при з'єднанні з базою даних.

Рисунок 3.5 – Додавання IP-адреси

На рисунку 3.5 показано додану IP-адресу. Сюди можна додати декілька адрес, і тільки з них можна буде відправляти запити до бази даних.

На цьому підготовка оточення закінчена.

3.4 Розробка серверної частини

```
async function bootstrap() {
  const app = await NestFactory.create<NestExpressApplication>(AppModule);
  app.enableCors({
    origin: '*',
  });
  app.setBaseViewsDir(join(__dirname));
  app.setViewEngine('hbs');

  Swagger(app);

  // app.use(TenantMiddleware);
  await app.listen(PORT);

  Promise.resolve(app).then(() => console.log('[Nest]: Application is running'));
}
bootstrap();
```

У коді вище представлений main.ts файл. У ньому відбувається конфігурація серверної частини.

Далі необхідно сформувати схеми.

```
import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
import { Date, Document } from 'mongoose';

@Schema({ strict: false })
export class User {
  @Prop({ required: true, type: String })
```

```
  name: string;

  @Prop({ required: true, type: String })
  email: string;

  @Prop({ required: true, type: Date })
  password: Date;

  @Prop({ required: true, type: Date })
  createdDate: Date;
}

export type UserDocument = User & Document;

export const EventSchema = SchemaFactory.createForClass(User);
```

У коді вище додано схему користувача. Він матиме поля: name, email, password, createdDate.

```
import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
import { Document } from 'mongoose';

@Schema({ collection: 'usersocialmedia' })
export class UserSocialMedia {
  @Prop({ required: true, type: String })
  name: string;

  @Prop({ required: true, type: Date })
  connectionDate: Date;

  @Prop({ type: String, required: true })
  url: string;
}

export type UserSocialMediaDocument = UserSocialMedia & Document;

export const ProductTypeSchema = SchemaFactory.createForClass(UserSocialMedia);
```

Наступним кром є додання схеми для соціальних мереж користувача. Необхідними полями є name, connectionDate, url.

```
import { Prop, SchemaFactory, Schema } from '@nestjs/mongoose';
import { Document } from 'mongoose';

@Schema({ strict: false })
export class Post {
  @Prop({ required: true, type: String })
```

```
    title: string;

    @Prop({ required: true, type: Date })
    createdDate: Date;

    @Prop({ required: true, type: String })
    url: string;

    @Prop({ required: true, type: Number })
    userLikes: number;

    @Prop({ required: true, type: Number })
    usersViews: number;

    @Prop({ required: true, type: Object })
    author: Object;
}

export type PostDocument = Post & Document;

export const PostSchema = SchemaFactory.createForClass(Post);
```

Далі на черзі схема посту. Обов'язковими полями є title, createdDate, url, usersLikes, usersViews, author.

Далі необхідно написати модулі для різних функцій. Спочатку авторизація користувача у сервісі.

```
@Injectable()
export class AuthService {
 constructor(
  private usersService: UsersService,
  private jwtService: JwtService,
 ) {}

 async validateUser(username: string, password: string): Promise<any> {
  const user = await this.usersService.findByUsername(username);

  if (user && (await bcrypt.compare(password, user.password))) {
   const { password, ...result } = user;
   return result;
  }

  return null;
 }

 async login(user: User) {
  const payload = { username: user.username, sub: user.userId };
  return {
   access_token: this.jwtService.sign(payload),
```

```
  };
 }
}
```

Вище вказано клас AuthService. У цьому сервісі ми перевіряємо чи є користувач у нашій системі. У випадку позитивної відповіді від бази даних, відбувається авторизація.

```
async validate(username: string, password: string): Promise<any> {
  const user = await this.authService.validateUser(username, password);

  if (!user) {
   return Promise.reject('Invalid credentials');
  }

  return user;
 }
```

Також варто обробити випадок, коли користувача у базі даних не знайдено і повертати на фронтенд відповідну інформацію.

Далі, нам необхідно описати логіку отримання соціальних мереж, які обрав користувач.

```
@Controller('social-networks')
export class SocialNetworksController {
 constructor(private usersService: UsersService) {}

 @UseGuards(JwtAuthGuard)
 @Post()
 async updateSocialNetworks(@Request() req, @Body() socialNetworksDto: SocialNetworksDto) {
  const userId = req.user.userId;
  const networks = socialNetworksDto.networks;

  await this.usersService.updateSocialNetworks(userId, networks);

  return { success: true, message: 'Social networks added successfully' };
 }
}
```

У коді вище представлено контролер для отримання списку соціальних мереж користувача. У цьому коді ми отримуємо даний список, далі передаємо його у відповідний сервіс і після цього зберігаємо у базі даних.

Далі для аналізу постів та збереження даних про них, необхідно описати це в коді.

```
@Controller('posts')
export class PostsController {
 constructor(private postsService: PostsService) {}

 @UseGuards(JwtAuthGuard)
 @Post()
 async createPost(@Request() req, @Body() createPostDto: CreatePostDto) {
  const userId = req.user.userId;

  const post = await this.postsService.createPost(userId, createPostDto);

  return { success: true, post };
 }
}
```

Отже, у цьому контролері ми отримуємо пости і після цього переходимо до сервісу, який їх обробляє.

```
export class PostsService {
 constructor(
  @InjectRepository(Post)
  private postRepository: Repository<Post>,
 ) {}

 async createPost(userId: number, createPostDto: CreatePostDto): Promise<Post> {
  const { title, content } = createPostDto;

  const post = this.postRepository.create({
   title,
   content,
   userId,
  });

  await this.postRepository.save(post);

  return post;
 }
}
```

У цьому сервісі, ми проводимо дані до потрібного формату та зберігаємо інформацію про пост у базу даних.

3.5 Розробка клієнтської частини

Для початку необхідно виконати необхідні міри для авторизації користувача. Оскільки було вирішено додати декілька варіантів авторизації, необхідно підготувати відповідні компоненти.

Для початку необхідно описати стилі:

```
const AuthWrapper = styled.div`
  max-width: 400px;
  margin: 0 auto;
  padding: 20px;
  border: 1px solid #ddd;
  border-radius: 8px;
`;

const Input = styled.input`
  width: 100%;
  margin-bottom: 10px;
  padding: 8px;
`;

const Button = styled.button`
  width: 100%;
  padding: 10px;
  background-color: #007bff;
  color: #fff;
  border: none;
  border-radius: 4px;
  cursor: pointer;

  &:hover {
    background-color: #0056b3;
  }
`;
```

У коді наведені стилі для форми авторизації у додатку. Користувачу необхідно буде ввести email та пароль для авторизації.

```
interface AuthComponentProps {
  onLogin: (username: string, password: string) => void;
}

const AuthComponent: React.FC<AuthComponentProps> = ({ onLogin }) => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');

  const handleLogin = () => {
```

```
  onLogin(username, password);
 };

 return (
  <AuthWrapper>
   <h2>Login</h2>
   <label>
    Username:
    <Input
     type="text"
     value={username}
     onChange={(e) => setUsername(e.target.value)}
    />
   </label>
   <label>
    Password:
    <Input
     type="password"
     value={password}
     onChange={(e) => setPassword(e.target.value)}
    />
   </label>
   <Button onClick={handleLogin}>Login</Button>
  </AuthWrapper>
 );
};
```

Далі описуємо логіку авторизації. Зберігаємо у стейт відповідні дані з форми, після чого, вони будуть відправлені у store Redux Toolkit, а далі – на серверну частину.

