ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Кафедра інженерії програмного забезпечення

**КУРСОВИЙ ПРОЕКТ**

з дисципліни

. «Вебтехнології» .

.(назва дисципліни)

на тему: Соціальна мережа Y

**КПІПЗ.2401188.01.07.ПЗ**

(шифр, назва)

Рівень вищої освіти             Перший (бакалаврський)

Галузь знань                      12 «Інформаційні технології»

Спеціальність     121 «Інженерія програмного забезпечення»

Освітня програма   «Інженерія програмного забезпечення»

Студента. I курсу, група . ІПЗс-24-1. \_\_\_\_\_\_\_\_\_\_\_ Максим ЛАПКО .

(шифр) (підпис) (ініціали, прізвище)

Керівник. канд. техн. наук, доцент . \_\_\_\_\_\_\_\_\_\_ Оксана ЯШИНА.

(посада, вчене звання, науковий ступінь) (підпис) (ініціали, прізвище)

Кількість балів \_\_\_\_\_\_\_

Оцінка за шкалою:

національною\_\_\_\_\_\_\_\_\_/ЄКТС\_\_\_\_

Члени комісії: \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (ініціали, прізвище)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (ініціали, прізвище)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (ініціали, прізвище)

Хмельницький, 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

*Спеціальність.* 121 «Інженерія програмного забезпечення»

Шифр, назва

ЗАТВЕРДЖУЮ

Завідувач кафедри \_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_202\_\_\_ року

**ЗАВДАННЯ**

**НА КУРСОВИЙ ПРОЕКТ**

            Лапко М. О.

Прізвище, ім’я, по батькові студента

1. Тема проекту Соціальна мережа Y

керівник проекту Яшина Оксана Миколаївна канд. техн. наук, доцент

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

2. Строк подання студентом проекту на кафедру . 27.11.2024 .

3. Вихідні дані до проекту: методичні вказівки до курсового проектування для студентів спеціальності 121 «Інженерія програмного забезпечення»» з курсу «Вебтехнології»

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) дослідження предметної області, проектування структури застосування, детальне проектування, розробка структури інформаційної системи, розробка модулів програми

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслень)… Рисунки (0 шт.), Таблиці (0 шт.)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Дата видачі завдання\_\_07.09.2024\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

………………...................................................................................................==…………………………………….....

**КАЛЕНДАРНИЙ ПЛАН**

виконання курсового проектування

з дисципліни «Вебтехнології»

на І-й семестр 2024-2025 навчального року

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів курсового проекту | Термін виконання етапів проекту  (тиждень) | Відмітка про виконання |
| *1* | *Одержання теми курсового проекту та завдання* | *1* |  |
| *2* | *Вступ* | *2* |  |
| *3* | *Особливості та обмеження реалізації проекту* | *3* |  |
| *4* | *Аналіз існуючих програмних продуктів* | *4* |  |
| *5* | *Проектування зовнішнього вигляду сайту та навігації* | *5* |  |
| *6* | *Проектування БД* | *6* |  |
| *7* | *Вибір засобів для розробки сайту* | *7* |  |
| *8* | *Розробка структури проекту* | *8* |  |
| *9* | *Підключення та розробка БД* | *9* |  |
| *10* | *Розробка зовнішнього вигляду сайту* | *10-11* |  |
| *11* | *Розробка програмних модулів* | *12-13* |  |
| *12* | *Тестування проекту* | *14* |  |
| *13* | *Опрацювання висновків та переліку літератури* | *15* |  |
| *14* | *Оформлення пояснювальної записки* | *16* |  |
| *15* | *Захист* | *17* |  |

#### Керівник проекту Яшина О.М. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

#### Затверджено:

#### Зав. кафедри ІПЗ Бедратюк Л.П. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (підпис)

**АНОТАЦІЯ**

Курсовий проект «Соціальна мережа Y».

Автор роботи: Лапко М.О.

Керівник роботи: Яшина О.М.

Обсяг – 0 с., 0 рис., 0 табл., 0 додатки, 0 джерел.

Мета курсового проекту: розробити та реалізувати соціальну мережу з базовими функціональними можливостями для комунікації так взаємодії між користувачами.

У курсовому проекті виконана розробка веб-сервісу із застосуванням мови програмування Go для серверної частини, з використанням фреймворків REST та gRPC для комунікації між мікросервісами, а також MySQL для зберігання даних. Інтерфейс реалізовано на React для забезпечення інтерактивної взаємодії користувачів.

Дата Підпис

ЗМІСТ

*Змн.*

*Арк.*

*№ докум.*

*Підпис*

*Дата*

*Арк.*

*5*

КПІПЗ.2401188.01.07.ПЗ

*Розроб.*

*Лапко М.О.*

*Перевір.*

*Яшина О.М.*

*Реценз.*

*Н. Контр.*

*Затверд.*

Соціальна мережа Y

*Літ.*

*Акрушів*

69

*ХНУ. ІПЗс-24-1*

Перелік скорочень 6

ВСТУП 7

1 Дослідження предметної області 9

1.1 Характеристика функціональної структури предметної області 9

1.2 Аналіз останніх публікацій, досліджень та існуючих рішень 11

1.3 Аналіз вимог до програмного забезпечення. 14

1.4 Розробка технічного завдання 18

2. Проектування та визначення та визначення методів і засобів розробки 20

2.1 Проектування архітектури ПЗ 20

2.2 Проектування бази даних 24

2.3 Модель проектування 32

2.4 Вибір засобів розробки 36

3. Програмна реалізація 39

3.1 Структура та функціональне призначення модулів ПЗ, їх взаємозв’язок. 35

3.2 Розробка програмних модулів 37

3.3 Інструкція користувача 40

3.4 Вимоги до програмно-технічного середовища функціонування ПЗ 42

4 Тестування ПЗ 43

4.1 Методика тестування ПЗ 43

4.1 Результати тестування ПЗ 44

ВИСНОВОК 45

Перелік посилань 47

Додаток А 50

## ПЕРЕЛІК СКОРОЧЕНЬ

|  |  |  |
| --- | --- | --- |
| БД | – | База даних |
| SQL | **–** | Structured Query Language |
| ORM | – | Object-Relational Mapping |
| СКБД | – | Система керування базами даних |
| ПЗ | – | Програмне забезпечення |
| IDE | – | Integrated Development Environment |
| UML | **–** | Unified Modeling Language |
| Backend | – | Серверна частина застосунку |
| Frontend | – | Клієнтська частина застосунку |
| API | – | Application Programming Interface |
| HTTP | – | HyperText Transfer Protocol |
| gRPC | – | Remote Procedure Call |
| REST | – | Representational State Transfer |
| JSON | – | JavaScript Object Notation |
| JWT | – | JSON Web Token |

## ВСТУП

## У сучасному світі, де цифрові технології та соціальні платформи відіграють важливу роль у повсякденному житті, існує постійна потреба у нових інструментах для комунікації, взаємодії та обміну інформацією. Саме соціальні мережі стали важливими майданчиками для обміну думками, ідеями, контентом та досвідом. З урахуванням цих тенденцій виникає необхідність створення інноваційної соціальної платформи, яка забезпечить користувачам зручні й ефективні можливості для взаємодії.

## Курсовий проект спрямований на розробку соціальної мережі "Y" — веб-сервісу, що дозволить користувачам ділитися власними думками, створювати контент, обмінюватися ідеями та досвідом. Основною метою є створення простору, де користувачі зможуть легко знаходити спільні інтереси, підтримувати зв’язок, ділитися контентом та будувати спільноту.

## Об'єктом дослідження є процес розробки веб-сервісу для соціальної мережі, який включає технічні, дизайнерські та функціональні аспекти створення такої платформи. Це передбачає розробку надійної серверної архітектури, розробку клієнтського інтерфейсу для зручної взаємодії з користувачем, забезпечення безпеки даних та захисту приватної інформації користувачів.

## Предметом дослідження є ключові аспекти розробки соціальної мережі, зокрема технічні рішення, архітектурні особливості, дизайн інтерфейсу, а також вплив соціальної платформи на користувачів та її роль у підтримці спільноти.

## Завдання дослідження включають наступне: створення ефективної архітектури платформи, розробку зручного користувацького інтерфейсу, реалізацію функцій реєстрації та управління обліковими записами, забезпечення безпеки та конфіденційності, розробку механізмів пошуку, фільтрації та взаємодії з контентом, а також тестування та оптимізацію продукту.

## Зміст роботи полягає в аналізі сучасних трендів у сфері соціальних мереж, огляді інструментів для розробки веб-додатків, проектуванні бази даних для зберігання інформації про користувачів та контент, а також у створенні користувацького інтерфейсу й інтерактивних функцій.

## Мета роботи – розробити та реалізувати соціальну мережу "Y", яка забезпечить ефективні засоби для спілкування та обміну контентом між користувачами.

## Практична значимість роботи полягає у створенні зручного інструменту для спілкування, що відповідає потребам сучасних користувачів та сприяє формуванню активної спільноти.

1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Характеристика функціональної структури предметної області

У сучасному інформаційному суспільстві соціальні мережі стали невід'ємною частиною комунікації та обміну інформацією між людьми. У зв'язку із зростаючою популярністю соціальних платформ виникає потреба створити соціальну мережу, яка забезпечить користувачам зручний і безпечний спосіб ділитися контентом, обговорювати теми, що цікавлять, і формувати спільноти за інтересами. Щоб детально зрозуміти вимоги до такої платформи, необхідно провести аналіз предметної області, визначити основні поняття, розглянути невирішені питання та проблеми, враховуючи потреби користувачів.

Предметна область соціальної мережі охоплює не тільки механізми публікації та обміну контентом, а й взаємодію між користувачами через коментарі, оцінки (лайки), повідомлення та інші соціальні функції. Аналізуючи структуру соціальної мережі, важливо враховувати типи контенту, який створюватиметься на платформі, а також особливості взаємодії між користувачами, зокрема можливості для персоналізації та модерації контенту.

Функціональні аспекти охоплюють процеси публікації постів, пошуку, фільтрації контенту, а також механізми для забезпечення безпеки та конфіденційності. Необхідно також забезпечити зручний інтерфейс, який полегшить користувачам навігацію на сайті, та впровадити надійні методи захисту даних для запобігання несанкціонованому доступу.

Загальний аналіз цих аспектів дозволить створити більш глибоке розуміння предметної області та визначити ключові вимоги до розробки соціальної мережі, що сприятиме успішній реалізації проєкту. Таким чином, було виділено наступні етапи та об'єкти для автоматизації в соціальній мережі "Y":

* автоматизація процесу створення та публікації постів;
* автоматизація пошуку та фільтрації контенту за темами або ключовими словами;
* автоматизація системи сповіщень для інформування користувачів про оновлення.

Необхідність автоматизації цих функцій виникає з потреби спростити взаємодію користувачів із платформою та забезпечити ефективний механізм взаємодії з контентом.

