Хмельницький національний університет
Факультет інформаційний технологій
Кафедра інженерії програмного забезпечення

ДИПЛОМНИЙ ПРОЕКТ

Веб-система соціальної мережі «Blanket»
Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр ДПІПЗ.180104.18.02.ПЗ

Виконав студент IV курсу група ІПЗ-18-1 _____ Є. Д. Зубашевський
Підпис Ініціали, прізвище

Керівник канд. техн. наук, доцент _____ І. В. Гурман
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер канд. техн. наук, доцент _____ Ю. В. Форкун
Підпис Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії
програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

6 червня 2022 р.

Хмельницький 2022

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційний технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
05 02 2022 р.

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ)

Зубашевському Єгору Денисовичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Веб-система соціальної мережі «Blanket»

Керівник проекту (роботи) Гурман Іван Васильович, канд. техн. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 05.02.2022 р. № 11
2. Строк подання студентом проекту (роботи) на кафедру 01.06.2022 р.
3. Вихідні дані до проекту (роботи) Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація, тестування програмного забезпечення
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Презентаційні матеріали (слайди, 14 шт.)

6. Консультанти розділів дипломного проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийня
Нормоконтроль	Форкун Ю. В., к.т.н. доцент		
Антиплагіат	Гурман І. В., к.т.н. доцент		

7. Дата видачі завдання « 05 » лютого 2022р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1 Ознайомлення з тематикою дипломного проектування (ДП), визначення та узгодження індивідуальних там ДП	01.12 – 30.12.2021	
2 Дослідження предметної області, в якій планується використання програмного засобу (ПЗ), визначення задач та вимог, розробка технічного завдання	02.01 – 31.01.2022	
3 Проектування програмного забезпечення	01.02 – 28.02.2022	
4 Програмна реалізація	01.03 – 10.04.2022	
5 Тестування програмного забезпечення	11.04 – 30.04.2022	
6 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки ДП згідно вимог стандартів	01.05 – 25.05.2022	
7 Попередній захист ДП	Травень 2022 (згідно графіка)	
8 Перевірка ДП на плагіат, нормконтроль, отримання відгуків та рецензій. Брошурування (зшиття) пояснювальної записки	26.05 – 30.05.2022	
9 Підготовка до захисту та захист ДП	з 01.06.2022	

Студент _____ Є. Д. Зубашевський
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ І.В. Гурман
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема дипломного проекту: Веб-система соціальної мережі «Blanket».

Автор проекту: Зубашевський Єгор Денисович.

Керівник проекту: Гурман Іван Васильович.

Пояснювальна записка: 73 с., 19 рис., 4 табл., 3 дод., 19 джерел.

Графічна частина: 14 слайдів.

СОЦІАЛЬНА МЕРЕЖА, ВЕБ-ДОДАТОК, JAVASCRIPT, TYPESCRIPT, NODE.JS, REACT.JS, GRAPHQL.

Метою проекту є розробка веб-системи у вигляді соціальної мережі, яка дозволятиме її користувачам оприлюднювати та взаємодіяти із публікаціями , а також реєструватись та авторизуватись.

У дипломному проекті було проаналізовано предметну область, з'ясовані причини частого виникнення недовіри користувачів соціальним мережам, низької швидкості та поганої оптимізації, проведено аналіз схожого програмного забезпечення, були порівняні архітектурні стилі для розробки веб-додатків, визначені модулі системи та здійснена їх реалізація.

Для розробки мобільного додатку була використана мова програмування TypeScript, фреймворк Node.js, фреймворк React.js та база даних PostgreSQL.

В результаті було розроблено веб-систему для оприлюднення та взаємодії із публікаціями.

31.05
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	ДПІПЗ.180104.18.02.ПЗ	Пояснювальна записка	73		
2	А4		Завдання на дипломний	1		
			проект			
3	А4		Анотація	1		
			Графічні документи			
4	А4		Презентаційні матеріали	14		

Змн.	Арк.	№ докум.	Підпис	Дата	ДПІПЗ.180104.18.02.ВД			
Виконав		Зубашевський Є. Д.		31.09	Веб-система соціальної мережі «Blanket»	Літ.	Арк. 1	Аркушів 1
Керівник		Гурман І.В.		3.06				
Н. Контр.		Форкун Ю.В.		07.06	Відомість документів	ХНУ, ІПЗ-18-1		
Зав. Каф.		Бедратюк Л.П.		7.06				

ЗМІСТ

Змн.	Арк.	№ докум.	Підпис	Дата	ДПІПЗ.180104.18.02.ВД		
Виконав		Зубашевський Е. Д.		31.05	Веб-система соціальної мережі «Blanket»	Літ.	Арк. 1 / Аркушів 1
Керівник		Гурман І.В.		3.06			
Н. Контр.		Форкун Ю.В.		07.06	Відомість документів	ХНУ, ІПЗ-18-1	
Зав. Каф.		Бедратюк Л.П.		7.06			

ВСТУП

Менш ніж за покоління соціальні мережі еволюціонували від прямого електронного обміну інформацією до віртуального місця збору, до роздрібної платформи, до життєво важливого інструменту маркетингу 21-го століття.

У певному сенсі соціальні мережі почалися 24 травня 1844 року з серії електронних точок і тире, вистукиваних вручну на телеграфній машині. Хоча коріння цифрової комунікації сягають глибоко, більшість сучасних розповідей про витоки сучасного Інтернету та соціальних мереж вказують на появу в 1969 році мережі агентства передових дослідницьких проектів – ARPANET.

Ця рання цифрова мережа, створена Міністерством оборони Сполучених Штатів, дозволила вченим із чотирьох взаємопов'язаних університетів обмінюватися програмним, апаратним та іншими даними.

У 1987 році прямий попередник сучасного Інтернету з'явився, коли Національний науковий фонд запустив більш надійну загальнонаціональну цифрову мережу, відому як NSFNET. Через десять років, у 1997 році, була запущена перша справжня платформа соціальних мереж.

У 1980-х і 90-х роках, зростання Інтернету дозволило запровадити онлайн-комунікаційні послуги, такі як CompuServe, America Online та Prodigy. Вони познайомили користувачів із цифровим спілкуванням за допомогою електронної пошти, обміну повідомленнями на дошці оголошень та онлайн-чату в режимі реального часу. Це породило перші соціальні мережі, починаючи з короткочасної служби завантаження профілю Six Degrees у 1997 році.

За цією послугою у 2001 році послідував Friendster. Ці елементарні платформи привернули мільйони користувачів і дозволили зареєструвати адресу електронної пошти та створити базову мережу в Інтернеті. Веблоги, або блоги, ще одна рання форма цифрового соціального спілкування, почали набувати популярності з запуском сайту видавництва LiveJournal у 1999 році. Це співпало

із запуском видавничої платформи Blogger технологічною компанією Pyra Labs, яку Google придбала у 2003 році.

У 2002 році LinkedIn був заснований як мережевий сайт для професіоналів, які прагнуть до кар'єри. До 2020 року його кількість зросла до понад 675 мільйонів користувачів у всьому світі. Він залишається улюбленим сайтом соціальних мереж для тих, хто шукає роботу, а також менеджерів з персоналу, які шукають кваліфікованих кандидатів. Два інших серйозних виходу на соціальні мережі провалилися після сплеску початкового успіху. У 2003 році був запущений Myspace. У 2006 році це був найвідвідуваніший веб-сайт на планеті, завдяки можливості користувачів ділитися новою музикою безпосередньо на сторінках профілю.

У 2008 році його затьмарив Facebook. У 2011 році музикант Джастін Тімберлейк придбав Myspace за 35 мільйонів доларів, але з тих пір він перестав існувати як популярна соціальна мережа.

Спроба Google пробитися в ландшафт соціальних мереж, Google+, була запущена в 2012 році. Невдале існування закінчилося в 2018 році після того, як приватна інформація майже п'ятиста тисяч користувачів Google+ була скомпрометована порушенням безпеки даних.

Спочатку соціальні мережі існували, щоб допомогти кінцевим користувачам зв'язатися в цифровому вигляді з друзями, колегами, членами сім'ї та однодумцями, з якими вони, можливо, ніколи не зустрічалися особисто. Доступ із комп'ютера до сервісів дошки оголошень, таких як CompuServe і Prodigy, полегшив створення безкоштовних онлайн-спільнот, не виходячи з дому. Винахід смартфона звільнив соціальні мережі від настільного та портативного комп'ютерів. Перший iPhone від Apple, випущений Стівом Джобсом у 2007 році, допоміг перенести фокус створення онлайн-спільноти на мобільні пристрої. Facebook, Twitter, Snapchat, Instagram, TikTok та інші соціальні мережі процвітали в середовищі мобільних додатків.

Технологічні вдосконалення – зокрема, потужні камери в телефоні – перемістили фокус мобільних додатків на відео та зображення. На додаток до письмових повідомлень, кінцеві користувачі тепер могли транслювати в режимі реального часу.

Instagram, зокрема, став улюбленим додатком для користувачів соціальних мереж, які цікавляться подорожами, розвагами, модою та іншими візуально орієнтованими темами.

У міру того як компанії соціальних мереж розширили свою базу користувачів до сотень мільйонів, бізнес-додатки Facebook, Twitter та інших соціальних платформ почали формуватися. Компанії соціальних мереж мали доступ до найбагатших даних користувачів, які коли-небудь були створені.

Facebook почав розміщувати рекламу на своїй платформі ще в 2006 році. Twitter включив рекламу в 2010 році. LinkedIn, Instagram, Pinterest, Snapchat і TikTok намагалися монетизувати свої послуги за допомогою різних форм спонсорованої реклами.

Окрім розміщення реклами на платформах соціальних мереж, компанії виявили потенційну користь від культивування активної та зацікавленої присутності в соціальних мережах. У той час як реклама в соціальних мережах має бути оплачуваною, акт створення та поширення інформаційного чи розважального контенту на Facebook, Instagram, Twitter та інших платформах є спробою брендів органічно збільшити аудиторію, іншими словами, не сплачуючи за це безпосередньо.

Сьогодні існує величезна різноманітність соціальних мереж. Це створює середовище, в якому користувачі можуть охопити максимальну кількість людей, не жертвуючи інтимністю спілкування від людини до людини. Ми можемо лише здогадуватися про те, яким може виглядати майбутнє соціальних мереж у найближче десятиліття або навіть через 100 років, але здається очевидним, що воно існуватиме в тій чи іншій формі доти, доки будуть живі люди.

Мета проекту – розробка веб-додатку у вигляді соціальної мережі, яка дозволятиме її користувачам оприлюднювати публікації, реєструватися, авторизуватися та реагувати на публікації інших користувачів за допомогою елементів інтерфейсу користувача та забезпечуватиме безпеку даних користувача.

Завдання, які необхідно вирішити для досягнення мети:

– провести аналіз предметної області для визначення її головних особливостей;

– провести аналіз наявного програмного забезпечення;

– сформувати технічне завдання;

– створити програмний продукт;

– ознайомитися та провести аналіз існуючих рішень;

– ознайомитись з сучасними методоми забезпечення безпеки користувацької інформації;

– виконати програмну реалізацію проекту використовуючи при цьому сучасні та безпечні технології;

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Веб-розробка – це робота, пов’язана з розробкою веб-сайту для Інтернету (World Wide Web) або інтранету (приватної мережі). Веб-розробка може варіюватися від розробки простої статичної сторінки звичайного тексту до складних веб-додатків, електронних компаній та служб соціальних мереж. Більш повний список завдань, до яких зазвичай відноситься веб-розробка, може включати веб-інженерію, веб-дизайн, розробку веб-контенту, зв’язок із клієнтом, сценарії на стороні клієнта/сервера, конфігурацію безпеки веб-сервера та мережі та розробку електронної комерції.

Серед веб-фахівців «веб-розробка» зазвичай відноситься до основних не-дизайнерських аспектів створення веб-сайтів: написання розмітки та кодування. Веб-розробник може використовувати системи керування вмістом (CMS), щоб зробити зміни вмісту простішими та доступними за допомогою базових технічних навичок.

У сучасному світі з’являються нові технології та стандарти веб-розробки, а багато старих технологій розвиваються та змінюються майже щохвилини. Деякі з них не зазнали істотних змін з моменту свого народження, тому в багатьох випадках такі технології вже не знаходяться в активному використанні.

Але, як і скрізь, є винятки, одним з яких є архітектура веб-додатків RESTful. RESTful був заснований в 2000 році американським вченим Роєм Філдінгом. У рік свого створення це був революційний винахід, який започаткував нову еру веб-розробки, але зараз, в еру швидкого розвитку, я вважаю, що технологія, яка не змінювалася протягом 22 років не може бути стандартом архітектури серверної частини веб-розробки.

Тому я почав вивчати та аналізувати альтернативи застарілій архітектурі, та зупинився на внутрішній розробці Facebook – GraphQL.

GraphQL – це мова запитів і маніпуляцій даними, а також середовище виконання запитів із наявними даними. GraphQL був створений у 2012 році, у 2015 році потрапив у відкритий доступ усім бажаючим.

REST пропонує створити об'єкт даних, запитаних клієнтом, і надіслати значення об'єкта у відповідь користувачеві. Наприклад, якщо користувач запитує фільм у Бангалорі в певному місці та в певний час, ви можете створити об’єкт на стороні сервера.

Отже, тут у вас є об’єкт, і ви надсилаєте стан об’єкта. Ось чому REST відомий як репрезентаційна передача стану. REST(Representational State Transfer) – це архітектурний стиль, а також підхід до комунікаційних цілей, який часто використовується при розробці різних веб-сервісів.

Архітектурний стиль REST допомагає використовувати меншу пропускну здатність, щоб зробити додаток більш придатним для Інтернету. Його часто вважають «мовою Інтернету» і повністю базується на ресурсах.

Щоб краще зрозуміти, давайте зануримося трохи глибше і подивимося, як саме працює REST API. В основному, REST API розбиває транзакцію, щоб створити невеликі модулі. Тепер кожен із цих модулів використовується для вирішення певної частини транзакції. Цей підхід забезпечує більшу гнучкість, але вимагає багато зусиль для створення з самого нуля.

Існує чотири основних принципів, закладених Філдінгом. Нижче наведено чотири керівних принципів REST:

– Невизначений

Запити, надіслані від клієнта до сервера, будуть містити всю необхідну інформацію, щоб сервер зрозумів запити, надіслані від клієнта. Це може бути як частина URL-адреси, параметри рядка запиту, тіло або навіть заголовки. URL-адреса використовується для однозначної ідентифікації ресурсу, а тіло містить стан запитуваного ресурсу. Після того, як сервер обробляє запит, відповідь надсилається клієнту через тіло, статус або заголовки.

– Клієнт-Сервер

Архітектура клієнт-сервер забезпечує єдиний інтерфейс і відокремлює клієнтів від серверів. Це покращує переносимість на кількох платформах, а також масштабованість компонентів сервера.

– Кешування

Для забезпечення кращої продуктивності програми часто кешуються. Це робиться шляхом позначення відповіді від сервера як кешується або не кешується або неявно, або явно. Якщо відповідь визначена як кешована, то кеш клієнта може повторно використовувати дані відповіді для еквівалентних відповідей у майбутньому.

– Багатошарова система

Багатошарова архітектура системи дозволяє додатку бути більш стабільним, обмежуючи поведінку компонентів. Цей тип архітектури допомагає підвищити безпеку програми, оскільки компоненти кожного рівня не можуть взаємодіяти за межами наступного безпосереднього рівня, на якому вони знаходяться. Крім того, він дозволяє балансувати навантаження та надає спільні кеші для підвищення масштабованості.

В REST обмін даними між клієнтською частиною та серверною відбувається частіше за все за допомогою чотирьох основних HTTP методів, а саме:

– Метод GET:

Метод GET зазвичай використовується для отримання або отримання всіх або окремих ресурсів веб-сервера. Цей метод зазвичай використовується для відображення даних на стороні клієнта

– Метод PUT:

Метод PUT – це метод оновлення даних, присутніх у базі даних, і тут цей метод зазвичай можна використовувати для зміни значень, які зберігаються в базі даних;

– Метод POST:

Метод POST в основному використовується для передачі даних від клієнта до сервера. Припустимо, що користувач заповнює форму своїми

обліковими даними на будь-якому веб-сайті, тоді ці дані будуть надалі зберігатися в базі даних за допомогою методу POST;

– Метод DELETE:

Як зрозуміло з назви, ми використовуємо метод DELETE для видалення окремих або всіх ресурсів із бази даних.

GraphQL зазвичай описують як технологію API, спрямовану на клієнтську частину, оскільки вона дозволяє розробникам клієнтської частини запитувати дані набагато простіше, ніж будь-коли раніше. Запроваджена Facebook, мета цієї мови запитів полягає в тому, щоб сформулювати клієнтські програми, сформовані в інстинктивному та настроюваному форматі, щоб відобразити передумови їх даних, а також взаємодії. Найкраще те, що мова не залежить від жодної конкретної системи керування базами даних і насправді підтримується вашими поточними даними та кодуванням.

Однією з основних проблем зі звичайним REST є нездатність клієнта вимагати персоналізований набір даних. На додаток до цього, запуск і контроль кількох кінцевих точок є ще однією складністю, оскільки клієнти в основному потрібні для запиту даних з різноманітних кінцевих точок.

Під час створення сервера GraphQL важливо мати єдину URL-адресу для повного отримання та зміни даних. Таким чином, користувач може запросити набір даних від сервера, передавши рядок запиту, вказавши, що йому потрібно.

Говорячи про схожість, REST і GraphQL використовуються для створення API. Крім того, обома ними можна керувати через HTTP. Що стосується відмінностей, то REST є насамперед структурною концептуалізацією програмного забезпечення, орієнтованого на мережу, не має специфікацій і не має певного набору інструментів. Він більше зосереджується на довговічності API, а не на оптимізації продуктивності.

GraphQL, з іншого боку, –це мова запитів, розроблена для роботи над однією кінцевою точкою через HTTP, покращуючи продуктивність та адаптивність. Деякі з інших помітних відмінностей включають:

– Отримання даних:

Вибір даних, безумовно, є одним з найпереконливіших досягнень, представлених GraphQL. У стандартному API REST для отримання або відновлення даних нам може знадобитися робити запити до численних кінцевих точок. Для порівняння GraphQL пропонує єдину кінцеву точку, через яку ми отримуємо доступ до даних, доступних на сервері.

– Понад або занижена вибірка даних:

Значно легше отримати більше даних, ніж вам потрібно в REST, ніж у GraphQL, оскільки кожна кінцева точка в специфікації REST містить узгоджене формування даних. Аналогічно, з REST порівняно легше отримати недобір даних, що дозволяє клієнтам робити додаткові запити для отримання відповідних даних.

– Управління помилками:

Управління помилками в специфікації REST досить просте. Все, що нам потрібно зробити, це перевірити заголовки HTTP, щоб дізнатися позицію відповіді. Залежно від коду стану ми можемо швидко вказати помилку, а також відповідний спосіб її вирішення. З іншого боку, ми завжди отримуємо статус 200 OK у разі GraphQL.

REST було введено у відповідь на вимогу більш практичного та адаптивного підходу до публікації та використання веб-сервісів. Концепція була дотепно прямою та повністю без громадянства, отже, відкидала будь-які невідповідні ускладнення. На додаток до цього, цей підхід також забезпечив легке змішування з JSON і XML. Але знову ж таки, уніфікація даних була найбільшою перешкодою. Крім того, ще однією проблемою були розбіжності щодо того, як слід керувати версіями. Щоб вирішити цю ситуацію, Facebook виступив і надав розробникам рішення, яке пропонує найкраще з обох світів – GraphQL.

GraphQL – не єдине рішення без конкретного й практичного пояснення. RESTful API мають перевірену ефективність і продуктивність протягом багатьох років. GraphQL затьмарює відсутність REST, тоді як REST заповнює

порожнечі, наявні в GraphQL. Примітно, що ситуація з GraphQL і REST цілком ідентична ситуації з реляційними та NOSQL базами даних.

Використовуючи GraphQL, HTTP, безумовно, є найбільш кращим варіантом для протоколу клієнт-сервер, і це значною мірою через його поширеність. Однак продуктивність викликає сумніви, коли справа доходить до обслуговування через HTTP 2. Хоча GraphQL вирішує чимало проблем, які ми маємо, все одно було б важко вибрати будь-яку специфікацію API, оскільки, швидше за все, у якийсь момент ви повинні мати обидві.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Існує безліч соціальних мереж різного призначення, які займають різні особливі місця серед користувачів. Деякі з них призначені лише для обміну відео, деякі для обміну фотографіями, деякі для публікації вакансії. Проте найпопулярнішою досі залишається соціальна мережа розважального характеру. Більшість соціальних мереж орієнтовані на різну аудиторію та працюють за різними принципами, як соціальними, так і технічними.

Розглянемо найпопулярніші соціальні мережі у відповідних сферах, усі вони працюють із підключенням до Інтернету. Доступ до соціальних мереж можна отримати не тільки через Інтернет-браузер, а й через мобільний додаток.

Першою розглянемо Facebook, продукт компанії Meta. Facebook – це веб-сайт, який дозволяє користувачам, які зареєструвалися, спілкуватися в Інтернеті з друзями, колегами по роботі або людьми, яких вони не знають. Це дозволяє користувачам ділитися зображеннями, музикою, відео та статтями, а також власними думками.

Після прийняття два профілі пов’язані з обома користувачами, які можуть бачити, що публікує інша особа. Користувачі можуть публікувати на своїй сторінці практично будь-що, знімок того, що відбувається в їх колі спілкування

в будь-який момент часу, а також можуть вступати в приватний чат з іншими друзями, які в мережі.

Люди з профілями вказують інформацію про себе. Незалежно від того, чим вони займаються, де навчаються, віком чи іншими особистими даними, багато користувачів публікують багато інформації, яка легко доступна їхнім друзям та іншим. Крім того, користувачі можуть реагувати на інші сторінки, які їх цікавлять.

