Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Соціальна мережа для збору даних про попит на ринку програмного забезпечення
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 КвРІПЗ.190129.01.04.ПЗ

Виконав студент IV курсу група ІПЗ-19-1 _____ С. А. Грига
Підпис Ініціали, прізвище

Керівник канд. пед. наук, доцент _____ Н. І. Праворська
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер канд. техн. наук, доцент _____ О.М. Яшина
Підпис Ініціали, прізвище

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

1 червня 2023 р.

Хмельницький, 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних 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
Л. П. Бедратюк
05.02.2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Грига Сергій Андрійович

Прізвище, ім'я, по батькові студента

1. Тема кваліфікаційної роботи Соціальна мережа для збору даних про попит на ринку програмного забезпечення

Керівник кваліфікаційної роботи канд. пед. наук, доцент Праворська Н. І.

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.03.2023 р. № 5

2. Строк подання студентом роботи на кафедру 01.06.2023 р.

3. Вихідні дані до роботи Матеріали переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Характеристика предметної області та постановка задачі, проектування програмного забезпечення ,програмна реалізація.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Три креслення е формату А3:ER-діаграма бази даних, діаграма декомпозиції першого рівня для процесу створення публікації, діаграма декомпозиції другого рівня для процесу авторизації

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Яшина О. М., к.т.н, доцент	25.05.23	6.06.2023
Антиплагіат	Гурман І. В., к.т.н, доцент	1.06.2023	1.06.2023

7. Дата видачі завдання « 02 » січня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1. Збір матеріалу за темою кваліфікаційної роботи (КвР), дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення здач та вимог, розробка технічного завдання	02.01–31.01 2023	
2. Проектування програмного забезпечення	01.02–28.02 2023	
3. Програмна реалізація програмного забезпечення	01.03–10.04 2023	
4. Тестування програмного забезпечення	11.04–30.04 2023	
5. Написання вступу, загальних висновків, оформлення переліку джерел та посилання на додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05–25.05 2023	

Студент _____ Грига. С. А.
Підпис Ініціали, прізвище

Керівник роботи _____ Праворська Н. І.
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема кваліфікаційної роботи «Соціальна мережа для збору даних про попит на ринку програмного забезпечення».

Автор роботи: Грига Сергій Андрійович.

Керівник роботи: Праворська Наталія Іванівна

Пояснювальна записка: 63 с., 32 рис., 1 табл., 2 дод., 44 джерел.

Графічна частина: 3 креслення формату А3.

Об'єктом дослідження є структура організації соціальних мереж, які використовують систему постів для поширення інформації.

Предметом дослідження даного дипломного предмету є процес дослідження та розробки соціальної мережі.

Головною метою при виконанні даної кваліфікаційної роботи є розробка програмного забезпечення з функціоналом для поширення інформації між користувачами системи за допомогою постів та збору різноманітних даних на основі якої можна провести детальний аналіз стану на ринку програмного забезпечення у сфері соціальних мереж.

У роботі проведено аналіз предметної області, вивчено вже наявні на ринку програмні продукти, їх переваги та недоліки. На основі отриманої інформації було визначено вимоги до програмного забезпечення та здійснено його детальне проектування з вибором технологій розробки.

Використовуючи попередні розділи було розроблене програмне забезпечення, яке відповідає усім поставленим вимогам та за потреби може бути швидко модифікована у майбутньому.

30.05.2023
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.190129.01.04.ПЗ	Пояснювальна записка	63		
2	А4		Завдання на кваліфікаційну роботу	1		
3	А4		Анотація	1		
			Графічні документи			
4	А4		Презентаційні матеріали (Слайди)	18		
5	А3	КвРІПЗ.190129.01.04.Е8	ER-діаграма бази даних	1		
6	А3	КвРІПЗ.190129.01.04.Е8	Діаграма декомпозиції першого рівня для процесу створення публікації	1		
7	А3	КвРІПЗ.190129.01.04.Е8	Діаграма декомпозиції другого рівня для процесу авторизації	1		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.190129.01.04.ВД		
Виконав		Грига С. А.		5.06	Соціальна мережа для збору даних про попит на ринку програмного забезпечення	Літ.	Арк. 1 / Аркушів 1
Керівник		Праворська Н.І.		5.06			
Н. контр.		Яшина О.М.		5.06		ХНУ, ІПЗ-19-1	
Зав. каф.		Бедраюк Л. П.		6.06			

ЗМІСТ

					КвРІПЗ.190129.01.04.ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Виконав		Грига С. А.		5.06	Соціальна мережа для збору даних про попит на ринку програмного забезпечення	Літ.	Арк. 4 / Аркушів 63
Керівник		Праворська Н.І.		5.06			
Н. контр.		Яшина О.М.		5.06		ХНУ, ІПЗ-19-1	
Зав. каф.		Бедраюк Л. П.		6.06			

ВСТУП

Проаналізувавши ряд українських та зарубіжних статей, доповідей, статистичних даних та інших типів джерел було зроблено припущення, що у найближчі декілька років попит на нові соціальні мережі, які є не перевантаженими різноманітними можливостями значно виросте. Дане припущення було зроблено на основі наступних даних.

По перше, ряд великих компаній, які на сьогодні домінують у сфері соціальних мереж, виділяють велику частину власних ресурсів на розвиток інших сфер діяльності, що у свою чергу призводить до зменшення темпів розвитку їх основного продукту у довгостроковій перспективі.

По друге, лідери даного ринку перебуваючи у постійній конкуренції між собою відтворюють ідентичний функціонал не звертаючи уваги на його доцільність. Таким чином один програмний продукт включає у себе різноманітний функціонал, але не завжди задовільної якості. Це призводить до зниження якості окремих послуг, зменшує їх конкурентоспроможність та значно підвищує складність усієї системи.

По третє, у деяких компаніях відбувається ряд складних змін у керівній верхівці та структурі управління, наприклад реструктуризація, через що змінюється подальший план розвитку та основні цілі. У зв’язку з цим ряд користувачів висловлює незадоволеність та навіть про відмови від використання їхніми програмними продуктами.

Для підтвердження даного припущення було вирішено, що доцільно розробити мінімально життєздатний продукт за допомогою якого можна збирати данні для подальшого аналізу ринку соціальних мереж, на попит серед користувачів з можливістю його подальшої модифікації.

Отже, через вплив багатьох факторів на ринок програмного забезпечення у сфері соціальних мереж на сьогоднішній день актуальною темою стає дослідження попиту на ринку та можливість подальшої конкуренції нового

програмного забезпечення із вже існуючими рішеннями, які зарекомендували себе як якісні.

Об’єктом дослідження є структура організації соціальних мереж, які використовують систему постів для поширення інформації.

Предметом дослідження даної кваліфікаційної роботи є процес дослідження та розробки соціальної мережі на основі системи поширення інформації між користувачами за допомогою системи постів.

Головною метою при виконанні кваліфікаційної роботи є розробка програмного забезпечення з функціоналом для поширення інформації між користувачами системи за допомогою публікацій та збору різноманітної статистики, на основі якої можна провести детальний аналіз стану на ринку програмного забезпечення у сфері соціальних мереж.

Для досягнення мети проекту необхідно вирішити наступні завдання:

– провести детальний аналіз предметної області;

– проаналізувати переваги та недоліки вже існуючих рішень;

– на основі попереднього аналізу встановити вимоги до програмного продукту, на основі яких необхідно обрати найкращі та найефективніші методи і засоби для розробки програмного забезпечення;

– розробити ряд діаграм для візуального відображення структури та архітектури програмного продукту;

– розробити програмне забезпечення відповідно до раніше встановлених вимог та розроблених діаграм;

– провести тестові випробування.

1 ХАРАКТЕРИСТИКА ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

Соціальна мережа – це програмне забезпечення яке надає функціональні можливості для обміну різного виду інформацією між користувачами системи, які об’єднуються за певними ознаками.

Соціальні мережі поділяються на наступні види:

– соціальні мережі для спілкування. До них можна віднести Facebook, Discord, Telegram та інші;

– соціальні мережі для обміну контентом, зазвичай фото та відео інформацією. До них відносяться YouTube та Instagram;

– соціальні мережі для колективних переговорів, наприклад Reddit;

– соціальні мережі для авторського запису. До таких мереж можна віднести Twitter та Blogger;

– соціальні мережі по інтересам;

– сервіси соціальних закладок, де користувачі зберігають інформацію в свою особисту бібліотеку.

Необхідно відмітити, що даний перелік є умовним і в залежності від джерела може відрізнятись, адже їх класифікація відбувається і досі, а деякі соціальні мережі можуть відноситись відразу до декількох категорій.

Як вже було наведено вище розроблювана соціальна мережа передбачає обмін різноманітною інформацією між користувачами за допомогою системи постів між підписниками, тому подальше вивчення предметної області здійснювалось опираючись на дану систему поширення інформації.

Пост, інша назва публікація – це повідомлення у соціальній мережі яке може включати у себе текстову інформацію, фото, відео чи аудіо файли. У більшості випадках, інші користувачі соціальної мережі можуть виражати

власні думки про побачену інформацію за допомогою текстових повідомлень чи різних систем оцінки.

При такій реалізації соціальної мережі поширення інформації відбувається за допомогою системи підписок та рекомендації на основі інтересів користувача та раніше переглянутих ним постів.

Система підписок, її ще називають відслідковуванням, передбачає, що при зацікавлені користувачем інформацією, яку поширює інший користувач, перший повідомляє соціальну мережу про свою зацікавленість за допомогою відслідковування. Після даних маніпуляцій усі нові пости, які створить другий користувач будуть відображатись у стрічці новин чи отримувати інформацію про зміни у його профілі за допомогою системи нотифікації.

Стрічка новин – це окреме вікно у соціальних мережах, де користувач системи може переглядати пости від інших користувачів, на яких він підписався, рекомендації та здійснювати пошук постів за власними критеріями. Також дане вікно може містити додатковий функціонал, який може змінюватись в залежності від обраного типу соціальної мережі.

Будь-яка соціальна мережа передбачає поділ користувачів на групи в залежності від причин використання, наданого функціоналу та інших факторів. Зазвичай такий поділ розділяють на три основні категорії, а саме користувачів, адміністрацію та підтверджених користувачів. Адміністрація відповідає за модерацію соціальної мережі, відслідковує коректність її роботи та обслуговування. Користувачами називають людей, які поширюють між собою інформацію, а перевірені користувачі, це люди, які зарекомендували себе як достовірне джерело інформації, за що отримали додаткові переваги. За потреби дані групи можуть бути розбиті на менші підгрупи.

Метою даної кваліфікаційної роботи є розробка соціальної мережі для ефективного збору даних та подальшого аналізу ринку програмного забезпечення. У даному випадку доцільним є обмежити охоплення аудиторії. Найефективнішим підходом для вирішення даної задачі є позиціювання

соціальної мережі, як платформи для поширення інформації одного типу, але разом з цим не обмежуючи її однією темою, наприклад політикою чи релігією.

Після детального аналізу останніх тенденцій на ринку програмного забезпечення було встановлено, що переважна більшість інформації, яка поширюється соціальними мережами різного типу є розважальною і, у більшості випадків, низької якості без будь-якого смислового навантаження. У зв'язку з перенасиченням ринку розважального контенту використання даної тематики вважається недоцільною.

Впродовж останніх декількох років популярності набула різноманітна науково-популярна інформація з різних галузей науки. Досить часто вона буває складною у розумінні, а її вивчення може займати багато часу. При зменшені розміру тексту, або подання його невеликими розділами така інформація зможе ефективно поширюватись за допомогою соціальних мереж. З даних причин при розробці програмного забезпечення його позиціювання відбувалось, як соціальної мережі призначеною для поширення наукової інформації у зрозумілій для користувача формі.

Відповідно до позиціювання потенційними користувачами системи є чоловіки та жінки віком від 16 до 55 років будь якої національності, віри чи інших факторів, та які бажають дізнаватись цікаву наукову інформацію у зрозумілому для них вигляді при мінімальних часових витратах на її розуміння та пошук.

Основними категоріями користувачів, які необхідні у кінцевому програмному продукту є:

– адміністрація;

– користувач;

– підтверджений користувач.

Відповідно до встановлених категорій та їхніх функціональних можливостей у даній соціальній мережі було розроблено діаграму варіантів

використання для кожної категорії користувачів. Отриманий результат можна побачити на рисунку 1.1.

Рисунок 1.1 – Діаграма варіантів використання

З рисунку видно, що підтверджений користувач має деякі переваги у порівнянні зі звичайним користувачем, але основним їхнім функціоналом є поширення інформації, тоді як у адміністратора інструменти переважно використовуються для модерації.

1.2 Аналіз наявного програмно-технічного забезпечення

На сучасному ринку програмного забезпечення існує значний вибір різноманітних соціальних мереж, які задовольняють різноманітні потреби користувачів. Перший з видів програм використовується для обміну повідомленнями. До них можна віднести Telegram, Viber та WhatsApp, які забезпечують швидкий та зручний спосіб спілкування між користувачами.

Інший вид програм, спрямовані на обмін відеофайлами та розважальним контентом, наприклад YouTube та TikTok. Ці платформи надають користувачам можливість ділитися відеоматеріалами, викликаючи популярність та спільноту навколо цих вмісту.

Також варто згадати про соціальні мережі, які дозволяють обмінюватися комбінованою інформацією та обговорювати її за допомогою постів. Один з яскравих прикладів – Reddit, де користувачі можуть публікувати пости на різні теми та вступати в дискусії з іншими учасниками.

Це лише кілька прикладів програмних продуктів, які присутні на ринку програмного забезпечення. Існує ще багато інших платформ, кожна з яких має свої унікальні особливості та спрямована на задоволення певних потреб користувачів. З розширенням їхньої кількості з'являються нові можливості для комунікації, обміну інформацією та взаємодії у віртуальному середовищі.

Як вже було зазначено у попередніх розділах, розроблюваний програмний продукт передбачає використання системи постів, які можуть включати у себе різноманітну текстову інформацію, відео та зображення різноманітного змісту, тому при досліджені вже існуючих рішень було розглянуте програмне забезпечення, яке використовує схожу систему поширення інформації.

Найдоцільнішим було б розпочати огляд з Facebook. Його розвиток розпочався у 2004 році як соціальна мережа для студентів декількох американських університетів, але згодом програма змогла стати серйозним конкурентом на світовому ринку програмного забезпечення.

На сьогоднішній день, за даними DataReporter, Facebook вважається найбільш популярною соціальною мережею майже з трьома мільйонами активних користувачів[4].

Facebook надає користувачам можливість поширення різноманітної інформації за допомогою системи постів, які активно поширюються між користувачами за допомогою системи підписок та рекомендацій. Кожен пост може включати у себе текстову інформацію разом з фото та відео матеріалами. Кожен з цих елементів може бути використано одночасно, або ж самостійно. Приклад посту на Facebook зображено на рисунку 1.2 [7].

Рисунок 1.2 – Пост на Facebook

З даного рисунку видно, що кожен пост надає можливості іншим користувачам системи виразити власну думку, щодо наданої інформації. Для цього соціальна мережа надає користувачам два основні інструменти – систему коментарів та вподобайок.

За допомогою системи коментарів, користувачі мають можливість залишати відгуки у формі текстової інформації. Це відкриває можливості для обговорення цікавих моментів публікації або задавання різноманітних запитань, пов’язаних з вмістом посту, та надання на них відповідей. Коментарі створюють атмосферу взаємодії між користувачами, дозволяючи їм обмінюватися думками та поглядами.

З іншого боку існує система вподобайок, яка надає швидкий спосіб вираження власних вражень від посту без необхідності детального висловлення думки. Вона дозволяє користувачам швидко вибрати одну з багатьох доступних емоцій, що відображають їхнє ставлення до переглянутої інформації. Це дає змогу виразити позитивне або негативне сприйняття вмісту, показати зацікавленість або незадоволення з метою покращення майбутніх публікацій, не вдаючись до докладного розповіді.

Таким чином, система коментарів і система вподобайок взаємодіють між собою, створюючи багатогранну платформу для спілкування та вираження емоцій у віртуальному середовищі соціальної мережі. Вони доповнюють одна одну, надаючи користувачам різні способи взаємодії з вмістом та вираження своїх думок та почуттів.

Варто відзначити, що платформа Facebook, крім можливості створення постів від імені користувачів, надає можливості створення груп на різні тематики. Групи в даній соціальнім мережі мають схожий основний функціонал, що і в користувачів, але дозволяють декільком користувачам з відповідними правами створювати пости від імені відповідної групи.

Це може бути корисною функцією у багатьох випадках. Наприклад, коли декілька людей спільно керують певним проектом, і вони хочуть публікувати пости від імені групи, щоб зберегти єдність та спільність комунікації відповідно до обраної тематики та стилю публікацій.

Групи також можуть бути корисним інструментом для організацій, спільнот, навчальних закладів та багатьох інших сфер, де потрібна спільна платформа для обміну інформацією та спілкування. Також їх можна використовувати для реклами власного бізнесу, продажу продукції, тощо, що допомагає збільшити зацікавленість користувачів.

Окрім системи постів Facebook надає інші інструменти, наприклад короткі відео та власний месенджер для спілкування між користувачами, але їх використання є опціонним, а деякі користувачі уникають їх. З даних причин їх опис вважається недоцільним.

Серед основних переваг Facebook є:

– велика база користувачів;

– великий спектр інструментів для підприємців;

– дешева та ефективна реклама;

– можливість створення груп;

– покращена система вподобайок.

Основними недоліками даної соціально мережі є:

– низький рівень конфіденційності користувачів через обмежений функціонал для налаштування профілю користувача;

– низький рівень модерації, що призводить до поширення недостовірної інформації серед користувачів;

– наявністю великої кількості шахраїв;

– перенавантажений інтерфейс;

– велика кількість різноманітного функціоналу, який перенавантажує загальну систему.

Серед інших існуючих рішень також необхідно розглянути соціальну мережу Twitter. За відвідуваністю користувачами у мережі Інтернет він займає одинадцяте місце у всьому світі і використовує схожу систему постів та методи, для поширення інформації між користувачами системи. Його головну сторінку зображено на рисунку 1.3 [8].

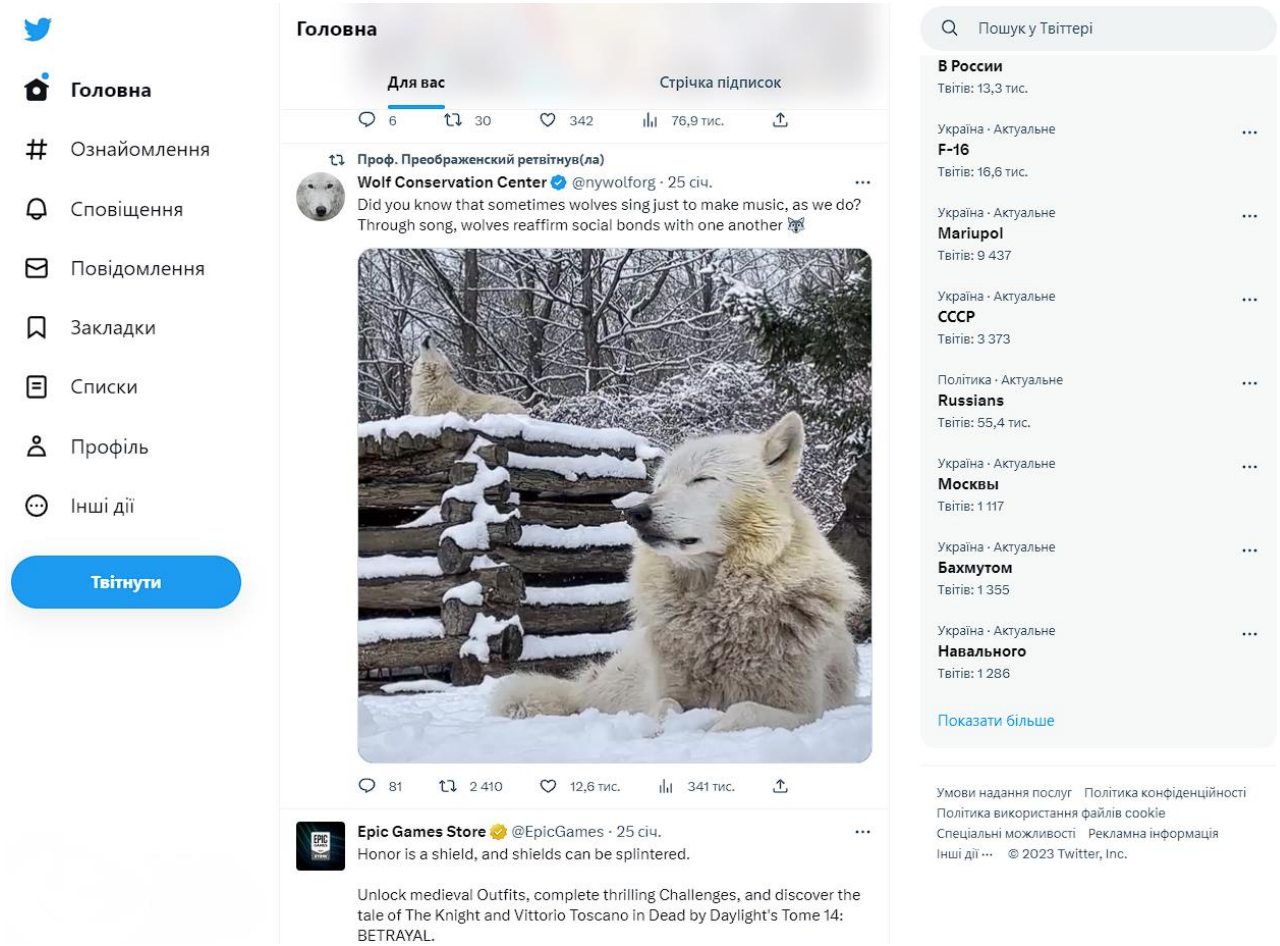


Рисунок 1.3 – Головна сторінка Twitter.

Основний функціонал постів у даній соціальній мережі, яку називають «Твіти», майже ідентичний, але має різну реалізацію порівняно з іншими платформами. Кожен пост в Twitter має обмеження тексту у 280 символів, що може вважатись як перевагою, так і недоліком. Ця обмежена кількість символів спонукає користувачів до стислого висловлення своїх думок, що стимулює кращу конкретизацію та уникнення зайвих деталей, але разом із цим можуть

виникати проблеми пов’язані з розкриттям комплексних, або важких для розуміння тематики.

Система вподобайок у Twitter надає можливість лише позитивної оцінки посту, тоді як у Facebook користувачам надається вибір серед декількох різних емоцій, що дозволяє точніше виразити їхні реакції на вміст. Це можна віднести до недоліків мережі, але разом із цим такий вид реалізації дозволяє спростити зовнішній вигляд застосунку та зробити взаємодію з інтерфейсом користувача швидшою та інтуїтивно зрозумілою.

Також варто окремо виділити додатковий функціонал, який отримала система коментарів. На відміну від інших соціальних мереж, Twitter дозволяє користувачам на особистій сторінці переглядати як свої власні пости, так і коментарі до постів, які залишив власник профілю. Це дозволяє більш широкому колу користувачів бачити та відслідковувати інформацією, що була опублікована. Такий підхід збільшує кількість потенційних переглядачів та залучає більше уваги до конкретного посту.

Однією з головних особливостей соціальної мережі Twitter є система хештегів. Вони представляють собою ключові слова або фрази без пробілів, перед якими ставиться символ «#». Вони використовуються для позначення теми чи контексту повідомлення.

Завдяки хештегам, користувачі мережі можуть швидко знайти пов'язану інформацію на певну тему. Якщо хештеги правильно використовуються, вони допомагають згруповувати повідомлення з однаковими хештегами разом. Це робить пошук інформації більш зручним та ефективним для користувачів.

Крім того, використання хештегів дозволяє збільшити охоплення користувачів для публікацій. Коли користувачі шукають або натискають на певний хештег, вони можуть побачити всі публікацій, до яких було додано цей хештег. Це дозволяє поширювати інформацію на більш широку аудиторію, залучати більше уваги до постів та стимулювати взаємодію між користувачами соціальної мережі.

Необхідно відмітити, що інші соціальні мережі, наприклад, вже раніше згаданий Facebook, також використовують дану систему, але завдяки заохочення користувачів до їх активного використання вона набула найбільшої популярності та отримала одну з найкращих реалізації у Twitter.

Основними перевагами даної соціальної мережі є:

– великий спектр налаштувань конфіденційності користувача;

– повна відсутність реклами;

– розвинута система хештегів;

– велика база користувачів.

Основними недоліками Twitter є:

– обмеження у 280 символів;

– велика кількість неактивних користувачів;

– ряд можливостей доступний за додаткову плату.

Підсумовуючи аналіз даних соціальних мереж, можна сказати, що соціальні мережі, які використовують систему постів, у більшості випадків, мають схожий основний функціонал, але з різними відмінностями, покращеннями чи модифікаціями, що допомагають виділитись їм серед інших конкурентів на ринку.

До таких можливостей можна віднести систему коментарів, вподобайок та різноманітні способи поширення інформації серед користувачів соціальних мереж з функціонал для пошуку інформації на конкретну тематику, його сортуванням, тощо.

До основних переваг соціальних мереж, які можуть сильно впливати на загальну думку користувачі про фінальний програмний продукт, можна віднести великий спектр налаштувань конфіденційності інформації користувача, простоту у використанні та не перевантаженість додатковим непотрібним функціоналом.

1.3 Постановка задач та вимог до програмного продукту

На основі характеристики предметної області та аналізі вже існуючих рішень було встановлено, що даний програмний продукт буде позиціонувати себе як соціальна мережа для поширення науково–популярного контенту. Таким чином було встановлено обмежене коло потенційних користувачів, що на етапі впровадження та проведення рекламної компанії допоможе максимально ефективно використати бюджетні кошти та краще встановити попит на ринку програмному забезпечення.

Архітектура програмного забезпечення передбачає реалізацію системи авторизації та реєстрації у соціальній мережі за допомогою якої користувачі зможуть отримувати доступ до всього основного функціоналу даної відповідно до ролі профілю користувача.

Передбачуваний поділ серед усіх користувачів у системі відбуватиметься за трьома основними ролями, а саме – адміністрація, користувачі та підтверджені користувачі.

Адміністрація буде відповідати за модерацію, перевірку стану роботи системи зі статистикою та повідомлення користувачів про зміни.

Модерація передбачає надання можливості для адміністраторів у видаленні постів та коментарів, які не відповідають вимогам соціальної мережі, а уся статистика повинна бути подана на окремому вікні у короткому вигляді з можливістю швидкого знаходження необхідних даних.

Авторизовані у системі користувачі зможуть обмінюватись різноманітною інформацією між собою за допомогою системи постів. Також кожен користувач системи матиме особистий профіль, де буде розміщена уся наявна та дозволена до перегляду інформація про нього та пости, які він створив. Додатково профіль користувача має функціонал для налаштування відображення особистої інформації та можливості її редагування.

Підтверджені користувачі не будуть відрізнятись функціоналом від звичайних, але матимуть додаткові переваги, наприклад пріоритет у пошуку, додаткові відмітки та інше.

Кожен пост може містить у собі текстову інформацію, фото чи відео файли. Також передбачена система вподобайок та текстових коментарів для оцінки чи думки про щойно прочитану інформацію з можливістю ігнорування їхнього використання.

Окрім цього до постів можна додавати різноманітні хештеги, які будуть вказувати на подальший вміст публікації та можуть бути використані для швидкого пошуку усіх постів з відповідним хештегом.

Поширення постів відбуватиметься за допомогою системи відслідковування користувачами інших користувачів, де вони будуть розміщуватись. Для перегляду усіх постів та їх пошуку передбачено окреме вікно у вигляді стрічки постів з додатковим функціоналом для пошуку постів, користувачів, тощо.

До нефункціональних вимог до даного програмного продукту найважливішим є забезпечення можливостей по одночасній роботі із системою великої кількості користувачів, для чого необхідно забезпечити високу стійкість системи до різного виду навантажень.

У подальшому передбачається можливість модифікації системи за допомогою реалізації нових функціональних можливостей, тому розроблювана система повинна надавати можливості по доданню нового функціоналу без необхідності внесення глобальних правок у програмний код.

У кольоровій гамі інтерфейсу користувача повинні переважати пастельні кольори з невеликою кількістю відтінків. Вікна інтерфейсу повинні містити функціонал, який відповідає назві самого вікна та бути згруповані у блоки відповідно за їхнім призначенням.

1.4 Висновки. Постановка основної задачі

Першим етапом виконання даного розділу було проведення детального аналізу області. Для цього було проаналізовано поділ соціальних мереж на види за призначенням для використання, визначено їхні основні функціональні можливості та інші особливості.