Для досягнення мети проєкту визначено наступні завдання:

* аналіз потреб користувачів у сфері обміну інформацією та створення контенту;
* проєктування структури соціальної мережі та її функціональності;
* розробка системи реєстрації та управління обліковими записами;
* створення бази даних для зберігання даних про користувачів та контенту;
* реалізація програмних модулів для публікації та пошуку контенту;
* розробка користувацького інтерфейсу;
* створення технічних вимог та інструкції користувача.

Очікувані результати включають:

* розроблену соціальну мережу, яка дозволить користувачам публікувати та переглядати контент;
* впроваджену систему автентифікації для захисту облікових записів;
* створену базу даних із необхідною інформацією для оптимальної роботи платформи;
* підготовлену інструкцію користувача, що пояснює правила користування соціальною мережею.

1.2 Аналіз останніх публікацій, досліджень та існуючих рішень

Аналіз існуючих рішень у контексті дослідження предметної області та постановки задач є важливим етапом, адже це дозволяє не лише зрозуміти поточний стан галузі та визначити основні тенденції, але й знайти нові ідеї для нашого проекту.

Процес аналізу можна умовно розділити на декілька етапів:

1. Визначення цілей та завдань аналізу.
2. Вибір джерел інформації.
3. Збір та аналіз інформації.
4. Формулювання висновків.

Перш за все було сформульовано цілі та завдання аналізу, з урахуванням інформації про проект та предметну область. У результаті аналізу мали бути визначені наявні рішення, що потенційно задовольняють потреби нашого проекту, а також їхні переваги й недоліки. Це допоможе знайти найкращі аспекти для інтеграції у нашому рішенні, а також виявити недоліки, які слід уникати.

Джерелом інформації стали популярні платформи, що вже охоплюють частини нашої предметної області. Одним із прикладів є «X.com»[1] – сучасний сервіс обміну інформацією і контентом, що наголошує на швидкості обміну та інтерактивності для своїх користувачів. На основі цього сервісу було проведено детальний аналіз функціональності й можливостей, які можуть бути застосовані для нашого проекту.

Основна інформація про мету «X.com» подається на сторінці «Про нас» і звучить як «швидка і зручна платформа для обміну інформацією та новинами». Це загальний підхід, але він має декілька особливостей, що варті уваги. Наприклад, «X.com» дозволяє користувачам створювати контент, слідкувати за іншими користувачами, взаємодіяти з дописами у форматі вподобань та коментарів, що створює спільноту навколо певних тем та інтересів.

Користувачі на «X.com» можуть отримувати доступ до стрічки з актуальним контентом та використовувати інструменти для пошуку релевантних дописів, які сортуються за алгоритмами популярності. Цей підхід дозволяє забезпечити динамічний і швидкий доступ до найпопулярнішої інформації, що є ключовим для сервісу, зосередженого на актуальних темах (див. Рисунок 1.2.1).

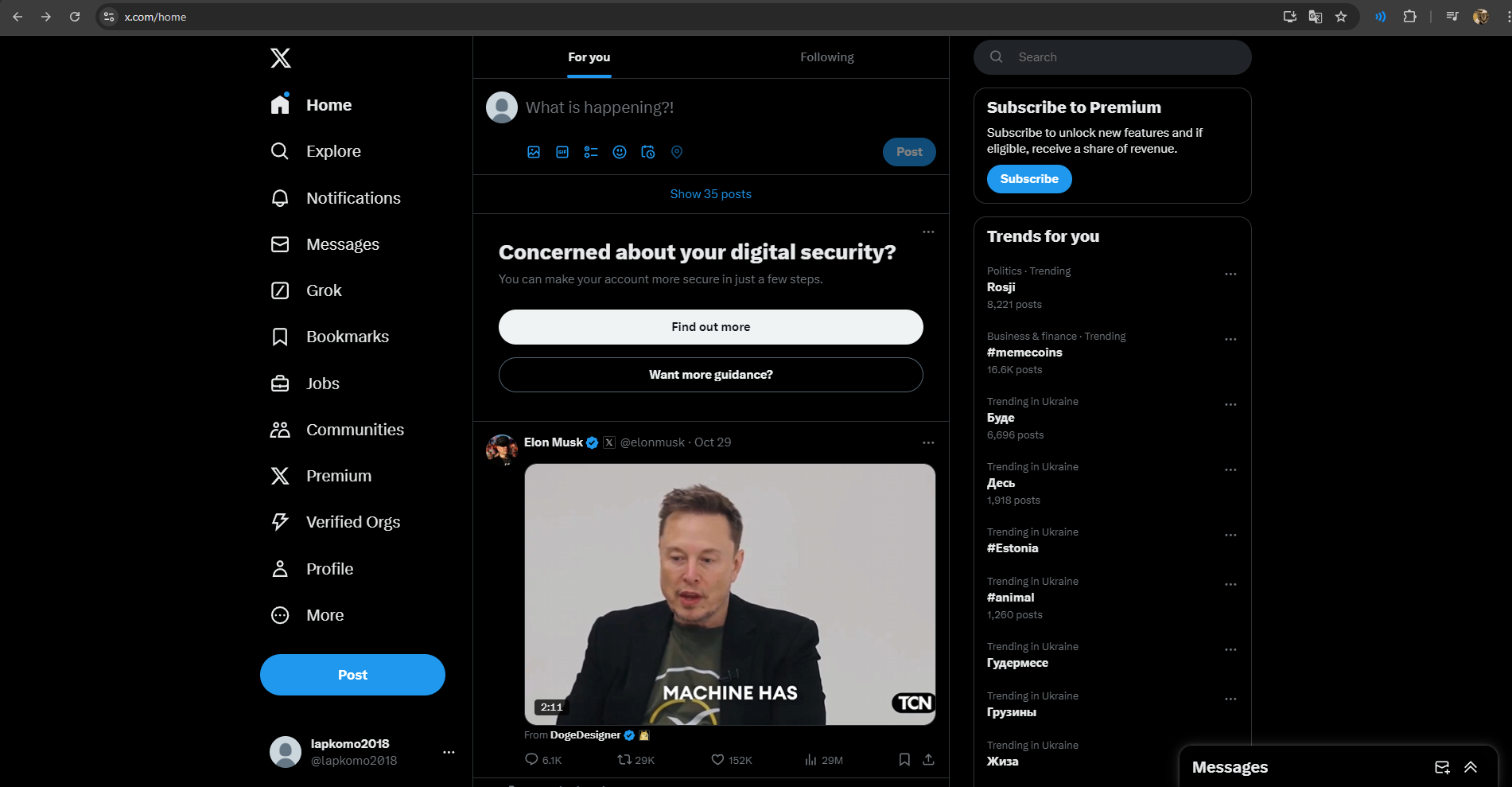


Рисунок 1.2.1 —Головна сторінка веб-сервісу «X.com»

Основним функціоналом для користувачів сервісу є можливість створення нових дописів, взаємодія з контентом та можливість створення зв’язків у спільноті шляхом підписки на інші профілі.

Важливою складовою функціоналу «X.com» є також підтримка обміну повідомленнями між користувачами, що сприяє налагодженню комунікації між користувачами, але не фокусується на умовах і критеріях обміну.

Основні підсумки щодо аналізу функціональності «X.com» такі: хоча цей сервіс відповідає частині вимог нашого проекту (швидкий обмін інформацією, інтерактивність, пошукові функції), його структура зосереджена на соціальних взаємодіях. Такий функціонал, як обмін на певну кількість одиниць іншого ресурсу, відсутній, оскільки сервіс орієнтований на соціальну взаємодію.

Отже, проведений аналіз показав, що хоча деякі елементи функціоналу «X.com» можуть бути корисними для побудови зручного інтерфейсу і пошукової системи, для досягнення мети нашого проекту слід застосовувати індивідуальний підхід. Це дозволить створити рішення, що забезпечує саме рівноцінний обмін контентом, відслідковування користувачів, а також уможливлює модерацію за допомогою репортів.

1.3 Аналіз вимог до програмного забезпечення та розробка технічного завдання

Виконання аналізу вимог до програмного забезпечення та розробка технічного завдання є невід'ємною складовою процесу розробки програмних продуктів. У сучасному інформаційному суспільстві, де програмне забезпечення використовується в різних сферах діяльності, важливо забезпечити якість та відповідність програмного продукту вимогам його користувачів. Чітке розуміння вимог є основою для успішного проектування та розробки програмного забезпечення.

Аналіз вимог до програмного забезпечення передбачає збір, аналіз та документування потреб користувачів. Цей процес вимагає взаємодії зі зацікавленими сторонами, включаючи замовників, кінцевих користувачів та фахівців з інформаційних технологій. Він допомагає зрозуміти функціональні та нефункціональні вимоги до програмного продукту, його обмеження та контекст використання. Результатом аналізу вимог є документ, який описує всі виявлені вимоги та їх пріоритети.

Для веб-сервісу "Social Network Y" було проведено формування бізнес-моделі, яка визначає стратегію реалізації проекту та його комерційний потенціал. Бізнес-модель окреслює цілі проекту, його мету і способи досягнення успіху, а також розкриває ключові аспекти, такі як цільова аудиторія, цінова політика, канали розповсюдження та прибуткові потоки.

Перш за все, було визначено потенційних користувачів системи. Для веб-сервісу "Social Network Y" потенційними користувачами є люди, які цікавляться спілкуванням, обміном інформацією та взаємодією з іншими користувачами. Цільова аудиторія сервісу охоплює людей віком від 18 років, які проживають в Україні та інших країнах світу.

Цінність сервісу полягає в забезпеченні зручного та доступного способу взаємодії між користувачами, включаючи можливість створення дописів, коментування, лайкання та обміну повідомленнями. Основними каналами залучення користувачів до "Social Network Y" є веб-сайт та мобільні додатки, а також можливість співпраці з партнерами для реклами.

Основними ресурсами, якими володітиме веб-сервіс, будуть: інфраструктура проекту (веб-сайт, база даних, сервер), програмне забезпечення та контент сервісу. Ключовими потенційними партнерами можуть бути: маркетологи, агенції з реклами та сервіси доставки.

Для визначення користувацьких вимог було вирішено провести опитування з метою збору інформації про потреби та побажання потенційних користувачів веб-сервісу "Social Network Y". Опитування допомогло визначити, які функції та можливості повинні бути включені в сервіс, щоб він був максимально корисним і зручним для користувачів. Для проведення опитування було використано онлайн-інструменти, такі як Google Forms, а також соціальні мережі та месенджери.

Основні питання для проведення опитування:

1. Як часто ви користуєтеся соціальними мережами?
2. Які функції соціальних мереж ви вважаєте найбільш важливими?
3. Чи є у вас досвід використання соціальних мереж для обміну інформацією або комунікації?
4. Які можливості ви хотіли б бачити у соціальній мережі?
5. Чи зацікавлені ви у функціях, що дозволяють адміністрування та модерування контенту?

За результатами проведення опитування були отримані наступні відповіді від потенційних користувачів системи.

Як часто ви користуєтеся соціальними мережами?

