ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмних засобів для підтримки комунікації з клієнтами»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Єлизавета ЖЕЖЕНКО
(підпис)

Виконала: здобувачка вищої освіти групи ПД-42

Єлизавета ЖЕЖЕНКО

Керівник: Тимур ДОВЖЕНКО

к.т.н., доцент

Рецензент:

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти Бакалавр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Жеженко Єлизаветі Олександрівні

1. Тема кваліфікаційної роботи: «Розробка програмних засобів для підтримки комунікації з клієнтами»
керівник кваліфікаційної роботи к.т.н., доц. кафедри ІПЗ Тимур ДОВЖЕНКО, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: технічна документація мов програмування та фреймворків, приклади реалізації веб-застосунків із функціями комунікації.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Визначення актуальності цифрової комунікації з клієнтами у веб-середовищі.
2. Аналіз сучасних платформ та технологій для реалізації систем взаємодії з користувачами.
3. Розроблення веб-застосунку для підтримки комунікації: архітектура, функціональність, реалізація інтерфейсу.
4. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма діяльності.
5. Процес інтеграції.
6. Екранні форми веб-інтерфейсу.
7. Апробація результатів дослідження.

6. Дата видачі завдання«25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд існуючих мов програмування, фреймворків та архітектурних підходів для розробки веб-додатків	15.03-17.04.2025	
4	Проєктування ключових модулів веб-додатку (реєстрація, чат, пости)	18.04-19.04.2025	
5	Програмна реалізація застосунку	20.04-26.04.2025	
6	Тестування застосунку	27.04-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувачка вищої освіти _____ Єлизавета ЖЕЖЕНКО

(підпис)

Керівник

кваліфікаційної роботи _____ Тимур ДОВЖЕНКО

(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: бакалаврської роботи: 59 стор., 5 табл., 22 рис., 30 джерел.

Об'єкт дослідження – процес взаємодії між організацією та клієнтами за допомогою цифрових каналів комунікації.

Предмет дослідження – веб-застосунок для підтримки комунікації з клієнтами, що забезпечує централізовану обробку звернень, інтеграцію з Telegram, WhatsApp та інструментами для аналітики взаємодій.

Мета роботи – оптимізувати обробку клієнтських звернень через цифрові інструменти.

Короткий зміст роботи: У роботі проаналізовано предметну галузь цифрової комунікації та сучасні технології, що використовуються для розробки систем взаємодії з клієнтами. Проведено огляд рішень, які поєднують месенджери, CRM-системи та веб-інтерфейси, для автоматизації обробки звернень. Розроблено веб-застосунок із функціоналом реєстрації, створення публікацій, коментування, обміну повідомленнями та збереження історії взаємодії.

Система має адаптивний інтерфейс, реалізована архітектура MVC, передбачена інтеграція з Telegram-ботом для швидкої комунікації. Також реалізовано адміністративну панель, REST API, авторизацію та механізми сповіщень. Веб-застосунок побудовано з використанням Spring Boot (Java) для серверної частини та Bootstrap/HTML/JS — для фронтенду. Збереження даних реалізовано на основі PostgreSQL. Проведено тестування застосунку та порівняння з аналогами.

Область застосування — малий бізнес, освітні платформи, служби підтримки, де необхідна швидка та якісна цифрова взаємодія з користувачами.

КЛЮЧОВІ СЛОВА: ВЕБ-ДОДАТОК, SPRING BOOT, ІНТЕГРАЦІЯ МЕСЕНДЖЕРІВ, POSTGRESQL, РЕЄСТРАЦІЯ КОРИСТУВАЧА, ВЗАЄМОДІЯ, КОМУНІКАЦІЯ З КЛІЄНТАМИ.

ЗМІСТ

ВСТУП

У сучасному світі цифровізації особливої актуальності набувають програмні засоби, що забезпечують ефективну взаємодію між користувачами, зокрема між бізнесом і клієнтами. У сфері малого бізнесу, де ресурси часто обмежені, важливо мати гнучкий і доступний інструмент комунікації, який дозволяє швидко реагувати на звернення, обробляти повідомлення та підтримувати постійний зв'язок із цільовою аудиторією.

Особливого значення набувають веб-додатки для обміну повідомленнями, які об’єднують зручний інтерфейс, адаптивність під різні пристрої та можливість масштабування під конкретні потреби користувача. Такі інструменти дають змогу автоматизувати частину взаємодії, покращити якість обслуговування та забезпечити персоналізований підхід до кожного звернення.

Розробка власного веб-додатку для спілкування відкриває нові можливості для побудови ефективної системи взаємодії, орієнтованої на потреби кінцевого користувача та адаптованої до умов малого бізнесу.

Об’єкт дослідження – процес взаємодії між організацією та клієнтами за допомогою цифрових каналів комунікації.

Предмет дослідження – веб-застосунок для підтримки комунікації з клієнтами, що забезпечує централізовану обробку звернень, інтеграцію з месенджерами та інструментами для аналітики взаємодій.

Мета роботи – оптимізувати обробку клієнтських звернень через цифрові інструменти.

Наукові завдання:

– проаналізувати сучасні підходи до цифрової взаємодії з користувачами в малому бізнесі;

– дослідити технології та фреймворки, придатні для створення веб-систем комунікації;

– спроєктувати архітектуру веб-застосунку для підтримки повідомлень, публікацій та коментарів;

– реалізувати функціональні модулі: реєстрацію користувача, чат, стрічку повідомлень, обробку форм введення;

– забезпечити збереження історії взаємодії у базі даних та налаштувати обробку запитів;

– протестувати систему на функціональність, стабільність та зручність для кінцевого користувача.

Методи дослідження:

1. Аналіз літературних джерел і огляд існуючих рішень у сфері веб-комунікацій.

2. Порівняння мов програмування та вибір технологій розробки.

3. Проєктування архітектури системи з урахуванням вимог масштабованості та зручності.

4. Реалізація веб-застосунку та його тестування в умовах модельного сценарію використання.

Практичне значення одержаних результатів: мають практичну цінність для компаній і розробників, які прагнуть створити власний інструмент для взаємодії з клієнтами без залучення сторонніх месенджерів або CRM. Запропонований веб-додаток дозволяє реалізувати базову систему обміну повідомленнями, формувати інформаційне середовище навколо продукту чи послуги, а також покращити якість цифрової комунікації між користувачами. Розроблена система може бути застосована в освітніх, комерційних або сервісних проєктах, де важлива швидка, зручна та інтерактивна взаємодія з аудиторією.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Сучасні підходи до цифрової комунікації з клієнтами

Компанії спілкуються з клієнтами переважно через месенджери, чати, соцмережі та мобільні додатки. Це швидко, зручно й доступно 24/7. Такі канали дозволяють автоматично приймати звернення, відповідати на типові запитання, надсилати сповіщення й збирати статистику. Це знижує навантаження на персонал і покращує досвід клієнта [1].

Цей тип взаємодії набув особливого значення в умовах цифрової трансформації бізнесу, коли ефективна комунікація з клієнтами є критично важливою для забезпечення конкурентоспроможності, побудови лояльної аудиторії та оперативного реагування на запити споживачів. Цифрова комунікація охоплює широкий спектр інструментів – від електронної пошти та веб-чату до інтегрованих CRM-систем і месенджерів, що використовують API для зв’язку з користувачем у реальному часі.

У сучасних веб-застосунках цифрова комунікація реалізується через комплекс функціональних компонентів, що забезпечують обмін повідомленнями, сповіщеннями та зворотним зв’язком у реальному часі або з використанням асинхронних механізмів. Найпоширенішими інструментами є вбудовані чати, форми зворотного зв’язку, чат-боти, push-сповіщення та системи коментарів. Ці елементи інтегруються у веб-інтерфейс з використанням технологій, таких як WebSocket, AJAX, REST API або GraphQL, що дозволяє забезпечити динамічну взаємодію між клієнтською частиною та сервером без необхідності повного перезавантаження сторінки [2].

Крім того, веб-додатки часто включають механізми автентифікації користувачів, збереження історії повідомлень у базах даних (наприклад, MySQL,

MongoDB) та інтеграцію з зовнішніми сервісами – зокрема, месенджерами Telegram або Viber через відповідні API. Це дозволяє розширити канали комунікації, автоматизувати частину запитів і забезпечити персоналізовану взаємодію з клієнтом.

Швидка та інтерактивна взаємодія є умовою ефективної цифрової комунікації, оскільки вона безпосередньо впливає на якість обслуговування клієнтів, швидкість прийняття рішень та рівень задоволеності користувачів. У сучасному динамічному середовищі очікування клієнтів суттєво зросли: користувачі прагнуть отримати відповіді на свої запити негайно, без затримок і складних процедур [3]. Затримка у відповіді або відсутність інтерактивності може призвести до втрати довіри, зниження лояльності та відтоку клієнтів.

Інтерактивна взаємодія передбачає реагування в реальному часі, можливість двостороннього обміну інформацією, індивідуальний підхід до користувача, а також контекстну адаптацію вмісту, що подається. Наприклад, чат-боти можуть не лише відповідати на часті запитання, а й перенаправляти користувача до відповідального менеджера залежно від теми запиту. Таким чином, швидкість і динамічність взаємодії формують позитивний користувацький досвід (UX) та сприяють зміцненню взаємин між клієнтом і компанією.

З метою узагальнення ключових характеристик різних каналів цифрової комунікації доцільно проаналізувати їх за основними критеріями, такими як швидкість відповіді, рівень персоналізації, можливості автоматизації, інтеграція з іншими системами та зручність для кінцевого користувача [3]. У табл. 1.1 наведено порівняльну характеристику електронної пошти, месенджерів, чат-ботів та CRM-систем.

Таблиця 1.1

Інфографіка Типів Цифрової Комунікації

Канал	**Швидкість відповіді**	**Персоналізація**	**Автоматизація**	**Інтеграція з системами**	**Зручність для користувача**
Email	Низька	Обмежена	Мала	Середня	Середня
Месенджери	Висока	Середня	Середня	Середня	Висока
Чат-боти	Миттєва	Висока	Висока	Висока	Висока
CRM-системи	Залежить від налаштувань	Висока	Висока	Дуже висока	Залежить від реалізації

1.2 Роль CRM-систем і месенджерів у підтримці взаємодії

CRM-система — це інструмент, який допомагає бізнесу не втрачати клієнтів. Вона об’єднує всі звернення з месенджерів, email, сайтів в одну панель, зберігає історію комунікації та дозволяє швидко реагувати на запити. Через неї можна вести облік заявок, автоматизувати розсилки, аналізувати поведінку клієнтів і покращувати сервіс.

Основним призначенням CRM-систем є автоматизація взаємодії з клієнтом на всіх етапах його життєвого циклу – від першого звернення до післяпродажного супроводу. Завдяки CRM компанії можуть:

– зберігати й систематизувати інформацію про клієнтів;

– фіксувати історію комунікацій (дзвінки, повідомлення, email);

– налаштовувати автоматичні відповіді та тригери;

– сегментувати аудиторію та створювати персоналізовані пропозиції;

– інтегруватися з іншими цифровими сервісами.

Сучасні CRM-системи реалізовані як веб-орієнтовані програмні продукти, що можуть бути встановлені локально або доступні через хмарні сервіси (SaaS). Найпопулярніші з них – Bitrix24, Zoho CRM, HubSpot, Salesforce – надають

можливості масштабування, аналітики та звітності, що дозволяє адаптувати систему під конкретні бізнес-завдання [4].

Однією з ключових переваг CRM є інтеграція з месенджерами, яка дозволяє централізовано обробляти звернення з різних каналів (Telegram, Viber, Facebook Messenger тощо) в одному інтерфейсі. Це спрощує управління запитами, зменшує час обробки і дозволяє контролювати якість сервісу.

Загалом, CRM-системи виступають основою цифрової трансформації клієнтської взаємодії, сприяючи ефективнішій комунікації, підвищенню продуктивності персоналу та формуванню довготривалих відносин з клієнтами.