Базуючись на отриманій інформації було встановлено основну мету даної кваліфікаційної роботи, визначено необхідність обмеження групи потенційних користувачів, але разом з цим не робити її занадто обмеженою. На основі цього було встановлено потенційних користувачів системи, визначено поділ користувачів у системі та розроблено діаграму варіантів використання для демонстрації основних можливостей програмного продукту.

Наступним кроком було проведено аналіз вже існуючих рішень на основі популярних соціальних мереж Facebook та Twitter. На основі проведеного аналізу було встановлено основні особливості, переваги та недоліки даних соціальних мереж. Це допоможе при розробці програмного продукту уникнути недоцільного функціоналу та покращити його якість.

Використовуючи усю отриману інформацію при аналізі предметної області та вже існуючих рішень було визначено основні вимоги на розробку програмного продукту, його основний функціонал, вимоги до інтерфейсу користувача, тощо. Основними задачами на реалізацію є:

- реалізувати систему авторизації/реєстрації;
- реалізувати систему постів;
- реалізувати можливість налаштування даних користувачів;
- реалізувати систему статистики.

Виконання даного розділу допомогло встановити основні вимоги до програмного продукту, що у подальшому спростить процес вибору технологій та розробки програмного продукту.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вибір засобів розробки програмного забезпечення

Вибір технологій та засобів розробки програмного продукту є одним із найважливіших етапів у його створенні. Від даного вибору у подальшому залежить якість програмного продукту, швидкість виконання завдань співробітниками та можливі проблеми, які постануть перед розробниками у процесі роботи над проектом. З даних причин перед остаточним затвердженням технологій та методів розробки доцільно розглянути різноманітні мови програмування, підходи до розробки, тощо. Тому, врахувавши знання та навички розробників, можливості компанії у фінансуванні та інших аспектах, які впливають на процес розробки.

Для прикладу одна із найпопулярніших соціальних мереж Facebook розроблена на мові програмування PHP та компілюється за допомогою HipHop for PHP «трансформатор вихідного коду», який був розроблений командою Facebook, трансформує код PHP у оптимізований код C++. Це допомогло зменшити споживання енергії серверами компанії приблизно на 50% без зменшення швидкодії роботи соціальної мережі.

Ще одна популярна соціальна мережа Twitter розроблена на мові програмування Ruby з використанням фреймворку Ruby on Rails, який забезпечує швидку розробку, підтримку різноманітних баз даних та створення API. Також при створенні даної соціальної мережі використовувались такі мови програмування, як Java та JavaScript.

У випадку даного програмного продукту вибір фреймворку матиме вирішальний вплив на усі аспекти пов’язані з створенням та покращенням програми. Основними фреймворками, які детально розглядались з метою подальшого використання були ASP.NET Core, Angular та Flask.

ASP.NET Core – це фреймворк на основі мов програмування C# та F#, створений компанією Microsoft для розробки різноманітних веб-додатків. На

його основі створено сайти Microsoft, StackOverflow, Dell, тощо. До основних переваг даного фреймворку можна віднести:

– висока швидкість розробки;
– широкий спектр можливостей масштабування;
– велика кількість технологій та інструментів;
– висока продуктивність;
– високий рівень безпеки;
– наявність великої кількості документації;
– кросплатформеність.

Але разом із перевагами він має ряд недоліків, а саме ASP.NET Core є вимогливою для вивчення, має сурові вимоги до інфраструктури та в залежності від проекту, його специфіки та інших аспектів може потребувати великих фінансових витрат.

Angular – це фреймворк на основі мови програмування TypeScript, створений Google. Для різних завдань його використовували у PayPal, Lego.com, Upwork, Freelance та багато інший програмних продуктів. До основних переваг Angular відносять:

– висока швидкість розробки;
– широкий спектр інструментів та бібліотек різного призначення;
– підтримка багатьох платформ;
– можливості масштабування проекту.

До недоліків даного фреймворку можна віднести вимогливість до вивчення, необхідність постійного використання TypeScript та помірна підтримка зі сторони спільноти.

І останнім розглянутим фреймворком є Flask. Він базується на основі мови програмування Python, а його основною метою використання є швидка розробка різноманітних веб-додатків.

– швидка розробка;
– помірна складність вивчення;

– велика кількість навчальних матеріалів;

– широкий вибір бібліотек та пакетів;

– висока підтримка зі сторони спільноти.

Але, на відміну від попередніх фреймворків Flask є досить сильно обмеженим у масштабованості та має обмеження у підтримці платформ.

Для остаточного вибору фреймворку було вирішено зробити детальне порівняння основних можливостей та характеристик. Усі отримані дані було внесено у таблицю 2.1.

Таблиця 2.1 – Порівняння параметрів фреймворків

Параметр порівняння	ASP.NET Core	Angular	Flask
Тип фреймворку	Full-stack	Front-end	Back-end
Мови програмування	C#, F#, VB.NET	TypeScript, JavaScript	Python
Складність вивчення	Висока	Висока	Середня
Використання шаблону	MVC	MVVM	Різні шаблони
Підтримка розробки десктопних програм	Так, з використанням Xamarin	Так, з використанням Ionic	Ні
Підтримка розробки мобільних програм	Так, з використанням .NET Framework	Ні	Ні
Підтримка браузерів	Усі браузери	Усі браузери	Усі браузери
Підтримка серверів	Усі, які підтримують .NET Core	Не використовується	Усі сервери, які працюють на Python
Підтримка баз даних	Бази даних, з підтримкою .NET Core	Усі бази даних, які підтримують Angular	Усі бази даних, які підтримують Python
Підтримка тестування	Так	Так	Так
Підтримка безпеки	Так	Так	Так
Платіжні можливості	Так, з використанням ASP.NET SignalR	Так	Так

На основі аналізу даної таблиці було вирішено, що серед усіх наведених фреймворків найкращим вибором буде використання ASP.NET Core з врахуванням майбутніх перспектив розвитку програмного продукту та спектру можливостей, які він надає.

Так, як розроблюване програмне забезпечення є веб-застосунком його необхідною його складовою є HTML та CSS. Кожна з них виконує власні специфічні завдання, але їхня основна ціль це створення інтерфейсу користувача та його візуального оформлення.

HTML – це мова розмітки гіпертексту для документів перегляду різноманітних веб-сторінок у браузері. З її допомогою на сторінці розміщуються різноманітні елементи сторінок, наприклад, заголовки, параграфи, кнопки та посилання, тощо. Для цього використовуються теги та різні атрибути для визначення семантики та структури контенту сторінки.

CSS, або каскадна таблиця стилів – це мова стилів, яка надає можливості створення різноманітного зовнішнього вигляду для елементів HTML, наприклад кольори, шрифти, фони, зміна їхнього розміру чи розміщення та навіть зміна вигляду елементів при наведенні вказівника миші. Також він надає можливості відділити представлення від опису їхніх стилів, що дозволяє створити більш гнучку та підтримувану систему.

Наступним етапом вибору технологій було визначення архітектурних шаблонів, які будуть використані під час розробки. Так, як кінцевий програмний продукт є веб-застосунком з багатьма сторінками з різним наповнення використовуватиметься клієнт-серверна архітектура. З її використанням система буде поділено на дві основні частини – клієнтська та серверна. Для отримання різного виду даних клієнт буде відправляти запит за допомогою браузеру до віддаленого сервера. Після отримання запиту сервер оброблятиме його, сформує відповідь та відправить отриманий результат назад клієнтській частині. Загальний принцип роботи даної архітектури можна побачити на рисунку 2.1.

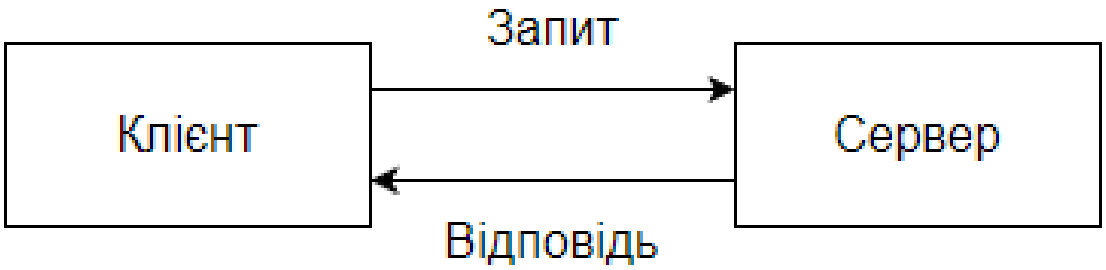


Рисунок 2.1 – Принцип роботи клієнт-серверної архітектуру

Ще однією важливою архітектурою, яку передбачено використовувати у даному програмного продукту є багатошарова архітектура, а саме архітектурний шаблон MVC. Головною особливістю даного шаблону проектування є те, що він розділяє весь програмний продукт на три незалежні рівні, які можуть взаємодіяти між собою, а саме:

– моделі;

– представлення;

– контролери.

Модель являє собою клас, який вміщує різноманітні дані над якими виконуються різноманітні дії. Також може включати у себе певні правила обробки даних.

Представлення являє собою HTML код сторінки, яку у подальшому бачить користувач на екрані власного монітору у вигляді графічного інтерфейсу користувача з яким можна взаємодіяти.

Контролер – це клас, який забезпечує зв’язок між системою та користувачем. Його основною задачею є обробка користувача запитів та відправлення результату.

Загальну схему взаємодії між даними компонентами було детально зображено на рисунку 2.2.

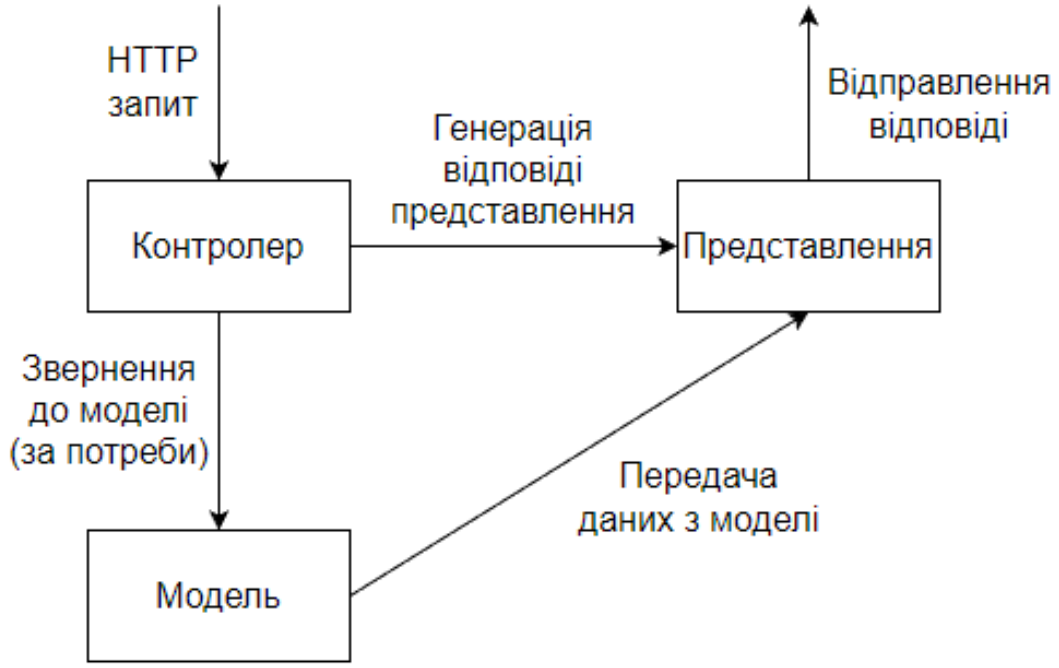


Рисунок 2.2 – Взаємодія між компонентами

За допомогою використання даної архітектури можна здійснювати зміни у окремих модулях, що дозволяє швидко модифікувати програмний продукт без необхідності внесення змін до коду інших модулів, що дозволяє унеможливити появлення помилок у інших частинах системи. Також у майбутньому така організація компонентів значно спростить модифікацію програмного продукту.

Одним із найважливіших компонентів даної соціальної мережі є база даних, оскільки у ній зберігається вся інформація, що використовується на всіх сторінках сайту. Швидкість доступу до бази даних має прямий вплив на ефективність роботи всієї системи. З метою спрощення взаємодії з базою даних та зменшення можливих помилок, використовується Entity Framework.

Entity Framework є фреймворком для платформи .NET, який надає розробникам зручність у роботі з базами даних, використовуючи об'єктно-орієнтовану модель програмування. Він дозволяє легко взаємодіяти з різноманітними реляційними базами даних, такими як Microsoft SQL Server, Oracle, MySQL та інші. Розробники можуть використовувати мови

програмування C# або Visual Basic .NET для взаємодії з базами даних за допомогою цього фреймворку.

Entity Framework дозволяє зосередитись на основній логіці програми та подальшій обробці даних, забезпечуючи автоматичне створення SQL-запитів та взаємодію з базою даних через об'єкти і класи. Він надає можливість розробникам працювати на вищому рівні абстракції, що спрощує розробку та підтримку програмного забезпечення.

До його основних переваг можна віднести:

– спрощення доступу до бази даних;

– зменшення кількості коду;

– підтримка LINQ, що дозволяє працювати з даними, використовуючи вищий рівень абстракції, що може запобігати появі різноманітних помилок, пов'язаних з SQL-запитами;

– вбудована підтримка міграцій;

– підтримка транзакцій та ізоляції рівня, що забезпечує цілісність даних та їх безпеку.

У зв’язку з тим, що різноманітні дані необхідно часто оновлювати, наприклад кількість вподобайок чи коментарів, передбачено використання технології AJAX. З її допомогою можна оновлювати частину веб-сторінки без необхідності її повного перезавантаження, що зменшує навантаження на сервер та збільшує комфорт при використанні програмного продукту. Дана технологія використовує мову програмування JavaScript для зчитування даних з сервера асинхронно, що забезпечує підвищення швидкодії та ефективності роботи сторінки, але у зв’язку з цим може підвищитись складність розуміння коду. Також необхідно відмітити, що використання даної технології можуть виникати проблеми з безпекою при некоректній реалізації.

Окрім цього JavaScript може використовуватись у даному програмному продукті для реагування на дії користувача на HTML сторінці, динамічні змінювати, видаляти чи додавати елементи та багато іншого.

2.2 Розробка прототипу інтерфейсу користувача

Графічний інтерфейс користувача є важливим елементом будь-якого програмного продукту, оскільки він визначає спосіб взаємодії користувача з програмою протягом усього часу. При розробці інтерфейсу користувача необхідно враховувати специфіку програмного забезпечення, встановлені вимоги та потреби користувачів. Такий підхід дозволяє забезпечити ряд наступних переваги в розробленому інтерфейсі користувача застосунку:

– зменшення кількості помилок. При роботі з програмним продуктом через непоінформованість чи некомпетентність користувачі систем можуть допускатись помилок, наприклад некоректно введені дані, але за допомогою відповідних поміток та графічних індикаторів їх кількість можна зменшити;

– підвищення ефективності. Інтуїтивно зрозумілий інтерфейс разом із ефективним та зрозумілим розміщенням кнопок в залежності від частоти їхнього використання допомагають зменшити часові витрати на виконання різного виду завдань та збільшує швидкість взаємодії з програмним продуктом;

– підвищення комфорту використання. Візуально привабливий та зрозумілий інтерфейс зробить взаємодію з програмною більш зручною для користувачів, що у свою чергу збільшить задоволеність при використанні. Це допоможе у майбутньому зробити потенційних користувачів більш лояльними як до програмного продукту, так і до компанії його розробника;

– збільшення кола потенційних користувачів. Коректно розроблений інтерфейс допоможе збільшити доступність для різних категорій користувачів, включаючи людей з різним рівнем технічної грамотності.

Погано продуманий інтерфейс користувача може призводити до проблем у його використанні. Наприклад, низька ефективність роботи, погано розміщені кнопки, тощо, що призводить до незадоволення користувачів від взаємодії з програмою. Дискомфорт при використанні також може бути наслідком незручного розташування елементів навігації.

У подальшому ці аспекти можуть привести до зменшення кількості активних користувачів, оскільки конкуренти можуть запропонувати інші аналоги з поліпшеним інтерфейсом, більшою зручністю та ефективністю, тощо. Користувачі шукають продукти, які пропонують зручність, ефективність і задоволення від використання. Тому важливо вкладати зусилля в розробку зручного інтерфейсу, щоб залучити і утримати активних користувачів.

Відповідно до раніше встановлених вимог до інтерфейсу було передбачено використання обмеженої кількості кольорів пастильних відтінків та розміщення функціоналу у блоках відповідно до визначеного призначення сторінки. З метою візуалізації інтерфейсу, було розроблено прототип інтерфейсу користувача для основних сторінок програмного продукту.

Однією з головних сторінок соціальної мережі є стрічка новин. Передбачається, що користувачі будуть перебувати на ній більшу частину свого часу про використані застосунку. З його допомогою можна переглядати різноманітні публікації та здійснювати пошук за допомогою різноманітних фільтрів. Відповідний прототип даної сторінки було зображено на рисунку 2.3.

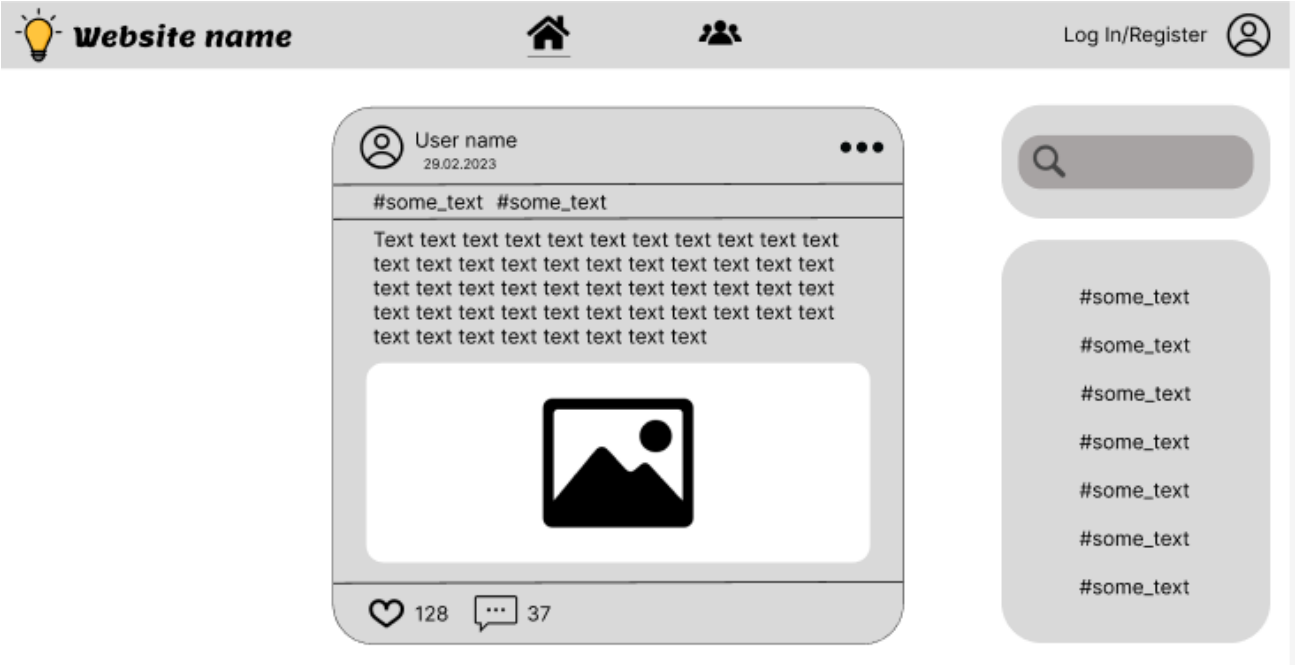


Рисунок 2.3 – Прототип стрічки новин

З даного рисунку видно, що у верхній частині сторінки було розміщено навігаційне меню. У його лівій частині розміщено логотип соціальної мережі та назву. У центрі розміщено кнопки для відкриття інших сторінок, а саме стрічки новин, профіль користувача та список користувачів. При авторизації як адміністратор на даній стрічні буде розміщено кнопку переходу до вікна перегляду статистики, а при авторизація як користувач буде доступна кнопка переходу до вікна налаштувань. У правій частині стрічки розміщено кнопки для переходу до вікон реєстрації та авторизації у системі.

У центральній частині екрану будуть розміщуватись пости користувачів соціальної мережі. Кожен пост виглядає як невеликий блок з текстовою інформацією та може містити різноманітні файли. У верхній частині блоку можна ознайомитись з автором публікації та точною датою його створення. Там же розміщено список хештегів, які були прикріплені користувачем та кнопка різноманітних налаштувань посту, функціонал якої може змінюватись в залежності від конкретного профілю користувача чи інших факторів.

У нижній частині розміщено кнопку вподобайок та коментарів з їх вказаною кількістю. За необхідності при натисканні на даний блок передбачено відкриття даного посту, але у більшому розмірі. У такому випадку відкритий пост містить блок для створення та перегляду коментарів. Така реалізація допоможе зменшити використання робочого простору на головній сторінці та зможе приховати додатковий функціонал, який не завжди є необхідним.

У правій частині екрану було вирішено розмістити блоки зі схожим функціоналом. Перший блок використовується для пошуку користувачів та постів з відповідними хештегами. За необхідності на у них можна розмістити додаткові параметри для пошуку чи фільтрації.

На другому блоці передбачено розміщення декількох хештегів, які найчастіше використовувались користувачами соціальної мережі в продовж останніх двадцяти чотирьох годин. При натисканні на них буде здійснено автоматичний пошук усіх постів, які містять даний хештег.

Наступним важливим вікном у даному програмному продукті є профіль користувача. Його прототип можна побачити на рисунку 2.4.

Рисунок 2.4 – Профіль користувача

У лівій частині вікна було розташовано блок, який містить основну інформацію про користувача, яку він вказав при реєстрації в соціальній мережі. Така інформація може включати його ім'я, фотографію профілю, деякі особисті дані, наприклад рік народження, або інші важливі деталі. Також у даному блоці була розміщена кнопка для початку або закінчення відслідковування користувача, в залежності від поточного стану.

Якщо користувач бажає дізнатися більше деталей про інших користувачів він може натиснути на відповідну кнопку, яка розміщена в кінці даного блоку. При цьому відкриється вікно, де буде відображена детальна інформація про користувача, яку він сам вказав та залишив доступною в базі даних соціальної мережі. Це можуть бути його додаткові контактні дані, освіта, зацікавлення, опис або інші персональні відомості. В залежності від побажань користувача інформацію можна приховати від інших.

У центральній частині вікна розміщена панель посту. Всі публікацій, які можна переглянути на сторінці, були створені власником сторінки та мають ідентичний функціонал та вигляд із стрічки новин.

Така організація інформаційних блоків та функціональних елементів сприяє зручності використання соціальної мережі. Користувачі можуть швидко знаходити необхідну інформацію про себе та інших користувачів, а також з легкістю створювати та переглядати пости конкретного користувача.

При відкритті профілю власником сторінки та наявній активній авторизації у системі у даному вікні додатково відображатиметься кнопка для створення нової публікації у полі блоці основної інформації про користувача.

Раніше було визначено, що кожен профіль можна детально налаштувати з урахуванням особистих побажань користувачів. Це означає, що вікно налаштування профілю повинно надавати широкий спектр параметрів, щоб забезпечити достатню гнучкість налаштування кожного профілю

У вікні налаштування профілю користувач повинен мати можливість змінювати особисті дані, такі як, ім’я, пошта, тощо. Крім того, користувачу необхідно надати ряд налаштувань приватності профілю, тобто чи робити інформацію публічною. Прототип даного вікна зображено на рисунку 2.5.

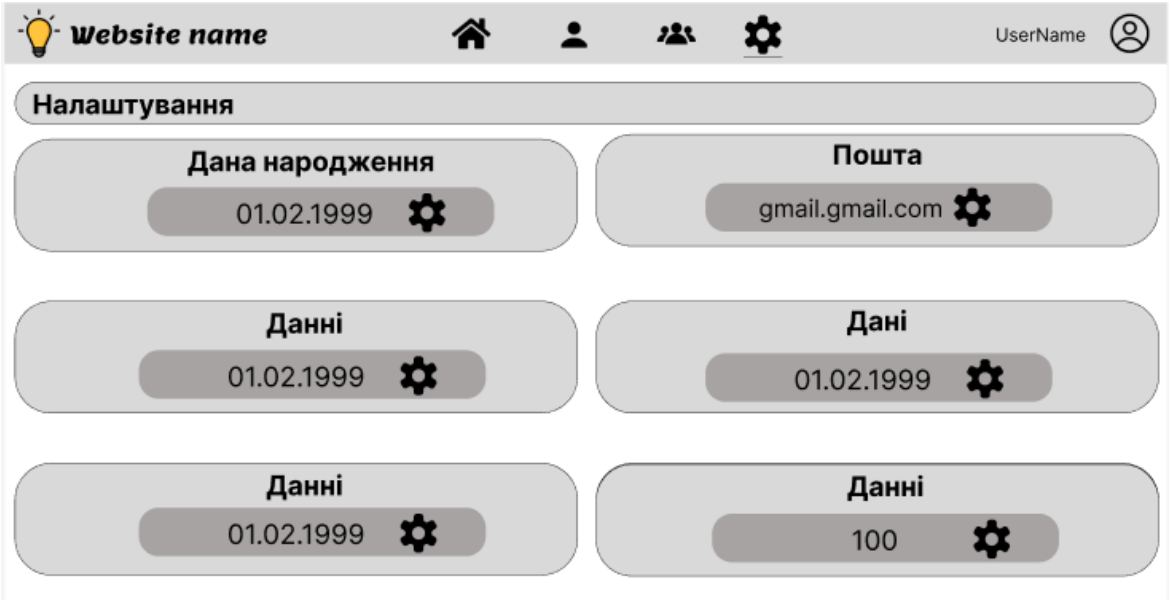


Рисунок 2.5 – Прототип вікна налаштувань профілю

З рисунку видно, що вікно налаштування профілю складається з багатьох незалежних блоків, які мають ідентичний зовнішній вигляд. Кожен із блоків містить у собі заголовок, який відповідає вмісту, значення параметру та кнопку для відкриття вікна детальних налаштувань. Інформація у даних блоках може бути різною, наприклад день народження, пошта, тощо.

Оскільки профіль користувача може містити велику кількість параметрів було прийнято рішення, що для спрощення зовнішнього вигляду вікна доцільним використовувати систему попереднього відкриття окремих вікон поверх основного для зміни відповідного параметра профіля користувача.

Це означає, що при бажанні змінити конкретні дані користувач повинен натиснути на відповідну кнопку. Це призведе до відкриття нового вікна поверх основного, де можна змінити інформацію. Такий підхід дозволяє розбити велику кількість налаштувань на менші групи, що полегшує їх сприйняття.

Ще одним важливим вікном, яке регулярно використовуватиметься є список як усіх користувачів. З його допомогою у соціальній мережі люди зможуть шукати користувачів, які можуть бути цікавими для них. Детально ознайомитись з даним прототипом можна на рисунку 2.6.

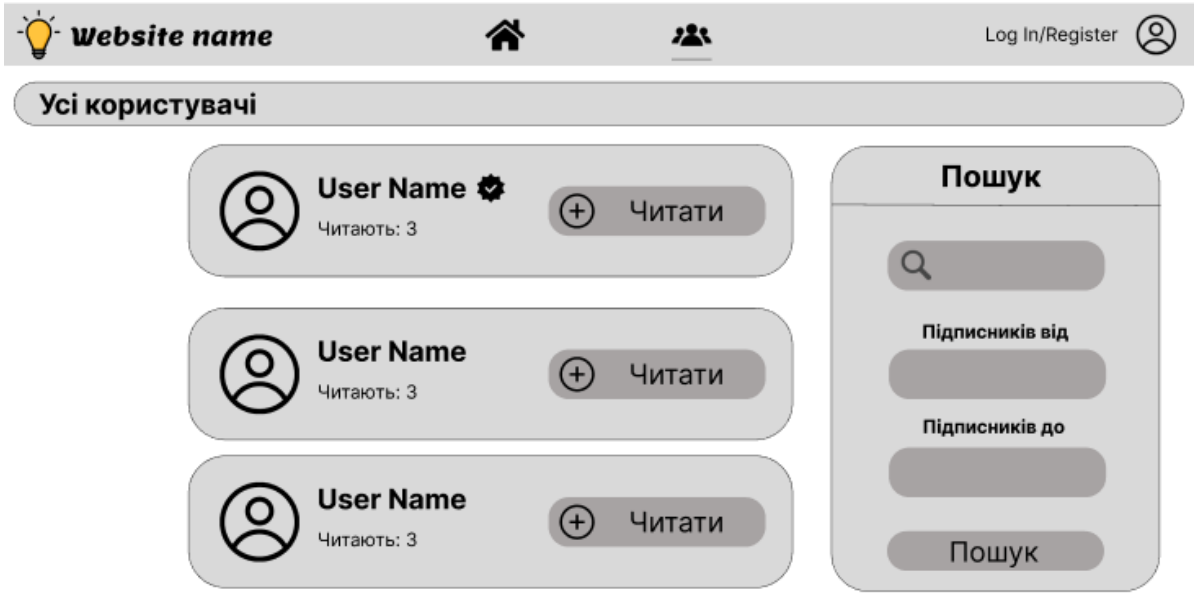


Рисунок 2.6 – Прототип вікна списку користувачів

З даного рисунку можна зрозуміти, що вікно представляє собою список усіх користувачів соціальної мережі з основною інформацію про них та додатковою кнопкою для початку відслідковування їхньої діяльності. Якщо профіль користувача є підтвердженим адміністрацією мережі, то біля його імені буде розміщено відповідну помітку, а у списку він відображатиметься вище.