* Щодня. Я заходжу в соціальні мережі кожен день, щоб перевірити новини та оновлення від друзів.
* Кілька разів на тиждень. Я заходжу, коли маю вільний час або щось цікаве.
* Рідко. Я не використовую соціальні мережі часто, тільки коли це необхідно.
* Ніколи. Я не користуюсь соціальними мережами.

Які функції соціальних мереж ви вважаєте найбільш важливими?

* Створення та публікація контенту. Можливість ділитися своїми думками та фото.
* Лайки та коментарі. Можливість взаємодії з контентом інших користувачів.
* Інструменти для адміністрування. Можливість керувати контентом і взаємодією в групах.
* Групи та спільноти. Можливість приєднуватися до інтересів та обговорювати теми з однодумцями.
* Обмін повідомленнями. Можливість спілкуватися з друзями в реальному часі.

Чи є у вас досвід використання соціальних мереж для обміну інформацією або комунікації?

* Так. Я активно використовую соціальні мережі для спілкування з друзями та обміну інформацією.
* Частково. Я іноді використовую їх для обміну інформацією, але не часто.
* Ні. Я не використовую соціальні мережі для обміну інформацією.

Які можливості ви хотіли б бачити у соціальній мережі?

* Більш детальна система фільтрації контенту. Я хотів би мати можливість налаштувати, що я бачу у своїй стрічці.
* Інструменти для створення подій. Я хотів би організовувати та запрошувати друзів на події.
* Розширений пошук. Я хотів би легше знаходити користувачів, пости та групи за інтересами.
* Можливість редагувати свої пости. Я хотів би мати можливість змінювати свої публікації після їх створення.
* Анонімність. Я хотів би мати можливість використовувати платформу анонімно для певних обговорень.

Чи зацікавлені ви у функціях, що дозволяють адміністрування та модерування контенту?

* Так. Я вважаю, що адміністрування і модерування важливі для підтримки порядку в соціальних мережах.
* Можливо. Це цікаво, але я не знаю, наскільки це важливо для мене.
* Ні. Я не цікавлюсь адмініструванням та модеруванням контенту.

Відповіді на ці питання можуть допомогти зрозуміти потреби користувачів у вашій соціальній мережі "Social Network Y". Наприклад, якщо більшість респондентів використовують соціальні мережі щодня, це вказує на високий попит на функціональність платформи. Визначення важливих функцій, таких як публікація та коментування контенту, дозволить зосередитися на їх розробці. А також відгуки про адміністрування та модерування контенту можуть вказати на потребу у функціях для забезпечення безпеки та комфорту користувачів.

1.4 Розробка технічного завдання

Технічне завдання — це документ, який визначає основні вимоги до майбутнього продукту або послуги. Він містить інформацію про призначення продукту, його функціональні та нефункціональні вимоги, технічні характеристики, вимоги до безпеки та якості, а також інші важливі відомості про програмну систему. Завдяки цьому документу забезпечується узгодження основних вимог та функціональності системи.

Існує кілька стандартів або шаблонів, що регламентують зміст, загальну структуру та наповнення документа. Вибір стандарту залежить від специфікації конкретного проекту, що обумовлює широкий перелік доступних шаблонів, таких як:

* IEEE STD 830-1998
* ISO/IEC/IEEE 29148-2011
* ISO/IEC/IEEE 29148-2018
* RUP
* SWEBOK
* BABOK

Після аналізу переваг і недоліків кожного з шаблонів, було прийнято рішення використовувати шаблон ISO/IEC/IEEE 29148:2018 для розробки технічного завдання програмної системи. Стандарт IEEE 29148-2018, також відомий як ISO/IEC/IEEE 29148:2018, є міжнародним стандартом, що визначає процеси і продукти, пов’язані з розробкою вимог для систем та програмних продуктів. Він охоплює весь життєвий цикл розробки вимог — від їх збору до аналізу, уточнення, затвердження та управління. Основні процеси, визначені стандартом, включають: – збір вимог; – аналіз вимог; – затвердження вимог; – управління вимогами.

Стандарт також передбачає два обов’язкові шаблони для специфікації вимог, які слід використовувати при розробці системи: – System Requirements Specification (SyRS) — шаблон для визначення технічних вимог до системи та забезпечення її взаємодії з користувачем. – Software Requirements Specification (SRS) — шаблон для визначення функціональних і нефункціональних вимог до програмного забезпечення.

Отже, для створення технічного завдання було обрано шаблон SRS, оскільки він найбільш підходить для опису функціональних та нефункціональних вимог до програмного забезпечення, що дозволить точно охарактеризувати систему. Обраний шаблон допускає мінімальні зміни в структурі та змісті документа, тому технічне завдання міститиме лише незначні корективи, які забезпечать точнішу відповідність темі та меті проекту. Ці зміни стосуватимуться, зокрема, опису вимог, що будуть адаптовані до специфічних потреб проекту, а також уточнення нефункціональних вимог з урахуванням умов експлуатації системи. Ознайомитися з технічним завданням можна у Додатку …

2. ПРОЕКТУВАННЯ ТА ВИЗНАЧЕННЯ МЕТОДІВ І ЗАСОБІВ РОЗРОБКИ

2.1 Проектування архітектури ПЗ

## У рамках проектування архітектури соціальної мережі визначення архітектури програмного забезпечення є критично важливим етапом. Загальна архітектура системи формує структуру програмного забезпечення, що включає розподіл компонентів, їхню взаємодію та загальний механізм роботи системи. Вона також містить ключові рішення щодо організації програмної системи, зокрема вибір структурних елементів і їх інтерфейсів, а також поведінку, що визначена у взаємодії між цими елементами. Однією з основних функцій архітектури програмного забезпечення є подолання труднощів, що виникають при проектуванні і розробці складних програмних систем.

## Тому перед початком розробки необхідно чітко визначити правильну архітектуру програмного забезпечення, щоб забезпечити надійність і фундаментальну організацію системи. Спочатку було розглянуто основні види архітектури, кожна з яких має свої переваги та недоліки, що частково визначають можливості масштабування і стабільної роботи системи. В рамках інженерії програмного забезпечення існує багато архітектурних підходів, але в даному проекті розглянуто основні з них.

## Архітектура клієнт-сервер (Client-Server Architecture) [2]. Ця архітектура базується на принципі, що користувач (клієнт) взаємодіє з сервером, надсилаючи запити і отримуючи відповідь з необхідними даними чи послугою. Ця модель може функціонувати як на одному пристрої, так і через велику мережу.

## Переваги:

## висока масштабованість.

## поліпшена безпека.

## висока надійність.

## Недоліки:

## залежність від доступності сервера.

## можливість перевантаження сервера.

## Монолітна архітектура (Monolithic Architecture) [3]. Весь додаток складається з одного великого блоку, де всі компоненти і залежності інтегровані разом, що полегшує розробку та розгортання.

## Переваги:

## простота розробки.

## легкість в утриманні та відлагодженні.

## одинична точка розгортання.

## Недоліки:

## обмежена масштабованість;

## складність при внесенні змін;

## можливість витрат на невикористані ресурси.

## Мікросервісна архітектура (Microservices Architecture) [4]. Додаток розбивається на невеликі, незалежні сервіси, кожен з яких відповідає за конкретну функціональність, що полегшує масштабованість та розвиток додатку.

## Переваги:

## висока масштабованість.

## висока гнучкість.

## швидкість розробки.

## незалежність сервісів.

## ефективне використання ресурсів.

## Недоліки:

## складність управління комунікацією між сервісами.

## складність керування даними.

## складність відлагодження та моніторингу.

## Аналізуючи переваги та недоліки розглянутих архітектур програмного забезпечення, можна зробити висновок, що для розроблюваної соціальної мережі найкраще підходить мікросервісна архітектура. Цей тип архітектури дозволяє поділити систему на невеликі, незалежні сервіси, кожен з яких відповідає за певну функціональність. Мікросервісна архітектура надає низку переваг у контексті створення веб-сервісу, який забезпечує обмін інформацією та взаємодію між користувачами.

## По-перше, ця архітектура дозволяє підвищити масштабованість і гнучкість, оскільки кожен мікросервіс можна розробляти, розгортати та масштабувати незалежно від інших. Це також спрощує підтримку і розвиток кожної окремої частини системи, що полегшує внесення змін і впровадження нових функцій.

## Крім того, розподіл функцій між окремими мікросервісами дозволяє оптимізувати ресурси, а також підвищити надійність системи — відмова одного сервісу не впливає на роботу інших компонентів. Кожен сервіс має свою базу даних, що полегшує управління даними і забезпечує автономність даних кожного мікросервісу.

## У контексті обраної архітектури важливо визначити ключові терміни:

## Мікросервіс: Незалежний сервіс, який виконує конкретну функцію в системі, обробляє запити від клієнтів або інших мікросервісів, і повертає відповідь.

## Клієнт: Користувацька частина системи, яка надсилає запити до мікросервісів та обробляє отримані відповіді.

## Запит: Спосіб, яким клієнт передає інструкції або дані мікросервісу для отримання відповіді.

## Відповідь: Інформація або ресурс, який мікросервіс повертає клієнту чи іншому мікросервісу у відповідь на його запит.

## Протокол зв'язку: Специфікація, що визначає правила обміну даними між клієнтом і мікросервісами (наприклад, REST або gRPC).

## Загалом основну ідею мікросервісної архітектури можна представити у вигляді рисунка (див. Рисунок 2.1.1).

## 

## Рисунок 2.1.1 — Концепція мікросервісної архітектури

## Таким чином, було розглянуто різні архітектурні підходи для проекту соціальної мережі, і обрано мікросервісну архітектуру через численні переваги, що роблять її найбільш відповідною для розробки сервісу.

2.2 Проектування бази даних

## На основі обраної мікросервісної архітектури та вимог до системи було розроблено проект бази даних. Для цього використано модель «сутність-зв’язок», що допомогла визначити сутності, атрибути та зв’язки між ними, а також зрозуміти, як дані пов’язані між собою. Це дозволило спроектувати базу даних, яка буде ефективною та зручною для використання в умовах розподіленої архітектури.

## Діаграма ER (Entity-Relationship) [5] — це графічне представлення сутностей та їхніх зв’язків один з одним у базі даних. Вона є концептуальним інструментом, який використовується в розробці бази даних для опису даних та зв’язків, важливих для організації даних. ER-діаграма відображає зв’язки між сутностями, які визначають, як дані (атрибути) в цих сутностях взаємопов’язані. На ER-діаграмі зв’язки зображуються лініями, що з’єднують сутності.

## Основні типи зв’язків між сутностями включають:

## Один-до-одного (1:1): один екземпляр однієї сутності може бути пов’язаний з одним екземпляром іншої сутності.

## Один-до-багатьох (1): один екземпляр однієї сутності може бути пов’язаний з багатьма екземплярами іншої сутності.

## Багато-до-одного (N:1): багато екземплярів однієї сутності можуть бути пов’язані з одним екземпляром іншої сутності.

## Багато-до-багатьох (M): багато екземплярів однієї сутності можуть бути пов’язані з багатьма екземплярами іншої сутності.