Далі перейдемо до опису основних можливостей сервісу.

```
const PositiveFeedbackPercentage: React.FC<PositiveFeedbackPercentageProps> = ({
 likes,
 views,
}) => {
 const calculatePositivePercentage = () => {
  if (views === 0) {
   return 0;
  }

  const positivePercentage = (likes / views) * 100;
  return positivePercentage.toFixed(2);
 };

 return (
  <Wrapper>
   <LikeIcon>👍</LikeIcon>
   {`${calculatePositivePercentage()}% Positive Feedback`}
```

```
    </Wrapper>
  );
};
```

Вище представлено функцію, яка виконує розрахунки позитивного фідбеку користувачів на основі вподобань та переглядів.

```
const countryData = {
  labels: Object.keys(userCountryData),
  datasets: [
   {
    label: 'Users by Country',
    data: Object.values(userCountryData),
    backgroundColor: [
     'rgba(255, 99, 132, 0.6)',
     'rgba(54, 162, 235, 0.6)',
     'rgba(255, 206, 86, 0.6)',
     'rgba(75, 192, 192, 0.6)',
     // Добавьте дополнительные цвета по мере необходимости
    ],
   },
  ],
 };

 const ageData = {
  labels: Object.keys(userAgeData),
  datasets: [
   {
    label: 'Users by Age',
    data: Object.values(userAgeData),
    backgroundColor: 'rgba(75, 192, 192, 0.6)',
   },
  ],
 };

 const timeData = {
  labels: Array.from({ length: userTimeData.length }, (_, i) => i + 1),
  datasets: [
   {
    label: 'Time Spent by Users',
    data: userTimeData,
    fill: false,
    borderColor: 'rgba(75, 192, 192, 0.6)',
   },
  ],
```

Далі нам необхідно оброблювати дані про користувачів та надавати відповідні графіки на основі цих даних. Вище представлений код, для

формування графіків, на основі масиву з користувачами, які були отримані від соціальної мережі.

```
return (
  <div>
    <h2>Users by Country</h2>
    <Doughnut data={countryData} />

    <h2>Users by Age</h2>
    <Bar data={ageData} />

    <h2>Time Spent by Users</h2>
    <Line data={timeData} />
  </div>
 );
};
```

Вище представлено використання отриманих графіків для відображення на сторінці.

```
const SidebarContainer = styled.div<{ isOpen: boolean }>`
 position: fixed;
 top: 0;
 left: ${({ isOpen }) => (isOpen ? '0' : '-100%')};
 width: 250px;
 height: 100%;
 background-color: #333;
 padding-top: 60px;
 transition: left 0.3s ease-in-out;
`;

const CloseButton = styled.button`
 position: absolute;
 top: 20px;
 right: 20px;
 background: none;
 color: #fff;
 border: none;
 font-size: 20px;
 cursor: pointer;
`;

const MenuItem = styled.a`
 display: block;
 padding: 15px;
 color: #fff;
 text-decoration: none;
 font-size: 18px;

 &:hover {
```

```
    background-color: #555;
  }
`;

const Sidebar: React.FC<SidebarProps> = ({ isOpen, onClose }) => {
  const menuItems = ['Home', 'About', 'Services', 'Contact'];

  return (
    <SidebarContainer isOpen={isOpen}>
      <CloseButton onClick={onClose}>&times;</CloseButton>
      {menuItems.map((item, index) => (
        <MenuItem key={index} href="#">
          {item}
        </MenuItem>
      ))}
    </SidebarContainer>
  );
};
```

Далі додаємо бокове меню, де будуть необхідні пунки для зручної навігації користувача по додатку.

```
const Home: React.FC = () => {
  const userCountryData = { USA: 50, Canada: 30, UK: 20 };
  const userAgeData = { '18-24': 25, '25-34': 40, '35-44': 20, '45+': 15 };
  const userTimeData = [10, 15, 20, 25, 30, 35, 40];

  return (
    <MainContainer>
      <h1>Data Analytics Dashboard</h1>
      <Description>
        Welcome to our Data Analytics Service! We provide insights into user data
        from social networks, helping you understand demographics, user behavior,
        and engagement.
      </Description>

      <h2>User Analytics</h2>
      <ChartsComponent
        userCountryData={userCountryData}
        userAgeData={userAgeData}
        userTimeData={userTimeData}
      />
    </MainContainer>
  );
};
```

У коді вище представлена код головної сторінки додатку з коротким описом сервісу на графіків з тимчасовими даними, поки ми не маємо даних користувача.

3.6 Тестування розробленого сервісу та можливостей додатку

Важливою складовою тестування є написання unit-тестів для компонентів та функцій.

```
import React from 'react';
import { render, screen } from '@testing-library/react';
import ChartsComponent from './ChartsComponent';

describe('ChartsComponent', () => {
 it('renders Users by Country chart', () => {
  const userCountryData = { USA: 50, Canada: 30, UK: 20 };
  const userAgeData = { '18-24': 25, '25-34': 40, '35-44': 20, '45+': 15 };
  const userTimeData = [10, 15, 20, 25, 30, 35, 40];

  render(
   <ChartsComponent
    userCountryData={userCountryData}
    userAgeData={userAgeData}
    userTimeData={userTimeData}
   />
  );

  expect(screen.getByText('Users by Country')).toBeInTheDocument();
  expect(screen.getByLabelText('Users by Country')).toBeInTheDocument();
 });

 it('renders Users by Age chart', () => {
  const userCountryData = { USA: 50, Canada: 30, UK: 20 };
  const userAgeData = { '18-24': 25, '25-34': 40, '35-44': 20, '45+': 15 };
  const userTimeData = [10, 15, 20, 25, 30, 35, 40];

  render(
   <ChartsComponent
    userCountryData={userCountryData}
    userAgeData={userAgeData}
    userTimeData={userTimeData}
   />
  );

  expect(screen.getByText('Users by Age')).toBeInTheDocument();
  expect(screen.getByLabelText('Users by Age')).toBeInTheDocument();
 });

 it('renders Time Spent by Users chart', () => {
  const userCountryData = { USA: 50, Canada: 30, UK: 20 };
  const userAgeData = { '18-24': 25, '25-34': 40, '35-44': 20, '45+': 15 };
  const userTimeData = [10, 15, 20, 25, 30, 35, 40];

  render(
```

```
      <ChartsComponent
        userCountryData={userCountryData}
        userAgeData={userAgeData}
        userTimeData={userTimeData}
      />
    );

    expect(screen.getByText('Time Spent by Users')).toBeInTheDocument();
    expect(screen.getByLabelText('Time Spent by Users')).toBeInTheDocument();
  });
});
```

Вище представлений unit-тест для тестування компоненту та функції, які відповідають за побудову графіків в залежності від даних користувача. Використані тестові дані, також перевіряється чи відбувається рендер необхідних елементів.

```
describe('AuthModule', () => {
  let module: TestingModule;
  let authService: AuthService;

  beforeEach(async () => {
    module = await Test.createTestingModule({
      imports: [AuthModule],
    }).compile();

    authService = module.get<AuthService>(AuthService);
  });

  it('should be defined', () => {
    expect(module).toBeDefined();
  });

  describe('AuthService', () => {
    let usersService: UsersService;
    let userRepository: Repository<User>;

    beforeEach(() => {
      usersService = module.get<UsersService>(UsersService);
      userRepository = module.get<Repository<User>>(getRepositoryToken(User));
    });

    it('should be defined', () => {
      expect(authService).toBeDefined();
    });

    it('validateUser should return null for invalid credentials', async () => {
```

```
    jest.spyOn(usersService, 'findByUsername').mockResolvedValue(null);

    const result = await authService.validateUser('nonexistentUser', 'invalidPassword');
    expect(result).toBeNull();
  });

  it('validateUser should return user data for valid credentials', async () => {
    const mockUser = new User();
    mockUser.username = 'testUser';
    mockUser.password = await bcrypt.hash('testPassword', 10);

    jest.spyOn(usersService, 'findByUsername').mockResolvedValue(mockUser);

    const result = await authService.validateUser('testUser', 'testPassword');
    expect(result).toBeDefined();
    expect(result.username).toEqual('testUser');
    expect(result.password).toBeUndefined();
  });
});
```

Вище представлено unit-тест для тестування авторизації на стороні сервера. За допомогою тестових даних перевіряємо чи все відпрацьовує вірно, чи відпрацьовують необхідні запити та функції.

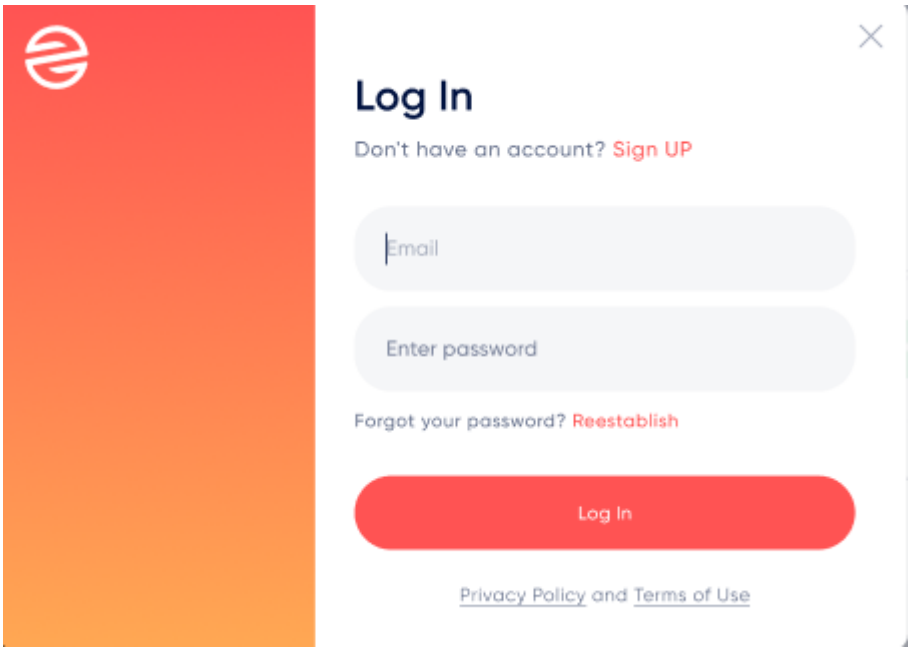


Рисунок 3.6 – Форма авторизації

На рисунку 3.6 представлена форма авторизації користувача. Необхідними даними є пошта та пароль. У випадку невірних даних система виділить поля.