Клієнт доступний у вигляді веб-додатка на всіх сучасних інтернет-браузерах і мобільних платформах.

Із ключових можливостей додатку варто відзначити наступні:

– Оприлюднення текстових публікацій із фото-контентом;
– Публікація відео-контенту;
– Проведення подій у реальному часі у формать відео-трансляції;
– Інтеграція контактів з адресної книги;
– Планування подій у реальних локаціях;
– Слідкування за цікавими користувачу людьми;
– Слідкування за цікавими користувачу публікаціями за “хештегами”.

В плані зовнішнього вигляду Facebook відрізняється з-поміж інших соціальних мереж простим та приємним інтуїтивно зрозумілим інтерфейсом користувача. У ньому досить легко орієнтуватися, а більшість дій можна виконати всього за декілька простих кроків. Зовнішній вигляд головної сторінки показано на рисунку 1.1.

Серед недоліків варто відзначити, що популярність цієї мережі спадає, тому що компанію, яка виробляє цей додаток, звинувачують у продажі даних користувачів. Тому дані користувача не є захищеними, якщо вони були розміщені на просторах цієї мережі.

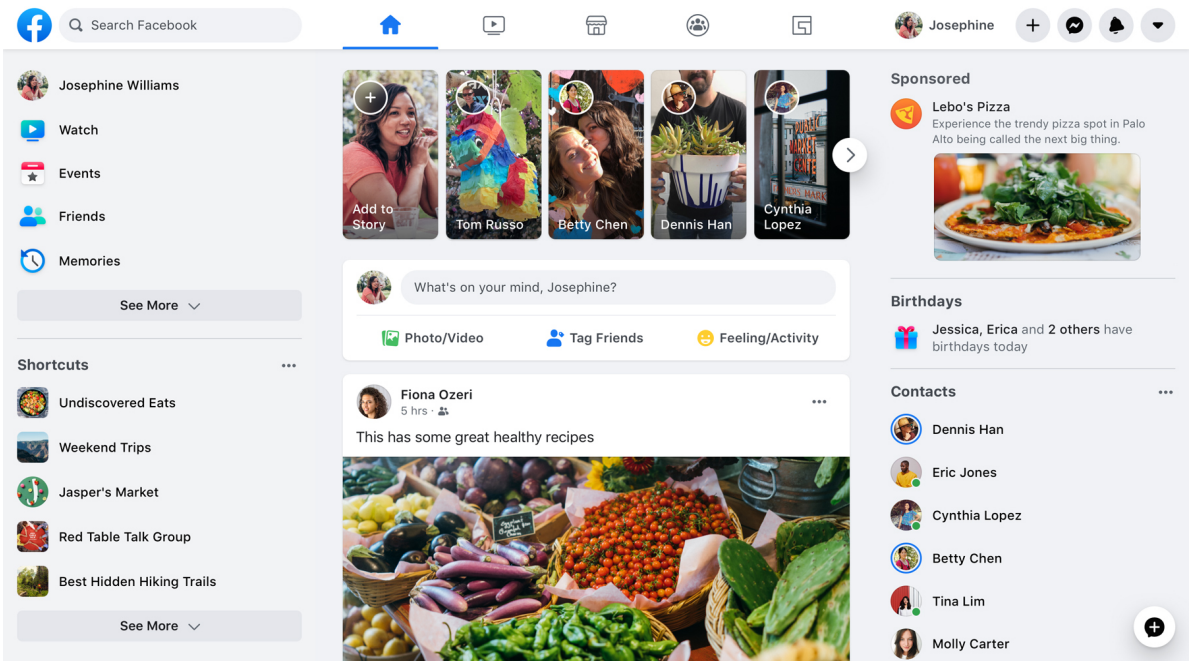


Рисунок 1.1 – Головна сторінка Facebook

Twitter – це безкоштовна соціальна мережа, де користувачі транслюють короткі повідомлення, відомі як твіти. Ці твіти можуть містити текст, відео, фотографії або посилання. Щоб отримати доступ до Twitter, користувачам потрібне підключення до Інтернету або смартфон для використання програми чи веб-сайту.

Це сервіс мікроблогів – комбінація ведення блогів і обміну миттєвими повідомленнями – для зареєстрованих користувачів, щоб публікувати, ділитися, лайкати та відповідати на твіти короткими повідомленнями. Незареєстровані користувачі можуть лише читати твіти. Користувачі вибирають, що вони хочуть бачити в Twitter, відстежуючи інших користувачів та шукаючи теми. Як правило, часова шкала відображає уподобання користувачів, але вони можуть бачити ретвіти від людей, на яких вони підписані, і рекламні твіти, які є платною рекламою.

До ключових особливостей додатку можна віднести:

– Оприлюднення публікацій;

– Відстежування інших користувачів;

– Пошук цікавих тем за спеціальнмим помітками(хештегами);

– Можливість розповсюдити публікацію іншого користувача на своїй сторінці(ретвіт).

Головна сторінка Twitter показано на рисунку 1.2.

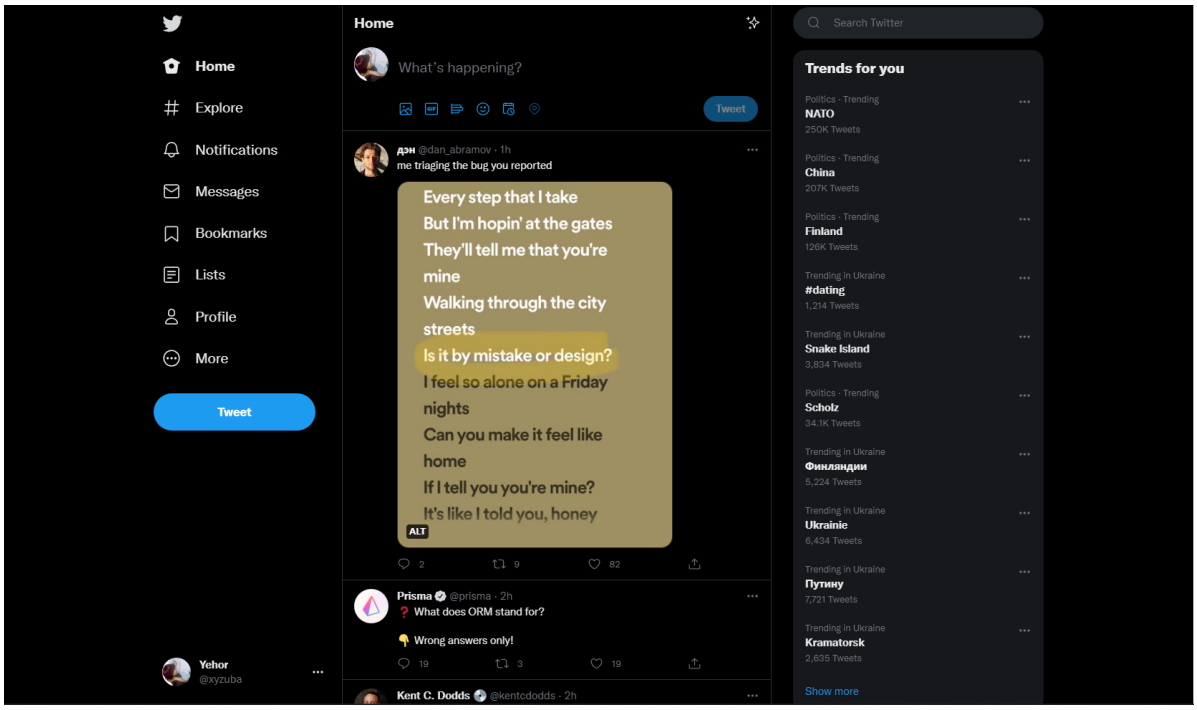


Рисунок 1.2 – Головна сторінка Twitter

Основним недоліком цієї соціальної мережі є велика кількість рекламних оголошень, які відображаються у формі повідомлень у секції коментарів та у вигляді публікацій. Реклама варіюється від фішингових оголошень до оголошень від найбільших автомобільних компаній.

YouTube – це популярний веб-сайт для обміну відео, де зареєстровані користувачі можуть завантажувати та ділитися відео з усіма, хто має доступ до сайту. Ці відео також можна вставляти та ділитися на інших сайтах. YouTube був розроблений колишніми співробітниками PayPal у 2005 році і був придбаний Google у 2006 році. доступна на будь-якому сучасному девайсі та веб-браузері, являється найпопулярнішою мережею для публікацію відео

Наразі також є однією із найпопулярніших платформ для трансляцій відео у реальному часі.

Головну сторінку YouTube показано на рисунку 1.3.

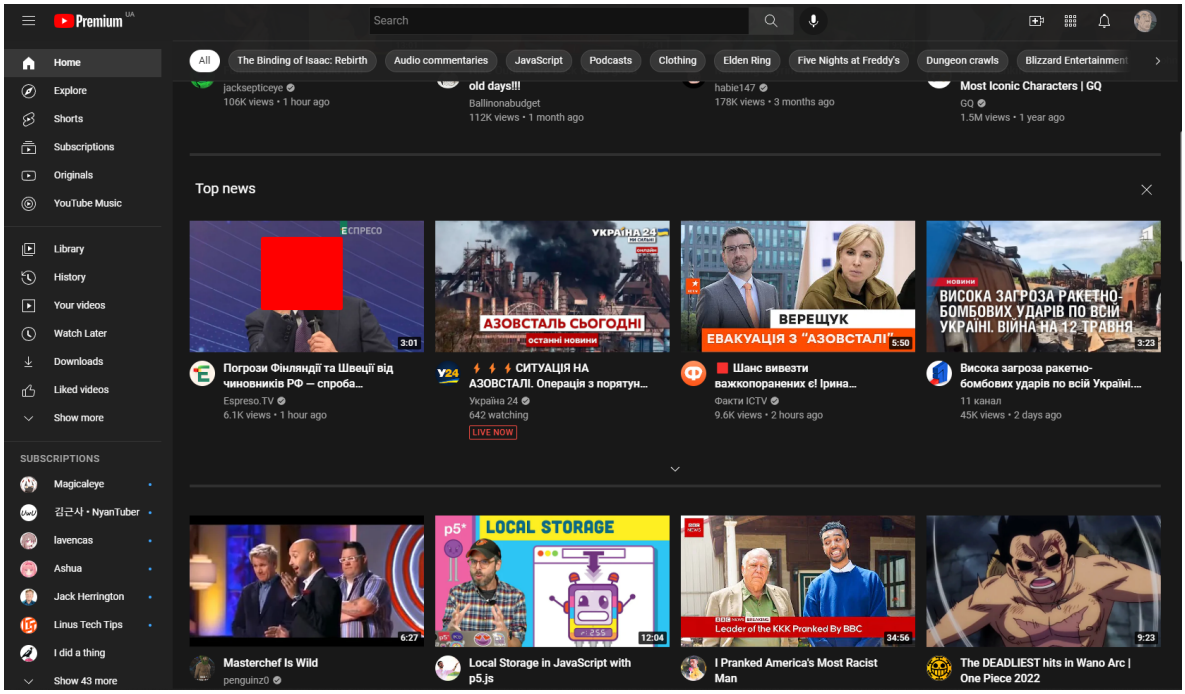


Рисунок 1.3 – Головна сторінка YouTube

Із ключових можливостей додатку варто відзначити наступні:

– Оприлюдення відео-контенту;

– Проведення онлайн-трансляцій у реальному часі;

– Опубліковані відео-ресурси назавджи залишаються у сховищі на сервері;

– Можливість збереження відео та аудіо при наявності коштовної підписки.

Сучасна версія платформи підтримує завантаження відео-контенту розширенням до 4К, а у випадку 360 ° відео навіть – 8К. Основним недоліком є велика кількість реклами та необхідність придбання коштовної підписки задля використання додатку у повному обсязі.

LinkedIn – це професійний мережевий веб-сайт. У той час як інші соціальні мережі, такі як Facebook, більше зосереджені на розвагах. LinkedIn дозволяє професіоналам створювати та підтримувати ділові зв'язки, орієнтовані на кар'єру, та просувати свої послуги чи професійні навички.

Користувачі LinkedIn створюють професійні профілі, включаючи такі деталі, як їхня освіта, історія роботи або роботи, список основних проектів і професійних досягнень, професійні сертифікати та членство в професійних організаціях. Вся ця інформація розміщується в розділі «Профіль» користувача.

LinkedIn також є ресурсом для рекрутерів. Завдяки багатству детальних професійних даних, доступних для пошуку, сайт перетворився на цінний інструмент для рекрутерів і кадрових агентств, а також тих, хто шукає роботу. Фактично, можливості пошуку роботи та працевлаштування LinkedIn перетворилися на одну з його найбільш часто використовуваних функцій. Щоб допомогти в цьому, LinkedIn надає функцію «Підписуватися», щоб користувачі могли бути в курсі пропозицій щодо роботи організації на додаток до інших новин про організацію.

Головну сторінку LinkedIn показано на рисунку 1.4.

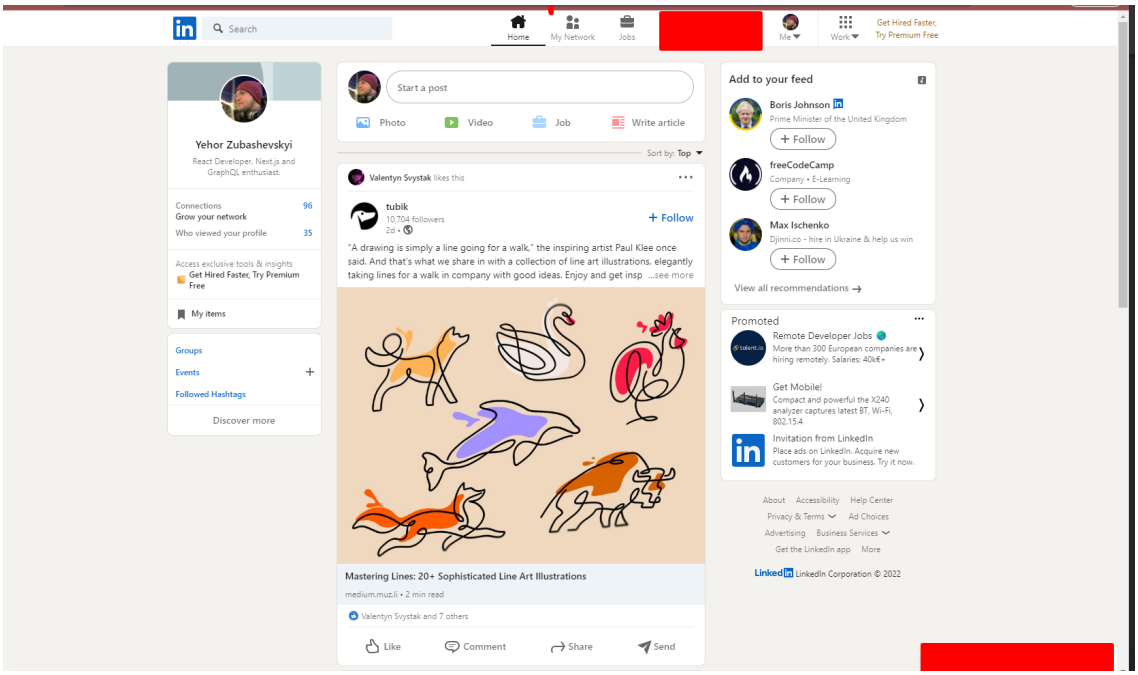


Рисунок 1.4 – Головна сторінка LinkedIn

Основні особливості LinkedIn:

– Користувачі можуть публікувати фотографії та переглядати профілі та фотографії інших користувачів;

– Користувачі можуть переглянути, скільки людей нещодавно шукали та переглядали їх, хоча для отримання більш детальної інформації потрібно платне оновлення до преміум-аккаунта;

– Роботодавці можуть перераховувати та шукати потенційних кандидатів;

Головні недоліки: багато некомпетентних компаній, які невірно оформлюють вакансії, рекрутери які пропонують роботу невідповідну твоїм навичкам, та необхідність придбання коштовної підписки задля використання платформи у повному обсязі.

Отже, не варто забувати, що для соціальної мережі важлива не лише зручність, швидкість, та спосіб шифрування файлів користувача, але й надійність та репутація в цілому. Занадто велика кількість популярних додатків так чи інакше продають та отримують несанкціоновану вигоду від даних користувачів, це можна помітити за відсутністю існуючих соціальних мереж з відкритими вихідним кодом. Та більшість додатків використовують застарівші та не релевантні у сучасному світі технології, що негативно сприяє на швидкість самої системи обміну даними та відкриває щілини для проведення хакерських атак, що так само ставить користувацькі дані у небезпеку.

Проведений аналіз подібного програмного забезпечення показав, що розроблюваний додаток повинен:

– Мати надійну систему обміну інформацією;

– Мати зрозумілий та простий для використання інтерфейс користувача;

– Мати можливість оприлюднення публікацій;

– Мати можливість редагувати та видаляти раніше створені публікацій;

– Мати можливість реагувати на публікації інших користувачів;

– Бути створений за допомогою сучасних та надійних технологій та стандартів веб-розробки;

– Мати можливість реєстрації та авторизації.

1.3 Визначення вимог до мобільного додатку

Опишемо вимоги до розроблюваного веб-додатку та покажемо, що саме зможе зробити користувач за допомогою діаграми варіантів використання (ВВ).

У таблиці 1.1 подано опис акторів (користувачів додатку), а в таблиці 1.2 – самих варіантів використання.

Таблиця 1.1 – Опис акторів

Актор	Короткий опис
Неавторизований користувач	Може проглядати раніше оприлюднені публікації іншими користувачами, зареєструватись, авторизуватись.
Авторизований користувач	Може створювати нові публікації, редагувати вже існуючі публікації, які були оприлюднені ним, вийти із облікового запису, реагувати на публікації інших корисувачів, видаляти публікації , які були оприлюднені ним.

Таблиця 1.2 – Опис варіантів використання

Актор	Назва ВВ	Опис ВВ
1	2	3
Не авторизований користувач	Реєстрація	Користувач може зареєструватися у додатку за допомогою електроної скриньки, логіну та паролю. Пароль після заповнення форми реєстрації шифрується за допомогою утиліти argon2, у форматі SHA-256.
	Авторизація	Користувач може авторизуватися у додатку за допомогою електроної скриньки або логіну та паролю.
	Вілнолення паролю	Користувач має можливість відновити пароль у разі втрати доступу до існуючого облікового запису.
	Пегегляд усіх публікацій	Користувач має можливість переглядати всі публікації створені усіма користувачами на головній сторінці веб-додатку.
	Пегегляд публікацій за автором	Користувач має можливість переглядати всі публікації створені користувачем на відповідному профілі.

Кінець таблиці 1.2

Актор	Назва ВВ	Опис ВВ
1	2	3
Авторизований користувач	Оприлюднення публікації	Користувач може створювати нові публікаціх для оприлюднення у додатку.
	Реагування на публікацію	Користувач може залишати реакцію на існуючу публікації опилюденну іншими користувачами
	Редагування публікації	Користувач може редагувати раніше створену публікацію, можна редагувати лише публікації створені авторизованим обліковим записом.
	Видалення публікації	Користувач може видалити раніше створену публікацію, можна видалити лише публікації створені авторизованим обліковим записом.
	Пегегляд публікацій за автором	Користувач має можливість переглядати всі публікації створені користувачем на відповідному профілі.
	Пегегляд усіх публікацій	Користувач має можливість переглядати всі публікації створені усіма користувачами на головній сторінці веб-додатку.
	Деавторизація	Користувач має можливість вийти із облікового запису за допомогою компонетну інтерфейсу користувача.

Відповідно до описаних акторів та варіантів використання побудовано діаграму, яка подана на рисунку 1.5.

Технічне завдання на розробку веб-додатку подано у додатку А.

Таким чином, у розділі було проаналізовано предметну область, з’ясовані причини частого виникнення недовіри у користувачів соціальних мереж, а також досліджені наявні веб-застосунки у вигляді соціальної мережі. В результаті було визначено їх переваги та недоліки, а також доведено потребу у розробці спеціалізованого програмного продукту, який призначатиметься для поліпшеня безпеки даних користувачів та оптимізації роботи веб-додатку на існуючих веб-браузерах. Сформовані вимоги до веб-додатку були описані за допомогою діаграми варіантів використання та формувалися, опираючись на інші додатки.