Інтеграція CRM-систем із популярними месенджерами, такими як Telegram, WhatsApp, відкриває нові можливості для побудови ефективного та персоналізованого клієнтського сервісу. У поєднанні ці інструменти формують єдину цифрову платформу, яка забезпечує швидку, зручну та керовану взаємодію з клієнтами у режимі реального часу [5].

CRM-система зберігає повну історію спілкування, облікові записи клієнтів, записи звернень, налаштування комунікаційних сценаріїв, автоматичні відповіді, тригери й шаблони повідомлень. Підключення месенджерів до CRM дає змогу:

– автоматично фіксувати кожен контакт із клієнтом (заявки, запити, повідомлення);

– використовувати чат-ботів для первинної обробки звернень (відповіді на часті запитання, збір даних, маршрутизація);

– оперативно відповідати на запити в єдиному інтерфейсі незалежно від того, з якого месенджера звернувся клієнт;

– персоналізувати повідомлення, враховуючи попередні дії користувача (наприклад, залишені заявки або історію покупок);

– автоматизувати маркетинг і повторну взаємодію через відкладені розсилки, нагадування, спеціальні пропозиції.

Завдяки високому рівню зручності для користувача месенджери часто стають основним каналом комунікації з бізнесом, особливо в сегменті малого та

середнього підприємництва. Водночас CRM дозволяє контролювати якість обслуговування, вести аналітику, призначати відповідальних співробітників та оптимізувати процеси.

Поєднання CRM і месенджерів забезпечує:

– зниження часу обробки запитів;

– зменшення навантаження на персонал завдяки автоматизації;

– підвищення лояльності клієнтів через швидку підтримку;

CRM-системи в комбінації з месенджерами виступають не лише технічним засобом комунікації, а й стратегічним інструментом підвищення якості обслуговування клієнтів та конкурентоспроможності компанії [5].

Для кращого розуміння прикладного значення CRM-систем і месенджерів у бізнесі доцільно представити типові сценарії їх використання (табл. 1.2). Такий підхід дозволяє систематизувати галузі застосування цифрових інструментів, визначити їхню функціональну роль та продемонструвати переваги поєднання систем управління клієнтськими відносинами з сучасними каналами комунікації. У таблиці наведено порівняння найбільш поширених бізнес-сценаріїв із фокусом на розподіл завдань між CRM-системами та месенджерами.

Таблиця 1.2

Приклади використання CRM-систем і месенджерів у різних сферах бізнесу

Сфера	**Приклади використання**	**Роль CRM**	**Роль месенджера**
Інтернет-магазини	Статус замовлень, розсилки, рекомендації	Збереження історії замовлень, аналітика	Повідомлення про доставку, чат-боти
Послуги (салони, клініки)	Онлайн-запис, нагадування, знижки	Клієнтська база, календар візитів	Запис через чат, нагадування в месенджері
Освіта (курси, школи)	Інформування про заняття, консультації	Індивідуальні картки студентів	Групові чати, повідомлення про зміни
B2B-компанії	Комунікація з партнерами	Історія перемовин	Обговорення умов, швидкий зв'язок

1.3 Огляд існуючих рішень для автоматизації комунікацій

Zoho CRM – це комплексна система управління відносинами з клієнтами, яка підтримує автоматизацію маркетингу, продажів і сервісу. Однією з ключових її функцій є можливість налаштування робочих процесів (workflows), інтеграція з популярними месенджерами та автоматичні сповіщення, що дозволяє оперативно реагувати на дії клієнта без залучення людини. Система також підтримує AI-асистента Zia, що аналізує поведінку користувачів і підказує оптимальні рішення [6].

Chatfuel – це платформа для створення чат-ботів без програмування, яка найчастіше використовується в месенджерах Telegram та Facebook Messenger. Вона дозволяє будувати логічні сценарії взаємодії, автоматизувати відповіді, збирати дані про користувачів і направляти звернення у CRM. Chatfuel використовується як фронт-енд для первинної обробки запитів, який знімає навантаження з операторів і підвищує швидкість обробки звернень.

ManyChat – сервіс, орієнтований на чат-боти для маркетингу, що підтримує мультиканальну комунікацію (Messenger, Telegram, WhatsApp, SMS, email). Особливістю є простий редактор сценаріїв та інтеграція з платформами e-commerce, що робить його популярним серед онлайн-магазинів.

SendPulse – багатофункціональна платформа, яка включає email-маркетинг, SMS-розсилки, web push-сповіщення та чат-боти для Telegram/Viber. Завдяки інтеграції з CRM-модулем, платформа дозволяє будувати повноцінні омніканальні стратегії взаємодії з клієнтами, що особливо цінно для середнього та малого бізнесу [7].

Кожен із наведених інструментів має свої переваги й обмеження, які варто враховувати при виборі платформи для малого бізнесу. У таблиці 1.3 наведено порівняльну характеристику популярних рішень автоматизації комунікацій, зокрема Zoho CRM, Chatfuel, ManyChat та SendPulse.

Таблиця 1.3

Порівняння CRM-рішень для автоматизації комунікацій

Інструмент	Переваги	Недоліки	Обмеження для малого бізнесу
Zoho CRM	Глибока аналітика, AI-асистент, розгалужені сценарії автоматизації	Висока складність для новачків, частина функцій доступна лише у платній версії	Складність інтеграції без технічних навичок, обмежений безкоштовний тариф
Chatfuel	Швидке створення ботів без коду, інтеграція з Telegram, Facebook	Обмежена логіка без коду, складно масштабувати	Безкоштовна версія з брендингом, обмежені функції аналітики
ManyChat	Зручний візуальний редактор, мультиканальна підтримка (Telegram, WhatsApp, email)	Обмежена аналітика, платні розширення	Додаткові функції тільки на Pro-тарифі, обмеження на кількість підписників
SendPulse	Омніканальність (email, чат-боти, push), простота використання	Менш розвинуті сценарії автоматизації в чат-ботах	Обмежений функціонал у безкоштовному плані, необхідність комбінування каналів

1.4 Типи клієнтських комунікацій і вимоги до ПЗ

Сьогодні компанії активно спілкуються з клієнтами через різні цифрові канали — месенджери, чат-боти, соцмережі, email тощо. Це вже не просто зручність, а необхідність для бізнесу. Люди хочуть швидко отримувати відповіді, дізнаватися про новини, акції або статус своїх замовлень. Щоб все це працювало злагоджено, потрібні спеціальні системи, які можуть зберігати всю інформацію про клієнта, об’єднувати різні канали зв’язку та автоматизувати частину процесів. Тому важливо правильно організувати цифрову комунікацію, щоб вона була ефективною як для бізнесу, так і для клієнта.

Онлайн-чат – це один із найоперативніших каналів взаємодії, який забезпечує обмін повідомленнями в реальному часі. Його використання дозволяє миттєво реагувати на звернення користувачів, проводити консультації, вирішувати технічні

чи адміністративні питання. В сучасних умовах онлайн-чати інтегруються з CRM-системами та підтримують автоматизацію за допомогою чат-ботів.

Форма зворотного зв’язку, сценарій, який передбачає заповнення користувачем форми для надсилання запитань, відгуків або скарг. Основною перевагою цього формату є структурованість даних, однак затримка у відповіді вимагає від системи реалізації повідомлень про прийняття звернення та призначення відповідального співробітника [8].

Розсилки (масові та тригерні), цей сценарій включає періодичне надсилання повідомлень із метою інформування, маркетингу чи технічного супроводу.

Коментарі та публічні повідомлення – функціональність, що дозволяє користувачам залишати відкриті повідомлення або реагувати на вміст інших користувачів. Такий формат є поширеним у сервісах з елементами соціальної взаємодії або підтримки спільнот. Від ПЗ вимагається реалізація механізмів модерації, фільтрації та сповіщень [9].

Автоматизовані діалоги (бот-сценарії) – цифрові агенти (чат-боти) дозволяють реалізувати попередньо налаштовані діалоги, що відповідають на типові запити, спрямовують користувача відповідно до сценарію або передають його оператору. Такі рішення потребують побудови гнучких сценаріїв, інтеграції з базами даних, CRM та сервісами авторизації.

Усі вищезгадані сценарії пред’являють до програмного забезпечення такі ключові вимоги:

– інтерактивність та швидкодія у відповідях;

– масштабованість із можливістю одночасного обслуговування великої кількості запитів;

– забезпечення конфіденційності та цілісності даних;

– інтеграція з зовнішніми платформами (месенджери, email, CRM);

– адаптивність інтерфейсу для різних типів пристроїв (мобільних, десктопних);

– аналітична підтримка для моніторингу ефективності взаємодії.

Таблиця 1.4

Порівняння типів комунікацій за ключовими характеристиками

Тип взаємодії	**Оперативність**	**Автоматизація**	**Інтеграція з CRM**	**Придатність для малого бізнесу**
Онлайн-чат	Висока	Середня/висока	Так	Так
Зворотний зв’язок	Середня	Середня	Так	Так
Розсилки	Середня	Висока	Так	Так
Коментарі/обговорення	Середня	Низька	Частково	Обмежено
Чат-боти	Висока	Висока	Так	Так

Ефективна реалізація цифрової комунікації передбачає дотримання низки функціональних та нефункціональних вимог до програмного забезпечення. Відповідність цим вимогам забезпечує стабільну, масштабовану та зручну у використанні систему, орієнтовану на потреби кінцевого користувача [10].

Функціональні вимоги визначають основний перелік дій, які система повинна забезпечувати для виконання своїх завдань:

– створення облікових записів, вхід через логін/пароль;

– збереження, редагування даних клієнтів;

– використання двостороннього чату та історії листування;

– інтеграція з месенджерами та email, автоматичне створення тікетів;

– створення заявок, зміна статусу, призначення на операторів;

– розробка статистики звернень, час відповіді, оцінка якості обслуговування;

Нефункціональні вимоги формують якісні характеристики системи та впливають на її надійність і користувацький досвід:

– швидкість завантаження не більше 1 секунди, автоматичне резервне копіювання;

– шифрування паролів;

– адаптація під мобільні пристрої [12].

2 ІНСТРУМЕНТИ І ТЕХНОЛОГІЇ ДЛЯ РОЗРОБКИ СИСТЕМИ

2.1 Огляд мов програмування для розробки комунікаційного ПЗ

У процесі створення програмних засобів для цифрової комунікації з користувачами важливе значення має вибір мови програмування. Мова визначає архітектурні рішення, продуктивність, сумісність із веб-технологіями, а також легкість розробки та підтримки системи. У цьому підрозділі розглянуто найбільш поширені мови, що використовуються для створення веб-комунікаційних платформ: JavaScript, Python, PHP, C++ та C# (див. рисунок 2.1) [13].

Рис. 2.1 Перелік мови програмування для розробки комунікаційного ПЗ

JavaScript є мовою програмування, яка найбільш активно застосовується у фронтенд-розробці веб-застосунків. Вона підтримується всіма сучасними браузерами і є основою інтерактивності веб-інтерфейсів.

Переваги:

– нативна підтримка у браузерах, що робить її незамінною для створення клієнтської частини;

– можливість реалізації повноцінного стеку (frontend + backend) завдяки середовищу Node.js;

– наявність великої кількості бібліотек і фреймворків (React, Vue.js, Express);

– асинхронна обробка запитів (через Promise, async/await), що критично важливо для обробки повідомлень у реальному часі [14].

Недоліки:

– слабка типізація, що ускладнює масштабування великих проєктів;

– складність відлагодження через асинхронність і залежність від браузерного середовища.

Python – мова високого рівня з читабельним синтаксисом, що широко використовується в бекенд-розробці, машинному навчанні та автоматизації.

Переваги:

– висока швидкість розробки завдяки простому синтаксису;

– потужні веб-фреймворки (Django, Flask) з підтримкою REST API та WebSocket;

– можливість реалізації інтеграційних сценаріїв із месенджерами та CRM через бібліотеки;

– активна спільнота та багата технічна документація.