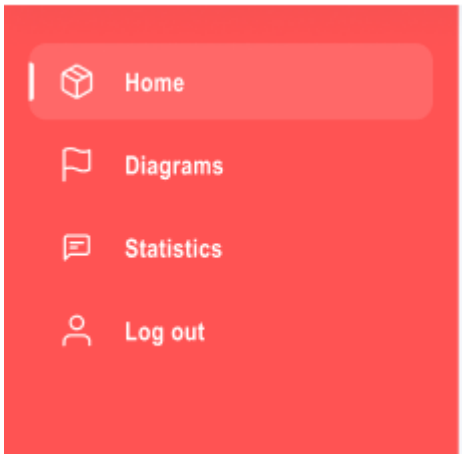


Рисунок 3.7 – Навігаційна панель

На рисунку 3.7 представлена навігаційна панель додатку. Основними пунктами є головна сторінка, діаграми, статистика та вихід.

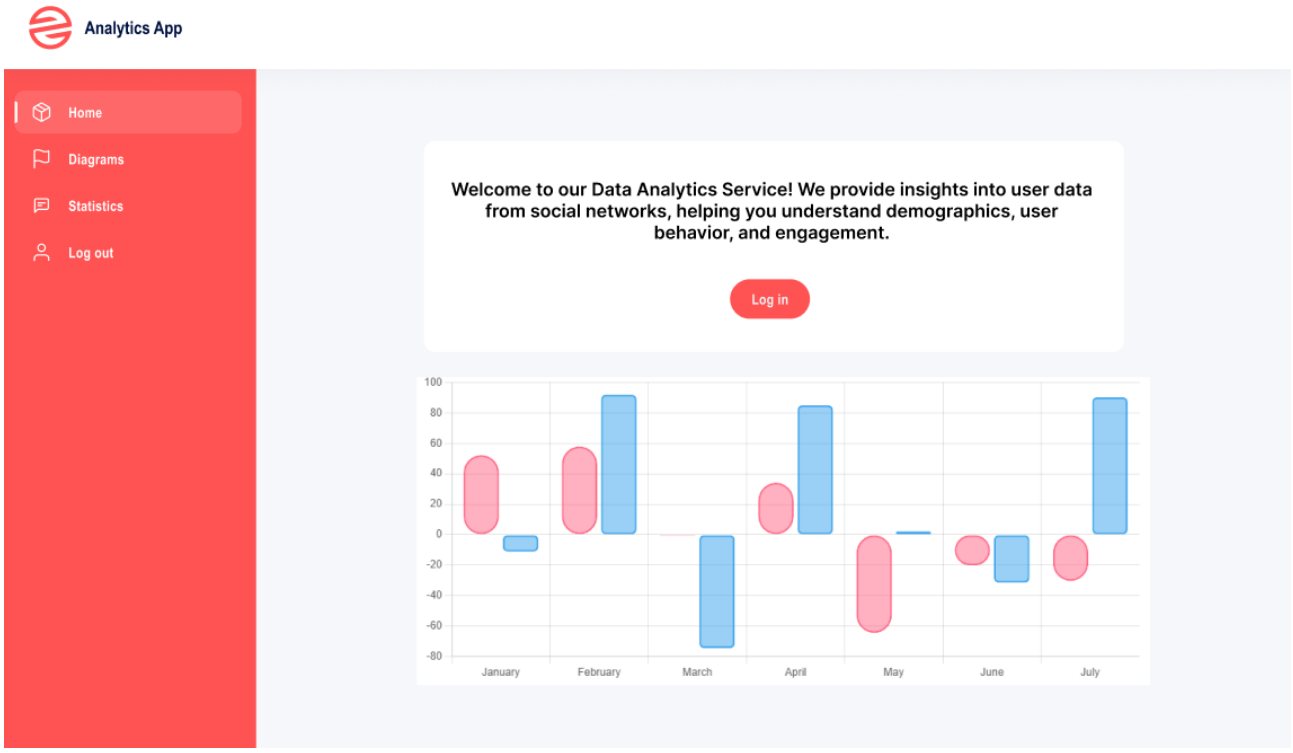


Рисунок 3.8 – Головна сторінка додатку

На рисунку 3.8 представлена головна сторінка додатку. Додано невеличкий опис сервісу, кнопка авторизації та приклад діаграми.

Рисунок 3.9 – Сторінка діаграм з аналітикою по віку аудиторії та країні

На рисунку 3.9 показано сторінку з діаграмами по віку та країні. Дані формуються динамічно з кожним оновленням. Це все відбувається без затримок користувача або помилок на сторінці.

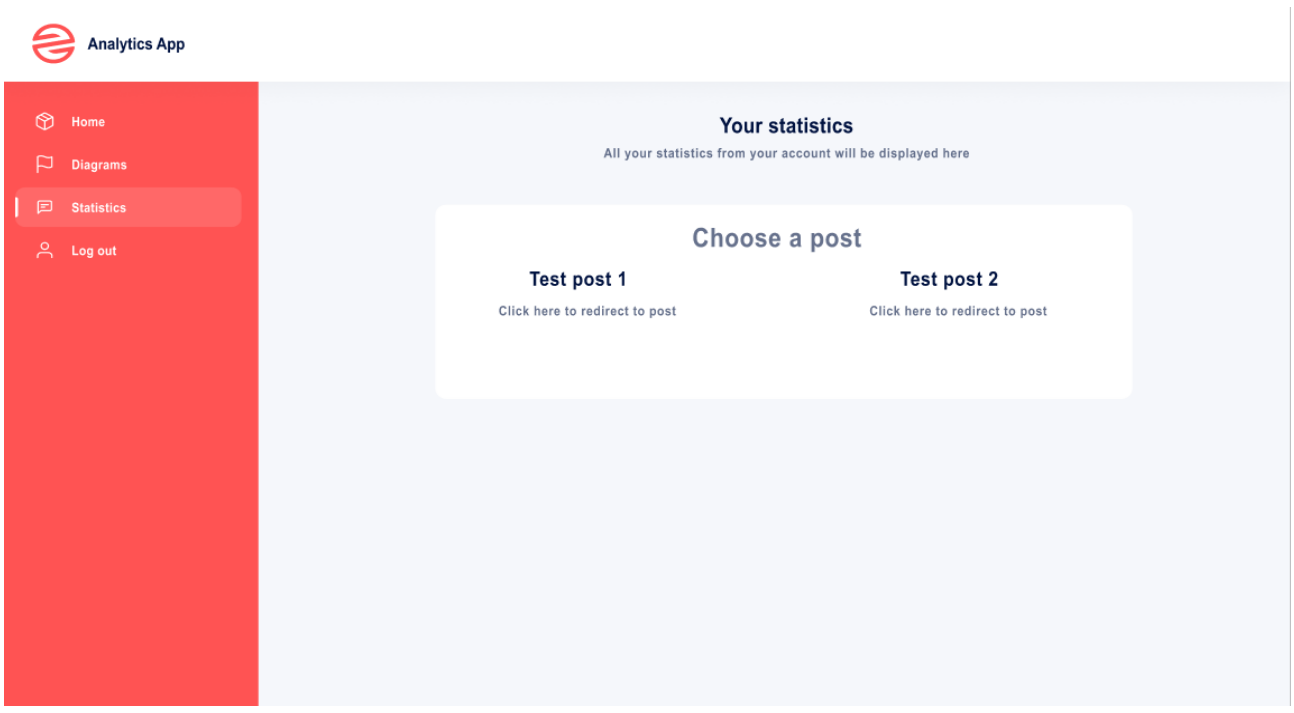


Рисунок 3.10 – Сторінка вибору посту

На рисунку 3.10 представлена сторінка статистики, де необхідно обрати пост. Після цього відбудеться розрахунок та відображення інформації.

