Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

РОЗРОБКА ТА ДОСЛІДЖЕННЯ WEB-ДОДАТКУ З ВИКОРИСТАННЯМ NEXT.JS

DEVELOPMENT AND RESEARCH OF A WEB APPLICATION USING NEXT.JS

спеціальність 122 Комп'ютерні науки

освітня програма «Комп'ютерні науки»

Виконав: здобувач вищої освіти
групи КНм-21
Левонюк Дмитро Сергійович
_____
(підпис)
Керівник: к.т.н., доцент
Кошелюк Віктор Андрійович
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«___» _____ 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Валерій Олександрович
_____
(підпис)

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерних наук*
Ступінь вищої освіти: *магістр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *122 Комп'ютерні науки*
Освітня програма: *«Комп'ютерні науки»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Валерій ЛІЩИНА
«14» травня 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Левонюк Дмитро Сергійович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Розробка та дослідження web-додатку з використанням Next.js»
Керівник к.т.н., доцент Кошелюк Віктор Андрійович
затверджені наказом закладу вищої освіти від «*14*» *травня* *2025* р. № 255/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи *«05» грудня 2025 р.*
3. Вихідні дані до роботи: *Сучасні методи і засоби розробки, технічна документація технологій розробки.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити):
Аналіз сучасного стану проблеми, існуючих методів і засобів її розв'язання, аналіз і вибір засобів проектування, опис функціонального наповнення об'єкта проектування, розробка й обґрунтування системного наповнення, експериментальне дослідження результативності предмету дослідження.

5. Перелік графічного матеріалу: 22 графічні рисунки, 10 лістингів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблематики за темою роботи та постановка завдань дослідження	*Кошелюк В. А.*		
Теоретичне дослідження та практична реалізація предмету дослідження	*Кошелюк В. А.*		
Експериментальне дослідження результативності предмету дослідження	*Кошелюк В. А.*		
Показник запозичень тексту	_____ %		
Інструментальна перевірка	*Кошелюк В. А.*		
Нормоконтроль	*Сачук В. О.*		
Гарант ОПП	*Ліщина В. О.*		

7. Дата видачі завдання *«14» травня 2025 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	*Провести огляд літературних джерел по темі кваліфікаційної роботи*	*до 30.06.2025 р*	
2	*Провести аналіз загальної проблеми і вибір напрямків дослідження*	*до 01.09.2025 р.*	
3	*Розробити функціональну схему роботи програмного продукту*	*до 01.10.2025 р*	
4	*Описати засоби розробки об'єкта проектування*	*до 15.10.2025 р.*	
5	*Практична реалізація об'єкта проектування*	*до 10.11.2025 р.*	
6	*Провести експериментальне дослідження результативності предмету дослідження*	*до 25.11.2025 р.*	
7	*Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру*	*до 05.12.2025 р.*	

Здобувач вищої освіти _____ Дмитро ЛЕВОНЮК

Керівник роботи _____ Віктор КОШЕЛЮК

АНОТАЦІЯ

Левонюк Д. С. Розробка та дослідження web-додатку з використанням Next.js. Рукопис.

Кваліфікаційна робота магістра ОП «Комп’ютерні науки». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота магістра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі роботи виконано аналіз проблематики за темою роботи та постановку завдань дослідження. У другому розділі магістерської роботи проведено теоретичне дослідження та подано практичну реалізацію предмету дослідження. Третій розділ присвячено експериментальному дослідженню результативності предмету дослідження. У висновках узагальнено інформацію, відображену у попередніх частинах. Результати розробок можуть бути застосовані в практичній діяльності.

Ключові слова: Web, Next.js, додаток, соціальна мережа.

ANNOTATION

Dmytro Levoniuk. Development and research of a web application using Next.js. Manuscript.

Master’s qualifying thesis of the OP «Computer Sciences». Lutsk National Technical University. Lutsk, 2025.

The master’s qualification work consists of an introduction, three chapters, conclusions, a list of used sources and appendices.

The problem area of the research is analyzed, and the research objectives are defined in the first chapter of the work. The second chapter of the thesis contains a theoretical study and the practical implementation of the research subject. The third chapter is devoted to the experimental evaluation of the effectiveness of the research subject. The conclusions summarize the information presented in the previous parts. Development results can be applied in practical activities.

Keywords: Web, Next.js, application, social network.

ЗМІСТ

ВСТУП

Майже кожен сьогодні користується онлайн сервісами, проводить час в інтернеті, щось дивиться, щось шукає. Хтось це робить, щоб поспілкуватися зі своїми родичами по WhatsApp чи Viber, хтось замовляє собі піцу через Glovo, а хтось, щоб просто відпочити, гортає стрічку в Instagram. Соціальні мережі зараз чи не найпопулярніший формат онлайн-платформ, де сотні мільйонів людей кожен день діляться своїм життям, переживаннями та спогадами, спостерігають за життям своїх знайомих, знаходять собі нових друзів і таке інше.

За останні роки фото та відео контент став дуже популярним серед молоді. Одні користувачі виставляють контент, а інші споживають його, залишають позитивні або негативні коментарі, роблять репости та ставлять вподобайки, просуваючи цей контент вгору. А серед усіх тем помітно вирізняється їжа. Всім цікаво дізнатися, хто де і що їсть, побачити візуально привабливі страви, дізнатись для себе новий рецепт чи просто надихнутись ідеями, щоб потім створити свій кулінарний шедевр.

Наукова новизна роботи полягає у тому, що пропонується концепція спеціалізованої соціальної мережі, орієнтованої на обмін кулінарним контентом. Запропонований підхід буде поєднувати функції класичних соціальних мереж і сайтів з рецептами. Це на мою думку дозволить підвищити зацікавленість користувачів, створити більш цілеспрямовану взаємодію між ними та сформувати тематичну спільноту.

Практична цінність роботи полягає у тому, що створюється зручний і функціональний інструмент для спільного обміну рецептами та кулінарними ідеями. Розроблений додаток посприяє розвитку тематичних спільнот і може бути використаний як основа для реального стартапу.

Однак для реалізації веб-сервісу недостатньо лише продумати його функціонал, а ще треба обрати технологію, яка здатна витримати любі навантаження, буде швидко працювати та мати сучасну архітектуру. Тому для реалізації було обрано саме Next.js, у якому поєднуються можливості React, коли

частина роботи виконується на сервері і з ним зручніше працювати з динамічним контентом, що дозволяє нам створювати легко масштабовані та зручні для користувачів веб-застосунки.

Мета дослідження – створити веб-додаток соціальної мережі для обміну рецептами страв, який забезпечить користувачів зручним доступом до рецептів, посприяє активній взаємодії між користувачами через коментарі, особисті повідомлення, вподобайки та підписки на інших авторів.

Об’єкт дослідження – процес створення веб-додатку соціальної мережі для обміну контентом та взаємодії користувачів у цифровому середовищі.

Предмет дослідження – багатофункціональний, але простий web-додаток соціальної мережі, призначений для вирішення таких завдань: публікація рецептів з описами; взаємодія користувачів через коментарі, особисті повідомлення та підписки; створення тематичної спільноти користувачів, які зацікавлені у food контенті; створення інтуїтивно зрозумілого інтерфейсу, що спрощує взаємодію між користувачами.

Завдання кваліфікаційної роботи:

– проаналізувати предметну область соціальних мереж та існуючі аналоги у сфері відео та фото контенту, визначити їх можливості, обмеження та перспективи;

– сформувати функціональні та нефункціональні вимоги до веб-додатку, визначити ролі користувачів, сценарії взаємодії та загальну архітектуру системи;

– обґрунтувати вибір технологічного стеку та засобів розробки, а також виконати теоретичне моделювання структури системи;

– розробити інформаційну модель системи, структуру бази даних та алгоритми взаємодії між основними компонентами веб-сервісу;

– провести експериментальне дослідження результативності розробленого продукту, оцінити його ефективність та швидкість при взаємодії користувача і системи.

Апробація результатів дослідження здійснювалася шляхом підготовки та публікації статті у збірнику «Студентський науковий вісник» (додаток Д).

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз сучасного стану предметної області

Соціальні мережі вже давно стали звичайною частиною життя кожної людини. Вони потрібні нам і для комунікації, і для творчості, а також там нереальна концентрація корисної і не дуже інформації. Взагалі соцмережі придумали для того, щоб зблизити користувачів, які зацікавлені у конкретних темах, у пошуку друзів, у створенні спільнот, всередині яких однодумці будуть обмінюватись між собою інформацією, емоціями та враженнями.

Перші соціальні мережі це були по суті сервіси для обміну повідомленнями, якісь форуми, які пройшли довгий шлях до теперішнього вигляду соціальних платформ, які стали частиною повсякденного життя сьогоднішніх користувачів. Тому варто коротко розглянути, з чого почалася історія соціальних мереж, яка була першою і коли трапився бум соціальних мереж.

Перша спроба об’єднати людей через інтернет була ще у 1970-х роках. Тоді з’явилися перші текстові чати, електрона пошта, а також Usenet – система груп новин, яка дозволяла користувачам обмінюватися повідомленнями. Ці групи новин були схожі на тематичні дошки, де користувачі могли публікувати свої повідомлення, відповідати на чужі і навіть обмінюватись файлами. Ця система функціонувала за допомогою величезної кількості взаємопов’язаних серверів, де кожне повідомлення після розміщення поширювалось по мережі, забезпечуючи його широку доступність. Хоч Usenet і не була соцмережею в сучасному плані, але прийнято вважати, що саме вона заклала основу для онлайн-спільнот. У 1990-х роках з’явилися перші сервіси, що нагадували сучасні соціальні мережі, серед яких найбільш відомою була Six Degrees. Six Degrees вже дозволяла користувачам створювати особисті профілі і додавати друзів, тобто виконувала базові функції теперішніх соцмереж і на той час це вважалося дуже інноваційним. На початку 2000-х років почалася активна популяризація сервісів

для взаємодії між користувачами. В той час з’явилися Friendster та MySpace, які надали більше можливостей для пошуку друзів і персоналізації сторінок. Це була ера, коли мільйони людей відкривали для себе онлайн-дружбу та культурні спільноти. У 2004 році був створений Facebook, який перевернув все з ніг на голову. Від студентського стартапу до гіганта, який об’єднує сотні мільйонів користувачів. У наступні роки з’явилися YouTube, Twitter, Instagram та пізніше Tiktok. Всі вони принесли разом із собою нові формати: сторіз, короткі відео та меми. Зараз це все частина сучасної культури, що народжується у соцмережах. Теперішні соцмережі – це вже не просто розваги, вони можуть впливати на речі куди серйозніші. І вони впливають на політику, культуру, бізнес і навіть на те, як ми бачимо себе так світ навколо.

Я можу умовно поділити соцмережі на кілька типів за їх цільовим призначенням. Першій тип назвемо універсальні соцмережі і до її легко можна віднести такі платформи, як Facebook і Twitter, це платформи для великої кількості користувачів і різного типу контенту. Заходячи у Twitter, ти можеш і фото переглянути, і пост написати, і відео подивитись. Окремо я можу виділити професійні соцмережі, зокрема LinkedIn, який використовується для пошуку роботи або стажування, обміну досвідом та в цілому розвитку твоєї професійної частини. На відміну від інших соцмереж, які все ж більш про розваги, LinkedIn виключно про роботу та ділову сферу. Також можна виділити тематичні платформи, які фокусуються на конкретній темі, будь то література чи фітнес. Наприклад Goodreads використовують для оцінювання прочитаних книг, написання рецензій, перегляду чужих рецензій, для пошуку нових книг на основі своїх вподобань та взаємодії з іншими читачами. І останню, але не менш важливу категорію становлять контент соцмережі, такі як Instagram та YouTube, у яких фокус на поширенні фото та відео контенту. Раніше користувачі сиділи на форумах і єдине, що вони могли робити, це відправляти текстові повідомлення, які не факт, що хтось читав, а зараз же існує така кількість платформ, де вони можуть самовиражатися, де їх контент може побачити цілий світ, де вони показуючи себе можуть зробити з себе власний бренд і почати на собі заробляти.

Людям простіше сприймати інформацію, коли вона подана візуально, а не просто описана текстом. Чому люди сьогодні обирають фільми, а не книги? Бо ти одразу бачиш цілу картинку, хто перед тобою на екрані, у що він одягнутий, що він робить і таке інше. Фото й відео в соцмережах давно стали нормальним способом просувати себе чи бізнес.

Якось так склалося, що люди створіння соціальні і всім нам цікаво як хто живе, у зв'язку з чим фото контент повсякденності ну дуже популярний. Якщо зараз зайти в Instagram можна побачити тонну фотографій із подорожей, як люди сьогодні вділися, чим люди сьогодні снідали в кафе і таке інше. Чому це всім так цікаво? По-перше це про індивідуальність, воно передає естетику людини, естетику її життя, життєві цінності та стиль людини. Фото сприймаються не тільки як момент фіксації певної події, а як спосіб самовираження та натхнення.

Зображення їжі набули особливої популярності впродовж останніх років. Останніми роками фотографування їжі перетворилося на окремий тренд, це справжній спосіб для самовираження у цифровому середовищі. Для когось це спосіб продемонструвати свої здібності, а хтось хоче просто поділитися, що вспів спробувати у своїй подорожі до Італії у красивому закладі. Також зображення їжі часто можна побачити у рекламі, яка може привернути потенційних клієнтів. Але на одних фотографіях кулінарний світ не зупинився, користувачі обмінюються текстовими рецептами там, обмінюються відео рецептами тут. Всі люди надихаються, всі навчаються чомусь новому, вони прагнуть не лише споживати інформацію, а й взаємодіяти між собою.

У будь-якій соцмережі стрічка давно підлаштовується під твої інтереси. Це коли система завдяки алгоритмам рекомендацій підбирає саме той контент, який відповідає твоїм інтересам, система аналізує на кого ти підписаний і коли ти оформлюєш підписку на кілька каналів однієї тематики, то вона сама починає тобі підсовувати канали цієї тематики. Але в більшості сучасних соцмереж алгоритми рекомендацій працюють так, щоб ти як користувач проводив на їх платформі більше часу, ніж планував.

Тому, як на мене, саме спеціальні соцмережі дозволять вирішити цю проблему. Зараз іде тенденція на індивідуалізацію соцмереж і формування невеликої, але активної та цілеспрямованої екосистеми. Любите читати книги, ось вам книжкова соцмережа, займаєтеся спортом ось вам соцмережа для спортсменів і таке інше. Тому я рахую, що створення соцмережі з рецептами, яка буде орієнтована виключно на food контент, на мій погляд цілком логічним продовженням сучасних тенденцій розробки.

Але не тільки соцмережі розвиваються, а ще й паралельно з ними розвиваються веб-технології, без яких цих соцмереж ніколи могло не існувати. Завдяки сучасним фреймворкам можна створювати веб-додатки, які одночасно складні всередині, і зручні для користувача, при цьому працюють швидко та добре разом із серверною частиною. Використовуючи ці технології ми спробуємо створити нову соціальну мережу для обміну фото та відео рецептами. Вона буде мати простий, але сучасний дизайн та широкій функціонал, у якому я спробує поєднати найкращі аспекти вже існуючих соцмереж.

Якщо коротко прогрес у програмуванні зробив можливим те, що ми маємо таку кількість соцмереж, однак жодна із них на мій погляд не задовольняє потреби людей, які зацікавлені лише у food контенті, що у свою чергу дає початок для розробки такої соцмережі, яка буде поєднувати простоту користування, буде мати інтуїтивно зрозумілий дизайн, тематику їжі і технічну ефективність.

Актуальність мого дослідження полягає у створенні зручного інноваційного web-додатку з фокусом на food контент та активну соціальну складову. Сервіс, який я розробляю буде поєднуватиме функціонал соцмережі, збірника рецептів та месенджера, що дасть можливість користувачу отримати комплексний досвід користування платформи. В нього буде можливість постити власні рецепти страв, обмінюватись рецептами з іншими користувачами, отримувати сповіщення, взаємодіяти з іншими користувачами платформи через написання коментарів та особистих повідомлень, підписуватись на інших авторів контенту і таке інше.

Результати, які я отримав у ході аналізу сучасного стану соцмереж стануть основою для подальшого проектування платформи, для вибору оптимального технологічного стеку та реалізації самого програмного продукту. Після аналізу стало зрозуміло, на яких моментах потрібно зосередитися: як саме користувачі взаємодіятимуть між собою, що вони очікують від сервісу і які функції варто реалізувати першими. Крім того, результати аналізу дадуть можливість продумати, як зробити платформу зручною та швидкою, і як забезпечити її подальше розширення та появу нових функцій у майбутньому.

1.2 Аналіз аналогічних програм, баз даних та систем

На сучасному ринку цифрових сервісів ми можемо побачити, що існує велика кількість соцмереж, які реалізують ідею обміну фото контентом, але жодна з них не поєднує всі необхідні можливості в межах однієї платформи за певною тематикою. Найвідомішим прикладом соцмережі для обміну фото є Instagram, представлений на рисунку 1.1. Він являє собою соцмережу, яка була побудована навколо обміну фотографіями. Людині треба один раз зареєструватися і він вже може ділитися власними фото, отримувати з них фідбек, читати коментарі під ними та відповідати коментаторам.

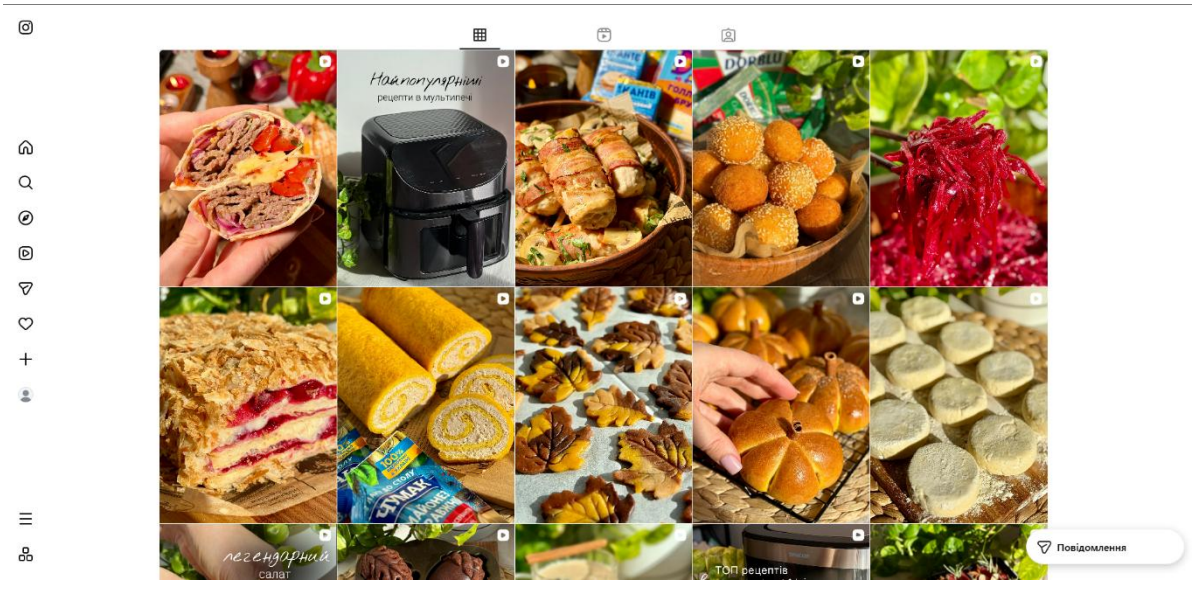

Рисунок 1.1 – Зовнішній вигляд web-додатка Instagram [1]

У Instagram стрічка підлаштовується під твої попередні вподобання. Ти бачиш контент, який відповідає тому, що тобі сподобався чи на кого підписувався раніше. Наприклад, якщо підписатися на кілька акторів, спортсменів чи кулінарних сторінок, коли ти в наступний раз відкриєш додаток, то твоя стрічка вся буде в акторах, спортсменах, їжі і не тільки з твоїх підписок. Система буде кожен раз рекомендувати тобі нові і нові обличчя в надії, що тобі хтось сподобається і ти на нього підпишешся. Але попри великий функціонал Instagram все одно залишається універсальною соцмережею і не має фокуса на кулінарній тематиці, що якраз мені і потрібно. Instagram більше про випадкові натхнення. Побачив красивий сніданок, натиснув зберегти і думаєш, що колись приготуєш, але це не спеціалізована кулінарна платформа.

Іншим популярним веб-ресурсом є Pinterest (рис. 1.2). Це така платформа натхнення, де користувачі шукають, зберігають та діляться фотографіями. Це як віртуальна дошка, де користувач вписує у пошуку, що його цікавить і потім кожен раз коли він буде вертатися на веб-сайт, то буде бачити багато фотографій на теми, які шукав в попередні рази.

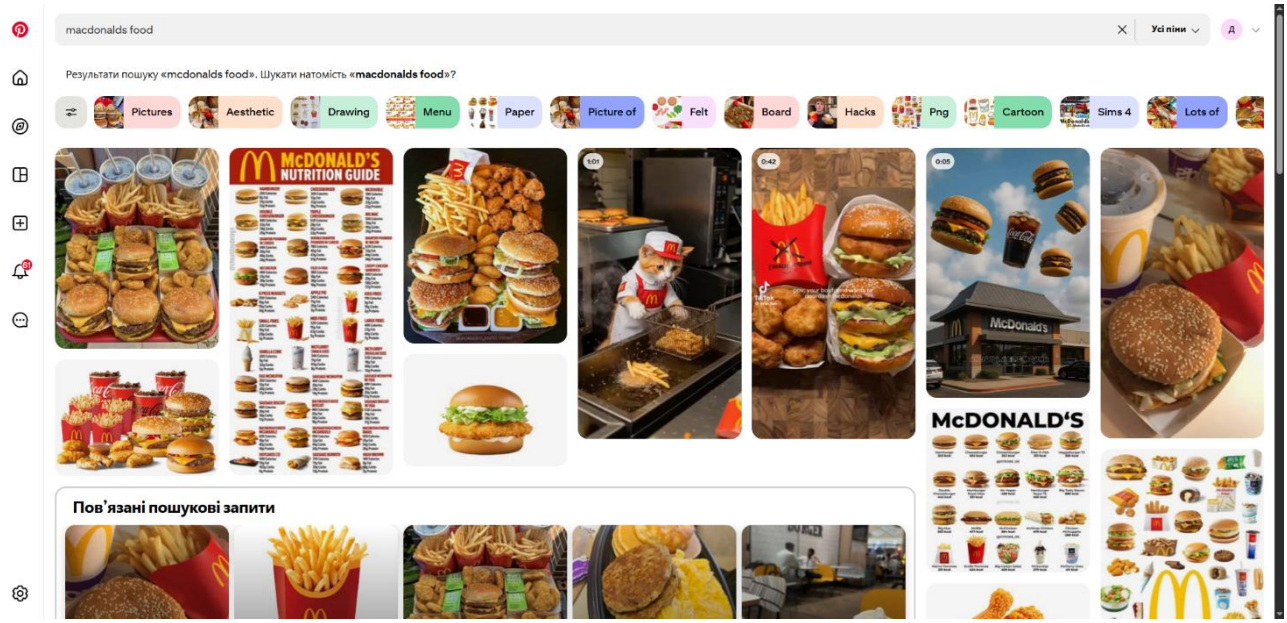

Рисунок 1.2 – Зовнішній вигляд web-додатка Pinterest [2]

Pinterest працює по системі пінів, в тебе є свій профіль і все що тобі подобається ти зберігаєш там по колекціям для подальшого використання. Люди використовують Pinterest насамперед як місце, де можна надихнутися та знайти ідеї для будь-якої сфери життя, будь то як облаштувати дитячу кімнату, чи що приготувати на сніданок. Pinterest також добре працює для бізнесу: там легко привернути увагу до товару. Тут легко можна просунути товар, який унікально виглядає або має цікавий функціонал, він буде привертати увагу і люди будуть переходити на сайт самого товару, який можна додати при публікації і можливо люди його куплять. Але якщо порівнювати з Instagram, то Pinterest майже не має соціальної складової, так тут є особисті повідомлення, але Pinterest більш про особисте колекціонування естетики, ніж про безпосереднє спілкування та обмін досвідом, а також сервіс охоплює мільйон тем, тому увага не до рецептів.