Недоліки:

– нижча продуктивність у порівнянні з компільованими мовами (C++, C#);

– менш придатний для фронтенд-розробки, тому вимагає комбінування з JavaScript.

PHP є класичною мовою для створення серверної логіки веб-застосунків, особливо для CMS і простих сайтів.

Переваги:

– висока сумісність із веб-хостингами;

– велика кількість готових рішень і CMS (WordPress, Joomla);

– фреймворки типу Laravel дозволяють швидко створювати API для обміну повідомленнями.

Недоліки:

– морально застарілий підхід до архітектури в класичному PHP;

– обмежена масштабованість та складнощі з асинхронною обробкою запитів у реальному часі.

C++ – мова системного рівня, яка рідко використовується у веб-розробці, але може бути актуальною для низькорівневих компонентів системи або високонавантажених серверів.

Переваги:

– висока продуктивність і контроль над ресурсами;

– можливість створення оптимізованих серверних рішень для обробки великого обсягу повідомлень.

Недоліки:

– висока складність розробки та підтримки;

– відсутність нативної підтримки веб-стеків, потреба в зовнішніх фреймворках або модулях інтеграції.

C# – мова програмування, що активно використовується для створення веб-додатків у середовищі Microsoft (.NET Core / ASP.NET) [15].

Переваги:

– потужна інтеграція з корпоративними сервісами та базами даних;

– підтримка Web API, SignalR для обміну даними в реальному часі;

– високий рівень безпеки, типізація та продуктивність.

Недоліки:

– залежність від екосистеми Microsoft;

– складність розгортання в умовах обмежених ресурсів (особливо для малого бізнесу).

Мови програмування, які застосовуються у сфері веб-комунікацій, мають різні сильні сторони. JavaScript є незамінним на клієнтському боці та зручним для

реалізації Node.js-серверів. Python забезпечує швидку розробку та інтеграцію з зовнішніми сервісами. PHP залишається популярним для невеликих рішень, а C# є оптимальним для корпоративних систем. C++ варто застосовувати лише у разі необхідності максимальної продуктивності. Вибір мови має здійснюватися з урахуванням вимог до швидкодії, масштабованості, простоти підтримки та бюджету проєкту.

2.2 Вибір фреймворків, бібліотек і середовища розробки

Для реалізації програмного засобу підтримки комунікації з клієнтами було обрано низку сучасних фреймворків, бібліотек і середовищ розробки, що забезпечують ефективну організацію процесу створення веб-додатку, спрощують інтеграцію з зовнішніми сервісами та гарантують масштабованість і безпеку системи [16].

Основними елементами вибраного технологічного стеку є:

– Node.js – середовище виконання JavaScript на стороні сервера, яке дозволяє реалізувати високопродуктивну серверну частину з підтримкою асинхронної обробки запитів. Використання Node.js спрощує створення REST API та WebSocket-з’єднань, необхідних для обміну повідомленнями в реальному часі.

– Express – мінімалістичний веб-фреймворк для Node.js, який дозволяє швидко розгорнути серверну логіку, налаштувати маршрутизацію, обробку запитів і підключення middleware.

– React – JavaScript-бібліотека для створення користувацьких інтерфейсів, що забезпечує побудову динамічного, адаптивного та зручного для користувача фронтенду. React дозволяє організувати компонентну структуру програми, що сприяє легкому масштабуванню та повторному використанню коду.

– Bootstrap – CSS-фреймворк, який використовується для швидкого створення адаптивного дизайну, забезпечуючи коректне відображення інтерфейсу на різних типах пристроїв (десктопи, планшети, мобільні телефони).

У разі реалізації серверної частини на базі екосистеми Java доцільним є використання таких компонентів:

– Spring Boot – фреймворк для швидкого створення автономних, готових до використання додатків на Java, що спрощує конфігурацію та розгортання сервісів.

– Spring Data JPA – модуль Spring, який спрощує роботу з базами даних, дозволяє реалізовувати CRUD-операції без необхідності написання SQL-запитів вручну.

– Spring Web MVC – компонент Spring, що реалізує модель MVC (Model-View-Controller) для організації чіткої структури веб-додатку та забезпечення зручної обробки HTTP-запитів.

– Spring Security – потужний інструмент для налаштування автентифікації та авторизації користувачів, що гарантує безпечний доступ до ресурсів системи.

– Spring Boot DevTools – набір інструментів для полегшення процесу розробки, зокрема підтримка автоматичного перезавантаження застосунку після змін у коді, що значно прискорює тестування та налагодження [17].

Обраний стек технологій забезпечує:

– надійну роботу серверної частини;

– зручну побудову клієнтської частини з адаптивним інтерфейсом;

– можливість гнучкої інтеграції з зовнішніми сервісами (Telegram, Viber, email);

– високий рівень безпеки та підтримку масштабованості проєкту.

Поєднання зазначених фреймворків та бібліотек є оптимальним вибором для реалізації програмного засобу цифрової комунікації з клієнтами, що відповідає вимогам до функціональності, продуктивності та зручності експлуатації.

2.3 Огляд СУБД і способи зберігання даних клієнтів

СУБД – це інструмент, який допомагає зручно працювати з великими обсягами даних, наприклад: зберігати інформацію про клієнтів, повідомлення, замовлення, історію взаємодії.

Функції СУБД:

– Створення бази даних і таблиць (структура з полями: ім’я, телефон, email тощо).

– Додавання, редагування, видалення, пошук даних.

– Забезпечення цілісності й безпеки даних.

– Управління одночасним доступом багатьох користувачів.

– Резервне копіювання та відновлення даних.

Серед сучасних СУБД, що можуть бути використані для побудови такої системи, особливу увагу заслуговують MySQL, PostgreSQL, MongoDB, SQLite, IBM Db2 та H2 Database. Кожна з них має свої переваги та обмеження, які варто враховувати при виборі технологічного рішення залежно від специфіки проєкту [18].

MySQL є найпоширенішою реляційною системою управління базами даних, яка базується на використанні мови SQL і характеризується високою продуктивністю при роботі з великими обсягами структурованих даних. Вона активно застосовується для створення веб-застосунків, зокрема в поєднанні з мовами програмування PHP, Python, Node.js. MySQL підтримує реплікацію, що дає змогу організовувати резервне копіювання та горизонтальне масштабування, а також забезпечує цілісність та надійність даних за рахунок підтримки транзакцій. Завдяки відкритості коду та наявності безкоштовної версії MySQL залишається одним із найпопулярніших виборів для проєктів малого та середнього бізнесу. Проте варто зазначити, що при реалізації складних транзакційних сценаріїв або складної логіки обробки запитів MySQL поступається більш потужним системам, таким як PostgreSQL.

PostgreSQL є потужною об'єктно-реляційною системою управління базами даних, яка підтримує не лише стандартну SQL-логіку, але й розширені можливості

для роботи з JSON, XML, масивами та власними типами даних. Ця СУБД відзначається високим рівнем відповідності стандартам ACID, що гарантує атомарність, консистентність, ізоляцію та довговічність операцій з даними. PostgreSQL дозволяє створювати збережені процедури, власні оператори та функції, а також ефективно обробляти складні SQL-запити. Серед її переваг варто відзначити підтримку шардінгу та реплікації, що робить цю систему придатною для великих проєктів з високими навантаженнями. Однак для початківців конфігурація PostgreSQL може здатися дещо складнішою порівняно з MySQL, оскільки вимагає глибшого розуміння налаштувань оптимізації продуктивності та захисту даних.

MongoDB представляє собою документоорієнтовану NoSQL СУБД, яка зберігає дані у вигляді BSON-документів (розширена бінарна форма JSON). Цей підхід до зберігання дозволяє гнучко працювати з напівструктурованими та неструктурованими даними, що особливо корисно в умовах швидко змінної структури інформації. MongoDB має високі можливості горизонтального масштабування завдяки вбудованій підтримці шардінгу, а також підтримує реплікацію для забезпечення відмовостійкості системи. Окремою перевагою є розвинена система агрегації, яка дозволяє виконувати складні вибірки та аналіз даних без написання додаткового коду на стороні застосунку. Незважаючи на свою гнучкість, MongoDB не завжди є найкращим вибором для систем, де критично важлива жорстка відповідність ACID-принципам, хоча в останніх версіях система вже підтримує багатодокументні транзакції [18].

SQLite є вбудованою реляційною СУБД, яка не потребує окремого серверного програмного забезпечення та зберігає всі дані у вигляді одного файлу. Вона активно використовується в мобільних застосунках, десктопних програмах та IoT-пристроях завдяки своїй простоті, компактності та швидкодії при роботі з невеликими обсягами даних. SQLite підтримує більшість SQL-операцій, що робить її зручною для реалізації локальних баз даних без складних налаштувань сервера. Основним обмеженням цієї СУБД є її масштабованість, адже вона не призначена для роботи в умовах великої кількості одночасних підключень або великих обсягів інформації.

IBM Db2 – це високопродуктивна корпоративна реляційна СУБД, яка розроблена компанією IBM та орієнтована на використання в складних інформаційних системах з великими транзакційними навантаженнями. Db2 підтримує як реляційне, так і NoSQL-зберігання, що дозволяє ефективно поєднувати структуровані та неструктуровані дані в межах однієї системи. Однією з переваг цієї СУБД є інтеграція з аналітичними інструментами та засобами машинного навчання, що забезпечує розширені можливості для бізнес-аналітики та прийняття рішень на основі даних. Високий рівень безпеки, підтримка масштабованості та відповідність корпоративним стандартам роблять IBM Db2 придатною для великих організацій. Проте висока вартість ліцензування та складність адміністрування можуть бути обмеженням для невеликих компаній.

H2 Database є легкою вбудованою реляційною системою управління базами даних, яка написана на Java і часто використовується для цілей тестування, розробки та прототипування. Вона підтримує SQL і сумісна з JDBC, що спрощує її інтеграцію в Java-застосунки. H2 може працювати як у вбудованому режимі, так і в клієнт-серверній архітектурі. Завдяки можливості зберігання даних у пам’яті, ця СУБД забезпечує високу швидкість обробки запитів, що робить її особливо корисною для автоматизованих тестів. Основним обмеженням є те, що H2 Database не призначена для великих продакшн-систем та не має розвинених можливостей для реплікації чи масштабування.

Вибір конкретної СУБД для реалізації системи підтримки комунікації з клієнтами залежить від вимог до обсягів зберігання, необхідності масштабування, підтримки транзакцій, рівня безпеки та простоти інтеграції з іншими компонентами програмного забезпечення.

2.4 Інструменти для створення веб- та мобільного інтерфейсу

Ефективна взаємодія користувача з інформаційною системою неможлива без зручного, інтуїтивно зрозумілого та адаптивного інтерфейсу. Для реалізації клієнтської частини веб-застосунку, орієнтованого на підтримку комунікації з

клієнтами, доцільно використовувати сучасні технології та бібліотеки, що забезпечують швидку розробку, адаптивність під різні типи пристроїв, а також легку інтеграцію з серверною логікою.

Однією з технологій для побудови інтерфейсу є мова розмітки HTML (HyperText Markup Language), яка відповідає за структуру та базове форматування елементів сторінки. HTML є стандартом для розробки веб-сторінок і визначає, які елементи (текстові блоки, зображення, кнопки, форми) будуть відображатися в браузері та яким чином вони будуть організовані. Всі сучасні інструменти побудови інтерфейсу використовують HTML як основу для створення клієнтської частини.

Для спрощення генерації HTML-коду в Java-застосунках активно використовується шаблонізатор Thymeleaf. Цей інструмент дозволяє динамічно підставляти дані з серверної частини безпосередньо в HTML-сторінки, що особливо зручно при створенні форм, таблиць, повідомлень та інших елементів динамічного контенту. Thymeleaf інтегрується з Spring Boot, що забезпечує ефективне поєднання серверної логіки та клієнтського інтерфейсу в одному середовищі розробки [19].