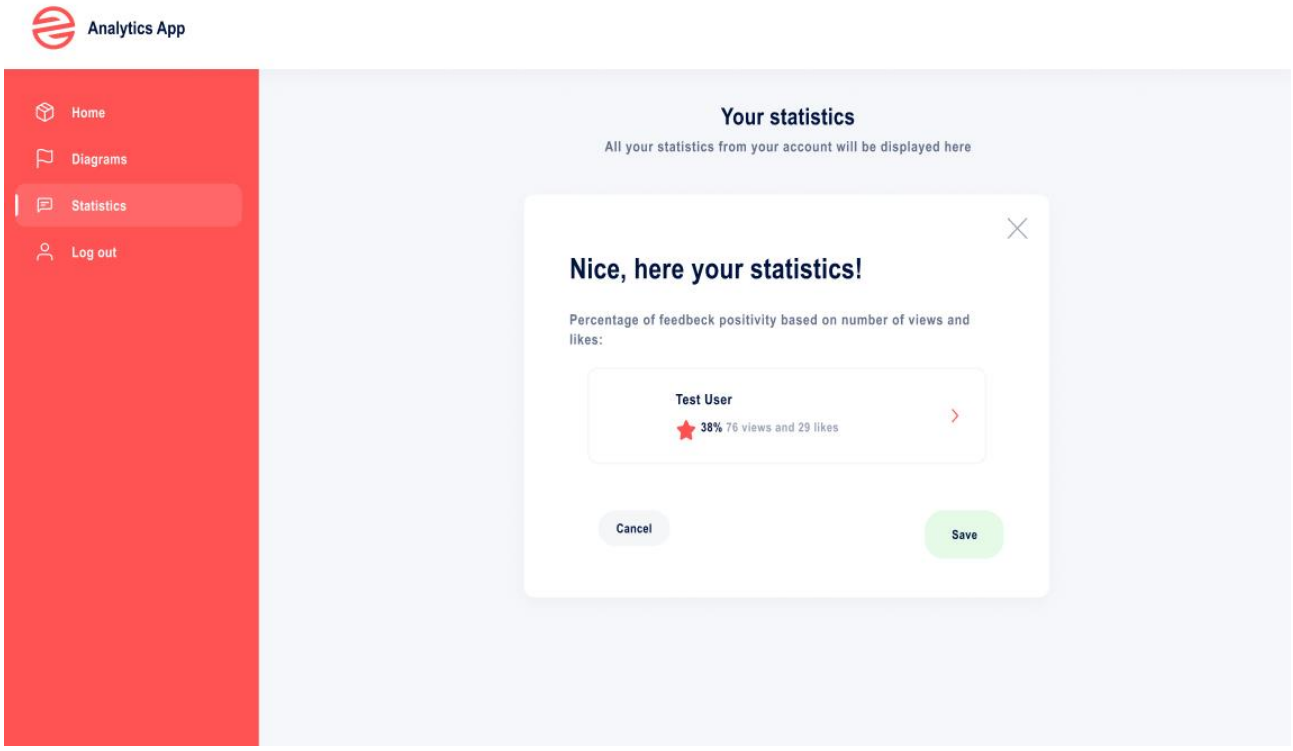


Рисунок 3.11 – Сторінка статистики, коли дані були розраховані

На рисунку 3.11 представлено сторінку статистики, після того, як дані були підраховані. Показана інформація по вдалий розрахунок, ім'я користувача, процент фідбеку та вибірку. Також присутня можливість збереження розрахунку.

ВИСНОВКИ

В результаті виконання роботи було розроблено інформаційну систему для аналізу активності користувачів соціальних мереж. Додано весь запланований функціонал. Було розглянуто різноманітні підходи до реалізації отриманого завдання, виділення вразливих місць майбутньої системи.

Проведена наступна робота:

- Аналіз конкурентів, виявлення їх плюсів та мінусів
- Аналіз проблем та пошук шляхів їх вирішення
- Підбір найбільш підходящого API для взаємодії з сервісом
- Створено функціонал з динамічної побудови графіків на основі даних від API
- Реалізовано надання користувачу необхідної інформації про аудиторію акаунту
- Створено функціонал з розрахунком позитивності фідбеку
- Реалізована повна клієнт-серверна взаємодія
- Розроблено мінімалістичний дизайн клієнтської частини
- Проведено тестування усіх можливостей сервісу на популярних нині соціальних мережах
- Покрито код unit-тестами

За результатами, отриманими у ході дослідження та розробки сервісу можна зробити висновок, що розроблена інформаційна система покриває усі необхідні вимоги користувача і може конкурувати з іншими, більш популярними аналогами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ресурс для пошуку роботи в Україні [Електронний ресурс] - Режим доступу до ресурсу: https://robota.ua/
2. Посібник: аналіз аудиторії у соціальних мережах [Електронний ресурс] – Режим доступу до ресурсу: https://www.post-up.com.ua/analiz-audytoriyi-u-soczialnyh-merezhah/
3. Посібник: як обробляти велику кількість інформації [Електронний ресурс] - режим доступу до ресурсу: https://developers.redhat.com/blog/2019/07/05/how-to-store-large-amounts-of-data-in-a-program
4. Інструмент для побудови таблиць та графіків [Електронний ресурс] – режим доступу до ресурсу: https://miro.com
5. Посібник: знайомство з React [Електронний ресурс] – Режим доступу до ресурсу: https://ru.reactjs.org/
6. Посібник: знайомство з Axios [Електронний ресурс] – Режим доступу до ресурсу: https://axios-http.com/docs/intro
7. Посібник: знайомство з React-router-dom [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/react-router-dom
8. Посібник: знайомство з TypeScript [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/
9. Посібник: знайомство з Redux Toolkit [Електронний ресурс] – Режим доступу до ресурсу: https://redux-toolkit.js.org/

10. Посібник: знайомство з styled-components [Електронний ресурс] – Режим доступу до ресурсу: https://styled-components.com/docs
11. Посібник: знайомство з react-redux[Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/react-redux
12. Посібник: знайомство з React Testing Library [Електронний ресурс] – Режим доступу до ресурсу: https://testing-library.com/docs/react-testing-library/intro
13. Посібник: знайомство з Jest [Електронний ресурс] – Режим доступу до ресурсу: https://jestjs.io/docs/tutorial-react
14. Посібник: знайомство з ts-jest [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/ts-jest
15. Посібник: знайомство з Nest.js [Електронний ресурс] – Режим доступу до ресурсу: https://docs.nestjs.com/
16. Посібник: знайомство з MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/docs/
17. Посібник: знайомство з Passport-jwt [Електронний ресурс] – Режим доступу до ресурсу: https://www.passportjs.org/packages/passport-jwt/
18. Посібник: знайомство з Bctypt [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/bcryptjs
19. Посібник: знайомство з Dotenv [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/dotenv
20. Посібник: знайомство з jsonwebtoken [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/jsonwebtoken
21. Посібник: знайомство з Mongoose [Електронний ресурс] – Режим доступу до ресурсу: https://mongoosejs.com/docs/guide.html
22. Посібник: знайомство з Eslint [Електронний ресурс] – Режим доступу до ресурсу: https://eslint.org/docs/latest/
23. Посібник: знайомство з Prettier [Електронний ресурс] – Режим доступу до ресурсу: https://prettier.io/docs/en/

24. Посібник: знайомство з Husky [Електронний ресурс] – Режим доступу до ресурсу: https://typicode.github.io/husky/
25. Посібник: знайомство з Eslint-plugin-prettier [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/eslint-plugin-prettier
26. Посібник: знайомство з eslint-plugin-simple-import-sort [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/eslint-plugin-simple-import-sort
27. Посібник: знайомство з eslint-import-resolver-typescript [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/eslint-import-resolver-typescript
28. Посібник: знайомство з eslint-plugin-import [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/eslint-plugin-import
29. Документація Meta for Developers [Електронний ресурс] – Режим доступу до ресурсу: https://developers.facebook.com/docs/
30. Документація: Facebook Pages API [Електронний ресурс] – режим доступу до ресурсу: https://developers.facebook.com/docs/pages-api/