## Загалом модель сутність-зв’язок може бути представлена в різних нотаціях, зокрема в нотації Бахмана, Петера Чена, IDEF1X, Вороняча лапка тощо. У контексті обраного проекту для розробки бази даних використано нотацію IDEF1X [6], оскільки вона відповідає як особистим уподобанням, так і об’єктивним вимогам проекту щодо обробки та зберігання даних для забезпечення коректної роботи сервісу.

## IDEF1X — це метод проектування баз даних, який дозволяє створити концептуальну схему даних через ідентифікацію сутностей та зв’язків між ними. Концептуальна схема є абстрактним зображенням даних, незалежним від конкретної програми або фізичного представлення. Моделі IDEF1X складаються з таких основних компонентів:

## Сутності: представляють реальні об’єкти або концепції.

## Атрибути: визначають характеристики об’єкта.

## Відносини: визначають зв’язки між об’єктами.

## 

## Рисунок 2.2.1 — Компоненти нотації IDEF1X

## Для формування початкових таблиць було розглянуто, як система мала функціонувати в загальному сценарії, що дало уявлення про сутності та атрибути, необхідні для проектування. В основі процесу лежали сформовані функціональні вимоги до соціальної мережі. Сценарій, на основі якого здійснювалося проектування:

## Користувач реєструється на платформі, вказуючи особисті дані.

## Користувач авторизується, щоб отримати доступ до основних функцій.

## Користувач створює та публікує нові дописи.

## Користувач коментує дописи інших користувачів.

## Користувач ставить позначки «подобається» на дописи.

## Користувач редагує власний профіль (аватар, нікнейм, електронну адресу).

## Адміністратор перевіряє звіти про порушення та модерує дописи.

## Далі було виділено імена об'єктів у кожному пункті сценарію. Вони стали кандидатами на сутності для подальшого етапу проектування бази даних. На основі загального сценарію роботи системи було визначено шість основних сутностей, які зберігаються в різних базах даних для підвищення масштабованості та незалежності мікросервісів:

## Користувачі — зберігається в БД auth service.

## Профіль — зберігається в БД user service.

## Пости — зберігається в БД post service.

## Коментарі — зберігається в БД post service.

## Лайки — зберігається в БД post service.

## Репорти — зберігається в БД report service.

## Кожен із зазначених мікросервісів використовує окрему базу даних для управління своїми специфічними даними, що забезпечує автономність та розширюваність кожного з компонентів. Після цього сценарій був розбитий на окремі компоненти для виявлення потенційних атрибути всіх сутностей. Опис цих атрибутів наведено за допомогою неформального тексту. Цей неформальний опис атрибутів дає загальне розуміння того, які дані можуть бути використані для моделювання цих сутностей.

## Користувач реєструється в системі:

## При реєстрації користувач вказує username, електронну адресу, пароль.

## Система присвоює користувачеві унікальний ідентифікатор.

## Дані зберігаються у БД auth service та user service.

## Користувач створює та публікує пост:

## Користувач вказує заголовок, текст, прикріплює зображення, відео.

## Пости отримують унікальний ідентифікатор.

## Дані про пости зберігаються у БД post service.

## Користувач коментує дописи інших користувачів:

## Користувач вводить текст коментаря до допису.

## Коментарі отримують унікальний ідентифікатор та прив’язку до поста і користувача.

## Дані про коментарі зберігаються у БД post service.

## Користувач ставить позначки «подобається»:

## При взаємодії з постом зберігається ID користувача, ID поста.

## Система зберігає інформацію про кожну позначку «подобається» у БД post service.

## Користувач редагує профіль:

## Користувач змінює аватар, нікнейм, електронну пошту, username.

## Дані про профіль зберігаються в окремій БД user service та обновляється в auth service якщо змінюєтся почта чи username.

## Адміністратор перевіряє репорти про порушення:

## Система зберігає репорти, які користувачі подають на інші пости.

## Звіти містять ID користувача, ID поста, причину, відповідь та статус.

## Дані про звіти зберігаються у БД report service.

## Із наявної інформації було виділено окремі сутності із притаманними їм атрибутами. Сутності із атрибутами були представлені у вигляді таблиць та заповнені орієнтовними даними.

## Сутності в auth service

## Таблиця 1. Сутність «User»

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| id | email | username | password | is\_activated | created\_at | updated\_at | deleted\_at |
| 1 | lapkomo2018@gmail.com | lapkomo2018 | hashedPassword | 1 | datetime | datetime | datetime |

## Таблиця 2. Сутність «Activation\_Token»

|  |  |
| --- | --- |
| User\_id | token |
| 1 | randomtoken |

## Таблиця 3. Сутність «Refresh\_Token»

|  |  |
| --- | --- |
| User\_id | token |
| 2 | jwtToken |

## Сутності в user service

## Таблиця 4. Сутність «User»

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| id | email | username | nickname | avatar | created\_at | updated\_at | deleted\_at |
| 1 | lapkomo2018@gmail.com | lapkomo2018 | lapkomo2018 | static image url | datetime | datetime | datetime |

## Таблиця 5. Сутність «Ban»

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| id | user\_id | banned\_by | ban\_reason | banned\_at | duration | expired\_at | unban\_reason | unbanned\_by | unbanned\_at | created\_at | updated\_at | deleted\_at |
| 1 | 2 | 2 | abuse | datetime | time.Duration | datetime | reason | 2 | datetime | datetime | datetime | datetime |

## Сутності в post service

## Таблиця 6. Сутність «Post»

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| id | user\_id | title | content | image\_urls | video\_urls | posted\_at | edited\_by | created\_at | updated\_at | deleted\_at |
| 1 | 2 | Hello | Hello World | massive of static image urls | massive of static video urls | datetime | 0 | datetime | datetime | datetime |

## Таблиця 7. Сутність «Like»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | user\_id | post\_id | created\_at | deleted\_at |
| 1 | 2 | 1 | datetime | datetime |

## Таблиця 8. Сутність «Comment»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| id | user\_id | post\_id | content | edited\_by | created\_at | deleted\_at |
| 1 | 2 | 1 | wow | 0 | datetime | datetime |

## Таблиця 9. Сутність «Tag»

|  |  |
| --- | --- |
| id | name |
| 1 | Hello |

## Сутності в report service

## Таблиця 10. Сутність «Report»

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| id | user\_id | post\_id | reason | admin\_id | answer | status | created\_at | updated\_at | deleted\_at |
| 1 | 2 | 1 | reason | 0 |  | pending | datetime | datetime | datetime |

## Після визначення основних сутностей і їх атрибутів було прийнято рішення встановити зв’язки між ними, з урахуванням вимог до функціональності системи.

## Спочатку розглянемо зв’язок між сутностями "User" у сервісах auth і user. Взаємозв’язок між цими сутностями формується через унікальні ідентифікатори користувачів та основні атрибути для аутентифікації та профілювання, такі як email, username, password (у auth сервісі) і додаткові атрибути, як nickname, avatar (у user сервісі). Основний зв’язок – «один користувач має один профіль», що вказує на зв’язок типу 1:1.

## Далі, аналізуються сутності у post сервісі, зокрема "User" і "Post". Зв'язок між ними формулюється так: «Один користувач може створювати безліч постів, і кожен пост належить одному користувачу». Це вказує на зв’язок «один до багатьох» 1:N.

## Наступний зв’язок — між сутностями "Post" та "Tag", де кожен пост може бути позначений кількома тегами, а кожен тег може бути пов’язаний із кількома постами. Це вказує на зв’язок «багато до багатьох» M:N. Для нормалізації бази даних було створено проміжну таблицю post\_tags, яка містить ідентифікатори поста та тегу, що дозволяє зберігати зв’язок між ними як два окремих зв’язки «один до багатьох» M:N між post\_tags і Post та між post\_tags і Tag.

## Сутність "Like" також пов’язана з "User" та "Post". Охарактеризувати цей зв'язок можна так: «Один користувач може вподобати безліч постів, і кожен пост може мати безліч вподобань від різних користувачів». Це знову є зв’язок «багато до багатьох», і для цього передбачено таблицю Like.

## У сервісі report основний зв’язок — між сутностями "Report" і "Post". Один користувач може надіслати декілька скарг на різні пости, тому тут знову маємо зв’язок «один до багатьох» 1:N. Крім того, кожен звіт може бути оброблений різними адміністраторами, що вказує на зв’язок «один до багатьох» між сутностями "Report" і "User".

## Таким чином, усі сутності були інтегровані у відповідних сервісах, що сприяє зручному розподіленню даних і полегшує роботу з різними базами даних для кожного сервісу.

## Отже, було проведено комплексне проектування бази даних. Спочатку обрано відповідні методології та нотації для проектування, після чого визначено основні сутності й атрибути. На основі попередньо сформованого сценарію роботи системи були відображені зв’язки між сутностями в межах кожного сервісу.

## Такий підхід забезпечив ефективну організацію та збереження даних, забезпечуючи оптимальний доступ до них та високу продуктивність системи. Вірно встановлені зв’язки та оптимізована структура бази даних дозволили сервісам системи взаємодіяти з користувачами та іншими компонентами максимально ефективно, сприяючи надійній та масштабованій архітектурі.

2.3 Модель проектування

## Модель проєктування — це абстрактне уявлення системи, яке використовується під час її розробки для опису структури, поведінки та взаємодії компонентів. Оскільки модель проєктування базується на вимогах до системи, визначених під час етапу аналізу, для створення такого представлення доцільно використовувати інструменти, які є невід’ємною частиною уніфікованого процесу розробки програмного забезпечення, зокрема UML-діаграми.

## UML-діаграми [7] — це тип візуального представлення, що використовується для моделювання й документування програмних систем. Вони по суті є графічними схемами, які допомагають розробникам зрозуміти і візуалізувати структуру та поведінку системи. Залежно від рівня абстракції або деталізації проєкту можуть використовуватися різні типи діаграм, які в загальному можна представити у вигляді структури UML-діаграм (див. Рисунок 2.3.1).

## Рисунок 2.3.1 — Структура діаграм UML

## undefined

## Рисунок 2.3.1 — Структура діаграм UML

## На цьому етапі проєктування було вирішено почати з опису загальної поведінки та особливостей системи, зокрема функцій та взаємодій між потенційними користувачами та системою. Для цього доречним стало використання одного з інструментів моделювання — діаграми варіантів використання.

## UML-діаграма варіантів використання [8] — це вид UML-діаграми, який застосовується для розробки структури інформаційної системи та демонстрації її загального функціоналу. Ця діаграма не передбачає деталізації конкретних методів, які будуть використовуватися в системі; тому при створенні діаграми необхідно врахувати всі функціональні вимоги та загальні вимоги до системи (див. Рисунок 2.3.2).