Останнім вікном сайту, для якого було розроблено прототип, було вікно статистики. Його вигляд зображено на рисунку 2.7.

Рисунок 2.7 – Прототип вікна із статистикою

Доступ до даного вікна матиме лише адміністрація при авторизації під профілем з відповідною роллю. Після авторизації на панелі навігації буде відображатись кнопка для переходу до даного вікна.

У його лівій частині було розміщено блок для вибору конкретного типу інформації, наприклад, детальна статистика про користувачів, пости тощо, а у центральній відображатиметься відповідна інформація. Як приклад статистики можна навести кількість активних користувачів, кількість нових профілів користувачів за місяць, рік, тощо.

2.3 Розробка структури проекту

Першим етапом розробки структури проекту було вирішено встановити опис залежностей. Відповідно до архітектурного шаблону MVC стає зрозумілим, що основні залежності у застосунку відбуваються між моделями, представленнями та контролерами, а у зв'язку з тим, що попередньо було вирішено використовувати Entity Framework для його використання та спрощення взаємодії з базою даних структури залежностей також входить модуль під назвою «DbContext». Саме з його допомогою процес отримання інформації з базою даних буде спрощено, а сам елемент виступатиме у ролі посередника між моделями даних та контролерами, при цьому загальний принцип роботи не зміниться.

Відповідно до наведеного опису залежностей було розроблено відповідну діаграму, яку можна побачити на рисунку 2.8.

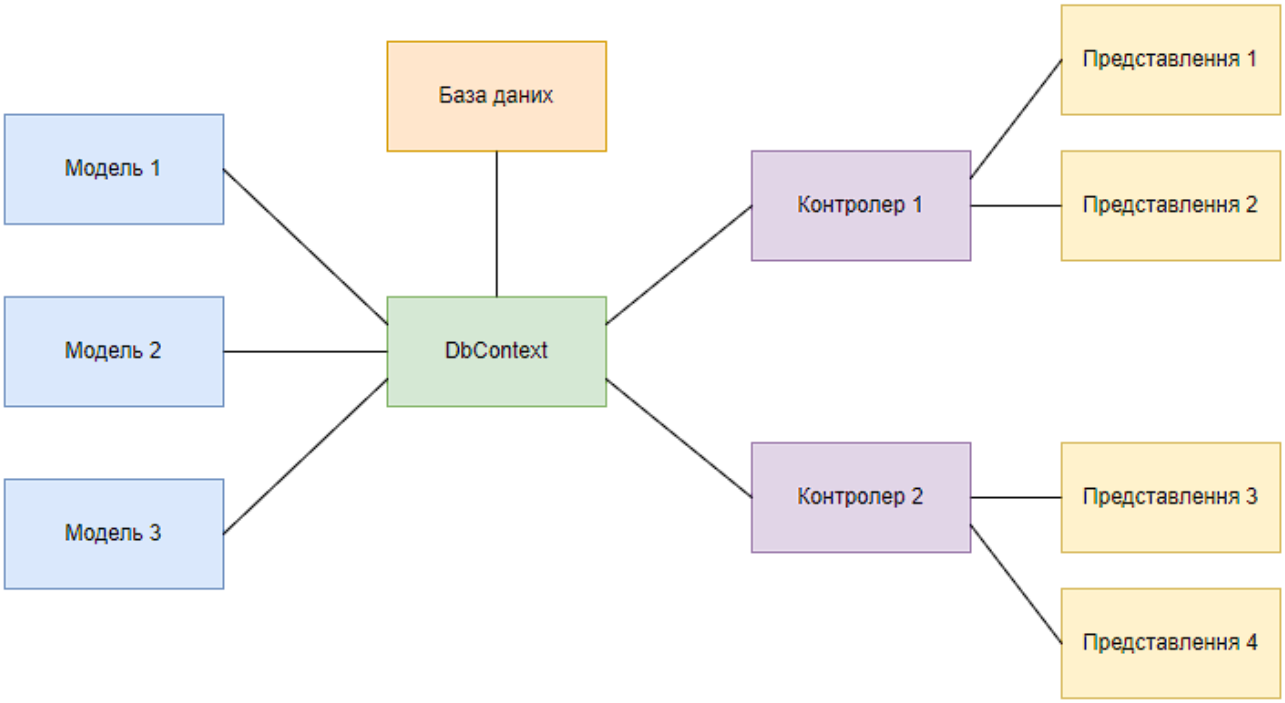


Рисунок 2.8 – Діаграма залежностей

Для того, щоб розробити якісний програмний продукт необхідно чітко розуміти основні функціональні можливості, які надаються користувачам системи. Як вже було встановлено раніше основною групою потенційних користувачів даної соціальної мережі є люди, які бажають отримувати або поширювати інформацію за допомогою системи постів. У зв'язку з цим було вирішено, що доцільно розробити контекстну діаграму верхнього рівня для процесу створення нової публікації. Відповідно до усіх встановлених вимог було розроблено відповідну діаграму, яку можна побачити на рисунку 2.9.

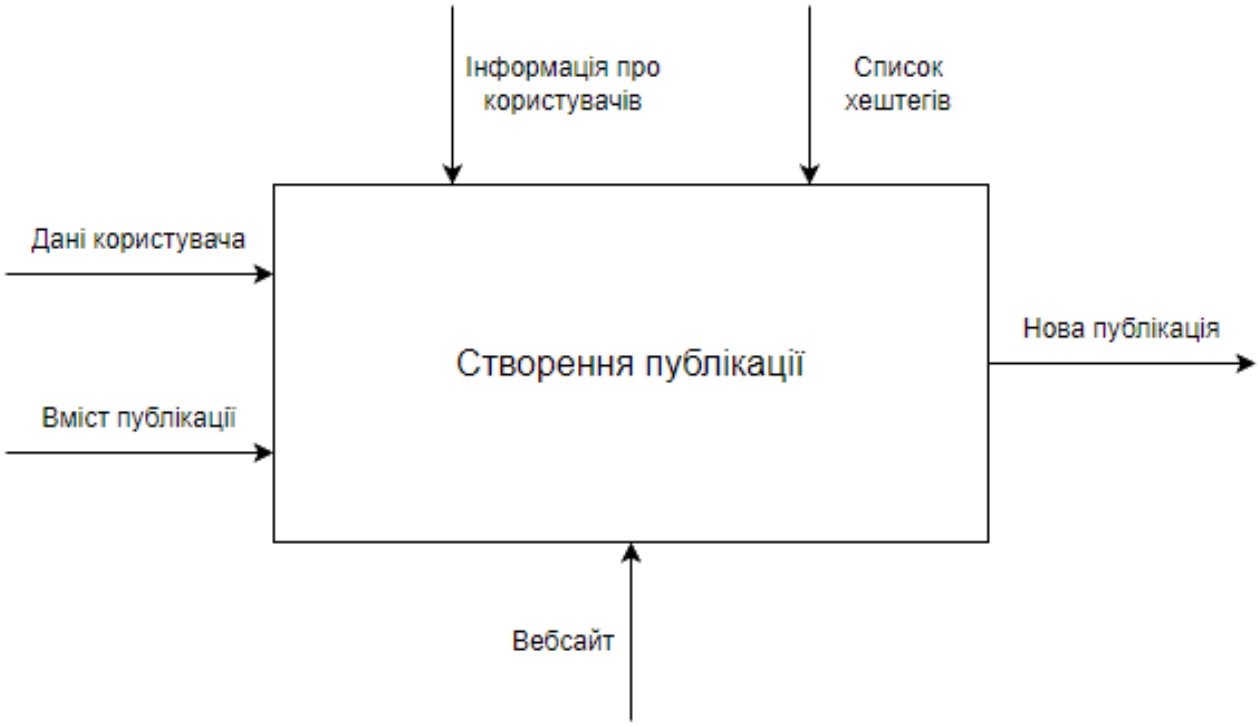


Рисунок 2.9 – Контекстна діаграма верхнього рівня

З даного рисунка видно, що для створення нового посту користувачу необхідно мати дані користувача та інформацію, яку він хоче опублікувати на даній платформі. Всі маніпуляції будуть виконуватись на веб-сайті з бази даних якого буде використано список хештегів та певну інформацію про користувача.

На основі даної контекстної діаграми було створено діаграму декомпозиції першого рівня, яка зображена на рисунку 2.10.

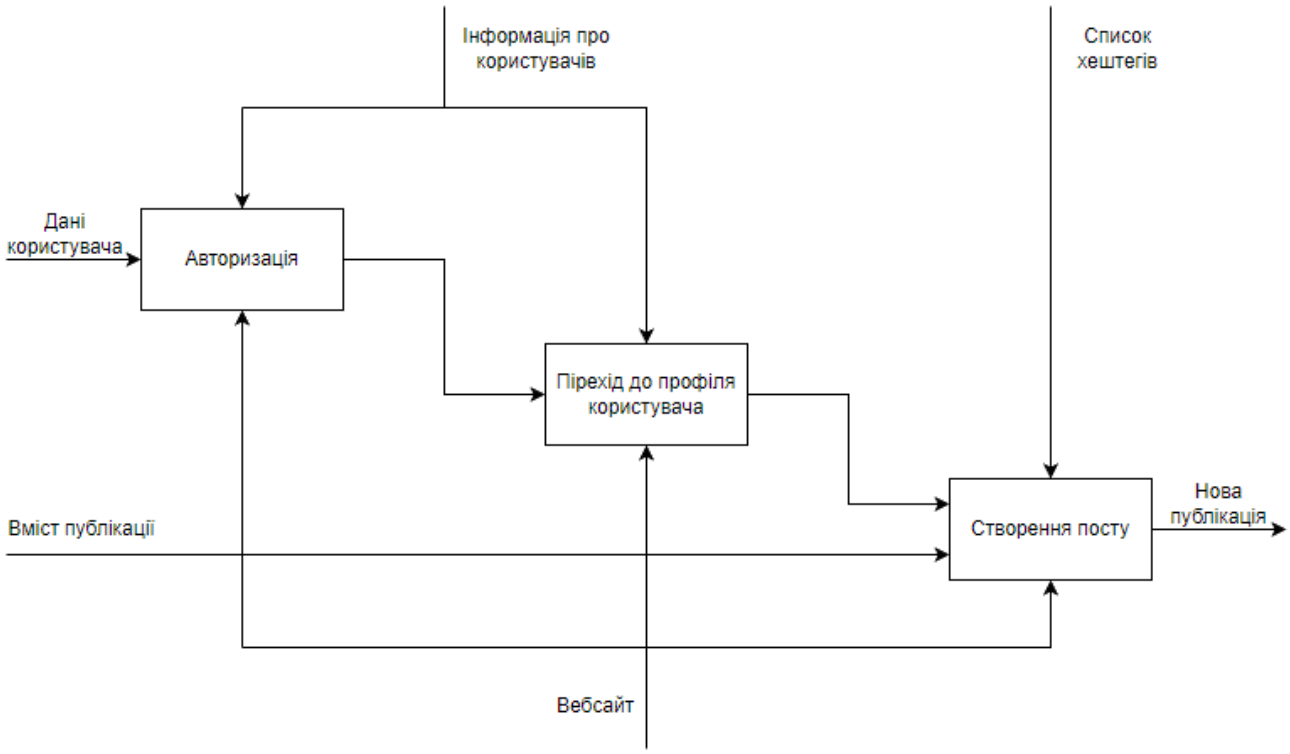


Рисунок 2.10 – Діаграма декомпозиції першого рівня

З рисунку видно, що увесь процес складається із трьох основних блоків кожен з яких відповідає за певний спектр дій і в залежності від вимог може вимагати певну інформацію як від користувача, так і від бази даних. По завершенню виконання усіх маніпуляцій до бази даних соціальної мережі буде збережено новий пост, який у подальшому можуть переглядати та оцінювати усі користувачі системи.

Особливу увагу було приділено процесу авторизації та створення посту, адже від успішності та якості їх виконання залежить результат роботи та швидкість її виконання. З даних причин наступною розробленою діаграмою є діаграма декомпозиції другого рівня блоку авторизації користувача у системі, яку можна побачити на рисунку 2.11.

Необхідно відмітити, що при її розробці вважалось, що користувач вже зареєстрований у системі, тому блок реєстрації було виключено з неї.

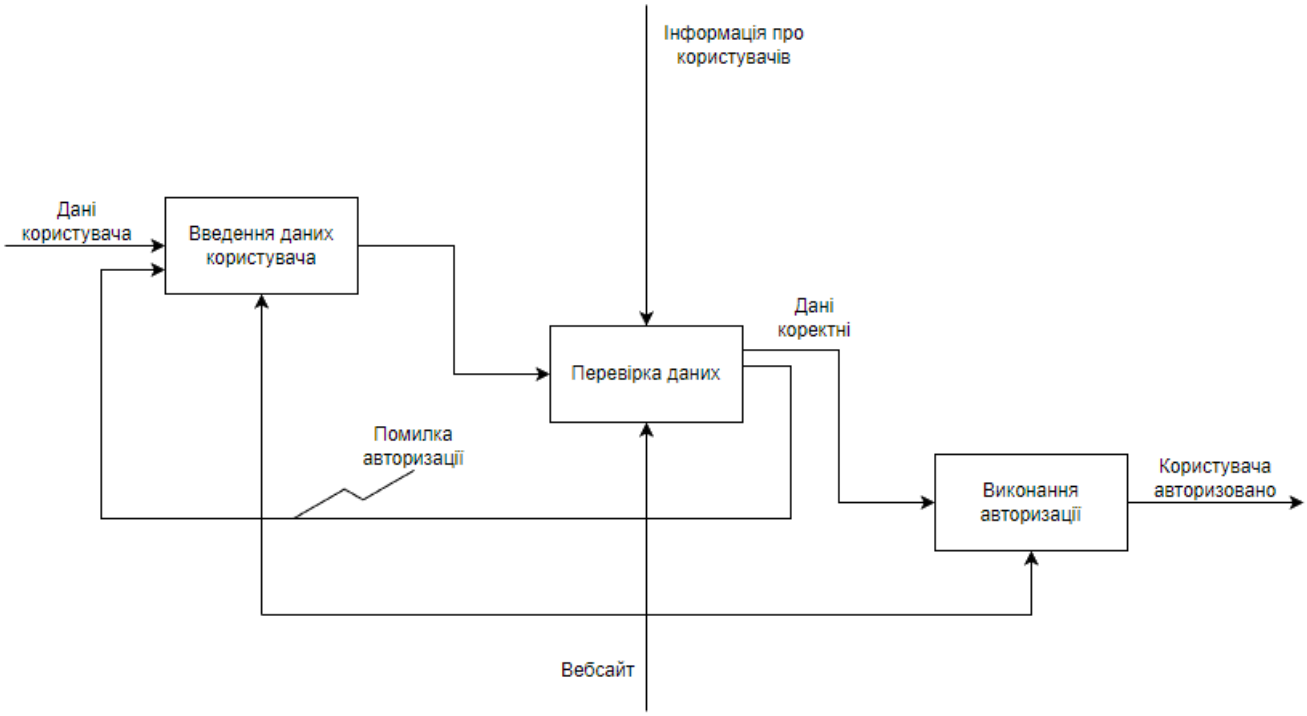


Рисунок 2.11 – Діаграма декомпозиції другого рівня блоку авторизації

Проаналізувавши дану діаграму стає очевидним, що процес авторизації складається з трьох основних блоків, але користувач безпосередньо взаємодії лише з першим.

Перший блок відповідає за введення у відповідні поля даних користувача для авторизації, а саме логін та пароль від облікового запису. Далі система порівняє введену інформацію з наявними у базі даних. При повному збігу введеної інформації із раніше записаною користувач система здійснить авторизацію користувача під відповідним обліковим записом після чого він виконання блоку авторизації вважається завершеним. Якщо ж користувач введе невірні дані, то він автоматично повернеться до попереднього блоку де йому необхідно перевірити дані на коректність введення та зробити нову спробу авторизації у соціальній мережі.

Останньою розробленою діаграмною декомпозиції другого рівня була зроблена для блоку створення поста, яка є самою важливою для отримання бажаного кінцевого результату. Її можна побачити на рисунку 3.12.

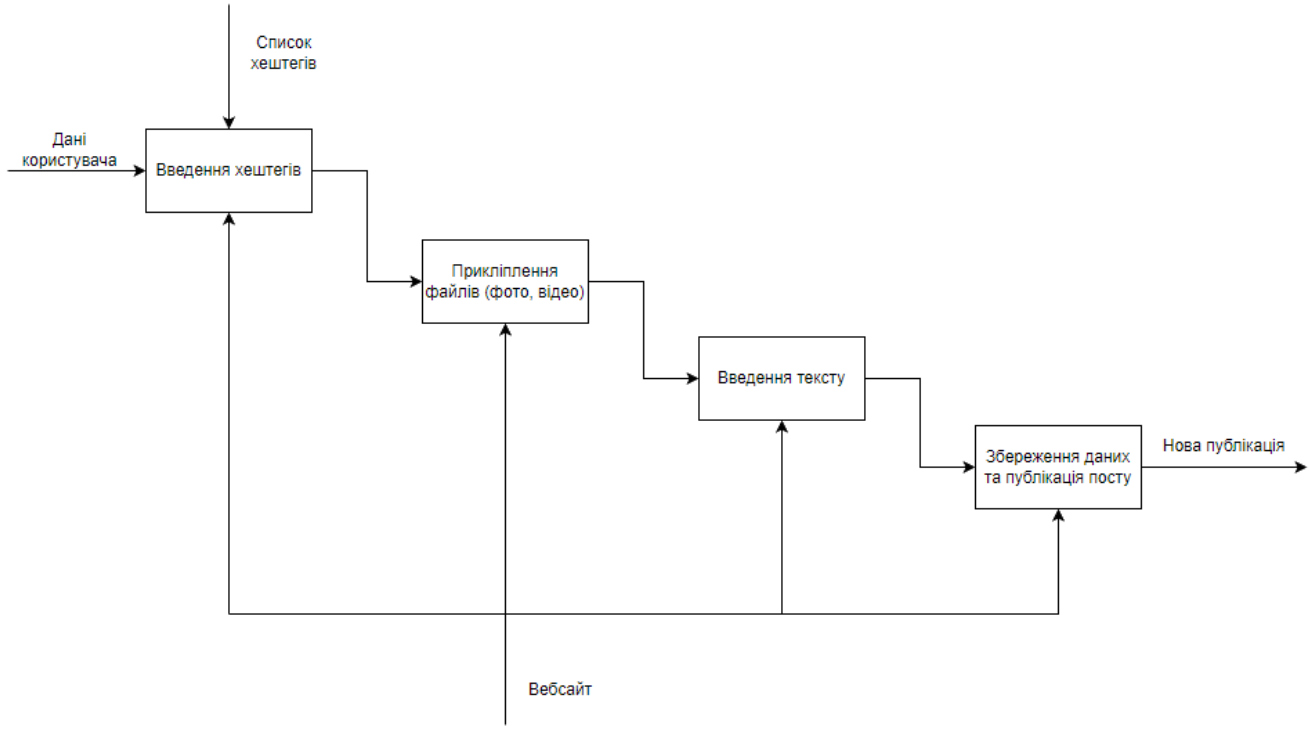


Рисунок 2.12 – Діаграма декомпозиції другого рівня блоку створення посту

З рисунку стає зрозуміло, що процес розбитий на декілька блоків, кожен з яких відповідає за певний етап створення посту. Спочатку користувачу пропонується вибрати хештеги або створити нові, які у подальшому будуть збережені. Далі користувач може прикріпити фали до посту у вигляді фото чи відео матеріалів та додати текст до публікації. Після успішного виконання даних маніпуляцій до бази даних зберігається новий пост.

Перед розробкою програмного продукту потрібно чітко усвідомлювати, яку інформацію необхідно зберігати у базі даних. З даних причин було вирішено вважати найкращим варіантом є розробка ER діаграми бази даних, яка у максимально зрозумілій формі зобразити усю інформацію, яка буде зберігатись соціальною мережею та їх тип. Для виконання поставленого завдання на основі розроблених вимог до даного програмного продукту та з врахуванням технічних обмежень було розроблено детальну ER діаграму бази даних, яку можна побачити на рисунку 2.13.

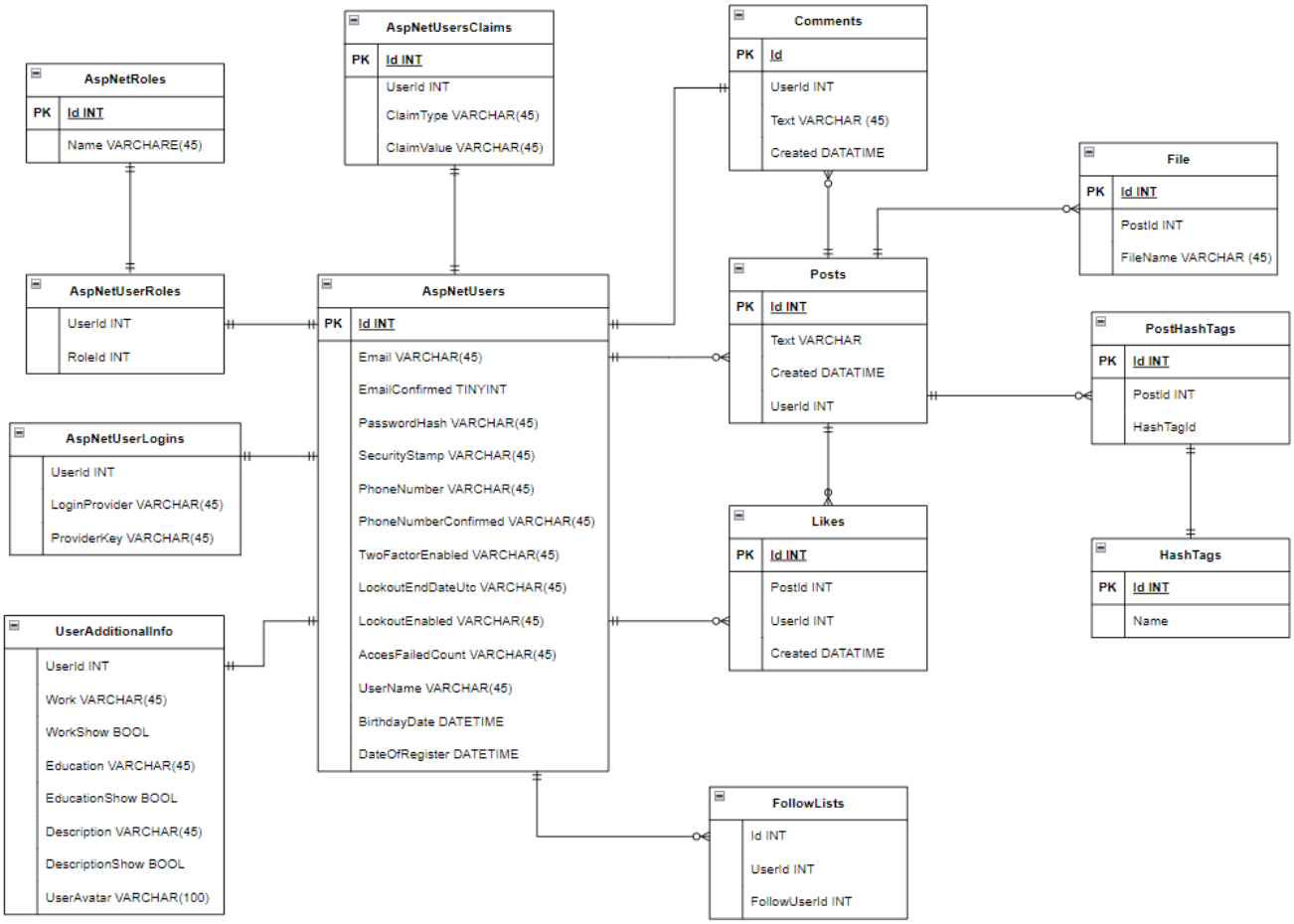


Рисунок 2.13 – ER діаграма бази даних

Умовно дану діаграму можна поділити на два основні блоки. У першому, який розміщений у правій частині рисунку будуть зберігатись усі дані, які пов’язані з користувачами системи. До них можна віднести логін, пароль, дату реєстрації, ролі у системі, тощо.

У другому блоці, на правій частині діаграми, умовно було розміщено блок, який відповідає за збереження даних пов’язаних з постами. До них можна віднести хештеги, коментарі, прикріплені файли, тощо.

За допомогою даних діаграм було покращено розуміння структури програмного продукту та здійснено його подальшу розробку.

2.4 Висновки

Першим етапом написання даного розділу було встановлення технологій та архітектурних шаблонів для даного програмного продукту. Для цього було здійснено аналіз декількох технологій на основі чого було здійснено вибір фреймворку ASP.NET Core з використанням архітектурних шаблонів проектування MVC та клієнт-серверна архітектура для яких було розроблено діаграми, які демонструють принцип їхньої роботи. Також передбачається можливість використання AJAX для динамічного оновлення наповнення сторінок та Entity Framework для спрощення взаємодії з базою даних.

На наступному кроці було розроблено прототип інтерфейсу користувача для основних вікон соціальної мережі із здійсненням їхнього детального опису. У майбутньому це допоможе розробити уніфікований інтерфейс користувача, що допоможе спростити взаємодію з ним та покращити та покращить користувацький досвід його використання.

Коли основні технології були встановлені, а прототип інтерфейсу розроблений, то доцільно було здійснити опис структури програмного продукту відповідно до наявних обмежень та вимог.

Для цього відповідно до використання технології Entity Framework та архітектури MVC було розроблена діаграма, яка демонструє залежності між різними компонентами системи.

Для кращого розуміння основних функціональних можливостей програмного продукту додатково було розроблено контексту діаграму верхнього рівня та декілька діаграм декомпозиції для процесу створення застосунку. На кінець було розроблена ER-діаграма бази даних для відображення наявної у системі реалізації.

Виконання даного розділу допоможе на етапі розробці програмного забезпечення на різних етапах, наприклад розробка структури, бази даних чи інтерфейсу користувача.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Реалізація бази даних та вікон програми

На даному етапі виконання кваліфікаційної роботи можна вважати, що основні етапи з підготовки до подальшої програмної реалізації завершені, а на їх основі можна здійснювати подальшу розробку відповідно до раніше встановлених вимог та технічних обмежень.

Для успішного початку виконання розробки програмного забезпечення було здійснено кілька важливих кроків підготовки. На першому з них було вирішено доцільним розробити базу даних для даної соціальної мережі, використовуючи як основу для неї раніше створену ER-діаграму бази даних.

Для цього опираючись на архітектурний шаблон MVC було створено ряд моделей кожна з яких має назву та містить дані відповідно до передбачених таблиць у базі даних. На рисунку 3.1 можна побачити вигляд відповідний код для моделі Post.

```
public class Post
{
    [Key]
    public int Id { get; set; }

    [Required]
    public string UserId { get; set; }
    public string? Text { get; set; }

    [Required]
    public DateTime CreatedDateRime { get; set; }

    public virtual MyMainDiplomProjectUser User { get; set; }
    public virtual List<HashTags>? PostHashTags { get; set;}
    public virtual List<Likes>? Likes { get; set; }
    public virtual List<Files>? Files { get; set; }
    public virtual List<Comments>? Comments { get; set; }
}
```

Рисунок 3.1 – Контекст даних

З даного фрагменту коду можна зрозуміти, що окрім полів використовуються атрибути перед назвами змінних. Вони надають додаткову

інформацію про параметри відповідних полів моделі та допомагають встановити певні вимоги та обмеження на дані, що зберігаються в цих полях.