ДОДАТОК

Авторизація на стороні сервера:

```
import { Module } from '@nestjs/common';
import { PassportModule } from '@nestjs/passport';
import { JwtModule } from '@nestjs/jwt';
import { AuthService } from './auth.service';
import { LocalStrategy } from './local.strategy';
import { JwtStrategy } from './jwt.strategy';
import { UsersModule } from '../users/users.module';

@Module({
 imports: [
   PassportModule,
   JwtModule.register({
     secret: '',
     signOptions: { expiresIn: '1h' },
   }),
   UsersModule,
 ],
 providers: [AuthService, LocalStrategy, JwtStrategy],
 exports: [AuthService],
})
export class AuthModule {}

import { Injectable } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { UsersService } from '../users/users.service';
import * as bcrypt from 'bcrypt';

@Injectable()
export class AuthService {
 constructor(
   private usersService: UsersService,
   private jwtService: JwtService,
 ) {}

 async validateUser(username: string, password: string): Promise<any> {
```

```
    const user = await this.usersService.findByUsername(username);

    if (user && (await bcrypt.compare(password, user.password))) {
      const { password, ...result } = user;
      return result;
    }

    return null;
  }

  async login(user: any) {
    const payload = { username: user.username, sub: user.userId };
    return {
      access_token: this.jwtService.sign(payload),
    };
  }
}

import { Injectable } from '@nestjs/common';
import { Strategy } from 'passport-local';
import { PassportStrategy } from '@nestjs/passport';
import { AuthService } from './auth.service';

@Injectable()
export class LocalStrategy extends PassportStrategy(Strategy) {
  constructor(private authService: AuthService) {
    super();
  }

  async validate(username: string, password: string): Promise<any> {
    const user = await this.authService.validateUser(username, password);

    if (!user) {
      return Promise.reject('Invalid credentials');
    }

    return user;
  }
}

import { Injectable } from '@nestjs/common';
import { PassportStrategy } from '@nestjs/passport';
import { ExtractJwt, Strategy } from 'passport-jwt';
import { AuthService } from './auth.service';

@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
  constructor(private authService: AuthService) {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      secretOrKey: '',
    });
  }
```

```
  async validate(payload: any) {
    return { userId: payload.sub, username: payload.username };
  }
}
```

Отримання списку соціальних мереж:

```
import { IsArray, IsString } from 'class-validator';

export class SocialNetworksDto {
  @IsArray()
  networks: string[];
}

import { Controller, UseGuards, Request, Put, Body } from '@nestjs/common';
import { JwtAuthGuard } from '../auth/jwt-auth.guard';
import { SocialNetworksDto } from './social-networks.dto';
import { UsersService } from '../users/users.service';

@Controller('social-networks')
export class SocialNetworksController {
  constructor(private usersService: UsersService) {}

  @UseGuards(JwtAuthGuard)
  @Put()
  async updateSocialNetworks(@Request() req, @Body() socialNetworksDto: SocialNetworksDto) {
    const userId = req.user.userId;
    const networks = socialNetworksDto.networks;

    await this.usersService.updateSocialNetworks(userId, networks);

    return { success: true, message: 'Social networks updated successfully' };
  }
}

import { Injectable } from '@nestjs/common';
import { User } from './user.entity';
import { Repository } from 'typeorm';
import { InjectRepository } from '@nestjs/typeorm';

@Injectable()
export class UsersService {
  constructor(
    @InjectRepository(User)
    private userRepository: Repository<User>,
  ) {}

  async updateSocialNetworks(userId: number, networks: string[]): Promise<void> {
    const user = await this.userRepository.findOne(userId);

    if (!user) {
      throw new Error('User not found');
```

```
    }

    user.socialNetworks = networks;
    await this.userRepository.save(user);
  }
```

Отримання постів:

```
import { IsString, IsNotEmpty } from 'class-validator';

export class CreatePostDto {
  @IsString()
  @IsNotEmpty()
  title: string;

  @IsString()
  @IsNotEmpty()
  content: string;
}

import { Controller, UseGuards, Request, Post, Body } from '@nestjs/common';
import { JwtAuthGuard } from '../auth/jwt-auth.guard';
import { CreatePostDto } from './post.dto';
import { PostsService } from './posts.service';

@Controller('posts')
export class PostsController {
  constructor(private postsService: PostsService) {}

  @UseGuards(JwtAuthGuard)
  @Post()
  async createPost(@Request() req, @Body() createPostDto: CreatePostDto) {
    const userId = req.user.userId;
    const post = await this.postsService.createPost(userId, createPostDto);

    return { success: true, post };
  }
}

import { Injectable } from '@nestjs/common';
import { Post } from './post.entity';
import { Repository } from 'typeorm';
import { InjectRepository } from '@nestjs/typeorm';
import { CreatePostDto } from './post.dto';

@Injectable()
export class PostsService {
  constructor(
    @InjectRepository(Post)
    private postRepository: Repository<Post>,
  ) {}

  async createPost(userId: number, createPostDto: CreatePostDto): Promise<Post> {
```

```
    const { title, content } = createPostDto;

    const post = this.postRepository.create({
      title,
      content,
      userId,
    });

    await this.postRepository.save(post);

    return post;
  }
}
```

main.ts

```
import { join } from 'path';

import { NestFactory } from '@nestjs/core';
import { NestExpressApplication } from '@nestjs/platform-express';

import { AppModule } from '#domains/app/app.module';
import { Swagger } from '#domains/swagger/index';
import envConfig from '#utils/config/env';
const PORT = envConfig().port;

async function bootstrap() {
    const app = await NestFactory.create<NestExpressApplication>(AppModule);
    app.enableCors({
        origin: '*',
    });
    app.setBaseViewsDir(join(__dirname));
    app.setViewEngine('hbs');

    Swagger(app);

    // app.use(TenantMiddleware);
    await app.listen(PORT);

    Promise.resolve(app).then(() => console.log('[Nest]: Application is running'));
}
bootstrap();
```

Схема користувача:

```
import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
import { Date, Document } from 'mongoose';

@Schema({ strict: false })
export class User {
@Prop({ required: true, type: String })
name: string;
@Prop({ required: true, type: String })
```

```
email: string;

@Prop({ required: true, type: Date })
password: Date;

@Prop({ required: true, type: Date })
createdDate: Date;
}

export type UserDocument = User & Document;

export const EventSchema = SchemaFactory.createForClass(User);
```

Схема для отримання соціальних мереж:

```
import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
import { Document } from 'mongoose';
@Schema({ collection: 'usersocialmedia' })
export class UserSocialMedia {
@Prop({ required: true, type: String })
name: string;

@Prop({ required: true, type: Date })
connectionDate: Date;

@Prop({ type: String, required: true })
url: string;
}

export type UserSocialMediaDocument = UserSocialMedia & Document;

export const ProductTypeSchema = SchemaFactory.createForClass(UserSocialMedia);
```

Схема для отримання поста:

```
import { Prop, SchemaFactory, Schema } from '@nestjs/mongoose';
import { Document } from 'mongoose';

@Schema({ strict: false })
export class Post {
@Prop({ required: true, type: String })
title: string;

@Prop({ required: true, type: Date })
createdDate: Date;

@Prop({ required: true, type: String })
url: string;

@Prop({ required: true, type: Number })
        userLikes: number;
```

```
@Prop({ required: true, type: Number })
usersViews: number;

@Prop({ required: true, type: Object })
author: Object;
}

export type PostDocument = Post & Document;

export const PostSchema = SchemaFactory.createForClass(Post);
```

Оновлення сторінки при розрахунку:

```
import * as path from 'path';

import { Controller, Get, Param, Res, UseGuards, Req } from '@nestjs/common';
import { Response } from 'express';

import { PostService } from '#domains/products/products.service';
import { Public } from '#utils/decorators/Public';
import { SSEGuard } from '#utils/guards/sse.guard';

@Controller('/v1.0/sse')
export class UpdateViewController {
  constructor(private readonly productsService: PostService) {}
  @Public()
  @UseGuards(SSEGuard)
  @Get()
  getUpdatedView(@Req() req, @Res() res: Response) {
    res.setHeader('Content-Type', 'text/event-stream');
    res.setHeader('Cache-Control', 'no-cache');
    res.setHeader('Connection', 'keep-alive');

    const fs = require('fs');
    const htmlPath = path.join(__dirname, '../../', 'index.html');
    const htmlContent = fs.readFileSync(htmlPath, 'utf8');

    const jsonContent = { html: htmlContent };

    res.write(`data: ${JSON.stringify(jsonContent)}\n\n`);

    const interval = setInterval(() => {
      res.write(`${JSON.stringify(jsonContent)}`);
    }, 10000);

    req.on('close', () => {
      clearInterval(interval);
    });
  }
```

```
  @Public()
  @UseGuards(SSEGuard)
  @Get('/products/:productType')
  async listenProducts(@Res() res: Response, @Param('post') post: string) {
    res.setHeader('Content-Type', 'text/event-stream');
    res.setHeader('Cache-Control', 'no-cache');
    res.setHeader('Connection', 'keep-alive');

    const sendEvent = (data: string) => {
      res.write(`data: ${data}\n\n`);
    };

    let posts = [];

    if (post) {
      posts = await this.postService();
    } else {
      posts = await this.postService.delete()
    }

    sendPostData(JSON.stringify(posts));

    const intervalId = setInterval(async () => {
      let newPosts = [];

      if (posts) {
        newPosts = await this.productsService.fetchPosts();
      }
      if (newPosts.length > 0) {
        sendPost(JSON.stringify(newPosts));
      }
    }, 5000);

    res.on('close', () => {
      clearInterval(intervalId);
      res.end();
    });
  }
}
```