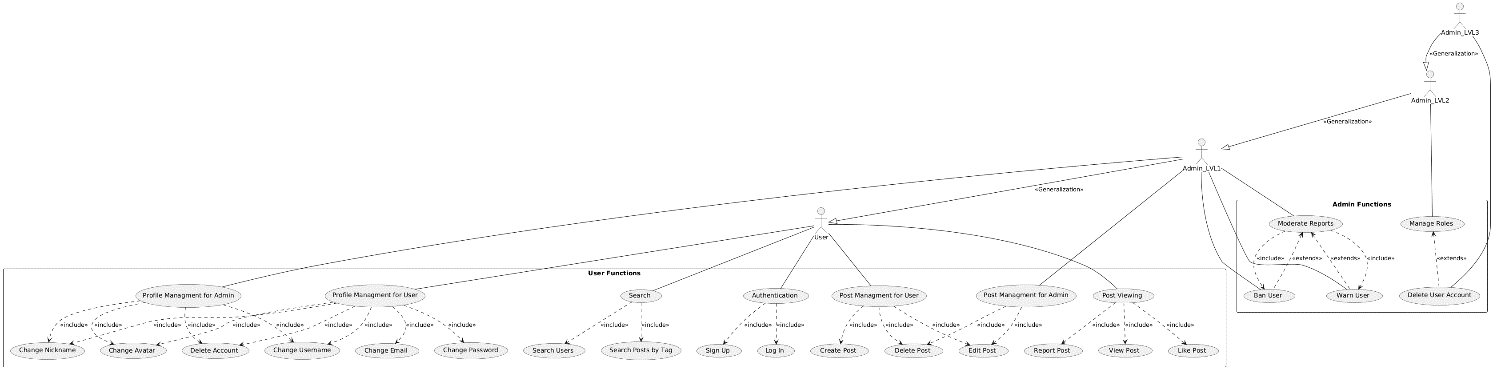


Рисунок 2.3.2 — Діаграма Use Case для соціальної мережі

Після опису загальної поведінки системи було детальніше проаналізовано процес покрокової взаємодії компонентів сервісу, для чого була використана діаграма послідовності.

UML-діаграма послідовності [9] — це тип діаграми UML, що ілюструє порядок повідомлень між об'єктами в процесі їхньої взаємодії. Діаграма послідовності складається з групи об'єктів, представлених лініями життя, та повідомлень, якими вони обмінюються протягом взаємодії. Для наочності використано спрощену діаграму, яка демонструє основні кроки взаємодії (див. Рисунок 2.3.3).

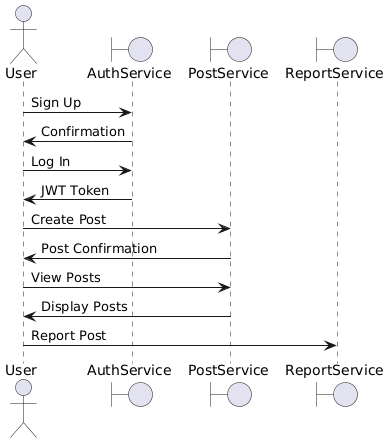


Рисунок 2.3.3 — Діаграма послідовності для соціальної мережі

У межах деталізації аналізу поведінки системи було розглянуто, як саме система виконує поставлені завдання, що були продемонстровані на діаграмі послідовностей. Для цього було використано ще один інструмент UML — діаграму активності.

UML-діаграма активності [19] є типом діаграми UML, яка використовується для опису динамічної поведінки системи. Вона відображає потік управління через систему та взаємодії між різними етапами діяльності за допомогою блок-схем.

Були створені та описані діаграми активності для основних процесів у головних сценаріях роботи системи. Як приклад наведено діаграму процесу модерації, що виконується адміном (див. Рисунок 2.3.4).

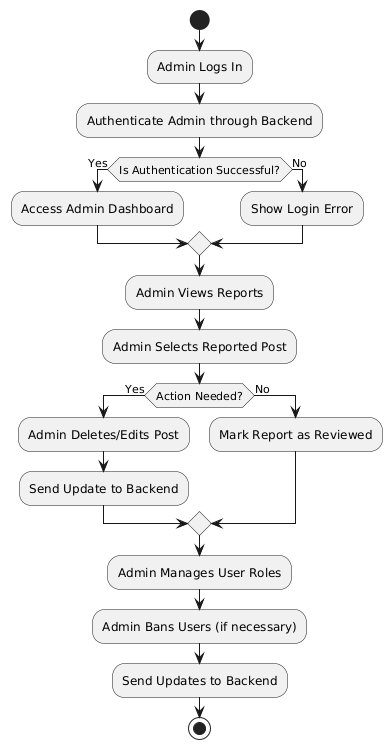


Рисунок 2.3.4 — Діаграма активності для процесу «Модерації» адміном

Таким чином, розробивши діаграми активності для основних функцій системи, було детально описано процес виконання різних дій усередині програмної системи та проаналізовано структуру роботи продукту на конкретному прикладі. Це дозволило створити детальну та комплексну модель програмної системи, що відображає ключові процеси, такі як реєстрація користувачів, модерація постів та управління користувачами в соціальній мережі.

2.4 Вибір засобів розробки

Розробка будь-якої програмної системи є складним і багатогранним процесом, що вимагає використання різноманітних інструментів і технологій. Тому, обираючи засоби розробки системи, важливо враховувати характеристики найпопулярніших мов програмування та інших технологій для створення веб-сервісів. Це допоможе зробити оптимальний вибір, який забезпечить ефективну та якісну розробку.

У контексті вибору засобів для створення соціальної мережі було розглянуто, порівняно та обрано ключові компоненти: мова програмування для бекенду, фреймворк для фронтенду та система керування базами даних (СКБД). Також було розглянуто можливість використання додаткових компонентів, таких як бібліотеки та сторонні сервіси для інтеграції.

Вибір засобів розробки розпочато з визначення мови програмування для бекенду, оскільки цей компонент є фундаментальним для розробки будь-якої системи. Було обрано Go, яка є однією з найкращих мов для створення серверної частини веб-сервісів завдяки своїй високій продуктивності та простоті використання.

Go [11] — це сучасна мова програмування, розроблена компанією Google, яка підходить для створення надійних і продуктивних веб-додатків. Вона відрізняється простим синтаксисом, високою швидкістю виконання та вбудованою підтримкою паралельного програмування.

Плюси:

* Висока продуктивність.
* Простий і зрозумілий синтаксис.
* Вбудована підтримка паралелізму.
* Підтримка численних бібліотек і фреймворків.

Мінуси:

* Менша кількість сторонніх бібліотек у порівнянні з іншими мовами.
* Порівняно нова, тому деякі підходи можуть бути менш зрозумілими для новачків.

Для фронтенду було обрано React [12], популярну бібліотеку JavaScript для створення користувацьких інтерфейсів. React дозволяє ефективно розробляти інтерактивні веб-додатки з компонентним підходом, що спрощує розробку та підтримку проекту.

Плюси:

* Компонентний підхід, що дозволяє повторно використовувати код.
* Велика спільнота та безліч бібліотек для інтеграції.
* Підтримка стану додатка через контекст та хуки.

Мінуси:

* Потребує додаткових знань про JavaScript та JSX.
* Може бути перевантаженим для простих проектів.

## Як систему керування базами даних (СКБД) було обрано MySQL [13], одну з найпопулярніших реляційних баз даних, що забезпечує високу продуктивність, надійність та гнучкість у зберіганні даних. MySQL добре інтегрується з Go і React, що робить її ідеальним вибором для даної соціальної мережі.

## Плюси:

## Висока продуктивність запитів.

## Широка підтримка спільноти та документації.

## Гнучка система управління даними.

## Мінуси:

## Може бути складною для налаштування при роботі з великими обсягами даних.

## Обмежена підтримка деяких типів даних у порівнянні з NoSQL базами даних.

## Таким чином, після детального аналізу обраних мов програмування та технологій було вирішено використовувати Go для розробки бекенду соціальної мережі, React для створення фронтенду та MySQL як СКБД. Цей вибір забезпечить високу продуктивність, зручність у розробці та легкість у підтримці проекту, що особливо важливо для соціальної мережі, де взаємодія з користувачами та обробка даних є критичними аспектами.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Програмні модулі, їх структура та призначення

## Програмний модуль [30] — це логічно-незалежна частина програмного забезпечення, яка виконує певну функцію і може бути використаний як самостійно, так і в складі інших програм. Модулі використовуються для модулювання програмного забезпечення, тобто для поділу програми на логічно пов'язані блоки, що полегшує розробку, тестування та обслуговування програмного забезпечення. Для визначення програмних модулів було виконати декомпозицію загального сценарію роботи програми.

## Існує декілька способів декомпозиції програми на модулі:

## технологічний спосіб;

## функціональний спосіб;

## декомпозиція з використанням абстрактних типів даних;

## принцип приховування інформації.

## Технологічний спосіб. При технологічному способі декомпозиції програми на модулі алгоритм програми розбивається на фрагменти, які відповідають основним технологічним операціям з обробки даних.

## Функціональний спосіб. При функціональному способі декомпозиції програми на модулі алгоритм програми розбивається на фрагменти, які відповідають функціям задачі, що автоматизується.

## Декомпозиція з використанням абстрактних типів даних. При декомпозиції з використанням абстрактних типів даних алгоритм програми розбивається на фрагменти, які відповідають структурам даних і об'єднані в одному модулі всі процедури їх обробки.

## Принцип приховування інформації. Принцип приховування інформації заснований на тому, що модуль реалізує (приховує) одне проектне рішення. При цьому інформація про проектне рішення поділяється на загальнодоступну та приховану від користувачів інформацію.

## Оскільки, функціональний підхід передбачає, що кожен модуль програми відповідає за одну конкретну функцію, то це дозволяє легко визначити відповідальність кожного компонента MVC-патерну, обраного при виборі засобів розробки. Таким чином, було визначено, що функціональний підхід найкраще підходить для декомпозиції загального сценарію роботи програми при застосуванні MVC-патерну, оскільки він дозволяє забезпечити ефективний розподіл відповідальності між трьома основними компонентами цього патерну, а саме: представленнями, контролерами та моделями.

## Для декомпозиції загального сценарію роботи програми на окремі модулі були повторно визначені основні функції системи, до яких відносяться:

## реєстрація та авторизація користувача;

## створення профілю користувача, його заповнення;

## додавання, редагування та видалення книг;

## пошук книг за категоріями;

## обмін книгами;

## надсилання повідомлень.

## Таким чином за функціональним підходом відповідні функції системи було розділено на наступні програмні модулі:

## модуль ідентифікації користувача (управляючий);

## модуль управління користувачами (робочий);

## модуль управління книгами (робочий);

## модуль обміну книгами (робочий);

## модуль обміну повідомленнями (робочий);

## модуль конфігурації (допоміжний).

## Таким чином, було сформовано список модулів системи та описано основну їхню структуру та призначення. Цей крок був важливим етапом розробки програмного забезпечення, оскільки він дозволив полегшити розробку та подальше тестування програмної системи.