Серед більш спеціалізованих рішень на кулінарну тематику звичайно можна виділити всякі сервіси з рецептами типу Foodgawker (рис. 1.3). Так, вони тримають фокус на кулінарному контенті, там можна переглядати рецепти, ділитися своїми власними, зберігати ті що сподобались та коментувати чужі, але це просто рецепти і там не має естетики соцмережі.

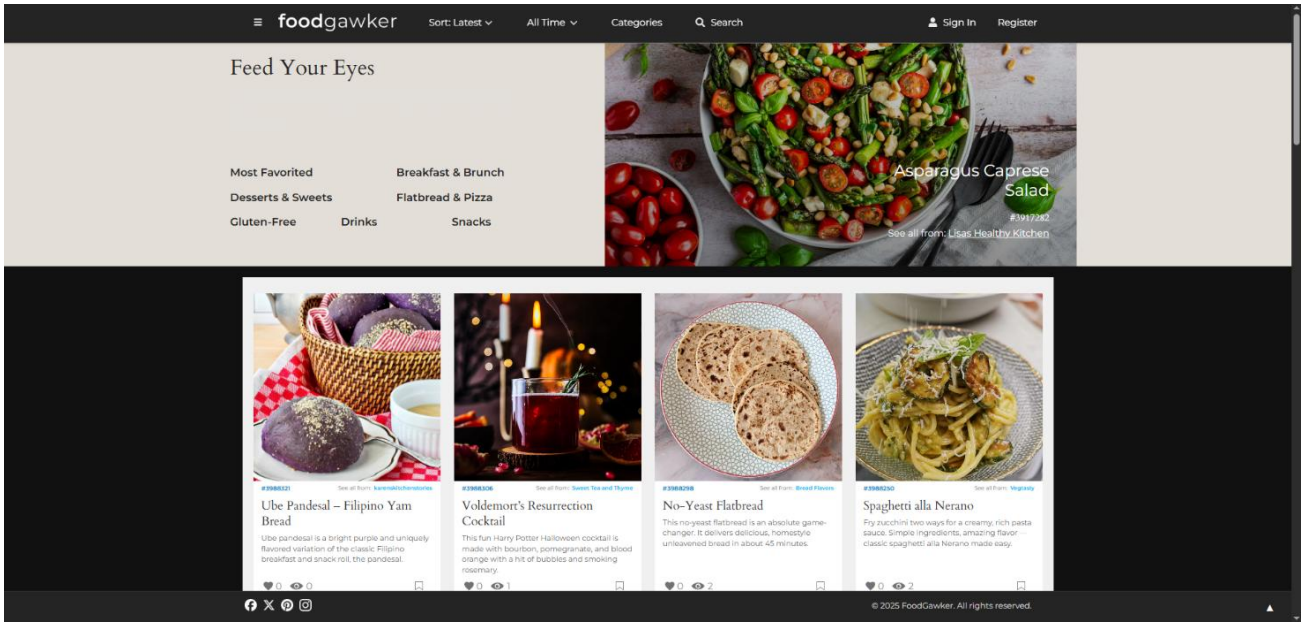

Рисунок 1.3 – Зовнішній вигляд web-додатка Foodgawker [3]

Вони мають дуже обмежений функціонал, якщо говорити про взаємодію між користувачами, там не має системи рекомендацій, перевантажений і менш інтуїтивно зрозумілий інтерфейс, якщо порівнювати його зі звичайними соцмережами. Крім того, візуальний аспект, який у 2025 році є важливим для користувачів, у таких сервісах відходить на другий план, він сильно застарілий і поступається місцем для текстових інструкцій та списку інгредієнтів. Такі сервіси на мій погляд сприймаються людьми більше як місце знань, а не як місце для натхнення чи спілкування. Тут бракує особливої атмосфери, яка є у звичайних соцмережах.

Після порівняння цих платформ можна побачити, що жодна не закриває всі потрібні вимоги, а саме соціальна взаємодія між користувачами, візуальна привабливість, інтуїтивна зрозумілість, простота у використанні та кулінарна тематика. Instagram має систему комунікації, але не має тематичної спрямованості. Pinterest має цікаву систему зі збереженням контенту, але слабкий для соціальної взаємодії і не має тематичної спрямованості. А спеціалізовані кулінарні сервіси взагалі не мають нічого спільного з соціальними мережами і виступають просто як збірник рецептів, де нульова соціальна взаємодія.

Тому і виникла ідея створити соцмережу, яка поєднає зручність сучасних платформ та фокус саме на кулінарії. Мета цього проекту створити платформу, де користувачі зможуть не лише шукати чужі і публікувати свої рецепти, не відволікаючись на зайві теми, а також взаємодіяти з іншими користувачами платформи, залишати коментарі, писати особисті повідомлення, надихатися новими ідеями та ділитися власним кулінарним досвідом. Також платформа дозволить структурувати кулінарний контент у більш зручній формі, що спростить пошук потрібних матеріалів та зменшить інформаційний шум. Завдяки продуманій системі фільтрів і тематичних добірок користувачів зможуть швидше знаходити рецепти, які відповідають їхнім вподобанням та рівню навичок. Усе це поступово формуватиме атмосферу підтримки і взаємодії, що є важливою складовою сучасних тематичних онлайн платформ.

1.3 Аналіз і вибір засобів проектування

Добре підібрати технологічний стек на початковій стадії розробки дуже важливо. Сьогодні існує безліч різноманітних інструментів, технологій та платформ, тому важливо підходити до аналізу можливостей з холодною головою. Кожна технологія має свої сильні та слабкі сторони, які можна виявити на ранніх етапах розробки. Саме тому я прийшов до порівняльного аналізу доступних варіантів.

Враховуючи, що проект вимагав створення сервісу з інтерактивним інтерфейсом, гнучким функціоналом та можливістю легкого масштабування, спершу розглядалися класичні серверні мови програмування, такі як PHP та Python. PHP – це скриптова сценарна мова програмування. Вона традиційно вважається стандартом у веб-розробці через легкість у використанні, підтримці на хостингах та наявності готових рішень, що сильно прискорюють розробку [4]. Майже всі круті фреймворки для веб-розробки написані саме на PHP. Серед них точно знайдеться той, що підходить під ваше завдання. На ньому можна писати застосунки для Windows, Linux та MacOS, в тому числі кросплатформні. PHP чудово підходить для побудови веб-додатків, які взаємодіють із базами даних, такими як MySQL, PostgreSQL чи SQLite. І попри появу нових мов і фреймворків, PHP залишається однією із найпоширеніших мов програмування для веб-розробки і до сіх пір приблизно 70 % усіх веб-сайтів все ще використовують її у тому чи іншому вигляді. Проте аналіз архітектури показав, що PHP більше підходить для створення традиційних веб-сайтів. Особливо в умовах сучасної розробки, де акцент змістився на швидку взаємодію користувача та миттєве оновлення даних, PHP на цьому фоні виглядає менш ефективним.

Ще одним потенційним варіантом був Python із фреймворком Django [5]. Цей варіант був привабливий за рахунок простоти коду, широкого набору готових компонентів та гнучкої архітектури. Django пропонує нам вбудовані інструменти для авторизації, маршрутизації, обробки форм та управління базами даних, що значно пришвидшує час, необхідний для розробки основного

функціоналу. До того ж Python відомий своїми бібліотеками для аналітики, що може бути корисним для подальшого аналізу поведінки користувачів. Django використовує Python, щоб працювати з базами даних за допомогою ORM (Object-Relational Mapping), що полегшує взаємодію з різними типами баз даних, що у свою чергу дозволяє писати код, який автоматично перетворюється в SQL запити, без необхідності безпосередньо працювати з самим SQL. Також Python і Django мають велику спільноту розробників, що сильно полегшує пошуки документацій та відповідей на будь-які запитання, які прийдуть в вашу голову. Незважаючи на переваги, коли мова йде про велику кількість одночасних з'єднань, то виникають певні обмеження. Django був створений для синхронної роботи, тож інтеграція асинхронних процесів є складнішою. У підсумку, через обмеження Django і потребу в адаптивному фронтенді було вирішено не використовувати Python, як основну технологію в кваліфікаційній роботі.

Було вирішено використовувати сучасний стек технологій на основі JavaScript, який за останні роки зарекомендував себе як універсальний інструмент як для front, так і для backend розробників. JavaScript – це високорівнева мова програмування, що дає змогу використовувати імперативний, а також функціональний та подієво-орієнтований підходи [6]. Використовування JavaScript дозволяє зменшити кількість мов у проекті та забезпечити кращу і простішу взаємодію між клієнтською та серверною частинами проекту. Особливий акцент можна зробити на бібліотеці React для створення інтерфейсу користувача. Завдяки активній спільноті, популярності та великої кількості бібліотек, React став ідеальним рішенням для створення сучасного, адаптивного та масштабованого інтерфейсу.

Для того, щоб об'єднати клієнтську та серверну частини в єдиному середовищі, було обрано фреймворк Next.js [7]. Він не тільки дозволить реалізувати серверний рендеринг, що покращить швидкість завантаження сторінок, але й забезпечить зручну систему маршрутизації, можливість створення API, автоматичну оптимізацію ресурсів та підтримку статичної генерації. Все це робить Next.js оптимальним вибором для розробки сучасного

веб-додатку, який буде враховувати всі вимоги бізнес-логіки, масштабованості та ефективності.

Таким чином, остаточне рішення про вибір інструментів для проекту було прийнято на основі технічних тестів, аналізу майбутніх потреб та відповідності вимогам до стабільності, продуктивності та інтерактивності. Обраний стек технологій є не тільки актуальним для ринку сьогодні, але й забезпечує міцний фундамент для подальшого розвитку та модернізації проекту в щось більше.

1.4 Постановка завдання на кваліфікаційну роботу магістра

Основною ціллю кваліфікаційної роботи є комплексне дослідження та створення веб-додатку у форматі соціальної мережі, який базується на сучасних принципах взаємодії користувачів із фото та відео контентом. Для досягнення цього результату передбачено виконання низки поетапних цілей, що охоплюють аналіз предметної області, формування вимог до системи, обґрунтування технологічних рішень, моделювання структури сервісу та оцінку його ефективності після реалізації.

Першою ключовою ціллю є детальний аналіз сучасних соціальних платформ, що працюють у сфері мультимедійного контенту. У межах дослідження планується розглянути популярні аналоги, оцінити їхні функціональні можливості, сильні та слабкі сторони, а також тенденції розвитку соціальних сервісів. Такий аналіз дозволить сформувати базу для проектування власного рішення та визначити, які підходи до роботи з контентом, стрічкою публікацій та взаємодією користувачів є найбільш ефективними та актуальними.

Наступною важливою ціллю є визначення чітких функціональних і нефункціональних вимог до веб-додатку, формування набору ролей користувачів і опис основних сценаріїв їхньої взаємодії із системою. В рамках цієї цілі планується сформувати загальну архітектурну концепцію майбутнього сервісу, яка включатиме логіку роботи клієнтської частини, серверного компонента та механізмів обробки мультимедійного контенту. Досягнення цієї

мети дозволить закласти міцний фундамент для подальшої розробки програмного продукту.

Третьою ціллю є обґрунтування вибору технологічного стеку, що буде використаний при реалізації додатку. Планується провести аналіз можливостей React, Next.js, Node.js та супутніх інструментів, визначивши їх переваги, обмеження та причини, з яких вони є доцільними для створення швидкого, адаптивного та масштабованого веб-сервісу. Паралельно з цим буде виконано теоретичне моделювання структури системи, включаючи логіку маршрутизації, рендерингу інтерфейсу та взаємодії клієнт сервер.

Окремою ціллю є розроблення інформаційної моделі системи та структури бази даних, що забезпечуватиме ефективне зберігання, обробку та відображення користувацького контенту. У рамках цієї цілі передбачається описати модель користувачів, структуру публікацій, алгоритми формування стрічки, принципи роботи вподобайок, коментарів, підписок та іншої взаємодії. Важливим аспектом є також побудова чіткої логіки роботи між компонентами сервісу, що гарантуватиме узгодженість даних і стабільність застосунку.

Завершальною ціллю кваліфікаційної роботи є експериментальна оцінка продуктивності створеного веб-додатку, під час якої планується провести вимірювання швидкості завантаження сторінок, ефективності роботи стрічки публікацій, швидкодії запитів та стабільності системи під навантаженням. Результати дослідження дадуть змогу оцінити якість реалізації, визначити потенційні напрями оптимізації та підтвердити відповідність створеного продукту сформульованим вимогам.

У результаті досягнення всіх поставлених цілей буде створено сучасний, функціональний та масштабований веб-додаток соцмережі, який демонструватиме практичну реалізацію теоретичних принципів веб-розробки та підтверджуватиме можливість використання обраних технологій для створення соціально орієнтованих сервісів з певною тематикою.

РОЗДІЛ 2
ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРЕДМЕТУ ДОСЛІДЖЕННЯ

2.1 Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання

Якщо сьогодні подивитися на будь-який всесвітньо відомий додаток, то він, швидше за все, написаний з використанням JavaScript. Це легко можна пояснити тим, що мова дуже гнучка, є велика кількість готових бібліотек та вона ефективно працює як на стороні клієнта, так і на стороні сервера. Саме тому для реалізації кваліфікаційної роботи було обрано стек технологій, який поєднує у собі сучасні інструменти, високу продуктивність та зручність у самій розробці.

Почну я з фреймворку Next.js, одного з найбільш відомих серед open-source фреймворків для створення веб-додатків. Його розробила компанія Vercel у 2016 році з метою спростити створення web-додатків на React з підтримкою SSR (Server-Side Rendering). І до його появи саме React вважали основою розробки інтерфейсу, але треба закрити очі на всі його плюси, бо у нього був помітний мінус – налаштування SSR забирало багатенько часу. Server-Side Rendering – це технологія, при якій веб-сторінка генерується не в браузері користувача, а на сервері і це робить завантаження сторінок набагато швидшим. Ви як користувач робите на сторінці запит, далі сервер отримує ваш запит, генерує повний код сторінки з усіма даними і відправляє його назад у ваш браузер, а ваш браузер у свою чергу швидко відображає вже готовий HTML. І щоб прискорити якраз налаштування SSR був розроблений свіжий фреймворк Next.js. Він був створений на базі React і має широкі можливості для рендерингу (SSG, SSR, ISR і API роути), маршрутизації (автоматичне формування маршрутів, яке відбувається на основі файлової структури в директорії pages), оптимізації веб-ресурсів (він в собі мав вмонтовані інструменти, які ненавмисно оптимізують завантаження компонентів і зображень) і в цілому покращення функціональності. Цей фреймворк буде простим у використанні, матиме гнучкий

редеринг, підтримуватиме TypeScript, що підвищує надійність коду і ще багато плюсів. Із недоліків я можу виділити: SSR вимагає більше серверних ресурсів для масштабніх проектів, JavaScript файли незважаючи на оптимізацію можуть важити багато і у деяких проектах налаштування SEO може додатково ускладнити процес розробки. Також треба нагадати, що Next.js добре працює щ сервісом Vercel, де можна легко і швидко розгорнути свій додаток, а також він дозволяє швидко створювати тестові середовищи.

React – це JavaScript бібліотека з відкритим кодом, яка використовується розробниками для створення інтерфейсів користувача [8]. Основна фішка бібліотеки React це комплексний підхід. Розробник може створити якісь незалежний компонент, наприклад кнопку і далі він цю кнопку може повторно використовувати та легко змінити при необхідності, що в свою чергу дуже полегшує розробку, тести та підтримку вашого коду. Всі ці елементи являють собою окремі інгредієнти остаточної страви, які в зібраному вигляді утворюють повноцінний користувацький інтерфейс веб-додатку. І саме цей підхід дозволив React стати настільки успішним в сфері фротенд-розробки. Сьогодні завдяки його швидкості і ефективності він застосовується для розробки тисяч самих різних додатків (Facebook, Netflix, Uber, Airbnb і багато інших). Він може використовуватись як у клієнта в браузері, так і на сервері, його використовують для розробки застосунків на IOS і Android, інтерфейси з React миттєво реагують на дії користувача і ще дуже багато плюсів. Якщо говорити за мінуси, то це не самий простий синтаксис для новачків, також React постійно оновлюється і тобі як розробнику потрібно завжди вдосконалюватись, вчитися чомусь новому і адаптуватися до змін, а ще одного React вам буде недостатньо для повноцінної розробки, це лише один інструмент, а не повноцінний варіант. Чому я його обрав і в чому його основна перевага? Завдяки йому розробники можуть писати модульний код, який можна використовувати вдруге, з ним легко працювати, його можна розширювати додатковими бібліотеками і фреймворками, він має активне ком’юніті розробників та купу усіляких інформаційних ресурсів для вдосконалення тебе як сучасного програміста.

TypeScript – це мова програмування, яку створила компанія Microsoft на початку 2010-х як альтернатива JavaScript [9]. Тоді JavaScript активно розвивалася, даючи забагато свободи та недостатньо контролю і коли проекти на JS розростались до величезних масштабів, то підтримувати ці проекти ставало ну дуже складно. Для чого нам взагалі цей TypeScript? Головна відмінність мови програмування TypeScript – це можна створювати стабільні та великі системи. Програмістам, які попадуть у в проект через рік, буде набагато простіше розібратися в коді тих, хто почав роботу. Статистична типізація це великий плюс, кожен програміст може просто зрозуміти, яка очікувана фінальна структура даних і типи аргументів функцій, що максимально полегшує роботу в колективі і допоможе запобігти майбутнім помилкам. Статична типізація – це механізм, при якому тип змінної визначається на етапі створення коду, а не під час компіляції, що дозволяє запобігти помилкам до запуску програми. TypeScript дозволив програмістам результативніше використовувати класи і інтерфейси, надав шанс створювати модульний код, який ви будете повторно використовувати, з ним вільніше трудитися в команді, він підтримується як для мобільної так і веб-розробки, має інтеграцію з усілякими бібліотеками та технологіями і ще багато різних фішок. Із недоліків можна виділити: складність на старті, більше часу на розробку, більше часу на компіляцію і важчі налаштування самого проекту. TypeScript став потужним інструментом в сучасній розробці, використовуючи його розробники можуть завчасно виявляти помилки, будувати надійніші застосунки і він має величезний потенціал незалежно від того над чим ви працюєте.

PostgreSQL – це безкоштовна та відкрита об’єктно-реляційна система управління базами даних [10]. Вона є одною із самих популярних СУБД і її використовують тисячі організацій по всьому світу. Це об’єктно-реляційна система тобто вона в собі поєднує переваги реляційних баз даних і об’єктно-орієнтованого програмування. Її активно використовують в різних галузях, у веб-розробці вона служить основую для динамічних web-додатків і застосунків. Наприклад, у вас є застосунок електронної комерції, якому треба обробляти

тисячі транзакцій, зберігати профілі клієнтів і забезпечувати стабільність і масштабованість – тоді ваш вибір PostgreSQL. Чи наприклад у соцмережі він може служити основою, де мільйони користувачів зможуть взаємодіяти, обмінюватись контентом і отримувати персоналізовані рекомендації. PostgreSQL без перебільшення можна використовувати у будь-яких проектах. Який це web-додаток без бази даних, він не зможе без її працювати у будь-якому випадку. Із плюсів є підтримка різних типів даних, можна працювати з великими обсягами роботи, PostgreSQL підтримує складні запроси, можна розширювати його функції безкоштовно в будь-який момент. Якщо говорити за мінуси, то це складність у налаштуванні (велика кількість функцій може збити новачків), підвищене споживання ресурсів (потрібно більше ресурсів, ніж іншим СУБД) і якщо порівнювати з комерційними аналогами то PostgreSQL трохи уступає по функціональності. У підсумку, це крута, структурована і безпечна система баз даних, її можна використовувати як для маленьких стартапів, так і для великих web-додатків та вона буде розвиватися разом із вашим проектом.

Neon – це хмарна, без серверна та повністю керована служба бази даних, яка базується на PostgreSQL. Вона була створена для спрощення управління базами даних. Використовуючи її, користувачу не потрібно турбуватися про обслуговування, Neon бере на себе відповідальність за патчі та оновлення, це проект з відкритим кодом та обчислювальні ресурси та сховище в ньому розподілені і масштабуються окремо. Як я її використовую у своїй роботі? Я обрав для себе СУБД (PostgreSQL) і ця PostgreSQL розміщена на сервері Neon, але не це все, для зв’язку мого коду і хмарного сервісу Neon мені потрібна ORM (Object-Relational Mapping), для спрощення взаємодії з базою даних. ORM – це технологія, яка об’єднує компоненти об’єктно-орієнтованого програмування з базою даних. Ось як це працює повністю: мій застосунок Next.js взаємодіє з ORM системою, яка перетворює звичайний код у SQL запити до віддаленої бази PostgreSQL, яка знаходиться на хмарному сервісі Neon. Як ви можете бачити зв’язка ORM і Neon працює та нові записи з’являються на своїх місцях. На сторінці керування Neon ви можете побачити структуру бази даних, список

доступних таблиць, також можна додати нові записи вручну, відфільтрувати наявні та проаналізувати, що взагалі відбувається у вас з таблицями (рис. 2.1).