Bootstrap – CSS-фреймворк, що надає готові стилі та компоненти для побудови адаптивного дизайну. Завдяки використанню Bootstrap можна швидко організувати коректне відображення веб-інтерфейсу на різних пристроях, включаючи десктопи, планшети та смартфони. До переваг цього фреймворку належать простота налаштування, наявність вбудованих класів для оформлення кнопок, форм, карток, таблиць та інших елементів, а також система сіток (grid system), яка полегшує створення адаптивних макетів.

Для реалізації більш складних інтерактивних елементів, таких як форми авторизації, динамічне оновлення вмісту без перезавантаження сторінки або робота з REST API, доцільно використовувати сучасні JavaScript-фреймворки та бібліотеки. Серед них особливе місце займає Angular – повноцінний фреймворк для створення односторінкових застосунків (SPA). Angular базується на TypeScript, має потужну систему компонентів, підтримку двостороннього зв'язку (two-way data

binding). Це дозволяє створювати масштабовані клієнтські застосунки з чіткою структурою та високим рівнем повторного використання коду.

Альтернативою Angular є React, також відомий як React.js або ReactJS. Це популярна JavaScript-бібліотека, створена компанією Facebook для побудови інтерфейсів користувача. React реалізує концепцію компонентного підходу, де інтерфейс складається з незалежних блоків (компонентів), кожен із яких відповідає за певну частину сторінки. Однією з ключових переваг React є використання Virtual DOM, що дозволяє оптимізувати оновлення сторінки та забезпечити високу продуктивність навіть при великій кількості динамічних змін. React активно застосовується для створення як веб-, так і мобільних застосунків (з використанням React Native).

Поєднання HTML, Thymeleaf, Bootstrap, Angular і React забезпечує можливість реалізувати зручний, адаптивний та функціональний інтерфейс користувача, що відповідає сучасним стандартам веб-розробки. Вибір конкретного інструменту залежить від складності проєкту, вимог до інтерактивності, необхідності підтримки мобільних пристроїв та інтеграції з бекенд-частиною системи.

Застосування цих технологій дозволяє забезпечити високу якість користувацького досвіду, зменшити час розробки та спростити подальший супровід веб-застосунку. Крім того, гнучкість інтеграції між фронтендом і бекендом створює основу для масштабування системи відповідно до зростаючих потреб бізнесу або користувачів.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСОБУ

3.1 Архітектура системи підтримки клієнтських комунікацій

Розроблена система підтримки цифрової комунікації з клієнтами реалізована на основі клієнт-серверної архітектури, що забезпечує розподіл функціональних обов’язків між клієнтською частиною (фронтендом), серверною логікою та системою управління базами даних (див. рисунок 3.1). Такий підхід дозволяє централізовано обробляти запити користувачів, реалізовувати функції обміну повідомленнями, створення публікацій та коментарів, а також забезпечує надійне збереження інформації про взаємодію у структурованому сховищі. Завдяки цьому система є масштабованою, зручною для користувача та легкою у супроводі.

На стороні користувача (Client) реалізовано доступ до веб-інтерфейсу, який дозволяє здійснювати реєстрацію, надсилати повідомлення, створювати публікації та переглядати комунікаційну активність у системі. Клієнтська частина (Frontend) реалізована з використанням HTML/CSS, JavaScript та Bootstrap, що забезпечує інтуїтивну взаємодію та адаптивний дизайн для різних пристроїв. Передача даних до серверної частини відбувається через HTTP-запити із застосуванням REST API. Серверна частина (Server) реалізована за допомогою Java (Spring Boot), де розгорнуто основну бізнес-логіку: обробка повідомлень, керування користувачами, постами та коментарями. Вона забезпечує обробку запитів, валідацію, маршрутизацію та відповідь клієнту.

Уся інформація про взаємодію, облікові записи, публікації та історію комунікацій зберігається в базі даних (Database) — PostgreSQL, яка забезпечує надійне зберігання даних, їх цілісність і можливість масштабування. Обрана архітектура дозволяє легко доповнювати систему новими модулями, підтримувати безперервну роботу та гарантує стабільність навіть при великій кількості одночасних запитів.

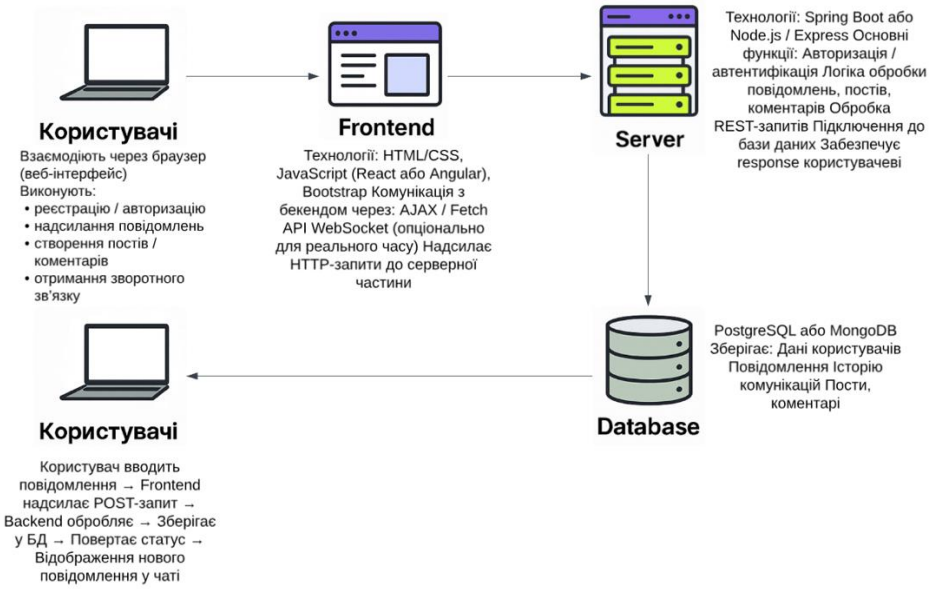


Рис. 3.1 Схематичне зображення клієнт-серверної архітектури системи підтримки клієнтських комунікацій

Серверна частина розробленого додатку виконує ключові функції з обробки даних, організації бізнес-логіки та взаємодії з базою даних. Вона складається з кількох основних компонентів: бази даних, класів-сутностей, сервісних класів, а також контролерів, які реалізують взаємодію між різними шарами архітектури.

На рисунку 3.2 представлено ієрархічну структуру класів, яка відображає взаємозв'язки між різними складовими серверної частини додатку [18-19].

Основні елементи серверної частини:

– DiplomaApplication – головна точка входу до додатку, з якої починається виконання програми.

– DataLoader – клас, що відповідає за початкову ініціалізацію бази даних у консольному режимі.

Контролери (пакет controller):

– WebController – базовий контролер у межах MVC-архітектури, забезпечує передачу інформації від моделі (Model) до представлення (View).

– UserController – контролер для управління операціями з користувачами: створення, редагування, видалення та відображення.

– PostController – контролер, який відповідає за дії з постами в системі.

Моделі (пакет model):

– User – клас-сутність, що представляє користувача у базі даних.

– Post – клас-сутність, що описує об’єкти постів.

– Comment – клас-сутність, яка зберігає інформацію про коментарі.

– Role – перелік ролей користувачів, що визначають їхні права в системі.

Репозиторії (пакет repository):

– UserRepository – репозиторій для взаємодії з таблицею користувачів, успадковує інтерфейс CrudRepository.

– PostRepository – репозиторій для управління постами.

– CommentRepository – репозиторій для роботи з коментарями.

Сервіси (пакет service):

– UserService – сервісний інтерфейс для створення та перегляду користувачів.

– PostService – сервіс, що відповідає за керування постами, включно з їх створенням, редагуванням, переглядом і видаленням.

– CommentService – сервісний інтерфейс для обробки операцій із коментарями.

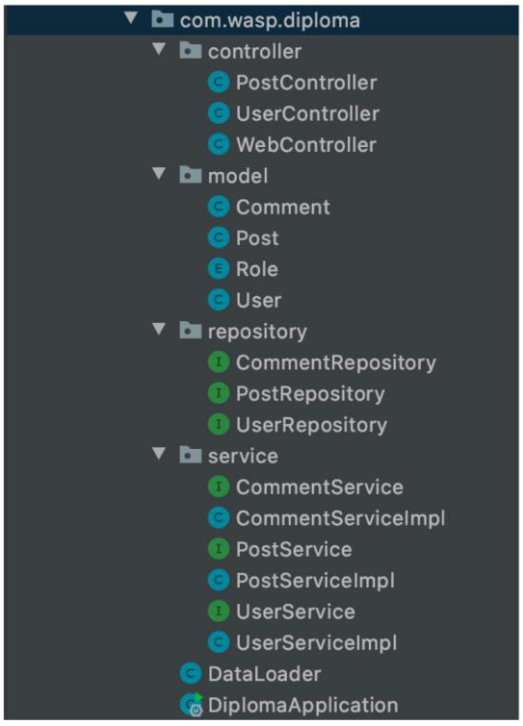


Рис. 3.2 Ієрархічна структура класів серверної частини додатку

Для коректної роботи додатку необхідно попередньо визначити всі необхідні залежності, а також налаштувати властивості, які будуть використовуватись під час процесу компіляції та виконання програми [20].

Це дозволяє забезпечити стабільну інтеграцію з зовнішніми бібліотеками та сервісами, спростити налаштування середовища розробки, а також підвищити зручність розгортання проєкту.

На рисунку 3.3 представлено структуру конфігураційних файлів додатку, які відповідають за:

– оголошення сторонніх залежностей (наприклад, бібліотек, фреймворків);

– визначення властивостей додатку, що завантажуються під час запуску;

– конфігурацію параметрів середовища, таких як підключення до бази даних, налаштування портів, безпека та інші критичні для роботи системи опції.

Ця структура є невід'ємною частиною архітектури проєкту, оскільки забезпечує автоматизоване управління складовими додатку та їх взаємодію [21].

Рис. 3.3 Структура конфігураційних файлів для управління залежностями та властивостями додатку

Серед усієї структури конфігураційних файлів, зображеної на рисунку 4.3, особливу увагу слід приділити саме файлу build.gradle. Цей файл виконує ключову роль у процесі збирання додатку, оскільки містить опис усіх необхідних залежностей, що мають бути автоматично завантажені та підключені для коректного функціонування системи.

У конфігураційному файлі build.gradle визначаються зовнішні бібліотеки, плагіни та модулі, що забезпечують роботу окремих компонентів додатку, таких як база даних, веб-фреймворки, інструменти тестування тощо. Завдяки цьому процес інтеграції сторонніх рішень відбувається без потреби ручного налаштування кожної з бібліотек, що значно спрощує як розробку, так і подальше обслуговування проєкту [22].

Клієнтська частина є важливим компонентом веб-додатку, оскільки саме вона забезпечує взаємодію кінцевого користувача із системою. Основне завдання клієнтської частини – забезпечити зручний і зрозумілий інтерфейс для роботи з функціоналом додатку, що передбачає відображення інформації, отриманої від серверної частини, а також обробку дій користувача.

На рисунку 3.4 представлено структуру клієнтської частини розробленого веб-додатку, яка включає:

– HTML-сторінки – відповідають за структуру представлення інформації та взаємодію з елементами інтерфейсу.

– CSS-стилі – забезпечують візуальне оформлення елементів сторінок, підтримуючи єдиний дизайн і підвищуючи зручність користування.

– JavaScript-скрипти – реалізують динамічну поведінку елементів інтерфейсу, обробляють події, надсилають запити до серверної частини через API та обробляють відповіді.

– Файли конфігурації – визначають параметри роботи клієнтської частини та взаємодії з сервером (наприклад, налаштування маршрутів або підключення до API).