Наприклад, [Key] вказує на те, що певне поле є первинним ключем сутності. Він визначає унікальний ідентифікатор для кожного запису в таблиці. Атрибут [Required] вказує на обов'язкову наявність значення у даному полі.

У попередніх розділах було встановлено, що доступ до бази даних буде забезпечено за допомогою Entity Framework. Цей фреймворк дозволяє спростити взаємодію з базою даних, надаючи зручні засоби для виконання операцій збереження, оновлення, видалення та отримання даних. В якості системи керування базами даних для даного проекту було обрано MySQL. Він є потужним та надійним інструментом, що дозволяє ефективно управляти реляційними базами даних, забезпечувати безпеку і цілісність даних.

Наступним кроком створення та налаштування бази даних є реалізація посередника у вигляді контексту даних. Контекст даних є основним класом, який забезпечує взаємодію з базою даних та відповідає за ініціалізацію з'єднання з базою даних, встановлення конфігураційних параметрів, тощо. З його кодом можна ознайомитись на рисунку 3.2.

```
public class MyMainDiplomProjectDbContext : IdentityDbContext<MyMainDiplomProjectUser>
    {

        public MyMainDiplomProjectDbContext(DbContextOptions<MyMainDiplomProjectDbContext> options)
            : base(options)
        {
        }

        public DbSet<Comments> Comments { get; set; }
        public DbSet<Files> Files { get; set; }
        public DbSet<FollowList> FollowLists { get; set; }
        public DbSet<HashTags> HashTags { get; set; }
        public DbSet<Likes> Likes { get; set; }
        public DbSet<Post> Posts { get; set; }

        protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);
        }

    }
```

Рисунок 3.2 – Контекст даних

Він виступатиме посередником для доступу до інформації, яка зберігається у базі даних та між функціоналом, наприклад, створення, редагування, видалення чи перегляду даних за допомогою контролерів.

З його допомогою у проекті можна зручно виконувати різні операції з базою даних за допомогою методів, які він надає. Наприклад, контексту даних допомагає легко створювати нові записи в базі даних, редагувати та оновлювати їх, тощо. Контекст даних напряму взаємодіє з контролерами, які відповідають за обробку різних запитів від користувача та маніпулювання даними через моделі. Окрім цього контекст даних дозволяє забезпечити цілісність даних та допомагає розподілити відповідальність між різними компонентами системи.

Після завершення усіх підготовчих робіт та створення необхідної структури, настала черга створити базу даних. Цей процес зазвичай виконується за допомогою механізму міграції бази даних, що дозволяє автоматично створити базу даних та її таблиці на основі раніше визначених моделей у програмі, а також здійснити ряд необхідних початкових налаштувань.

Процес міграції бази даних передбачає надання можливостей зберігати, здійснювати контроль за версіями та керувати структурою бази даних у зручний для користувача спосіб. У переважній більшості випадків він здійснюється за допомогою використанням міграційних скриптів, які містять в собі інструкції для створення, зміни або видалення таблиць, стовпців, індексів, тощо.

Під час виконання міграції бази даних система автоматично перевіряє наявність змін у моделях даних та порівнює їх із наявною схемою. Вона виявляє різницю між поточним станом та очікуваний результатом після чого автоматично застосовує необхідні зміни до бази даних.

Однією з основних переваг такої реалізації є можливість вносити різноманітні зміни та оновлення до бази даних, наприклад при модифікації програми, додання нових полів, тощо. Таким чином розробники можуть вносити різноманітні зміни безпосередньо в процесі розробки та підтримки продукту. Це дозволяє легко адаптувати базу даних до вимог.

Відповідно до вимог було встановлено, що всі вікна сайту повинні мати однорідний зовнішній вигляд з ідентичною стилізацією усіх наявних елементів навігації, панелями та іншими компонентами. Такий спосіб реалізації дозволяє забезпечити єдиний стиль інтерфейсу користувача, спрощує навігацію між вікнами та покращує загальний досвід взаємодії з програмним продуктом.

З цією метою було вирішено на початковому етапі розробки створити зовнішній вигляд для усіх майбутніх сторінок та використовувати його як основу для подальшого усього подальшого інтерфейсу користувача.

Для досягнення цієї мети було виконано ряд попередніх приготувань, які спрощують процес створення користувацького інтерфейсу.

Першим етапом, який було вирішено реалізувати є розроблення основних стилів для різних елементів вікон, що активно використовується у інтерфейсі.

Даний етап передбачав створення візуального оформлення для кнопок, форм, заголовків, списків та інших компонентів інтерфейсу користувача системи з метою їхньої уніфікації та полегшення подальшого використання.

З допомого використання даного підходу до розробки інтерфейсу користувача кількість можливих проблем та помилок при роботі з відповідним програмним продуктом, які виникають зі сторони користувача у зв’язку з незрозумілістю інтерфейсу, недостатніми знаннями чи з інших причин можна значно зменшити.

Необхідно відмітити, що даний підхід також дозволяє ефективно використовувати створені стилі, що прискорює розробку нових сторінок, оскільки багато компонентів та їх стилів можна повторно використовувати, а при необхідності їх можна швидко модифікувати.

Далі, коли усі основні стилі були завершені та готові до подальшого використання, було вирішено розробити загальний макет для усіх сторінок програмного продукту. У випадку даної роботи основним елементом макету є навігаційне меню з допомогою якого користувачі здійснюватимуть перехід до інших вікон програми. Код відповідного вікна можна побачити на рисунку 3.3.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - Diplom</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
    <link rel="stylesheet" href="~/Diplom.styles.css" asp-append-version="true" />
    <link rel="stylesheet" href="~/css/styles.css" asp-append-version="true" />

    <link rel="stylesheet" href="./css/styles.css">
    <link href="https://fonts.googleapis.com/css?family=Open+Sans:300,400,600,700&display=swap" rel="stylesheet">

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.3/css/all.min.css" />
    <link href="https://netdna.bootstrapcdn.com/font-awesome/3.2.1/css/font-awesome.css" rel="stylesheet">
</head>
<body>
    <header>
        <div class="logo">
            <a href="#" class="logo">
                <i class="fas fa-lightbulb"></i><span align-items="center">Lets Study</span>
            </a>
        </div>
        <div>
        </div>
        <nav>
            <ul>
                <li><a href="~/Home/Index"><i class="fa fa-home"></i></a></li>
                <li><a href="~/Posts/UserProfile/@User.Identity.Name"><i class="fa fa-users"></i></a></li>
                @if (User.Identity.IsAuthenticated)
                {
                    <li><a href="~/UserAdditionalInfo/Profile"><i class="fa fa-user"></i></a></li>
                    <li><a href="~/UserAdditionalInfo/Setings"><i class="fa fa-cog"></i></a></li>
                }
                @if(User.Identity.AuthenticationType == "Admin")
                {
                    <li><a href="~/UserAdditionalInfo/AdminPanel"><i class="fa fa-cog"></i></a></li>
                }
            </ul>
        </nav>
        <partial name="_LoginPartial.cshtml" />
    </header>

    <div class="container">
        <main role="main" class="pb-3">
            @RenderBody()
        </main>
    </div>
```

Рисунок 3.3 – Макет сторінок

Даний рисунок демонструє використання HTML розмітки разом з застосуванням мови програмування C#. Перед початком написання коду було розміщено символ «@» для позначення вбудованих кодових блоків або виразів, що дозволяють використовувати відповідний код. Цей символ є частиною базового синтаксису Razor, що використовується у фреймворку ASP.NET Core.

У даному контексті символ «@» використовується перевірки, чи є користувач авторизований у системи на момент здійснення запиту та перевірка до якої ролі відноситься його особистий профіль. Залежно від значень отриманих у ході перевірки, на даній панелі відображаються додаткові кнопки.

Такий підхід до реалізації вікон програмного продукту дозволяє детально налаштовувати відображення елементів в залежності від прав доступу, ролі користувача чи інших параметрів. Також використання вбудованого коду дозволяє динамічно контролювати вміст, наповнення і поведінку сторінок в залежності від умов, що спрощує розробку багатофункціональних веб-додатків.

У нижній частині коду була розміщена директива «@RenderBody». Вона використовується у макеті сторінки в ASP.NET Core MVC для вказівки на точне місце розташування, де необхідно вставляти відповідний вміст представлень, які використовують даний шаблон як базовий. Це дозволяє динамічно відображати різноманітний вміст у макеті, залежно від конкретного представлення, що необхідно відобразити користувачу після виконання запиту.

Для кращого розуміння зовнішнього вигляду розробленого коду було наведено вигляд вікна авторизація, яке можна побачити на рисунку 3.4.

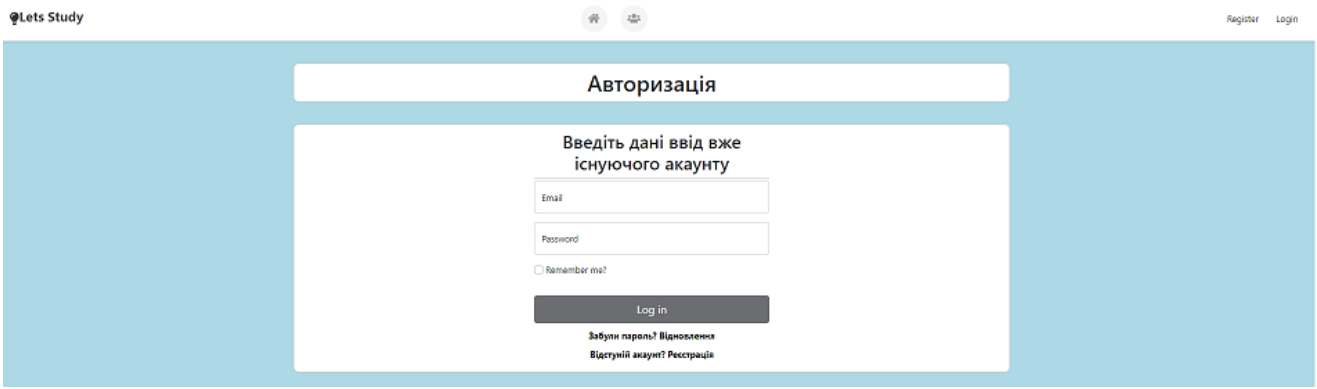


Рисунок 3.4 – Вигляд вікна авторизації

На даному етапі можна вважати, що усі основні початкові задачі, які були поставлені на розробку даного програмного продукту були успішно виконані відповідно до раніше встановлених вимог та обмежень, що означає подальший перехід до розробки усього основного функціоналу для соціальної мережі за допомогою використання контролерів та відповідно до їхніх методів інших представлень, які необхідні для подальшої коректної роботи усієї системи та відображення різного виду інформації.

3.2 Програмна реалізації основних модулів системи

Наступним і одночасно найважливішим етапом для успішного завершення виконання даної кваліфікаційної роботи є розробка різноманітних функціональних модулів. Ці модулі відіграють ключову роль у забезпеченні основного функціоналу соціальної мережі, дозволяючи користувачам реєструватись та авторизуватись у системі, створювати та переглядати нові пости, взаємодіяти з іншими користувачами та багато іншого.

Як вже було встановлено раніше при розробці даного програмного продукту передбачалось використання архітектурного шаблону проектування MVC. Таким чином даний програмний продукт містить ряд контролерів кожен з яких містить у собі різноманітний функціонал та виступає посередником між моделями даних та представленнями.

Окрім архітектурного шаблону MVC даний програмний продукт використовує клієнт-серверну архітектуру. Це означає, що усі обов'язки, які лежать на даній соціальній мережі, були розподілені між клієнтською та серверною частиною, як двома основними взаємодіючими компонентами сайту.

Серверна сторона відповідає за обробку бізнес-логіки програми, зберігання та обробку наявних даних, реєстрацію та авторизацію користувачів у системі, а також виконання різноманітних запитів з сторони клієнта.

Таким чином використання даного шаблону забезпечує високу надійність та безпеку обробки даних, а також реалізацію основних функцій програми.

Після додаткового опису відповідної структури проекту, яка використовувалась для розробки основних функціональних блоків, було вирішено, що є доцільним розглянути найголовніші функціональні можливості користувачів, використання яких буде відбуватись на постійній основі.

Першими важливими функціональними можливостями програмного забезпечення які було реалізовано у соціальній мережі є механізм авторизація та реєстрація у системі.

Для реалізації даної системи було використано вбудовану бібліотекою «Identity», яка надає функціональні можливості для роботи з аутентифікацією, авторизацією та управлінням користувачами для різноманітних видів веб-додатків. Також з її допомогою було впроваджено систему користувачів з різними функціями, наприклад, реєстрації, входу, керування ролями та іншими пов'язаними задачами.

Фрагмент коду, який використовується для реєстрації нових користувачів у системі в даному програмному продукті можна побачити на рисунку 3.5.

```
public async Task OnGetAsync(string returnUrl = null)
{
    ReturnUrl = returnUrl;
    ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
}

public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl ??= Url.Content("~/");
    ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
    if (ModelState.IsValid)
    {
        var user = CreateUser();
        user.userNikName = Input.userNikName;
        user.dateOfBirthday = Input.dateOfBirthday;
        user.registerDate = DateTime.Now;

        await _userStore.SetUserNameAsync(user, Input.Email, CancellationToken.None);
        await _emailStore.SetEmailAsync(user, Input.Email, CancellationToken.None);
        var result = await _userManager.CreateAsync(user, Input.Password);

        if (result.Succeeded)
        {
            _logger.LogInformation("User created a new account with password.");

            var userId = await _userManager.GetUserIdAsync(user);
            var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
            code = WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(code));
            var callbackUrl = Url.Page(
                "/Account/ConfirmEmail",
                pageHandler: null,
                values: new { area = "Identity", userId = userId, code = code, returnUrl = returnUrl },
                protocol: Request.Scheme);

            await _emailSender.SendEmailAsync(Input.Email, "Confirm your email",
                $"Please confirm your account by <a href='{HtmlEncoder.Default.Encode(callbackUrl)}'>clicking here</a>.");

            if (_userManager.Options.SignIn.RequireConfirmedAccount)
            {
                return RedirectToPage("RegisterConfirmation", new { email = Input.Email, returnUrl = returnUrl });
            }
            else
            {
                await _signInManager.SignInAsync(user, isPersistent: false);
                return LocalRedirect(returnUrl);
            }
        }
        foreach (var error in result.Errors)
        {
            ModelState.AddModelError(string.Empty, error.Description);
        }
    }

    return Page();
}
```

Рисунок 3.5 – Фрагмент коду для реєстрації

Необхідно відмітити, що даний спосіб реалізації передбачав наявність базової інформації про користувачів програмного продукту, тому відповідна система було модифікована для додання нових даних до таблиці, а саме ім’я користувача, день народження та дата реєстрації нового профілю.

Після виконання даних маніпуляцій кількість інформації, яку необхідно вказати для подальшої успішної реєстрації збільшилась, тому до вікна та процесу створення нового профілю користувача було зроблено ряд відповідних змін. У результаті даних маніпуляцій вікно реєстрації користувача у системі було модифіковано, що можна побачити при ознайомлені з рисунком 3.6.

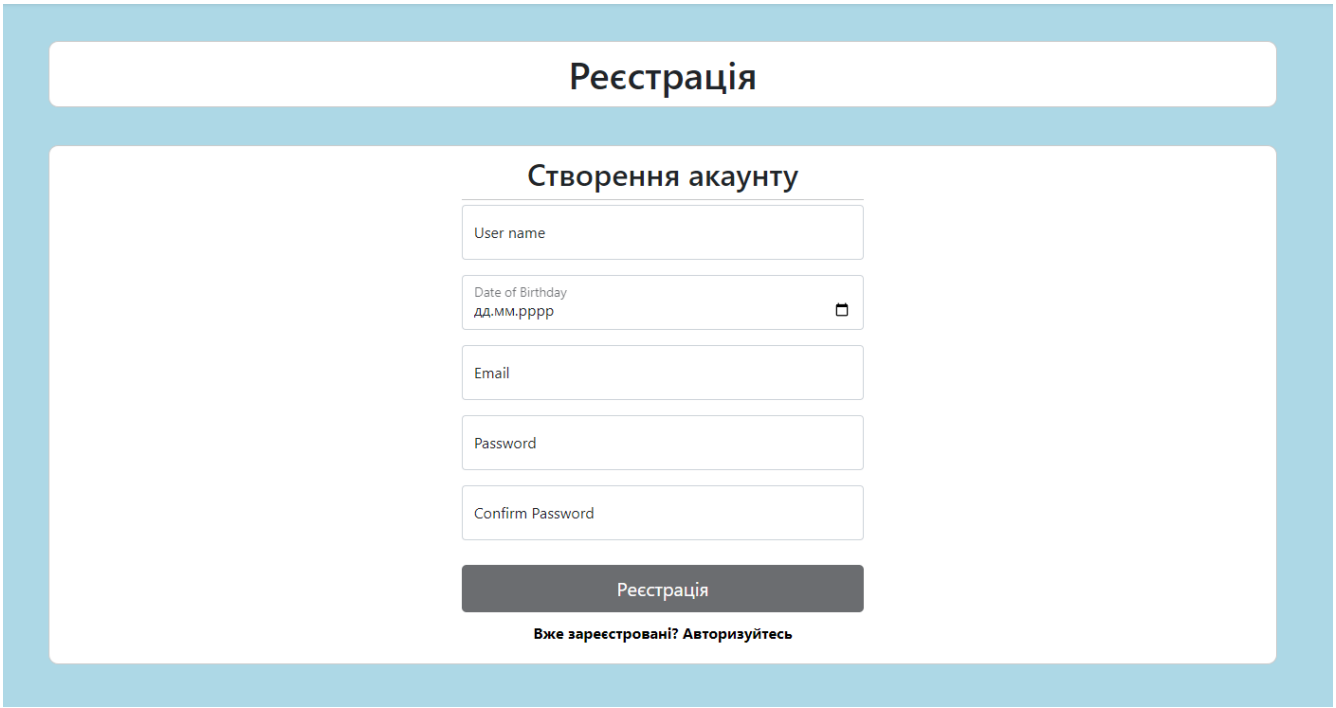


Рисунок 3.6 – Вікно реєстрації користувача

Наступною важливою функціональністю програмного продукту є можливість створення нових публікацій користувачами, які мають активну авторизацію в системі. Ця можливість надає користувачам спосіб, щоб ділитися своїми думками, ідеями та враженнями з іншими учасниками соціальної мережі.

Наступною важливою функціональністю програмного продукту є можливість створення нових публікацій користувачами, які мають активну авторизацію в системі на момент створення посту. Ця можливість надає користувачам спосіб, щоб ділитися своїми думками, ідеями та враженнями з іншими учасниками соціальної мережі.

При створенні нового посту користувачеві необхідно вказати основний вміст будь-якої публікації. Обов'язковим для введення параметром є текст посту, де користувач може вільно висловити свої думки, враження, переживання з певної події чи ситуації, або поділитися з іншими користувачами цікавою, на його думку, інформацією.

Крім текстового наповнення користувач може прикріпити до посту хештеги, що допомагають класифікувати та швидко знаходити пов'язані за змістом та тематико публікації. Додатково передбачена можливість використовувати власні рисунки для візуального вираження свого повідомлення чи доповнення вже наданої інформації. Використання рисунків є опціональним параметром, що означає, що пост може бути створений без зображень, або може містити кілька різних рисунків.

Поміж параметрів, які також відносяться до постів і зберігаються в базі даних, але додаються автоматично залежно від різних факторів, варто згадати особистий ідентифікатор користувача, який слугує унікальним кодом автора публікації, а також точну дату та час публікації відповідного поста. Ці дані є важливими для відстеження авторства та хронології публікацій, які розміщуються у соціальній мережі.

Також варто відзначити, що кожен пост додатково має різноманітні зв'язки з іншими даними, які зберігаються в інших таблицях бази даних. Зокрема, до цих даних відносяться вподобайки та коментарі, які пов'язані з конкретним постом.

Вподобайки відображають кількість разів, коли користувачі натиснули кнопку «Подобається» для відповідної публікації, яка зображена у вигляді

рисунку піднятого до гори великого пальця, що дозволяє визначити популярність публікації.

Коментарі, з свого боку, дають можливість користувачам обговорювати пост та взаємодіяти між собою, розміщуючи свої відгуки та думки.

Для зручності розуміння програмної реалізації, як створюються нові пости, на рисунку 3.7 наведений фрагмент коду, який відповідає за цей процес.

```
[Authorize]
[HttpPost]
public async Task<IActionResult> Store(PostViewModel model)
{
    if (ModelState.IsValid)
    {
        var post = new Post
        {
            UserId = Convert.ToString(_context.Users.Where(i => i.Email == User.Identity.Name).FirstOrDefault().Id),
            Text = model.Text,
            CreatedDateRime = DateTime.Now,
            Files = new List<Files>(),
            PostHashTags = new List<HashTags>()
        };

        if (model.HashTags != null)
        {
            foreach (var tag in model.HashTags)
            {
               if(_context.HashTags.Where(i => i.Name == tag).Count() > 0)
               {
                    post.PostHashTags.Add(_context.HashTags.Where(i => i.Name == tag).FirstOrDefault());
               }
               else
                {
                    post.PostHashTags.Add(new HashTags() { Name = tag });
                }
            }


        }
```

Рисунок 3.7 – Фрагмент коду створення нового поста

Проаналізувавши даний рисунок можна зрозуміти, що можливістю створення нових постів можуть користуватись лише авторизовані користувачі. Це досягнуто за допомогою використання атрибуту [Authorize], який додається перед ім'ям методу контролер що вказує на необхідність автоматичної перевірки авторизації перед виконанням самого методу контролера.

У випадку спроби створення посту неавторизованим користувачем, система автоматично перенаправить його на сторінку авторизації, де він зможе здійснити авторизацію, або створити новий обліковий запис.

Відповідно до усіх описаних вимог до створення нової публікації було розроблено представлення з яким можна ознайомитись на рисунку 3.8.

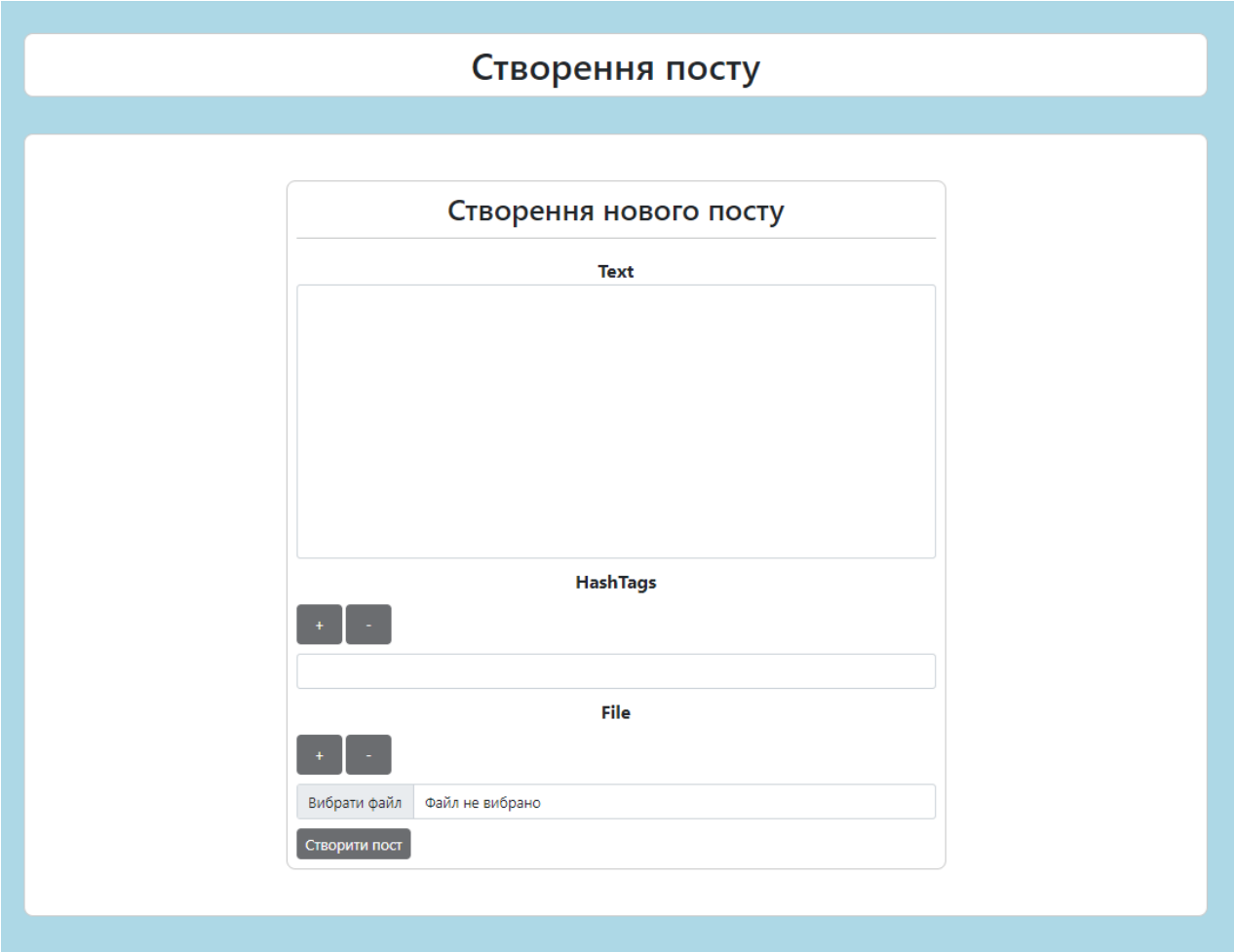


Рисунок 3.8 – Інтерфейс вікна створення нового поста

Наступною важливою частиною програмної реалізації яку необхідно детальніше описати є відкриття великого посту при натисканні на відповідний пост у стрічці новин чи у профілі користувача.

Даний функціонал використовується з двох основних причин, а саме при необхідності перегляду саме одного конкретного посту у кращій якості та надання доступ до можливості створення коментарів, що відсутнє упри звичайному перегляді постів. Така реалізація доступу до функціоналу використовується з метою покращення читабельності усіх постів та спрощення їхнього зовнішнього вигляду. Також це допомагає приховати коментарі від користувачів, які можуть бути не цікаві користувачам. Відповідне вікно можна побачити на рисунку 3.9.

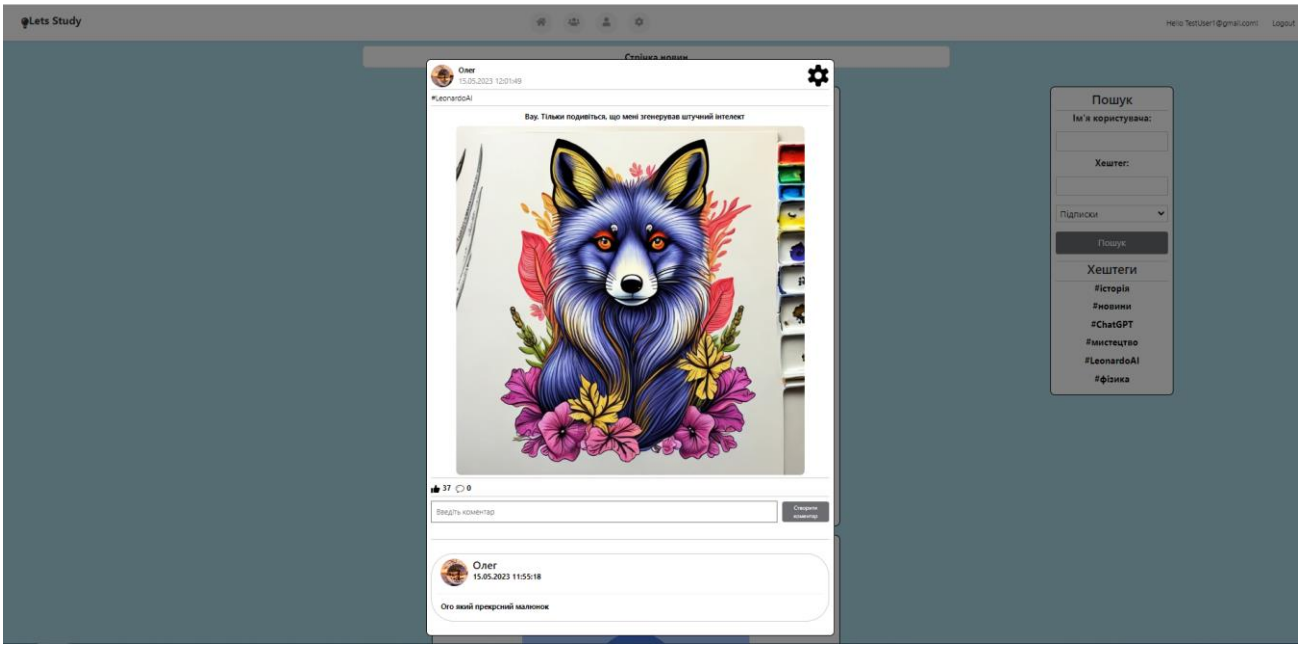


Рисунок 3.9 – Вікно відкриття посту

З даного рисунку можна побачити, що коментарі та блок їхнього створення були розміщені у нижній частині вікна. Це забезпечує зручний доступ до функціоналу коментування, а розміщення блоку у нижній частині забезпечує їхню логічну послідовність і зберігає зорову єдність сторінки.

Варто зазначити, що відкриття посту не відбувається у новому вікні, а відображається поверх поточної сторінки. Після завершення перегляду посту закриття вікна призведе до повернення користувача до того місця, де він перебував до його подальшого відкриття. Це покращує користувацький досвід та дозволяє зручно здійснювати переходи між постами та взаємодіяти з ними без втрати загального контексту.

Даний функціонал був успішно реалізований за допомогою використання технології Ajax. Вона дозволяє динамічно оновлювати окремі елементи різних представлень без необхідності повного перезавантаження всієї сторінки.

Для впровадження даного функціоналу у макеті, що використовується на всіх сторінках програмного продукту, були додані відповідні файли Ajax, які необхідні для коректної роботи даної технології.

Також були підключені два скрипти, які відповідають за відкриття та закриття вікна перегляду публікації. За допомогою даних скриптів здійснено можливість відображати даний великий пост поверх поточної сторінки та зручно здійснювати маніпуляції ним. Для детального ознайомлення та перегляду відповідного коду даних скриптів можна звернутись до рисунку 3.10.