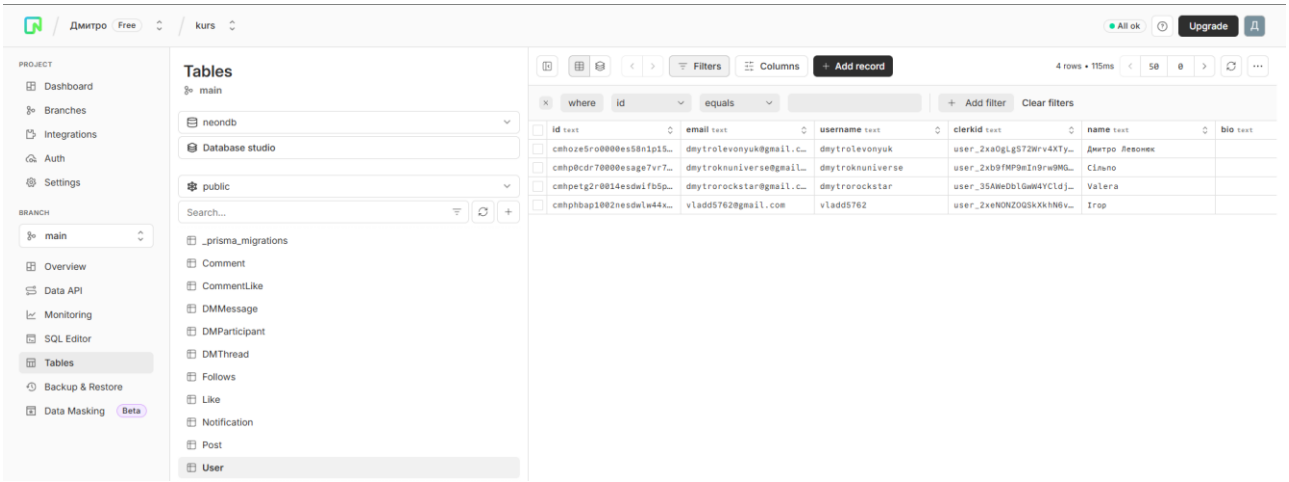

Рисунок 2.1 – Демонстрація роботи хмарного сервісу Neon [11]

Prisma – це сучасний ORM інструмент, який сильно полегшує вашу роботу с базами даних [12]. Якщо більш точно, то це технологія, яка дозволяє працювати з реляційними і нереляційними базами даних за допомогою JavaScript чи TypeScript. Замість написання складних SQL запитів, Prisma дає нам зручний API, який дозволяє виконувати операції з даними за мінімально можливий час. Також вона має систему міграцій на основі змін, які відбулися в вашій схемі. Наприклад, у початковій структурі в мене були моделі User, Like та Post і я захотів додати на свій сайт коментарі, я додаю у свій файл shema.prisma модель Comment, після внесення змін проводжу міграцію і автоматично створюється відповідна таблиця в базі даних. Але не без мінусів і великим мінусом є відсутність зворотної міграції, тобто не можна відкотити міграцію, наче її не було і єдиним варіантом все повернути є переробка міграції.

Radix UI – це бібліотека нестилізованих React компонентів, які представляють собою будівельні блоки для створення інтерфейсів користувачів. Radix UI легко інтегрується з популярними фреймворками і конкретно в моїй роботі з Next.js. Компоненти Radix UI за замовчуванням доступні і оптимізовані, а також їх легко можна стилізувати за допомогою всіх знайомих нам бібліотек і

фреймворків. Ще у Radix UI дуже активна спільнота програмістів і купа інформаційних ресурсів для вдосконалення, включаючи офіційну документацію, форуми і таке інше.

Мною було вирішено для цього проекту не розробляти свою окрему систему авторизації і реєстрації, а взяти щось з готових варіантів і знайшов для себе Clerk. Clerk – це платформа для автентифікації і управління користувачами, яка дуже швидко і легко інтегрується в ваш проект. Clerk дотримується найвищих стандартів безпеки даних ваших клієнтів, є готові компоненти, які треба просто додати. Як виглядає реєстрація: користувач натискає кнопку зареєструватися, перед ним відкривається вікно реєстрації, можна зареєструватися одразу через google акаунт чи через email і пароль, після цього на ваш email прийде одноразовий код і після цього ви вже вважаєтесь зареєстрованим і система вас додає в список користувачів (рис. 2.2). Також у налаштуваннях є двофакторна автентифікація (2FA), вона працює по смс і email.

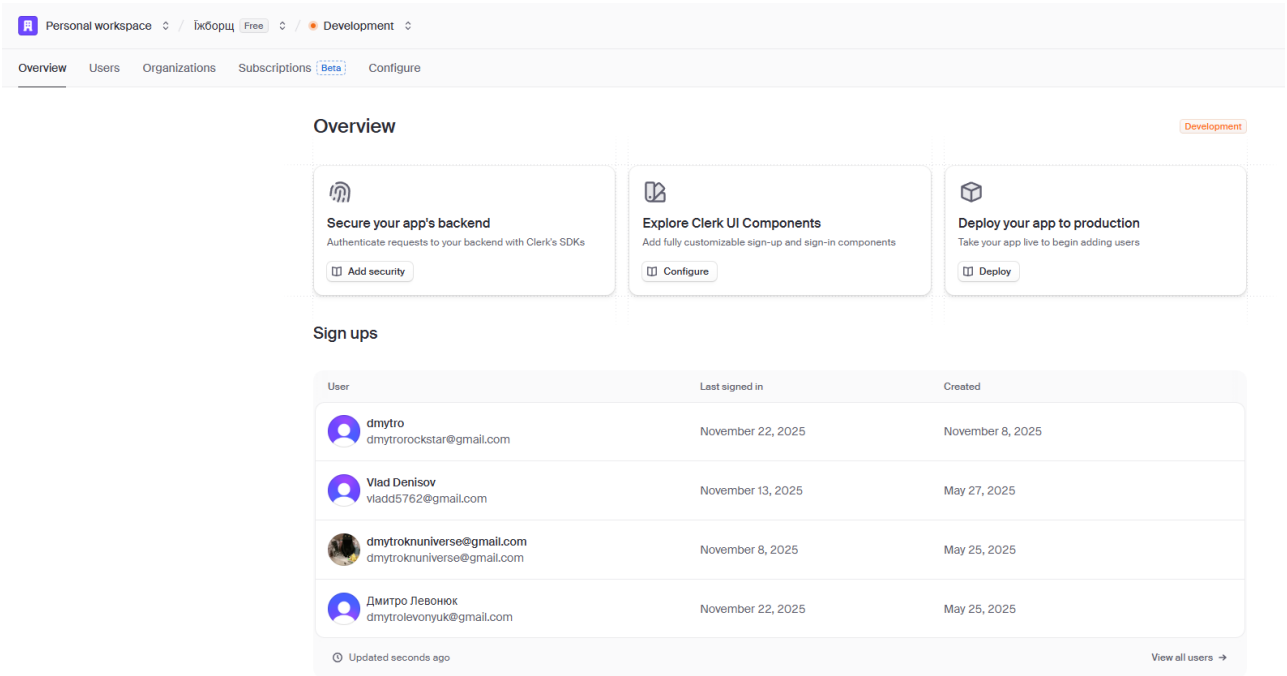

Рисунок 2.2 – Демонстрація роботи сервісу Clerk [13]

Ні одна соцмережа не може бути без відео та фото контенту і цей контент треба якось завантажувати та десь зберігати і для цього мені потрібне якесь

хмарне файлове сховище. Зазвичай web-додатки користуються сервісом S3 (Amazon Simple Storage Service), але я хотів знайти альтернативу і здається знайшов. UploadThing – сервіс, який спрощує завантаження файлів. Сервіс від Amazon важко настроювати, потрібно писати backend під завантаження, а з UploadThing все набагато простіше. І тому для спрощення роботи з файлами в проекті я застосовував саме його (рис. 2.3).

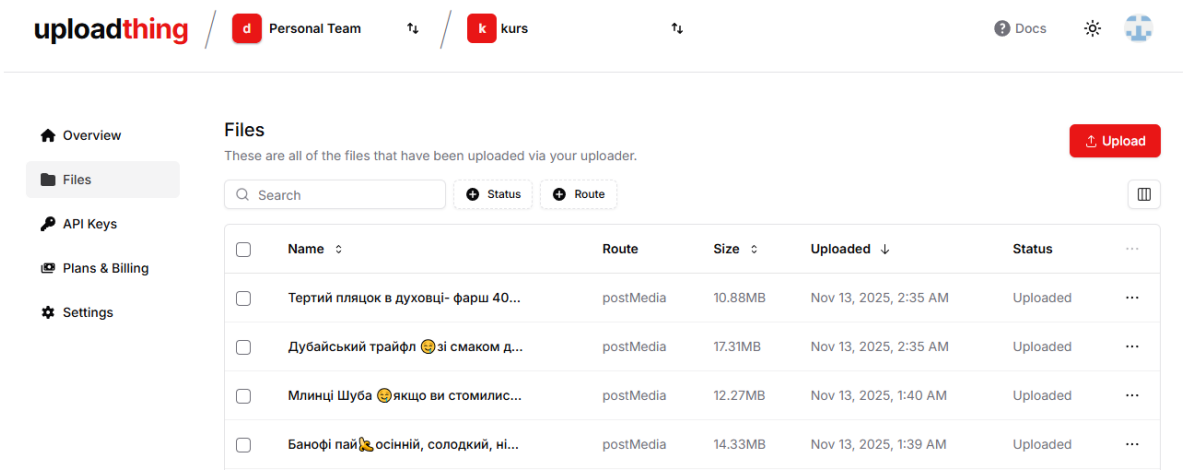

Рисунок 2.3 – Демонстрація роботи сервісу Uploadthing [14]

Також однією з функцій моєї соцмережі будуть особисті повідомлення. Ви зможете надсилати запит на переписку з любим користувачем і якщо він захоче, він схвалить цей запит і ви зможете переписуватись. Також в переписці ви зможете відправляти емодзі, фото та відео. І щоб не нагружати UploadThing увесь медіа контент з переписки буду зберігатися на Cloudinary. Cloudinary – це платформа для управління та оптимізації зображень і відео, яка забезпечує їх швидку доставку і просту інтеграцію у web-додатки. Вона надає нам інструментарій для швидкого стиснення любих файлів, автоматичної конвертації у сучасні формати, а також для інших маніпуляцій з медіа контентом з метою покращення продуктивності. Cloudinary пропонує API інтерфейси та гнучкі можливості адміністратора для інтеграції з новими та вже існуючими веб-додатками. Забігаючи наперед вибір Cloudinary реально став дуже вдалим рішенням, а розділення файлів особистих повідомлень та файлів пов’язаних з

публікаціями рецептів дозволило краще структурувати дані, спростити їх обробку та підвищите безпеку (рис. 2.4).

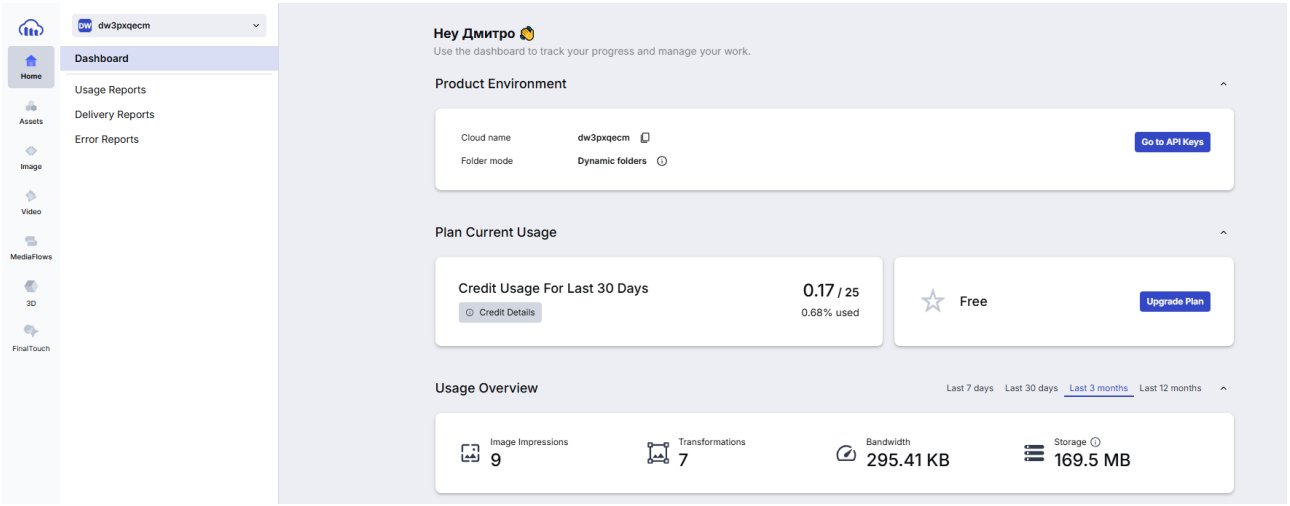

Рисунок 2.4 – Демонстрація роботи сервісу Cloudinary [15]

2.2 Функціонально-структурна схема роботи об’єкта проектування

Ціль системи – створення ефективної сучасної платформи для обміну рецептами страв, яка забезпечить користувачів зручним доступом до рецептів та посприяє активній взаємодії між ними.

Основні задачі системи:

– надати користувачам актуальні кулінарні рецепти: забезпечення доступу до нових рецептів на платформі;

– реалізувати функціонал для завантаження, редагування та видалення контенту: можливість завантажувати, редагувати та видаляти рецепти;

– створити систему фільтрів: реалізація системи, яка надає можливість пошуку за новизною, популярністю і за категоріями страв;

– реалізувати систему реєстрації та авторизації: реалізація системи реєстрації та авторизації користувачів з підтвердженням пошти, двофакторною автентифікацією та з можливістю відновлення пароля;

– реалізувати систему сповіщень: реалізація системи, при якій кожна вподобайка, коментар, вподобайка на ваш коментар, відповідь на ваш коментар,

підписка на ваш профіль чи особисте повідомлення приходить на ваш акаунт у формі сповіщення;

– реалізувати систему коментарів із підтримкою відповідей: реалізація системи, при якій кожен авторизований користувач може залишити під вашим рецептом коментар, поставити вподобайку на свій чи любий інший коментар, відповісти на любий коментар і також видалити свій коментар;

– реалізувати систему особистих повідомлень: реалізація системи, при якій кожен авторизований користувач може відправити іншому користувачу запит на спілкування і якщо той користувач його підтвердить, то вони можуть спілкуватися, відправляти емодзі, фото та відео один одному, а також видаляти свої повідомлення;

– реалізувати особисту сторінку для кожного користувача: реалізація особистої сторінки, де кожен авторизований користувач зможе змінювати інформацію про себе, змінювати аватарку, ставити своє місцезнаходження, додавати посилання на інші соцмережі, переглядати свої дописи рецептів, а також переглядати рецепти, яким сам користувач поставив вподобайку;

– реалізувати систему підписок: реалізація системи підписок, при якій кожен авторизований користувач зможе підписуватись на творців контенту, чиї рецепти йому сподобалися і система з наступного заходу на веб-сервіс почне частіше йому показувати рецепти саме цього користувача.

Функціональні характеристики системи, що розробляється:

– початкове наповнення бази даних: користувачі можуть публікувати власні фото та відео рецепти, коментувати та відповідати на коментарі, ставити вподобайки, підписуватись на інших авторів та відправляти особисті повідомлення;

– оновлення та видалення інформації: система дозволяє автору у разі потреби оновити чи видалити раніше створений допис з рецептом, коментар чи особисте повідомлення;

– забезпечення безпеки даних: web-додаток доповнений системою Clerk, а її розробники дуже переймаються за безпеку ваших даних під час авторизації;

– забезпечення цілісності даних: система включає механізми для збереження цілісності даних та запобігання їх втраті;

– гнучка система сповіщень: користувачі отримують сповіщення за бодай кожну дію пов’язану з його контентом;

– адаптивний дизайн: інтерфейс системи оптимізований для використання на різних пристроях, включаючи мобільні телефони та планшети.

Далі для розробки веб-додатку були визначені основні ролі в системі, ключові сценарії її використання, можливі помилки та системні вимоги. Це дозволило мені як розробнику чітко зрозуміти поведінку користувачів, як компоненти працюють між собою та умови для коректної роботи веб-додатку.

Актори:

– неавторизований користувач (User): може переглядати рецепти, фільтрувати рецепти, переглядати коментарі та відповіді на них, але не може взаємодіяти з контентом (сам не може нічого створювати і не може писати в особисті повідомлення);

– авторизований користувач (Registered User): після реєстрації або авторизації може створювати рецепти самостійно, ставити вподобайки, коментувати, відповідати на коментарі, підписуватися на інших користувачів, писати в особисті повідомлення, змінювати інформацію про себе в профілі, переглядати свої попередні та обрані рецепти, а також він отримує сповіщення про активність інших користувачів на своїй сторінці.

Основний сценарій використання для неавторизованого користувача:

− користувач заходить на web-додаток;

− сервер відправляє рецепти у стрічку користувача;

− користувач переглядає рецепти, коментарі та відповіді на них, але без можливості взаємодії.

Основний сценарій використання для авторизованого користувача:

− користувач заходить на web-додаток;

− користувач реєструється або проходить авторизацію у системі Clerk;

− система надає повний функціонал;

− користувач створює нову публікацію, пише до рецепта опис та натискає кнопку завантажити;

− користувач ставить вподобайку та коментує рецепт свого друга;

− користувач відповідає на негативний коментар під своїм рецептом;

− користувач підписується на нового для себе автора;

− користувач переглядає свої сповіщення;

− користувач пише в особисті повідомлення своєму другові;

− користувач вирішує видалити невдалий рецепт зі своєї сторінки;

− користувач змінює про себе дані у профілі;

− користувач закриває web-додаток.

Альтернативний сценарій з можливими помилками.

У користувача відсутній інтернет:

− користувач заходить на web-додаток;

− браузер користувача видає помилку і повідомляє, що користувач не може увійти;

− користувач закриває web-додаток.

Особливі системні вимоги:

− сервер повинен обробляти велику кількість запитів з мінімальними затримками;

− інтерфейс користувача повинен однаково чітко відображатися у всіх браузерах на всіх гаджетах;

− система повинна забезпечувати захист персональних даних користувачів та безпеку автентифікації.

Для демонстрації роботи особистих повідомлень мною було створено діаграму послідовності. Діаграма послідовності (Sequence Diagram) – це діаграма, на якій ми можемо побачити взаємодії об’єктів, упорядковані за часом їхнього прояву. Організованість за часом слід розуміти як послідовність дій, що відбуваються один за одним і не треба плутати з часовими діаграмами. Діаграми послідовності дуже корисні при проектуванні функціоналу пов’язаного з

обміном даними, як наприклад у нашому випадку з обміном особистими повідомленнями (рис. 2.5).

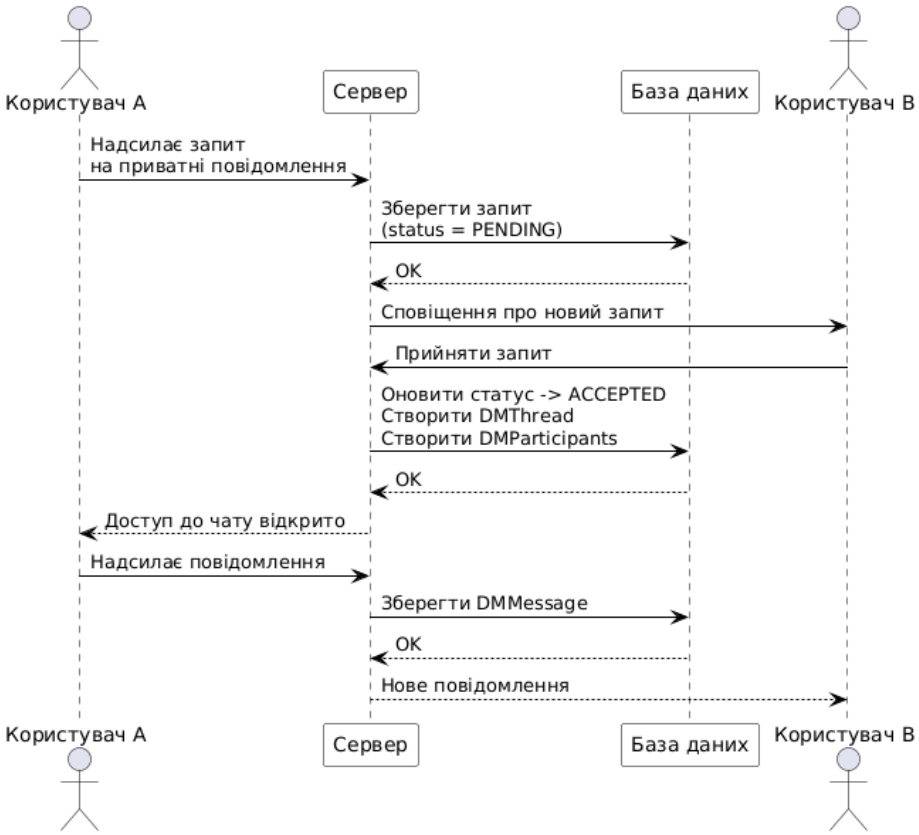

Рисунок 2.5 – Діаграма послідовності (Sequence Diagram)

Джерело: розроблено автором

Алгоритм обміну приватними повідомленнями між користувачем А і В:

– користувач А надсилає запит на початок листування;

– сервер приймає запит і зберігає його у базі даних зі статусом PENDING (в очікуванні);

– база даних підтверджує збереження цього запиту;

– сервер надсилає користувачу В повідомлення про новий запит на листування;

– користувач В отримує запит і приймає його;

– сервер оновлює статус запиту і зараз він ACCEPTED (прийнятий), далі створюються DMThread (чат) і DMParticipants (учасники чату);

– база даних підтверджує створення чату;

– користувач А зараз має доступ до приватного чату;

– користувач А надсилає повідомлення;

– сервер зберігає повідомлення в базі;

– база даних підтверджує збереження;

– сервер надсилає користувачу В нове повідомлення.

На діаграмі класів (рис. 2.6) детально зображені основні моделі системи, їх атрибути та взаємозв’язки, що надає нам чітке уявлення про внутрішню структуру та функціональні можливості кожної моделі, реалізованої за допомогою ORM Prisma.