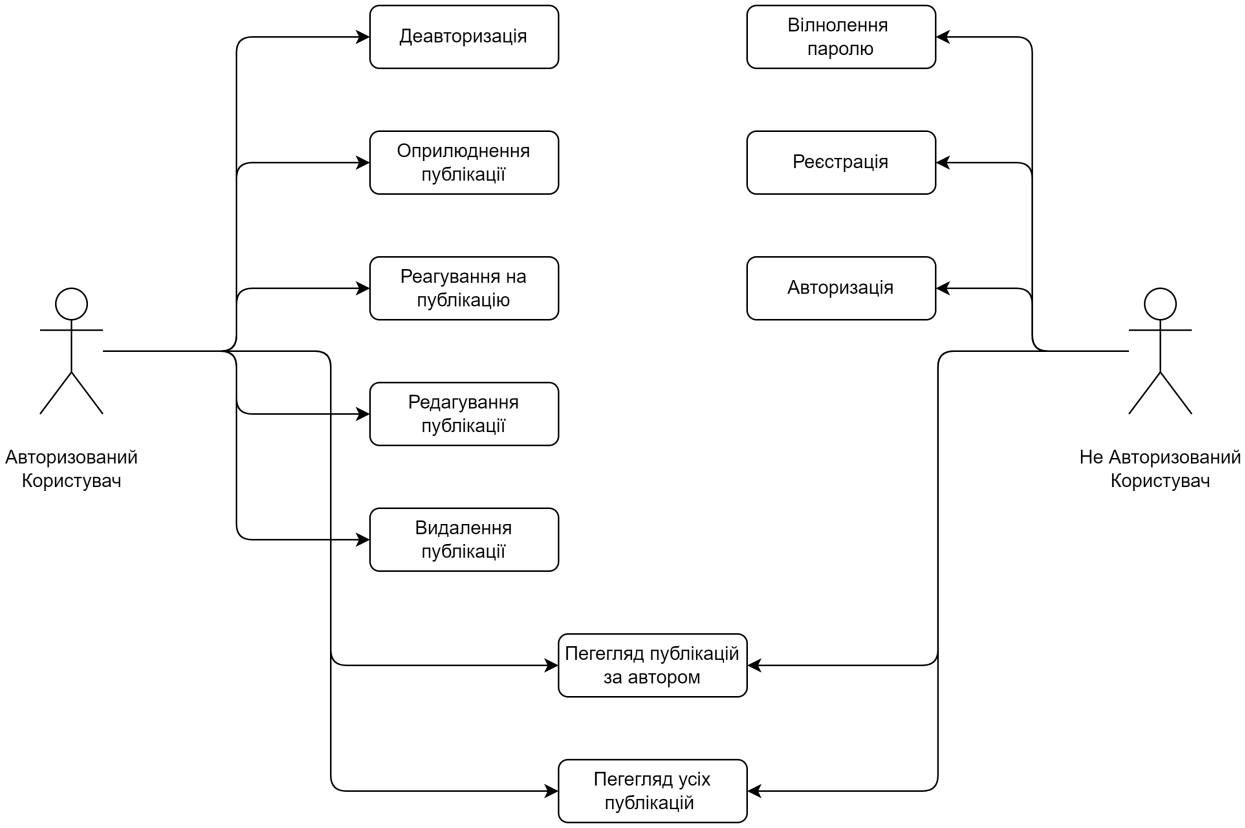


Рисунок 1.5 – Діаграма варіантів використання

2 ПРОЕКТУВАННЯ ВЕБ-ДОДАТКУ

2.1 Архітектура та функціональна структура веб-додатка

Існують різні шаблони архітектури веб-додатків, які дозволяють охопити різні критерії для високопродуктивних рішень. Ми повинні враховувати вимоги користувача, розробника та власника програмного продукту.

Вимоги користувачів, як правило, зосереджені на зручності використання. Він включає час, що витрачається на оновлення інформації на сторінках, можливість перемикатися між сторінками або зберігати посилання та закладки, а також параметри для роботи в автономному режимі.

Вимоги розробника в основному зосереджені на продуктивності, масштабованості та швидкості розробки. Розробники – це ті, хто впроваджує нові функції, реструктуризує код і паралелізує процес розробки програмного забезпечення, вони також можуть мінімізувати час відгуку сервера, збільшити обчислювальну потужність, надати узгоджені та доступні дані.

Архітектура веб-додатків має вирішальне значення для врегулювання мінливих ринкових тенденцій та очікувань користувачів, що змінюються.

Добре продумана архітектура веб-програм може швидко адаптуватися в міру зміни вимог без зайвих збоїв або затримок, забезпечуючи чудовий досвід, який ще більше покращує продуктивність як зараз, так і в осяжному майбутньому.

Архітектура веб-додатків – це план того, як розробити ваш додаток, щоб його можна було масштабувати та підтримувати. Він включає в себе всі компоненти, бази даних, системи проміжного програмного забезпечення, інтерфейси користувача із серверами в додатку – як на стороні клієнта, так і на стороні сервера, як-от стратегії розгортання або системи реєстрації помилок, які допоможуть вам тримати в курсі будь-яких проблем, що виникають у середовищі реального використання, де користувачі можуть зіткнутися з простоєм, якщо вони не будуть виправлені досить швидко.

Архітектуру веб-додатків можна розділити на три основні компоненти: інтерфейс користувача, рівень даних і бізнес-логіка:

– Інтерфейс користувача – це місце, де користувачі взаємодіють з вашим веб-додатком для виконання різних завдань, наприклад створення нового або редагування наявного вмісту;

– Рівень даних відповідає за збереження інформації про інформацію вашої веб-програми, щоб згодом до неї могли отримати доступ інші частини веб-програми;

– Нарешті, бізнес-логіка має справу з керуванням взаємодією даних у веб-додатку.

Існує п’ять важливих типів архітектури веб-додатків, які слід враховувати при розробці власного веб-додатка: MVC, SPA, MVVM і PAC. У цій статті кожен з них буде розглянуто більш детально.

MVC (Model-View-Controller) є однією з найпопулярніших і відомих архітектур для веб-додатків. Ідея MVC полягає в тому, що у вас є три окремі компоненти, які працюють разом, щоб створити цілісну веб-програму: моделі(Model), представлення(View) та контролери(Controller).

Модель(Model) відома як найнижчий рівень, що означає, що вона відповідає за збереження даних. Додавання або отримання даних здійснюється в компоненті моделі. Вона відповідає на запити контролера, оскільки контролер ніколи не спілкується з базою даних сам по собі. Модель спілкується з базою даних, а потім передає необхідні дані контролеру.

Представлення(View) даних здійснюється компонентом перегляду. Він фактично створює для користувача інтерфейс користувача або інтерфейс користувача. Подання створюються за допомогою даних, які збирає компонент моделі, але ці дані беруться не безпосередньо, а через контролер, тому представлення розмовляє лише з контролером.

Контролер(Controller) відомий як основний компонент, оскільки контролер є компонентом, який забезпечує взаємозв’язок між представленням і моделлю, тому він діє як посередник. Контролер не повинен турбуватися про

обробку логіки даних, він просто вказує моделі, що робити. Отримавши дані від моделі, він обробляє їх, а потім бере всю цю інформацію, яку надсилає до представлення та пояснює, як це представити користувачеві.

Тобто, MVC – це архітектура, яка розділяє ваше програмне забезпечення на менші компоненти. Модель має справу з даними та логікою вашої системи. У представленні відображаються лише дані, а контролер підтримує зв'язок між моделлю та представленням. Цей поділ забезпечує читабельність і модульність, а також полегшує частину тестування.

Загальна архітектурна схема MVC показана на рисунку 2.1.

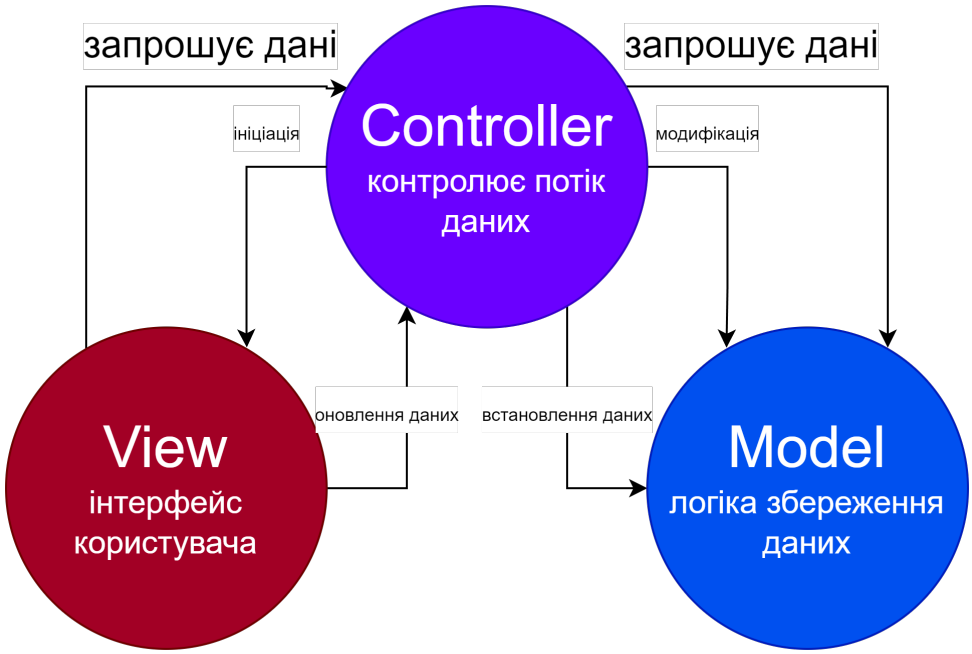


Рисунок 2.1 – Загальна архітектурна схема MVC

SPA (односторінкові програми) робить ідею MVC ще далі, повністю відокремлюючи подання як від контролера, так і від моделі. Це означає, що коли користувач взаємодіє з вашою веб-програмою, представлення завантажується в браузер, і всі подальші взаємодії між користувачем і веб-додатком відбуваються в межах цього одного представлення.

Односторінкові програми працюють у браузері, не вимагаючи перезавантаження сторінки та додаткового часу очікування. SPA самостійно

робить запит до DOM і даних, а потім відтворює сторінку безпосередньо в браузері. Сторінку не потрібно оновлювати, оскільки вміст завантажується автоматично. Односторінкові програми використовують HTML5 і Ajax для створення веб-сторінки. Це забезпечує плавну адаптивну веб-сторінку, а також постійну сторінку, яка перезавантажується самостійно. SPA використовує JavaScript для виконання максимальної кількості обробки на стороні клієнта. Після повного завантаження сторінки кодом HTML і JavaScript усі наступні оновлення виконуються за допомогою викликів AJAX. Після цього дані завантажуються за допомогою виклику API JSON із сервера. Потім ці дані оновлюються на веб-сторінці без перезавантаження.

Загальну схему шаблону SPA показано на рисунку 2.2.

Рисунок 2.2 – Загальна схема шаблону SPA

MVVM (Model-View-ViewModel) – популярна архітектура для веб-додатків. Він дуже схожий на MVC тим, що у вас є три окремі компоненти, але ключова відмінність полягає в тому, що в MVVM ViewModel замінює контролер.

Це означає, що замість контролерів, які обробляють введення користувача та вказують моделям, що робити, у вас є ViewModel, який просто містить дані, необхідні для представлення.

Це може бути корисно для веб-програм, які містять велику кількість даних, оскільки позбавляє від необхідності передавати дані між різними компонентами.

Шаблон MVVM дозволяє прив’язувати елемент перегляду до шаблону перегляду замість того, щоб зв’язувати подання з інтерфейсом. Відповідно, контролер перегляду звільнений від коду взаємодії з моделлю, оскільки тепер відповідальність за це бере на себе модель перегляду. Саме подання має зберігати посилання на шаблон перегляду, але він не знає, де саме воно відображається. Це дозволяє використовувати один і той же шаблон перегляду з різними переглядами.

Модель представлення по суті є обгорткою для тих даних моделі, які підлягають зв’язуванню. Тому модель представлення може спостерігати за змінами в звичайній моделі, а оскільки представлення та модель представлення пов’язані між собою, то всі зміни в моделі автоматично відображатимуться в графічному інтерфейсі користувача. Крім того, ви можете налаштувати прив’язки для виконання в обох напрямках, що означає, що зміни в даних інтерфейсу користувача будуть негайно відображені в шаблоні. Це дозволяє по максимуму використовувати можливості додатка за рахунок гнучкої взаємодії з його користувачами та звільняє від необхідності написання зайвого шаблонного коду. Представлення взаємодіє із своєю моделлю через команди, які також зв’язуються. Загальну схему архітектурного шаблону MVVM показано на рисунку 2.3.

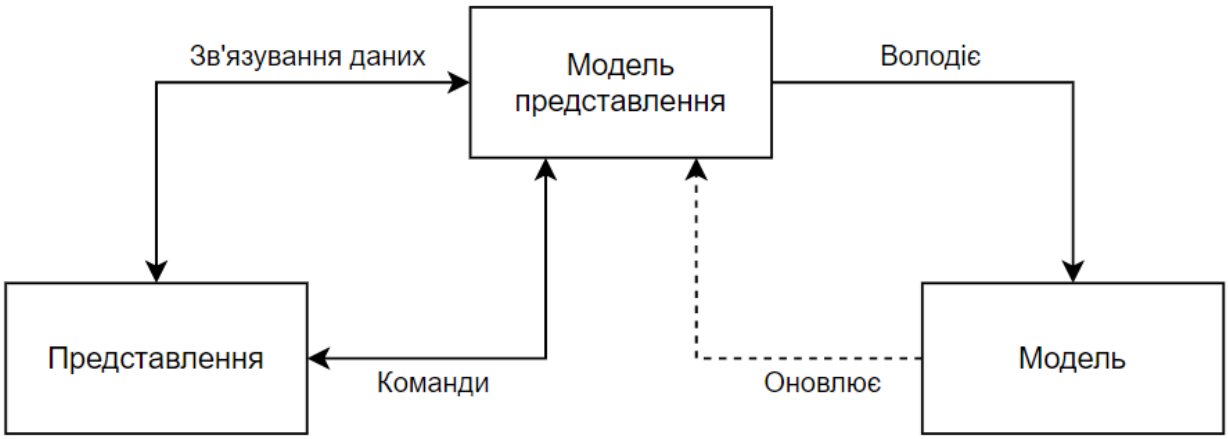


Рисунок 2.3 – Схема шаблону MVVM

PAC (контейнер AJAX на основі прототипів) – це новий тип архітектури веб-додатків, створений Microsoft. Ідея PAC полягає в тому, що це структура дизайну веб-додатків, а не архітектура веб-додатків сама по собі.

Замість того, щоб використовувати конкретні компоненти або дотримуватися встановленого процесу під час створення веб-програми, вам просто потрібно впровадити чотири шаблони AJAX на основі прототипів у будь-яку веб-програму. На риснку 2.3 зображено схему шаблому PAC.

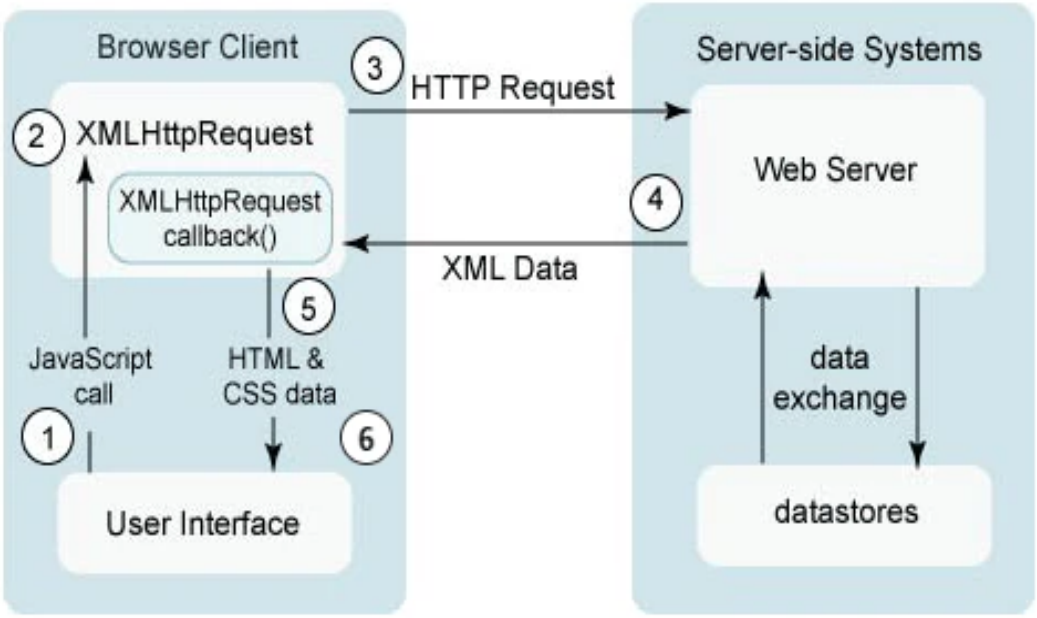


Рисунок 2.3 – Схема шаблону PAC

Веб-додатки зазвичай створюються з використанням однієї з двох архітектур: клієнт-сервер або одноранговий. Клієнт-серверні системи використовують центральний сервер, який керує всіма даними та функціями веб-додатка. Цей тип системи найкраще підходить для програм, яким потрібно керувати великою кількістю даних або вимагають широких заходів безпеки. Однорангові системи використовують розподілену мережу серверів для керування функціональністю та даними програми. Цей тип системи краще підходить для програм, які потрібно швидко й легко оновлювати або які мають великий обсяг трафіку.

Веб-додатки створюються з використанням різних архітектур з багатьох причин. Клієнт-серверні системи дозволяють користувачам отримувати доступ до всіх даних і функцій з будь-якого браузера, але їм потрібні дорогі сервери з високою обробною потужністю для роботи з великими обсягами трафіку або складними заходами безпеки. Однорангові системи розподіляють своє робоче навантаження між кількома комп'ютерами, тому жоден окремий комп'ютер не повинен бути потужним або дорогим, але їх може бути важко оновлювати, і вони можуть бути не такими безпечними. Найкраща архітектура веб-програми для вашої програми залежить від її конкретних потреб.

Розробка веб-додатків – це процес, який постійно змінюється, що вимагає від розробників йти в ногу з новими інструментами та передовими методами. Веб-технології постійно оновлюються, тому для веб-розробників дуже важливо бути в курсі тенденцій у галузі, залишаючись активними в онлайн-спільнотах або відвідуючи місцеві події для розробників.

Архітектура веб-додатків визначає, як буде функціонувати веб-додаток підприємства та які інструменти будуть використовуватися для його створення; стек технологій вирішує, якою мовою буде написаний веб-сайт. Веб-розробники повинні знати обидві ці речі, щоб створити веб-додаток. Отже, глибоке розуміння архітектури веб-додатків має вирішальне значення перед початком розробки.

2.2 Визначення основних модулів додатка

В результаті аналізу функціональної структури додатка, було визначено п’ять модулів, з яких складатиметься веб-додаток “Blanket”:

– модуль даних;
– модуль доступу до даних;
– модуль взаємодії з клієнтською частиною;
– модуль серверної частини;
– модуль клієнтської частини;

Розглянемо кожен з цих модулів детальніше.

Модуль даних буде використовуватися для довготривалого зберігання даних, а також шаблонів програм.

Модуль доступу до даних буде оболонкою для попереднього рівня, яка приховає свою внутрішню продуктивність від коду виклику та забезпечить простий інтерфейс для створення, читання, оновлення та видалення даних програми.

Модуль взаємодії серверної та клієнтських частин відповітами за коректне спілкування між сервером та браузером.

Сервер є базовою системою більшості програм. Існує багато програм, які можуть запускати без внутрішньої системи, їм не потрібна база даних чи веб-сервіс для запуску. Подумайте про простий додаток для будильника. Крім того, вам не потрібен сервер баз даних або веб-сервер, навіть файловий сервер. Вам нема чого зберігати в базі даних або файловому сервері.

Крім перерахованого вище, цей модуль також міститиме логіку шифрування та дешифрування паролю. Шифрування буде виконуватися за допомогою утиліти argon2 та за стандартом шифрування SHA-256. Цей компонент також дозволить вам отримати хеш від пароля (який буде зберігатися в базі даних) і перевірити його.

Але якщо мова йде про додаток, як Blanket, ви суворо прив’язані до серверу та бази даних. Дані, які використовує Blanket, надходять із сервера баз даних, файли, які він використовує, надходять із файлового сервера. Все це називається бекенд.

Розробка фронтенду, також відома як розробка інтерфейсу користувача, відноситься до створення веб-інтерфейсів, частин програми, які користувач бачить і з якими взаємодіє. Розробка фронтенд твердо стоїть на трьох стовпах: HTML для розмітки, CSS для стилів і JavaScript для логіки та взаємодії. Хоча ці три технології витримали випробування часом, основні інструменти та рамки постійно змінюються.

Розробка фронтенду також тісно пов’язана з веб-дизайном та користувацьким досвідом, і межа між ними може відрізнятися в кожному веб-додатку. Діаграму описаних модулів веб-додатку та їх відношень між собою показано на рисунку 2.4.

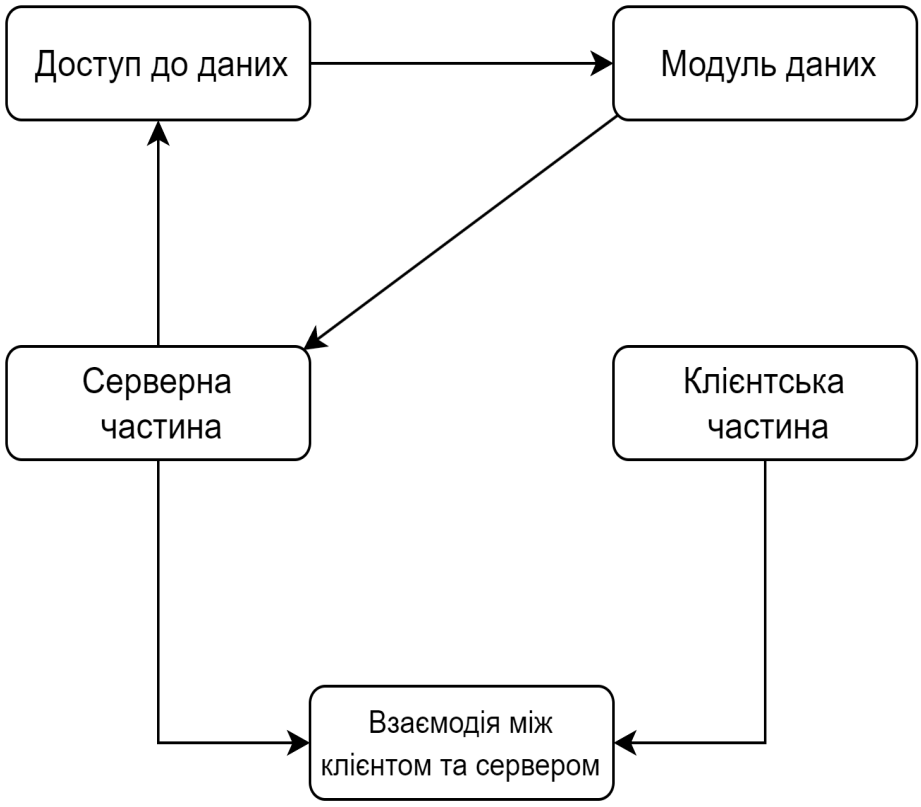


Рисунок 2.4 – Відношення модулів веб-додатку

2.3 Проектування моделі бази даних

Веб-додаток буде використовувати базу даних для збереження інформації про створені публікації. Як було сказано раніше, найкраща система управління даними у рамках цього веб-додатку це – PostgreSQL.