Компонент авторизації:

```
import React, { useState } from 'react';
import styled from 'styled-components';

const AuthWrapper = styled.div`
  max-width: 400px;
  margin: 0 auto;
  padding: 20px;
  border: 1px solid #ddd;
  border-radius: 8px;
`;
```

```
const Input = styled.input`
  width: 100%;
  margin-bottom: 10px;
  padding: 8px;
`;

const Button = styled.button`
  width: 100%;
  padding: 10px;
  background-color: #007bff;
  color: #fff;
  border: none;
  border-radius: 4px;
  cursor: pointer;

  &:hover {
    background-color: #0056b3;
  }
`;

interface AuthComponentProps {
  onLogin: (username: string, password: string) => void;
}

const AuthComponent: React.FC<AuthComponentProps> = ({ onLogin }) => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');

  const handleLogin = () => {
    onLogin(username, password);
  };

  return (
    <AuthWrapper>
      <h2>Login</h2>
      <label>
        Username:
        <Input
          type="text"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
        />
      </label>
      <label>
        Password:
        <Input
          type="password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
        />
      </label>
      <Button onClick={handleLogin}>Login</Button>
```

```
  </AuthWrapper>
 );
};

export default AuthComponent;
```

Компонент розрахунку фідбеку:

```
import React from 'react';
import styled from 'styled-components';

interface PositiveFeedbackPercentageProps {
 likes: number;
 views: number;
}

const Wrapper = styled.div`
 display: flex;
 align-items: center;
 font-size: 14px;
`;

const LikeIcon = styled.span`
 margin-right: 5px;
 color: #007bff;
`;

const PositiveFeedbackPercentage: React.FC<PositiveFeedbackPercentageProps> = ({
 likes,
 views,
}) => {
 const calculatePositivePercentage = () => {
  if (views === 0) {
   return 0;
  }

  const positivePercentage = (likes / views) * 100;
  return positivePercentage.toFixed(2);
 };

 return (
  <Wrapper>
   <LikeIcon>+</LikeIcon>
   {`${calculatePositivePercentage()}% Positive Feedback`}
  </Wrapper>
 );
};
```

Компонент побудови діаграм:

```
import React from 'react';
import { Bar, Doughnut, Line, Pie } from 'react-chartjs-2';
```

```
interface ChartsComponentProps {
  userCountryData: { [key: string]: number };
  userAgeData: { [key: string]: number };
  userTimeData: number[];
}

const ChartsComponent: React.FC<ChartsComponentProps> = ({
  userCountryData,
  userAgeData,
  userTimeData,
}) => {
  const countryData = {
    labels: Object.keys(userCountryData),
    datasets: [
      {
        label: 'Users by Country',
        data: Object.values(userCountryData),
        backgroundColor: [
          'rgba(255, 99, 132, 0.6)',
          'rgba(54, 162, 235, 0.6)',
          'rgba(255, 206, 86, 0.6)',
          'rgba(75, 192, 192, 0.6)',
        ],
      },
    ],
  };

  const ageData = {
    labels: Object.keys(userAgeData),
    datasets: [
      {
        label: 'Users by Age',
        data: Object.values(userAgeData),
        backgroundColor: 'rgba(75, 192, 192, 0.6)',
      },
    ],
  };

  const timeData = {
    labels: Array.from({ length: userTimeData.length }, (_, i) => i + 1),
    datasets: [
      {
        label: 'Time Spent by Users',
        data: userTimeData,
        fill: false,
        borderColor: 'rgba(75, 192, 192, 0.6)',
      },
    ],
  };

  return (
    <div>
```

```
      <h2>Users by Country</h2>
      <Doughnut data={countryData} />

      <h2>Users by Age</h2>
      <Bar data={ageData} />

      <h2>Time Spent by Users</h2>
      <Line data={timeData} />
    </div>
  );
};

export default ChartsComponent;
```

Файл Eslint.js

```
{
  "settings": {
    "import/resolver": {
      "typescript": true,
      "node": true
    }
  },
  "env": {
    "browser": true,
    "es6": true
  },
  "parser": "@typescript-eslint/parser",
  "parserOptions": {
    "ecmaFeatures": {
      "jsx": true
    },
    "ecmaVersion": "latest",
    "sourceType": "module"
  },
  "extends": [
    "plugin:@nx/react",
    "eslint:recommended",
    "plugin:react/recommended",
    "plugin:react-hooks/recommended",
    "plugin:@typescript-eslint/recommended",
    "plugin:import/recommended",
    "plugin:import/typescript",
    "plugin:prettier/recommended",
    "plugin:cypress/recommended"
  ],
  "root": true,
  "ignorePatterns": ["**/*", "*.css"],
  "plugins": ["@nx", "react", "@typescript-eslint", "simple-import-sort"],
  "overrides": [
    {
      "files": ["*.ts", "*.tsx", "*.js", "*.jsx"],
```

```
    "rules": {
      "@nx/enforce-module-boundaries": [
        "error",
        {
          "enforceBuildableLibDependency": true,
          "allow": [],
          "depConstraints": [
            {
              "sourceTag": "*",
              "onlyDependOnLibsWithTags": ["*"]
            }
          ]
        }
      ],
      "simple-import-sort/imports": [
        "warn",
        {
          "groups": [
            ["^react"],
            ["^axios|react-redux|uid|@redux"],
            ["^@app"],
            ["^@pages"],
            ["^@widgets"],
            ["^@features"],
            ["^@entities"],
            ["^@shared"],
            ["^@packages"],
            ["^@public"],
            ["^\\u0000", "^@?\\w"],
            ["^\\.\\.(?!/?$)", "^\\.\\./?$"],
            ["^\\./(?=.*/)(?!/?$)", "^\\.(?!/?$)", "^\\./?$"],
            ["^.+\\.?(css)$"]
          ]
        }
      ]
    }
  },
  {
    "files": ["*.ts", "*.tsx"],
    "extends": ["plugin:@nx/typescript"],
    "rules": {}
  },
  {
    "files": ["*.js", "*.jsx"],
    "extends": ["plugin:@nx/javascript"],
    "rules": {}
  }
],
"rules": {
  "import/no-named-as-default": "off",
  "import/namespace": "off",
  "@typescript-eslint/ban-ts-comment": "warn",
  "@typescript-eslint/no-inferrable-types": "off",
```

```
    "@typescript-eslint/explicit-function-return-type": "off",
    "import/newline-after-import": [
      "warn",
      {
        "count": 1
      }
    ],
    "react/display-name": "off",
    "no-extra-boolean-cast": "off",
    "react/react-in-jsx-scope": "off",
    "prefer-const": "warn",
    "react-hooks/rules-of-hooks": "warn",
    "import/no-unresolved": "off",
    "import/default": "off",
    "quotes": ["warn", "single"],
    "max-len": ["warn", 120],
    "prettier/prettier": [
      "warn",
      {
        "endOfLine": "auto"
      }
    ],
    "react/jsx-closing-bracket-location": [
      1,
      {
        "selfClosing": "tag-aligned",
        "nonEmpty": "after-props"
      }
    ],
    "react/jsx-filename-extension": [
      "warn",
      {
        "extensions": [".tsx"]
      }
    ],
    "import/extensions": [
      "error",
      "ignorePackages",
      {
        "ts": "never",
        "tsx": "never"
      }
    ]
  }
}
```