3.2 Розробка програмного продукту

## Після визначення основних модулів проекту був виконаний опис безпосередньої розробки цих модулів, оскільки розробка програмного продукту — це доволі складний та багатогранний процес, який складається із безлічі кроків та послідовних дій, було вирішено для наочності розглянути розробку декількох основних модулів системи у хронологічній послідовності розробки, які реалізують обраний MVC-патерн та в сукупності становлять готовий продукт.

## Було вирішено почати із розгляду модулю «Управління книгами». Відповідно до правил та норм розробки систем із застосуванням MVC-патерну розробку потрібно починати із створення моделей, які визначають дані, які будуть використовуватися при роботі із системою.

## Код моделі «Book»:

public class Book

{

public int BookId { get; set; }

public string Name { get; set; }

public string Author { get; set; }

public string? ImageUrl { get; set; }

public IFormFile ImageFile { get; set; }

public string? OwnerId { get; set; }

public ApplicationUser? Owner { get; set; }

public Guid? BookLink { get; set; }

public string? Description { get; set; }

public DateTime? CreatedAt { get; set; }

public bool? Exchangeable { get; set; }

public virtual ICollection<Wishlist> Wishlists { get; set; }

public List<BookCategory> BookCategories { get; set; }

}

## Загалом, ця модель буде використовуватися упродовж повного циклу роботи системи тому і допускається модифікація цієї моделі упродовж процесу розробки програмного продукту.

Таким чином було отримано наступний набір властивостей цієї моделі, який включає в себе: унікальний ідентифікатор книги (BookId), назву книги (Name), автора книги (Author), посилання на фото книги (ProfilePicture), безпосередньо саме фото книги (ImageFile), ідентифікатор власника книги (OwnerId), екземпляр класу власника книги (Owner), посилання на індивідуальну сторінку книги (BookLink), опис книги (Description), дату додавання книги (CreatedAt), статус обміну, тобто чи книга доступна для обміну (Exchangeable), колекція списків бажаного, у які додали книгу (Wishlists) та колекція категорій, до яких відноситься книга (BookCategories). Також модель містить один метод, який виконує завантаження категорій для поточної книги.

Метод завантаження категорій:

public void LoadCategories(DatabaseContext context)

{

this.BookCategories = context.BookCategories

.Where(bc => bc.BookId == BookId)

.Include(bc => bc.Category)

.ToList();

}

Цей метод відповідає за завантаження списку категорій, пов'язаних з конкретною книгою (визначеною змінною BookId). Він виконує наступні дії:

* Фільтрує категорії: Використовується метод Where(bc => bc.BookId == BookId), щоб отримати тільки категорії, пов'язані з книгою, що має ідентифікатор BookId;
* Завантажує пов'язані дані: Використовується метод Include(bc => bc.Category), щоб разом з кожною категорією завантажити інформацію про відповідну категорію з таблиці Categories бази даних.
* Присвоює список категорій: Отриманий список категорій присвоюється змінній this.BookCategories, що дозволяє використовувати цей список в інших методах класу.

Після визначення основних властивостей моделі, було виконано розробку контролеру для виконання функцій додавання, видалення, редагування та перегляду книги у сервісі. Для цього були створені відповідний контролер під назвою «BookController». Для початку було розглянуто основний конструктор класу контролера.

Конструктор класу:

public BookController(DatabaseContext context, IFileService fileService, UserManager<ApplicationUser> userManager)

{

\_context = context;

\_fileService = fileService;

\_userManager = userManager;

}

Цей конструктор класу BookController відповідає за ініціалізацію залежностей, необхідних для роботи контролера. Він приймає три аргументи:

* DatabaseContext context: Екземпляр класу DatabaseContext, який забезпечує доступ до бази даних.
* IFileService fileService: Екземпляр інтерфейсу IFileService, який надає методи для роботи з файлами.
* UserManager<ApplicationUser> userManager: Екземпляр класу UserManager<ApplicationUser>, який використовується для управління користувачами програми.

У контролері було визначено декілька дій, які користувач може робити із книгами, а саме перелік операцій типу СRUD та стандартний метод Index для відображення книг у представленні:

Код методу Index():

public IActionResult Index()

{

if (User.IsInRole("Admin"))

{

var books = \_context.Books

.Include(b => b.Owner)

.ToList();

return View(books);

}

var currentUserBooks = \_context.Books.Where(b => b.OwnerId == \_userManager.GetUserId(User));

return View(currentUserBooks);

}

Цей метод відповідає за отримання даних про книги та відображення їх у представленні. Він виконує наступні дії:

* Якщо користувач є адміністратором (User.IsInRole("Admin")), то виконується блок коду, який отримує всі книги з бази даних, включаючи інформацію про власників книг (Include(b => b.Owner)). Після цього получений список книг передається представленню View.
* Якщо користувач не є адміністратором, то виконується блок коду, який отримує книги, що належать лише поточному користувачу (currentUserBooks). Для цього використовується метод GetUserId(User) класу UserManager, щоб отримати ID користувача, а потім фільтруються книги по власнику (Where(b => b.OwnerId == \_userManager.GetUserId(User))). Отриманий список книг також передається представленню View.
* Відображення даних: Незалежно від ролі користувача, отриманий список книг передається представленню View, яке відповідає за відображення інформації про книги користувачу. Представлення може використовувати відомості про власників книг, якщо вони доступні, для відображення додаткової інформації користувачу.

Для початку було розглянуто метод, який відповідає за підготовку всіх необхідних даних для користувача для додавання книги.

Код методу AddBook():

[Authorize(Roles = "Admin,User")]

[HttpGet]

public ActionResult AddBook()

{

var model = new Book();

model.BookCategories = new List<BookCategory>();

ViewBag.Categories = \_context.Categories.ToList();

return View(model);

}

Цей метод відповідає за підготовку даних для додавання нової книги та відображення форми додавання користувачу.

Важливим уточненням є той факт, що для кожної із дій над книгами було передбачено два методи, які реалізовують GET та POST запити. GET-запит використовується відповідно для отримання даних, POST-запит для відправки даних на сервер. Таким чином було розглянуто метод, який відповідає за відправку введених користувачем даних про нову книга на сервер.

Код методу AddBook():

[Authorize(Roles = "Admin,User")]

[HttpPost]

public async Task<ActionResult> AddBook(Book book)

{

book.OwnerId = \_userManager.GetUserId(User);

book.BookLink = Guid.NewGuid();

var selectedCategoryIds = Request.Form["BookCategories"].Select(int.Parse);

var bookCategories = selectedCategoryIds.Select(categoryId => new BookCategory { BookId = book.BookId, CategoryId = categoryId });

book.BookCategories = bookCategories.ToList();

book.CreatedAt = DateTime.Now;

book.Exchangeable = true;

book.Owner = await \_userManager.GetUserAsync(User);

var saveImageResult = \_fileService.SaveImage(book.ImageFile);

if (saveImageResult.Item1 == 1)

{

var oldImage = book.ImageUrl;

book.ImageUrl = saveImageResult.Item2;

var deleteResult = \_fileService.DeleteImage(oldImage);

}

\_context.Books.Add(book);

\_context.SaveChanges();

return RedirectToAction("Index", "Offers");

}

Цей метод відповідає за додавання нової книги до системи, виконуючи наступні дії:

* Встановлення власника книги та посилання на сторінку. book.OwnerId присвоюється ID поточного користувача, отриманий за допомогою менеджера користувачів. book.BookLink отримує унікальне значення GUID, яке необхідне для відображення унікальної сторінки книги;
* Зв'язування з обраними категоріями. Відбувається отримання ID обраних користувачем категорій з форми через запит із представлення. Створюються об'єкти BookCategory для кожної обраної категорії, що зв'язують книгу з цими категоріями, у свою чергу ці об'єкти додаються до колекції book.BookCategories;
* Заповнення додаткових полів: book.CreatedAt встановлюється на поточну дату та час, book.Exchangeable встановлюється на true, що дозволяє обмінювати книгу, book.Owner присвоюється поточний користувач, отриманий асинхронно за допомогою менеджера користувачів;
* Обробка зображення книги. Зображення книги зберігається в репозиторій проекту за допомогою файлового сервісу. Якщо зображення збережено успішно, то адреса нового зображення присвоюється відповідній змінній book.ImageUrl;
* Збереження книги в базі даних. Книга додається до контексту бази даних та зберігається.

Таким чином, метод AddBook() реалізовує повний цикл додавання нової книги до системи, включаючи обробку даних, зображення книги, зв'язок з категоріями та збереження в базу даних.

Варто зазначити, що добавлена книга може бути відображена у цілій низці представлень, які включають в себе домашню сторінку, персональну сторінку користувача, сторінку із загальними пропозиціями від користувачів, унікальну сторінку книги та інші, тому розглядати конкретне представлення не є доцільним. Лише варто зазначити, що взаємодія представлення та даних із контролера відбувалась за допомогою Razor-хелперів та ін’єкції моделі у безпосередньо саме представлення. Таким чином було описано загальну розробку програмного продукту на прикладі розробки модуля «Додавання книги», який забезпечує можливість додавання книги до веб-сервісу авторизованим користувачам програмної системи.

3.3 Інструкція користувача

Роботу із веб-сервісом для обміну книгами доцільно почати із реєстрації, оскільки більшість функцій сервісу не будуть доступні неавторизованим користувачам системи. Для реєстрації користувач повинен натиснути на відповідну кнопку у меню (див. Рисунок 3.3.1).

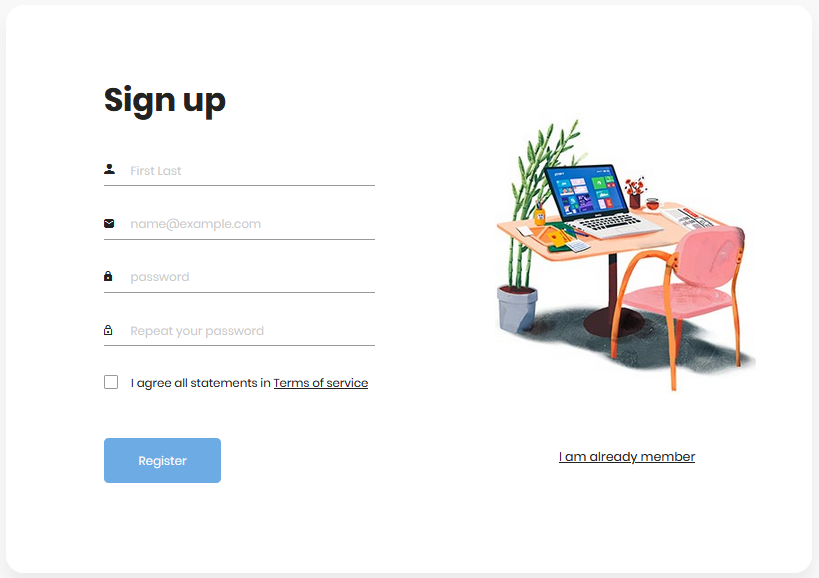


Рисунок 3.3.1 — Сторінка реєстрації користувача

Після успішної реєстрації користувача буде перенаправлено на сторінку авторизації. На цій сторінці користувач може увійти на сайт, для цього йому необхідно буде ввести логін та пароль (див. Рисунок 3.3.2).