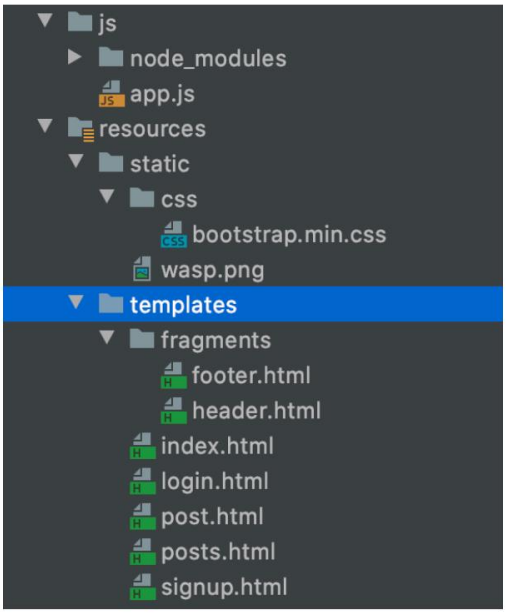


Рис. 3.4 Структура клієнтської частини веб-додатку

Клієнтська частина веб-додатку розроблена з дотриманням принципів модульності, що забезпечує зручність у підтримці, масштабованість та можливість повторного використання окремих компонентів інтерфейсу. Кожен модуль відповідає за певне відображення сторінки та має чітко визначену функціональність [22-24].

Основні елементи клієнтської частини представлені такими модулями:

index.html – головна сторінка додатку, яка відображає стартову інформацію для користувача.

login.html – сторінка авторизації, де користувач може ввести свої облікові дані для входу в систему.

signup.html – сторінка реєстрації нового користувача.

post.html – відображення окремого поста з повною інформацією та можливими коментарями.

posts.html – стрічка постів, яка показує перелік усіх доступних повідомлень у системі.

fragments/header.html – спільна верхня частина сайту (шапка), яка включає елементи навігації та інші загальні елементи інтерфейсу [25].

fragments/footer.html – спільна нижня частина сторінок, що використовується

для повторного застосування елементів футера.

Динамічна логіка клієнтської частини реалізована через файл app.js, який відповідає за рендеринг компонентів, організацію взаємодії між ними та надсилання запитів до серверної частини в реальному часі. Це дозволяє забезпечити інтерактивність користувацького інтерфейсу та швидке оновлення даних без необхідності перезавантаження сторінки.

На рисунку 3.5 представлено структуру файлів, що реалізують функціональність клієнтської частини з використанням ReactJS, що додатково підвищує гнучкість і модульність додатку.

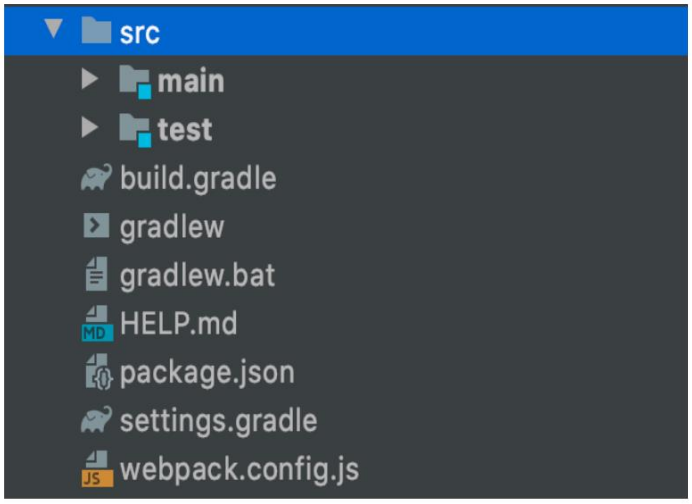


Рис. 3.5 Структура клієнтських файлів для реалізації ReactJS-компонентів

Окрім основних HTML-файлів і JavaScript-компонентів, важливу роль у клієнтській частині додатку відіграють конфігураційні файли, які забезпечують правильне збирання та налаштування фронтенд-частини проєкту.

До таких файлів належать [26]:

Package.json – конфігураційний файл, що містить метадані проєкту, включаючи назву додатку, версію, перелік залежностей, необхідних для роботи проєкту, а також скрипти для запуску, тестування та збирання додатку. Цей файл є основою для системи керування пакетами (npm або yarn) і забезпечує автоматизацію процесів розробки.

Webpack.config.js – конфігураційний файл для збирача модулів Webpack. У

цьому файлі визначено, що app.js є точкою входу додатку, а вихідним файлом є bundle.js, який формується під час компіляції.

Саме bundle.js вбудовується у HTML-сторінку та відповідає за коректне завантаження й виконання клієнтської логіки. Використання цих конфігураційних файлів дозволяє автоматизувати процеси компіляції та оптимізації клієнтської частини, зменшуючи розмір кінцевого файлу та покращуючи продуктивність додатку.

3.2 Загальний огляд проєкту та структура функціональних модулів

Розроблений проєкт має чітко структуровану архітектуру, яка складається з трьох основних компонентів:

Бекенд – відповідає за реалізацію бізнес-логіки додатку, взаємодію з базою даних та обробку серверних запитів.

Веб-інтерфейс користувача – забезпечує взаємодію кінцевого користувача із системою через зручний графічний інтерфейс.

Сервер – виконує функції з обробки HTTP-запитів, маршрутизації та передачі даних між клієнтом і бекендом [27].

Для ефективного управління залежностями, а також для автоматизації процесів збірки та запуску додатку, у даному проєкті було обрано систему збирання Gradle.

Особливості використання Gradle:

Gradle – це потужна система автоматизації побудови з відкритим кодом, яка поєднує в собі гнучкість Apache Ant та стандартизованість Apache Maven, але замість XML для конфігурації використовує власну декларативну мову, засновану на Groovy. Це значно спрощує написання скриптів і налаштувань.

Основною перевагою Gradle є використання спрямованого ациклічного графа (DAG), що дозволяє точно визначити порядок виконання завдань, уникаючи зайвих операцій. Gradle підтримує інкрементальну збірку, тобто автоматично аналізує, які частини проєкту були змінені, і збирає лише ті компоненти, що

потребують оновлення, що значно економить час розробника.

Ця система чудово підходить для масштабних проєктів із багатьма модулями, оскільки дозволяє централізовано керувати всіма залежностями та процесами збірки.

На рисунку 3.6 наведено приклад структури та результатів, які формуються після виконання команди gradle build [28].

```
> gradle build

:compileJava
:processResources
:classes
:jar
:assemble
:compileTestJava
:processTestResources
:testClasses
:test
:check
:build

BUILD SUCCESSFUL
```

Рис. 3.6 Результат виконання команди gradle build із відображенням етапів збірки проекту

– compileJava – компіляція основного Java-коду проєкту.

– processResources – обробка ресурсів, що використовуються у додатку.

– classes – створення класів після компіляції.

– jar – упаковка зібраних класів та ресурсів у архів .jar.

– assemble – етап збору всіх необхідних артефактів.

– compileTestJava – компіляція тестового Java-коду.

– processTestResources – обробка тестових ресурсів.

– testClasses – створення класів для тестування.

– test – запуск модульних тестів для перевірки працездатності окремих частин додатку.

– check – перевірка результатів тестування та інших можливих умов збірки.

– build – фінальна стадія збірки, яка об’єднує всі попередні кроки [29-30].

Вибір мови програмування

Аналізуючи різні варіанти реалізації програмного забезпечення, описані в попередньому розділі, можна зробити висновок, що для поставленого завдання найбільш доцільним є використання однієї з таких мов програмування: Python, PHP або Java. Інші мови, розглянуті під час аналізу, були відхилені через обмежену кросплатформеність або тісну прив’язаність до операційної системи Windows, тоді як розробка проєкту ведеться в середовищі Linux.

У результаті порівняння було прийнято рішення обрати саме Java. Ця мова програмування є однією з найпопулярніших у світі та активно застосовується для створення як десктопних, так і веб-додатків. Однією з ключових переваг Java є її незалежність від конкретної платформи завдяки використанню віртуальної машини Java (JVM). Це гарантує стабільність роботи застосунків на різних операційних системах без необхідності адаптації коду.

Ще однією вагомою перевагою Java є передбачуваність результатів виконання програми: незалежно від компілятора чи середовища розробки, поведінка додатку залишається однаковою. Це суттєво спрощує тестування та забезпечує високу надійність програмного продукту.

Java та Python

На етапі вибору мови програмування для реалізації даного проєкту особливу увагу було приділено двом популярним мовам – Python та Java. Обидві вони є потужними інструментами для створення різноманітних програмних рішень, однак мають певні відмінності, які варто враховувати при прийнятті рішення.

Python – це мова високого рівня, яка повністю підтримує об'єктно-орієнтоване програмування та відзначається простотою у використанні. Завдяки своїй лаконічності та читабельності коду, Python часто використовується як "клейова" мова, яка з'єднує різні компоненти системи. Вона ідеально підходить для розробки сценаріїв автоматизації, обробки даних, а також для побудови невеликих веб-додатків. Однак, незважаючи на свою гнучкість і простоту, Python має і певні

недоліки. Основним з них є повільніший час виконання програм у порівнянні з Java.

Це може стати критично важливим фактором для проєктів, які вимагають високої продуктивності та швидкодії [30].

Java є однією з найпопулярніших мов програмування у світі, широко використовуваною для створення складних багатокомпонентних систем, корпоративних рішень та веб-додатків. Хоча Java не є "чистою" об'єктно-орієнтованою мовою (оскільки в ній можливе використання простих типів даних поза класами), вона забезпечує високу стабільність, кросплатформеність і безпеку. Серед переваг Java також варто виділити значно ширшу підтримку бібліотек та інструментів для розробки великих проєктів, що робить її більш привабливою для реалізації продуктивних систем із розвиненою архітектурою.

Ще одним етапом у виборі мови програмування для реалізації проєкту було порівняння Java та PHP – двох мов, які активно використовуються для створення серверних застосунків, але мають суттєві відмінності у підходах та концепціях.

PHP є мовою сценаріїв, орієнтованою на серверну обробку запитів і призначеною переважно для створення динамічних веб-сторінок. Однією з характерних особливостей PHP є слабка типізація, що дозволяє програмісту не оголошувати типи змінних явно. Це часто робить розробку на PHP швидшою та простішою на початкових етапах, особливо в невеликих проєктах із обмеженими термінами.

Однак слабка типізація може призводити до меншої передбачуваності коду, збільшення кількості помилок на етапі виконання та ускладнювати підтримку проєкту в довгостроковій перспективі. Крім того, архітектурною особливістю PHP є те, що віртуальна машина перезапускається після кожного запиту, що потенційно знижує продуктивність та ускладнює реалізацію складних високонавантажених систем.

Java, на відміну від PHP, є строго типізованою мовою загального призначення. Це означає, що програміст повинен чітко вказувати типи даних для змінних і методів, що забезпечує додаткову стабільність, підвищує якість коду та полегшує його масштабування й підтримку.

Ще однією суттєвою перевагою Java є її кросплатформеність завдяки використанню Java Virtual Machine (JVM), що дозволяє запускати розроблені застосунки на будь-якій операційній системі без потреби модифікації коду. Крім того, Java має багату екосистему бібліотек та фреймворків, що значно полегшує реалізацію складних бізнес-процесів [30].

Вибір системи управління базами даних (СУБД)

Для реалізації бакалаврського проєкту було прийнято рішення використовувати реляційну систему управління базами даних H2 Database. Це сучасна, відкрита та кросплатформенна СУБД, написана мовою Java, що робить її ідеальним вибором для інтеграції з обраною серверною частиною проєкту.

Однією з ключових переваг H2 Database є її компактність (розмір дистрибутива складає трохи більше 1 МБ), а також широка функціональність, доступна без потреби встановлення додаткових компонентів.