Файл API:

```
import { ModalSlice } from '@app/store/slices/modal.slice';
import { ProductsSlice } from '@app/store/slices/products.slice';
import { StatusSlice } from '@app/store/slices/status.slice';
import { UploadBannerSlice } from '@app/store/slices/uploadbanner.slice';
```

```
import { EventEndpoint } from '@shared/enums/ApiEndpoints.enums';
import { ModalType } from '@shared/enums/Modal.enums';
import { AppConfig } from '@shared/interfaces/App.interfaces';
import {
  NewProductTypeData,
  PreparedProductData,
  ProductData,
  ProductTypes,
} from '@shared/interfaces/Products.interfaces';


import { HttpMethods } from '@packages/shared/enums/HttpMethod.enums';


import { baseApi } from './base.api';


const backendUrl = import.meta.env.VITE_NEST_BE_ACCESS_TOKEN;
const baseUrl = import.meta.env.VITE_NEST_BACKEND_APP_LOCAL;


const { setIsLoading, setNotification } = StatusSlice.actions;
const { setProducts, clearNewProduct, setProductTypes } = ProductsSlice.actions;
const { setModalType } = ModalSlice.actions;
const { clearBannersState } = UploadBannerSlice.actions;


declare const globalAppConfig: AppConfig;


export const Api = baseApi.injectEndpoints({
  overrideExisting: true,
  endpoints: (builder) => ({
    getPosts: builder.query<ProductData[], null>({
      query: () => ({
        url: baseUrl + EventEndpoint.GET,
        method: HttpMethods.GET,
        headers: { Authorization: backendUrl, 'X-Tenant-Name': globalAppConfig.tenant },
      }),
      providesTags: ['Posts'],
      async onQueryStarted(_, { dispatch, queryFulfilled }) {
        try {
          dispatch(setIsLoading(true));
          const { data } = await queryFulfilled;
          dispatch(setPosts(data));
          dispatch(setIsLoading(false));
        } catch (err) {
          if (err instanceof Error) {
            dispatch(setIsLoading(false));
            throw new Error(err.message);
          }
```

```
      }
    },
  }),
  addPost: builder.mutation<NewPost, NewPost>({
    query: (data: NewPost) => ({
      url: baseUrl + EventEndpoint.ADD_PRODUCT_TYPE,
      method: HttpMethods.POST,
      body: data,
      headers: { Authorization: backendUrl, 'X-Tenant-Name': globalAppConfig.tenant, Accept: 'text/event-
stream' },
    }),
    async onQueryStarted(_, { dispatch, queryFulfilled }) {
      try {
        dispatch(setIsLoading(true));
        const { data } = await queryFulfilled;
        dispatch(setIsLoading(false));
      } catch (err) {
        if (err instanceof Error) {
          dispatch(setIsLoading(false));
          throw new Error(err.message);
        }
      }
    },
  }),
  createPostModel: builder.mutation<PostModel, PostModel>({
    query: (data: FormData) => ({
      url: baseUrl + EventEndpoint.CREATE,
      method: HttpMethods.POST,
      body: data,
      headers: { Authorization: backendUrl, 'X-Tenant-Name': globalAppConfig.tenant },
    }),
    invalidatesTags: ['PostModel'],
    async onQueryStarted(_, { dispatch, queryFulfilled }) {
      try {
        dispatch(setIsLoading(true));
        dispatch(setModalType(ModalType.LOADING));
        const { data } = await queryFulfilled;
        if (data) {
          dispatch(setModalType(null));

        }
        dispatch(setIsLoading(false));
        dispatch(clearPostModel());

      } catch (err) {
        if (err instanceof Error) {
          dispatch(setIsLoading(false));
          throw new Error(err.message);
        }
      }
    },
  }),
    },
```

```
    }),
  }),
});
```

Навігаційне меню:

```
import React from 'react';
import styled from 'styled-components';


interface SidebarProps {
  isOpen: boolean;
  onClose: () => void;
}


const SidebarContainer = styled.div<{ isOpen: boolean }>`
  position: fixed;
  top: 0;
  left: ${({ isOpen }) => (isOpen ? '0' : '-100%')};
  width: 250px;
  height: 100%;
  background-color: #333;
  padding-top: 60px;
  transition: left 0.3s ease-in-out;
`;


const CloseButton = styled.button`
  position: absolute;
  top: 20px;
  right: 20px;
  background: none;
  color: #fff;
  border: none;
  font-size: 20px;
  cursor: pointer;
`;


const MenuItem = styled.a`
  display: block;
  padding: 15px;
  color: #fff;
  text-decoration: none;
  font-size: 18px;


  &:hover {
    background-color: #555;
  }
`;
```

```
const Sidebar: React.FC<SidebarProps> = ({ isOpen, onClose }) => {
  const menuItems = ['Home', 'About', 'Services', 'Contact'];


  return (
    <SidebarContainer isOpen={isOpen}>
      <CloseButton onClick={onClose}>&times;</CloseButton>
      {menuItems.map((item, index) => (
        <MenuItem key={index} href="#">
          {item}
        </MenuItem>
      ))}
    </SidebarContainer>
  );
};


export default Sidebar;
```

Компонент навігаційного меню:

```
import React, { useState } from 'react';
import styled from 'styled-components';
import Sidebar from './Sidebar';

const MainContent = styled.div`
  margin-left: 250px;
  padding: 20px;
`;

const ToggleButton = styled.button`
  position: fixed;
  top: 20px;
  left: 20px;
  background: none;
  color: #333;
  font-size: 24px;
  border: none;
  cursor: pointer;
`;

const SideBar: React.FC = () => {
  const [isSidebarOpen, setSidebarOpen] = useState(false);

  const toggleSidebar = () => {
    setSidebarOpen(!isSidebarOpen);
  };

  return (
    <div>
      <ToggleButton onClick={toggleSidebar}>&#9776;</ToggleButton>
      <Sidebar isOpen={isSidebarOpen} onClose={toggleSidebar} />
```

```
    <MainContent>
      <h1>Your App</h1>
      <p>Main content goes here.</p>
    </MainContent>
   </div>
  );
};

export default SideBar;
```

Компонент головної сторінки:

```
import React from 'react';
import styled from 'styled-components';
import ChartsComponent from './ChartsComponent';

const MainContainer = styled.div`
  padding: 20px;
`;

const Description = styled.p`
  font-size: 18px;
  margin-bottom: 20px;
`;

const Home: React.FC = () => {
  const userCountryData = { Ukraine: 50, Poland: 30, USA: 20 };
  const userAgeData = { '18-24': 25, '25-34': 40, '35-44': 20, '45+': 15 };
  const userTimeData = [10, 15, 20, 25, 30, 35, 40];

  return (
   <MainContainer>
    <h1>Data Analytics Dashboard</h1>
    <Description>
     Welcome to our Data Analytics Service! We provide insights into user data
     from social networks, helping you understand demographics, user behavior,
     and engagement.
    </Description>

    <h2>User Analytics</h2>
    <ChartsComponent
     userCountryData={userCountryData}
     userAgeData={userAgeData}
     userTimeData={userTimeData}
    />
   </MainContainer>
  );
};

export default Home;
```

Unit-тест для головної сторінки:

```
import React from 'react';
import { render, screen } from '@testing-library/react';
import ChartsComponent from './ChartsComponent';

describe('ChartsComponent', () => {
 it('renders Users by Country chart', () => {
  const userCountryData = { USA: 50, Canada: 30, UK: 20 };
  const userAgeData = { '18-24': 25, '25-34': 40, '35-44': 20, '45+': 15 };
  const userTimeData = [10, 15, 20, 25, 30, 35, 40];

  render(
   <ChartsComponent
    userCountryData={userCountryData}
    userAgeData={userAgeData}
    userTimeData={userTimeData}
   />
  );

  expect(screen.getByText('Users by Country')).toBeInTheDocument();
  expect(screen.getByLabelText('Users by Country')).toBeInTheDocument();
 });

 it('renders Users by Age chart', () => {
  const userCountryData = { USA: 50, Canada: 30, UK: 20 };
  const userAgeData = { '18-24': 25, '25-34': 40, '35-44': 20, '45+': 15 };
  const userTimeData = [10, 15, 20, 25, 30, 35, 40];

  render(
   <ChartsComponent
    userCountryData={userCountryData}
    userAgeData={userAgeData}
    userTimeData={userTimeData}
   />
  );

  expect(screen.getByText('Users by Age')).toBeInTheDocument();
  expect(screen.getByLabelText('Users by Age')).toBeInTheDocument();
 });

 it('renders Time Spent by Users chart', () => {
  const userCountryData = { USA: 50, Canada: 30, UK: 20 };
  const userAgeData = { '18-24': 25, '25-34': 40, '35-44': 20, '45+': 15 };
  const userTimeData = [10, 15, 20, 25, 30, 35, 40];

  render(
   <ChartsComponent
    userCountryData={userCountryData}
    userAgeData={userAgeData}
    userTimeData={userTimeData}
   />
  );
```

```
    expect(screen.getByText('Time Spent by Users')).toBeInTheDocument();
    expect(screen.getByLabelText('Time Spent by Users')).toBeInTheDocument();
  });
});
```

Юніт тест для авторизації:

```
import { Test, TestingModule } from '@nestjs/testing';
import { AuthModule } from './auth.module';
import { AuthService } from './auth.service';
import { LocalStrategy } from './local.strategy';
import { JwtStrategy } from './jwt.strategy';
import { UsersService } from '../users/users.service';
import { getRepositoryToken } from '@nestjs/typeorm';
import { User } from '../users/user.entity';
import { Repository } from 'typeorm';
import * as bcrypt from 'bcrypt';


describe('AuthModule', () => {
  let module: TestingModule;
  let authService: AuthService;


  beforeEach(async () => {
    module = await Test.createTestingModule({
      imports: [AuthModule],
    }).compile();


    authService = module.get<AuthService>(AuthService);
  });


  it('should be defined', () => {
    expect(module).toBeDefined();
  });


  describe('AuthService', () => {
    let usersService: UsersService;
    let userRepository: Repository<User>;


    beforeEach(() => {
      usersService = module.get<UsersService>(UsersService);
      userRepository = module.get<Repository<User>>(getRepositoryToken(User));
    });


    it('should be defined', () => {
      expect(authService).toBeDefined();
```

```
  });

  it('validateUser should return null for invalid credentials', async () => {
    jest.spyOn(usersService, 'findByUsername').mockResolvedValue(null);

    const result = await authService.validateUser('nonexistentUser', 'invalidPassword');
    expect(result).toBeNull();
  });

  it('validateUser should return user data for valid credentials', async () => {
    const mockUser = new User();
    mockUser.username = 'testUser';
    mockUser.password = await bcrypt.hash('testPassword', 10);

    jest.spyOn(usersService, 'findByUsername').mockResolvedValue(mockUser);

    const result = await authService.validateUser('testUser', 'testPassword');
    expect(result).toBeDefined();
    expect(result.username).toEqual('testUser');
    expect(result.password).toBeUndefined();
  });
});
```