PostgreSQL – це передова система управління об'єктно-реляційною базою даних який підтримує розширену підмножину стандарту SQL, у тому числі транзакції, зовнішні ключі, підзапити, тригери, визначені користувачем типиі функції. Це також одна з найбільш використовуваних баз даних у галузі для веб-розробки. Як реляційна база даних PostgreSQL зберігає дані в таблицях (так звані відносини), що містять кортежі, що представляють сутності (такі як документи та люди) і відносини (наприклад, авторство). Відносини містять атрибути фіксованого типу, що представляють властивості сутності (наприклад, заголовок) разом із первинним ключем. Типи атрибутів можуть бути атомарними (наприклад, ціле число, з плаваючою комою або логічним значенням), або структурованими (наприклад, масив, вкладений JSON або процедура).

Отже, БД матиме наступні сутності:

– Post (містить інформацію про створену публікацію);

– User (зберігає інформацію створених та внесених до бази даних користувачів);

– Upvote (зберігає інформацію про реакції, які будуть закріплені за існуючими публікаціями);

Розглянемо сутності більш детально.

Сутність Post містить наступні атрибути:

– id (основний авто-згенерований ключ);

– title (заголовок публікації);

– text (тексктова частина публікації);

– picture (картинка прикріплена до публікації);

– points (визначає кількість рекцій на публікацію);

– voteStatus (визначає чи конкретний користувач реагував на конкретну публікацію);

– authorId (визначає основний ключ користувача створившого публікацію);

– author (містить повну інформацію про користувача створившого публікацію).

– upvotes (містить повну інформацію про користувачів відреагувавших на публікацію та реакції)

– createdAt (дата створення публікації);

– updatedAt (дата оновлення публікації);

–

Сутність User містить наступні атрибути:

– id (первинний ключ);

– username (ідентифікатор Telegram чату в який було відправлено файл);

– email (ідентифікатор повідомлення, яким було відправлено файл);

– password (розмір документу; не завжди дорівнює розміру оригінального файлу, оскільки він міг бути розділений на декілька частин);

– posts (зовнішній ключ який позначає, що один файл може бути вивантажений в Telegram декількома повідомленнями);

– upvotes (зовнішній ключ, який позначає, що всі частини вихідного файлу посилаються на одну єдину іконку).

– createdAt (дата створення користувача);

– updatedAt (дата оновлення користувача);

Сутність Upvote містить наступні атрибути:

– value (числове значення величини);

– userId (основний ключ відреагувавшого користувача);

– user (містить повну інформацію про користувача відреагувавшого на публікацію);
– postId (основний ключ публікації на яку відреагували);
– post (містить повну інформацію про публікацію на яку відреагували).

Діаграму зв’язків показано на рисунку 2.6.

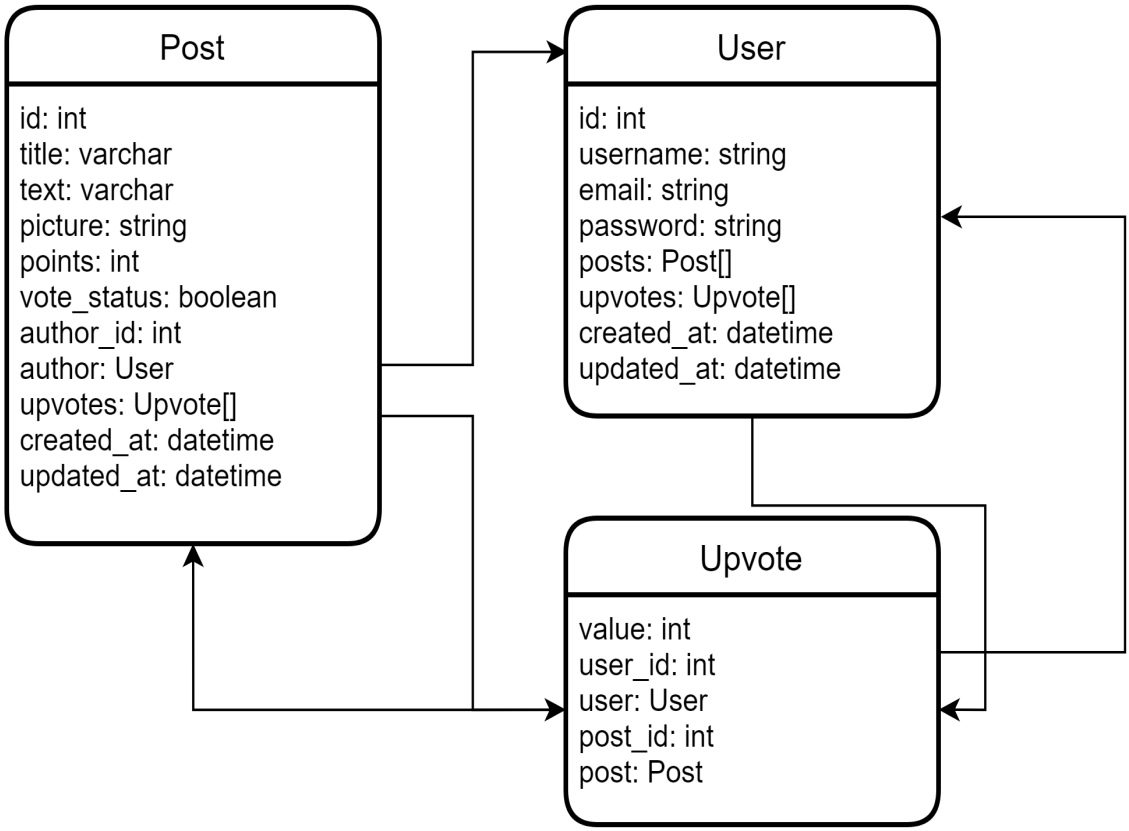


Рисунок 2.6 –Модель Бази Даних

2.4 Проектування інтерфейсу користувача

Для оформлення дизайну додатку було обрано три основних кольори: білий, сірий і зелений. Сам додаток повинен мати легкий та зрозумілий дизайн інтерфейсу. Перш за все, розробимо головну сторінку та навігаційну панель веб-

додатку, яка буде зустрічати як авторизованих, так і не авторизованих користувачів. Фоном було обрано звичайний білий колір, по середині сторінки розміщені мініфіковані карточки раніше створених публікацій. З самого верху сторінки буде розміщена панель навігації, яка в собі містить логотип, який вміщає в собі кнопку переходу на головну сторінку, та посилання на сторінки авторизації та реєстрації, для не авторизованих користувачів (зображено на рисунку 2.7), а для авторизованих буде містити кнопку-посилання на сторінку створення нової публікації, назва профілю, яка містить посилання на профіль авторизованого користувача та кнопка деавторизації (зображено на рисунку 2.8).

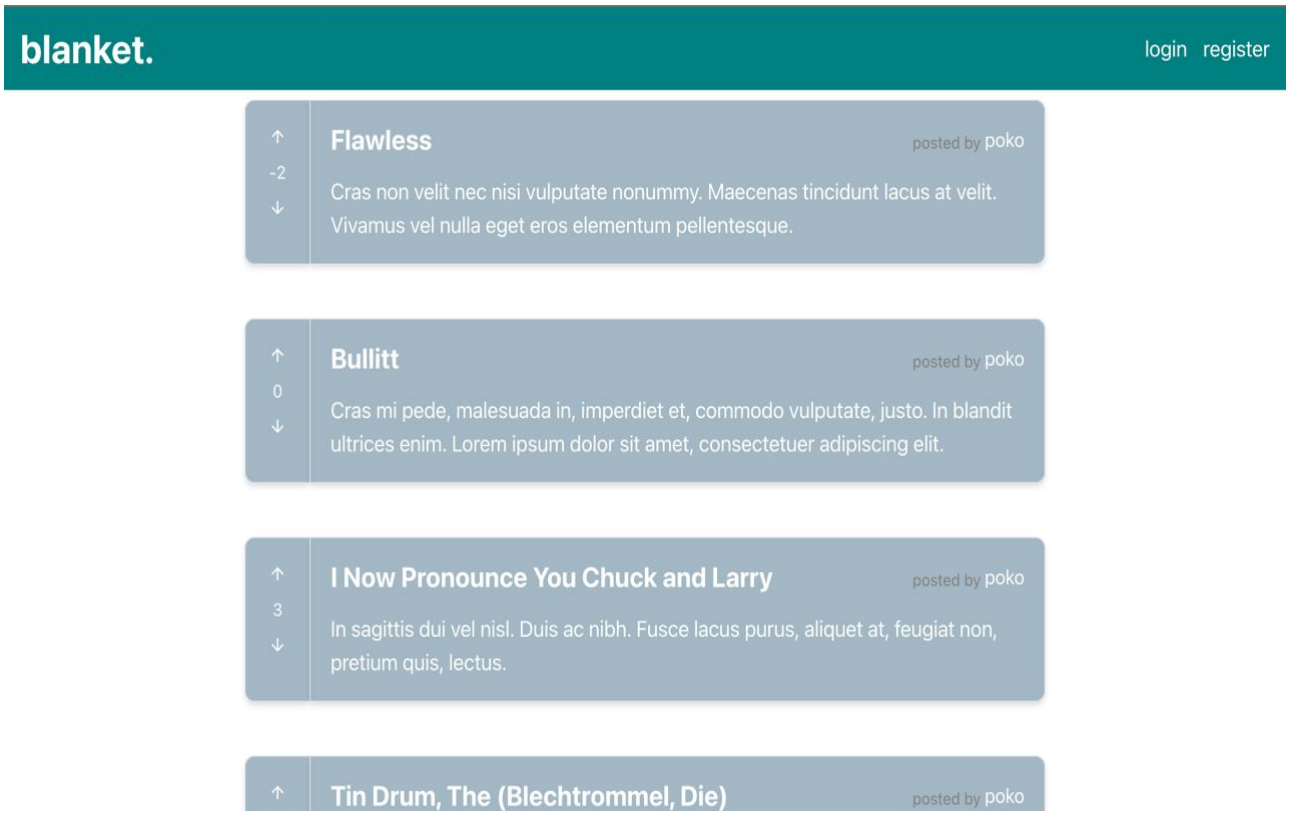


Рисунок 2.7 – Вигляд головної сторінки для не авторизованого користувача

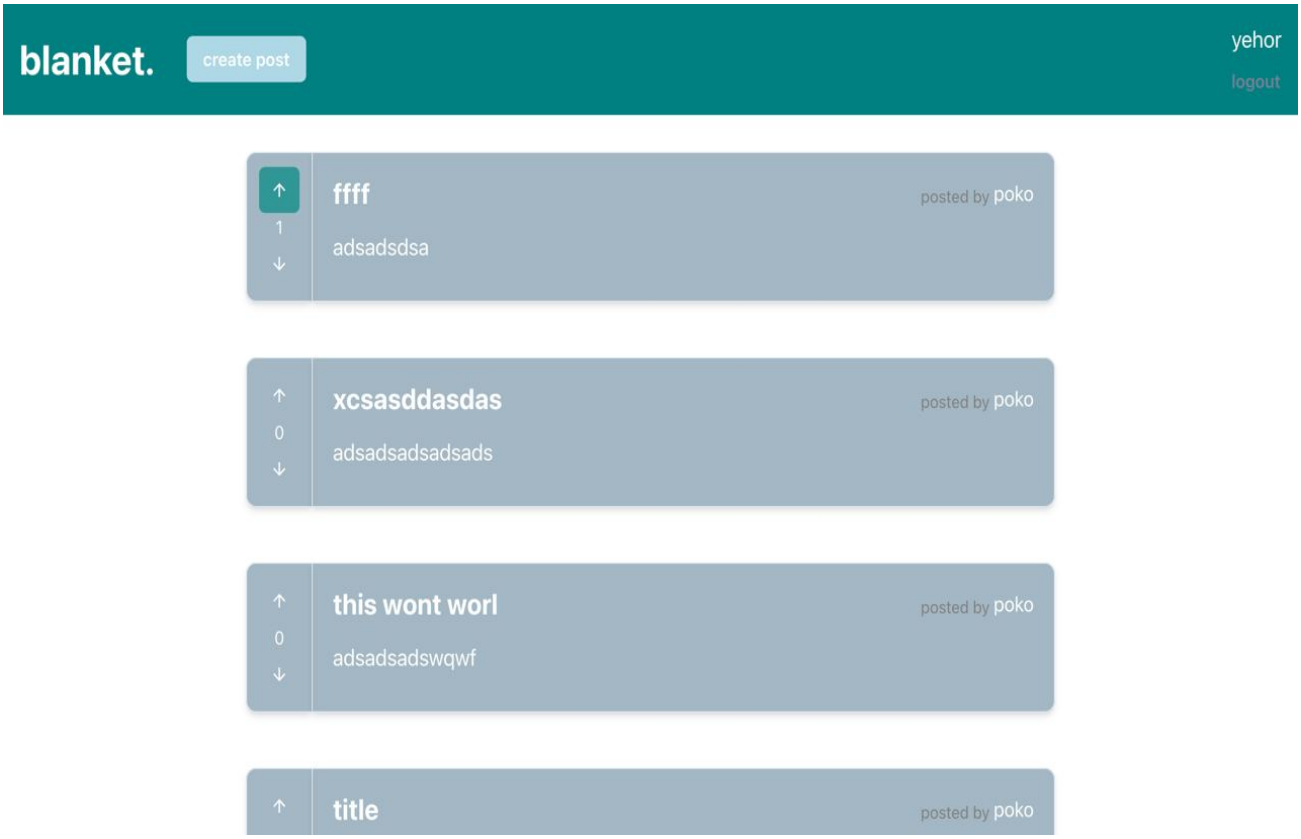


Рисунок 2.8 – Вигляд головної сторінки для авторизованого користувача

Як можна помітити кнопки-посилання для створення нової публікації не існує на навігаційній панелі не авторизованого користувача. Як авторизований так і не авторизований користувачі можуть проглядать раніше оприлюднені публікації, не тільки на головній сторінці а й на окремій сторінці для публікації та на сторінці профілю користувача, але там відображені тільки публікації створені даним користувачем. Натиснувши на будь-яку публікацію, користувача буде переміщений на окрему сторінку публікції, дизайн та вигляд окремої сторінки для публікації показано на рисунку 2.9.

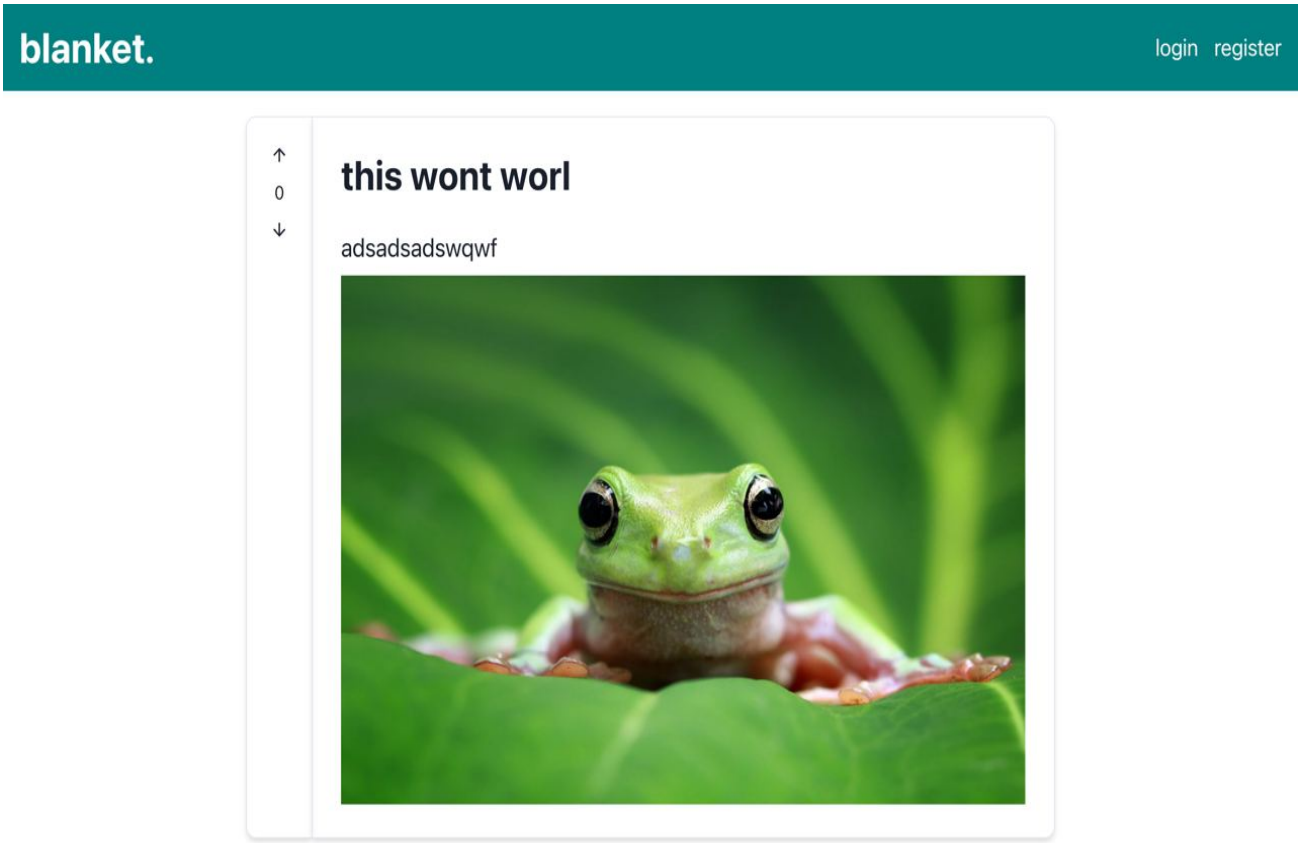


Рисунок 2.9 – Вигляд окремої сторінки для публікації

Далі розглянемо дизайн мініфікованої картки публікації та окремої сторінки публікації. Як можна помітити на вищевказаних рисунках мініфікована картка та окрема сторінка мають різний вигляд та різний текст, рішення рендерити максимум 25 перших слів та не відображати картинку із ціллю поліпшити швидкість додатку та оптимізувати веб-додаток на старих пристроях, або на пристроях із повільним інтернет покриттям. У картці знаходятся контейнер для реакцій на публікацію, назва профілю автора публікації, заголовок та текст. Дизайн мініфікованої картки зображено на рисунок 2.10.

Так саме мініфікована картка буде мати інший вигляд якщо публікація була розміщена в існуючий момент авторизованим користувачем, вона буде мати невеличку панель керування, яка буде автору дозволяти видалити або редагувати дану публікацію. Дизайн мініфікованої картки для автора публікації зображено на рисунок 2.11.

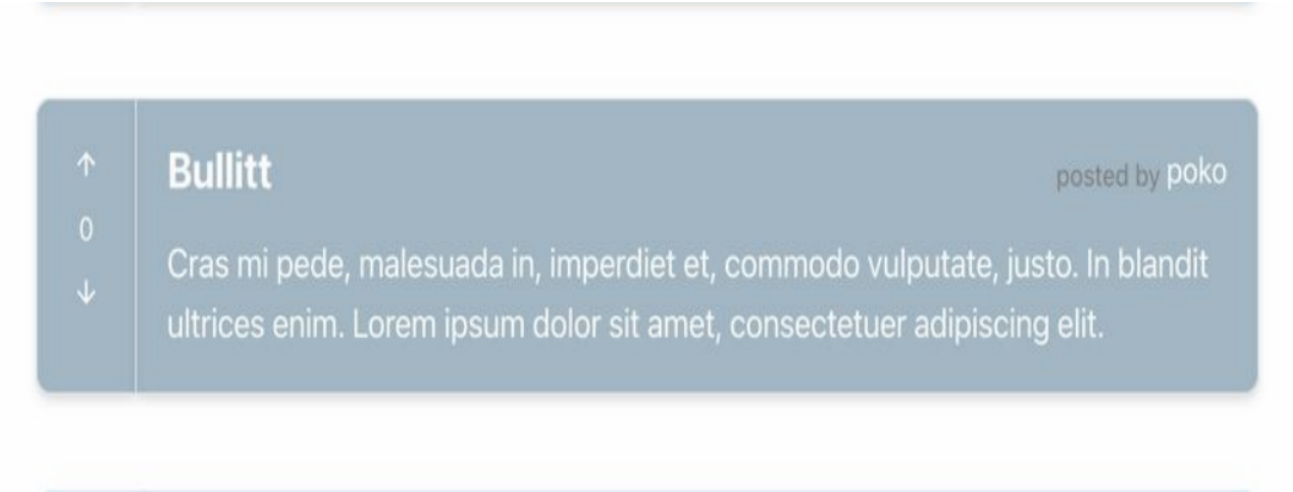


Рисунок 2.10 – Дизайн мініфікованої картки

Рисунок 2.11 –Вигляд мініфікованої картки
для автора публікації

Далі розглянемо вигляд контейнеру мініфікованих карток публікацій на головному екрані. Контейнер буде містити максимум 15 карток після чого підгрузка наступних 15 карток може бути ініційована за допомогою компонента інтерфейсу користувача, це рішення було прийняте задля поліпшення простоти використання та підвищення оптимізації. Дизайн контейнеру та кнопки Load More зображено на рисунку 2.12.