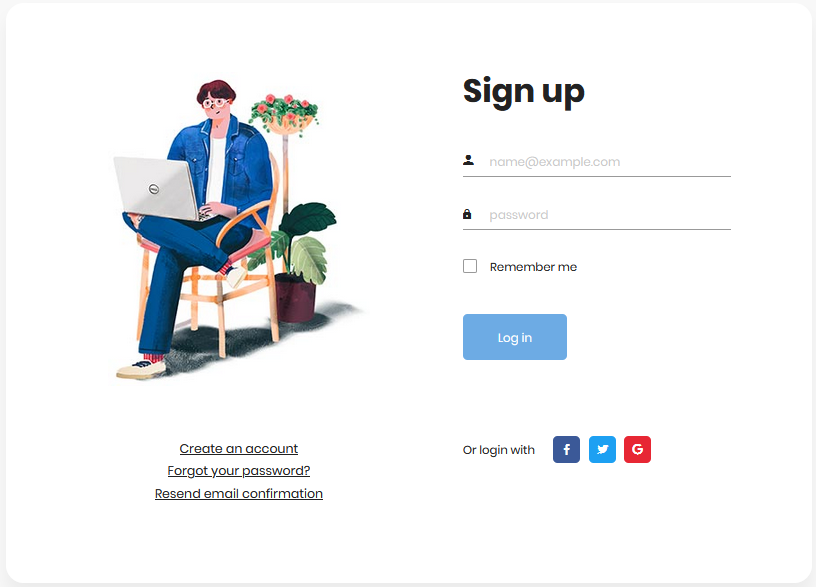


Рисунок 3.3.2 — Сторінка авторизації користувача

Після проходження авторизації користувач може повною мірою користуватися усіма функціями веб-сервісу. Перше, що зможе побачити користувач на головній сторінці — це список нових пропозицій про обмін, які будуть відсортовані за часом створення (див. Рисунок 3.3.3).

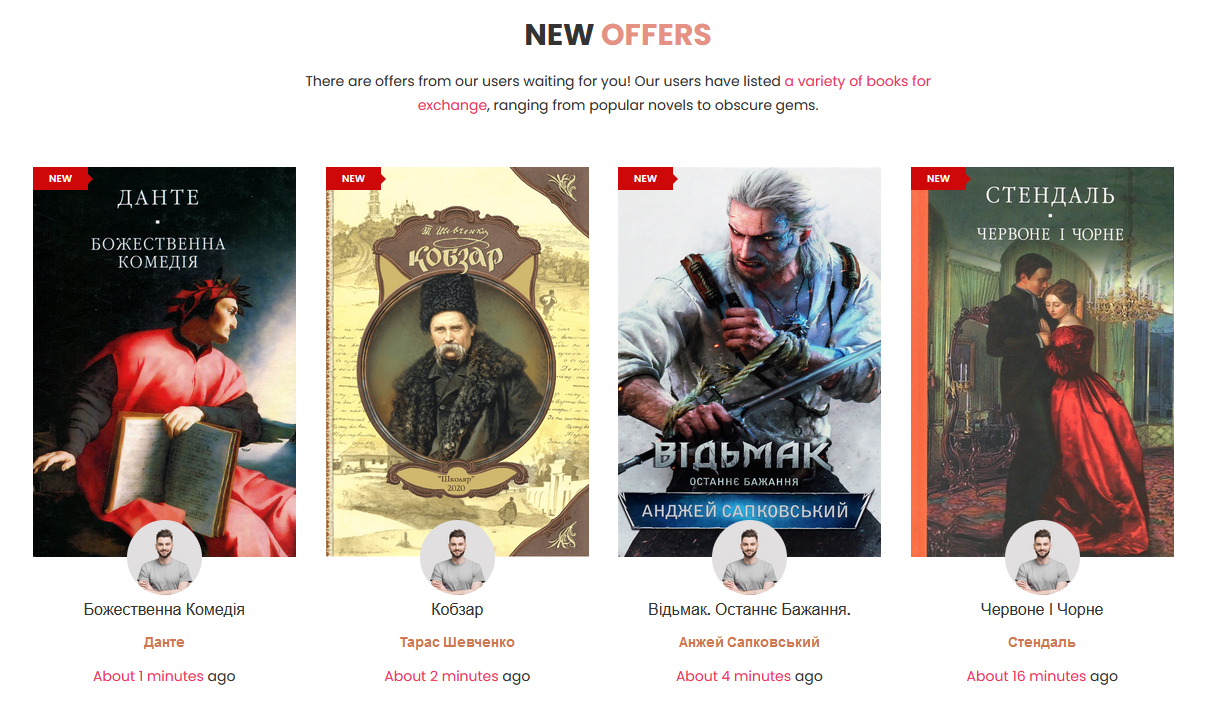


Рисунок 3.3.3 — Нові пропозиції для обміну

Далі користувач для виконання обміну може перейти на індивідуальну сторінку книжки, де він зможе переглянути основну інформацію про книгу та користувача, який додав її у систему (див. Рисунок 3.3.4).

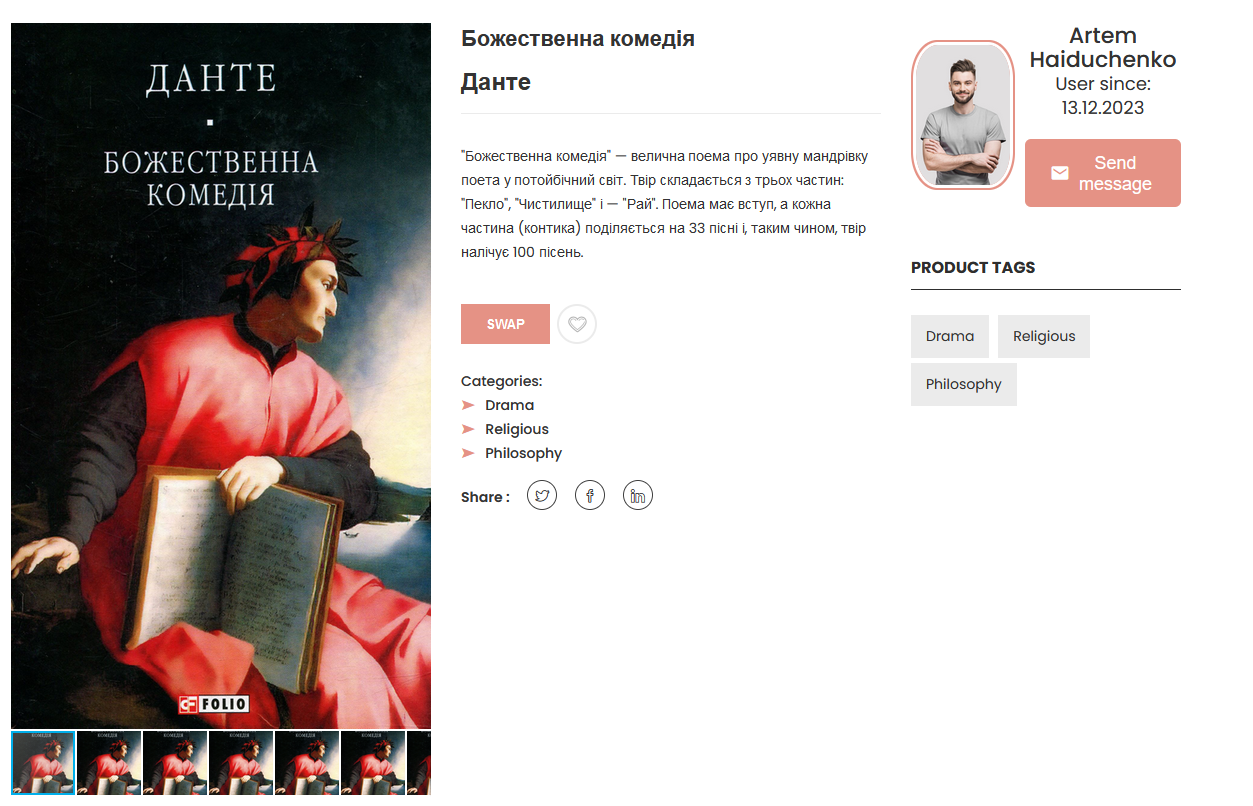


Рисунок 3.3.4 — Унікальна сторінка книги

3.4 Вимоги до програмно-технічного середовища

## Вимоги до програмно-технічного середовища — це кількісні параметри, які визначають здатність пристрою виконувати певні завдання, тобто це сукупність технічних характеристик, які повинен мати пристрій для коректної роботи програмної системи, яка розробляється. Визначимо вимоги для потенційного користувача веб-сервісу та для серверної частини розроблюваної системи (див. Таблиці 7-8).

## Таблиця 7. Системні вимоги для користувача веб-сервісу

|  |  |
| --- | --- |
| Операційна система | Windows, macOS, Linux |
| Браузер | Microsoft Edge, Google Chrome, Mozilla Firefox, Safari та інші. |
| Дисплей | 1024x768 пікселів |
| Процесор | 1,6 ГГц |
| Оперативна пам'ять | 1 ГБ |
| Місце на диску | 100 МБ |

## Таблиця 8. Системні вимоги для серверної частини

|  |  |
| --- | --- |
| Операційна система | Windows, macOS, Linux |
| Інтегроване середовище розробки (IDE) | Visual Studio 2023 |
| Мова програмування | C# |
| СКБД | MS SQL Server |
| .NET SDK | 7.0.12 |
| Додаткові інструменти та бібліотеки: | ASP .NET CORE MVC,Entity Framework Core, Bootstrap 5, jQuery |
| Веб-сервер | IIS |

4. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Методика тестування програмного забезпечення

## Тестування програмного забезпечення — це метод оцінки функціональності програмного забезпечення. Процес перевіряє, чи фактичне програмне забезпечення відповідає очікуваним вимогам, і забезпечує відсутність помилок у програмному забезпеченні. Головним чином він спрямований на вимірювання специфікації, функціональності та продуктивності програмного забезпечення або програми.

## У контексту виявлення та опису програмних модулів системи, їх структури та призначення, які були проведені раніше, доцільним було застосувати методику тестування програмного забезпечення, яке здійснює безпосереднє тестування окремих частин програмної системи, для цього було розглянуто загальну піраміду рівнів тестування (див. Рисунок 4.1.1).

## 

Рисунок 4.4.1 — Піраміда рівнів тестування

## Таким чином було наочно продемонстровано, що тестування модулів та компонентів системи є фундаментальним рівнем у піраміді тестування, тому було вирішено обрати цю методику тестування.