Основні можливості H2 Database:

– Два режими роботи: вбудований та клієнт-сервер;
– Два варіанти зберігання даних: у файловій системі або в оперативній пам’яті;
– Підтримка планів виконання SQL-запитів;
– Кластеризація та реплікація для масштабування;
– Шифрування даних для підвищення безпеки;
– Зовнішні таблиці (linked tables);
– Наявність ODBC-драйвера для інтеграції з іншими системами;
– Повнотекстовий пошук;
– Визначення доменів та послідовностей;
– Мультиверсійний конкурентний доступ (MVCC);
– Тимчасові таблиці та обчислювані стовпці;
– Підтримка власних агрегатних функцій та збережених процедур;
– Робота з CSV-файлами (читання та запис);
– Вбудована браузерна консоль управління базою даних.

Ці характеристики роблять H2 Database привабливим вибором для навчальних проєктів і невеликих веб-додатків, де важлива простота налаштування та можливість швидкого розгортання.

Наведено порівняльну таблицю 3.5 H2 Database з іншими популярними реляційними системами управління базами даних, що дозволяє наочно продемонструвати її переваги [30].

Таблиця 3.5

Порівняльна таблиця H2 Database та інших популярних реляційних СУБД

Характеристика	H2	Derby	HSQLDB	MySQL	PostgreSQL
Pure Java	Yes	Yes	Yes	No	No
Memory Mode	Yes	Yes	Yes	No	No
Encrypted Database	Yes	Yes	Yes	No	No
ODBC Driver	Yes	No	No	Yes	Yes
Fulltext Search	Yes	No	No	Yes	Yes
Multi Version Concurrency	Yes	No	Yes	Yes	Yes
Footprint (embedded)	~2 MB	~3 MB	~1.5 MB	—	—
Footprint (client)	~500 KB	~600 KB	~1.5 MB	~1 MB	~700 KB

Вибір супровідних бібліотек

Для розробки проєкту були обрані наступні основні бібліотеки та технології, які забезпечують реалізацію бізнес-логіки, роботу з базою даних та побудову REST API.

Spring Data REST – це частина екосистеми Spring, яка спрощує створення RESTful веб-сервісів на основі репозиторіїв Spring Data. Вона автоматично генерує HTTP-ресурси на основі доменної моделі, що дозволяє швидко створювати CRUD-операції без написання зайвого коду.

Переваги використання Spring Data REST:

– Масштабованість взаємодії між компонентами системи;

– Єдність інтерфейсів;

– Незалежність реалізації окремих частин системи;

– Можливість використання проміжних компонентів для підвищення безпеки та зменшення затримок [30-31].

JPA – це офіційна специфікація для об’єктно-реляційного відображення (ORM) у Java-додатках. Вона визначає підходи до зберігання Java-об’єктів у реляційній базі даних. JPA не є конкретною бібліотекою, а лише задає стандарти, які реалізуються через різні ORM-фреймворки.

Hibernate – це одна з найпопулярніших реалізацій JPA та один із найзріліших ORM-фреймворків для Java. Hibernate відповідає за автоматичне перетворення Java-класів у таблиці бази даних і навпаки. Це дозволяє працювати з базою даних, оперуючи об’єктами, а не SQL-запитами.

Основні переваги Hibernate:

– Автоматичне відображення класів і полів у таблиці та стовпці БД;

– Зниження ризику помилок при ручному написанні SQL-запитів;

– Гнучке налаштування мапінгу об’єктів та зв’язків між ними;

– Підтримка транзакцій, кешування, зв'язків між сутностями.

Вибір саме Hibernate як реалізації JPA пояснюється його стабільністю, широкою підтримкою та простотою інтеграції в проєкти на основі Spring.

3.3 Реалізація фронтенду та інтеграції зі сторонніми сервісами

Для створення клієнтської частини веб-додатку було обрано комбінацію технологій Thymeleaf та Bootstrap, які забезпечують зручну генерацію HTML-сторінок і адаптивне відображення інтерфейсу на різних пристроях.

Thymeleaf – це Java-шаблонізатор для генерації XML, XHTML та HTML5-документів, який добре інтегрується з фреймворком Spring та підтримує побудову динамічних сторінок у середовищах MVC.

Основні переваги Thymeleaf:

– Працює як у веб-, так і в офлайн-середовищах, без жорсткої прив'язки до сервлетів.

– Підтримує модульну систему діалектів (стандартний та Spring-діалект), які можна розширювати.

– Має кілька режимів шаблонів: XML, XHTML, HTML5 з автоматичним приведенням коду до коректної структури.

– Підтримує інтернаціоналізацію та кешування шаблонів для підвищення продуктивності.

– Автоматично обробляє DOCTYPE для перевірки шаблонів та вихідного HTML-коду.

Ця технологія дозволяє ефективно організувати розділення логіки й представлення, а також спрощує роботу з HTML-шаблонами в екосистемі Spring.

Bootstrap – це популярний CSS-фреймворк для розробки адаптивного та мобільно-орієнтованого інтерфейсу.

Основні можливості Bootstrap:

– Система сіток для зручного розміщення елементів сторінки.

– Готові шаблони типографіки, таблиць, форм, медіа-контенту.

– Інструменти для навігації, сповіщень, алертів, модальних вікон та підказок.

– Адаптивний дизайн, що дозволяє коректно відображати інтерфейс на різних екранах та пристроях.

Комбінація Thymeleaf та Bootstrap дозволяє швидко створити сучасний веб-інтерфейс із чітким поділом логіки та представлення, забезпечуючи при цьому адаптивність і зручність користування.

Використання React та Webpack у клієнтській частині React

Для реалізації інтерактивного веб-інтерфейсу в проєкті використовується бібліотека React. Основною особливістю React є компонетний підхід, який

дозволяє розділити інтерфейс на незалежні блоки, що спрощує розробку та підтримку додатку.

Ключові особливості React [31]:

- Інтерфейс складається з компонентів, кожен із яких є окремою частиною сторінки.

Компоненти можуть бути:

- Функціональними – створюються за допомогою функцій, повертають JSX-розмітку.
- Класовими – створюються через ES6 класи, мають власний стан (state) та можуть передавати дані дочірнім компонентам через props.
- Під час рендерингу компоненти отримують вхідні дані у вигляді реквізитів (props).
- React може використовуватися не лише для рендерингу в DOM, але й для створення інших типів UI (наприклад, Canvas або серверна генерація HTML).

React активно застосовується великими компаніями, такими як Facebook, Netflix, PayPal для побудови динамічних і масштабованих веб-додатків.

Для збирання та організації клієнтської частини проєкту використовується Webpack – популярний збирач модулів із відкритим кодом.

Основні функції Webpack:

– Побудова графа залежностей та формування фінального пакету (bundle.js) з урахуванням всіх підключених модулів.

– Підтримка різних типів ресурсів: JavaScript, CSS, HTML, зображень (з використанням відповідних loaders).

– Гнучке налаштування через конфігураційний файл webpack.config.js, де задаються правила обробки файлів, плагіни, точки входу тощо.

– Можливість використання як через командний рядок, так і через налаштування в проєкті.

– Для роботи Webpack необхідне встановлене Node.js.

Ця технологічна зв'язка дозволяє ефективно організувати клієнтську частину додатку, забезпечити оптимізацію продуктивності та зручність масштабування.

3.4 Розгортання серверної частини і забезпечення безперервної роботи

Вибір серверного середовища: Spring Boot та Apache Tomcat

Для реалізації серверної частини додатку обрано фреймворк Spring Boot, який значно спрощує конфігурацію та розгортання Java-застосунків. Однією з ключових переваг Spring Boot є можливість запуску вбудованого веб-сервера без необхідності окремого налаштування зовнішнього сервера, що значно пришвидшує процес розробки та тестування.

Особливості Spring Boot:

– Автоматичне конфігурування проєкту за допомогою механізму auto-configuration;

– Вбудований веб-сервер Apache Tomcat (за замовчуванням), що дозволяє запускати застосунок як самостійне Java-застосування (standalone);

– Простий запуск застосунку через метод `main()` з анотацією `@SpringBootApplication`, яка об’єднує в собі `@Configuration`, `@EnableAutoConfiguration` та `@ComponentScan`;

– Відсутність потреби в ручному створенні архіву WAR чи окремого розгортання на зовнішньому сервері.

Apache Tomcat як вбудований сервер

Apache Tomcat – це широко використовуваний веб-сервер і контейнер сервлетів, який підтримує специфікації Java EE, такі як Servlet, JSP, WebSocket. У Spring Boot Tomcat інтегровано як вбудований сервер, що дозволяє запускати застосунок "із коробки" без додаткового налаштування [32].

Основні можливості Apache Tomcat:

– Підтримка HTTP/HTTPS (SSL/TLS);

– Просте керування портами та мережею;

– Гнучкі можливості конфігурації через властивості, YAML-файли або параметри командного рядка;

– Легка інтеграція з Spring MVC для побудови RESTful сервісів.

Налаштування Tomcat у Spring Boot.

За замовчуванням сервер запускається на порту 8080. Щоб змінити порт, достатньо вказати в application.properties:

server.port=9090

Для ввімкнення HTTPS можна додати SSL-конфігурацію у той самий файл:

server.port=8443

server.ssl.key-store=classpath:keystore.p12

server.ssl.key-store-password=yourpassword

server.ssl.keyStoreType=PKCS12

server.ssl.keyAlias=tomcat

При потребі використання іншої версії Tomcat достатньо змінити параметр tomcat.version у файлі збірки (build.gradle або pom.xml).

Тестування REST-ендпойнтів

Після запуску застосунку кінцеві точки можна протестувати, перейшовши у браузер за адресою:

http://localhost:8080/

Або ж скориставшись інструментами для API тестування, такими як Postman чи cURL.

Чому саме вбудований Tomcat у Spring Boot:

– Значно зменшується час на початкову конфігурацію;

– Відсутність необхідності в розгортанні WAR-файлів на сторонніх серверах;

– Зручність для мікросервісної архітектури, де кожен сервіс є незалежним застосунком зі своїм сервером;

– Гнучкість налаштування та автоматизація за допомогою Spring Boot.

Такий підхід повністю відповідає сучасним тенденціям розробки серверних додатків і спрощує процес CI/CD (безперервної інтеграції та доставки).

3.5 Взаємодія користувача з системою та приклади сценаріїв роботи

На рисунку 3.7 представлено структуру взаємодії між класами, сервісами та контролерами серверної частини веб-додатку. Діаграма ілюструє логіку організації основних елементів архітектури згідно з принципами MVC (Model-View-Controller) та використання сервісного шару для обробки бізнес-логіки.