Рисунок 2.12 –Вигляд контейнеру та кнопки Load More

На останок розберемо сторінку профілю користувача. На даній сторінці можна побачити назву профілю, дату створення профілю(реєстрації) та усі створенні даним користувачем публікації, контейнер карток майже такого ж самого вигляду, як і на головній сторінці, та має майже ідентичний функціонал, єдина відмінність це колір самого контейнеру та відсутність імені автора публікації. Дизайн сторінки профілю на рисунку 2.13.

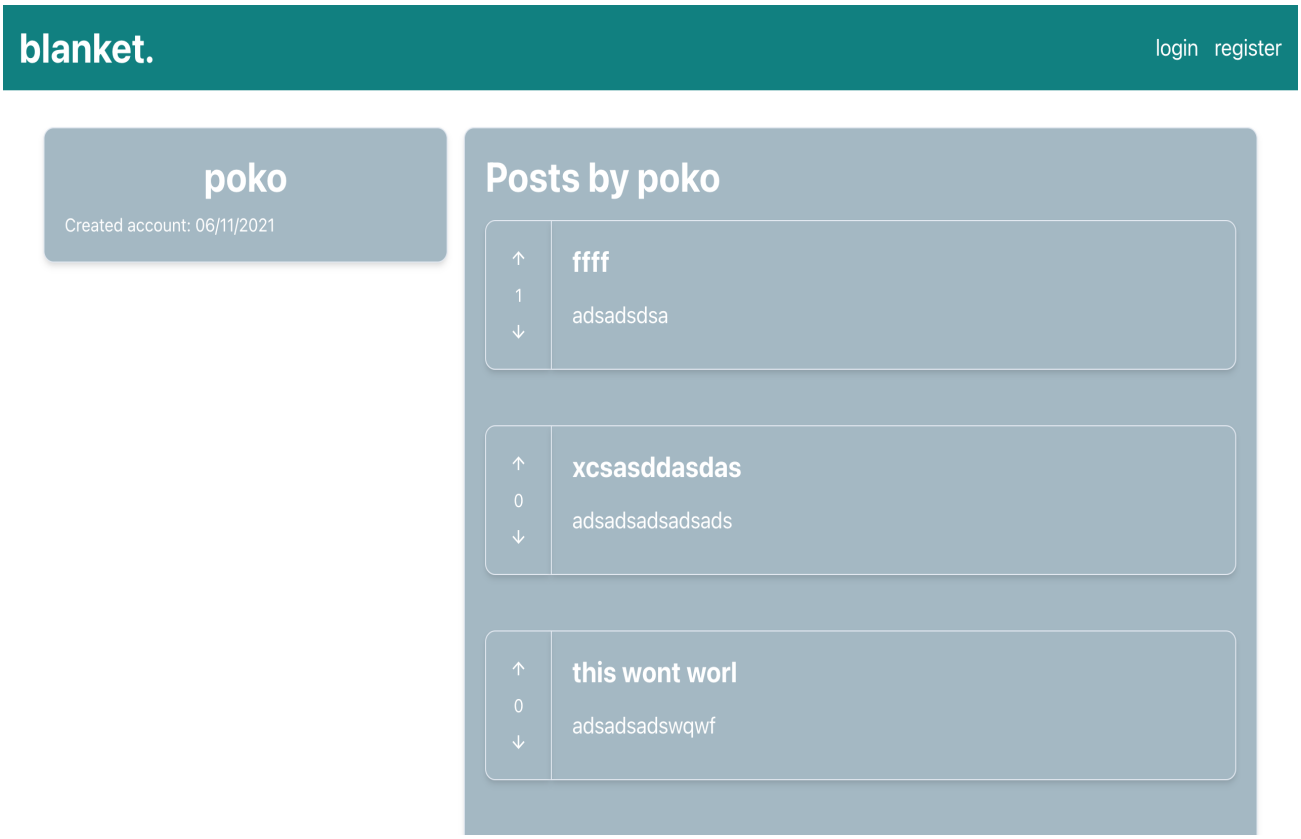


Рисунок 2.13 –Вигляд сторінки профілю

В додатку буде 8 унікальник сторінок, а назви та схему всіх можливих переходів між ними показано на рисунку 2.16.

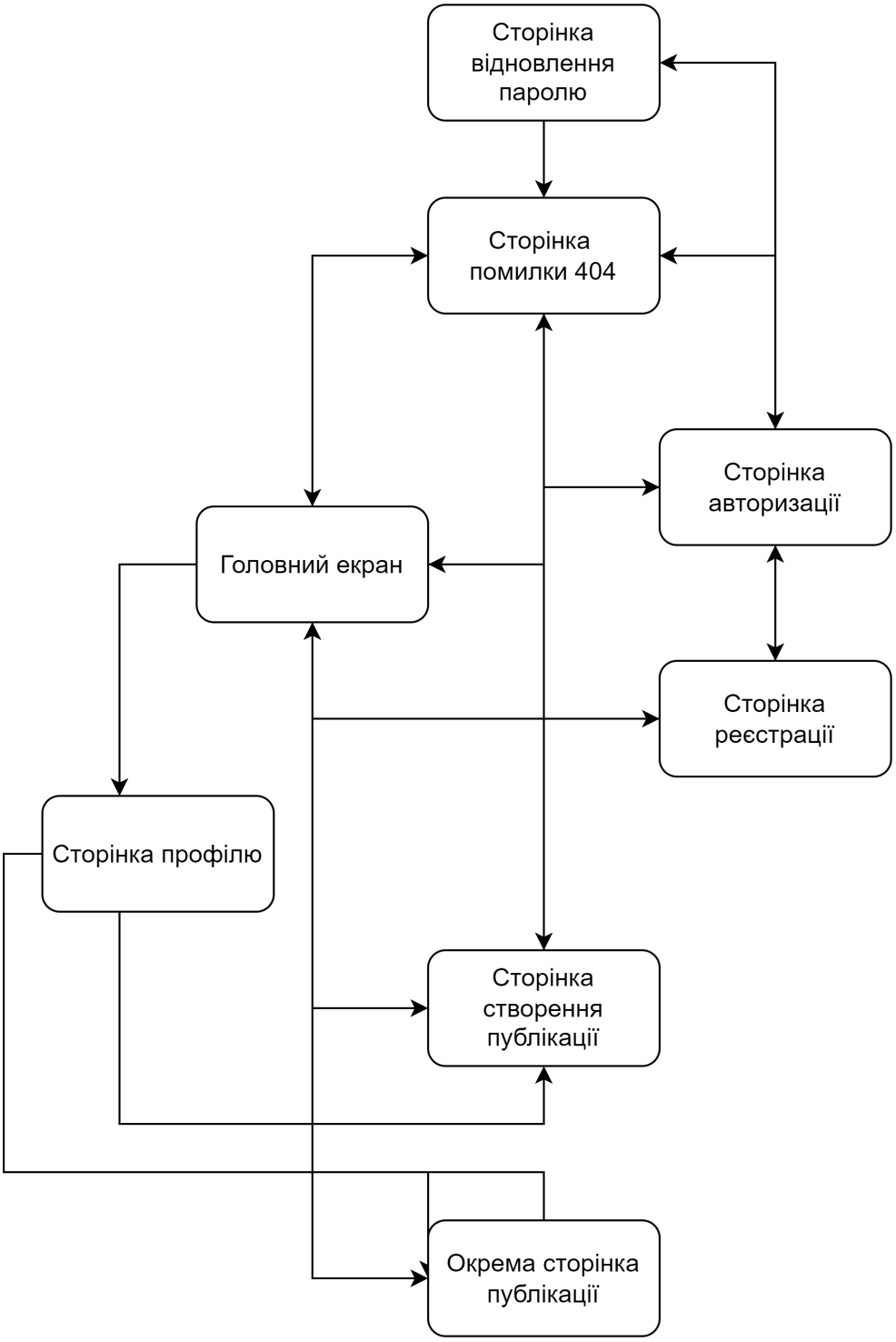


Рисунок 2.16 – Схема переходів між екранами додатку

2.5 Аналіз та вибір технологій і методів реалізації додатка

Як вже зазначалось раніше, додаток розроблятиметься за допомогою мову програмування TypeScript, я на серверній, так і на клієнтській частинах. Задля створення серверної частину буде використовутись фреймворк Node.js, мова запитів GraphQL, фреймворк Express та провідник суворий типізації для мови запитів – Type-GraphQL. Для клієнтської частину будуть використовуватись фреймворки React.js, Next.js та провідник компонетнтів найвищого порядку Formidable Labs Urql.

Для збереження та обробки даних буде використовуватись реляційна база даних PostgreSQL, останьої на момент створення веб-додатку версії(14) оскільки вона найкраща у випадках коли діло йдеться із великими та динамічними дата-сетами.

Для доступу та взаємодії із БД буде використано TypeORM, який дозволяє генерувати нативний SQL код. Окрім цього, завдяки використанню даного фреймворку, відпадає необхідність написання SQL запитів вручну. TypeORM – це структура об’єктно-реляційного відображення (ORM). Загалом, частина об’єкта відноситься до домену/моделі у вашій програмі, реляційна частина відноситься до зв’язку між таблицями в системі керування реляційною базою даних і, нарешті, частина відображення відноситься до акту з’єднання моделі та наших таблиць.

ORM – це тип інструменту, який зіставляє сутності з таблицями бази даних. ORM забезпечує спрощений процес розробки шляхом автоматизації перетворення об'єкта в таблицю і таблицю в об'єкт. Як тільки ви зможете написати свою модель даних в одному місці, оновлювати, підтримувати та повторно використовувати код стає легше.

Оскільки модель слабо зв’язана з рештою програми, ви можете змінити її без жорсткої залежності від іншої частини програми та легко використовувати її в будь-якому місці програми. TypeORM є дуже гнучким, абстрагує систему БД

від програми та дозволяє нам отримати вигоду від використання концепції OOPS.

Я використав бібліотеку Express задля ініціалізації серверної частини веб-додатку. Ключовим аспектом Express є те, що Express є розширюваним. Express забезпечує дуже мінімальну структуру, і ви можете додавати різні частини функціональності Express за потреби, замінюючи те, що не відповідає вашим потребам. Так багато фреймворків дають вам усе, залишаючи перед вами роздутий, загадковий і складний проект, перш ніж ви навіть напишете один рядок коду. Дуже часто першим завданням є витрачання часу на видалення непотрібної функціональності або заміну функціональності, яка не відповідає вимогам. Express використовує протилежний підхід, дозволяючи додавати те, що потрібно, коли це потрібно.

На клієнтській частині було вирішено використовувати, фреймворк для JavaScript – React.js. Все в React є «елементом» або «компонентом». Елементи створюються за допомогою чогось, що називається «JSX» або синтаксичного JavaScript. В основному це схоже на оголошення вмісту HTML як констант, змінних, функцій тощо. JSX не є абсолютно необхідним, але, як правило, є кращим під час кодування програми React. Компоненти схожі на будівельні блоки будь-якої програми React. У них може бути кілька елементів, які запрограмовані на взаємодію та поведінку певним чином. Життєвий цикл компонента React існує для захисту стану компонента. Стан компонента не можна змінювати, поки React малює компонент. Замість цього компонент переходить у відомий стан, малює, а потім відкриває життєвий цикл для ефектів, оновлень стану та подій.

У результаті детального аналізу було визначено та описано основні модулі додатка і те, як вони взаємодіють між собою. Отриману інформацію було продемонстровано за допомогою діаграми пакетів.

Окрім раніше згаданого, було також розроблено основні елементи інтерфейсу та проведено дослідження їх дієспроможності. Список всіх екранів,

а також способи переходів між ними, було продемонстровано за допомогою схеми переходів.

Насамкінець проведений аналіз технологій і засобів, які будуть використовуватись в процесі розробки програмного додатку.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Розробка бази даних

Як було визначено в попередньому розділі, база даних PostgreSQL буде використовуватися для зберігання даних, а TypeORM – як об’єктно-орієнтована технологія доступу до даних. Тобто нам не потрібно буде писати SQL-запити, оскільки ми не будемо працювати безпосередньо з таблицями даних. Ми будемо працювати з сутностями і використовувати TypeScript та GraphQL для створення шаблонів із збережених елементів.

Entity – це клас, який представляє таблицю бази даних, у файлі класу – стовпці таблиці, typeorm використовує декоратори, щоб позначити клас як сутність і зіставляє його з базою даних як таблицю з наданими полями. Кожна сутність має бути позначена декоратором @Entity, щоб повідомити typeorm, що це сутність. Для кожного поля воно має бути позначено за допомогою декоратора @Column, тип стовпця буде відокремлено від типу атрибута за допомогою машинопису або ви можете надати власний тип за допомогою декоратора стовпців. Ви також можете вказати довжину стовпця, а також значення NULL або унікальність.

Перш за все, необхідно створити сутності та моделі. Кожен такий клас в результаті представлятиме окрему таблицю в базі даних. Код класу Post показано нижче.

```
@ObjectType()
@Entity()
export class Post extends BaseEntity {
  @Field()
  @PrimaryGeneratedColumn()
  id!: number;

  @Field()
  @Column()
  title!: string;

  @Field()
  @Column()
```

```
    text!: string;

    @Field({ nullable: true })
    @Column({ nullable: true })
    picture?: string;

    @Field()
    @Column({ type: "int", default: 0 })
    points!: number;

    @Field(() => Int, { nullable: true })
    voteStatus: number | null;

    @Field()
    @Column()
    authorId: number;

    @Field()
    @ManyToOne(() => User, (user) => user.posts)
    author: User;

    @OneToMany(() => Upvote, (upvote) => upvote.post)
    upvotes: Upvote[];

    @Field(() => String)
    @CreateDateColumn()
    createdAt: Date;

    @Field(() => String)
    @UpdateDateColumn()
    updatedAt: Date;
  }
```

Дана модель показує створення сутності та столбців до цієї сутності. У цьому прикладі наглядно відображено можливості реляційної моделі системи керування даними та відношень, за домогою яких ми можемо прив’язати один клас до іншого, без великих зусиль. Рядок з однієї таблиці може мати кілька відповідних рядків в іншій таблиці, а рядок в іншій таблиці також може мати кілька відповідних рядків у першій таблиці, це відношення визначається як відношення багато до багатьох(Many To Many). А один до багатьох(One To Many), рядок з однієї таблиці може мати кілька відповідних рядків в іншій таблиці. Цей тип зв'язку можна створити за допомогою зв'язку первинний ключ-зовнішній ключ.

Далі показано код класу User, який містить інформацію про користувача.

```
@ObjectType()
@Entity()
export class User extends BaseEntity {
  @Field()
  @PrimaryGeneratedColumn()
  id!: number;

  @Field()
  @Column({ unique: true })
  username!: string;

  @Field()
  @Column({ unique: true })
  email!: string;

  @Column()
  password!: string;

  @Field(() => [Post], { nullable: true })
  @OneToMany(() => Post, (post) => post.author)
  posts: Post[];

  @OneToMany(() => Upvote, (upvote) => upvote.user)
  upvotes: Upvote[];

  @Field(() => String)
  @CreateDateColumn()
  createdAt: Date;

  @Field(() => String)
  @UpdateDateColumn()
  updatedAt: Date;
}
```

На додаток до властивостей, які стануть стовпцями однойменної таблиці, даний клас перевизначає оператори приведення типів таким чином, щоби тип User міг бути приведений до типу Post або Upvote.

Нижче показано код класу Upvote, який містить інформацію про усі існуючі реакції на публікації.

```
@ObjectType()
@Entity()
export class Upvote extends BaseEntity {
  @Field()
  @Column({ type: "int" })
  value: number;
```

```
  @Field()
  @PrimaryColumn()
  userId: number;

  @Field(() => User)
  @ManyToOne(() => User, (user) => user.upvotes)
  user: User;

  @Field()
  @PrimaryColumn()
  postId: number;

  @Field(() => Post)
  @ManyToOne(() => Post, (post) => post.upvotes)
  post: Post;
}
```

Після того як всі сутності готові можем перейти до наступного підрозділу де будем опрацювувати логіку взаємодії сервера із нашою базою даних.

3.2 Реалізація серверної частини

Після створення моделей та сутностей для нашої бази даних можемо розпочати із ініціалізації сервера веб-додатку Blanket. Спочатку ми переконаємося, що express, express-graphql і функція buildSchema з пакета graphql імпортуються. Далі ми створюємо просту схему GraphQL за допомогою функції buildSchema.

Щоб створити схему, ми викликаємо функцію та передаємо рядок, який містить код IDL (GraphQL Interface Definition Language), який використовується для опису схеми. Для опису повної системи типів API використовується схема GraphQL. Він включає повний набір даних і визначає, як клієнт може отримати доступ до цих даних. Кожен раз, коли клієнт здійснює виклик API, цей виклик перевіряється схемою. Лише якщо перевірка пройшла успішно, дія виконується. В іншому випадку повертається помилка.

Далі створюється кореневий роздільник. Резолвер містить відображення дій у функції.

Нарешті, сервер express створюється з кінцевою точкою GraphQL: /graphql. Щоб створити кінцеву точку GraphQL, спочатку в додатку зберігається нова інстацнія express. Фрагмент коду відображає основну функцію запуски сервера main, в якому відбувається початкова ініціалізація додатку:

```
const main = async () => {
  const app = express();

  const RedisStore = connectRedis(session);
  const redis = new Redis();
  const apolloServer = new ApolloServer({
    schema: await buildSchema({
      resolvers:[PictureResolver, PostResolver, UserResolver],
      validate: false,
    }),
    context: ({ req, res }) => ({
      req,
      res,
      redis,
      userLoader: createUserLoader(),
      upvoteLoader: createUpvoteLoader(),
    }),
    plugins: [ApolloServerPluginLandingPageGraphQLPlayground({})],
  });

  app.use(graphqlUploadExpress());

  await apolloServer.start();
  apolloServer.applyMiddleware({
    app,
    cors: false,
  });
  app.listen(4000, () => {
    console.log("------------server started at localhost:4000----------");
  });
};
main().catch((err) => {
  console.error(err);
});
```

Метод main() здійснює послідовний виклик методів для початкової ініціалізації додатка, які у своїй внутрішній реалізації викликають методи. Для підключення GraphQL провайдера, який безпосередньо надає можливисть працювати напряму із мовою запитів. Після цього ми ініціалізуємо GraphQLPlayground. Після старту серверу відбувається перевірка на цілісність імпортованих локальних файлів та імпортованих бібліотек.

Після того як ми створили наш сервер можна підключати та ініціалізувати з’єднання із базою даних. Кожна сутність буде автоматично додана до об’єкта

з’єднання typeorm, оскільки вона знаходиться в папці сутності, зазначеній у файлі ormconfig.json, або ініціаціалізована у головному файлі нашого сервера за допомогою спеціального методу createConnection(), в який ми прописуємо усі параметри нашої бази даних. Розгляно другий варіант:

```
const conn = await createConnection({
    type: "postgres",
    database: process.env.DB_NAME,
    username: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    logging: true,
    synchronize: true,
    migrations: [path.join(__dirname, "/migrations/*")],
    entities: [Post, User, Upvote],
  });
```

Як можна помітити більшість параметрів закриті для перегляду, це називається env, env – це змінна середовища, це змінна, значення якої встановлюється за межами програми, як правило, через функціональні можливості, вбудовані в операційну систему або мікросервіс. Змінна середовища складається з пари ім'я/значення, і будь-яке число може бути створено і доступне для посилання в певний момент часу. Змінення та випуск коду програми є відносно складним і збільшує ризик впровадження небажаних побічних ефектів у виробництво. Коли параметри визначаються змінною середовища замість коду, процес зміни складається з перевірки дійсності нових параметрів та змінних, оновлення відповідної змінної середовища за допомогою команди операційної системи або оновлення файлу конфігурації.

Ось як виглядають env змінні для бази даних:

```
DB_NAME = blanket
DB_USER = postgres
DB_PASSWORD = postgres
```

Після того як база даних була підключена до серверу, можна розпочинати створювати логіку нашого додатку. В першу чергу почнемо зі створення

методів для сутності User, всі методи створені відповідно до технічного завдання. Для створення нових методів нам потребується три утилітарних класа, які існують задля простоти сприяти коду:

```
  @ObjectType()
class FieldError {
  @Field()
  field: string;
  @Field()
  message: string;
}
```

FieldError – це клас який описує структури гіпотетичної помилки, яка може виникнути у хожі розробки додатку.

```
@ObjectType()
class UserResponse {
  @Field(() => [FieldError], { nullable: true })
  errors?: FieldError[];

  @Field(() => User, { nullable: true })
  user?: User;
}
```

UserResponse – це клас який описує структури та тип відповіді сервера н відправлений йому запит який запитує користувача.

```
@InputType()
export class UsernamePasswordInput {
  @Field()
  email: string;
  @Field()
  username: string;
  @Field()
  password: string;
}
```

UsernamePasswordInput – це клас який типізую вхідні дані необхідні для створення нового користувача.