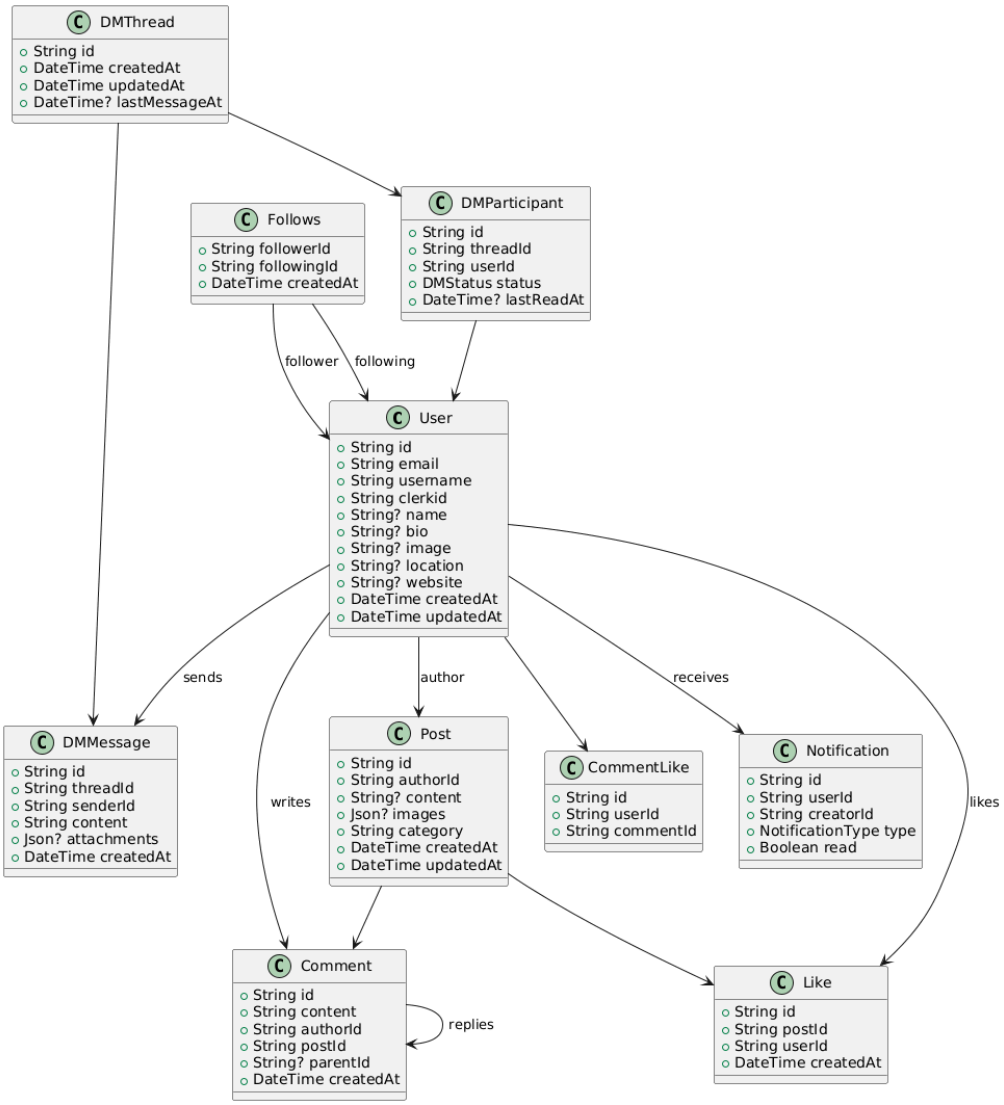

Рисунок 2.6 – Діаграма класів (Class Diagram)

Джерело: розроблено автором

Опис основних моделей:

– модель user: вона містить дані про користувача (id, email, ім’я, юзернейм, біографію і таке інше), також ця модель бере участь у кожній частині функціоналу web-додатку (створення публікацій, коментарів, вподобайок, підписок, сповіщень і приватних повідомлень), це по суті серце системи;

– модель post: вона зберігає дані про публікацію (опис, фото, відео та категорію рецепта), також ця модель пов’язана з коментарями, вподобайками та сповіщеннями;

– модель comment: вона містить текст коментаря, інформацію хто і де його залишив, підтримує вкладені коментарі (відповіді), а також співпрацює з моделями вподобайок на коментар (CommentLike) та сповіщеннями користувачів;

– модель like: вона проводить фіксацію, коли користувачі жмуть вподобайку (userId та postId) і також вона використовується у сповіщеннях;

– модель commentLike: з назви не важко здогадатися відповідає за вподобайки на коментарях і також генерує сповіщення;

– модель follows: зберігає дані про підписки кожного користувача і теж задіяна у сповіщеннях;

– модель notification: зберігає дані про сповіщення (стан прочитання, хто ініціатор та хто одержувач), використовується для інформування користувачів про активність на їх сторінці;

– модель dmThread: представляє приватний чат, де зберігається час створення, час оновлення і коли було останнє повідомлення;

– модель dmParticipant: містить інформацію про участь користувача у чаті, статус доступу (в очікувані, прийнятий і заблокований), а також час останнього прочитання;

– модель dmMessage: вона зберігає приватні повідомлення та вкладення (фото, відео та pdf файли), містить посилання на відправника, використовується у системі особистого листування.

2.3 Практична реалізація об’єкта проектування

Для першого етапу розробки моїм стеком мені знадобиться Visual Studio Code [16], Node,js і npm. Першим ділом відкриваємо термінал та вставляємо скопійовану з офіційного веб-сайту Next.js команду для ініціалізації проекту (npx create-next-app@latest). У результаті виконання команди ми маємо готову початкову структуру, яку ми згодом розширимо відповідно до функціональних вимог (рис. 2.7).

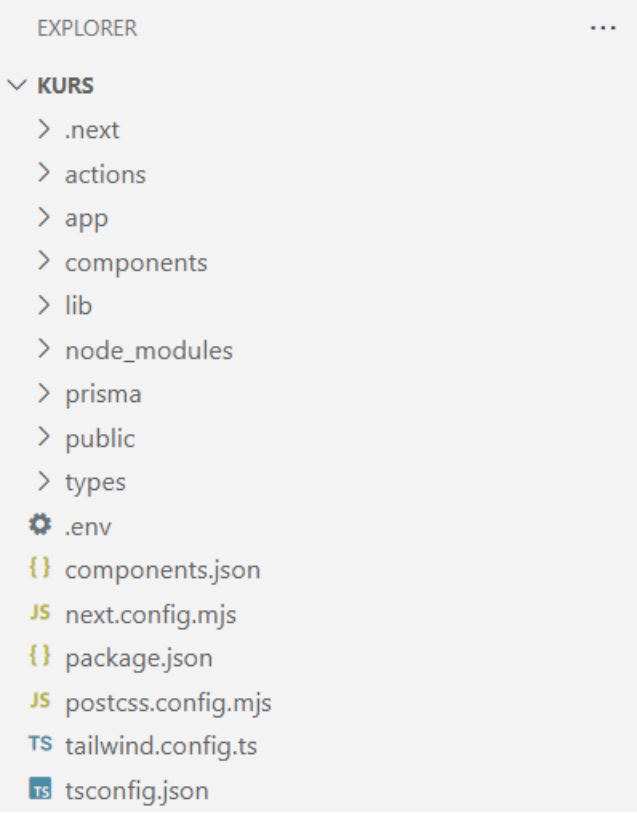

Рисунок 2.7 – Структура файлів та директорій проекту

Джерело: розроблено автором

Основну логіку застосунку ви можете переглянути у папці app. Там я прописав шрифти, там знаходяться майже всі мої сторінки (сповіщення, повідомлення та сторінка особистого профілю), серверні та клієнтські компоненти, а ще там в папці app є папка api, і вже в цій папці прописані Cloudinary та UploadThing. У каталозі components я зібрав всі багаторазові

елементи (усілякі кнопки, фільтрація, завантаження зображення, карточка самої публікації і таке інше). Наступна папка actions і вона містить серверні функції (пов’язані напряму з користувачем, публікаціями, сповіщеннями, створенням самих дописів і так далі), тобто ті функції, які виконуються на стороні сервера та напряму працюють з базою даних. У директорії lib зосереджені файли помічники, які я використовую у рандомних частинах застосунку. Наступне саме цікаве, каталог prisma там знаходиться моя schema.prisma, у якій розписані всі моделі, всі атрибути, що с чим працює і на основі цієї схеми Prisma створює таблиці, які ми можемо особисто побачити на хмарному сервісі Neon. Папку public я використав для зберігання там логотипу веб-додатку та аватарці користувача, у якого її не має (дефолтна іконка з чоловічком). У файлі .env я тримаю всі дані, які не можна показувати – для підключення до хмарної бази даних, ключі авторизації для сервісу Clerk, токени для роботи різних сервісів та конфіденційні url адреси. Всі ці папки окремо нічого не варті, але коли вони разом, то вони формують міцну та впорядковану структуру web-додатку.

Як же я розробляв головну сторінку свого веб-сайту? Все почалося з функції shuffle, яка потрібна нам, щоб рандомити рецепти на головній сторінці. Мені хотілося, щоб користувач оновлюючи сторінку кожен раз бачив нові рецепти, але допоки я не зробив систему рекомендацій як наприклад в Instagram, то найкращим варіантом стала функція shuffle (перетасовати), при якій кожне ваше оновлювання сторінки просто рандомить підбірку рецептів. Далі було додано функцію Home. Вона відповідальна за отримання авторизованого користувача та завантаження контенту із бази даних. Також була закладена структура сторінки, а конкретніше центр займають форма для створення ваших рецептів та панель фільтрації по новизні, популярності і категоріям страв. А в правій частині я розташував маленький блок з рекомендаціями, типу вам можуть бути цікаві ці автори контенту, ви можете клікнути та перейти до них у профіль і при бажанні підписатися. І звичайно на головній сторінці було реалізовано динамічне оновлення при виконанні любої дії (створення нової публікації,

фільтрація та оновлення сторінки). Код головної сторінки моєї соціальної мережі для обміну рецептами наведено у лістингу 2.1.

Лістинг 2.1 – Файл page.tsx

```
      /*...*/
      function shuffle<T>(arr: T[]): T[] {
        return arr
          .map((value) => ({ value, sort: Math.random() }))
          .sort((a, b) => a.sort - b.sort)
          .map(({ value }) => value); }
      export default async function Home() {
        const user = await currentUser();
        const dbUserId = await getDbUserId();
        let postsFromDb = await getPosts();
        postsFromDb = shuffle(postsFromDb);
        const posts = postsFromDb.map((post) => ({
          ...post,
          images: Array.isArray(post.images)
            ? post.images.filter((img): img is { url: string; type: string
} => typeof img === "object" &&
                img !== null &&
                "url" in img &&
                "type" in img): [], }));
        return (
          <div className="grid grid-cols-1 lg:grid-cols-10 max-w-[1200px]"
            style={{ marginRight: "50px" }}>
            <div className="lg:col-span-8">
              {user && <CreatePost />}
              <FilterBar posts={posts} dbUserId={dbUserId} />
            </div>
            <div className="hidden lg:block lg:col-span-2 sticky top-20">
              <WhoToFollow />
            </div></div>
        );
      }
```

кінець лістингу 2.1

Далі я вам розкажу вам про принцип роботи компонента DesktopNavBar, який відповідає за коректне відображення верхньої панельки навігації для комп’ютерів. Спочатку DesktopNavBar дивиться хто до нього прийшов, це авторизований користувач, чи це не авторизований і перевіряє він це за допомогою сервісу Clerk. І вже на основі цієї інформації компонент відображає кнопки переходу до сповіщень, повідомлень, профілю, а також значок вашого

акаунту з аватаркою (якщо вона поставлена). Але якщо користувач не авторизований, то для нього з’явиться кнопка реєстрації. Також тут є компонент ModeToggle, який я зробив для перемикання темної і світлої теми веб-додатку. А ще всі кнопки сторінок, які бачить авторизований користувач я доповнив відповідними іконками, щоб верхня панелька не виглядала занадто пусто. Код компоненту DesktopNavBar було наведено у лістингу 2.2.

Лістинг 2.2 – Файл DesktopNavbar.tsx

```
     /*...*/
     async function DesktopNavbar() {
       const user = await currentUser();
       return ( <div className="hidden md:flex items-center space-x-4">
     <ModeToggle />
<Button variant="ghost" className="flex items-center gap-2" asChild>
          <Link href="/">
         <LayoutDashboardIcon className="w-4 h-4" />
     <span className="hidden lg:inline">Головна</span></Link>
     </Button> {user ? ( <>
<Button variant="ghost" className="flex items-center gap-2" asChild>
                 <Link href="/notifications">
             <BellRingIcon className="w-4 h-4" />
<span className="hidden lg:inline">Сповіщення</span></Link></Button>
<Button variant="ghost" className="flex items-center gap-2" asChild>
           <Link href="/messages">
             <SendIcon className="w-4 h-4" />
<span className="hidden lg:inline">Повідомлення</span></Link></Button>
<Button variant="ghost" className="flex items-center gap-2" asChild>
                 <Link href={`/profile/${
                     user.username ??
                 user.emailAddresses[0].emailAddress.split("@")[0] }`}>
                 <SmileIcon className="w-4 h-4" />
      <span className="hidden lg:inline">Профіль</span></Link></Button>
     <UserButton /> </> ) : (
            <SignInButton mode="modal">
              <Button variant="default">Зареєструватися</Button>
            </SignInButton>)}</div>);}
     export default DesktopNavbar;
```

кінець лістингу 2.2

Я хотів, щоб головна сторінка web-додатку не здавалась пустою, тому я зробив справа невеличкий блок WhoToFollow, де користувачу буде відображати кілька рандомних акаунтів, на які можна підписатися. Спочатку ми робимо

запит, щоб отримати випадковий список користувачів із бази даних. Якщо база пуста, то цей блок просто не буде відображатися, але якщо хтось в базі є, то цей блок з’явиться. Далі я прописав кожному користувачу аватарку, ім’я та кількість його підписників. Ви можете клікнути на аватарку і перейти в профіль, або одразу підписатися, я справа від кожного користувача додав кнопку підписки. Таким чином в мене вийшов невеличкий, але дуже корисний блок рекомендацій, що буде спонукати людей шукати нових творців рецептів. Код компоненту WhoToFollow наведено у лістингу 2.3.

Лістинг 2.3 – Файл WhoToFollow.tsx

```
     /*...*/
     async function WhoToFollow() {
       const users = await getRandomUsers();
       if (users.length === 0) return null;
       return (
         <Card className="w-72">
       <CardHeader>
         <CardTitle>Можуть вам сподобатися:</CardTitle>
       </CardHeader><CardContent>
         <div className="space-y-4">
           {users.map((user) => (
  <div key={user.id} className="flex items-center justify-between gap-2">
   <Link href={`/profile/${user.username}`} className="flex gap-2">
                 <Avatar>
                   <AvatarImage src={user.image ?? "/avatar.png"} />
                 </Avatar>
                 <div className="text-xs">
                   <p className="font-medium">{user.name}</p>
                <p className="text-muted-foreground">@{user.username}</p>
            <p  className="text-muted-foreground">{user._count.followers}
підписників</p>
                 </div></Link>
               <div className="flex flex-col items-end">
                 <FollowButton userId={user.id} />
               </div></div>))}</div></CardContent></Card>);}
     export default WhoToFollow;
```

кінець лістингу 2.3

Коли я закінчив роботу над головною сторінкою web-додатка, треба було думати, а що далі, які ще сторінки зазвичай мають всі соцмережі. Звичайно це особиста сторінка профілю, весь код сюди би не вліз, тому я виніс всю логіку у

серверний компонент ProfilePage (ліст. 2.4). Спочатку ми пробуємо дістати дані про користувача через функцію getProfileByUsername і якщо нам це не вдається, то нас перенаправляють на сторінку з помилкою, де вам пишуть, що такого користувача не існує і ви можете повернутись назад, клікнувши на кнопку попередня сторінка. Інша ситуація, коли ми все ж таки знайшли користувача. Коли профіль знаходиться, починають підгружатися дописи цього користувача і дописи, які йому сподобалися (я там окремі 2 кнопки зробив, щоб між ними переключатися). Далі по коду перевіряється чи підписаний той, хто зараз переглядає профіль, на цього автора, це ми робимо по id з бази даних. Ну і людина відповідно бачить кнопку підписатися чи відписатися. І все все це передається у клієнтський компонент, який займається рендерингом сторінки. Тобто інтерфейс це клієнтський компонент, а логіка і запити це серверний.

Лістинг 2.4 – Файл ProfilePage.tsx

```
/*...*/
interface PageProps {
  params: { username: string };
}
export default async function ProfilePage({ params }: PageProps) {
  const { username } = params;
  const user = await getProfileByUsername(username);
  if (!user) return notFound();
  const posts = await getUserPosts(user.id);
  const likedPosts = await getUserLikedPosts(user.id);
  const dbUserId = await getDbUserId()
  const following = dbUserId ? await isFollowing(user.id) : false;
  return (
    <ProfilePageClient
      user={user}
      posts={posts}
      likedPosts={likedPosts}
      isFollowing={following}
    />
  );
}
```

кінець лістингу 2.4

Далі я думав, чого ще не хватає моїй соцмережі і прийшов до того, що не має можливості відправляти особисті повідомлення своїм друзям і треба з цим

щось зробити. Я хотів додати функцію переписок більш для спілкування з друзями, але як система буде відрізняти хто друг, а хто нє. В моїй голові була думка, щоб могли переписуватись тільки ті, хто взаємно підписаний. Проте я вирішив, що всі зможуть писати всім, але з невеличким нюансом. Спочатку ти типу відправляєш запит на спілкування, а людина по той бік вирішує чи хоче вона з тобою мати справи. Розроблена мною сторінка для особистих переписок MessagesPage винесена у додаток А. На цій сторінці є дві кнопки (вхідні та запити) та список ваших діалогів. Для того, щоб коректно відображалися всі переписки я створив невеличкий компонент, де вказано ім’я, аватарка людини та останнє написане їм повідомлення, ну щоб ти відкривав цю сторінку і одразу бачив який діалог тобі треба. Також всі компоненти було мною вирівняно по центру екрану, щоб очі користувачів не розбігалися і звичайно ну все клікабельне.

В програмуванні класно, коли щось можна розділити на окремі елементи, щось 1 раз зробити і потім скрізь по коду це використовувати. І взагалі розподіляти обов’язки, розподіляти код на серверний і клієнтський, ну це круто. Так і я у своєму проекті, не хотів мішати клієнтський інтерфейс і серверну логіку, благо в Next.js є така штука як actions. Це невеличкі модулі, кожен з яких відповідаю за конкретну частину функціоналу. Вони в мене і для особистих повідомлень, і для сповіщень, і для публікацій, ну для кожної можливої функції. Це крутий підхід, при якому компоненти, про які я розписував раніше, займаються тільки відображенням, а вся робота з даними на плечах у екшенів, які виконуються на сервері і тому виконуються безпечно та швидко.

Для роботи зі сповіщеннями було створено окремий екшен getNotifications. З назви не важко здогадатися, що він головний у справах отримання всієї активності, що стосується користувача. Нагадаю, що люба дія додатково створює сповіщення, яке приходить користувачу на сторінку сповіщень. Спочатку функція перевіряє чи авторизований користувач перед нею і якщо ні, то вона просто нічого зайвого йому не показує. А якщо все ж таки авторизований, то далі наша ORM Prisma звертається до таблиці в базі даних і витягує всю потрібну

інформацію за допомогою одного запиту. Другий екшн для сповіщень (markNotificationsAsRead) відповідальний за маркування сповіщення як прочитаного. На сторінці зі сповіщеннями справа у блоці є поле з кількістю непрочитаних, яке зменшується, коли ви заходити на сторінку. Функція приймає список ID і коли список порожній, то вона просто завершує роботу. А якщо ID є, то Prisma виконує масове оновлення встановлюючи всім прапорці read: true для вибраних сповіщень. Код файлу notification.action.ts було наведено у лістингу 2.5.

Лістинг 2.5 – Файл notification.action.ts

```
     /*...*/
     export async function getNotifications() {
       try { const userId = await getDbUserId();
         if (!userId) return [];
         const notifications = await prisma.notification.findMany({
           where: { userId },
include: { creator: {select: {id: true, name: true, username: true,
      image: true,},},
post: {select: {id: true, content: true, images: true,},},
comment: {select: {id: true, content: true, createdAt: true,},},
replyComment: {select: {id: true, content: true, createdAt: true,},},},
         orderBy: { createdAt: "desc" },});
         return notifications;} catch (error) {
         console.error(" Error fetching notifications:", error);
         throw new Error("Failed to fetch notifications");}}
export async function markNotificationsAsRead(notificationIds: string[])
       try { if (!notificationIds  notificationIds.length === 0) {
           return { success: true };}
         await prisma.notification.updateMany({
          where: { id: { in: notificationIds } }, data: { read: true },});
         return { success: true };} catch (error) {
         console.error(" Error marking notifications as read:", error);
         return { success: false };}}
```

кінець лістингу 2.5

Як я вже казав вище, є ситуація, коли користувач відкриває профіль, а профілю не існує і щоб це його не шокувало був створений наступний компонент. Ця сторінка відповідає за зрозуміле відображення помилки 404 (сервер не може знайти запитану веб-сторінку або ресурс). Тут в коді ви можете побачити я розмістив число помилки, а під помилкою текст, що такого користувача не було знайдено. Що в такий ситуації робити людині? А все дуже

просто, тут я додав дві кнопки, щоб вона могла повернутися на попередню сторінку, чи вийти одразу на головну сторінку. Цей компонент потрібен, щоб користувач став більш проінформований о ситуації та міг одразу повернутися на веб-сайт і продовжити їм користуватися. Код файлу not-found.tsx було наведено у лістингу 2.6.

Лістинг 2.6 – Файл not-found.tsx

```
     /*...*/
     export default function NotFound() { return (
         <div className="min-h-[80vh] grid place-items-center px-4">
         <Card className="w-full max-w-md"><CardContent className="pt-6">
         <div className="text-center space-y-6">
         <p className="text-8xl font-bold text-primary font-mono">404</p>
          <div className="space-y-2">
     <h1  className="text-2xl  font-bold  tracking-tight">Користувача  не
знайдено</h1>
     <p className="text-muted-foreground">Користувача, якого ви шукаєте,
не існує </p> </div>
       <div className="flex flex-col sm:flex-row gap-3 justify-center">
                <Button variant="default" asChild><Link href="/">
                      <HomeIcon className="mr-2 size-4" />
                       Головна </Link></Button>
                   <Button variant="outline" asChild><Link href="/">
                       <ArrowLeftIcon className="mr-2 size-4" />
                         Попередня сторінка
                      </Link></Button></div></div></CardContent>
          </Card>
         </div>
       );
     }
```

кінець лістингу 2.6

Моделюємо ситуацію, у якій ви користувач мого web-додатку. Ви створили публікацію з вашим рецептом і вам він перестав подобатися. Ви обрали його і клікнули на іконку з корзиною – все, публікацію видалено. Потім ви походили, добре подумали, повернулися на веб-сайт, а все, рецепт треба загружати з нуля, не має кнопки повернути. Чи ще тупіша ситуація, ви випадково натиснули на корзину і допис видалився. І щоб такого було менш, мною було створено окремий компонент DeleteAlertDialog. Він відкривається у вигляди вікна, коли ви натискаєте на корзину рядом з вашою публікацією. У цьому вікні

дві кнопки (скасувати та видалити) і відповідно натискаючи на одну, у вас просто закривається вікно, а інша кнопка – запускає процес видалення. Це зроблено для зручності користувача, щоб він зайвий раз подумав перед тим, як щось видаляти. Код файлу DeleteAlertDialog.tsx було наведено у лістингу 2.7.

Лістинг 2.7 – Файл DeleteAlertDialog.tsx

```
     /*...*/
     interface DeleteAlertDialogProps {
       isDeleting: boolean;
       onDelete: () => Promise<void>;
       title?: string;
       description?: string; }
     export function DeleteAlertDialog({ isDeleting, onDelete,
       title = "Видалити публікацію",
description = "Цю дію не можна скасувати.", }: DeleteAlertDialogProps) {
       return (<AlertDialog><AlertDialogTrigger asChild>
             <Button variant="ghost"size="sm"
            className="text-muted-foreground hover:text-red-500 -mr-2">
 {isDeleting ? (<Loader2Icon className="size-4 animate-spin" /> ) : (
                 <Trash2Icon className="size-4" /> )} </Button>
           </AlertDialogTrigger><AlertDialogContent><AlertDialogHeader>
              <AlertDialogTitle>{title}</AlertDialogTitle>
           <AlertDialogDescription>{description}</AlertDialogDescription>
             </AlertDialogHeader><AlertDialogFooter>
               <AlertDialogCancel>Скасувати</AlertDialogCancel>
               <AlertDialogAction onClick={onDelete}
         className="bg-red-500 hover:bg-red-600" disabled={isDeleting}>
         {isDeleting ? "Видаляється..." : "Видалено"} </AlertDialogAction>
              </AlertDialogFooter></AlertDialogContent></AlertDialog>
       );
     }
```

кінець лістингу 2.7

В мене соцмережа з рецептами, а це значить, що якось треба завантажувати фото та відео матеріали, і для цього нам знадобиться наступний компонент ImageUpload, винесений у додаток Б. Я зробив так, що коли ти пересуваєш фото та відео в спеціальний блок у формі створення рецептів, вони одразу списком з’являються у вигляді міні прев’ю. Ще біля кожного фото чи відео є хрестик і можна зайве фото легко прибрати. Також я додав обмеження на максимальну кількість файлів, гадаю для рецептів 15 фото, більш ніж достатньо, а відео можна

загрузити лише одне. У результаті ImageUpload показав себе як гарний універсальний інструмент для керування файлами.