```
<script>

    function PopUpShow() {
        $("#PopUp").fadeIn();
        $("body").attr("style", "overflow-y: hidden;");
    }

    function PopUpHide() {
        $("#PopUp").fadeOut();
        $("body").attr("style", "overflow-y: scroll;");
    }

</script>
```

Рисунок 3.10 – Скрипти для відкриття вікна

На основі поданого коду на даному рисунку можна зрозуміти, що до проекту було додано дві функції на мові програмування JavaScript, кожен з яких відповідає за відкриття та закриття нового вікна поверх вже відкритого при натисканні на відповідний елемент навігації розробленого інтерфейсу користувача соціальної мережі.

Наступний етап реалізації даної системи передбачав внесення змін до HTML коду сторінки, який відповідає за вікно перегляду посту, у якому кожному тегу було прописано ідентифікатор, які надають можливість звернення до конкретних елементів в коді за допомогою мови програмування JavaScript, або за потреби та наявності до інших скриптових мов для подальшої маніпуляції

та зміни їхніх властивостей.. З відповідним HTML код можна детально ознайомитись на рисунку 3.11.

```
<div id="PopUp" style=" background: rgba(0, 0, 0, 0.5); width: 100%; height: 100%;
       position: fixed  ; top: 0; left: 0; overflow-y: scroll; display: none; z-index: 9999999">
    <div class="container mt-5">
        <div class="row">
            <div class="col-12 pt-4 pb-5 mt-5" id="PopUpContent" style="color: white;">
            </div>
            <button class="col-1 offset-9 btn btn-primary" style="position: fixed; top: 10%;"
onclick="PopUpHide();">✖</button>
        </div>
    </div>
</div>
```

Рисунок 3.11 – HTML код блоку відображення

Останнім кроком реалізації даної функціональної можливості є додання посилання елементам при натисканні на яких буде відкриватись вікно.

У випадку використанні посилання при натисканні на пост було вирішено, що його доцільно розмістити на блок з основним наповненням публікації, а саме блоку, у якому розміщено тексту та рисунки. Зроблено було це у зв’язку з тим, що даний блок не містить ніяких тегів, які не можуть бути вкладені у тег посилання. Відповідний код можна побачити на рисунку 3.12.

```
        <a data-ajax-success="PopUpShow();" data-ajax="true" data-ajax-mode="replace" data-ajax-update="#PopUpContent"
href="/Posts/BigPost/@post.Id">
            <div class="post-content">
                <div class="post-text">
                    <p>@post.Text</p>
                </div>
                @if (post.Files.Any())
                {
                    <img class="post-image" src="@Url.Content(post.Files.First().FileName)" alt="Post image" />
                }

            </div>

            <div class="post-stats">
                <button class="post-button">
                    <img class="heart-icon" src="https://cdn0.iconfinder.com/data/icons/freebies-2/24/like-1-48.png"
alt="Heart icon">
                </button>

                <span class="heart-count">37</span>
                <button class="post-button">
                    <img class="comment-icon" src="https://cdn3.iconfinder.com/data/icons/linecons-free-vector-icons-
pack/32/bubble-48.png" alt="Comment icon">

                </button>

                <span class="comment-count">0</span>
            </div>
        </a>
    </div>
```

Рисунок 3.12 – HTML код для відкриття вікна

Проаналізувавши даний рисунок можна зрозуміти, що відповідний тег посилання вказує на метод контролера постів, до якого передається ідентифікатор посту, який пробує відкрити користувач у збільшеному вигляді.

Це забезпечує зв'язок між посиланням і відповідним постом в базі даних соціальної мережі. Після передачі відповідного ідентифікатора посту, відбувається відкриття представлення.

Також важливо зазначити, що аналогічним чином відкриваються вікна для редагування інформації про користувача на відповідній сторінці налаштувань профілю користувача. Детально ознайомитись з його зовнішнім виглядом можна на рисунку 3.13.

Рисунок 3.13 – Вікно параметрів користувача

З даного рисунка видно, що вікно складається з багатьох незалежних блоків, які містять заголовок, відповідне наповнення та кнопки для відкриття

вікна редагування. Також можна відмітити, що деякі поля не мають кнопки зміни даних чи у них відсутнє наповнення. Пусте поле вказує на те, що користувач не вказав інформацію, а відсутність можливості зміни дати народження та дати реєстрації у зв'язку з тим, що дана інформація є важливою для проведення аналізу статистики.

Окрім основного призначення дане вікно використовується для перегляду інформації про інших користувачів. У такому випаду кнопки зміни даних будуть відсутніми. Якщо ж дані будуть мати помітку приховані, або відсутні у базі даних текст поля буде встановлений як «невідомо».

Ще однією і останньою з важливих функцій у системі є блок статистики. Даний блок може бути використаний лише адміністрацією соціальної мережі для вивчення частоти використання функціональних можливостей, моніторингу кількості нових користувачів та інших важливих показників, на основі яких можна робити детальний аналіз та припущення щодо популярності соціальної мережі, її майбутніх перспектив, доцільності і майбутнього напряму її подальшого розвитку. Більш детально ознайомитись із зовнішній виглядом вікна панелі адміністратора соціальної мережі можна на рисунок 3.14.

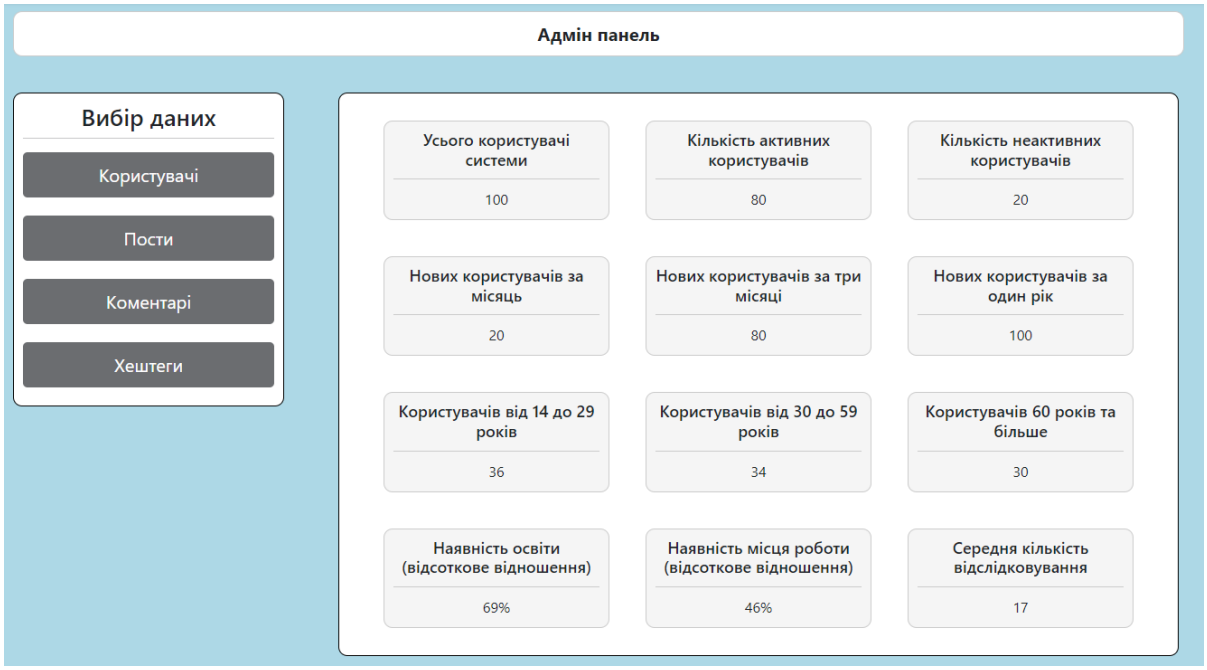


Рисунок 3.14 – Вікно статистики адміністратора

З метою поліпшення візуального подання інформації та спрощення зовнішнього вигляду інтерфейсу, з наведеного рисунку можна побачити, що перегляд інформації було організовано за принципом поділу на категорії. Вони дозволяють групувати схожу інформацію та забезпечити більш зручний спосіб її перегляду. Такий підхід дозволяє краще організувати роботу з інформацією.

При перегляді кожної категорії у центральній частині вікна відображається інформація, яка стосується обраної категорії. Це дозволяє адміністрації соціальної мережі швидко оперувати великим обсягом.

Оскільки окрема таблиця для збереження статистичної інформації в базі даних не була передбачена вимогами, для її отримання виконуються різноманітні запити. Наприклад, для отримання кількості на момент перевірки, виконується запит, що підраховує кількість записів у відповідній таблиці бази даних, де зберігається уся основна інформація про користувачів системи.

Деякі запити можуть бути складнішими, наприклад, отримання кількості нових користувачів за певний період часу. Для цього виконується запит, який шукає користувачів, дата реєстрації яких відповідає вказаному періоду, і після цього відбувається підрахунок кількості знайдених користувачів.

Ще одним складним запитом є отримання відсоткових відношень. При його виконанні для отримання відповідних даних спочатку здійснюється пошук загальної кількості, для якого здійснюється пошук відношення, наприклад загальна кількість користувачів. Після цього відбувається відбір необхідного показника, наприклад користувачі, які вказали власну освіту та здійснюється розрахунок відношення.

Використання такого підходу дозволяє динамічно отримувати актуальну статистику без необхідності зберігання окремої таблиці. Запити до бази даних дозволяють отримувати потрібну інформацію в реальному часі і виводити її відповідним чином, щоб адміністрація соціальної мережі могла проводити аналіз та моніторинг різних показників.

3.3 Тестування програмного продукту

Тестування програмного забезпечення є невід'ємною складовою розробки високоякісного програмного продукту. Воно відіграє ключову роль у забезпеченні якості виявленні та виправленні помилок, тощо

До основних переваг тестування можна віднести:

– виявлення помилок та слабких місць у програмному забезпеченні, що дозволяє внести зміни до його коду ще до випуску або оновлення;

– тестування гарантує, що програмне забезпечення відповідатиме вимогам та зможе функціонувати надійно та безперебійно у різних ситуаціях;

– коректне тестування програмного продукту зменшує кількість ризиків та їхній вплив у разі появи;

– виявлення та виправлення вимог на ранніх стадіях розробки допомагає знизити кінцеву вартість продукту, оскільки усунення проблем на пізніших етапах може вимагати додаткових матеріальних, часових та інших видів витрат;

– тестування допомагає перевірити коректність роботи функціоналу та відповідність до його очікуваного результату та вимогам, що допомагає забезпечити високу якість кінцевого продукту та задоволення користувачів;

Процес тестування програмного забезпечення є комплексним і розгортається на різних рівнях. Воно дозволяє перевірити правильність роботи цих компонентів та їх взаємодію з іншими частинами системи.

Для тестування даної роботи було проведено декілька видів тестувань, починаючи з функціонального тестування. Воно дозволяє перевірити програму на відповідність встановленим вимогам. Воно включає проведення окремих тестів для перевірки функцій програми, взаємодії між ними та виконання різних завдань. З метою забезпечення ефективного функціонального тестування було вирішено використати Unit-тести.

Unit тести – це методи тестування програмного забезпечення, при якому здійснюється перевірка окремих компонентів на коректність роботи. Дані

перевірки спрямовані на тестування найменших одиниць програмного коду для виявлення помилок та забезпечення коректності їх роботи у ізоляції.

Зазвичай даний вид тестування використовується для виявлення помилок на ранніх стадіях розробки програмного забезпечення з метою забезпечення стабільності та надійності. Також дані тести забезпечують документацію для компонентів програми, описуючи очікуваний результат їхньої роботи.

Для здійснення Unit-тестування вже існуючого проекту було додано спеціальний модуль. До нього було додано посиланням на основний проект, щоб забезпечити подальшу можливість взаємодії між ними та можливість використання вже наявних контролерів, моделей, тощо.

Далі, у щойно створеному проекті було створений новий клас для проведення тестування, а також було реалізовано низку тестових методів.

У методах тестування здійснювався перевірка правильності роботи окремих компонентів програми в ізоляції. З частиною коду даних методів можна детально ознайомитись на рисунку 3.15.

```
[TestClass]
  public class UnitTest1
  {
      [TestMethod]
      public void TestMethodPostStore()
      {
          PostsController obj = new PostsController(null);
          PostViewModel model = new PostViewModel
          {
              Text = "It is test text"
          };
          obj.Store(model);
      }

      [TestMethod]
      public void TestMethodPostDelete()
      {
          PostsController obj = new PostsController(null);
          int PostId = 0;
          obj.Delete(PostId);
      }