Після цього можна приступати до створення методу реєстрації нового користувача. Ми отримуємо вхідні дані від користувачу, шифруєм пароль та

записуєм нового користувача у таблицю бази даних User, за допомогою

методу createQueryBuilder(), та після реєстації ми записуємо згенероване cookies у сховище сессії у браузері користувача. Для того щоб ми могли користуватись cookies, потрібно щоб хостинг знаходився за захищеним SSL-сертифікатом доменом, та ініціалізувати метод запису cookies у головну файлі index.ts, та підключити додаткова базу даних Redis, фрагмент коду відповідний до ініціалізації cookies:

```
app.use(
    session({
      name: COOKIE_NAME,
      store: new RedisStore({
        client: redis as any,
        disableTouch: true,
      }),
      cookie: {
        maxAge: 1000 * 60 * 60 * 24 * 365 * 10, //10 years
        httpOnly: true,
        sameSite: "lax", //csrf
        secure: __prod__, //cookie only work in https
      },
      secret:
"bla12312eadsdiuh12i3u1asdiHAnSDuiasdjhqwouieh12asd6969ket",
      resave: false,
      saveUninitialized: false,
    })
  );
```

Фрагмент коду відображає метод необхідний для створення нового користувача у системі:

```
@Mutation(() => UserResponse)
  async register(
    @Arg("options") options: UsernamePasswordInput,
    @Ctx() { req }: MyContext
  ): Promise<UserResponse> {
    const ..........  hashedPassword= await argor2.hash(options.password);

    const errors = validateRegister(options);
    if (errors) {
      return { errors };
    }
    let user;
    try {
      const result = await getConnection()
        .createQueryBuilder()
        .insert()
        .into(User)
```

```
          .values({
            username: options.username,
            email: options.email,
            password: hashedPassword,
          })
          .returning("*")
          .execute();

        user = result.raw[0];
      } catch (err) {
        //duplicate user error
        if (err.code === "23505") {
          return {
            errors: [
              {
                field: "username",
                message: "username already taken",
              },
            ],
          };
        }
      }
      req.session.userId = user.id;
      return { user };
    }
```

Одразу після реєстрації необхідно створити метод для авторизації уже записаного у базу даних користувача. Створимо новий метод login(), який буде очікувати електрону скриньку або юзернейм та пароль. Необхідно також не забути дешифрувати раніше зашифрований пароль. Фрагмент коду відображає метод необхідний для створення нового користувача у системі:

```
@Mutation(() => UserResponse)
  async login(
    @Arg("usernameOrEmail") usernameOrEmail: string,
    @Arg("password") password: string,
    @Ctx() { req }: MyContext
  ): Promise<UserResponse> {
    const user = await User.findOne(
      usernameOrEmail.includes("@")
        ? { where: { email: usernameOrEmail } }
        : { where: { username: usernameOrEmail } }
    );
    if (!user) {
      return {
        errors: [
          {
            field: "usernameOrEmail",
            message: "no user with this username",
          },
        ],
      };
    }
    const valid = await argor2.verify(user.password, password);
    if (!valid) {
      return {
```

```
        errors: [
          {
            field: "password",
            message: "incorrect password",
          },
        ],
      };
    }

    req.session.userId = user.id;

    return { user };
  }
```

Відповідно до двох вищевказаних методів необхідно створити метод, який надасть змогу авторизованому користувачу вийти із облікового запису. Все що треба зробити у наступному методі – це отримати токен cookies з клієнтської частини та очистити сховище сессії у браузері користувача. Фрагмент коду відображає метод необхідний для деавторизації користувача із системи:

```
@Mutation(() => Boolean)
  logout(@Ctx() { req, res }: MyContext) {
    return new Promise((resolve) =>
      req.session.destroy((err) => {
        res.clearCookie(COOKIE_NAME);
        if (err) {
          // console.log(err);
          resolve(false);
          return;
        }

        resolve(true);
      })
    );
  }
```

Далі розглянемо не менш важливі методи для роботи із сутністю Post, всі методи створені відповідно до технічного завдання. Для створення нових методів нам потребується два утилітарних класа та один утилітарний метод, які існують окремо задля простоти сприяти коду:

```
@InputType()
class PostInput {
  @Field()
  title: string;
  @Field()
```

```
  text: string;
}
```

PostInput – це клас який описує очікуєму структури вхідних даних, необхідних для створення нової публікації.

```
@ObjectType()
class PaginatedPosts {
  @Field(() => [Post])
  posts: Post[];
  @Field()
  hasMore: boolean;
}
```

PaginatedPosts – це клас який описує структуру пагінації, необхідної для виведення всіх публікацій на головній сторінці, не провокуючи довге завантаження сторінки.

```
export const isAuth: MiddlewareFn<MyContext> = ({ context }, next) => {
  if (!context.req.session.userId) {
    throw new Error("You're not authinticated");
  }

  return next();
};
```

isAuth() – це функція яка перевіріє чи запит прийшов від авторизованого користувача.

Спочатку ініціалізуємо методи для створення та редагування публікацій. Для створення нового посту згідно із технічним завданням необхідно щоб користувач був авторизований, а для редагування та видалення щоб активний користувач створив публікації сам, тому почнем з фрагменту який відображає метод створення публікації:

```
@Mutation(() => Post)
  @UseMiddleware(isAuth)
  async createPost(
    @Arg("input") input: PostInput,
    @Arg("img", { nullable: true }) img: string,
    @Ctx() { req }: MyContext
  ): Promise<Post> {
    return Post.create({
```

```
      ...input,
      authorId: req.session.userId,
      picture: img,
    }).save();
  }
```

Все що ми робимо у даному методі – це берем спочатку перевіряєм чи є користувач надіславший запит є авторизований, а вже потім берем вхідні дані і записуємо їх у відповідну таблицю в базі даних.

Далі розберем фрагмент відповідаючий ща редагування публікації:

```
 @Mutation(() => Post, { nullable: true })
@UseMiddleware(isAuth)
async updatePost(
  @Arg("title") title: string,
  @Arg("text") text: string,
  @Arg("id", () => Int) id: number,
  @Ctx() { req }: MyContext
): Promise<Post | null> {
  const result = await getConnection()
    .createQueryBuilder()
    .update(Post)
    .set({ title, text })
    .where('id = :id and "authorId" = :authorId', {
      id,
      authorId: req.session.userId,
    })
    .returning("*")
    .execute();

  return result.raw[0];
}
```

Цей метод не сильно різниться від попередньго, ми отримуємо id публікації, нові вхідні дані і за допмогою методу update(),оновлюємо існуючі записи у базі даних.

Якщо розбиття на сторінки зі зміщенням є масивом, то розбиття сторінки за курсором – це зв’язаний список. Якщо зміщення захоплює записи на основі їх розташування в таблиці, як індекс, курсори використовують вказівник, який вказує на певний запис, і захоплює записи, наступні за цим конкретним записом.

Далі розберемо метод який виводить та ввідає на клієнт пагіновані публікації:

```
@Query(() => PaginatedPosts)
  async posts(
    @Arg("limit", () => Int) limit: number,
    @Arg("cursor", () => String, { nullable: true }) cursor:
string,
    @Arg("id", () => Int, { nullable: true }) id: number,
    @Ctx() { req }: MyContext
  ): Promise<PaginatedPosts> {
    const realLimit = Math.min(50, limit);
    const realLimitPlus = realLimit + 1;
    const replacements: any[] = [realLimitPlus];
    if (req.session.userId) {
      replacements.push(req.session.userId);
    }
    let cursorIdx = 3;
    if (cursor) {
      replacements.push(new Date(parseInt(cursor)));
      cursorIdx = replacements.length;
    }
    if (id) {
      replacements.push(id);
    }
    const posts = await getConnection().query(
      `
    select p.*,
      ${
        req.session.userId
          ? '(select value from upvote where "userId" = $2 and
"postId" = p.id) "voteStatus"'
          : 'null as "voteStatus"'
      }
    from post p
    ${cursor ? `where p."createdAt" < $${cursorIdx}` : ""}
    ${id ? `where p."authorId" = $3` : ""}
    order by p."createdAt" DESC
    limit $1
   `,
      replacements
    );
    return {
      posts: posts.slice(0, realLimit),
      hasMore: posts.length === realLimitPlus,
    };
  }
```

У цьому розділі було створені, задокументовані та описані основні методи для коректної та повноціної роботи веб-додатку згідно із технічним завдання повний код основних методів із серверної частину веб-додатку Blanket можна переглянути у додатку Б

3.3 Реалізація клієнтської частини

Для реалізації клієнтської частини та користувацького інтерфейсу була використана бібліотека React.js. React.js – це набір інструментів для створення користувацьких інтерфейсів, який вперше було розгорнуто Facebook у 2011 році. По суті, React – це рішення проблеми, з якою стикалися розробники під час створення інтерфейсів користувача. Це дозволяє розробникам створювати складні користувацькі інтерфейси з компонентами, які з часом будуть часто змінюватися, без необхідності писати багато складного коду JavaScript.

Хоча React дуже схожий на інтерфейсний фреймворк, насправді він трохи відрізняється з точки зору функціональності. Технічно це бібліотека інтерфейсу користувача. Він включає деякі з тих самих аспектів інтерфейсних фреймворків, але його мета – організувати елементи HTML у компоненти. Замість звичної HTML-розмітки, React.js використуває унікальну розмітку веб-сторінки – JSX.

Блок основної логіки головної сторінки веб-додатку:

```
const [variables, setVariables] = useState({
    limit: 15,
    cursor: null as string | null,
  });

  const [{ data, error, fetching }] = usePostsQuery({
    variables,
  });
  if (!fetching && !data) {
    return <div>{error?.message}</div>;
  }
```

У цьому фрагменті викликаються методи необхідні для запиту усіх раніше створених та збережених у базі даних публікацій:

```
const [variables, setVariables] = useState({
    limit: 15,
    cursor: null as string | null,
  });
  const [{ data, error, fetching }] = usePostsQuery({
    variables,
  });
  if (!fetching && !data) {
    return <div>{error?.message}</div>;
  }
  return (
    <Layout />
  )
```

Фрагмент блоку логіки сторінки створення нової публікації:

```
const router = useRouter();
  useIsAuth();
  const [, createPost] = useCreatePostMutation();
  const [picture, setPicture] = useState(null);
  const [uploadedFile, setUploadedFile] = useState({
    public_id: "",
  });
  const onDrop = useCallback((acceptedFiles) => {
    const url = "https://api.cloudinary.com/v1_1/sreddit-yehor/upload";

    acceptedFiles.forEach(async (acceptedFile: any) => {
      const formData = new FormData();
      formData.append("file", acceptedFile);
      formData.append("upload_preset", "nzxecqni");

      const response = await fetch(url, {
        method: "post",
        body: formData,
      });
      const data = await response.json();
      setPicture(data);
      // console.log(data.public_id);
      setUploadedFile({ public_id: data.public_id });
    });
  }, []);

  const { getRootProps, getInputProps, isDragActive } = useDropzone({
    onDrop,
    multiple: false,
    accept: "image/*",
  });
```

Після створення розмітки освновних сторінок, можемо розпочати створення та ініціалізації клієнта, який буде виступати головним методом взаємодії між веб-браузером та серверною частиною нашого веб-додатку. Для цього використаємо дві бібліотеки – Formidable Labs URQL для створення запитів до нашого серверу та GraphQL Code Generator задля ініціалізації мосту між серверною частиною та веб-браузером. URQL – це універсальний клієнт GraphQL, за допомогою якого ви додаєте такі функції, як нормалізоване кешування, який можна налаштувати у міру зростання вашої програми. URQL можна розуміти як сукупність пов’язаних частин і пакетів. Тоді ми зможемо декларативно надсилати запити GraphQL до нашого API.

GraphQL Code Generator – це інструмент, який згенерує код із нашої схеми GraphQL. Аналізуючи схему та документи, GraphQL Code Generator виводить код у широкому діапазоні форматів, на основі попередньо визначених шаблонів або на основі шаблонів визначених користувачем.

Розглянемо фрагмент коду ініціалізації методу createUrqlClient(), в який для початку ми внесемо базовий шлях до нашого серверу, а пізніше будемо модифікувати задля задоволення раніше визначеного технічного завдання:

```
    export const createUrqlClient = (ssrExchange: any, ctx: any)
=> ({
       url: "http://localhost:4000/graphql",
    )}
```

Далі, модифікуємо цей клієнт так, щоб після авторизації користувач, навіть після тривалого періоду часу не мав потреби повторно авторизуватись. Для цього нам потрібно буде створити метод fetchOptions() який буде дозволяти використання cookies та сховище сесії у нашому веб-додатку, фрагмент коду відображає модифіковану версію методу createUrqlClient(), методом сприймання cookies:

```
    export const createUrqlClient = (ssrExchange: any, ctx: any)
=> ({
```

```
      url: "http://localhost:4000/graphql",
    fetchOptions: () => ({
        credentials: "include" as const,
        headers: {
          cookie: ctx?.req?.headers?.cookie,
        },
    )}
```

Одразу після першої модифікації нашого ініціаційного методу createUrqlClient() проведено невелике ручне тестування нашого додатку та помітимо, що пристворені, редагувані та видалені публікації інформації на екрані змінюється не одразу, а тільки після перезавантаження сторінки. Ця поведінка не є компетентною та додаток не відповідає на даний момент раніше узгодженому технічному завданню. Задля того, щоб привести додаток до відповідної поведінки, необхідно буде провести ще декілка модифікацій createUrqlClient(). Така поведінка обумовлена тим що у веб-браузері існує кеш, який зберігає початковий стан раніше відрендерених компонентів, щоб маніпулювати цей кеш напряму, потрібно створити методи: invalidateAllPosts() та invalidateAllFiltPosts(), розглянемо фрагмент коду відображаючий ці два методи :

```
    const invalidateAllPosts = (cache: Cache) => {
      const allFields = cache.inspectFields("Query");
      const fieldInfos = allFields.filter((info) => info.fieldName
=== "posts");
      fieldInfos.forEach((fi) => {
        cache.invalidate("Query", "posts", fi.arguments);
      });
    };
    const invalidateAllFiltPosts = (cache: Cache) => {
      const allFields = cache.inspectFields("Query");
      const fieldInfos = allFields.filter(
        (info) => info.fieldName === "filteredPosts"
      );
      fieldInfos.forEach((fi) => {
        cache.invalidate("Query", "filteredPosts", fi.arguments);
      });
    };
```

Після створення та запровадження цих методів, можна помітити, що наш веб-додаток має компетентну поведінки при проведені маліпуляцій із публікаціями.

Тепер після доведення нашого веб-додатку до задовільного стану, можна створити утилірарну особливість – пагінацію. Пагінація – це розділення інформації на так звані сторінки, є безліч існуючих варіантів запровадження пагінації у клієнтській частині веб-додатків, але я вирішив запровадити – курсорну пагінацію, або пагінацію за курсором. Курсорна пагінація – відрізняється від всім звичної пагінації за сторінками, тому що видимих кінцевому користувача сторінок не існує, пагінація даного типу частіше за все реалізується за допомогою кпонки “Завантажити ще”. Для того щоб імплементувати даний вид пагінації потрібно створити метод cursorPagination(), який приймає перші 15 публікацій з нашого серверу, і зміну яка буде показувати чи існують ще об’єкти після завантаження наступної сторінки. Розглянемо фрагмент коду відображаючий створення та імплементацію пагінації за курсором :

```
export const cursorPagination = (): Resolver => {
  return (_parent, fieldArgs, cache, info) => {
    const { parentKey: entityKey, fieldName } = info;

    const allFields = cache.inspectFields(entityKey);
    const fieldInfos = allFields.filter((info) => info.fieldName === fieldName);
    const size = fieldInfos.length;
    if (size === 0) {
      return undefined;
    }
    const fieldKey = `${fieldName}(${stringifyVariables(fieldArgs)})`;
    const isItInTheCache = cache.resolve(
      cache.resolve(entityKey, fieldKey) as string,
      "posts"
    );

    info.partial = !isItInTheCache;
    let hasMore = true;
    let results: string[] = [];
    fieldInfos.forEach((fi) => {
      const key = cache.resolve(entityKey, fi.fieldKey) as string;
      const data = cache.resolve(key, "posts") as string[];
      const _hasMore = cache.resolve(key, "hasMore");
      console.log("data:", hasMore, data);
      if (!_hasMore) {
        hasMore = _hasMore as boolean;
```

```
      }

      results.push(...data);
    });

    return {
      __typename: "PaginatedPosts",
      hasMore,
      posts: results,
    };
  };
};
```

3.4 Технічні характеристики веб-додатку

Для успішного використання та стабільної роботи, пристрій користувача повинен мати підключення до Інтернету та відповідати таким мінімальним системним вимогам:

– Будь-який сучасний веб-браузер;
– двохядровий процесор;
– 1000 МБ оперативної пам’яті;
– Частота процесора – 1.9 ггерц.

Рекомендовані вимоги:

– Будь-який сучасний веб-браузер;
– чотирьохядровий процесор;
– 1500 МБ оперативної пам’яті;
– Частота процесора – 3.1 ггерц.

Отже, в процесі написання даного розділу було реалізовано та здійснено опис бібліотеки серверної частини веб-додатку Blanket та клієнтської частини веб-додатку. Розроблений в результаті додаток задовольняє усі вимоги, які були поставлені до системи раніше. Окрім того, було написане керівництво користувача та складені вимоги до технічних характеристик та системних вимог, які дозволили б комфортно користуватися цим веб-додатком.

4 ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКА

4.1 Аналіз методів тестування веб-додатку

У сучасному світі технологій повністю домінують машини, і їх поведінка контролюється програмним забезпеченням, що забезпечує його живлення. Забезпечення якості або тестування програмного забезпечення має вирішальне значення, оскільки воно визначає помилки системи на самому початку.

Тестування програмного забезпечення – це процес оцінки функціональності програмного додатка для виявлення будь-яких програмних помилок. Він перевіряє, чи відповідає розроблене програмне забезпечення зазначеним вимогам, і визначає будь-які дефекти в програмному забезпеченні, щоб створити якісний продукт. Тестування є невід'ємною частиною будь-якого успішного програмного проекту. Типи тестування програмного забезпечення залежать від різних факторів, включаючи вимоги до проекту, бюджет, терміни, знання та придатність. Різні типи тестування програмного забезпечення є ключовою роллю, де тестувальник визначає правильне тестування для програм. Функціональне та нефункціональне тестування – це два види тестування, які виконує тестувальник програмного забезпечення.

Для тестування цієї веб-системи я обрав я використав практики моудульного тестування. Модульне тестування – це процес розробки програмного забезпечення, в якому найменші частини програми, які можна перевірити, так звані блоки, окремо та незалежно перевіряються на належну роботу. Основна мета модульного тестування – виділити написаний код для перевірки та визначити, чи працює він належним чином.

Окрім того, для модульного тестуванна коду TypeScript, існує бібліотека Jest, яку я обрав у якості фреймворку модульного тестування веб-системи "Blanket".

4.2 Тестування веб-додатку

Почнемо тестування додатка із розробки модульних тестів для серверної частини веб-додатку. Тестові сценарії показано у таблиці 4.1.

Таблиця 4.1 – Тестові сценарії серверної частини “Blanket”.

Назва сценарію	Модуль	Метод	Вхідні дані	Очікуваний результат
1	2	3	4	5
Реагування на публікацію	PostResolver	vote	Унікальний ідентифікатор публікації та числове значення яке відображає реакцію	Публікацію із зміненим полем реакцій
Створення нової публікації	PostResolver	createPost	Текстове значення відповідне до заголовку, текстове значення відповідне до тексту та необов’язкове текстове значення відповідне до посилання на картинку	Нову публікацію із відповідними вхідним даним полями
Редагування існуючої публікації	PostResolver	updatePost	Текстове значення відповідне до заголовку, текстове значення відповідне до тексту та унікальний ідентифікатор публікації	Змінену існуючу публікацію із відповідними вхідним даним полями
Видалення існуючої публікації	PostResolver	deletePost	Унікальний ідентифікатор публікації	Значення true, яке свідчить про вдале виконання операції
Авторизація користувача	UserResolver	login	Текстове значення відповідне до паролю або унікального ім’я та пароль	Згенероване кукі та збереження його у сховище сесії
Зміна паролю	UserResolver	changePassword	Токен згенерований для зміни паролю та текстове значення відповідне до нового паролю	Користувач із зміненим полем відповідним до паролю
Створення нового користувача (реєстрація)	UserResolver	register	Текстове значення відповідне до паролю, текстове значення відповідне до електроної пошти та унікальне ім’я	Новий користувач із відповідними вхідним значенням полями

Кінець таблиці 4.1

1	2	3	4	5
Ініціація запиту на зміну паролю	UserResolver	forgotPassword	Текстове значення відповідне до електроної скриньки	Значення true, якщо операція пройшла успішно
Деавторизація користувача	UserResolver	logout	Токен збережений у сховищі сесії	Значення true, якщо токен був видалений із сховища сесії

Для тестування клієнської(користувацької) частини веб-системи були використані практики та методики не-фукціонального тестування. Були використані веб-браузери різних поколінь, та пристрої різної специфікації та конфігурації.

Був протестований веб-додаток у:

Веб-браузерах:

- Microsoft Edge 100.0.1185.39;
- Google Chrome 101.0.4951.67;
- Mozilla Firefox 100.0.2;
- Safari 15.4;
- DuckDuckGo Browser 5.122. 0;
- Opera 63.3.3216.58675;
- Sizzy 61.0.0.

На операційних системах:

- ArchLinux 5.17.5;
- Linux Ubuntu 21.04;
- macOS 12 Monterey;
- Microsoft Windows 11;
- iOS 14.4;
- Android 11.

4.3 Аналіз результатів тестування веб-додатку

Результати модульного тестування серверної частини “Blanket” наведено у таблиці 4.2.

Таблиця 4.2 – Результати тестування серверної частини “Blanket”.

Назва сценарію	Метод	Вхідні дані	Отриманий результат	Результат
1	2	3	4	5
Реагування на публікацію	vote	Унікальний ідентифікатор публікації та числове значення яке відображає реакцію	Публікація із зміненим полем реакцій	Правильно
Створення нової публікації	createPost	Текстове значення відповідне до заголовку, текстове значення відповідне до тексту та необов’язкове текстове значення відповідне до посилання на картинку	Нова публікація із відповідними вхідним даним полями	Правильно
Редагування існуючої публікації	updatePost	Текстове значення відповідне до заголовку, текстове значення відповідне до тексту та унікальний ідентифікатор публікації	Змінена існуюча публікацію із відповідними вхідним даним полями	Правильно
Видалення існуючої публікації	deletePost	Унікальний ідентифікатор публікації	Значення true	Правильно
Авторизація користувача	login	Текстове значення відповідне до паролю або унікального ім’я та пароль	Згенероване кукі та збереження його у сховище сесії	Правильно
Зміна паролю	changePassword	Токен згенерований для зміни паролю та текстове значення відповідне до нового паролю	Користувач із зміненим полем відповідним до паролю	Правильно
Створення нового користувача (реєстрація)	register	Текстове значення паролю, текстове значення електроної пошти та ім’я	Новий користувач із відповідними вхідним значенням полями	Правильно
Ініціація запиту на зміну паролю	forgotPassword	Текстове значення відповідне до електроної скриньки	Значення true	Правильно
Деавторизація користувача	logout	Токен збережений у сховищі сесії	Значення true, якщо токен був видалений із сховища сесії	Правильно

Отже, в процесі написання даного розділу були визначені та описані основні методи проведення тестування веб-додатків. Для написання модульних

та інтеграційних тестів було обрано фреймворк Jest, а для тестування користувацького інтерфейсу та клієнтської частини в цілому були використані сучасні та актуальні методики та практикики не-функціонального тестування.