Щоб користувачі могли легко підписуватись один на одного, я реалізував наступний компонент FollowButton (ліст. 2.8). Спочатку він бере ID людини, на яку ви хочете підписатися і визиває серверний екшн toggleFollow. Поки виконується запит до бази даних, я блокую кнопку, а замість тексту на кнопці вам показується анімація. Якщо підписка вдалася, вам приходить повідомлення, що дія виконана, а якщо не вдалося підписатися відповідно виникає помилка. Цей компонент я зробив більше для своєї зручності, один раз створив і багато раз повторно використав.

Лістинг 2.8 – Файл FollowButton.tsx

```
     /*...*/
     function FollowButton({ userId }: { userId: string }) {
       const [isLoading, setIsLoading] = useState(false);
       const handleFollow = async () => { setIsLoading(true);
         try {await toggleFollow(userId);
           toast.success("User followed successfully");
         } catch (error) {
           toast.error("Error following user");
         } finally {
           setIsLoading(false);
         }
       };
       return (<Button
           size={"sm"}
           variant={"secondary"}
           onClick={handleFollow}
           disabled={isLoading}
           className="w-20" >
           {isLoading ? <Loader2Icon className="size-4 animate-spin" /> :
"Стежити"} </Button>
       );
     }
     export default FollowButton;
```

кінець лістингу 2.8

Хотілося, щоб у кожної сторінки мого веб-сайту була чітка структура і щоб це все було зібрано десь в одному місті, тому я створив наступний компонент Layout. Тут підключається все: шрифти, сервіс авторизації Clerk та оформлення

всього застосунку. Першим ділом я використав шрифти Geist Sans та Geist Mono, вони мене одразу зачіпили, коли я їх побачив. Далі я обгорнув цілий додаток у ClerkProvider, який відповідальний за систему авторизації та надає доступ до даних поточного користувача на любій сторінці. Далі я підключив ThemeProvide, він потрібен нам, щоб люди могли перемикати тему веб-сайту з темної на світлу і навпаки. Далі я додав SyncUserClient, який є відповідальним за актуальність інформації про поточного користувача у базі даних (він міняє аватарку і SyncUserClient одразу працює). У внутрішній структурі ви можете побачити навігаційну панельку, яка є на всіх сторінках зверху екрану, та основний контейнер для контенту, який залежить від сторінки і в кінці чисто для себе, щоб бачити помилки я додав компонент зі спливаючими повідомленням. Таким чином в мене вийшов гарний скелет, який задає всім сторінкам один стиль і об’єднує їх у повноцінний веб-додаток. Код файлу layout.tsx було наведено у лістингу 2.9.

Лістинг 2.9 – Файл layout.tsx

```
     /*...*/
     const geistSans = localFont({ src: "./fonts/GeistVF.woff", variable:
"--font-geist-sans", weight: "100 900", });
     const geistMono = localFont({ src: "./fonts/GeistMonoVF.woff",
     variable: "--font-geist-mono", weight: "100 900", });
     export const metadata: Metadata = { title: "Їжборщ", description:
"Їжборщ", };
     export default function RootLayout({ children, }: {
       children: React.ReactNode; }) { return ( <ClerkProvider>
           <html lang="uk" suppressHydrationWarning>
            <body className={`${geistSans.variable} ${geistMono.variable}
antialiased`}>
      <ThemeProvider attribute="class" defaultTheme="system" enableSystem
                disableTransitionOnChange >
            <SyncUserClient /> <div className="min-h-screen"><NavBar />
           <main className="py-8"> <div className="flex justify-center"
          style={{ paddingLeft: "300px", paddingRight: "325px", }} >
      <div className="w-full max-w-[1200px]">{children}</div></div>
</main></div><Toaster /></ThemeProvider></body></html>
</ClerkProvider>);
}
```

кінець лістингу 2.9

Як було зазначено вище, на моєму веб-сайті є можливість спілкуватися особистими повідомленнями, спочатку кидати запит, а потім вже спілкуватися. І ось для вікна діалогу мені потрібен наступний компонент ThreadPage, винесений у додаток В. Першим ділом перевіряємо, чи авторизований поточний користувач і якщо ні, то кидаємо його на сторінку з помилкою. Далі за допомогою ID завантажуємо діалог і визначаємо, хто з користувачів поточний, а хто людина якій він хоче написати. Також якщо якісь дані загубилися, зламалися або ще щось, то діалог, щоб уникнути помилок просто не відобразиться. Далі в нас формується інтерфейс самого чату, зверху аватарка та ім’я вашого друга, щоб ви розуміли кому пишете. Трохи нижче списком ідуть повідомлення, які можна скролити мишкою і подивитись старі повідомлення. А якщо ви у ситуації, коли перший раз зайшли в діалог до людини (коли ще не кинули запит на спілкування), там буде написано, щоб ви написали першим. Ще тут є окрема логіка для запитів. Коли запит має статус в очікуванні (PENDING), то користувач буде бачити перед собою блок прийняти запит. Після цього серверний екшн змінює статус діалогу і ви можете повноцінно спілкуватися. Ще забувся розказати про компонент MessageBuble, який тут підключений для інтерактивності повідомлень. Якби його не було, то не було видалення повідомлень, подвійного кліку для редагування і видалення, анімацій і так далі. Цей компонент (ThreadPage) працює як повноцінна сторінка особистого чату, охоплює всі сценарії та надає користувачу доступ до всіх функцій.

На головній сторінці моєї соцмережі з рецептами у мене є сітка з публікацій типу як в Instagram і ось наступний компонент відповідає за її відображення і зовнішній вигляд. Компонент PostGrid був винесений у додаток Г і відповідає він за сітку публікацій на головній та особистих сторінках усіх користувачів. Спочатку він отримує список дописів і формує з них 3 стовпчики, які будуть безкінечно скролитись поки не закінчяться. На кожному рецепті ми бачимо перше зображення, а якщо це відео, то можемо навести курсор і воно почне відтворювання. Також коли людина наводиться курсором на любий рецепт, можна побачити кількість вподобайок і коментарів, а якщо він клікне, то

відкривається окреме вікно допису з усією інформацією, коментарями і так далі. Я хотів, щоб сітка рецептів мала вигляд великої галереї як на телефоні і здається мені вдалося.

У 2025 році темна тема у застосунках це світовий стандарт, якого дотримуються всі сучасні цифрові гіганти і мій проект не міг його пропустити, тому я створив компонент ModeToggle. Він був створений для перемикання світлої та темної теми веб-сайту. Для нього мною було використано бібліотеку next-themes, проект все ж таки на Next.js і вона синхронізує тему застосунку з системними налаштуваннями пристрою користувача. Коли людинка натискає на кнопку змінити тему, іконка сонця змінюється на місяць і все це під красиву анімацію. На мій суб'єктивний погляд темна тема і візуально більш підходить моїй соцмережі з рецептами і в вечірній час набагато комфортніше знаходитися на веб-сайті, очі не болять. Реалізацію компоненту ModeToggle було наведено у лістингу 2.10.

Лістинг 2.10 – Файл ModeToggle.tsx

```
"use client";
import * as React from "react";
import { MoonIcon, SunIcon } from "lucide-react";
import { useTheme } from "next-themes";
import { Button } from "@/components/ui/button";
export default function ModeToggle()
{ const { theme, setTheme } = useTheme();
  return
(
    <Button
      variant="outline"
      size="icon"
      onClick={() =>
        setTheme(theme === "dark" ? "light" : "dark")}>
       <SunIcon className="h-[1.2rem] w-[1.2rem] rotate-0 scale-100
transition-all dark:-rotate-90 dark:scale-0" />
       <MoonIcon className="absolute h-[1.2rem] w-[1.2rem] rotate-90
scale-0 transition-all dark:rotate-0 dark:scale-100" />
      <span className="sr-only">Змінити тему</span>
    </Button>
  );
}
```

кінець лістингу 2.10

Спочатку мною було створено головну сторінку web-додатку. Якщо йти зверху і донизу, то спочатку іде логотип та назва соціальної мережі (Їжборщ), а трохи далі навігаційна панелька з такими пунктами: головна, сповіщення, повідомлення і профіль. Ну і відповідно ця панель дозволяє швидко переходити на потрібну сторінку, а кнопка головна повертає вас на сторінку Home. Також ліворуч від цих кнопок знаходиться перемикач світлої та темної теми веб-сайту. Нижче я додав блок створення нового рецепту з попереднім переглядом, тобто ви одразу бачити, як буде виглядати фінальна публікація. Ще була реалізована функція завантаження фото та відео, плюс є вибір категорії рецепту і кнопка публікації. Праворуч я додав блок рекомендацій, щоб веб-сайт не здавався пустим. Там користувачу будуть пропонуватися рандомні автори контенту, на яких він може підписатися. Під блоком створення публікацій я зробив систему фільтрів. Ви можете зробити сортування за категорією плюс новизною чи популярністю. А вже під фільтрами знаходиться сітка дописів, у який у 3 стовпчики розташовані публікації типу як в Instgram. Кожен елемент інтерактивний, можна навести курсор і побачити кількість вподобайок та коментарів, чи клікнути та відкрити повноцінне вікно публікації з можливістю написати свій коментар. Головну сторінку можна побачити на рисунку 2.8.

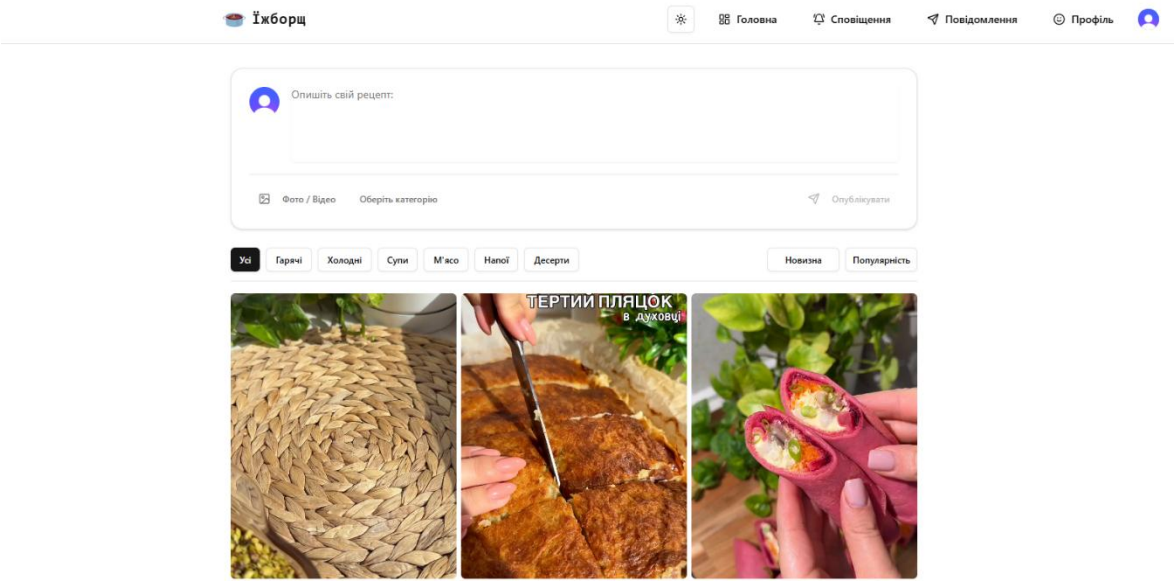

Рисунок 2.8 – Демонстрація роботи page.tsx

Джерело: розроблено автором

На головній сторінці я хотів об’єднати всі необхідні базові функції, які є у кожної соціальної мережі: перегляд чужого контенту, створення особистого, фільтрування вже існуючого та навігація між сторінками веб-сайту. Зовнішній вигляд виконаний у сучасному мінімалістичному стилі, темний фон і контрастні світлі назви кожного елементу добре читаються та зменшують навантаження на очі під час тривалого використання соцмережі. Далі я хочу зробити сторінку для сповіщень, особистих повідомлень та профілю поточного користувача. Щоб кожна сторінка мала сенс, певний функціонал і всі вони виглядали в одному загальному стилі.

На веб-сайті підключена дуже приємна система Clerk, яка сильно спрощує процес реєстрації та авторизації користувачів. Це готове рішення, яке ти просто підключаєш у свій проект і через секунду у тебе всі необхідні функції, які пов’язані з авторизацією та реєстрацією і тобі не треба нічого самостійно розробляти з нуля, це дуже круто і сильно скорочує обсяг власноруч написаного коду, а також це суттєво зменшить кількість можливих помилок у системі. Користувач, коли перший раз відкриває вікно реєстрації, може ввести свою пошту, або скористатися швидким входом через гугл акаунт, але і там і там вас попросять код з пошти. Також користувач може підключити двофакторну автентифікацію, чи він може змінити пароль прямо у налаштуваннях системи. Також що важливо, сервіс Clerk гарантує безпеку ваших особистих даних, там є шифрування даних і взагалі вони на своєму сайті мало не клянуться, що їх система найкраща по інформаційній безпеці.

На мій погляд інтеграція Clerk у систему стала виправданим технічним рішенням, яке поєднало у собі зручність для користувачів, зручність для мене як розробника та безпеку платформи. Крім цього, використання готового сервісу дозволило скоротити час розробки модуля авторизації та зосередити увагу на більш важливих речах, а саме основному функціоналі веб-застосунку. Ну і загалом Clerk підвищив рівень стабільності поточної системи, оскільки більшість критичних аспектів безпеки переклалося на зовнішнє та дуже перевірено

рішення. Інтерфейс вікна Clerk для авторизації та реєстрації у соціальній мережі зображений на рисунку 2.9.

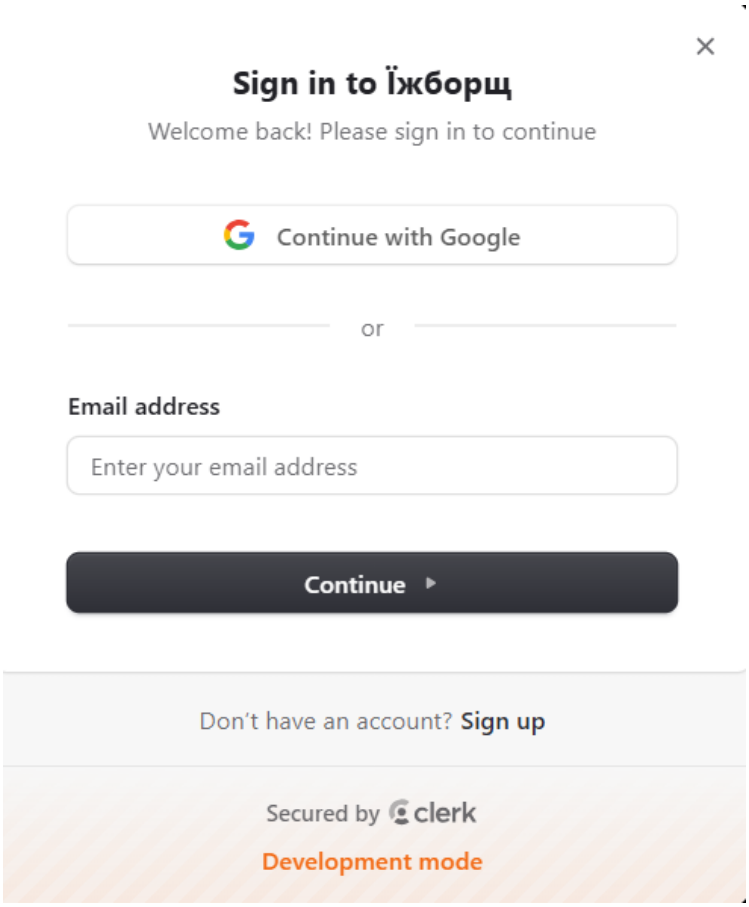

Рисунок 2.9 – Демонстрація роботи сервісу Clerk у системі

Джерело: розроблено автором

Далі хочеться описати карточку публікації, яка є ключовим елементом у системи. Зліва я розташував фотографії, які можна перемикати, або відеоролик, який можна поставити на паузу, відкрити на цілий екран, чи збільшите гучність. Права частина призначена для тексту. Спочатку іде текст рецепту, який підтримує форматування та емодзі, а за ним іде блок коментарів. У цьому блоці любий авторизований юзер може залишите свій коментар, відповісти на вже існуючий, поставити вподобайку на коментар або саму публікацію. Кожен коментар має інформацію про автора, час, коли був опублікований, про кількість вподобань та кількість відповідей. Також я додав кнопку емодзі, на яку ви клікаєте і перед вам сотні різних емодзі, якы відображаються у коментарях. Блок

з коментарями повністю інтерактивний, є можливість переходити у профілі користувачів просто натиснувши на їх аватарку. Інтерфейс вікна публікації зображено на рисунку 2.10.

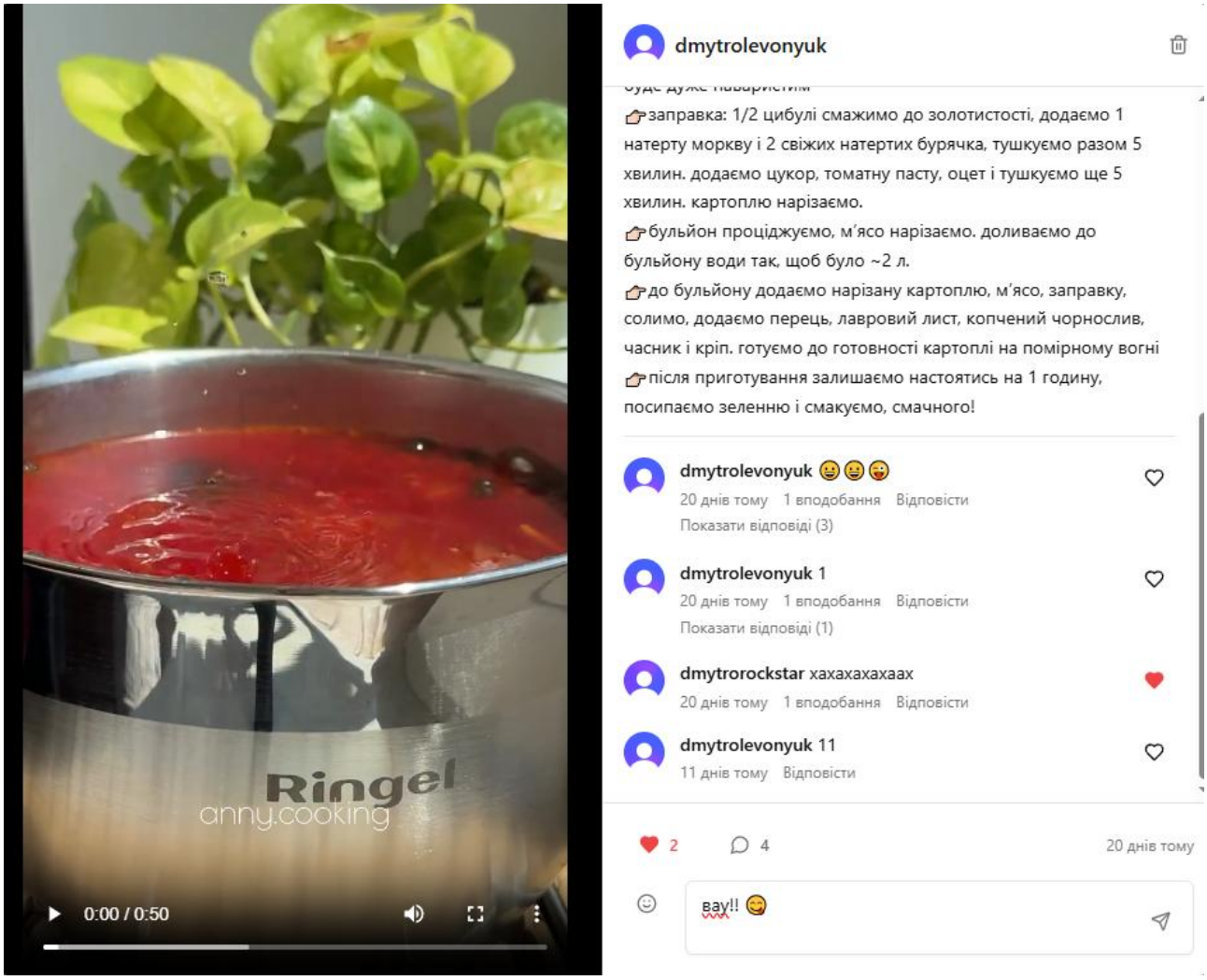

Рисунок 2.10 – Демонстрація роботи PostCard.tsx

Джерело: розроблено автором

Далі я розробив сторінку для профіля користувача, яка представляє собою персональний простір з інформацією про автора та невеличкою персоналізацією. Ви можете зверху побачити велику аватарку, ім'я, нікнейм, кількість публікацій, підписників та підписок. Також було додано кнопки стежити і повідомлення для початку приватного листування. А коли ви заходите на свою сторінку, замість кнопки стежити з'являється кнопка редагувати, за допомогою якої можна додатково про себе щось написати, додати місце проживання та посилання на

інший ресурс. Нижче я розмістив панельку публікацій і збереженого. Ви можете легко переключатися між ними при необхідності. Тут я також використав сітку публікацій PostGrid з головної сторінки, щоб нічого зайвого не вигадувати. Сторінка профіля вийшла дуже простою, але в той же час корисною (рис. 2.11).