      [TestMethod]
      public void TestMethodAddLike()
      {
          LikesController obj = new LikesController(null);
          int PostId = 0;
          obj.AddLike(PostId);
      }
  }
```

Рисунок 3.15 – Код Unit тесту

Після створення відповідних тестів, було запущено процес їхнього виконання для перевірки коректності роботи відповідних методів контролерів. Це призвело до відкриття діалогового вікна, спеціально призначеного для проведення тестування. Зовнішній вигляд цього діалогового вікна можна побачити на рисунку 3.16.

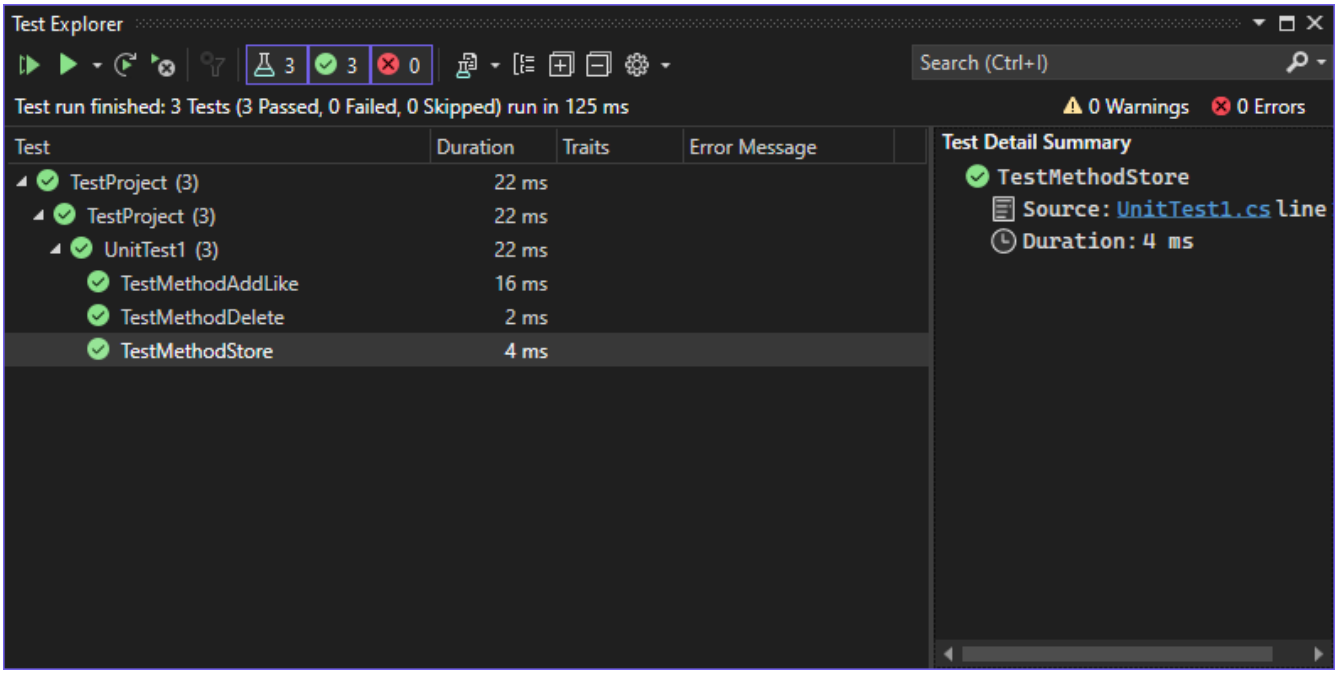


Рисунок 3.16 – Результати тестування

Діалогове вікно тестування надає можливість керувати тестами та спостерігати за їх виконанням. На ньому можна побачити перелік запланованих тестів, їхні стани, наприклад, «у черзі», «виконується», «пройшов успішно», а також додаткову інформацію про виконання окремо кожного тесту.

З рисунку видно, що біля назви кожного метода тестування розміщено помітку зеленого кольору. Вона вказує на те, що усі методи тестування завершились успішно.

Окрім тестування за допомогою Unit тестів було вирішено, що доцільним є проведення ряду різних ручних тестувань, що допоможуть краще перевірити готовий програмний продукт.

Ручним тестуванням називають процес перевірки та оцінки функціональності, якості та відповідності програмного забезпечення вимогам і очікуваним результатам. Дана форма тестування проводиться за рахунок використання людських ресурсів та їх прямій взаємодії з різними аспектами та функціональними можливостями програмного забезпечення.

Ручне тестування може включає в процес взаємодію з інтерфейсом користувача, введення різних типів даних, виконання різноманітних сценаріїв використання програмного забезпечення тощо. Цей процес дозволяє виявити можливі помилки, протиріччя або недоліки в роботі програмного продукту.

Ручне тестування є важливою складовою розробки програмного забезпечення, оскільки воно дозволяє виявити проблеми, які можуть бути недосяжними для автоматичних тестів. Взаємодія з реальними користувачами та їхніми реальними сценаріями використання допомагає виявити аспекти програми, які можуть бути непередбаченими або неочевидними.

Для даного програмного продукту було здійснено тестування за наступними видами ручного тестування:

– функціональне тестування;

– тестування інтерфейсу;

– тестування сумісності;

– тестування користувачем.

Основним завданням функціонального тестування є перевірка відповідності програмного продукту вимогам та реалізованому функціоналу.

Для проведення функціонального тестування було повторно переглянуто вимоги до програмного забезпечення. Це дозволило ознайомитись з очікуваними функціональними можливостями, які програмний продукт повинен виконувати. Після цього було проведено ручну перевірку програмного продукту з метою виявлення можливих помилок або відхилень від встановлених вимог.

При виявленні відхилень від очікуваних результатів були внесені зміни до коду. Це включає у себе виправлення помилок, оптимізацію функціоналу або

внесення інших необхідних змін. Після цього було проведене повторне тестування функціоналу.

Тестування інтерфейсу є важливим етапом в процесі розробки програмного забезпечення. Даний вид тестування передбачає перевірку інтерфейсу користувача на зручність та інтуїтивну зрозумілість при взаємодії користувачів з різним рівнем володіння техніки.

Для його проведення було залучено людей, які до цього не були знайомі з інтерфейсом та не використовували програмний продукт. Кожному з них було поставлено ряд завдань, які необхідно виконати та дати оцінку зручності інтерфейсу та власні рекомендації, якщо такі є, як його можна покращити.

Результати тестування інтерфейсу дозволяють отримати об'єктивну оцінку його якості та зрозуміти, як користувачі сприймають та взаємодіють з ним.

Для забезпечення широкої сумісності та коректної роботи програми на різних платформах і операційних системах було проведено тестування сумісності. Воно спрямоване на перевірку, чи працює застосунок на різних операційних системах, платформах та браузерах, без помилок.

Головна увага під час тестування сумісності стосувалась перевірці клієнтської частини програми, а саме коректності його роботи та відображення інтерфейсу на різних операційних системах та браузерах останніх версій.

Для цього були використані наступні операційні системах:

– Windows 7;
– Windows 10;
– Ubuntu Linux.

Тестування сумісності здійснювалось у наступних браузерах:

– Firefox;
– Google;
– Opera;
– Opera GX;
– Edge.

Цей процес тестування сумісності дозволив впевнитися, що програмний продукт працює на різних платформах та операційних системах без помилок. Таке тестування є важливим для забезпечення максимальної доступності та задоволення потреб різних користувачів.

Останнім видом ручного тестування, яке було проведене для даного програмного забезпечення є тестування користувачем. Даний вид тестування передбачає перевірку застосунку у реальних умовах користувачами для оцінки його придатності, ефективності роботи та зручності його використання.

Для перевірки основного функціоналу користувачам, які брали участь у даному тестуванні, було надано перелік основних завдань, які необхідно виконати. Основними його пунктами є:

– реєстрація у системі;

– вибір довільної публікації та детальне ознайомлення з його вмістом;

– відкриття посту у великому вигляді, створення коментаря до нього та його оцінка у вигляді вподобайки;

– почати відслідковувати декількох користувачів;

– відкриття власного профілю та створення нового поста;

– додання чи редагування додаткових даних профілю користувача;

– вихід із профілю користувача.

Після виконання завдань, кожному користувачеві ставили ряд запитань, пов'язаних з їхнім досвідом використання програмного продукту. Вони були спрямовувалися на збір відгуків щодо їхнього враження від використання програмного продукту. Користувачі могли оцінити його зручність, ефективність та інтуїтивність. Крім того, їм було надано можливість поділитися власними рекомендаціями та пропозиціями щодо, можливостей поліпшення програми.

Цей етап збору відгуків від користувачів був дуже цінним, оскільки він дозволив зібрати реальні враження та пропозиції від людей, які використовують програмний продукт у реальних умовах. Ця інформація може бути використана для подальшого вдосконалення програми та врахування потреб користувачів.

3.4 Висновки

Заключним етапом виконання даної кваліфікаційної роботи є програмна реалізації даної соціальної мережі, що у свою чергу складалось з декількох менших етапів.

Першим із них є розробка бази даних та вікон програми. Для цього було створено ряд моделей відповідно до раніше створених таблиць на ER-діаграмі. Далі за допомогою технології Entity Framework було створено контекст даних та здійснено процес міграції для автоматичного створення бази даних відповідно до наявних моделей.

Відповідно до раніше встановлених вимог до інтерфейсу користувача було розроблено основні стилі оформлення різних елементів HTML розмітки для спрощення подальшої розробки та здійснено опис основних особливостей відповідно до обраних технологій.

Здійснивши базове приготування було перейдено до розробки основних модулів програмного продукту. Кожен з них було детально описано відповідно до особливостей реалізації та наведено готовий результат його виконання у вигляді відповідної сторінки представлення.

Після завершення розробки програмного забезпечення було проведено його детальне тестування за допомогою Unit тестування окремих функціональних можливостей та здійснено ряд ручних тестувань для перевірки відповідно до вимог, коректності роботи та зручності роботи з інтерфейсом користувача. У випадку виявлення розбіжностей з вимогами чи некоректності роботи до відповідного коду було зроблено зміни та проведено повторне тестування відповідного функціоналу.

Усі дані етапи допомогли реалізувати програмний продукт, та здійснити його перевірку за багатьма факторами. У зв’язку з цим можна сказати, що програмний продукт реалізований у повному обсязі відповідно до усіх раніше поставлених вимог.

ВИСНОВКИ

На основі усіх попередніх етапів, які були виконані у для успішного виконання дані кваліфікаційній роботі, було проведено детальний аналіз підсумків їх виконання.

У першому розділі було проведено детальний аналіз відповідної предметної області після чого здійснено вивчення вже існуючих рішень на основі соціальних мереж Facebook та Twitter із встановленням їхніх особливостей, переваг та недоліків. На основі отриманої інформації було встановлено основні вимоги до розробки програмного забезпечення.

Наступним кроком, який було описано у другому розділі, було вивчено та детально проаналізовано ряд технологій та серед них вибрано ті, які доцільно використовувати у даному програмному забезпечені. Було вирішене використовувати фреймворк ASP.NET Core з архітектурними шаблонами MVC та клієнт-серверна архітектура. Окрім цього передбачене використання Entity Framework та AJAX. Далі було розроблено прототип інтерфейсу користувача для основних вікон програми та різноманітні діаграми, які відображають принципи роботи системи відповідно до встановлених вимог.

Після даних приготувань було виконано реалізацію програмного забезпечення відповідно до вимог та раніше вибраних технологій з проведенням детального аналізу окремих компонентів. По завершенню розробки додатково було проведено його тестування за допомогою Unit-тестів та декількох видів ручного тестування.

Розроблений програмний продукт у повній мірі відповідає поставленим функціональним та нефункціональним вимогам. За необхідності програмний продукт може бути швидко модифікований без необхідності внесення змін до вже наявного програмного коду.

ПЕРЕЛІК ДЖЕРЕЛ

1. Соціальна мережа. URL: https://igroup.com.ua/seo-articles/sotsialna-merezha/ (дата звернення: 10.02.2023).
2. Масленко, В. С. Класифікація користувачів соціальних мереж / В. С. Масленко // Актуальні задачі та досягнення у галузі кібербезпеки : матеріали Всеукр. наук.-практ. конф., м. Кропивницький, 23–25 листоп. 2016 р. – Кропивницький : КНТУ, 2016. – С. 196.
3. Літвак О. Соціальні мережі: поняття, історія виникнення. URL: https://zounb.zp.ua/resourse/zaporizkyy-kray/zaporizhzhya-bibliotechne/fahova-osvita/socialni-merezhi-piv (дата звернення: 12.02.2023).
4. Simon K. Digital 2022: Global overview report. URL: https://datareportal.com/reports/digital-2022-global-overview-report (дата звернення: 16.02.2023).
5. Top Websites Ranking – Most Visited Websites In The World. URL: https://igroup.com.ua/seo-articles/sotsialna-merezha/ (дата звернення: 21.02.2023).
6. The top 500 sites on the web. URL: https://web.archive.org/web/20180815002411/https:/www.alexa.com/topsites (дата звернення: 26.02.2023).
7. Facebook. URL: https://www.facebook.com/ (дата звернення: 26.02.2023).
8. Twitter. URL: https://twitter.com/ (дата звернення – 07.03.2023).
9. Hsian-Ching C. Hemalata L. Trends in Twitter Hashtag Applications: Design Features for Value-Added Dimensions to Future Library Catalogues. URL: https://muse.jhu.edu/article/485537 (дата звернення: 09.03.2023).
10. Trends in Twitter Hashtag Applications: Design Features for Value-Added Dimensions to Future Library Catalogues. URL: https://muse.jhu.edu/article/485537 (дата звернення: 12.03.2023).

11. What character can a hashtags include? URL: https://www.hashtags.org/featured/what-characters-can-a-hashtag-include/ (дата звернення: 12.03.2023).

12. Найважливіші архітектурні шаблони, які необхідно знати. URL: https://hi-news.pp.ua/kompyuteri/print:page,1,11675-arhtektura-programnogo-zabezpechennya-vidi-opis-rozrobka.html (дата звернення: 20.03.2023).

13. Архітектура програмного забезпечення: види, опис, розробка URL: https://devzone.org.ua/post/nayvazhlivishi-arkhitekturni-shabloni-yaki-neobkhidno-znati (дата звернення: 20.03.2023).

14. Клієнт–серверна архітектура. URL: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/ (дата звернення: 23.03.2023).

15. Клієнт-серверна архітектура та ролі сервера. URL: https://medium.com/@IvanZmerzlyi/клієнт-серверна-архітектура-та-ролі-серверів-9893d8048229 (дата звернення: 23.03.2023).

16. Розділяй та володарюй: що таке патерни MVC і MVP, та як їх використовувати. URL: https://highload.today/uk/blogs/shho-take-mvc-ta-mvp-patterni/ (дата звернення: 25.03.2023).

17. Програма Model-View-Controller MVC – що це. URL: https://hi-news.pp.ua/kompyuteri/14628-programa-model-view-controller-mvc-scho-ce-osoblivost-opis.html (дата звернення: 25.03.2023).

18. Roth D. Anderson R. Luttin S. Overview of ASP.NET Core. URL: https://learn.microsoft.com/en-gb/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0 (дата звернення: 27.03.2023).

19. Pehlivanov V. ASP.NET Core vs ASP.NET MVC: Which .NET Framework is better for You? URL: https://www.resolutesoftware.com/blog/asp-net-mvc-vs-asp-net-core/ (дата звернення: 27.03.2023).

20. Welcome to Flask – Flask Documentation (2.2.x). URL: https://flask.palletsprojects.com/en/2.2.x/ (дата звернення: 27.03.2023).

21. Introduction to Angular concepts. URL: https://angular.io/guide/architecture (дата звернення: 29.03.2023).

22. Харьковчук О. Як створити прототип сайту на папері, в програмі або онлайн. Для чого він вам потрібен? URL: https://web-promo.ua/ua/blog/kak-sozdat-prototip-sajta-na-bumage-v-programme-ili-onlajn-dlya-chego-vam-nuzhen/ (дата звернення: 02.04.2023).

23. Що таке UI і як інтерфейс користувача впливає на продажі інтернет магазину. URL: https://wezom.com.ua/ua/blog/chto-takoe-ui-i-kak-polzovatelskij-interfejs-vliyaet-na-prodazhi-internet-magazina (дата звернення: 04.04.2023).

24. Основні принципи створення інтерфейсу. URL: http://um.co.ua/8/8-10/8-109690.html (дата звернення: 05.04.2023).

25. Розробка інтерфейсів. Проектування графічного інтерфейсу користувача. URL: https://uaeu.top/digital-online/rozrobka-interfejsiv-proektuvannya-grafichnogo-interfejsu-koristuvacha.html (дата звернення: 08.04.2023).

26. Тестування UX-прототипів як потрібна ланка розробки продукту. Інтерактивні та статичні прототипи. URL: https://rustrackers.ru/uk/setting-up-software/testirovanie-ux-prototipov-kak-neobhodimoe-zveno-razrabotki/ (дата звернення: 13.04.2023).

27. Entity Framework, With .Net Core MVC, Code-First. URL: https://www.c-sharpcorner.com/article/entity-framework-with-net-core-mvc-4-code-first/ (дата звернення: 18.04.2023).

28. Entity Framework Core. URL: https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx (дата звернення: 18.04.2023).

29. Overview of ASP.NET Core MVC. URL: https://learn.microsoft.com/uk-ua/aspnet/core/mvc/overview?view=aspnetcore-7.0 (дата звернення: 17.04.2023).

30. ASP.NET Core Documentation. URL: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0 (дата звернення: 20.04.2023).

31. ASP.NET Core Tutorial. URL: https://www.tutorialspoint.com/asp.net_core/index.htm (дата звернення: 20.04.2023).

32. Introduction to ASP.NET Core MVC. URL: https://www.yogihosting.com/aspnet-core-introduction/ (дата звернення: 21.04.2023).

33. Murugan M. Custom User Management in ASP.NET Core MVC with Identity. URL: https://codewithmukesh.com/blog/user-management-in-aspnet-core-mvc/ (дата звернення: 21.04.2023).

34. Use AJAX to Deliver Dynamic Updates. URL: https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/nerddinner/use-ajax-to-deliver-dynamic-updates (дата звернення: 25.04.2023).

35. Ajax In .Net Core. URL: https://www.c-sharpcorner.com/article/ajax-in-net-core/ (дата звернення: 26.04.2023).

36. jQuery AJAX in ASP.NET Core MVC. URL: https://qawithexperts.com/article/asp-net/sending-data-to-controller-using-jquery-ajax-in-aspnet-core/373 (дата звернення: 02.05.2023).

37. How to post data to the controller using AJAX with validations in ASP.NET Core. URL: https://medium.com/c-sharp-progarmming/how-to-post-data-to-the-controller-using-ajax-with-validations-in-asp-net-core-a63af60867e9 (дата звернення: 03.05.2023).

38. ASP.NET Core MVC / Razor pages UI JavaScript AJAX API. URL: https://docs.abp.io/en/abp/latest/UI/AspNetCore/JavaScript-API/Ajax (дата звернення: 07.05.2023).

39. Огляд видів тестування. URL: https://training.qatestlab.com/blog/technical-articles/review-the-types-of-testing/ (дата звернення: 11.05.2023).

40. Леонов О. Тестування ПЗ (види тестування). URL: https://drukarnia.com.ua/articles/testuvannya-pz-vidi-testuvannya-JInS1#heading-3-4659 (дата звернення: 12.02.2023).

41. Ручне тестування – що це таке, типи, процеси, підходи, інструменти та інше. URL: https://www.zaptest.com/uk/ручне-тестування-що-це-таке-типи-проц (дата звернення: 15.02.2023).

42. Unit Testing in ASP.NET Core MVC. URL: https://code-maze.com/unit-testing-in-asp-net-core-mvc/ (дата звернення: 18.05.2023).

43. Unit testing in ASP.NET Core Web API. URL: https://medium.com/c-sharp-progarmming/unit-testing-in-asp-net-core-web-api-b2e6f7bdb860 (дата звернення: 19.05.2023).

44. Unit testing C# in .NET Core using dotnet test and xUnit. URL: https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-dotnet-test (дата звернення: 20.05.2023).

ДОДАТОК А

(обов’язковий)

ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ

A.1 Код контролера PostsController

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using MyMainDiplomProject.Data;
using MyMainDiplomProject.Models;
using MyMainDiplomProject.Models.ViewModel;
using MyMainDiplomProject.Models.ViewModels;

namespace MyMainDiplomProject.Controllers
{
    public class PostsController : Controller
    {
        private readonly MyMainDiplomProjectDbContext _context;

        public PostsController(MyMainDiplomProjectDbContext context)
        {
            _context = context;
        }

        public async Task<IActionResult> Index()
        {
            var topHashtags = _context.HashTags
                .GroupBy(ht => ht.Name)
                .OrderByDescending(g => g.Count())
                .Take(3)
                .Select(g => new { Name = g.Key, Count = g.Count() }).ToList();

            PostInfoViewModel model = new PostInfoViewModel
            {
                Posts = _context.Posts.Include(p => p.User).Include(i =>
i.PostHashTags).Include(i => i.Files).Include(i => i.Likes).Include(i =>
i.Comments).OrderByDescending(i => i.CreatedDateRime).ToList(),
                Users = _context.Users.ToList(),
                UserAdditionalInfos = _context.UserAdditionalInfo.ToList(),
```

```
            FollowLists = _context.FollowLists.ToList(),
            TopHasTag1 = topHashtags.Count >= 1 ? topHashtags[0].Name : "",
            TopHasTag2 = topHashtags.Count >= 2 ? topHashtags[1].Name : "",
            TopHasTag3 = topHashtags.Count >= 3 ? topHashtags[2].Name : "",
            TopHasTag4 = topHashtags.Count >= 4 ? topHashtags[3].Name : "",
            TopHasTag5 = topHashtags.Count >= 5 ? topHashtags[4].Name : ""
        };

        return RedirectToAction("Index1", model);
    }

    public async Task<IActionResult> Index1()
    {
        var topHashtags = _context.HashTags
            .GroupBy(ht => ht.Name)
            .OrderByDescending(g => g.Count())
            .Take(3)
            .Select(g => new { Name = g.Key, Count = g.Count() }).ToList();

        PostInfoViewModel model = new PostInfoViewModel
        {
            Posts = _context.Posts.Include(p => p.User).Include(i =>
i.PostHashTags).Include(i => i.Files).Include(i => i.Likes).Include(i =>
i.Comments).OrderByDescending(i => i.CreatedDateRime).ToList(),
            Users = _context.Users.ToList(),
            UserAdditionalInfos = _context.UserAdditionalInfo.ToList(),
            FollowLists = _context.FollowLists.ToList(),
            TopHasTag1 = topHashtags.Count >= 1 ? topHashtags[0].Name : "",
            TopHasTag2 = topHashtags.Count >= 2 ? topHashtags[1].Name : "",
            TopHasTag3 = topHashtags.Count >= 3 ? topHashtags[2].Name : "",
            TopHasTag4 = topHashtags.Count >= 4 ? topHashtags[3].Name : "",
            TopHasTag5 = topHashtags.Count >= 5 ? topHashtags[4].Name : ""
        };
        return View(model);
    }

    [Authorize]
    public IActionResult Create()
    {
        return View();
```

```
        }

        [Authorize]
        [HttpPost]
        public async Task<IActionResult> Store(PostViewModel model)
        {
            if (ModelState.IsValid)
            {
                var post = new Post
                {
                    UserId = Convert.ToString(_context.Users.Where(i => i.Email ==
User.Identity.Name).FirstOrDefault().Id),
                    Text = model.Text,
                    CreatedDateRime = DateTime.Now,
                    Files = new List<Files>(),
                    PostHashTags = new List<HashTags>()
                };

                if (model.HashTags != null)
                {
                    foreach (var tag in model.HashTags)
                    {
                       if(_context.HashTags.Where(i => i.Name == tag).Count() > 0)
                       {
                            post.PostHashTags.Add(_context.HashTags.Where(i => i.Name ==
tag).FirstOrDefault());
                       }
                       else
                       {
                            post.PostHashTags.Add(new HashTags() { Name = tag });
                       }
                    }
                }

                if (model.Files != null)
                {
                    foreach (var file in model.Files)
                    {
                       if(_context.Files.Where(i => i.Name == file).Count() > 0)
                       {
```

```
                        post.Files.Add(_context.Files.Where(i => i.Name == file).FirstOrDefault());
                    }
                    else
                    {
                        post.Files.Add(new Files() { Name = file });
                    }
                }
            };
            post.Files.Add(file);
            _context.Posts.Add(post);
            await _context.SaveChangesAsync();

            return RedirectToAction("Index", "Home");
        }
        return View("Create", model);
    }

    public async Task<IActionResult> UserProfile(string Id)
    {
        var myMainDiplomProjectDbContext = _context.Posts
            .Include(i => i.User)
            .Include(i => i.Files)
            .Include(i => i.PostHashTags)
            .Include(i => i.Files)
            .Include(i => i.Comments)
            .Include(i => i.Likes)
            .Where(i => i.UserId == Id)
            .FirstOrDefault();

        UserProfileViewModel model = new UserProfileViewModel
        {
            User = _context.Users.Where(i => i.Id == _context.Users.Where(i => i.Id == Id).FirstOrDefault().Id).FirstOrDefault(),
            Posts = _context.Posts.Include(i => i.Files).Include(i => i.Likes).Include(i => i.PostHashTags).Include(i => i.Comments).Where(i => i.UserId == _context.Users.Where(i => i.Id == Id).FirstOrDefault().Id).ToList(),
            UserAdditionalInfo = _context.UserAdditionalInfo.Where(i => i.UserId == _context.Users.Where(i => i.Id == Id).FirstOrDefault().Id).FirstOrDefault(),
            FollowLists = _context.FollowLists.Where(i => i.UserId == Id).ToList()
```

```
            };
            return View("UserProfile", model);
        }

        [Authorize]
        public async Task<IActionResult> MyUserProfile()
        {
            string Id = _context.Users.Where(i => i.Email == User.Identity.Name).FirstOrDefault().Id;

            UserProfileViewModel model = new UserProfileViewModel
            {
                User = _context.Users.Where(i => i.Id == Id).FirstOrDefault(),
                Posts = _context.Posts.Include(i => i.Files).Include(i => i.Likes).Include(i => i.PostHashTags).Include(i => i.Comments).Where(i => i.UserId == Id).ToList(),
                UserAdditionalInfo = _context.UserAdditionalInfo.Where(i => i.UserId == Id).FirstOrDefault(),
                FollowLists = _context.FollowLists.Where(i => i.UserId == Id).ToList()
            };
            return View("UserProfile", model);
        }

        public ActionResult BigPost(int Id)
        {
            var post = _context.Posts
                .Include(i => i.User)
                .Include(i => i.Files)
                .Include(i => i.PostHashTags)
                .Include(i => i.Files)
                .Include(i => i.Comments)
                .Include(i => i.Likes)
                .Where(i => i.Id == Id)
                .FirstOrDefault();

            PostInfoViewModel model = new PostInfoViewModel
            {
                Post = post,
                UserAdditionalInfo = _context.UserAdditionalInfo.Where(i => i.UserId == post.UserId).FirstOrDefault()
```

```
    };
    return PartialView(model);
}

public ActionResult SearchByHashTag(string HasTag)
{
    List<Post> filteredPosts = _context.Posts
        .Include(p => p.User)
        .Include(i => i.PostHashTags)
        .Include(i => i.Files)
        .Include(i => i.Likes)
        .Include(i => i.Comments)
        .Where(p => p.PostHashTags.Any(i => i.Name == HasTag))
        .OrderByDescending(i => i.CreatedDateRime)
        .ToList();

    PostInfoViewModel model = new PostInfoViewModel
    {
        Posts = filteredPosts,
        Users = _context.Users.ToList(),
        UserAdditionalInfos = _context.UserAdditionalInfo.ToList(),
        FollowLists = _context.FollowLists.ToList(),
    };
    return PartialView(model);
}

public ActionResult Search(string name, string hashtag, string userType)
{
    List<Post> filteredPosts = new List<Post>();
    List<FollowList> follow = _context.FollowLists.ToList();
    string userId;

    filteredPosts = _context.Posts
            .Include(p => p.User)
            .Include(i => i.PostHashTags)
            .Include(i => i.Files)
            .Include(i => i.Likes)
            .Include(i => i.Comments)
            .OrderByDescending(i => i.CreatedDateRime)
            .ToList();
```

```
            if (name != null)
            {
                filteredPosts = filteredPosts.Where(i => i.User.userNikName.Contains(name)).ToList();
            }

            if (hashtag != null)
            {
                filteredPosts = filteredPosts.Where(i => i.PostHashTags.Any(i => i.Name == hashtag)).ToList();
            }

            if (userType == "friend")
            {
                userId = Convert.ToString(_context.Users.Where(i => i.Email == User.Identity.Name).FirstOrDefault().Id);
                follow = _context.FollowLists.Where(i => i.UserId == userId).ToList();
                filteredPosts = filteredPosts.Join(
                    follow,
                    post => post.UserId,
                    f => f.FolloverUserId,
                    (post, f) => post
                ).ToList();
            }

            else if (userType == "all")
            {
                userId = Convert.ToString(_context.Users.Where(i => i.Email == User.Identity.Name).FirstOrDefault().Id);
                follow = _context.FollowLists.Where(i => i.UserId != userId).ToList();
                filteredPosts = filteredPosts.Join(
                    follow,
                    post => post.UserId,
                    f => f.FolloverUserId,
                    (post, f) => post
                ).ToList();
            }

            PostInfoViewModel model = new PostInfoViewModel
```

```
            {
                Posts = filteredPosts,
                Users = _context.Users.ToList(),
                UserAdditionalInfos = _context.UserAdditionalInfo.ToList(),
                FollowLists = follow,
            };
            return PartialView(model);
        }

    public async Task<IActionResult> DeletePost(int id)
    {
        var post = await _context.Posts.FindAsync(id);
        MyMainDiplomProjectUser user = _context.Users.Where(i => i.Email == User.Identity.Name).FirstOrDefault();

        if (post.UserId == user.Id)
        {
            if (post == null)
            {
                return NotFound();
            }

            _context.Posts.Remove(post);
            await _context.SaveChangesAsync();
        }

        return RedirectToAction("Index", "Home");
    }

    [HttpGet]
    public async Task<IActionResult> EditPost(int id)
    {
        var post = await _context.Posts.Include(p => p.PostHashTags).Include(i => i.Files).FirstOrDefaultAsync(p =>
p.Id == id);

        if (post == null)
        {
            return NotFound();
        }

        PostInfoViewModel model = new PostInfoViewModel
```

```
        {
            Post = post,
            PostHashTags = post.PostHashTags.Select(t => t.Name).ToList(),
        };


        return View(model);
    }

    [HttpPost]
    public async Task<IActionResult> EditPost(PostInfoViewModel model)
    {
        if (ModelState.IsValid)
        {
            var post = await _context.Posts.Include(p => p.PostHashTags).FirstOrDefaultAsync(p => p.Id ==
model.Post.Id);

            if (post == null)
            {
                return NotFound();
            }

            post.Text = model.Post.Text;

            if (model.HashTags != null)
            {
                post.PostHashTags.Clear();

                foreach (var tag in model.HashTags)
                {
                    if (_context.HashTags.Any(i => i.Name == tag))
                    {
                        var existingTag = _context.HashTags.FirstOrDefault(i => i.Name == tag);
                        post.PostHashTags.Add(existingTag);
                    }
                    else
                    {
                        var newTag = new HashTags { Name = tag };
                        post.PostHashTags.Add(newTag);
                    }
```

```
            }
        }

        _context.Posts.Update(post);
        await _context.SaveChangesAsync();

        return RedirectToAction("Index", "Home");
      }

      return View("Edit", model);
    }

  }
}
```

А.2 – Код контролера UserAdditionalInfoesController

```
using System.Data;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using MyMainDiplomProject.Data;
using MyMainDiplomProject.Models;
using MyMainDiplomProject.Models.ViewModel;
using MyMainDiplomProject.Models.ViewModels;

namespace MyMainDiplomProject.Controllers
{
    public class UserAdditionalInfoesController : Controller
    {
        private readonly MyMainDiplomProjectDbContext _context;

        public UserAdditionalInfoesController(MyMainDiplomProjectDbContext context)
        {
            _context = context;
        }

        public async Task<IActionResult> Index(string UserId)
```

```
        {
            if (UserId == "MyProfile")
            {
                UserId = _context.Users.Where(i => i.Email ==
User.Identity.Name).FirstOrDefault().Id;
            }

            if (_context.UserAdditionalInfo.Where(i => i.UserId ==
UserId).IsNullOrEmpty())
            {
                UserAdditionalInfo additionalInfo = new UserAdditionalInfo
                {
                    UserId = UserId,
                    ShowDescription = false,
                    ShowEducation = false,
                    ShowWorkPlase = false,
                    ShowUserInterests = false,
                    UserAvatar =
"https://upload.wikimedia.org/wikipedia/commons/thumb/5/59/User-avatar.svg/2048px-User-
avatar.svg.png"
                };
                _context.Add(additionalInfo);
                _context.SaveChanges();
            }
            var myMainDiplomProjectDbContext = _context.UserAdditionalInfo.Include(i =>
i.User).Where(i => i.UserId == UserId).FirstOrDefault();

            return View(myMainDiplomProjectDbContext);
        }

        public async Task<IActionResult> UserList()
        {
            foreach (var item in _context.Users)
            {
                if (_context.UserAdditionalInfo.Where(i => i.UserId ==
item.Id).IsNullOrEmpty())
                {
                    UserAdditionalInfo additionalInfo = new UserAdditionalInfo
                    {
                        UserId = item.Id,
```

```
                        ShowDescription = false,
                        ShowEducation = false,
                        ShowWorkPlase = false,
                        ShowUserInterests = false,
                        UserAvatar =
"https://upload.wikimedia.org/wikipedia/commons/thumb/5/59/User-avatar.svg/2048px-User-
avatar.svg.png"
                    };
                    _context.Add(additionalInfo);
                    _context.SaveChanges();
                }
            }
            var myMainDiplomProjectDbContext = _context.UserAdditionalInfo.Include(i =>
i.User) ;
            return View(await myMainDiplomProjectDbContext.ToListAsync());
        }

        public async Task<IActionResult> UserProfile(string Id)
        {
            var myMainDiplomProjectDbContext = _context.UserAdditionalInfo.Include(i =>
i.User).Where(i => i.UserId == Id).FirstOrDefault();

            return View(myMainDiplomProjectDbContext);
        }

        public async Task<IActionResult> MyUserProfile()
        {
            var myMainDiplomProjectDbContext = _context.UserAdditionalInfo.Include(i =>
i.User).Where(i => i.UserId == _context.Users.Where(i => i.Email ==
User.Identity.Name).FirstOrDefault().Id).FirstOrDefault();
            return View("UserProfile", myMainDiplomProjectDbContext);
        }

        public async Task<IActionResult> AdminStats()
        {
            return View();
        }

        public async Task<IActionResult> AdminStatsUser()
        {
```

```
            DateTime ThirtyDaysAgo = DateTime.Now.AddDays(-30);
            DateTime NinetyDaysAgo = DateTime.Now.AddDays(-90);
            DateTime YearAgo = DateTime.Now.AddDays(-365);
            DateTime FourteenYearsAgo = DateTime.Now.AddYears(-14);
            DateTime TwentyNineYearsAgo = DateTime.Now.AddYears(-29);
            DateTime SixtyYearsAgo = DateTime.Now.AddYears(-59);

            int TotalUsers = _context.Users.Count();
            int ActivUser = _context.Posts.Where(i => i.CreatedDateRime >=
ThirtyDaysAgo).Select(i => i.UserId).Distinct().Count();
            int UsersHaveWork = _context.UserAdditionalInfo.Where(i => i.WorkPlase !=
null).Count();
            int UsersHaveEducation = _context.UserAdditionalInfo.Where(i => i.Education
!= null).Count();

            StatsViewModel model = new StatsViewModel
            {
                TotalUsers = TotalUsers,
                ActivUser = ActivUser,
                PassiveUsers = TotalUsers - ActivUser,
                UsersForMonth = _context.Users.Where(i => i.registerDate >=
ThirtyDaysAgo).Count(),
                UsersForThreeMonth = _context.Users.Where(i => i.registerDate >=
NinetyDaysAgo).Count(),
                UsersForYear = _context.Users.Where(i => i.registerDate >=
YearAgo).Count(),
                UsersHaveWork = ((UsersHaveWork / TotalUsers) * 100),
                UsersHaveEducation = ((UsersHaveEducation / TotalUsers) * 100),
                AllFollowCount = _context.FollowLists.Count(),
                YoungUsers = _context.Users.Where(i => i.dateOfBirthday >=
TwentyNineYearsAgo && i.dateOfBirthday <= FourteenYearsAgo).Count(),
                MiddleAgedUsers = _context.Users.Where(i => i.dateOfBirthday >
TwentyNineYearsAgo && i.dateOfBirthday <= SixtyYearsAgo).Count(),
                OldAgedUsers = _context.Users.Where(i => i.dateOfBirthday >
SixtyYearsAgo).Count()
            };
            return View(model);
        }
```

```
        public async Task<IActionResult> AdminStatsPost()
        {
            DateTime ThirtyDaysAgo = DateTime.Now.AddDays(-30);
            DateTime NinetyDaysAgo = DateTime.Now.AddDays(-90);
            DateTime YearAgo = DateTime.Now.AddDays(-365);

            StatsViewModel model = new StatsViewModel
            {
                TotalPosts = _context.Posts.Count(),
                PostsForMonth = _context.Posts.Where(i => i.CreatedDateRime >=
ThirtyDaysAgo).Count(),
                PostsForThreeMonth = _context.Posts.Where(i => i.CreatedDateRime >=
NinetyDaysAgo).Count(),
                PostsForYear = _context.Posts.Where(i => i.CreatedDateRime >=
YearAgo).Count(),
                PostsWithPictures = _context.Posts.Count(i => i.Files != null),
                PostsWithHasTags = _context.Posts.Count(i => i.PostHashTags != null),
            };
            return View(model);
        }

        public async Task<IActionResult> AdminStatsComment()
        {
            DateTime ThirtyDaysAgo = DateTime.Now.AddDays(-30);
            DateTime NinetyDaysAgo = DateTime.Now.AddDays(-90);
            DateTime YearAgo = DateTime.Now.AddDays(-365);

            StatsViewModel model = new StatsViewModel
            {
                TotalComments = _context.Comments.Count(),
                CommentsFowMounth = _context.Comments.Where(i => i.CreateDate >=
ThirtyDaysAgo).Count(),
                CommentsForThreeMounth = _context.Comments.Where(i => i.CreateDate >=
NinetyDaysAgo).Count(),
                CommentsForYear = _context.Comments.Where(i => i.CreateDate >=
YearAgo).Count(),
            };
            return View(model);
        }
```

```
        public async Task<IActionResult> AdminStatsHashTags()
        {
            var topHashtags = _context.HashTags.GroupBy(ht =>
ht.Name).OrderByDescending(g => g.Count()).Take(3).Select(g => new { Name = g.Key, Count
= g.Count() }).ToList();

            StatsViewModel model = new StatsViewModel
            {
                TotalHasTags = _context.HashTags.Count(),
                TopHasTag1 = topHashtags.Count >= 1 ? topHashtags[0].Name : "",
                TopHasTag2 = topHashtags.Count >= 2 ? topHashtags[1].Name : "",
                TopHasTag3 = topHashtags.Count >= 3 ? topHashtags[2].Name : "",
            };
            return View(model);
        }

        public async Task<IActionResult> Search(string name, int? minFollowers, int?
maxFollowers, string numUsers, string userType)
        {
            string AutorizeUserId = _context.Users.Where(i => i.Email ==
User.Identity.Name).FirstOrDefault().Id;
            var query = _context.Users.AsQueryable();

            if (!string.IsNullOrEmpty(name))
            {
                query = query.Where(i => i.userNikName.Contains(name));
            }

            if (minFollowers.HasValue || maxFollowers.HasValue)
            {
                var followersQuery = _context.FollowLists.GroupBy(i =>
i.UserId).Select(i => new { UserId = i.Key, FollowersCount = i.Count() });

                if (minFollowers.HasValue)
                {
                    followersQuery = followersQuery.Where(i => i.FollowersCount >=
minFollowers);
                }

                if (maxFollowers.HasValue)
```

```
                {
                    followersQuery = followersQuery.Where(i => i.FollowersCount <=
maxFollowers);
                }

                var userIdsWithFollowers = await followersQuery.Select(i =>
i.UserId).ToListAsync();
                query = query.Where(i => userIdsWithFollowers.Contains(i.Id));
            }

            if (numUsers == "verified")
            {
                var verifiedUserIds = _context.UserRoles.Where(i => i.RoleId == "2");
                //query = query.Where(i => verifiedUserIds.Contains(i.Id));
            }

            if (userType == "friend")
            {
                var followerUserIds = await _context.FollowLists.Where(f =>
f.FolloverUserId == AutorizeUserId).Select(f => f.UserId).ToListAsync();
                query = query.Where(u => followerUserIds.Contains(u.Id));
            }
            var searchResults = await query.ToListAsync();
            return View(searchResults);
        }

        public ActionResult InputOrEdit(int ChangeId)
        {
            string UserId = _context.Users.Where(i => i.Email ==
User.Identity.Name).FirstOrDefault().Id;

                if (_context.UserAdditionalInfo.Where(i => i.UserId ==
UserId).IsNullOrEmpty())
                {
                    UserAdditionalInfo additionalInfo = new UserAdditionalInfo
                    {
                        UserId = UserId,
                        ShowDescription = false,
                        ShowEducation = false,
                        ShowWorkPlase = false,
```

```
                        ShowUserInterests = false,
                        UserAvatar =
"https://upload.wikimedia.org/wikipedia/commons/thumb/5/59/User-avatar.svg/2048px-User-
avatar.svg.png"
                    };
                    _context.Add(additionalInfo);
                    _context.SaveChanges();
                }
            ChangeSeatingsViewModel model = new ChangeSeatingsViewModel
            {
                Id = ChangeId,
                UserAdditional = _context.UserAdditionalInfo.Where(i => i.UserId ==
UserId).FirstOrDefault(),
            };
            return PartialView(model);
        }

        [HttpPost]
        public ActionResult InputOrEdit(ChangeSeatingsViewModel model)
        {
            if(model.Id != null)
            {
                UserAdditionalInfo NewInfo = _context.UserAdditionalInfo.Where(i =>
i.UserId == model.UserAdditional.UserId).FirstOrDefault();
                NewInfo.User = _context.Users.Where(i => i.Id ==
model.UserAdditional.UserId).FirstOrDefault();
                if(model.Id == 1)
                {
                    if (model.UserAdditional.Description != null)
                    {
                        NewInfo.Description = model.UserAdditional.Description;
                        NewInfo.ShowDescription = model.UserAdditional.ShowDescription;
                    }
                    else
                    {
                        NewInfo.Description = "";
                        NewInfo.ShowDescription = false;
                    }
                }
```

```
                else if (model.Id == 2)
                {
                    if(model.UserAdditional.WorkPlase != null)
                    {
                        NewInfo.WorkPlase = model.UserAdditional.WorkPlase;
                        NewInfo.ShowWorkPlase = model.UserAdditional.ShowWorkPlase;
                    }
                    else
                    {
                        NewInfo.WorkPlase = "";
                        NewInfo.ShowWorkPlase = false;
                    }
                }

                else if (model.Id == 3)
                {
                    if(model.UserAdditional.UserInterests != null)
                    {
                        NewInfo.UserInterests = model.UserAdditional.UserInterests;
                        NewInfo.ShowUserInterests =
model.UserAdditional.ShowUserInterests;
                    }
                    else
                    {
                        NewInfo.UserInterests = "";
                        NewInfo.ShowUserInterests = false;
                    }
                }

                else if (model.Id == 4)
                {
                    if(model.UserAdditional.Education != null)
                    {
                        NewInfo.Education = model.UserAdditional.Education;
                        NewInfo.ShowEducation = model.UserAdditional.ShowEducation;
                    }
                    else
                    {
                        NewInfo.Education = "";
                        NewInfo.ShowEducation = false;
```

```
                }
            }

            else if (model.Id == 5)
            {
                if (model.UserAdditional.UserAvatar != null)
                {
                    NewInfo.UserAvatar = model.UserAdditional.UserAvatar;
                }
                else
                {
                    NewInfo.UserAvatar = "";
                }
            }
            else if (model.Id == 6)
            {
                if (model.Name != null)
                {
                    NewInfo.User.userNikName = model.Name;
                }
                else
                {
                    NewInfo.User.userNikName = "UserName";
                }
            }
            else if (model.Id == 7)
            {
                if (model.Email != null)
                {
                    NewInfo.User.Email = model.Email;
                }
            }
            _context.Update(NewInfo);
            _context.SaveChanges();
        }
        return RedirectToAction("Index","UserAdditionalInfoes", new { UserId =
model.UserAdditional.UserId });
    }
}
}
```