Unit-тест для отримання соціальних мереж:

```
import { Test, TestingModule } from '@nestjs/testing';
import { SocialNetworksController } from './social-networks.controller';
import { UsersService } from '../users/users.service';
import { SocialNetworksDto } from './social-networks.dto';
import { JwtAuthGuard } from '../auth/jwt-auth.guard';

describe('SocialNetworksController', () => {
  let controller: SocialNetworksController;
  let usersService: UsersService;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      controllers: [SocialNetworksController],
      providers: [
        {
          provide: UsersService,
          useValue: {
            updateSocialNetworks: jest.fn(),
          },
        },
      ],
    })
    .overrideGuard(JwtAuthGuard)
```

```
    .useValue({ canActivate: () => true })
    .compile();

    controller = module.get<SocialNetworksController>(SocialNetworksController);
    usersService = module.get<UsersService>(UsersService);
  });

  it('should be defined', () => {
    expect(controller).toBeDefined();
  });

  describe('updateSocialNetworks', () => {
    it('should update social networks successfully', async () => {
      const userId = 1;
      const networks = ['Facebook', 'Twitter'];
      const socialNetworksDto: SocialNetworksDto = { networks };

      jest.spyOn(usersService, 'updateSocialNetworks').mockResolvedValueOnce();

      const result = await controller.updateSocialNetworks({ user: { userId } }, socialNetworksDto);

      expect(usersService.updateSocialNetworks).toHaveBeenCalledWith(userId, networks);
      expect(result).toEqual({ success: true, message: 'Social networks updated successfully' });
    });

    it('should handle errors during update', async () => {
      const userId = 1;
      const networks = ['Facebook', 'Twitter'];
      const socialNetworksDto: SocialNetworksDto = { networks };

      jest.spyOn(usersService, 'updateSocialNetworks').mockRejectedValueOnce(new Error('User not
found'));

      await expect(
        controller.updateSocialNetworks({ user: { userId } }, socialNetworksDto)
      ).rejects.toThrowError('User not found');
    });
  });
});
```

Unit-тест для отримання постів:

```
import { Test, TestingModule } from '@nestjs/testing';
import { PostsController } from './posts.controller';
import { PostsService } from './posts.service';
import { CreatePostDto } from './post.dto';

describe('PostsController', () => {
  let controller: PostsController;
  let postsService: PostsService;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
```

```
      controllers: [PostsController],
      providers: [
        {
          provide: PostsService,
          useValue: {
            createPost: jest.fn(),
          },
        },
      ],
    })
    .overrideGuard(JwtAuthGuard)
    .useValue({ canActivate: () => true })
    .compile();

    controller = module.get<PostsController>(PostsController);
    postsService = module.get<PostsService>(PostsService);
  });

  it('should be defined', () => {
    expect(controller).toBeDefined();
  });

  describe('createPost', () => {
    it('should create a post successfully', async () => {
      const userId = 1;
      const createPostDto: CreatePostDto = { title: 'Test Title', content: 'Test Content' };

      jest.spyOn(postsService, 'createPost').mockResolvedValueOnce(createPostDto as any);

      const result = await controller.createPost({ user: { userId } }, createPostDto);

      expect(postsService.createPost).toHaveBeenCalledWith(userId, createPostDto);
      expect(result).toEqual({ success: true, post: createPostDto });
    });

    it('should handle errors during post creation', async () => {
      const userId = 1;
      const createPostDto: CreatePostDto = { title: 'Test Title', content: 'Test Content' };

      jest.spyOn(postsService, 'createPost').mockRejectedValueOnce(new Error('Error creating post'));

      await expect(
        controller.createPost({ user: { userId } }, createPostDto)
      ).rejects.toThrowError('Error creating post');
    });
  });
});
```

Unit-тест для підрахунку позитивності фідбеку:

```
import React from 'react';
import { render } from '@testing-library/react';
import PositiveFeedbackPercentage from './PositiveFeedbackPercentage';
```

```
describe('PositiveFeedbackPercentage', () => {
  it('should render positive feedback percentage correctly', () => {
    const likes = 50;
    const views = 100;

    const { getByText } = render(<PositiveFeedbackPercentage likes={likes} views={views} />);

    const positiveFeedbackText = getByText(/50.00% Positive Feedback/i);
    expect(positiveFeedbackText).toBeInTheDocument();
  });

  it('should handle case when views are 0', () => {
    const likes = 50;
    const views = 0;

    const { getByText } = render(<PositiveFeedbackPercentage likes={likes} views={views} />);

    const positiveFeedbackText = getByText(/0% Positive Feedback/i);
    expect(positiveFeedbackText).toBeInTheDocument();
  });
});
```

Unit-тест для навігаційного меню:

```
import React from 'react';
import { render, fireEvent } from '@testing-library/react';
import Sidebar from './Sidebar';

describe('Sidebar', () => {
  it('renders correctly when open', () => {
    const { getByText } = render(<Sidebar isOpen={true} onClose={() => {}} />);

    const homeMenuItem = getByText(/Home/i);
    const closeButton = getByText(/×/);

    expect(homeMenuItem).toBeInTheDocument();
    expect(closeButton).toBeInTheDocument();
  });

  it('renders correctly when closed', () => {
    const { queryByText } = render(<Sidebar isOpen={false} onClose={() => {}} />);

    const homeMenuItem = queryByText(/Home/i);
    const closeButton = queryByText(/×/);

    expect(homeMenuItem).toBeNull();
    expect(closeButton).toBeNull();
  });

  it('calls onClose when close button is clicked', () => {
    const onCloseMock = jest.fn();
    const { getByText } = render(<Sidebar isOpen={true} onClose={onCloseMock} />);
```

```
    const closeButton = getByText(/×/);
    fireEvent.click(closeButton);

    expect(onCloseMock).toHaveBeenCalledTimes(1);
  });

  it('renders menu items correctly', () => {
    const { getByText } = render(<Sidebar isOpen={true} onClose={() => {}} />);

    const homeMenuItem = getByText(/Home/i);
    const aboutMenuItem = getByText(/About/i);
    const servicesMenuItem = getByText(/Services/i);
    const contactMenuItem = getByText(/Contact/i);

    expect(homeMenuItem).toBeInTheDocument();
    expect(aboutMenuItem).toBeInTheDocument();
    expect(servicesMenuItem).toBeInTheDocument();
    expect(contactMenuItem).toBeInTheDocument();
  });

  it('applies hover effect on menu item when hovered', () => {
    const { getByText } = render(<Sidebar isOpen={true} onClose={() => {}} />);

    const homeMenuItem = getByText(/Home/i);

    fireEvent.mouseOver(homeMenuItem);

    expect(homeMenuItem).toHaveStyle('background-color: #555');
  });
});
```

Unit-тест для компоненту навігаційного меню:

```
import React from 'react';
import { render, fireEvent } from '@testing-library/react';
import Sidebar from './Sidebar';

describe('Sidebar', () => {
  it('renders Sidebar component correctly', () => {
    const { getByText } = render(<Sidebar />);

    const toggleButton = getByText(/☰/i);
    const mainContent = getByText(/Your App/i);

    expect(toggleButton).toBeInTheDocument();
    expect(mainContent).toBeInTheDocument();
  });

  it('toggles the sidebar on button click', () => {
    const { getByText, queryByText } = render(<Sidebar />);
```

```
    const toggleButton = getByText(/三/i);

    fireEvent.click(toggleButton);

    const closeButton = getByText(/×/i);
    const mainContent = getByText(/Your App/i);

    expect(closeButton).toBeInTheDocument();
    expect(mainContent).toHaveStyle('margin-left: 0');

    fireEvent.click(toggleButton);

    expect(queryByText(/×/i)).toBeNull();
    expect(mainContent).toHaveStyle('margin-left: 250px');
  });
});
```