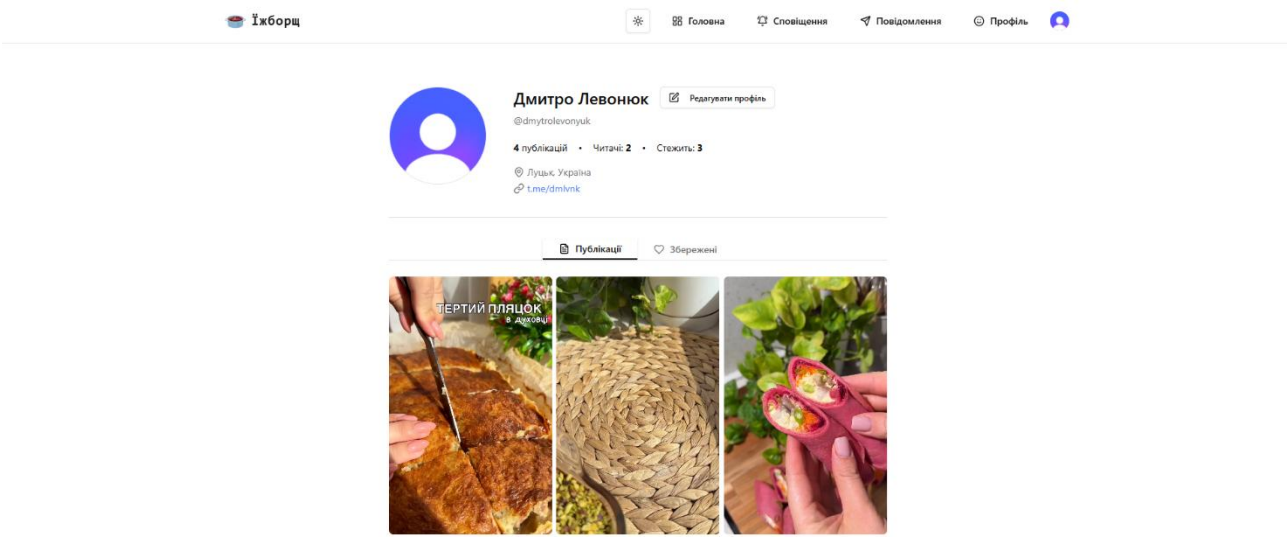

Рисунок 2.11 – Демонстрація роботи ProfilePageClient.tsx

Джерело: розроблено автором

Кожна ваша дія у системі додатково створює сповіщення про цю дію, а всі ці сповіщення треба щоб десь зберігалися, тому я і створив наступну сторінку для сповіщень. Ця сторінка відповідає за інформування користувача про всі важливі події з його акаунтом. У верхній частині я додав лічильник кількості непрочитаних сповіщень, щоб користувач одразу міг зрозуміти, чи треба комусь відповісти, чи хтось хоче з ним почати спілкуватися і таке інше. Основна частина складається зі списку сповіщень, які мають форму прямокутників і ідуть один за одним. Кожне сповіщення як окрема карточка, ви можете бачити аватарку людини, нікнейм, що вона зробила і до якої публікації ця дія має відношення. Наприклад, якщо користувач відповів на ваш коментар, у карточці відображається і ваш коментар, на який відповіли і сама відповідь цього користувача. Також я додав до кожного сповіщення час, коли воно було зроблено, додав іконки під кожну дію, а ще сповіщення інтерактивні, ви можете клікнути на любе і у вас відкриється повний допис. Для сповіщень, пов’язаних з

дописами, щоб їх одразу можна було впізнати я додав мініатюрки, до якого саме рецепта належить дія. Інтерфейс сторінки сповіщень зображено на рисунку 2.12.

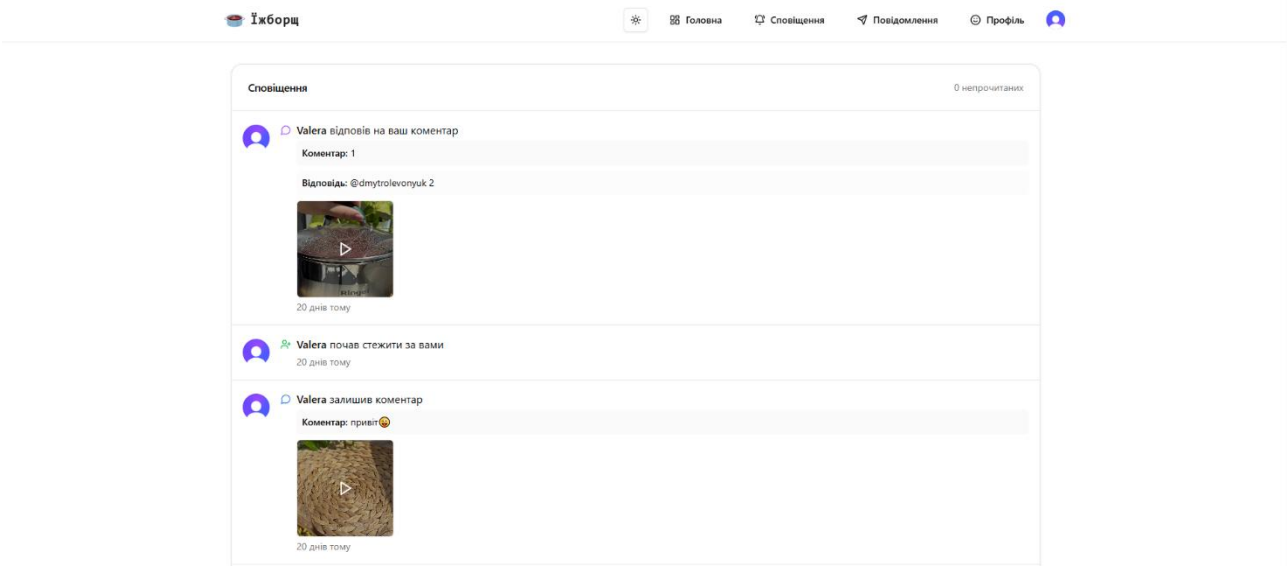

Рисунок 2.12 – Демонстрація роботи NotificationsPage.tsx

Джерело: розроблено автором

У моєму web-додатку користувачі зможуть спілкуватися як у повноцінному месенджері, відправляти емодзі, скидати один одному фото чи відео. Все що їм потрібно, це кинути запит на спілкування улюдини, з якою вони хочуть почати діалог. Я зробив окрему сторінку для повідомлень, де вони всі діляться на два типи. Перший для вже існуючих діалогів, а другий для запитів на спілкування. Це було зроблено, щоб зменшити кількість спаму зі сторони рандомних користувачів з їх повідомленнями. На сторінці є 2 кнопки для перемикання типів повідомлень, а також списком ідуть самі повідомлення. У кожного блоку повідомлень зліва є аватарка, нікнейм та останнє написане поточним користувачем або його співрозмовником. Також я розташував увесь інтерфейс модуля повідомлень по центру сторінки, щоб користувачам було зручно користуватись месенджером та їх увага концентрувалась на діалогах. Додатково було приділено увагу тому, щоб структура списку повідомлень залишалася зрозумілою та передбачуваною для користувача навіть при збільшенні кількості діалогів. Завдяки цьому інтерфейс залишається

інформативним, а навігація між різними типами повідомлень є швидкою та інтуїтивною. Інтерфейс сторінки повідомлень зображено на рисунку 2.13.

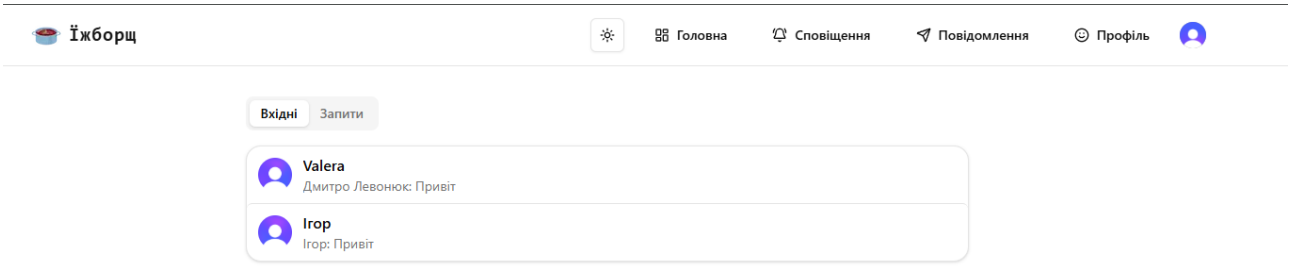

Рисунок 2.13 – Демонстрація роботи MessagesPage.tsx

Джерело: розроблено автором

Далі я хочу вам розказати трохи більше про зовнішній вигляд повідомлення запиту (рис. 2.14). Ви бачите перед собою аватарку, нікнейм і перше повідомлення, яке вам написали. Зараз у вас не має можливості відповісти цій людині, для цього спочатку треба прийняти запит на листування, клікнувши на кнопку внизу екрану, або проігнорувати цей запит.

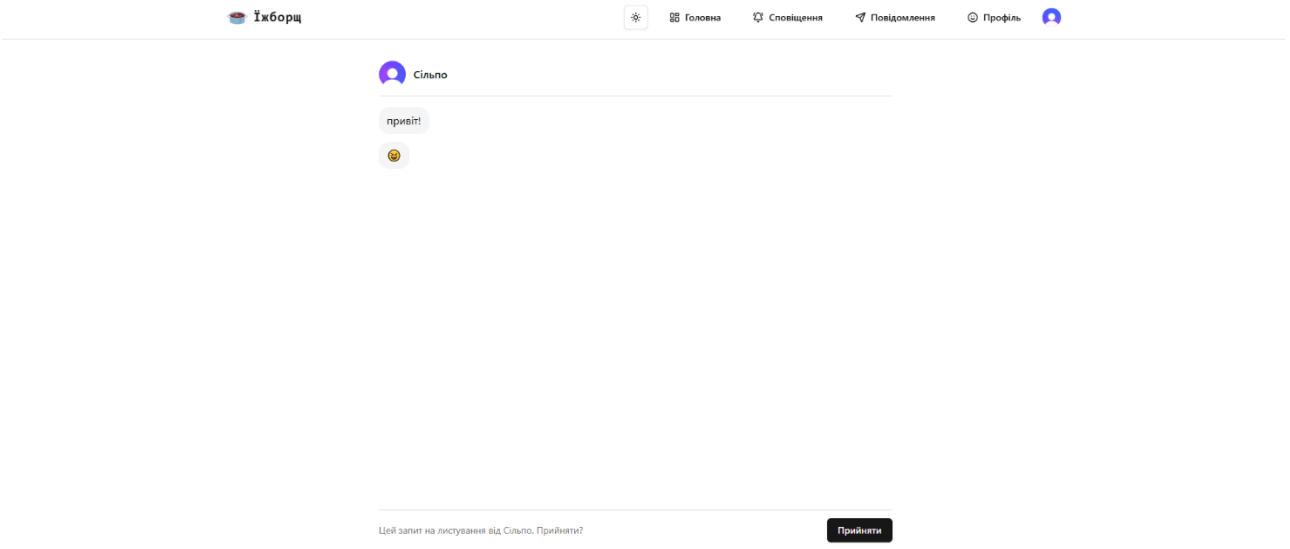

Рисунок 2.14 – Демонстрація роботи ThreadPage.tsx

Джерело: розроблено автором

На сторінці діалогу (рис. 2.15) користувач може побачити повну історію переписки зі своїм другом. Я зробив так, що ваші власні повідомлення справа, а повідомлення від вашого співрозмовника знаходяться зліва. В самому верху я додав аватарку і нікнейм людини, з якою ви спілкуєтеся, щоб ви розуміли з ким ведете листування. Знизу у нас поле для вводу тексту, кнопка з іконкою скріпки для вкладень (фото, відео та pdf файли), а також кнопка зі смайликом, яка відкриває емодзі меню. Якщо ви хочете видалити, або редагувати повідомлення, то вам потрібно всього лише дабл клікнути по необхідному повідомленню і відкриється вікно з редагування, кнопкою видалити і кнопкою скасувати дію. Всі елементи діалогу мають мінімалістичний та приємний дизайн і були розміщені по центру сторінки для комфортного та легкого спілкування між користувачами.

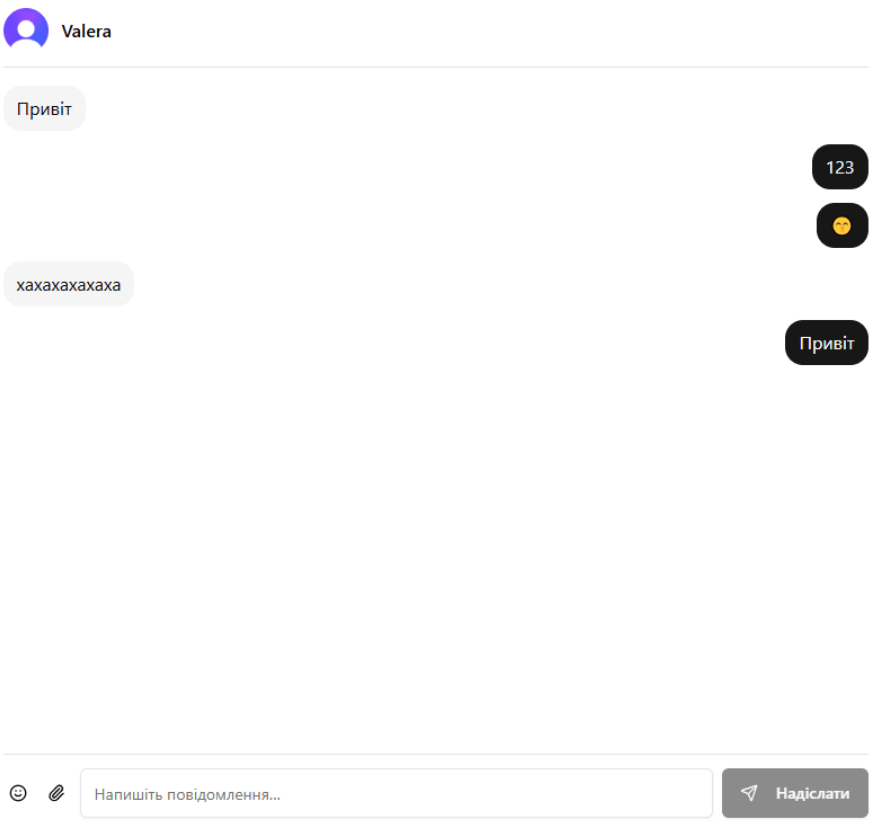

Рисунок 2.15 – Демонстрація роботи ThreadPage.tsx

Джерело: розроблено автором

РОЗДІЛ 3
ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РЕЗУЛЬТАТИВНОСТІ ПРЕДМЕТУ ДОСЛІДЖЕННЯ

3.1 Методика проведення дослідження

У цьому підрозділі буде описана методика експериментального дослідження продуктивності розробленої соціальної мережі з рецептами. У яких умовах треба проводити дослідження, які інструменти використовувати, а також способи вимірювання продуктивності.

Для дослідження було обрано Google Chrome, бо він має вбудований набір інструментів (DevTools), за допомогою якого ви можете протестувати і виміряти широкий спектр показників продуктивність без використання сторонніх програм.

Продуктивність web-додатку оцінювалася за такими показниками:

– аналіз швидкості завантаження сторінки (DOMContentLoaded, Load);

– оцінка затримки відповіді сервера (TTFB);

– аналіз показників Core Web Vitals (LCP, INP та CLS);

– швидкість завантаження медіа;

– аналіз швидкості SPA-навігації;

– оцінка навантаження на центральний процесор.

У DevTools на вкладці Network є дві метрики – DOMContentLoaded та Load. Перша відповідає за час, коли весь HTML-документ повністю завантажено, але це час без урахування завантаження зовнішніх ресурсів, таких як зображення. А друга метрика якраз відповідає за повне відображення всіх елементів сторінки. Ці показники дозволять нам дізнатися, наскільки швидко сторінка стає доступною для користувача.

Для оцінки продуктивності серверної частини використовується метрика TTFB. Time to First Byte – це метрика, яка описує від моменту відправлення браузером запиту сторінки до моменту, коли браузер отримав перший байт інформації з сервера. Вона нам потрібна, щоб оцінити затримку на сервері.

Для комплексної оцінки досвіду користувача дивляться на метрики Core Web Vitals. Core Web Vitals – це набір метрик, які проводять оцінку швидкості завантаження, інтерактивності так стабільності верстки. Є три ключових показника: LCP, FID та CLS. Largest Contentful Paint – це скільки часу потрібно на рендер найбільшого елемента сторінки. First Input Delay – це через скільки секунд користувач зможе взаємодіяти з вашою сторінкою. І Cumulative Layout Shift – це наскільки елементи на сторінці зміщуються під час її прогрузки.

Оскільки у мене соціальна мережа з фото та відео рецептами мені потрібно проаналізувати швидкість завантаження контенту, бо це напряму пливає на користувацький досвід. ля цього у вкладці Network використовується фільтр Media, який дозволяє відобразити лише ті запити, що належать фото та відео файлам.

Для аналізу переходів між сторінками використовуються асинхронні запити, бо наш веб-додаток працює за принципом SPA (Single Page Application). Вимірювання проводиться у вкладці Network з фільтром Fetch/XHR у середовищі Chrome Devtools. Це дозволить нам проаналізувати тривалість взаємодії клієнта та сервера при переході між сторінками.

І фінальний тест, який нам потрібен, пов’язаний з навантаженням веб-додатку на центральний процесор під час завантаження сторінок. Тут нам потрібен інструмент Performance у середовищі Chrome Devtools, а саме метрики Scripting та Rendering у блоці Summary. Вони відображають час, який потребується для виконання скриптів і відповідно рендерингу елементів. Ці метрики нам потрібні заради об’єктивної оцінки ефективності використання обчислювальних ресурсів клієнтською частиною веб-додатку.

3.2 Обробка та аналіз отриманих результатів

Перед тим як почати роботи вимірювання я залив проект на платформу для хостингу Render. Наступним кроком я відкрив інкогніто вкладку у браузері Google Chrome та авторизувався у своїй соціальній мережі з рецептами. Далі

натиснув F12, щоб відкрити набір інструментів DevTools та поставив галочку напроти Disable cache. Якщо її не поставити, то половина вимірювань будуть хибними. А також всі оновлення сторінок проводимо лише через Ctrl+Shift+R, це єдине повне перезавантаження сторінки, при якому нічого не залишається. А інтерфейс платформи Render із розгорнутим веб-сервісом ви можете побачити на рисунку 3.1.

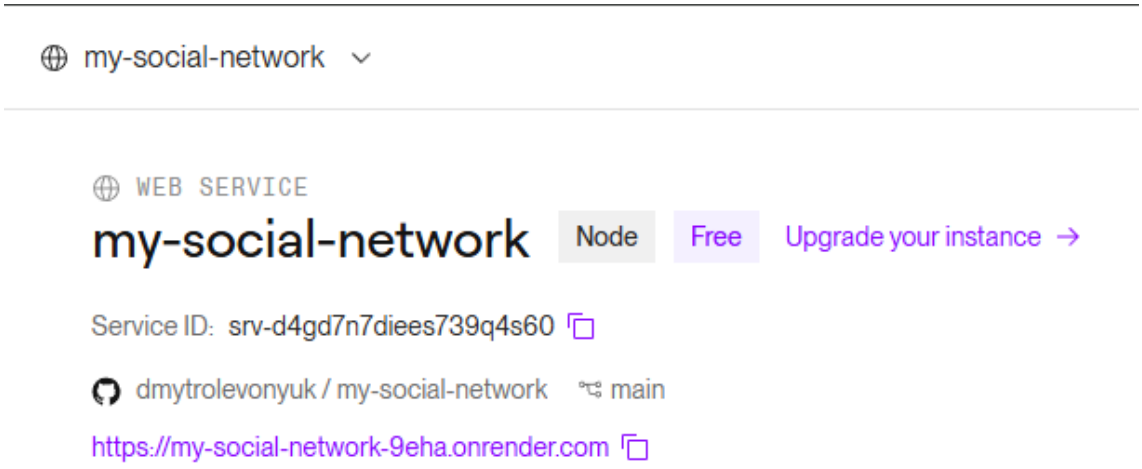

Рисунок 3.1 – Демонстрація роботи Render [17]

Я проводив вимірювання швидкості завантаження на головній сторінці свого веб-додатку. Додатково вимірювання проводилося не лише у стандартному режимі, але також у режимах 3G та Slow 4G, благо інструментарій Chrome Developer Tools дозволяє. А ще тут окрім готових пресетів навіть можна створити свій кастомний. І Для більш точних середніх цифр на кожному режимі я зробив по 10 перезавантажень сторінки. Середнє значення DOMContentLoaded та Load у тепличних умовах – 0.65 секунди (гарний результат) для завантаження HTML-документу і 3.18 секунди для повного завантаження всіх елементів. Середнє значення DOMContentLoaded та Load у режимі 3G – 6.95 секунди для HTML, а повне завантаження зайняло майже 40 секунд (39.78). На режимі Slow 4G ситуація трохи краща – 2 секунди це зайняло для HTML-документу та 11 секунд для повної загрузки. Ще там був режим Fast 4G, але я не бачив великого сенсу в ньому та вирішив не проводити ще одне вимірювання.

Вимірювання показника TTFB проводилось на головній сторінці веб-додатку. Так само у середовище Chrome DevTools переходимо у вкладку Network, натискаємо на файл нашої сторінки і у підрозділі Timing бачимо потрібний нам параметр (Waiting for server response). Кожне вимірювання проводилось 10 разів, щоб все ж таки результати були більш точні. Перед кожним вимірюванням я робив hard refresh. У нас вийшло, що середній час відгуку сервера займає приблизно 0.55 секунди, коли рекомендоване значення 0.2, а середній повний час завантаження сторінки займає 0.71 секунди. Зараз оптимізація знаходиться на задовільному рівні, але у майбутньому можна буде її зменшити шляхом оптимізації серверних запитів. Демонстрацію вимірювання метрики Time to First Byte зображено на рисунку 3.2.

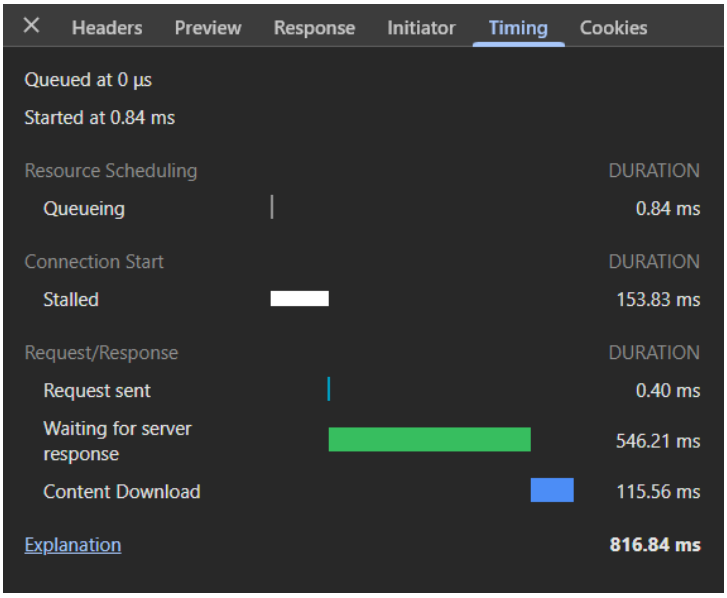

Рисунок 3.2 – Демонстрація вимірювання TTFB

Джерело: розроблено автором

Я проводив вимірювання LCP у двох режимах. Перший, це коли я геть нічого не натискав, щоб оцінити можливу швидкість, а другий, коли я після завантаження багато взаємодіяв з інтерфейсом. Кожен тест було проведено по 10 разів, з hard refresh і на головній сторінці. Ще треба додати, що у першому режимі FID та CLS не вимірюються, бо ми нічого не робимо. І що ж в мене вийшло? Середній LCP, коли я нічого не натискав, ніяк не взаємодіяв з інтерфейсом після

завантаження займає – 0.74 секунди. З десяти разів найшвидший раз був аж 0.55 секунди. А середній LCP у режимі активної взаємодії з інтерфейсом складає трохи менше 0.65 секунди, коли граничне значення 2.5 секунди. Це свідчить про швидке завантаження основного наповнення веб-сторінки. Також середні INP та CLS – 66 мілісекунд (порог 200) і 0.112 замість потрібних 0.1. Показник CLS трохи перевищує норму, а це свідчить про незначне порушення візуальної стабільності. А показник INP свідчить про високу швидкість реакції інтерфейсу на дії свого користувача. Демонстрацію вимірювання метрики LCP зображено на рисунку 3.3.