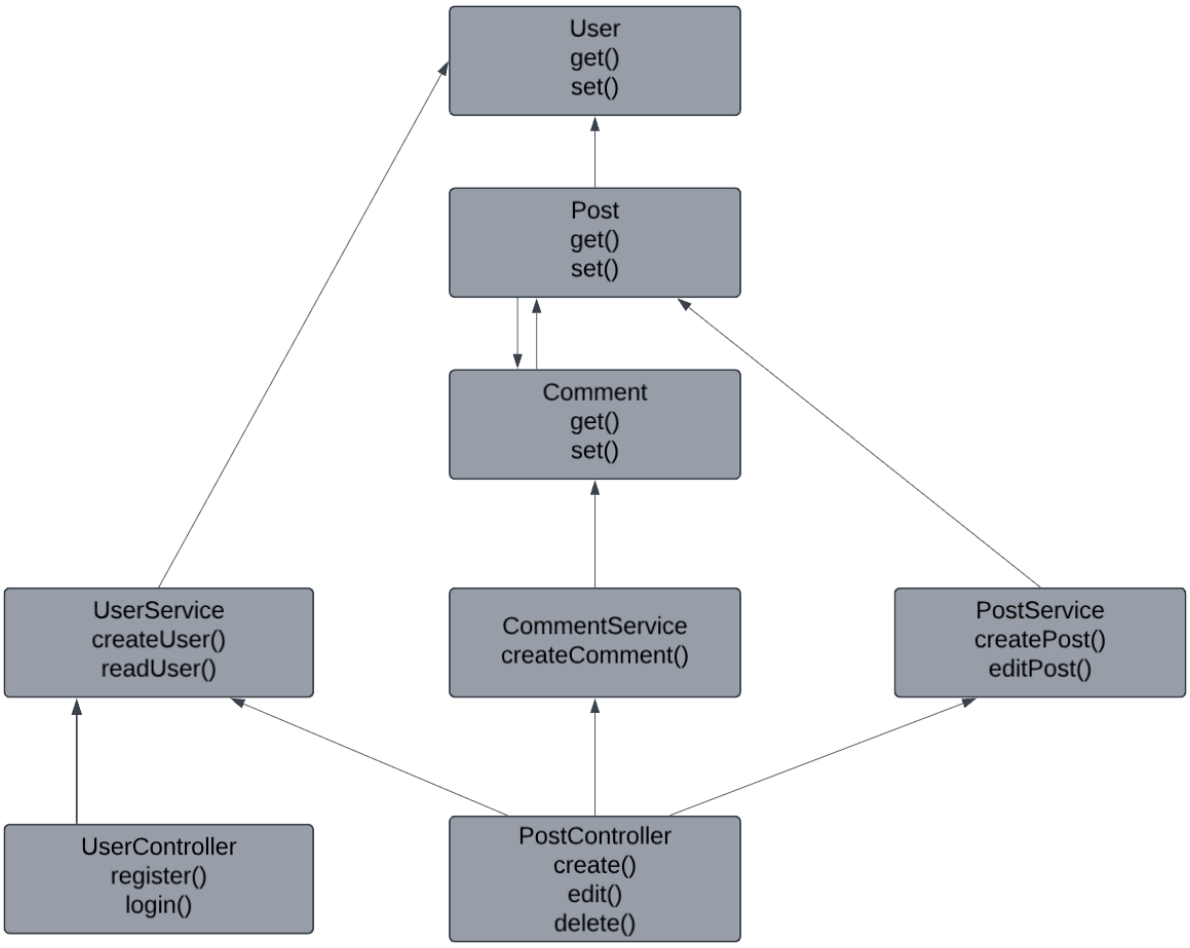


Рис. 3.7 Діаграма класів

На рисунку 3.8 зображено життєвий цикл бінів у Spring Framework.

Ця схема відображає послідовність етапів створення, ініціалізації та підготовки бінів до використання в контейнері Spring.

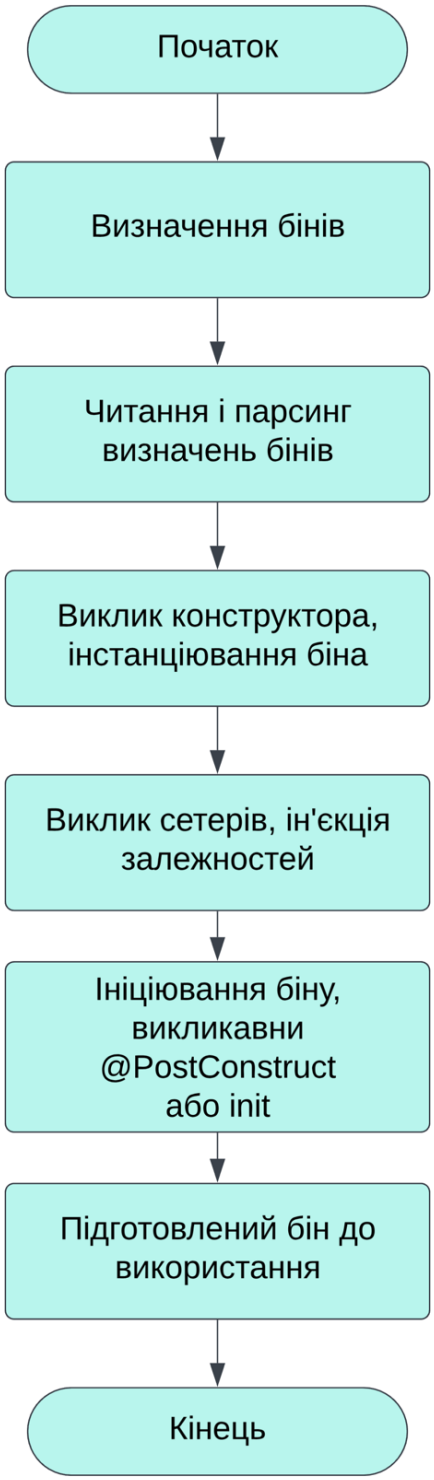


Рис. 3.8 Діаграма прецедентів

На рисунку 3.9 представлено діаграму варіантів використання (Use Case Diagram) для користувача системи. Діаграма описує основні сценарії взаємодії

користувача з додатком, включаючи можливості управління постами, редагування профілю та роботу із сесіями [33].

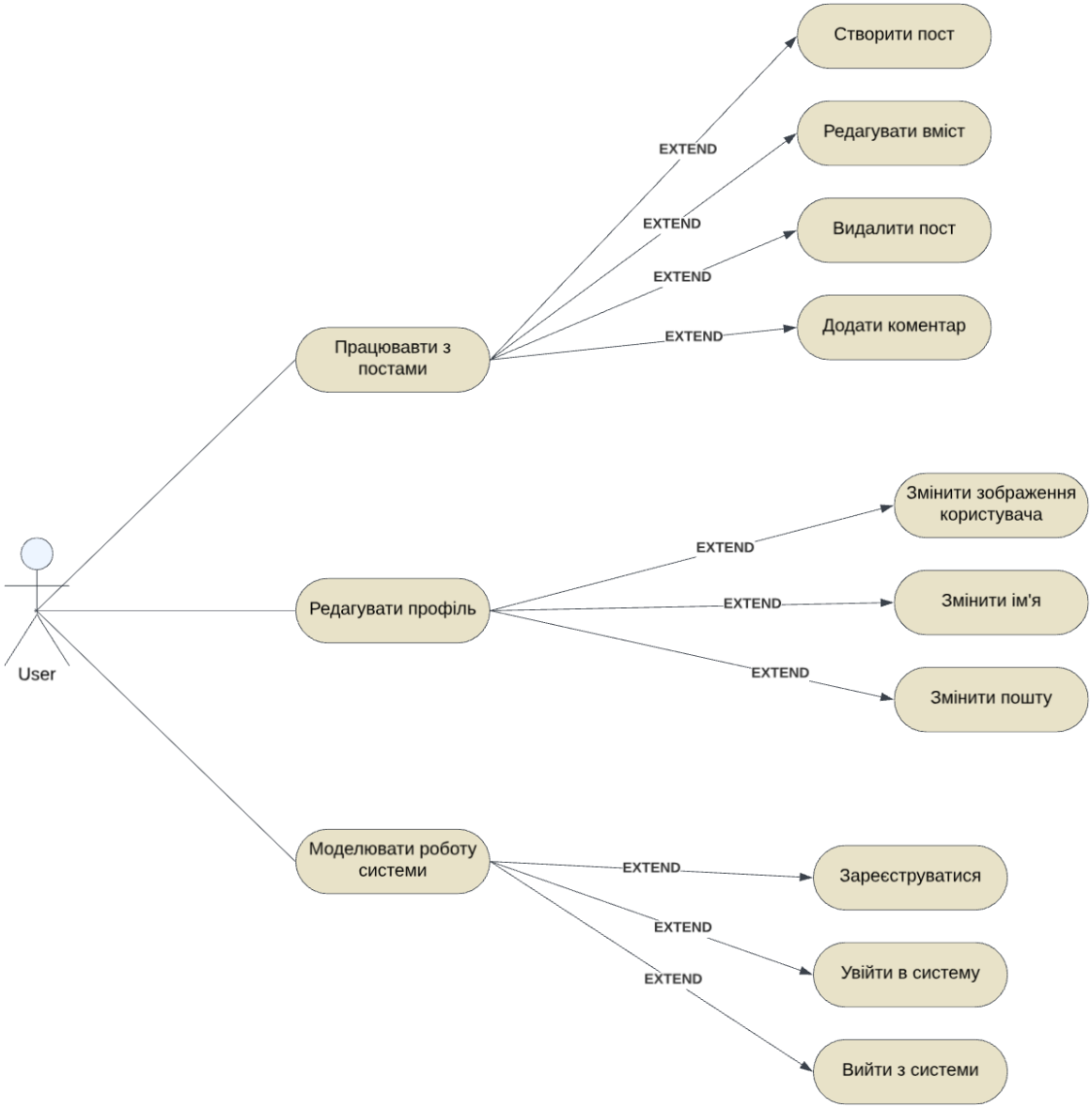


Рис. 3.9 Діаграма діяльності

Взаємодія користувача із системою починається зі сторінки реєстрації. Саме на цьому етапі відбувається створення нового облікового запису, що дозволяє користувачеві надалі повноцінно працювати з функціоналом додатку.

На сторінці реєстрації (див. рисунок 3.10, рисунок 3.11) користувач повинен заповнити просту форму, в якій вказуються такі дані:

– Електронна пошта – унікальний ідентифікатор користувача;

– Ім’я користувача – публічне ім’я, яке відображатиметься в системі;

– Пароль – секретний ключ для забезпечення безпеки облікового запису.

Цей етап є обов’язковим для початку роботи з системою, оскільки тільки зареєстровані користувачі мають доступ до функцій створення постів, редагування профілю, додавання коментарів та інших можливостей платформи [33].

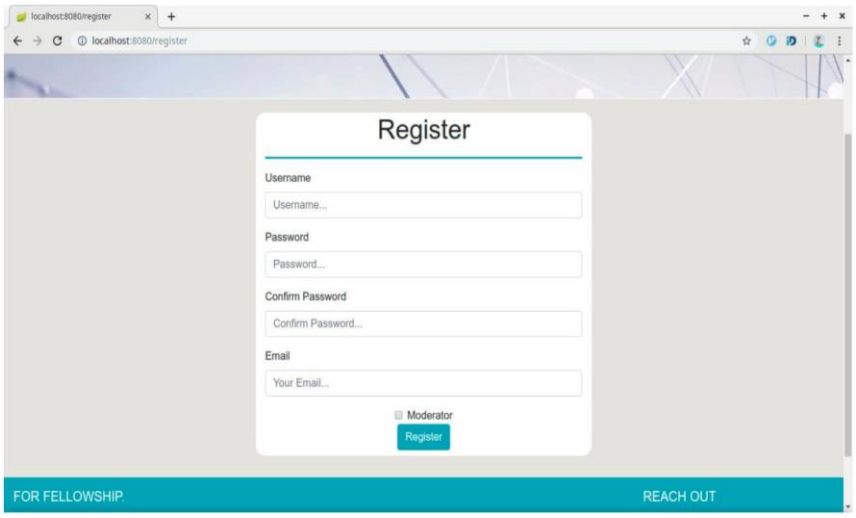


Рис. 3.10 Вигляд сторінки реєстрації

У разі, якщо користувач уже має обліковий запис у системі, наступним етапом його взаємодії буде авторизація. Для цього необхідно скористатися сторінкою входу до системи, де вводяться облікові дані (електронна пошта та пароль). На рисунках 3.11 та 3.12 представлено вигляд інтерфейсу сторінки логіну – до та після заповнення відповідних полів.