Як видно з результатів модульного тестування серверної частини ” веб-додатку “Blanket”, вся логіка серверної частини працює саме так, як і передбачалося.

Результати проведення не-функціонального тестування на реальних пристроях та веб-браузерах різних поколінь повністю задовільняють очікувані результати.

Виходячи із вказаного вище, можна зробити висновок що розроблена веб-система працює саме так як і було передбачено та задовільняє вимоги, які були визначенні технічним завданням.

ВИСНОВКИ

В процесі виконання дипломного проекту було проаналізовано предметну область, з’ясовані причини частого виникнення розповсюдженої проблеми із оптимізацією, проблеми частої недовіри користувача до соціальних мереж, а також досліджені наявні програмні засоби. В результаті було визначено їх переваги та недоліки, а також доведено потребу у розробці програмного продукту, який призначатиметься саме для безпечного та стабільного обміну інформацією. Сформовані вимоги були описані у технічному завданні та формувалися, опираючись на інші успішні додатки.

Наступним кроком стало проведення аналізу переваг та недоліків існуючих видів архітектури веб-додатків, і встановлено, що для розроблюваної системи найбільше підходить шаблон SPA.

У результаті детального аналізу було проведено аналіз технологій і засобів, які будуть використовуватись в процесі розробки веб-додатку, визначено та описано основні модулі додатка і те, яким чином вони взаємодіють між собою.

На основі отриманих даних було реалізовано та здійснено опис компонентів та основних методів серверної та клієнтскьої частин веб-додатку Blanket. Розроблений в результаті веб-додаток було протестовано та доведено що він задовольняє усім вимогам, які були поставлені до нього у технічному завданні.

В результаті виконання дипломного проекту було розроблено веб-систему, яка дозволяє її кінцевим користувачам створювати редагувати та видаляти публікації та взаємодіяти із іншими користувачами.

Таким чином, завдання поставлені у процесі дипломного проектування були виконані, а мета досягнута. Оскільки в процесі реалізації веб-додатку використовувались новітні технології та засоби, отримані знання та вміння знадобляться у подальшій професійній діяльності.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming, Marijn Haverbeke [Online] / Counterpoint. – Available: https://eloquentjavascript.net/

2.Node.js in Action Mike Cantelon, Marc Harter, T.J. Holowaychuk, and Nathan Rajlich Foreword by Isaac Z. Schlueter [Online] / Counterpoint. – Available: https://www.manning.com/books/node-js-in-action

3. Web Development with Node and Express: Leveraging the JavaScript Stack 1st Edition – Counterpoint: https://www.amazon.com/Web-Development-Node-Express-Leveraging/dp/1491949309

4. Express in Action: Writing, building, and testing Node.js applications 1st Edition – Available: https://www.amazon.com/Express-Action-Writing-building-applications/dp/1617292427

5. Mastering PostgreSQL 13: Build, administer, and maintain database applications efficiently with PostgreSQL 13, 4th Edition – Available: https://www.amazon.com/Mastering-PostgreSQL-administer-applications-efficiently/dp/1800567499

6. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database – Available: https://www.amazon.com/PostgreSQL-Running-Practical-Advanced-Database/dp/1491963417

7. PostgreSQL Official Documentation [Online] – Available: https://www.postgresql.org/docs/

8. Learning React: Functional Web Development with React and Redux – Available: https://www.amazon.com/Learning-React-Functional-Development-Redux/dp/1491954620

9. The Road to React: Your journey to master React.js in JavaScript (2022 Edition). – Available: https://www.amazon.com/Road-learn-React-pragmatic-React-js-ebook/dp/B077HJFCQX

10. Learning GraphQL: Declarative Data Fetching for Modern Web Apps. – Available: https://www.amazon.com/Learning-GraphQL-Declarative-Fetching-Modern/dp/1492030716

11. TypeORM Official Documentation [Online]. – Available: https://typeorm.io/

12. GraphQL Official Documentation. – Available: https://graphql.org/learn/

13. React A JavaScript library for building user interfaces [Online] – Available: https://reactjs.org/

14. JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language

15. You Don't Know JS: Scope & Closures [Online] – Available: https://www.amazon.com/You-Dont-Know-JS-Closures/dp/1449335586

16. Learning PHP, MySQL, JavaScript, CSS & HTML5: A Step-by-Step Guide to Creating Dynamic Websites [Online] – Available: https://www.amazon.com/Learning-MySQL-JavaScript-HTML5-Step/dp/1491949465

17. Grokking Algorithms: An Illustrated Guide for Programmers and Other Curious [Online] – Available: https://www.amazon.com/Grokking-Algorithms-illustrated-programmers-curious/dp/1617292230

18. JavaScript from Beginner to Professional: Learn JavaScript Quickly by Building Fun, Interactive, and Dynamic Web Apps, Games, and Pages People [Online] – Available: https://www.amazon.com/JavaScript-Beginner-Professional-building-interactive/dp/1800562527

19. The Clean Coder: A Code of Conduct for Professional Programmers [Online] – Available: https://www.amazon.com/Clean-Coder-Conduct-Professional-Programmers/dp/0137081073

ДОДАТОК А
(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Введення

Робота виконується в рамках проекту розробки веб-системи «Blanket». Технічне завдання розроблено у відповідності до стандарту ГОСТ 19.201–78.

1 Підстава для розробки

Підставою для розробки є «Завдання на дипломний проект», затверджене завідувачем кафедри інженерії програмного забезпечення. Найменування розробки: « Веб-система соціальної мережі «Blanket» ».

2 Призначення розробки

2.1 Функціональне призначення

Функціональним призначенням додатку є оприлюднення та розповсюдження публікацій у зазначеній системі, можливість авторизація, внесення змін у існуючі публікації та видалення існуючих публікацій.

2.2 Експлуатаційне призначення

Веб-додаток повинен експлуатуватися за допомогою веб-браузеру. Кінцевим користувачем додатку може виступати будь-яка особа.

3 Вимоги до програми

3.1 Вимоги до функціональних характеристик

Програма повинна забезпечувати можливості виконання таких функцій:

– функція створення публікацій;
– функція редагування та видалення публікацій;
– функція реєстрації;
– функція авторизиції;
– можливість створювати рекції на існуючі публікації;

3.2 Вимоги до надійності

Додаток повинен забезпечувати наступні вимоги до надійності:

– усі деструктивні дії (видалення медіа-файлу або тегів) повинні отримувати додаткове підтвердження від користувача перед тим, як ці зміни будуть застосовані;
– обробляти помилкові дії користувача і повідомляти його про це;
– можливість самовідновлення після збоїв;
– можливість резервного копіювання бази даних.

3.3 Умови експлуатації та вимоги до технічних засобів

Веб-додаток націлений на будь-який пристрій, який підтримує сучасний веб-браузер. Працездатність та коректна робота додатку не гарантується у разі використання користувачем пристрою, або веб-браузера, які не відповідають мінімальним системним вимогам та конфігурації. При цьому для виконання функцій пристрій повинен мати стабільне з'єднання з Інтернетом.

Для успішного використання веб-додатку веб-браузер та пристрій повинні відповідати наступним рекомендованим вимогам:

– Будь-який сучасний веб-браузер;

– чотирьохядровий процесор;

– 1500 МБ оперативної пам'яті;

– Частота процесора – 3.1 ггерц.

3.4 Вимоги до інформаційної та програмної сумісності

При розробці додатку використовуватиметься високорівнева мова програмуваня TypeScript, Node.js – фреймворк, призначений для створення серверної частини веб-додатків, React – фреймворк, призначений для створення користувацької частини веб-додатку. В якості бази даних використовуватиметься PostgreSQL, а TypeORM буде використовуватись як об'єктно-орієнтована технологія для спрощення роботи із нею.

3.5 Спеціальні вимоги

Програма повинна мати зручний та приємний дизайн інтерфейсу, зрозумілий для будь-якого користувача.

4 Вимоги до програмної документації

У момент здачі проекту замовнику надається наступний набір документів:

– текст програми;

– опис програми;

– технічне завдання;

– керівництво користувача.

5 Стадії та етапи розробки

Стадії та етапи розробки програмної системи для обміну інформацією та роботи із моделлю публікації подані у таблиці А.1.

Таблиця А.1 – Стадії та етапи розробки проекту

Стадія розробки	Етапи робіт	Зміст робіт
1	2	3
Технічне завдання 02.01.22 – 31.01.22	Обґрунтування необхідності розробки програми	Коротка характеристика програмного забезпечення; підстава і призначення розробки; вимоги до програмної системи і документація; стадії і етапи розробки програми; порядок контролю і приймання
Ескізний проект 01.02.22 – 26.02.22	Розробка ескізного проекту	Попередня розробка структури вхідних і вихідних даних; уточнення середовища програмування; розробка і опис загальної алгоритмічної структури
Технічний проект 29.02.22 – 19.03.22	Розробка технічного проекту	Уточнення структури вхідних і вихідних даних; розробка докладного алгоритму; розробка структури програми
Робочий проект 20.03.22 – 15.04.22	Розробка програмного забезпечення	Реалізація програмного забезпечення; відлагодження; проведення попереднього тестування
Розробка програмної документації 16.04.22 – 22.04.22	Розробка документації до програмного забезпечення	Розробка необхідної документації, передбаченої технічним завданням
Тестування системи 23.04.22 – 30.04.22	Проведення тестування програмного забезпечення	Розробка методики тестування; проведення основних тестів; коректування програмного забезпечення
Впровадження	Підготовка і передача програми	Підготовка і розгортання програмного забезпечення
Розробка програмної документації 16.04.22 – 22.04.22	Розробка документації до програмного забезпечення	Розробка необхідної документації, передбаченої технічним завданням

6. Порядок контролю та приймання

Контроль здійснюється кінцевими користувачами системи, підключеними на етапі тестування додатку.

ДОДАТОК Б
(обов’язковий)

КОД (ЛІСТИНГ) СЕРВЕРНОЇ ЧАСТИНИ BLANKET

Основний файл index.ts:

```
const main = async () => {
  const conn = await createConnection({
    type: "postgres",
    database: "gc-barb",
    username: "postgres",
    password: "postgres",
    logging: true,
    synchronize: true,
    migrations: [path.join(__dirname, "/migrations/*")],
    entities: [Post, User, Upvote],
  });
  conn.runMigrations();

  // await Post.delete({});

  const app = express();

  const RedisStore = connectRedis(session);
  const redis = new Redis();

  app.use(
    session({
      name: COOKIE_NAME,
      store: new RedisStore({
        client: redis as any,
        disableTouch: true,
      }),
      cookie: {
        maxAge: 1000 * 60 * 60 * 24 * 365 * 10, //10 years
        httpOnly: true,
        sameSite: "lax", //csrf
        secure: __prod__, //cookie only work in https
      },
      secret: "bla12312eadsdiuh12i3u1asdiHAnSDuiasdjhqwouieh12asd6969ket",
      resave: false,
      saveUninitialized: false,
    })
  );

  app.use(
```

```
    cors({
      credentials: true,
      origin: "http://localhost:3000",
    })
  );

  const apolloServer = new ApolloServer({
    schema: await buildSchema({
      resolvers: [PictureResolver, PostResolver, UserResolver],
      validate: false,
    }),
    context: ({ req, res }) => ({
      req,
      res,
      redis,
      userLoader: createUserLoader(),
      upvoteLoader: createUpvoteLoader(),
    }),
    plugins: [ApolloServerPluginLandingPageGraphQLPlayground({})],
  });

  app.use(graphqlUploadExpress());

  await apolloServer.start();
  apolloServer.applyMiddleware({
    app,
    cors: false,
  });

  app.listen(4000, () => {
    console.log("------------server started at localhost:4000------------");
  });
};

main().catch((err) => {
  console.error(err);
});
```

Опис моделі Post.ts:

```
@ObjectType()
@Entity()
export class Post extends BaseEntity {
  @Field()
  @PrimaryGeneratedColumn()
  id!: number;

  @Field()
  @Column()
  title!: string;

  @Field()
  @Column()
  text!: string;

  @Field({ nullable: true })
  @Column({ nullable: true })
  picture?: string;

  @Field()
  @Column({ type: "int", default: 0 })
  points!: number;

  @Field(() => Int, { nullable: true })
  voteStatus: number | null;

  @Field()
  @Column()
  authorId: number;

  @Field()
  @ManyToOne(() => User, (user) => user.posts)
  author: User;

  @OneToMany(() => Upvote, (upvote) => upvote.post)
  upvotes: Upvote[];

  @Field(() => String)
  @CreateDateColumn()
  createdAt: Date;

  @Field(() => String)
  @UpdateDateColumn()
```

```
  updatedAt: Date;
}
```

Опис моделі User.ts:

```
@ObjectType()
@Entity()
export class User extends BaseEntity {
  @Field()
  @PrimaryGeneratedColumn()
  id!: number;

  @Field()
  @Column({ unique: true })
  username!: string;

  @Field()
  @Column({ unique: true })
  email!: string;

  @Column()
  password!: string;

  @Field(() => [Post], { nullable: true })
  @OneToMany(() => Post, (post) => post.author)
  posts: Post[];

  @OneToMany(() => Upvote, (upvote) => upvote.user)
  upvotes: Upvote[];

  @Field(() => String)
  @CreateDateColumn()
  createdAt: Date;

  @Field(() => String)
  @UpdateDateColumn()
  updatedAt: Date;
}
```

Опис логіки postResolver.ts:

```
@InputType()
class PostInput {
```

```
  @Field()
  title: string;
  @Field()
  text: string;
}

@ObjectType()
class PaginatedPosts {
  @Field(() => [Post])
  posts: Post[];
  @Field()
  hasMore: boolean;
}

@ObjectType()
class FilteredPosts {
  @Field(() => [Post], { nullable: true })
  posts: Post[];
}

@Resolver(Post)
export class PostResolver {
  @FieldResolver(() => String)
  textSnippet(@Root() post: Post) {
    const snippet = post.text.split(" ").slice(0, 25).join(" ");
    return snippet;
  }

  @FieldResolver(() => User)
  author(@Root() post: Post, @Ctx() { userLoader }: MyContext) {
    return userLoader.load(post.authorId);
  }

  @FieldResolver(() => Int, { nullable: true })
  async voteStatus(
    @Root() post: Post,
    @Ctx() { upvoteLoader, req }: MyContext
  ) {
    if (!req.session.userId) {
      return null;
    }

    const upvote = await upvoteLoader.load({
      postId: post.id,
```

```
    userId: req.session.userId,
  });

  return upvote ? upvote.value : null;
}

@Mutation(() => Boolean)
@UseMiddleware(isAuth)
async vote(
  @Arg("postId", () => Int) postId: number,
  @Arg("value", () => Int) value: number,
  @Ctx() { req }: MyContext
) {
  const { userId } = req.session;
  const isUpvote = value !== -1;
  const realValue = isUpvote ? 1 : -1;

  const upvote = await Upvote.findOne({ where: { postId, userId } });

  if (upvote && upvote.value !== realValue) {
    await getConnection().transaction(async (tm) => {
      await tm.query(
        `
      update upvote
      set value = $1
      where "postId" = $2 and "userId" = $3
      `,
        [realValue, postId, userId]
      );

      await tm.query(
        `
      update post
      set points = points + $1
      where id = $2
      `,
        [2 * realValue, postId]
      );
    });
  } else if (!upvote) {
    await getConnection().transaction(async (tm) => {
      await tm.query(
        `
      insert into upvote ("userId", "postId", value)
```

```
      values ($1,$2,$3);
      `,
        [userId, postId, realValue]
      );

      await tm.query(
        `
      update post
      set points = points + $1
      where id = $2;

      `,
        [realValue, postId]
      );
    });
  }

  return true;
}
@Query(() => FilteredPosts, { nullable: true })
async filteredPosts(
  @Arg("id", () => Int) id: number
): Promise<FilteredPosts> {
  const posts = await Post.find({
    where: { authorId: id },
    order: { createdAt: "DESC" },
  });
  return { posts };
}

@Query(() => PaginatedPosts)
async posts(
  @Arg("limit", () => Int) limit: number,
  @Arg("cursor", () => String, { nullable: true }) cursor: string,
  @Arg("id", () => Int, { nullable: true }) id: number,
  @Ctx() { req }: MyContext
): Promise<PaginatedPosts> {
  const realLimit = Math.min(50, limit);
  const realLimitPlus = realLimit + 1;

  const replacements: any[] = [realLimitPlus];

  if (req.session.userId) {
    replacements.push(req.session.userId);
```

```
    }

    let cursorIdx = 3;
    if (cursor) {
      replacements.push(new Date(parseInt(cursor)));
      cursorIdx = replacements.length;
    }

    if (id) {
      replacements.push(id);
    }

    const posts = await getConnection().query(
      `
    select p.*,

      ${
        req.session.userId
          ? '(select value from upvote where "userId" = $2 and "postId" = p.id)
"voteStatus"'
          : 'null as "voteStatus"'
      }
    from post p
    ${cursor ? `where p."createdAt" < $${cursorIdx}` : ""}
    ${id ? `where p."authorId" = $3` : ""}
    order by p."createdAt" DESC
    limit $1
   `,
      replacements
    );

    return {
      posts: posts.slice(0, realLimit),
      hasMore: posts.length === realLimitPlus,
    };
  }

  @Query(() => Post, { nullable: true })
  post(@Arg("id", () => Int) id: number): Promise<Post | undefined> {
    return Post.findOne(id);
  }

  @Mutation(() => Post)
  @UseMiddleware(isAuth)
```

```
async createPost(
  @Arg("input") input: PostInput,
  @Arg("img", { nullable: true }) img: string,
  @Ctx() { req }: MyContext
): Promise<Post> {
  return Post.create({
    ...input,
    authorId: req.session.userId,
    picture: img,
  }).save();
}

@Mutation(() => Post, { nullable: true })
@UseMiddleware(isAuth)
async updatePost(
  @Arg("title") title: string,
  @Arg("text") text: string,
  @Arg("id", () => Int) id: number,
  @Ctx() { req }: MyContext
): Promise<Post | null> {
  const result = await getConnection()
    .createQueryBuilder()
    .update(Post)
    .set({ title, text })
    .where('id = :id and "authorId" = :authorId', {
      id,
      authorId: req.session.userId,
    })
    .returning("*")
    .execute();

  return result.raw[0];
}

@Mutation(() => Boolean)
@UseMiddleware(isAuth)
async deletePost(
  @Arg("id", () => Int) id: number,
  @Ctx() { req }: MyContext
): Promise<boolean> {
  const post = await Post.findOne(id);
  if (!post) {
    return false;
  }
```

```
    if (post.authorId !== req.session.userId) {
      throw new Error("not authorized");
    }
    await Upvote.delete({ postId: id });
    await Post.delete({ id });
    return true;
  }
}
```

Опис логіки userResolver.ts:

```
@ObjectType()
class FieldError {
  @Field()
  field: string;
  @Field()
  message: string;
}

@ObjectType()
class UserResponse {
  @Field(() => [FieldError], { nullable: true })
  errors?: FieldError[];

  @Field(() => User, { nullable: true })
  user?: User;
}

@Resolver(User)
export class UserResolver {
  @Query(() => [User])
  users(): Promise<User[]> {
    return User.find({});
  }

  @Query(() => User, { nullable: true })
  me(@Ctx() { req }: MyContext) {
    //you are not logged in
    if (!req.session.userId) {
      return null;
    }

    return User.findOne(req.session.userId);
  }
```

```
@Query(() => User, { nullable: true })
user(@Arg("id", () => Int) id: number): Promise<User | undefined> {
  return User.findOne(id);
}

@Mutation(() => UserResponse)
async changePassword(
  @Arg("token") token: string,
  @Arg("newPassword") newPassword: string,
  @Ctx() { redis, req }: MyContext
): Promise<UserResponse> {
  if (newPassword.length <= 2) {
    return {
      errors: [
        {
          field: "newPassword",
          message: "password lenght must be greater than 2",
        },
      ],
    };
  }

  const key = FORGET_PASSWORD_PREFIX + token;
  const userId = await redis.get(key);
  if (!userId) {
    return {
      errors: [
        {
          field: "token",
          message: "token expired",
        },
      ],
    };
  }

  const userIdNum = parseInt(userId);
  const user = await User.findOne(userIdNum);
  if (!user) {
    return {
      errors: [
        {
          field: "token",
          message: "user no longer exists",
        },
```

```
        ],
      };
    }

    await User.update(
      {
        id: userIdNum,
      },
      {
        password: await argor2.hash(newPassword),
      }
    );

    req.session.userId = user.id;

    await redis.del(key);

    return { user };
  }

  @Mutation(() => Boolean)
  async forgotPassword(
    @Arg("email") email: string,
    @Ctx() { redis }: MyContext
  ) {
    const user = await User.findOne({ where: { email } });
    if (!user) {
      return true;
    }

    const token = v4();

    await redis.set(
      FORGET_PASSWORD_PREFIX + token,
      user.id,
      "ex",
      1000 * 60 * 60 * 24 * 3
    );

    await sendEmail(
      email,
      `<a href="http://localhost:3000/change-password/${token}">reset
password</a>`
    );
```

```
  return true;
}

@Mutation(() => UserResponse)
async register(
  @Arg("options") options: UsernamePasswordInput,
  @Ctx() { req }: MyContext
): Promise<UserResponse> {
  const hashedPassword = await argor2.hash(options.password);

  const errors = validateRegister(options);
  if (errors) {
    return { errors };
  }

  let user;
  try {
    const result = await getConnection()
      .createQueryBuilder()
      .insert()
      .into(User)
      .values({
        username: options.username,
        email: options.email,
        password: hashedPassword,
      })
      .returning("*")
      .execute();

    user = result.raw[0];
  } catch (err) {
    //duplicate user error
    if (err.code === "23505") {
      return {
        errors: [
          {
            field: "username",
            message: "username already taken",
          },
        ],
      };
    }
  }
```

```
  req.session.userId = user.id;

  return { user };
}

@Mutation(() => UserResponse)
async login(
  @Arg("usernameOrEmail") usernameOrEmail: string,
  @Arg("password") password: string,
  @Ctx() { req }: MyContext
): Promise<UserResponse> {
  const user = await User.findOne(
    usernameOrEmail.includes("@")
      ? { where: { email: usernameOrEmail } }
      : { where: { username: usernameOrEmail } }
  );
  if (!user) {
    return {
      errors: [
        {
          field: "usernameOrEmail",
          message: "no user with this username",
        },
      ],
    };
  }
  const valid = await argor2.verify(user.password, password);
  if (!valid) {
    return {
      errors: [
        {
          field: "password",
          message: "incorrect password",
        },
      ],
    };
  }

  req.session.userId = user.id;

  return { user };
}

@Mutation(() => Boolean)
```

```
  logout(@Ctx() { req, res }: MyContext) {
    return new Promise((resolve) =>
      req.session.destroy((err) => {
        res.clearCookie(COOKIE_NAME);
        if (err) {
          // console.log(err);
          resolve(false);
          return;
        }

        resolve(true);
      })
    );
  }
}
```

ДОДАТОК В
(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Хмельницький Національний Університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

Дипломний проект на тему :

Веб-система соціальної мережі «Blanket»

Студент: Зубашевський Єгор Денисович
Керівник : Гурман Іван Васильович, кандидат технічних наук, доцент

Мета та завдання проекту

Мета проекту - розробка веб-системи у вигляді соціальної мережі, яка дозволятиме її користувачам оприлюднювати та взаємодіяти із публікаціями, а також реєструватись та авторизуватись

Завдання, які необхідно вирішити для досягнення мети:

- Провести аналіз предметної області ;
- Сформувати технічне завдання ;
- Спроектувати програмний продукт ;
- Виконати програмну реалізацію проекту ;
- Провести тестові випробування додатку .