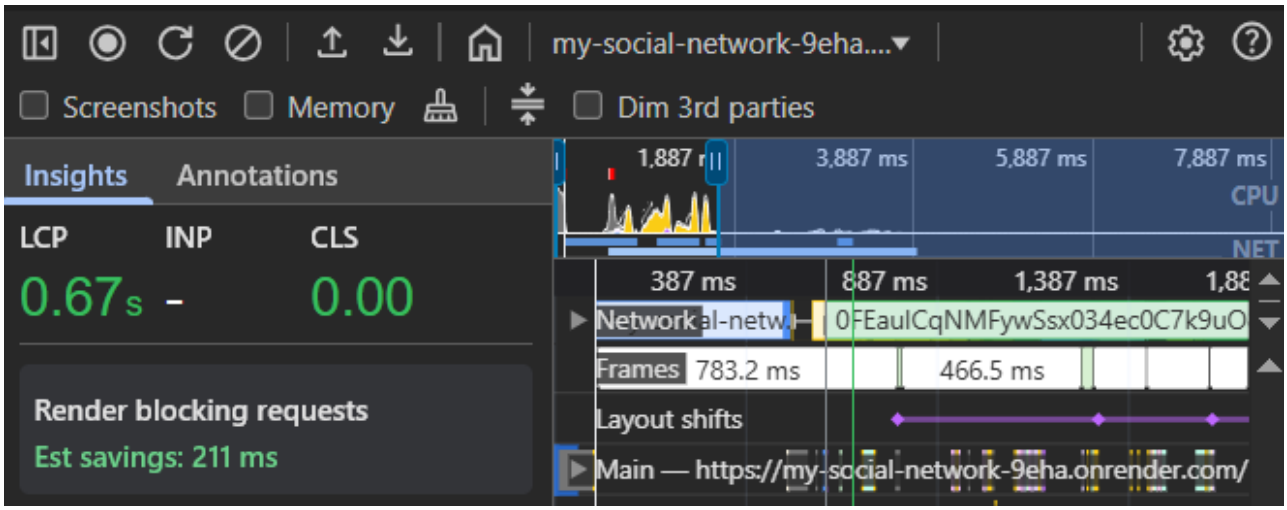

Рисунок 3.3 – Демонстрація вимірювання LCP

Джерело: розроблено автором

Далі було проаналізовано швидкість завантаження медіа контенту. По тому ж сценарії заходимо Chrome DevTools, вкладка Network, але в цей раз нам треба клікнути на фільтр Media, щоб тільки завантаження медіа файлів рахувалося. Я провів 10 перезавантажень і якось так сталося, що одне із перезавантажень має аномально великий результат. Чомусь на четвертій спробі, на веб-сайті на якому знаходилося 4 відео файли, у вимірюванні взяло участь цілих 8. І тому я порахував 2 середніх значення, з цим аномальним і без нього. Середній час завантаження медіа файлів з аномальним – 5.08 секунди. А без нього він складає приблизно 3.55 секунди, що є доволі прийнятним результатом.

Наступне вимірювання пов’язане з часом на перехід з однієї сторінки веб-сайту на іншу. Тут нам не підходять метрики DOMContentLoaded та Load, бо при переході між сторінками повноцінно нічого не перезавантажується. Ми проаналізували асинхронні запити у вкладці Network з фільтром Fetch/XHR у середовищі Chrome Devtools (рис. 3.4). При вимірюванні треба було переходити з головної сторінки на сторінку сповіщень, з поверненням на головну сторінку перед кожним новим вимірюванням. Середній Waiting for server response у мене вийшов 145 мілісекунд, а повний час переходу десь 152 мілісекунди, що є гарним результатом та підтверджує ефективність реалізованої клієнтської структури.

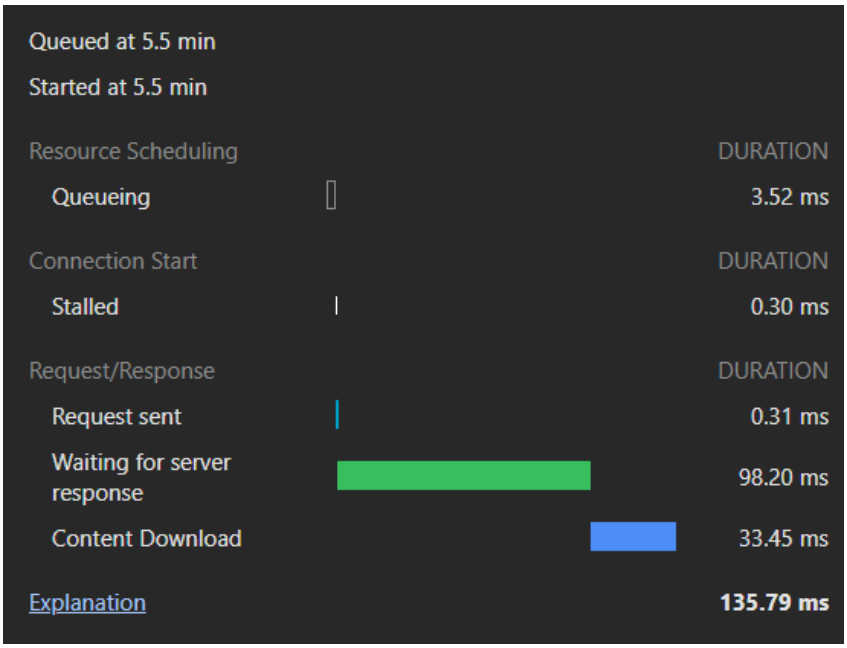

Рисунок 3.4 – Вимірювання часу відповіді сервера при SPA-навігації

Джерело: розроблено автором

Фінальним етапом є аналіз навантаження веб-додатку на центральний процесор під час завантаження сторінок. Кожну спробу експерименту виконувалося повне перезавантаження сторінки, а середнє значення навантаження CPU, яке рахується як Scripting + Rendering займає приблизно 318 мілісекунд, що свідчить нам про достатню ефективність використання обчислювальних ресурсів браузера без надмірних навантажень на систему.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи магістра було проаналізовано предметну область соціальних мереж, а також розглянуто наявні аналоги, орієнтовані на поширення фото та відео контенту. Особлива увага була приділена їхнім можливостям, характерним обмеженням, архітектурним особливостям та перспективам подальшого розвитку. Проведений аналіз дозволив виявити потребу у створенні спеціалізованої соціальної мережі з тематичним спрямуванням, зокрема сервісу для обміну кулінарними рецептами та взаємодії користувачів у форматі контент платформи. Результати досліджень підтвердили актуальність обраної теми й обґрунтували доцільність розробки веб-додатку такого типу.

На основі вивчення предметної області були сформовані функціональні та нефункціональні вимоги до системи, визначені ролі користувачів, описані ключові сценарії їхньої взаємодії та побудована загальна архітектура додатку. Це дало змогу створити чітке уявлення про логіку роботи майбутнього сервісу, структуру його основних модулів та способи організації обробки мультимедійного контенту. На основі сформованих вимог було розроблено структурований підхід до проектування та реалізації функціональних компонентів веб-додатку.

Важливою частиною роботи стало обґрунтування вибору технологічного стеку. У якості основних інструментів були обрані Next.js, React, TypeScript, PostgreSQL та низка супутніх технологій, сервісів і бібліотек, які забезпечують оптимальний баланс між продуктивністю, масштабованістю та зручністю розробки. Проведене теоретичне моделювання структури системи дозволило визначити принципи її побудови, гарантувати стабільність та передбачити потенційні напрями розширення функціональності у майбутньому.

У рамках виконання роботи було розроблено інформаційну модель системи та логічну структуру бази даних, визначено ключові сутності, типи зв’язків між ними та поведінку основних об’єктів. Створені алгоритми взаємодії

між компонентами веб-сервісу забезпечили узгодженість даних, передбачуваність поведінки системи та можливість ефективної роботи з користувацьким контентом. Реалізовано механізми авторизації, управління публікаціями, взаємодії користувачів через вподобайки, коментарі, підписки та приватні повідомлення, що відповідає встановленим вимогам.

Завершальним етапом стало проведення експериментального дослідження результативності розробленого веб-додатку. Була протестована коректність роботи основних модулів системи, поведінку сервісу під навантаженням, стабільність обробки запитів, швидкість генерації сторінок та взаємодії зі стрічкою публікацій. Отримані результати підтвердили відповідність реалізованої системи заявленим функціональним та нефункціональним вимогам, а також її готовність до практичного використання.

У результаті виконання кваліфікаційної роботи було створено повноцінний веб-додаток соціальної мережі для обміну кулінарними рецептами. Він поєднує в собі можливості сучасної соціальної платформи, інструментів для публікації фото та відео рецептів та засобів інтерактивної комунікації між користувачами. Розроблений продукт має практичну цінність, демонструє ефективність обраних технологічних рішень та може слугувати фундаментом для подальшого вдосконалення, масштабування та впровадження нових функціональних можливостей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Instagram. URL: https://www.instagram.com/ (дата звернення: 05.09.2025).
2. Pinterest. URL: https://www.pinterest.com/ (дата звернення: 06.09.2025).
3. FoodGawker. URL: https://foodgawker.com/ (дата звернення: 06.09.2025).
4. PHP. URL: https://www.php.net/ (дата звернення: 19.09.2025).
5. Python. URL: https://www.python.org/ (дата звернення: 22.09.2025).
6. JavaScript. URL: https://javascript.info/ (дата звернення: 28.09.2025).
7. Next.js. URL: https://nextjs.org/ (дата звернення: 05.10.2025).
8. React. URL: https://react.dev/ (дата звернення: 07.10.2025).
9. TypeScript. URL: https://www.typescriptlang.org/ (дата звернення: 15.10.2025).
10. PostgreSQL. URL: https://www.postgresql.org/ (дата звернення: 23.10.2025).
11. Neon. URL: https://neon.com/ (дата звернення: 26.10.2025).
12. Prisma. URL: https://www.prisma.io/ (дата звернення: 27.10.2025).
13. Clerk. URL: https://clerk.com/ (дата звернення: 30.10.2025).
14. UploadThing. URL: https://uploadthing.com/ (дата звернення: 06.11.2025).
15. Cloudinary. URL: https://cloudinary.com/ (дата звернення: 10.11.2025).
16. Visual Studio Code. URL: https://code.visualstudio.com/ (дата звернення: 05.10.2025).
17. Render. URL: https://render.com/ (дата звернення: 21.11.2025).

ДОДАТКИ

Додаток А

Сторінка особистих повідомлень MessagesPage.tsx

```
     export default async function MessagesPage() {
const [inbox, requests] = await Promise.all([getInbox(), getRequests()]);
const Item = ({ t }: { t: any }) => {
const other = t.participants[0]?.user;
 const last = t.messages[0];
 return (
<Link href={`/messages/${t.id}`} className="flex items-center gap-3 px-3
py-2 hover:bg-accent rounded-md">
      <Avatar className="w-9 h-9">
      <AvatarImage src={other?.image ?? "/avatar.png"} />
</Avatar>
           <div className="min-w-0">
           <div    className="font-medium    truncate">{other?.name    ??
other?.username}</div>
                <div className="text-sm text-muted-foreground truncate">
{last ? `${last.sender.name ?? last.sender.username}: ${last.content}` :
"Почніть переписку"}</div>
</div>
</Link>);
};
 return (<div className="max-w-3xl mx-auto">
     <Tabs defaultValue="inbox">
     <TabsList>
     <TabsTrigger value="inbox">Вхідні</TabsTrigger>
     <TabsTrigger value="requests">Запити</TabsTrigger>
     </TabsList>
<TabsContent value="inbox" className="mt-4"><Card className="divide-y">
{inbox.length ? inbox.map((t: any) => <Item key={t.id} t={t} />) :
<div className="p-6 text-muted-foreground">Поки порожньо</div>}
</Card>
 </TabsContent>
<TabsContent value="requests" className="mt-4">
      <Card className="divide-y">
 {requests.length ? requests.map((t: any) => <Item key={t.id} t={t} />) :
<div className="p-6 text-muted-foreground">Запитів немає</div>}
</Card>
</TabsContent>
</Tabs>
</div>
);
}
```

Додаток Б

Компонент для завантаження відео та фото матеріалів ImageUpload.tsx

```
interface ImageUploadProps { onChange: (files: UploadedFile[]) => void;
     value: UploadedFile[]; endpoint: «postMedia»; maxFiles?: number; }
     const UploadDropzone = generateUploadDropzone<OurFileRouter>();
     function ImageUpload({ endpoint, onChange, value, maxFiles = 10, }:
ImageUploadProps)
     {
        const handleRemove = (url: string) => { const updated =
value.filter((item) => item.url !== url);  onChange(updated); };
       return ( <div className=»flex flex-wrap gap-4»>
{value.map((file) => ( <div key={file.url} className=»relative w-40 h-40»>
             {file.type === «video» ? ( <video src={file.url} controls
className=»rounded-md w-full h-full object-cover»/>): (<img src={file.url}
              alt=»Uploaded»
              className=»rounded-md w-full h-full object-cover»/>)}
     <button onClick={() => handleRemove(file.url)}
className=»absolute top-0 right-0 p-1 bg-red-500 rounded-full shadow-sm»
 type=»button» >
        <Xicon className=»h-4 w-4 text-white» />
     </button>
     </div>
     ))
     }
     {value.length < maxFiles && (
               <div className=»w-50 h-40 flex items-center justify-center
overflow-hidden mt-[-20px] ml-[180px]»>
              <div className=»w-full h-full flex items-center justify-
center overflow-hidden»>
           <UploadDropzone endpoint={endpoint}
           onClientUploadComplete={(res) => { if (!res) return;
        const newFiles: UploadedFile[] = res.map((file) => (
     {
     url: file.serverData.fileUrl,
     type: file.serverData.fileType.startsWith(«video»)
     ? «video» : «image», })); onChange([…value, …newFiles]); }}
     onUploadError={(error: Error) =>
     console.error(«Upload error:», error)}/>
     </div>
     </div>
     )
     }
     </div>
     );
      }
     export default ImageUpload;
```

Додаток В

Компонент вікна діалогу ThreadIdPage.tsx

```
export default async function ThreadPage({
  params,
}: { params: { threadId: string };
}) {  const meId = await getDbUserId();
  if (!meId) notFound();
  const thread = await getThread(params.threadId);
  if (!thread) notFound();
  const myParticipant = thread.participants.find((p: any) => p.userId
=== meId) ?? null;
   let other = thread.participants.find((p: any) => p.userId !==
meId)?.user ??
    thread.messages.find((m: any) => m.senderId !== meId)?.sender;
  if (!myParticipant || !other) notFound();
  const isPending = myParticipant.status === "PENDING";
  async function accept() {"use server";
 await acceptRequest(params.threadId); }
  return (
<div className="max-w-3xl mx-auto flex flex-col h-[calc(100vh-200px)]">
      <div className="flex items-center gap-3 border-b pb-3">
        <Avatar className="w-10 h-10">
         <AvatarImage src={other.image ?? "/avatar.png"} /> </Avatar>
        <div className="font-medium">
         {other.name ?? other.username ?? "Користувач"} </div></div>
      <div className="flex-1 overflow-y-auto py-4 space-y-3">
         {thread.messages.length === 0 && (<div className="text-sm
text-muted-foreground">
          Тут поки що немає повідомлень. Напишіть першим.</div> )}
        {thread.messages.map((m: any) => ( <MessageBubble
            key={m.id}
            id={m.id}
            mine={m.senderId === meId}
            content={m.content}
attachments={Array.isArray(m.attachments) ? m.attachments : []} /> ))}
</div>
    {isPending ? (
         <div className="border-t pt-3 flex items-center justify-
between gap-3">
          <div className="text-sm text-muted-foreground">
                  Цей запит на листування від {other.name ??
other.username}. Прийняти? </div>
       <form action={accept}><Button type="submit">Прийняти </Button>
</form> </div> ) : (<MessageComposer threadId={params.threadId} />
      )}
  </div>);
  }
```

Додаток Г

Компонент для побудові сітки публікацій PostGrid.tsx

```
     export interface PostGridProps {
 posts: { id: string; images: UploadedFile[];
 _count: { likes: number };
        comments: any[]; category?: string;
        createdAt?: string;}[];
       dbUserId: string | null;
        shuffle?: boolean; }
     export default function PostGrid({ posts, dbUserId }: PostGridProps)
{
       const [selectedPost, setSelectedPost] = useState<any | null>(null);
       return ( <>
         <div  className="grid  grid-cols-1  sm:grid-cols-2  md:grid-cols-3
gap-[4px] sm:gap-[6px]">
             {posts.map((post) => (
      <div   key={post.id}   className="relative   cursor-pointer   group
overflow-hidden rounded-md"
                 onClick={() => setSelectedPost(post)}>
                 {post.images?.[0] ? (
                   <div className="relative w-full h-[340px] sm:h-[360px]
md:h-[380px] overflow-hidden flex items-center justify-center">
                          {post.images[0].type  ===  "video"  ?  (<video
src={post.images[0].url}
                        className="w-full h-full object-cover transition-
opacity duration-300 group-hover:opacity-100 opacity-100"
                         muted
                         playsInline
                         preload="metadata"
                         onMouseEnter={(e) => e.currentTarget.play()}
                         onMouseLeave={(e) => { e.currentTarget.pause();
                          e.currentTarget.currentTime = 0;}}/>) : ( <img
src={post.images[0].url}
                         alt="Post preview"
                 className="w-full h-full object-cover"/> )} </div>
     ) : (
                    <div className="w-full h-[380px] bg-neutral-800 flex
items-center justify-center text-gray-400 text-sm"> Без фото
     </div>)}
                     <div  className="absolute  inset-0  flex  items-center
justify-center  bg-black/20  opacity-0  group-hover:opacity-100  transition
text-white text-sm font-medium">
                    {post._count.likes}    {post.comments.length}
     </div>
     </div>))
     }
     </div>
 <Dialog open={!!selectedPost} onOpenChange={() => setSelectedPost(null)}>
```

```
      <DialogContent className="max-w-4xl p-0 bg-transparent border-none
shadow-none">
      <DialogHeader>
      <DialogTitle>
      </DialogTitle>
                  <DialogDescription>
      </DialogDescription>
      </DialogHeader>
       {selectedPost && (
<PostCard post={selectedPost} dbUserId={dbUserId} /> )}
</DialogContent>
      </Dialog> </>
       );
      }
```

Додаток Д

Стаття для наукового збірника «Студентський науковий вісник»

Луцький національний технічний університет, «Студентський науковий вісник», Випуск 54 – 2025

РОЗДІЛ 1. ТЕХНІЧНІ НАУКИ

УДК 004.4:004.738.5
Левонюк Д.С., ст. гр. КНм-21
Науковий керівник: к.т.н., доц. Кошелюк В.А.

РОЗРОБКА ТА ДОСЛІДЖЕННЯ WEB-ДОДАТКУ З ВИКОРИСТАННЯМ NEXT.JS

У статті розглядається програмна реалізація та дослідження web-додатку соціальної мережі з фокусом на обміні кулінарним контентом. Були проаналізовані сучасні тенденції розвитку соцмереж, обгрунтовано вибір Next.js як основи для системи, описано архітектуру додатку, його функціонал та практичне застосування. Показано, що спеціалізовані соцмережі дозволяють підвищити залученість користувачів та створити цілеспрямовану спільноту.
Ключові слова: Next.js, web-додаток, соціальна мережа.

Levoniuk D.

DEVELOPMENT AND RESEARCH OF A WEB APPLICATION USING NEXT.JS

The article describes the development and research of a web application designed as a social network for sharing culinary content. Modern trends in social networks are analyzed, the choice of Next.js technology is justified, and the architecture and functionality of the system are presented. The study shows that specialized thematic social networks help increase user engagement and create focused online communities.
Keywords: Next.js, web application, social network.

Постановка проблеми у загальному вигляді та її зв'язок з важливими науковими і практичними завданнями. Зайти зранку у кілька соціальних мереж це вже щоденна людська рутина, без якої важко уявити свій день. Всі використовують їх для комунікації, творчі люди для оприлюднення своєї творчості, а ще соцмережі це величезна інформаційний майданчик. Соцмережі існують, щоб люди зі спільними інтересами могли знайти один одного, почати дружити, створювали власні спільноти таких же однодумців, які зацікавлених у певній темі, де вони би обмінювались думками та враженнями між собою. З кожним роком фото та відео контент стає тільки популярнішим. Одні люди виставляють контент, а інші його переглядають, залишаючи свій фідбек у вигляді коментарів, вподобань та репостів. І серед всього, що можна зафіксувати на камеру саме їжа займає особливе місце у голові людей. Чомусь всім цікаво дізнатися хто і що їсть, існує тонна відео та фото рецептів на будь-який смак і колір. Одні люди діляться своїми стравами, а інші надихаються ідеями, запам'ятовують, щоб потім вигадати щось своє.

Всі соцмережі налаштовані так, що спочатку вони дізнаються вас краще, будують портрет користувача, підлаштовуючись під ваші інтереси. Кілька підписок однієї тематики і соцмережа ставить біля вас умовну галочку – йому подобається спорт, він любить автомобілі, він любить готувати. І вже з наступного заходу на веб-сайт система рекомендацій буде пропонувати відповідні вашим інтересам канали. Але зараз соціальні мережі настільки живуть

комерцією, що у їх на меті не те, щоб ви знайшли контент за яким прийшли, а щоб ви провели максимум свого часу у їхньому застосунку.

І тому мені здається, що створення спеціальних тематичних веб-сервісів дозволить не одразу позбавитися цієї проблеми, але стане правильним першим кроком, щоб якось це зменшити. Саме зараз, коли класичні соціальні мережі вже перевантажені контентом, нам і потрібні тематичні платформи, у яких будуть невеликі, але згуртовані та дружні спільноти однодумців, які будуть обмінюватись інформацією між собою. Хтось сильно цікавиться літературою – ось йому книжкова соціальна мережа. Інший активно займається спортом і хоче оточити себе такими ж активними та здоровими людьми – ось спортивна платформа. Через це створення соціальної мережі для обміну рецептами вписується у сучасну практику розробки веб-сервісів.

Аналіз останніх досліджень, у яких започатковано вирішення проблеми. Є безліч соцмереж, які реалізують ідею обміну фото та відео, але ні одна з них не об'єднала у собі все необхідне в межах одного застосунку за певною тематикою. Найвідоміша соціальна мережа для обміну фото це Instagram. Вона дуже проста у використанні, має зручний інтерфейс, систему рекомендацій, особистих повідомлень і коментарів, але все це було побудовано навколо обміну фотографіями. Користувачу потрібно один раз зареєструватися і він вже може ділитися власними фото, отримувати з них фідбек, відповідати коментаторам та заводити з ними особисті чати.