ДОДАТОК Б

(обов’язковий)

ПРЕЗЕНТАЦІЙНИЙ МАТЕРІАЛ

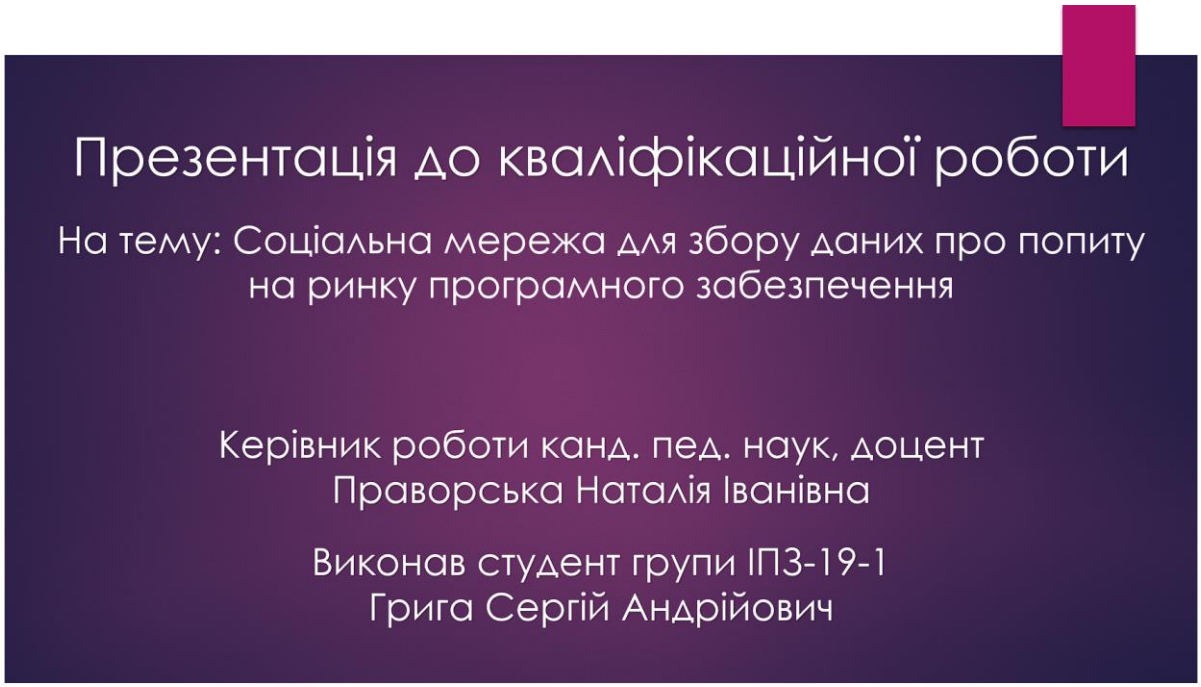


Рисунок Б.1 – Слайд №1

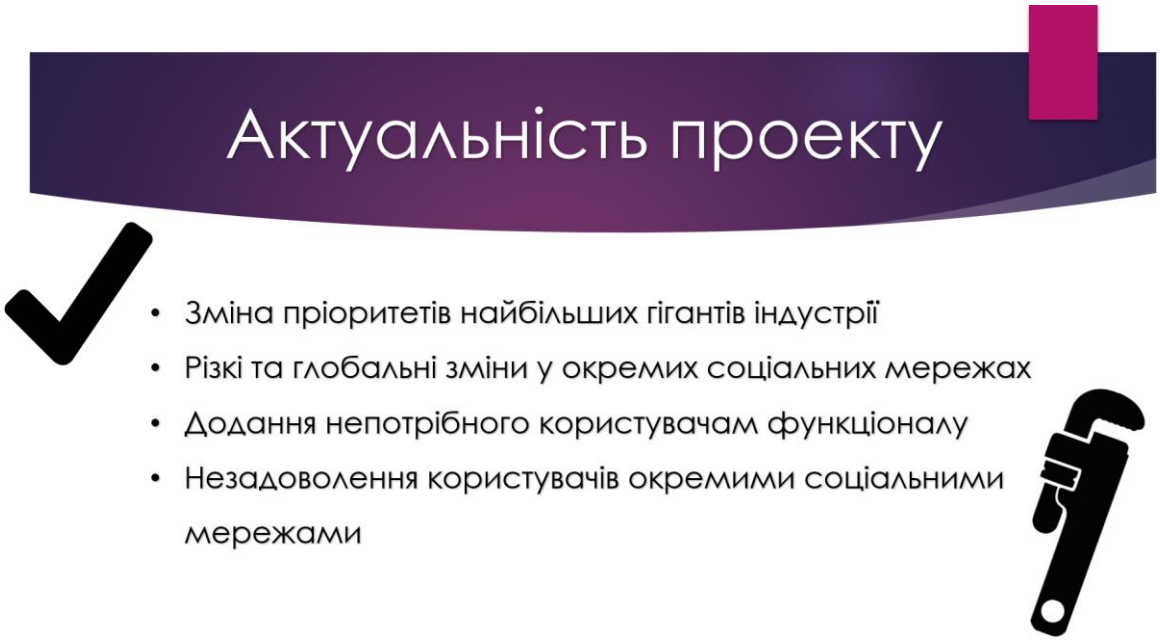


Рисунок Б.2 – Слайд №2

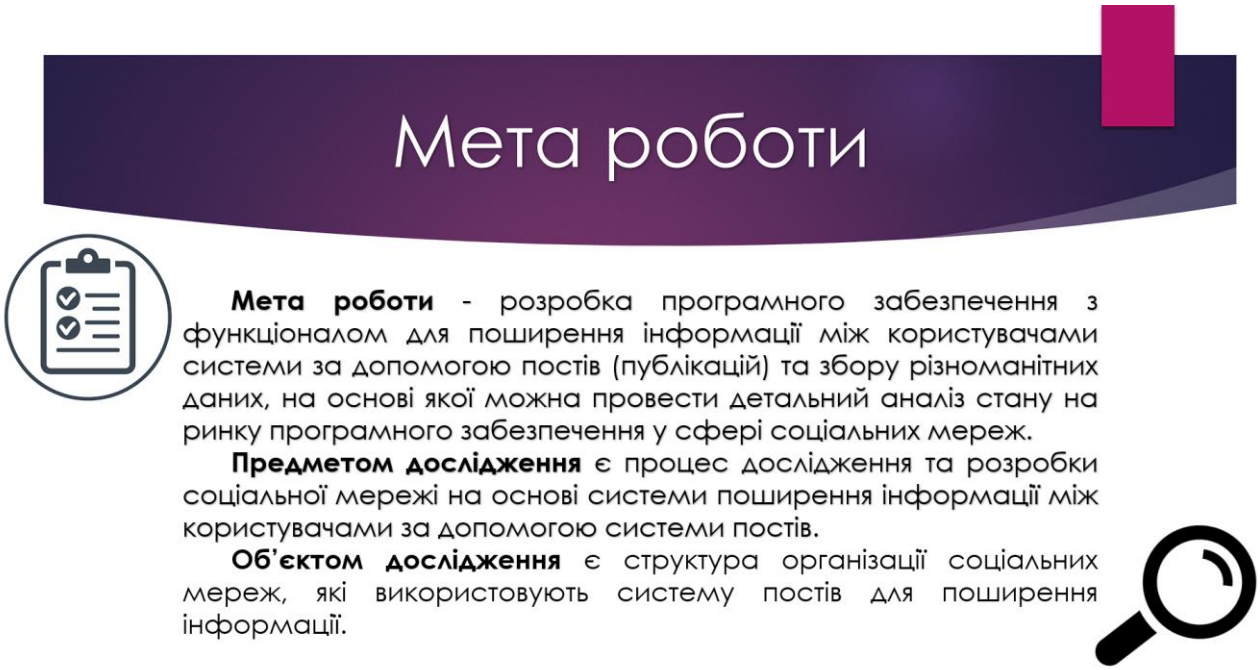


Рисунок Б.3 – Слайд №3

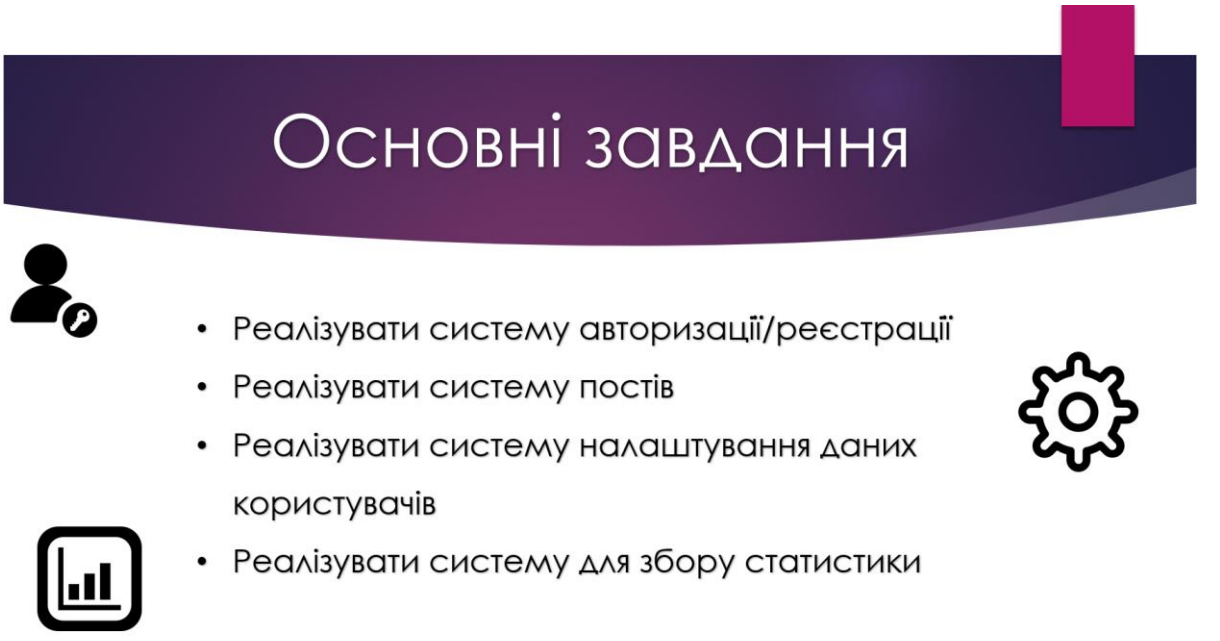


Рисунок Б.4 – Слайд №4

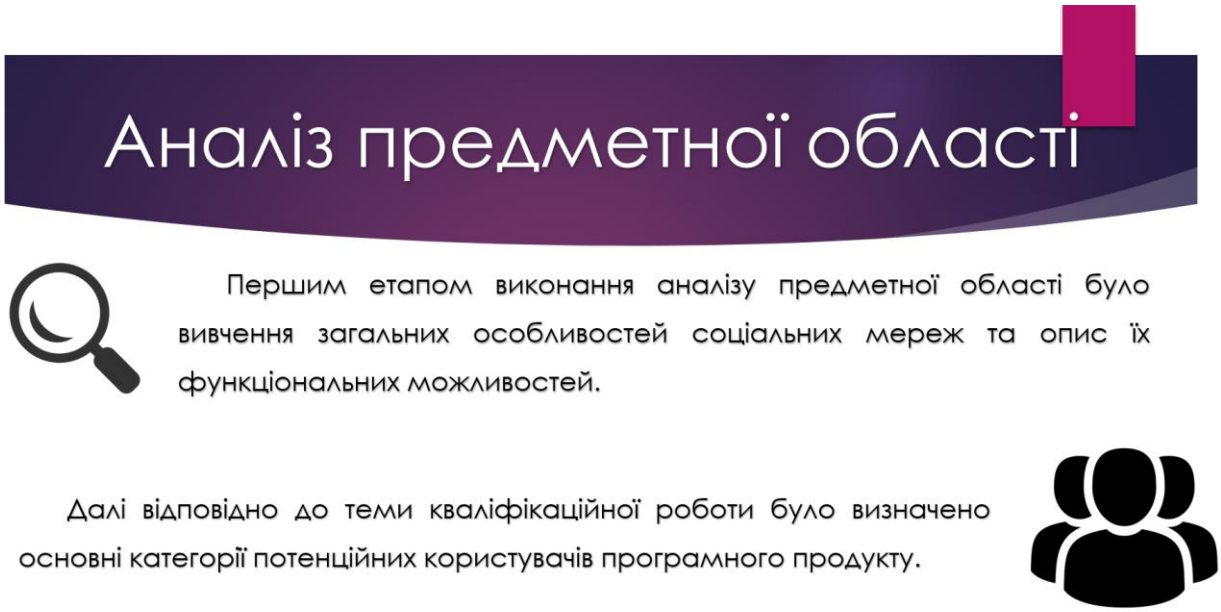


Рисунок Б.5 – Слайд №5

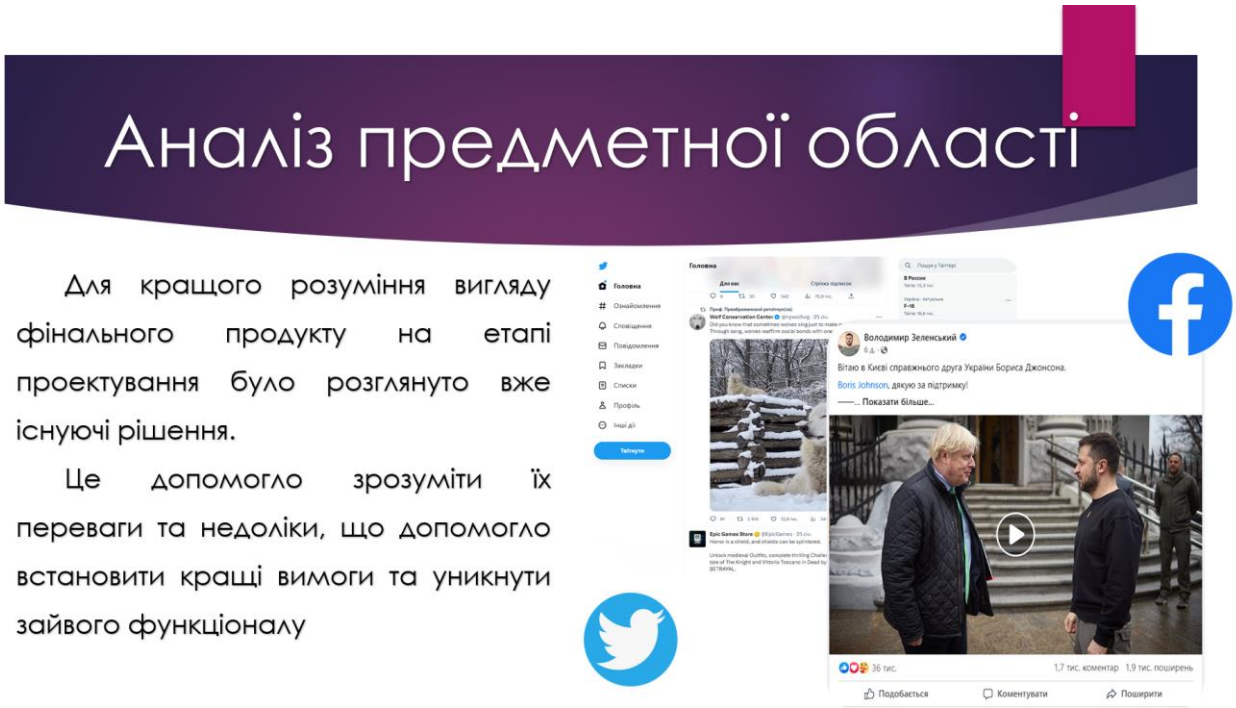


Рисунок Б.6 – Слайд №6

Рисунок Б.7 – Слайд №7

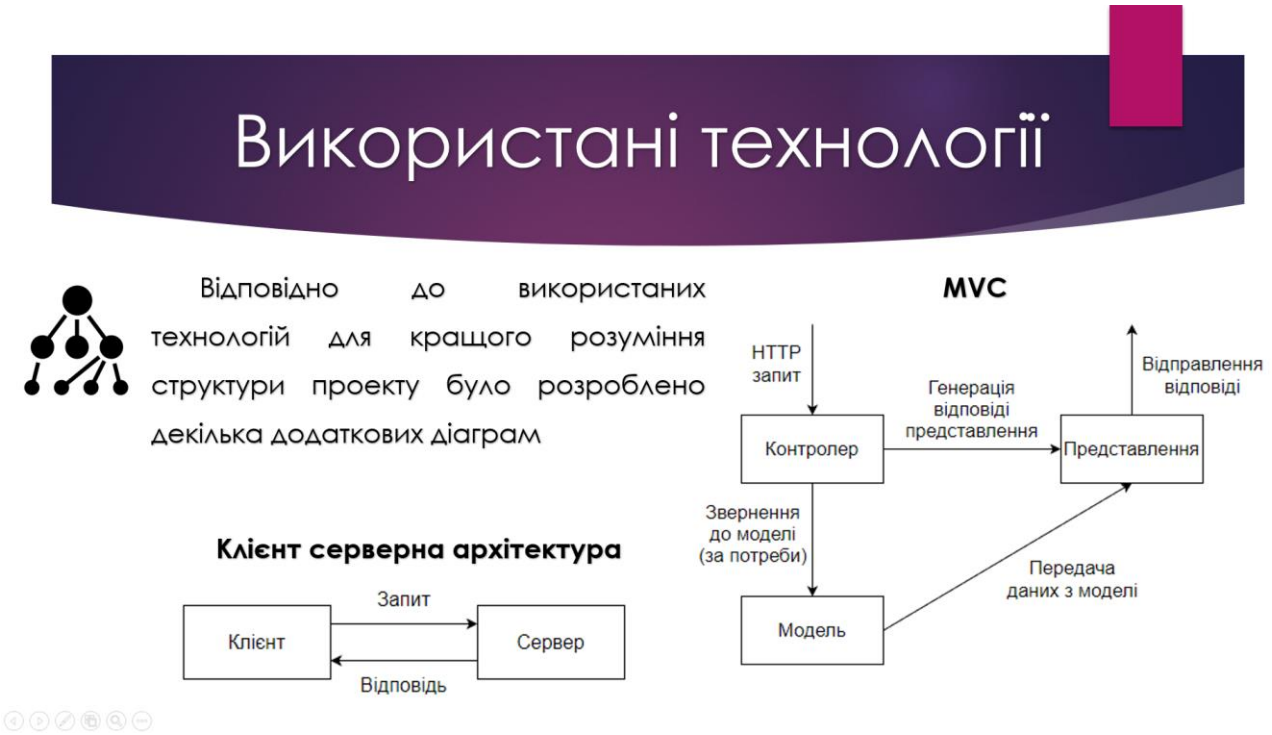


Рисунок Б.8 – Слайд №8

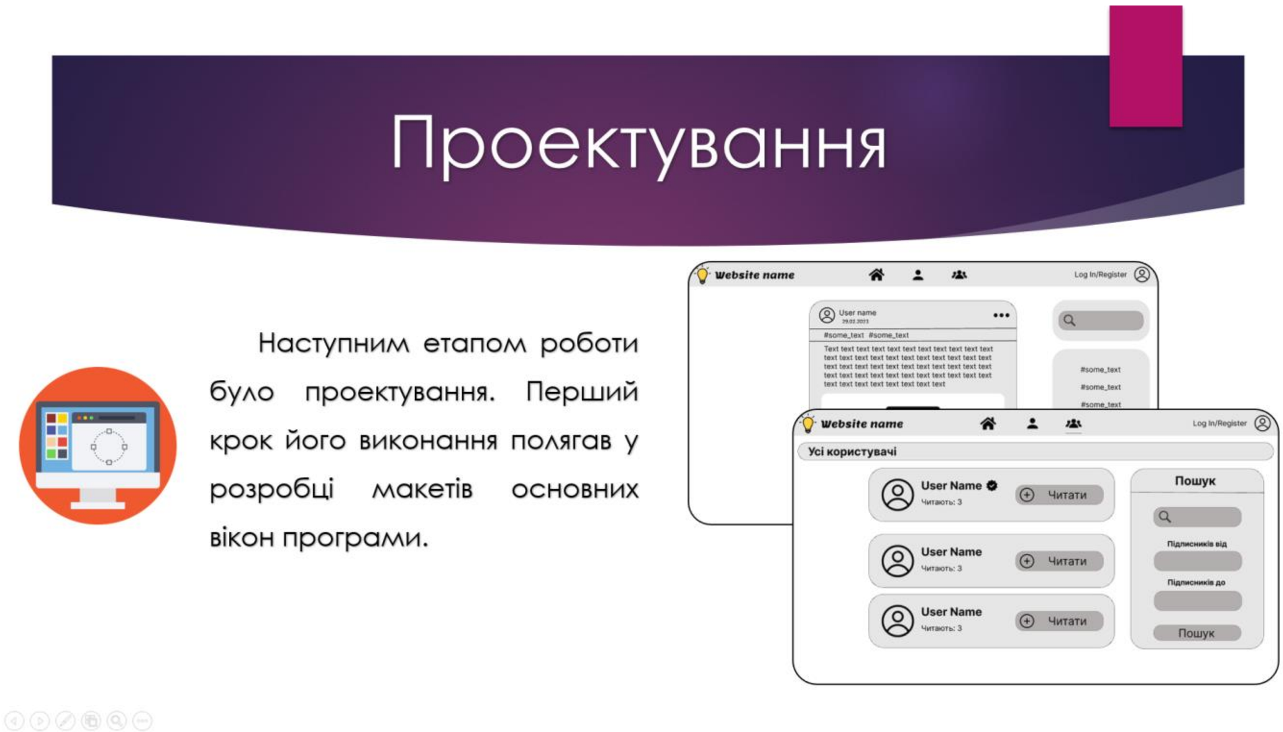


Рисунок Б.9 – Слайд №9

Рисунок Б.10 – Слайд №10

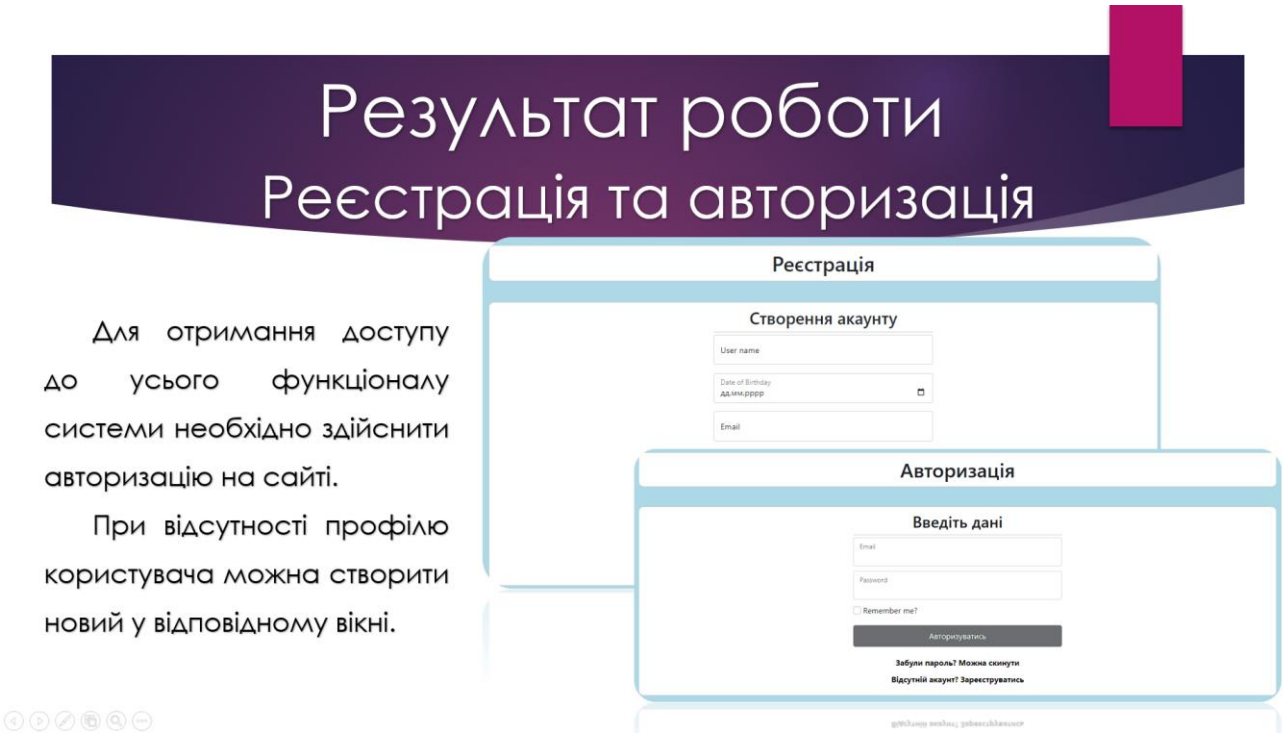


Рисунок Б.11 – Слайд №11

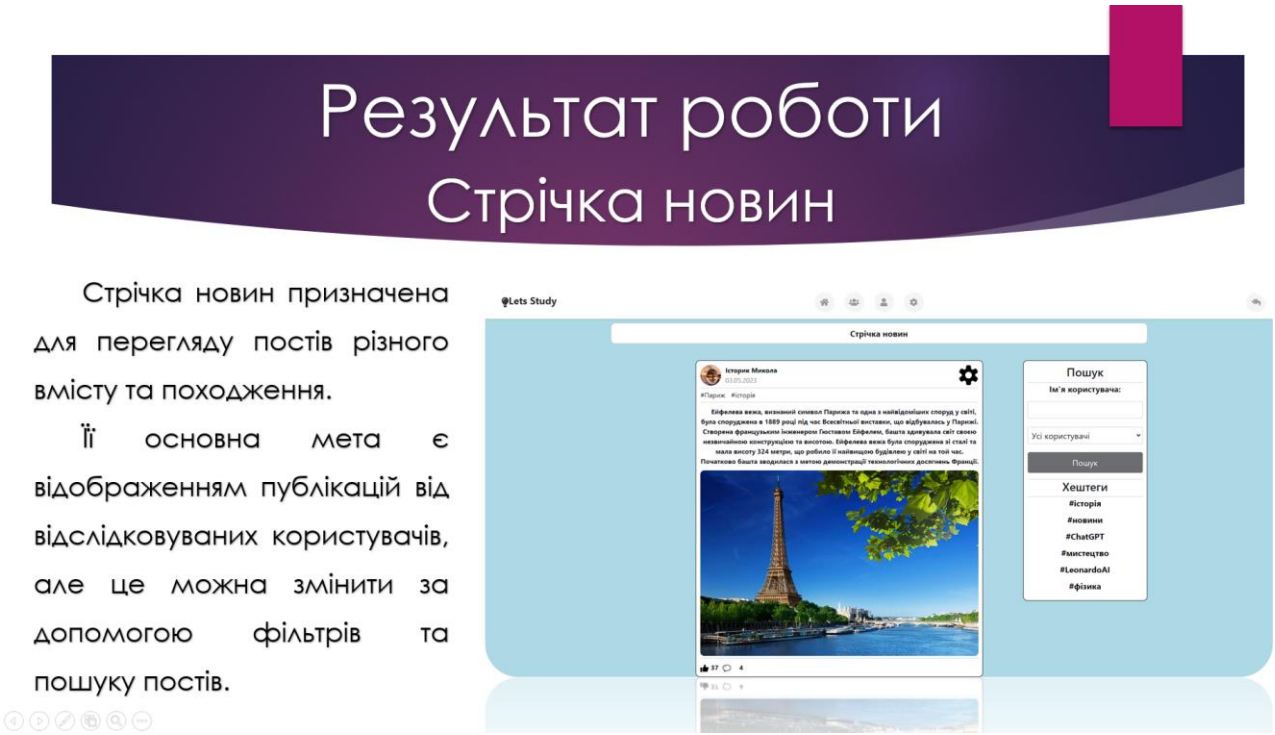


Рисунок Б.12 – Слайд №12

Результат роботи
Профіль користувача

Це особиста сторінка користувача на якій можна переглянути різноманітну інформацію про нього, усі його публікації та почати відслідковувати його активність.