Актуальність теми

Актуальність теми полягає в тому, що більшість існуючих соціальних мереж погано оптимізовані та через те що більшість з них були створені давно, немає можливості суттєво пришвидшити та оптимізувати вже існуючі системи.

Окрім того, найбільші гравці у сфері соціальних мереж так чи інакше продають та крадуть дані, тобто не доброчинно ставляться до даних користувача.

Ці два ключових фактора мені здається вже найближчим часом змусять більшість постійних користувач існуючих систем, перейти на більш оптимізовані та більш довірені системи.

Перейдемо до огляду архітектури та проектування веб-додатку.

Логічна модель бази даних

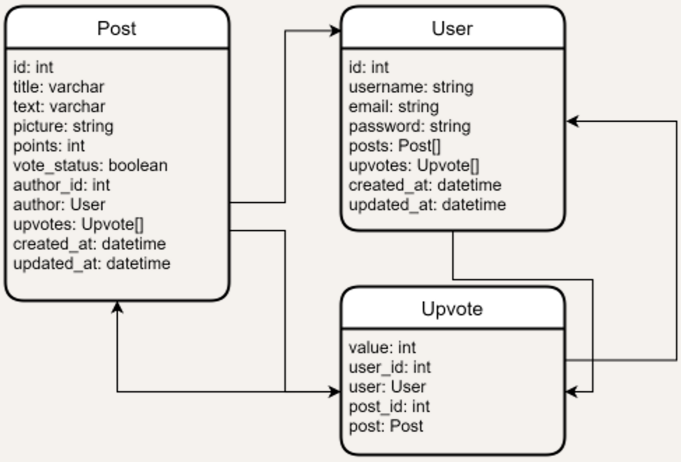
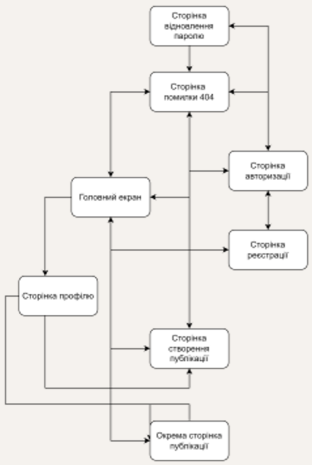


Схема переходів між сторінками



Технології використані для створення веб-додатку

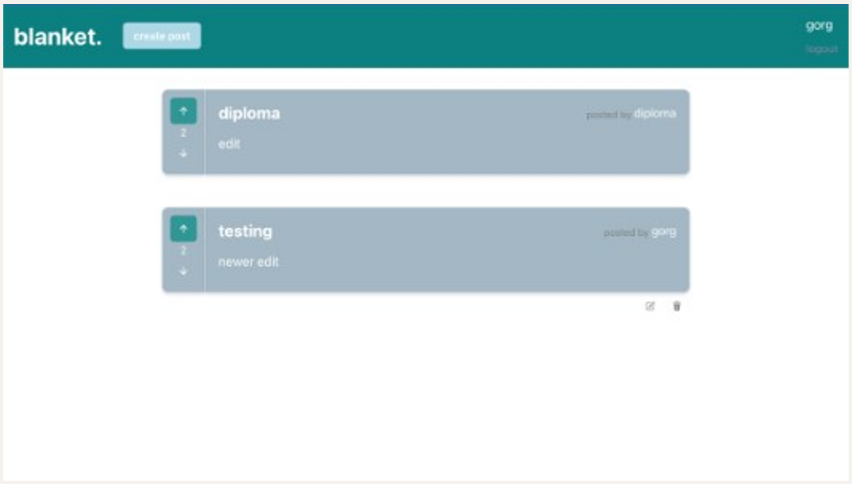
Серверна частина :

- Node.js
- Typescript
- GraphQL
- TypeORM
- PostgreSQL
- Apollo Server
- Express

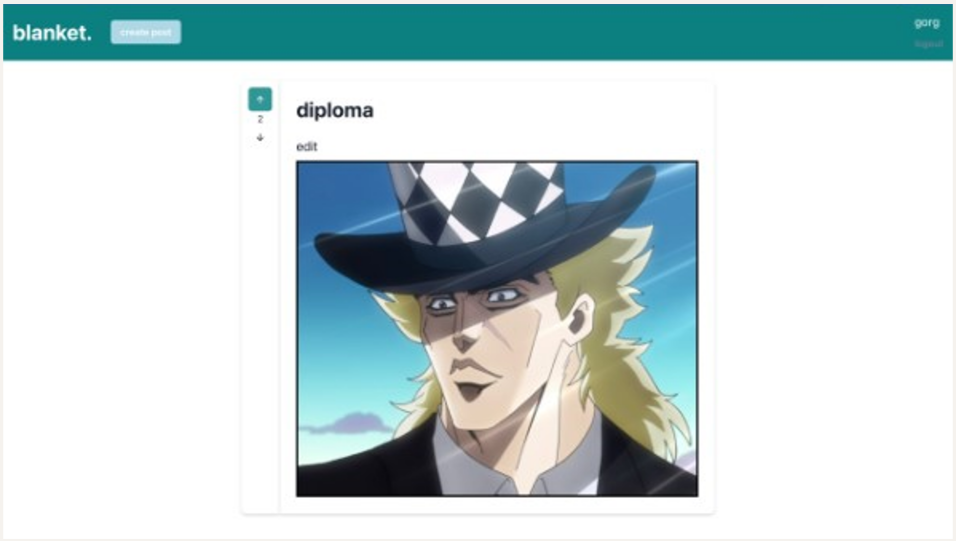
Клієнтська частина:

- React.js
- Next.js
- Typescript
- Chakra UI
- URQL

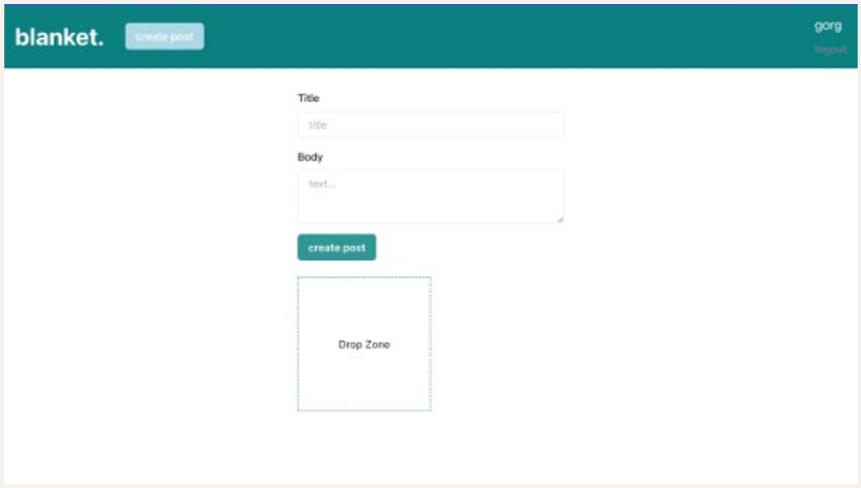
Головна сторінка



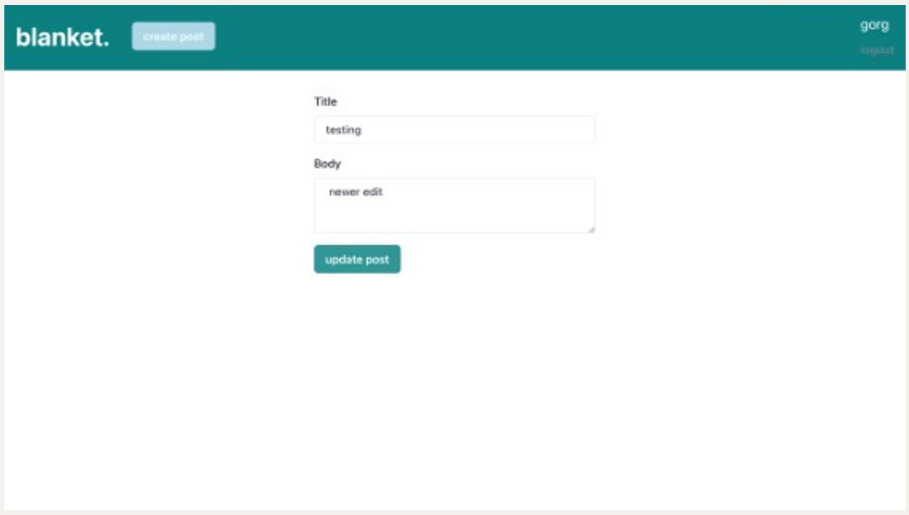
Сторінка публікації



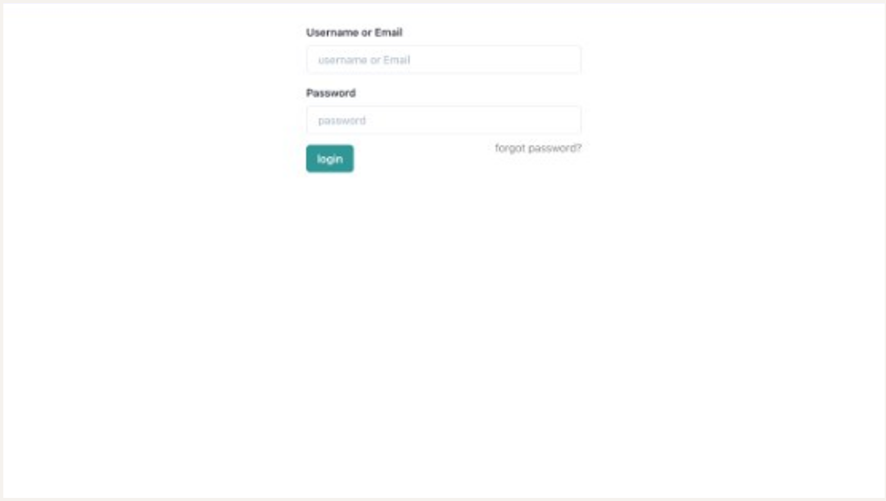
Сторінка створення публікації



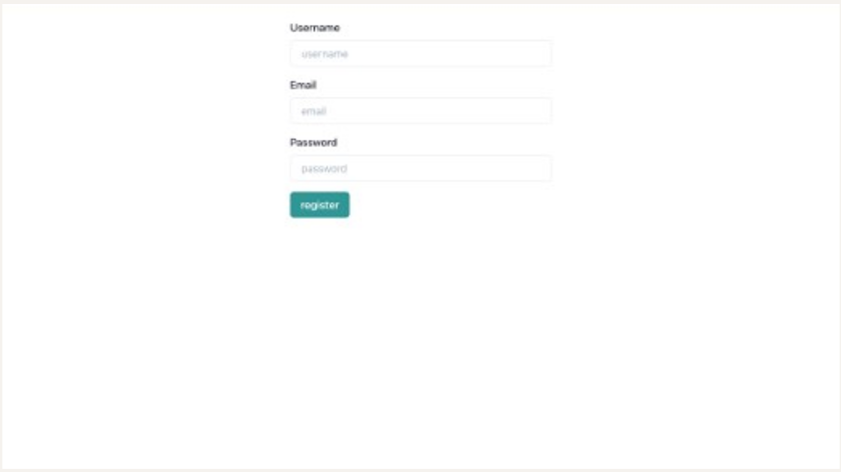
Сторінка редагування публікації



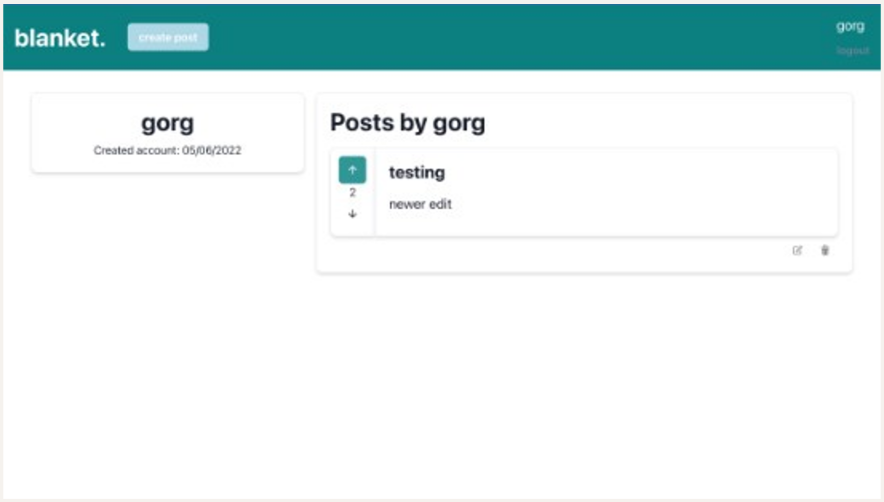
Сторінка авторизації



Сторінка реєстрації



Сторінка профілю користувача



Висновки

В процесі виконання дипломного проекту було проаналізовано предметну область, з'ясовані причини частого виникнення розповсюдженої проблеми із оптимізацією, проблеми частої недовіри користувача до соціальних мереж, а також досліджені наявні програмні засоби. В результаті було визначено їх переваги та недоліки, а також доведено потребу у розробці програмного продукту, який призначатиметься саме для безпечного та стабільного обміну інформацією. Сформовані вимоги були описані у технічному завданні та формувалися, опираючись на інші успішні додатки.

Наступним кроком стало проведення аналізу переваг та недоліків існуючих видів архітектури веб-додатків, і встановлено, що для розроблюваної системи найбільше підходить шаблон SPA.

У результаті детального аналізу було проведено аналіз технологій і засобів, які будуть використовуватись в процесі розробки веб-додатку, визначено та описано основні модулі додатка і те, яким чином вони взаємодіють між собою.

На основі отриманих даних було реалізовано та здійснено опис компонентів та основних методів серверної та клієнтскьої частин веб-додатку Blanket. Розроблений в результаті веб-додаток було протестовано та доведено що він задовольняє усім вимогам, які були поставлені до нього у технічному завданні.

В результаті виконання дипломного проекту було розроблено веб-систему, яка дозволяє її кінцевим користувачам створювати редагувати та видаляти публікації та взаємодіяти із іншими користувачами.

Таким чином, завдання поставлені у процесі дипломного проектування були виконані, а мета досягнута.

Оскільки в процесі реалізації веб-додатку використовувались новітні технології та засоби, отримані знання та вміння знадобляться у подальшій професійній діяльності.

Дякую за увагу!

Завідувачу кафедри інженерії програмного
забезпечення проф. Бедратюку Л. П.
здобувача вищої освіти
Зубашевського Є.Д.
факультет ІТ, 4 курс, група ІПЗ-18-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 30.05.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

31.05.2022 р.
дата

підпис

Thu Jun 02 08:39:35 EEST 2022, Хіврич Володимир Русланович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 3.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 12%**

	Документ		Сумарний збіг по Базі Даних	
ID: 104363 Назва: Веб-система соціальної мережі «Blanket» Додано в БД: 2022-06-02 Автора: Є. Д. Зубашевський Керівники: І. В. Гурман Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	72527	1102	4518 (6%)	64 (6%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1011425150

Дата перевірки:
02.06.2022 09:41:41 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
02.06.2022 09:47:52 EEST

ID користувача:
100005589

Назва документа: **ДП_ПЛАГІАТ**

Кількість сторінок: **75** Кількість слів: **12838** Кількість символів: **101203** Розмір файлу: **2.08 MB** ID файлу: **1011305727**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

16.1%
Схожість

Найбільша схожість: **8.82% з джерелом з Бібліотеки** (ID файлу: **1008357188**)

6.82% Джерела з Інтернету	219	Сторінка 77
10.3% Джерела з Бібліотеки	123	Сторінка 79

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування	14 сторінок

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНИЙ ПРОЕКТ
освітнього ступеня «Бакалавр»

Дипломник Зубашевський Єгор Денисович

Тема Веб-система соціальної мережі «Blanket»

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг дипломного проекту:

Кількість листів креслень _____; кількість сторінок записки _____

1.Короткий зміст пояснювальної записки та прийнятих рішень У дипломному проекті було досліджено і проаналізовано предметну область, усі функціональні та нефункціональні вимоги. Був проведений аналіз існуючих програм на ринку, розглянуто їх переваги і недоліки, та доведено актуальність розробки нового програмного забезпечення. Було розглянуто інструменти для реалізації дипломного проекту, в результаті чого створено програмне забезпечення. Також було проведено тестування програми, за результатами якого доведено, що розроблене програмне забезпечення працює коректно та готове до експуатації.

2. Висновок про відповідність проекту поставленому завданню Дипломний проект виконаний відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу проекту, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі доведено актуальність теми, визначено мету та завдання дипломноого проектування. У першому розділі проведено аналіз предметної області, розглянуто існуючі рішення та визначені функціональні і нефункціональні вимоги до розроблюваного програмного забезпечення. У другому розділі проведено аналіз сучасних архітектур, розглянуто їх переваги і недоліки та визначено, що система буде відповідати монолітній архітектурі та моделі клієнт-сервер. У третьому розділі підготовлено всі залежності для написання коду та виконано практичну розробку програмних модулів і описано їх особливості, в результаті чого створено програмний продукт. В четвертому розділі було виконано модульне тестування системи та проведено його у відповідності до функціональних вимог, в результаті було підтверджено коректну роботу програми.

4. Позитивні сторони проекту Тематика дипломного проекту є актуальною, оскільки на сьогодні у світі існує проблема приватності та безпеки користувацьких даних. Також дійсною проблемою є те що існуючі рішення, створенні використовуючи застарілі методики та технології, через це погано оптимізовані.

5. Негативні сторони проекту У проекті пошук публікацій та користувачів відсутній та відсутні методи комунікації за допомогою приватних повідомлень.

6. Оцінка графічного оформлення та пояснювальної записки проекту Графічне оформлення виконано відповідно до теми дипломного проекту та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про дипломний проект в цілому Дипломний проект заслуговує позитивної оцінки. Матеріал пояснювальної записки структурований, послідовний, чіткий та простий, що дозволяє чітко зрозуміти викладений матеріал у рамках тематики дипломного проекту. Графічний матеріал дає можливість наочно побачити деталі проектування системи.

8. Інші зауваження

9. Оцінка дипломного проекту Дипломний проект виконаний у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «добре».

РЕЦЕНЗЕНТ Бобровнікова Кіра Юліївна, к.т.н. доцент кафедри комп'ютерних систем та інформаційних технологій

“ 6 ” червня 2022 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: «Веб-система соціальної мережі «Blanket»»

Автор: Зубашевський Єгор Денисович

Спеціальність: 121 – Інженерія програмного забезпечення

Освітня програма: освітньо-професійна програма Інженерія програмного забезпечення

Науковий керівник: Гурман Іван Васильович, к.т.н, доцент.

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних). Робота приймається до захисту.	Відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Сумарні співпадіння документа складають 16.1%

Серед співпадінь з документами у глобальній мережі максимальне співпадіння з одним документом становить 6,82% та стосуються переліку джерел посилань.

Серед співпадінь з документами з бібліотеки максимальне співпадіння з одним документом становить 10,3% та стосуються стандартних сторінок поясновальних записок тобто – титульний аркуш, бланк завдання, тощо.

Керівник _____ І. В. Гурман

Гарант ОП _____ Л. П. Бедратюк

Завідувач кафедри _____ Л. П. Бедратюк