У Instagram стрічка підлаштовується під попередні вподобання користувача. Система показує контент, який відповідає тому, що користувач вподобав чи на що вже підписався. Наприклад, якщо підписатися на кілька акторів, спортсменів чи кулінарних сторінок, коли ви в наступний раз відкриєте веб-сайт, то ваша стрічка вся буде в акторах, спортсменах, їжі і не тільки з ваших підписок. З кожним оновленням сторінки система буде рекомендувати вам нові обличчя, сподіваючись, що ви на когось підпишетесь. Але незважаючи на всі плюси, Instagram все ж таки універсальна платформа, вона не має фокусу на готуванні і навіть якщо ви підпишетесь на 100 сторінок з рецептами, ваша стрічка публікацій не буде вся у рецептах, там буде реклама, там будуть відволікаючи публікації і так далі.

Наступним веб-сайтом для аналізу став Pinterest. Це така платформа, де користувачі зберігають, шукають та діляться фотографіями. Вона представляє собою віртуальну дошку, де користувач, вписуючи у пошуку, що його цікавить, у наступні заходи буде бачити велику кількість фотографій на теми, що він шукав, як на дошці з кримінальних серіалів. У тебе тут є свій профіль, де ти можеш зберігати вподобані публікації сортуючи їх по колекціям, але на веб-сайті порівнюючи з Instagram дуже слабка соціальна складова. Тут не прийнято ставити вподобайки, тут не пишуть коментарі, так є особисті повідомлення, але загалом це все ж таки платформа про власне колекціонування, а не про спілкування та обмін інформацією. А також сервіс дуже перевантажений, охоплює мільйон тем, серед яких рецепти страв блякнуть.

Якщо дивитися на більш кулінарні рішення, то це веб-сайти з рецептами. Там ви можете переглядати рецепти, ділитися власними, зберігати ті що сподобалися, на деяких можна залишати коментарі та оцінювати страви. Але знову ж таки, не має повноцінної соціальної складової. Ви не можете підписатись на свого друга і одразу в цьому ж місті з ним поспілкуватись через особисті повідомлення. Зазвичай веб-сайти з рецептами мають застарілий дизайн, обмежений функціонал, не мають особистою системи рекомендацій. Там не завжди інтуїтивно зрозумілий інтерфейс, візуальна складова, яка є дуже важливою для теперішніх користувачів там відходить на другій план. Ці веб-сайти користувачами і не сприймаються як повноцінні платформи, це більше як місце знань та досвіду.

Після порівняння усіх цих веб-сайтів можна побачити, що жодна з них повністю не відповідає поставленим вимогам, а саме соціальна складова (особисті повідомлення, підписки, коментарі), простота та зручність у використанні, візуальна привабливість та кулінарна тематика. Тому і виникла ідея створити платформу з рецептами, яка зможе поєднати функціональність та зручність сучасних соцмереж з кулінарною тематикою, а саме фото та відео рецепти.

Цілі статті. Метою цієї статі є розробка, реалізація та дослідження web-додатку соціальної мережі для обміну рецептами з використання Next.js.

Для досягнення мети статті поставлено такі завдання:

Аналіз предметної області та вже існуючих рішень: треба дослідити соціальні мережі як концепцію, оцінити у якому вони зараз стані, визначити плюси і мінуси соціальних мереж та кулінарних веб-сайтів, а також пояснити, чому зараз необхідно створити спеціальну соцмережу для обміну рецептами;

1. Обґрунтування вибору технологічного стеку: провести аналіз сучасних рішень для веб-програмування, пояснити вибір Next.js, React, PostgreSQL, Prisma, Neon та хмарних сервісів для деплою, зберігання контенту і автентифікації користувачів;
2. Розробка основного функціоналу веб-додатку: розробити систему для управління рецептами, надати користувачам можливість спілкуватися через особисті повідомлення, розробити систему сортування контенту, додати на веб-сайт можливість залишати фідбек (коментарі, відповіді, вподобайки), створити окрему сторінку для сповіщень, дати можливість користувачам обирати темну чи світлу тему веб-сайту, кастомізація сторінки профілю;
3. Інтеграція хмарних сервісів у проект: підключити сервіс Clerk для легкої авторизації та реєстрації, додати UploadThing для збереження фото та відео з рецептами, додати Cloudinary для зберігання фото у особистих чатах, а також підключити хмарну базу даних Neon для зберігання даних системи;

4. Тестування та оцінка платформи ефективності: провести вимірювання метрик ефективності у середовищі Chrome DevTools, а також проаналізувати можливі доопрацювання та нові функції застосунку.

Запропоновані цілі дозволять не тільки вирішити поставлену задачу (створення тематичної соцмережі), але і створити фундамент для подальших досліджень та розробок у сфері спеціальних соціальних платформ.

Виклад основного матеріалу дослідження. Перед реалізацією функціональної частини веб-додатку було обґрунтовано вибір технологічного стеку та хмарних сервісів.

Почну я з фреймворку Next.js, одного з найбільш відомих серед open-source фреймворків для створення веб-додатків [1]. Його розробила компанія Vercel у 2016 році з метою спростити створення web-додатків на React з підтримкою SSR (Server-Side Rendering). Server-Side Rendering – це технологія, при якій веб-сторінка генерується не в браузері користувача, а на сервері і це робить завантаження сторінок набагато швидшим. І щоб прискорити якраз налаштування SSR був розроблений фреймворк Next.js, який був створений на базі React і має широкі можливості для рендерингу, маршрутизації та оптимізації веб-ресурсів. Також треба нагадати, що Next.js добре працює з сервісом Render, де можна легко і швидко розгорнути свій додаток, а також він дозволяє швидко створювати тестові середовища.

React – це JavaScript бібліотека з відкритим кодом, яка використовується розробниками для створення інтерфейсів користувача [2]. Розробник може створити незалежний компонент і далі повторно використовувати його в різних частинах застосунку, що полегшує розробку, тести та підтримку коду.

TypeScript – це мова програмування, яку створила компанія Microsoft як альтернатива JavaScript [3]. Головна відмінність TypeScript полягає у підтримці статичної типізації, що дозволяє запобігти помилкам ще на етапі написання коду.

PostgreSQL – це безкоштовна та відкрита об'єктно-реляційна система управління базами даних [5]. Вона може використовуватися у великих web-додатках, де користувачі взаємодіють між собою, обмінюються контентом і отримують персоналізовані рекомендації.

Neon – це хмарна, без серверна та повністю керована служба бази даних, яка базується на PostgreSQL. Мій застосунок Next.js взаємодіє з ORM системою, яка перетворює звичайний код у SQL запити до віддаленої бази PostgreSQL, яка знаходиться на хмарному сервісі Neon.

Prisma – це сучасний ORM інструмент, який сильно полегшує роботу з базами даних. Замість написання складних SQL запитів, Prisma дає зручний API для виконання операцій із даними [4].

Ні одна соцмережа не може бути без фото і відео контенту, тому для зберігання медіафайлів було використано хмарні рішення. Для завантаження файлів у проекті я застосовував UploadThing, а для зберігання фото і відео в особистих повідомленнях – платформу Cloudinary.

Спочатку мною було створено головну сторінку web-додатку. Якщо йти зверху і донизу, то спочатку іде логотип та назва соціальної мережі (Їжборщ), а

трохи далі навігаційна панелька з такими пунктами: головна, сповіщення, повідомлення і профіль. Ну і відповідно ця панель дозволяє швидко переходити на потрібну сторінку, а кнопка головна повертає вас на сторінку Home. Також ліворуч від цих кнопок знаходиться перемикач світлої та темної теми веб-сайту. Нижче я додав блок створення нового рецепту з попереднім переглядом, тобто ви одразу бачити, як буде виглядати фінальна публікація. Ще була реалізована функція завантаження фото та відео, плюс є вибір категорії рецепту (гаряче, холодне, десерти і так далі) і кнопка публікації. Праворуч я додав блок рекомендацій, щоб веб-сайт не здавався пустим. Там користувачу будуть пропонуватися рандомні автори контенту, він може на них одразу підписатися, чи перейти у профіль, побачити що вони викладають і тоді оформити підписку. Під блоком створення публікацій я зробив систему фільтрів. Ви можете зробити сортування за категорією плюс новизною чи популярністю. Це було зроблено для комфорту і зручності користувачів. А вже під фільтрами знаходиться сітка дописів, у який у 3 стовпчики розташовані публікації типу як в Instagram. Кожен елемент (рецепт) інтерактивний, можна навести курсор і побачити кількість вподобайок та коментарів, чи клікнути та відкрити повноцінне вікно публікації з можливістю написати свій коментар, переглядом чужих коментарів і таке інше (рис. 1).

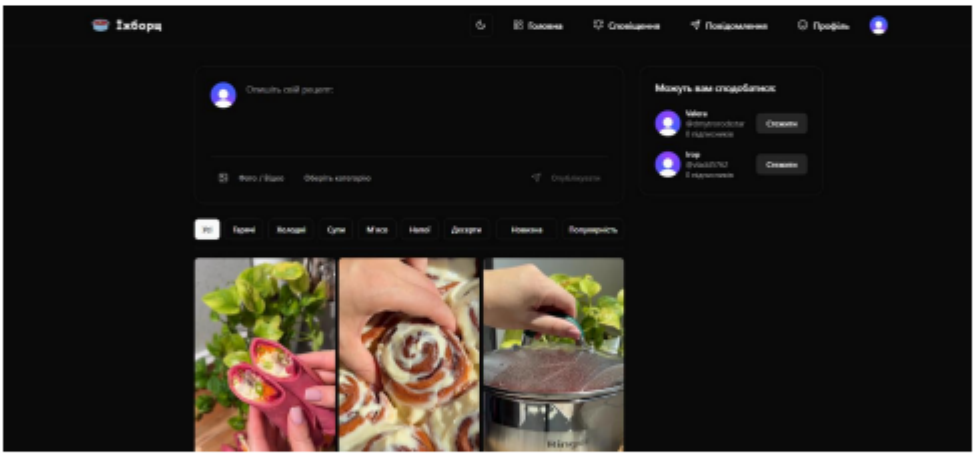

Рисунок 1 – Зовнішній вигляд головної сторінки

На головній сторінці я хотів об’єднати всі необхідні базові функції, які є у кожної соціальної мережі: перегляд чужого контенту, створення особистого, фільтрування вже існуючого та навігація між сторінками веб-сайту. Зовнішній вигляд виконаний у сучасному мінімалістичному стилі, темний фон і контрастні світлі назви кожного елементу добре читаються та зменшують навантаження на очі під час тривалого використання соцмережі.

На веб-сайті підключена дуже потужна система Clerk, яка сильно спрощує процес реєстрації та авторизації користувачів. Це готове рішення, яке ти просто підключаєш у свій проект і через секунду у тебе всі необхідні функції, які

пов'язані з авторизацією та реєстрацією і тобі не треба нічого самостійно розробляти з нуля, це дуже круто і сильно скорочує обсяг власноруч написаного коду, а також це суттєво зменшить кількість можливих помилок у системі. Користувач, коли перший раз відкриває вікно реєстрації, може ввести свою пошту, або скористатися швидким входом через гугл акаунт, але і там і там вас попросять код підтвердження. Також що важливо, сервіс Clerk гарантує безпеку ваших особистих даних, там є шифрування даних і загалом вони на своєму сайті пишуть, що їх система найкраща по інформаційній безпеці. На мій погляд інтеграція Clerk у систему стала виправданим технічним рішенням, яке по'єднує у собі зручність для користувачів, зручність для мене як розробника та безпеку платформи.

Далі хочеться описати карточку публікації (рис. 2), яка є ключовим елементом у системи. З чого вона складається, як виглядає і якій функціонал має.

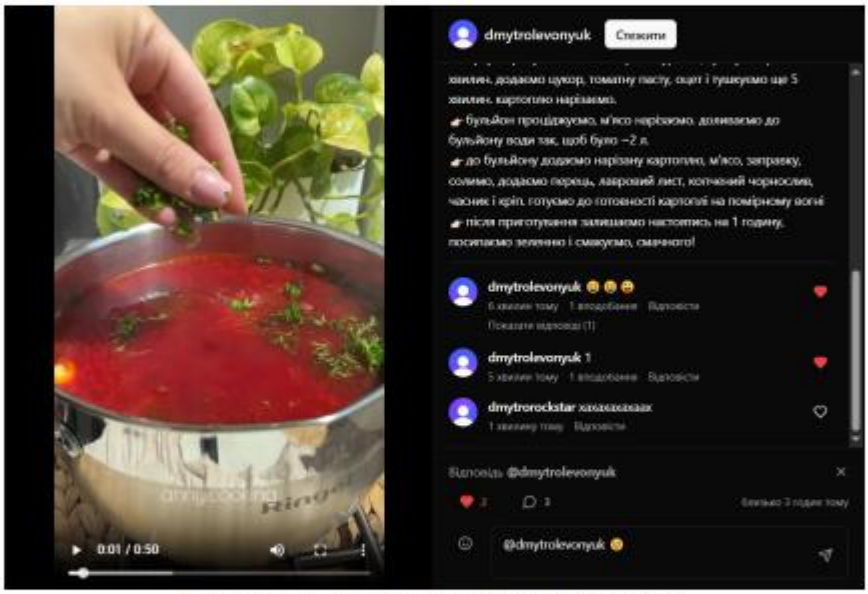

Рисунок 2 – Зовнішній вигляд публікації

Зліва я розташував фотографії, які можна перемикати, або відеоролик, який можна поставити на паузу, відкрити на цілий екран, чи збільшите повзунок гучності. Права частина картки призначена для тексту. Спочатку іде текст рецепту, який підтримує форматування та емодзі, а за ним іде блок коментарів. У цьому блоці любий авторизований юзер може залишите свій коментар, відповісти на вже існуючий, поставити вподобайку на коментар або саму публікацію. Кожен коментар має інформацію про автора, час, коли був опублікований, про кількість вподобань та кількість відповідей. Також я додав кнопку емодзі, на яку ви клікаєте і перед вам сотні різних емодзі і всі вони підтримується і коректно відображаються у коментарях. Блок з коментарями

повністю інтерактивний, є можливість переходити у профілі користувачів просто натиснувши на їх аватарку. Також я додав поле для вводу тексту коментаря, а коли ви комусь відповідаєте, то в цьому полі автоматично через @username ви типу звертаєтесь до людини. А ще я закріпив нікнейм автора, аватарку і кнопку стежити у самому верху публікації, тобто ви скролите і воно залишається на місті.

Далі я розробив сторінку для профіля користувача, яка представляє собою персональний простір з інформацією про автора та невеличкою персоналізацією. На цій сторінці ви можете побачити велику аватарку, ім'я, нікнейм, кількість публікацій, підписників та підписок. Також було додано кнопки стежити і повідомлення для початку приватного листування. А коли ви заходите на свою сторінку, замість кнопки стежити з'являється кнопка редагувати, за допомогою якої можна додатково про себе щось написати, додати місце проживання та посилання на інший ресурс. Нижче я розмістив панельку публікацій і збереженого. Ви можете легко переключатися між ними при необхідності. Сторінка профіля вийшла дуже простою, але в той же час корисною.

Кожна ваша дія у системі додатково створює сповіщення про цю дію, а всі ці сповіщення треба щоб десь зберігалися, тому я і створив наступну сторінку для сповіщень. Ця сторінка відповідає за інформування користувача про всі важливі події з його акаунтом. У верхній частині я додав лічильник кількості непрочитаних сповіщень, щоб користувач одразу міг зрозуміти, чи треба комусь відповісти, чи хтось хоче з ним почати спілкуватися і таке інше. Основна частина складається зі списку сповіщень, які мають форму прямокутників і ідуть один за одним. Кожне сповіщення як окрема карточка, ви можете бачити аватарку людини, нікнейм, що вона зробила і до якої публікації ця дія має відношення. Наприклад, якщо користувач відповів на ваш коментар, у карточці відображається і ваш коментар, на який відповіли і сама відповідь цього користувача. Також я додав до кожного сповіщення час, коли воно було зроблено, додав іконки під кожну дію, а ще сповіщення інтерактивні, ви можете клікнути на любе і у вас відкриється повний допис. Для сповіщень, пов'язаних з дописами, щоб їх одразу можна було впізнати я додав мініатюрки, до якого саме рецепта належить дія.

У моєму web-додатку користувачі зможуть спілкуватися як у повноцінному месенджері, відправляти емодзі, скидати один одному фото чи відео. Все що їм потрібно, це кинути запит на спілкування у особистому профілі людини, з якою вони хочуть почати діалог. Я зробив окрему сторінку для повідомлень, де вони всі діляться на два типи. Перший для вже існуючих діалогів, а другий для запитів на спілкування. Це було зроблено, щоб зменшити кількість спаму зі сторони рандомних користувачів з їх повідомленнями. У кожного блоку повідомлень зліва є аватарка, нікнейм та останнє написане поточним користувачем або його співрозмовником.

Перед вами плюс мінус вікно звичайного діалогу, але з нюансом. Ви бачите перед собою аватарку, нікнейм і перше повідомлення, яке вам написали. Зараз у вас не має можливості відповісти цій людині, для цього спочатку треба прийняти

запит на листування, клікнувши на відповідну кнопку внизу екрану, або проігнорувати цей запит (рис. 3).

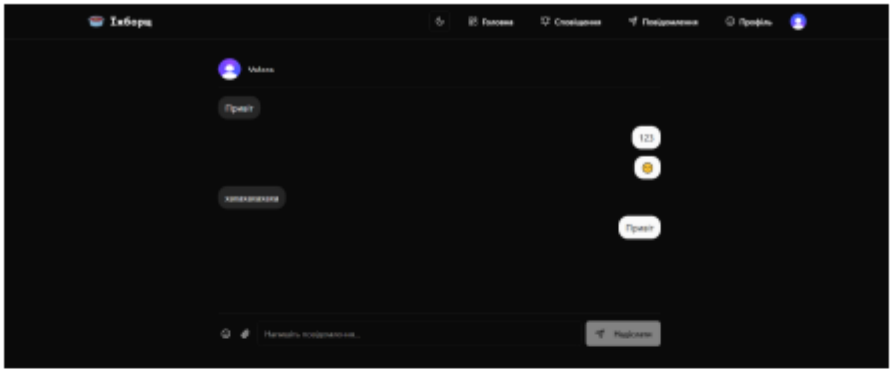

Рисунок 3 – Зовнішній вигляд запиту на листування

На сторінці діалогу (рис. 4) користувач може побачити повну історію переписки зі своїм другом. Я зробив так, що ваші власні повідомлення справа, а повідомлення від вашого співрозмовника знаходяться зліва. В самому верху я додав аватарку і нікнейм людини, з якою ви спілкуєтеся, щоб ви розуміли з ким ведете листування. Знизу у нас поле для вводу тексту, кнопка з іконкою скріпки для вкладень (фото, відео та pdf файли), а також кнопка зі смайликом, яка відкриває емодзі меню. Якщо ви хочете видалити, або редагувати повідомлення, то вам потрібно всього лише дабл клікнути по необхідному повідомленню і відкриється вікно з редагування, кнопкою видалити і кнопкою скасувати дію.

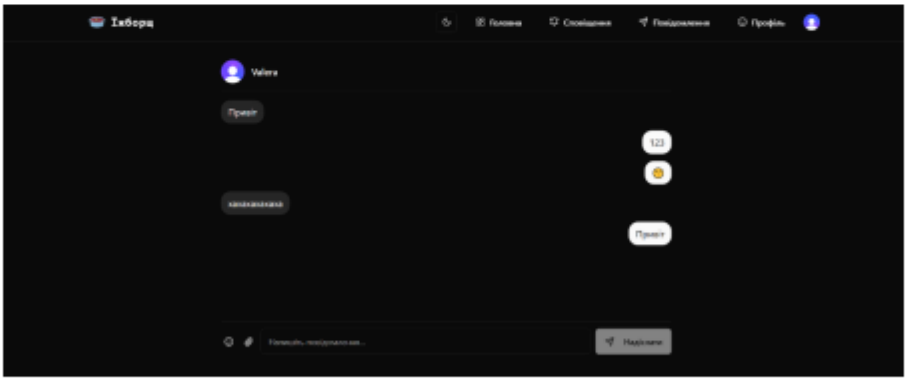

Рисунок 4 – Зовнішній вигляд листування

Я завершив усі необхідні вимірювання та комплексне тестування системи, після чого настав слушний час детально проаналізувати всі отримані результати й сформулювати відповідні висновки. Середній показник TTDB (Time to First Byte) склав 0.55 секунди, коли рекомендоване значення це до 0.2 секунди.

Можно це легко пояснити використанням хмарного сервісу Render і тим, що регіон серверу Франкфурт.

Але не все так погано, наприклад LCP 0.65 секунди, коли граничне значення 2.5 секунди. Це свідчить про швидке завантаження основного наповнення веб-сторінки. Показник CLS трохи перевищує норму, 0.112 замість потрібних 0.1. Це свідчить про наявність порушення візуальної стабільності. Значення INP склало 66 мілісекунд, що значно нижче порога (до 200) і свідчить про високу швидкість реакції інтерфейсу на дії свого користувача.

DOMContentLoaded займає у середньому 0.65 секунди, що характеризує швидку готовність структури сторінки до взаємодії. А повний час завантаження (Load) зайняв 3.18 секунди, що знаходиться у рамках рекомендованих значень.

Швидкість завантаження медіа зайняла 3.55 секунди, що є доволі прийнятним результатом для сторінки з медіа файлами. Середній чай SPA переходу з головної сторінки на сторінку сповіщень зайняв приблизно 152 мілісекунди, що є гарним результатом та підтверджує ефективність реалізованої клієнтської структури.

Аналіз навантаження на центральний процесор показав, що сумарний час Scripting + Rendering у середньому займає 318 мілісекунд, що свідчить нам про достатню ефективність використання обчислювальних ресурсів браузера без надмірних навантажень на систему.

Таким чином мені вдалося створити веб-додаток соціальної мережі, який демонструє стабільність та високу продуктивність у роботі, а також отримані експериментальні результати вимірювань підтверджують, що він у більшості відповідає сучасним вимогам до швидкодії та якості користувацького досвіду, з урахуванням використання хмарної платформи Render.

Висновки. У статті було проаналізовано процес проектування та розробки веб-додатку соціальної мережі для обміну рецептами із використанням Next.js. Розроблений застосунок демонструє стабільну роботу, зручний інтерфейс та ефективну обробку користувацьких запитів, що підтверджує доцільність створення спеціалізованих тематичних соцмереж.

Перелік джерел посилання

1. Next.js Documentation. URL: https://nextjs.org/docs (дата звернення: 03.10.2025)
2. React – офіційна документація. URL: https://react.dev/learn (дата звернення: 07.10.2025)
3. TypeScript Documentation. URL: https://www.typescriptlang.org/docs/ (дата звернення: 10.10.2025)
4. Prisma ORM – офіційна документація. URL: https://www.prisma.io/docs (дата звернення: 12.10.2025)
5. PostgreSQL: Official Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 05.10.2025)