При авторизації та відкритті власної сторінки на даному вікні відображається кнопка для створення нового посту.

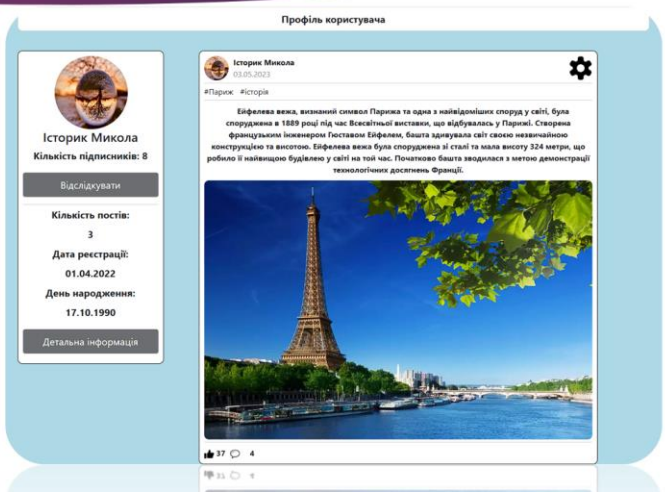


Рисунок Б.13 – Слайд №13

Результат роботи
Налаштування

Наявність багатьох параметрів налаштування особистих даних є важливим елементом будь якого програмного забезпечення.

З допомогою даного вікна користувачі можуть змінити особисті дані чи переглянути інформацію про інших

Рисунок Б.14 – Слайд №14

Результат роботи
Панель адміністратора

При авторизації за допомогою профіля користувача відкривається доступ до панелі адміністратора.

З її допомогою можна переглянути різноманітну статистику, яка пов'язана з активністю у соціальній мережі.

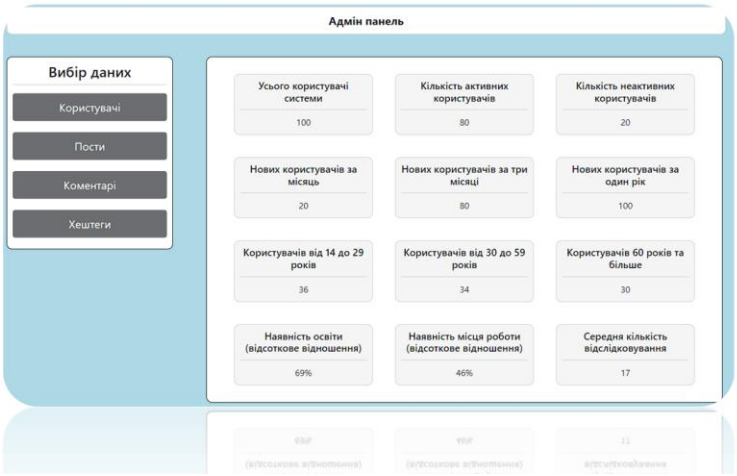


Рисунок Б.15 – Слайд №15

Тестування

Завершальним етапом будь-якого програмного забезпечення є проведення тестування.

Для даної кваліфікаційної роботи було вирішено провести Unit тестування та декілька видів ручного тестування ПЗ.

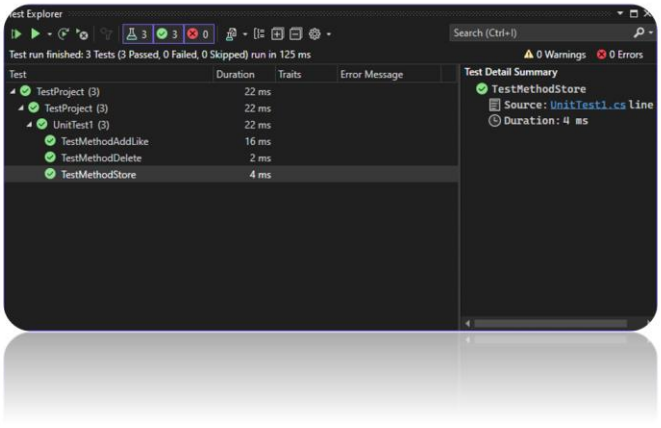


Рисунок Б.16 – Слайд №16

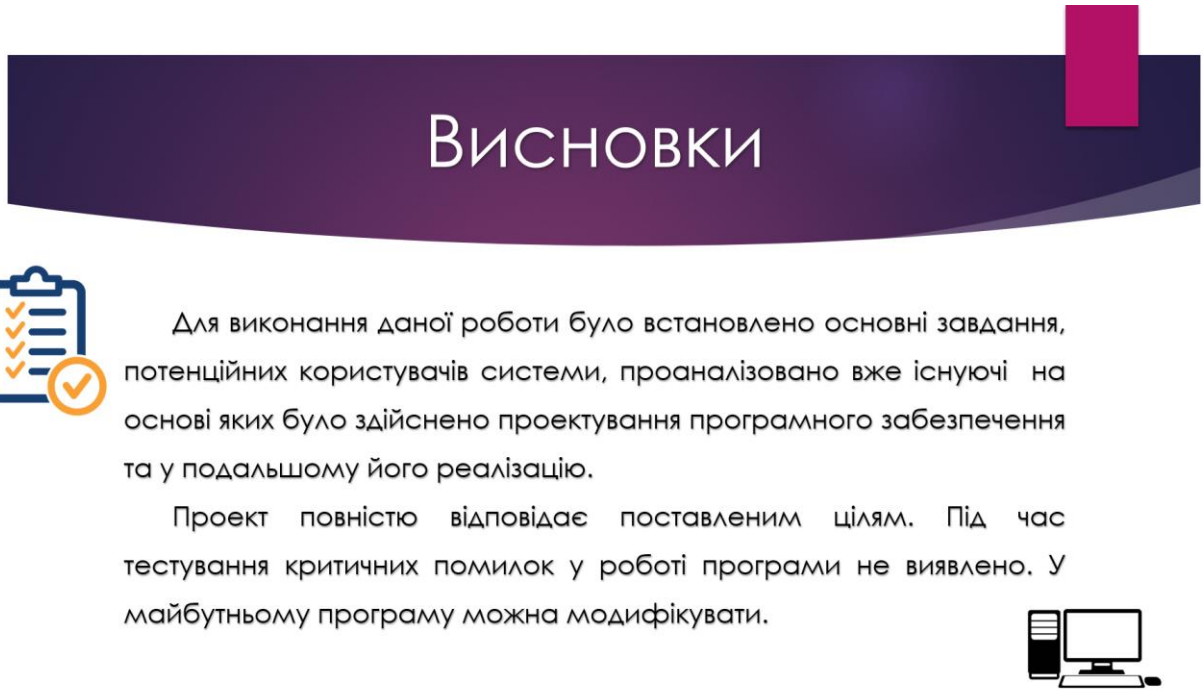


Рисунок Б.17 – Слайд №17

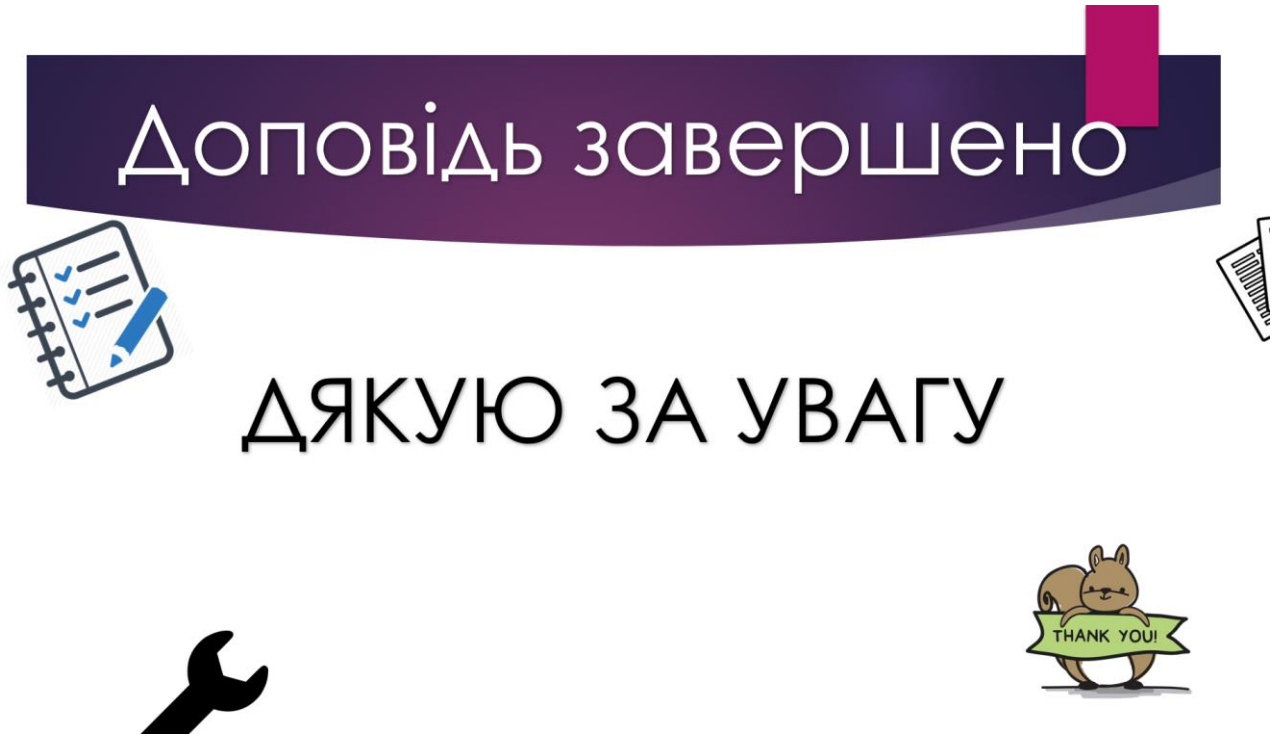
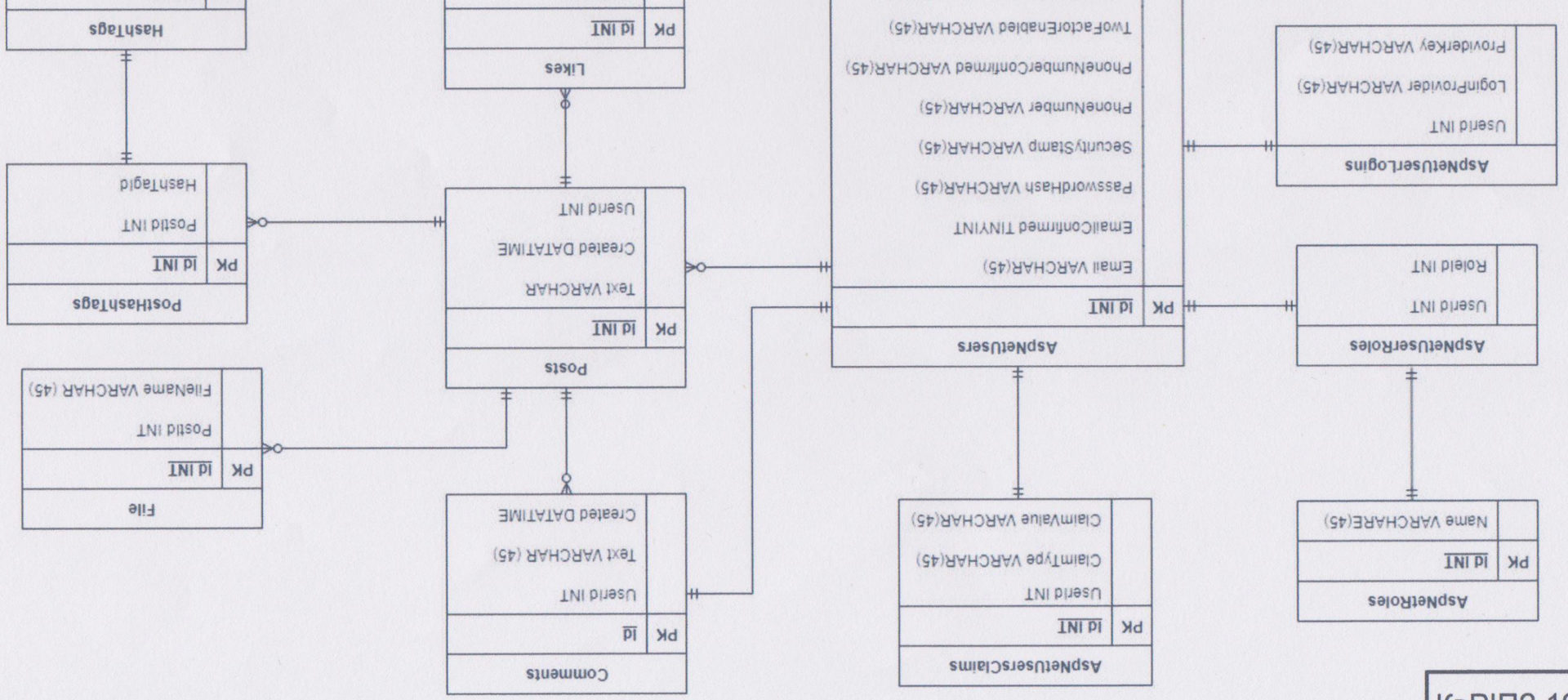


Рисунок Б.18 – Слайд №18

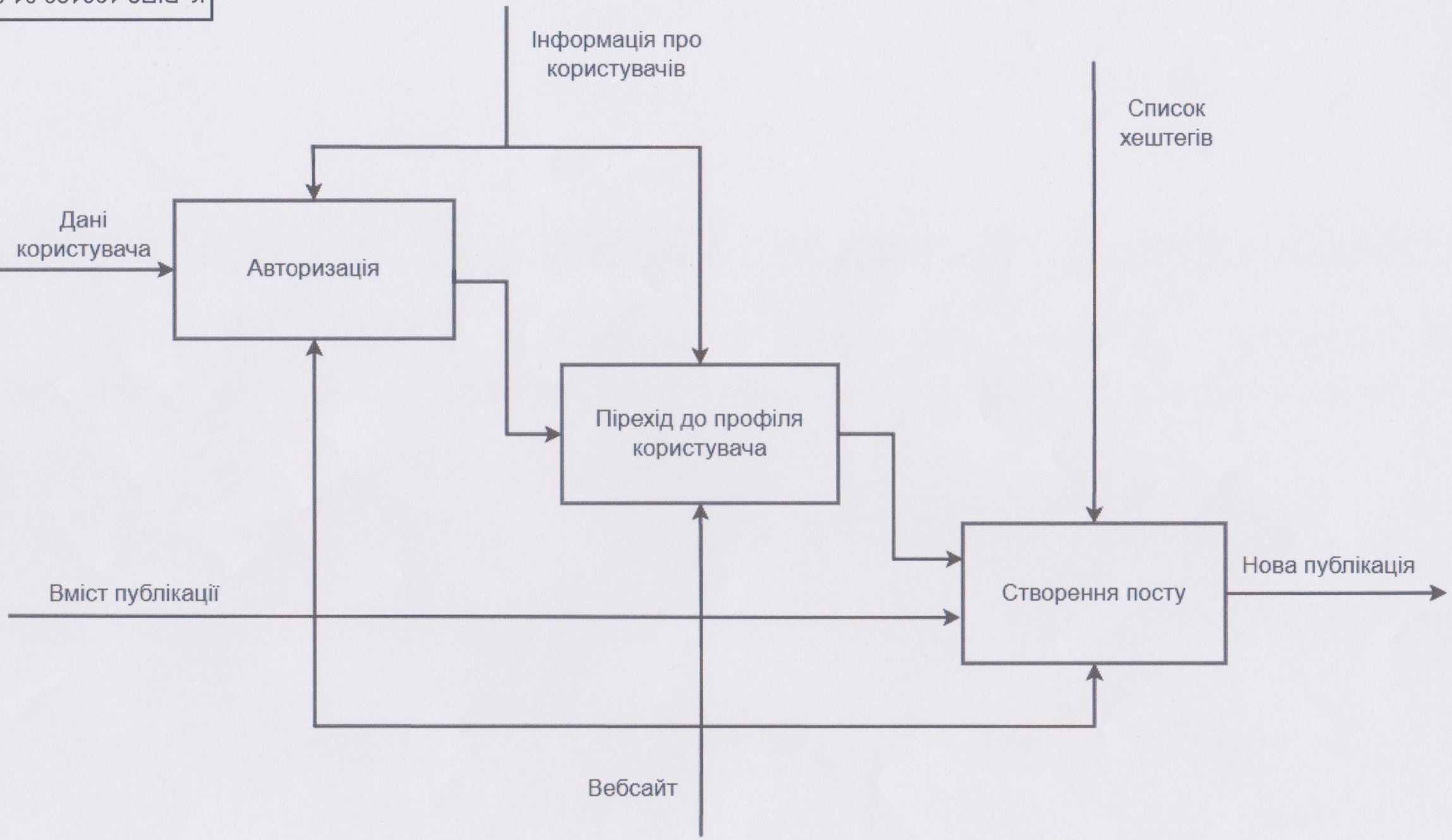
ГРАФІЧНА ЧАСТИНА

(обов’язкова)



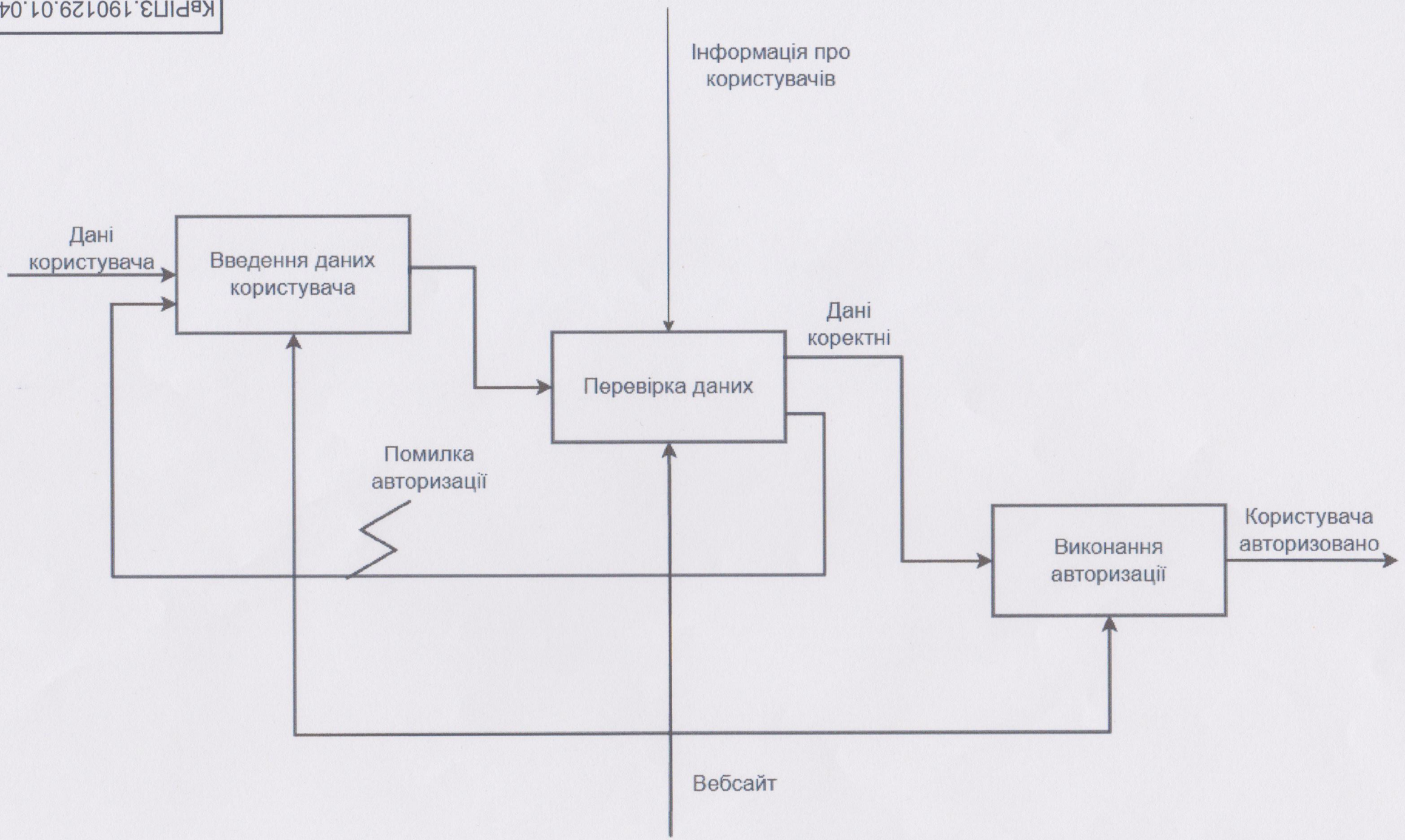
КвРІПЗ.190129.01.04.Е8

Зм.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.190129.01.04.Е8		
Розробив		Гура С. А.		12.4 - 5.06	ER-діаграма бази даних	Літера	Маса	Масштаб
Керівник		Гаврюшкіна Н. І.		5.06				
Консульт.						Аркуш 1	Аркушів 3	
Н. Контр.		Ящина О. М.		5.06				
Зав. каф.		Бедратюк Л. П.		5.06		ХНУ, ІПЗ-19-1		



					КвРІПЗ.190129.01.04.Е8	
Зм.	Арк.	№ докум.	Підпис	Дата	Діаграма декомпозиції першого рівня для процесу створення публікації	Літера / Маса / Масштаб
Розробив		Грига С. А.		5.06		
Керівник		Праворська Н. І.		5.06		Аркуш 2 / Аркушів 3
Консульт.						
Н. Контр.		Яшина О. М.		5.06		ХНУ, ІПЗ-19-1
Зав. каф.		Бедратюк Л. П.		5.06		

КвРІПЗ.190129.01.04.Е8



					КвРІПЗ.190129.01.04.Е8	
Зм.	Арк.	№ докум.	Підпис	Дата	Діаграма декомпозиції другого рівня для процесу авторизації	Літера / Маса / Масштаб
Розробив		Грига С. А.		5.06		
Керівник		Праворська Н. І.		5.06		Аркуш 3 / Аркушів 3
Консульт.						
Н. Контр.		Яшина О. М.		5.06		ХНУ, ІПЗ-19-1
Зав. каф.		Бедратюк Л. П.		5.06		

СУПРОВІДІНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти
Григи С. А.
Прізвище, ініціали

факультет ІТ, 4 курс, група ІПЗ-19-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності в Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і 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 й 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/або Anti-Plagiarism) і використання роботи для виявлення академічного 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.

05.02.2023
дата

підпис

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

Дата перевірки:
01.06.2023 17:56:18 EEST

Дата звіту:
01.06.2023 17:57:50 EEST

ID перевірки:
1015374264

Тип перевірки:
Doc vs Internet + Library

ID користувача:
100005589

Назва документа: **Кваліфікаційна_робота_Грига_ІПЗ_19_для_перевірки**

Кількість сторінок: **72** Кількість слів: **13051** Кількість символів: **101656** Розмір файлу: **3.01 MB** ID файлу: **1015040217**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

4.73%
Схожість

Найбільша схожість: **0.84%** з джерелом з Бібліотеки (ID файлу: **1011231407**)

3.92% Джерела з Інтернету	407	Сторінка 74
1.66% Джерела з Бібліотеки	70	Сторінка 76

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	17 сторінок

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 6%**

ID: 114488 Назва: БКР Соціальна мережа для збору даних про попит на ринку програмного забезпечення Додано в БД: 2023-06-01 Автора: Грига С.А. Керівники: Праворська Н.І. Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	88003	686	1673 (2%)	25 (4%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Грига Сергій Андрійович

Тема Соціальна мережа для збору даних про попит на ринку програмного забезпечення

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 ; кількість сторінок записки 63

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційні роботі було досліджено та проаналізовано предметну область, розглянуто вже існуючі на ринку рішення на прикладі соціальних мереж Facebook та Twitter та встановлено їхні переваги та недоліки, на основі чого було визначено вимоги до програмного забезпечення. Також було розглянуто відомі технології з наведенням їхнього порівняння, на базі чого було вибрано найбільш доцільний варіант, розроблено структуру програмного продукту та прототип інтерфейсу користувача, що у подальшому було використано при розробці програми. По завершенню розробки було проведено ряд тестувань для перевірки відповідності вимог та коректності роботи, за результатами чого можна стверджувати, що програмне забезпечення готове до експлуатації.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до усіх поставлених вимог та завдань.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі було доведено актуальність теми роботи, встановлено мету та основні завдання. У першому розділі було детально проаналізовано предметну область, розглянуто вже існуючі рішення з наведенням їхніх переваг та недоліків, на основі чого було здійснено постановку задач та вимог. У другому розділі було проведено аналіз та порівняння існуючих фреймворків, на основі чого було обрано найбільш підходящий для даної роботи варіант у вигляді ASP.NET Core з використанням клієнт-серверної та MVC архітектури та додаткових технологій. Також було розроблено прототип інтерфейсу користувача та здійснено проектування структури проекту з розробкою відповідних діаграм. У третьому розділі було виконано програмну реалізацію з наведенням коду основних модулів та їх детальним описом, у результаті чого було розроблено відповідний програмний продукт. Також було проведено ряд тестувань, а саме Unit тести та декілька видів ручного тестування для перевірки на коректність роботи та відповідність вимогам.

4. Позитивні сторони роботи Актуальність теми кваліфікаційної роботи було детально обґрунтовано на основі наведених фактів. При аналізі вже існуючих рішень на ринку програмного забезпечення було розглянуто різні статистичні данні. При виборі засобів розробки було виконано детальний опис технологій з наведенням переваг та недоліків та принципів їхньої роботи.

5. Негативні сторони роботи Реалізована система пошуку може бути покращена за рахунок додання додаткових параметрів пошуку. Також доцільно реалізувати систему різноманітних фільтрів для кращого та більш зручного перегляду статистичних даних з можливістю їх пошуку

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано у вигляді рисунків і діаграм та відповідає темі кваліфікаційної роботи. Пояснювальна записка виконана відповідно до вимог та чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує на позитивну оцінку. Матеріал пояснювальної записки структурований та послідовний, що дозволяє зрозуміти увесь викладений матеріал за відповідною тематикою, а наведений графічний матеріал наочно демонструє усі аспекти виконаної роботи.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі відповідно до усіх поставлених вимог та завдань і заслуговує на оцінку «відмінно».

РЕЦЕНЗЕНТ Бобровнікова Кіра Юліївна, кандидат технічних наук, доцент кафедри комп'ютерної інженерії та інформаційних систем (ККІС) ХНУ.

"06" 06 2023 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів подібності щодо роботи, продукованими програмно-технічним засобом (ами) перевірки текстів на плагіат:

Назва: «Соціальна мережа для збору даних про попит на ринку програмного забезпечення»

Автор: Грига Сергій Андрійович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Праворська Наталія Іванівна, кандидат педагогічних наук, доцент
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. 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 й 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:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, відомість документів), у структурі змісту, назвах розділів/підрозділів тощо, у назвах публікацій у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посилання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 4,73% і адресується до 477 джерел, що, з урахуванням наведених обгрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 6.06.23

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Наталія ПРАВОРСЬКА