## Модульне тестування [30] — це тип тестування програмного забезпечення, який зосереджується на окремих блоках або компонентах програмної системи. Метою модульного тестування є перевірка того, що кожна одиниця програмного забезпечення працює належним чином і відповідає поставленим вимогам. у контексті декомпозиції програмної системи на окремі модулі вибір цієї методики був обумовлений низкою переваг, серед яких варто виділити:

## можливість ізоляції та тестування окремих частин програми;

## можливість раннього виявлення помилок;

## незалежність результатів тестування окремих модулів програми від результатів тестування інших;

## підвищення базового розуміння алгоритму роботи модулів.

## За умови використання обраних технологій для виконання розробки програмної системи, а саме : С# та .NET, доцільним було обрати і відповідні інструменти та структури для виконання тестування за обраною методикою, які б у відповідали поставленим вимогам та вирішували поставлені завдання. У спільності розробників .NET існують декілька основних та потужних інструментів для тестування програмного забезпечення, серед таких варто виділити xUnit та Moq.

## xUnit.net — це фреймворк unit- або модульного тестування з відкритим вихідним кодом для .NET Framework.

## Moq — популярна імітаційна структура для C# і .NET, яка дозволяє створювати імітації залежностей під час модульного тестування. Це позбавляє від необхідності створювати власноруч складні макети або покладатися на зовнішні бібліотеки для цього.

## Таким чином, було визначено основну методику для тестування програмного забезпечення та визначені основні інструменти для цього.

4.2 Результати тестування програмного забезпечення

## Після визначення методики тестування, інструментів для тестування програмного забезпечення та встановлення потрібних бібліотек було вирішено розпочати процес розробки тестів для окремих модулів програми. Оскільки переважна частина кодової бази програмної системи здебільшого схожа, тому для наочності було продемонстровано тестування функціоналу системи, яка значно відрізняється від інших функцій програми, а саме отримання від користувача побажань у роботі сервісу та відправка йому повідомлення для зворотного зв’язку. Відповідний модуль використовує сервіс EmailSender, який реалізує інтерфейс IEmailSender та явно не записаний у наведеному коді.

## Код методу:

[HttpPost]

public IActionResult Feedback(Mail model)

{

if (model == null || !ModelState.IsValid)

{

return View("Error");

}

try

{

\_emailSender.SendMail(model.FirstName,model.LastName, model.Email, model.Msg);

}

catch (Exception ex)

{

\_logger.LogError($"Error: {ex.Message}");

}

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

## }

## Наведений метод реалізує POST-запит та очікує параметр типу Mail який, представляє модель, що містить інформацію про відгук, наданий користувачем, та виконує такі дії:

## Визначення інформації. Метод отримує ім'я, прізвище, адресу електронної пошти та повідомлення користувача з об'єкта model і передає їх методу SendMail служби \_emailSender;

## Якщо валідація пройде, метод спробує надіслати електронний лист за допомогою служби \_emailSender.

## Спершу було створено тест, який перевіряє, чи функція Feedback контролера MailController правильно обробляє коректну модель даних.

## Код тесту №1:

[Fact]

public void Feedback\_ValidModel\_ReturnsRedirectToActionResult()

{

var emailSenderMock = new Mock<IEmailSender>();

var controller = new MailController(emailSenderMock.Object, null);

var validModel = new Mail

{

FirstName = "John",

LastName = "Doe",

Email = "john.doe@example.com",

Msg = "Test message"

};

var result = controller.Feedback(validModel) as RedirectToActionResult;

Assert.NotNull(result);

Assert.Equal("Index", result.ActionName);

Assert.Equal("Home", result.ControllerName);

emailSenderMock.Verify(x => x.SendMail(validModel.FirstName, validModel.LastName, validModel.Email, validModel.Msg), Times.Once);

}

Загалом сценарій роботи тесту виглядає наступним чином:

* Створюється тестовий екземпляр класу MailController з використанням фіктивного об'єкта класу IEmailSender та коректна модель даних типу Mail з заповненими полями FirstName, LastName, Email, і Msg.
* Модель даних передається в метод Feedback контролера.
* Результат роботи методу перевіряється на виключення;
* Перевіряється, чи був викликаний метод SendMail об'єкта IEmailSender один раз з відповідними аргументами.

Далі було розроблено тест, який перевіряє поведінку методу, коли форма відгуків надсилається з недійсною моделлю.

## Код тесту №2:

[Fact]

public void Feedback\_InvalidModel\_ReturnsErrorView()

{

var emailSenderMock = new Mock<IEmailSender>();

var controller = new MailController(emailSenderMock.Object, null);

var invalidModel = new Mail();

controller.ModelState.AddModelError("Email", "Email is required");

var result = controller.Feedback(invalidModel) as ViewResult;

Assert.NotNull(result);

Assert.Equal("Error", result.ViewName);

}

Загалом сценарій роботи цього тесту виглядає наступним чином:

* Створює об’єкти, необхідні для функціонування методу, а саме: фіктивний об'єкт IEmailSender, новий екземпляр MailController з фіктивним відправником електронної пошти, недійсну модель Mail без заповненого поля "Email";
* Викликає функцію Feedback з недійсною моделлю;
* Перевіряє, чи результат виклику є об'єктом ViewResult та чи ім'я повернутого представлення дорівнює "Error".

Цей тест перевіряє, що контролер MailController правильно обробляє ситуацію, коли користувач надсилає null замість валідної моделі Mail через функцію Feedback.

## Код тесту №3:

[Fact]

public void Feedback\_NullModel\_ReturnsErrorView()

{

var emailSenderMock = new Mock<IEmailSender>();

var loggerMock = new Mock<ILogger<MailController>>();

var mailController = new MailController(emailSenderMock.Object, loggerMock.Object);

var result = mailController.Feedback(null) as ViewResult;

Assert.NotNull(result);

Assert.Equal("Error", result.ViewName);

## }

Загалом сценарій роботи тесту №3 виглядає наступним чином:

## Створює фіктивні об'єкти IEmailSender і ILogger<MailController> для імітації відправки електронної пошти та запису логів.

## Створює новий екземпляр MailController з фіктивними об'єктами.

## Викликає функцію Feedback з null замість моделі.

## Перевіряє, чи результат виклику є об'єктом ViewResult.

## Перевіряє, чи ім'я повернутого представлення дорівнює "Error".

## Далі було проведено мануальне тестування обраного функціоналу надсилання зворотного зв’язку. Для цього було створено фіктивний відгук про роботу сервісу із фіктивною поштою, далі на цю пошту було надіслане відповідне повідомлення (див. Рисунок 4.2.1).

## 

## Рисунок 4.2.1 — Надіслане повідомлення

ВИСНОВКИ

Основною метою роботи була реалізація зручного сервісу для обміну книгами, який забезпечить ефективний механізм обміну літературою. Таким чином, у роботі був виконаний повноцінний процес розробки програмного забезпечення для виконання поставлених завдань

Отже, першочергово було проведене дослідження предметної області, в ході якого було розроблена IDEF0-діаграма, яка забезпечила структурований підхід до аналізу бізнес-процесів, дозволяючи визначити основні функції, взаємодії у ході знайомства із предметною областю та аналізу уже існуючих рішень, що дало загальне уявлення про функції та завдання майбутньої програмної системи.

Після формування функціональних, нефункціональних вимог та вимог до системи в цілому, було вирішено обрати загальну архітектуру проекту для визначення загальної структури, взаємодії та поведінки системи. У ході дослідження переваг та недоліків усіх популярних видів архітектури ПЗ, було обрано «клієнт-серверну» архітектуру, як ту, яка найкраще підходить для виявлених вимог. Далі було проведено проектування середовища для зберігання даних або бази даних, за результатами якої були виявлені основні сутності та атрибути, створені таблиці, які за результатами проектування були приведені до 3НФ. Результатом проектування бази даних стала повноцінна ER-діаграма, яка відобразила основні зв’язки та відношення між виявленими сутностями. Перед безпосередньою розробкою системи було обрано основні інструменти розробки, які були застосовані у повній мірі під час розроби програмного забезпечення.

Після цього було виконано процес декомпозиції функціональності системи на окремі програмні модулі для полегшення процесу розробки, описано загальну структуру та призначення модулів. За результатами проведеного аналізу була проведена повноцінна розробка відповідних модулів за допомогою обраних інструментів розробки ПЗ.

Завершальним пунктом реалізації системи стало безпосереднє тестування програмного забезпечення. Для початку було обрано загальну методику тестування, далі за обраною методикою було проведене тестування окремих модулів, за результатами якого було підтверджено справність розроблених модулів програми.

Таким чином, за результатами проведеної роботи були виконані усі поставлені перед системою завдання та задовільнені усі вимоги, які були необхідні для створення повноцінного програмного забезпечення.

**ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ**

* 1. X [Електронний ресурс] — Режим доступу до ресурсу: <https://x.com/> (дата звернення 30.10.24).
  2. Client-Server Model [Електронний ресурс] — Режим доступу до ресурсу: <https://www.geeksforgeeks.org/client-server-model/> (дата звернення 30.10.24).
  3. Pattern: Monolithic Architecture [Електронний ресурс] — Режим доступу до ресурсу: <https://microservices.io/patterns/monolithic.html> (дата звернення 30.10.24).
  4. What are microservices? [Електронний ресурс] — Режим доступу до ресурсу: <https://microservices.io/> (дата звернення 30.10.24).
  5. Introduction of ER Model [Електронний ресурс] — Режим доступу до ресурсу: <https://www.geeksforgeeks.org/introduction-of-er-model/> (дата звернення 30.10.24).
  6. IDEF1X. Data Modeling Method [Електронний ресурс] — Режим доступу до ресурсу: <https://www.idef.com/idef1x-data-modeling-method/> (дата звернення 30.10.24).
  7. Unified Modeling Language (UML) | An Introduction [Електронний ресурс] — Режим доступу до ресурсу: <https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/> (дата звернення 30.10.24).
  8. What is Use Case Diagram? [Електронний ресурс] — Режим доступу до ресурсу: <https://www.geeksforgeeks.org/use-case-diagram/> (дата звернення 30.10.24).
  9. Sequence diagrams [Електронний ресурс] — Режим доступу до ресурсу: <https://www.ibm.com/docs/en/rsm/7.5.0?topic=uml-sequence-diagrams> (дата звернення 30.10.24).
  10. Unified Modeling Language (UML) | Activity Diagrams [Електронний ресурс] — Режим доступу до ресурсу: <https://www.geeksforgeeks.org/unified-modeling-language-uml-activity-diagrams/> (дата звернення 30.10.24).
  11. The Go Programming Language [Електронний ресурс] — Режим доступу до ресурсу: [https://go.dev/](https://go.dev/%20) (дата звернення 30.10.24).
  12. React [Електронний ресурс] — Режим доступу до ресурсу: [https://react.dev/](https://react.dev/%20) (дата звернення 30.10.24).
  13. MySQL [Електронний ресурс] — Режим доступу до ресурсу: <https://www.mysql.com/> (дата звернення 30.10.24).