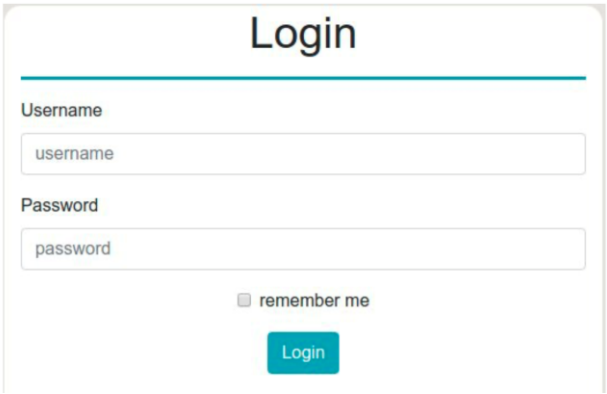


Рис. 3.11 Сторінка авторизації користувача перед введенням даних

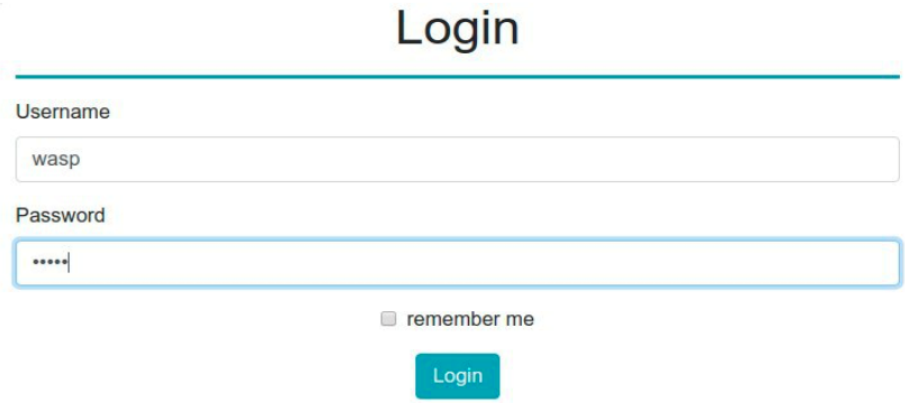


Рис. 3.12 Приклад заповненої форми авторизації користувача

Після успішної реєстрації або авторизації, користувач перенаправляється на головну сторінку веб-додатку, де відображається стрічка з останніми постами, створеними іншими користувачами (рисунок 3.13). Ця сторінка слугує основним інформаційним простором, з якого користувач може взаємодіяти з контентом платформи.

Для створення власного допису достатньо натиснути кнопку “Create Post”, після чого користувача буде перенаправлено на окрему сторінку створення нового поста (рисунок 3.14), де він зможе ввести заголовок, текст та інші дані публікації.

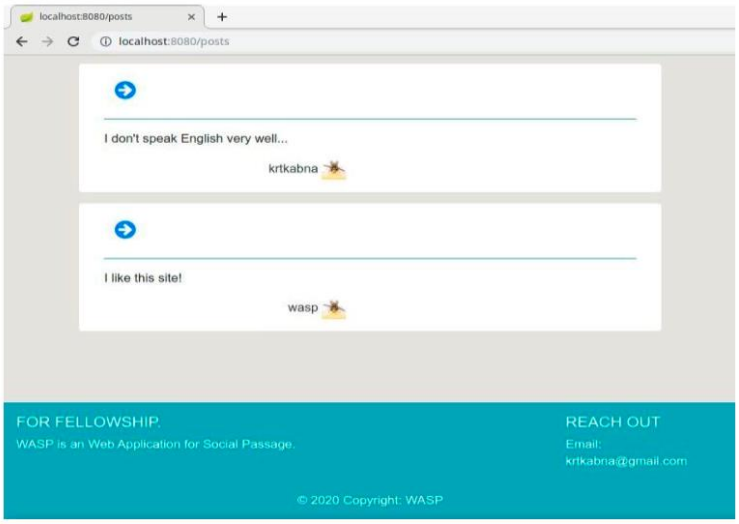


Рис. 3.13 Головна сторінка з відображенням останніх публікацій

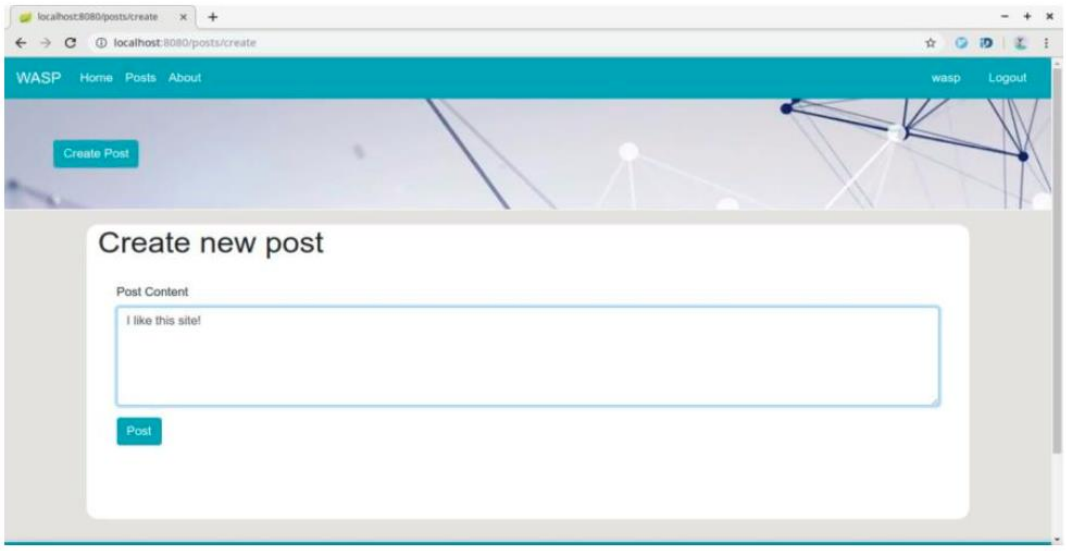


Рис. 3.14 Форма додавання нового допису користувачем

Для перегляду детальної інформації про пост користувач може натиснути на відповідну іконку переходу (синю стрілку) біля обраного допису. Після цього він потрапляє на сторінку перегляду конкретного поста, де відображається повний текст публікації та доступні коментарі інших користувачів (див. рисунок 3.15) [33].

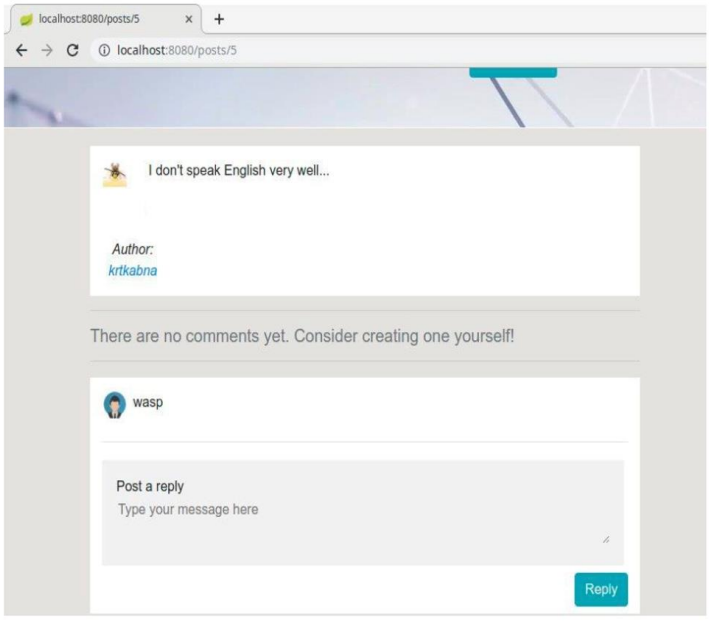


Рис. 3.15 Сторінка перегляду поста без коментарів

Як видно з попереднього рисунка, на даний момент для обраного поста відсутні коментарі. Користувач має можливість залишити власний відгук, ввівши

текст коментаря у відповідне поле та натиснувши кнопку “Reply” для його відправлення. Процес створення нового коментаря відображено на рисунку 3.16.

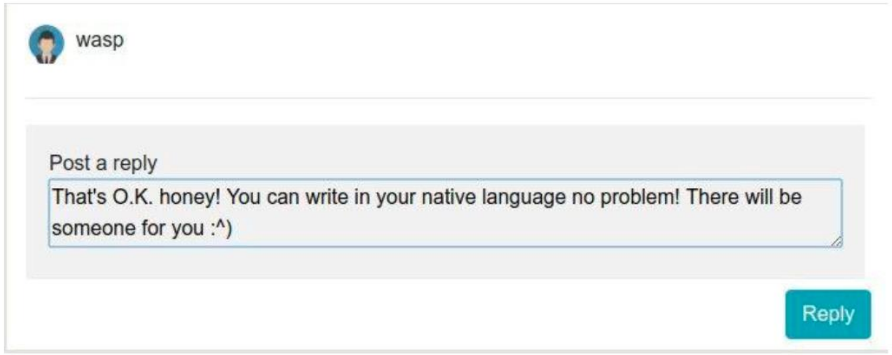


Рис. 3.16 Введення тексту коментаря та надсилання відгуку

На рисунку 3.17 представлено вигляд сторінки перегляду поста після успішного додавання коментаря. Коментар з’являється під текстом публікації в загальному списку, що дозволяє користувачам взаємодіяти між собою, залишаючи свої відгуки чи обговорення [34].

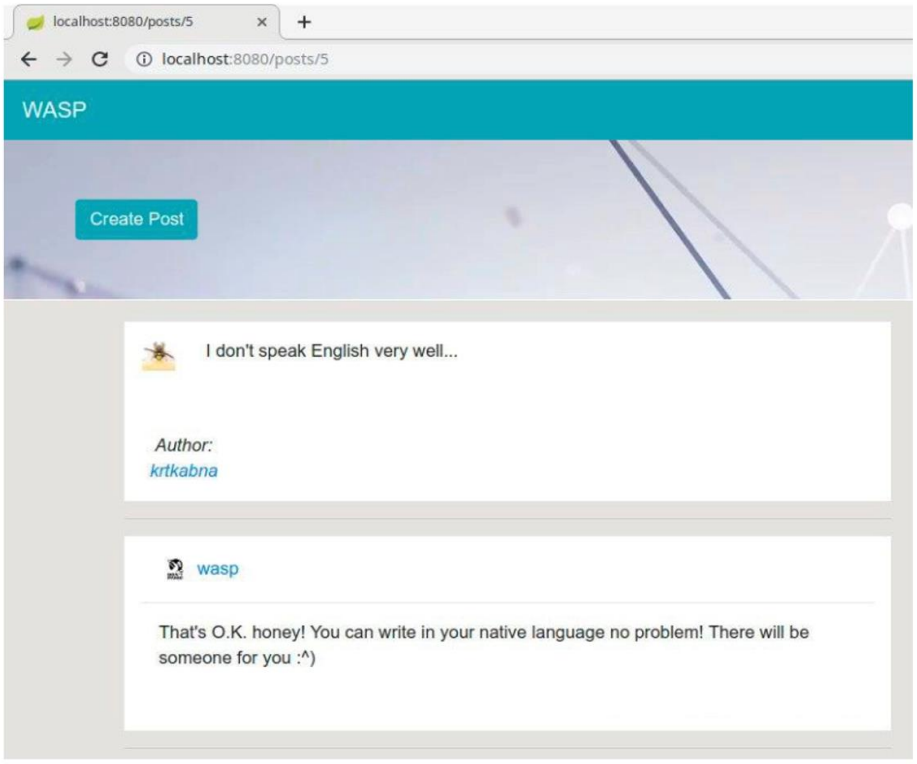


Рис. 3.17 Інтерфейс поста після додавання коментаря

Як видно з попередніх ілюстрацій, при перегляді поста користувач, який не є автором публікації, не має можливості змінювати чи видаляти її. У разі, якщо пост відкриває його автор, інтерфейс відображає додаткові елементи керування – кнопки “Edit” та “Delete”, що дозволяють змінювати або видаляти власний допис.

На рисунку 3.18 показано, як виглядає пост для його автора. Наявність кнопок редагування та видалення чітко відрізняє власні пости від чужих. При натисканні кнопки “Edit” автор переходить до форми редагування поста, приклад якої наведено на рисунку 3.19. У разі вибору “Delete” відбувається процес видалення публікації, що ілюструється на рисунку 3.20.

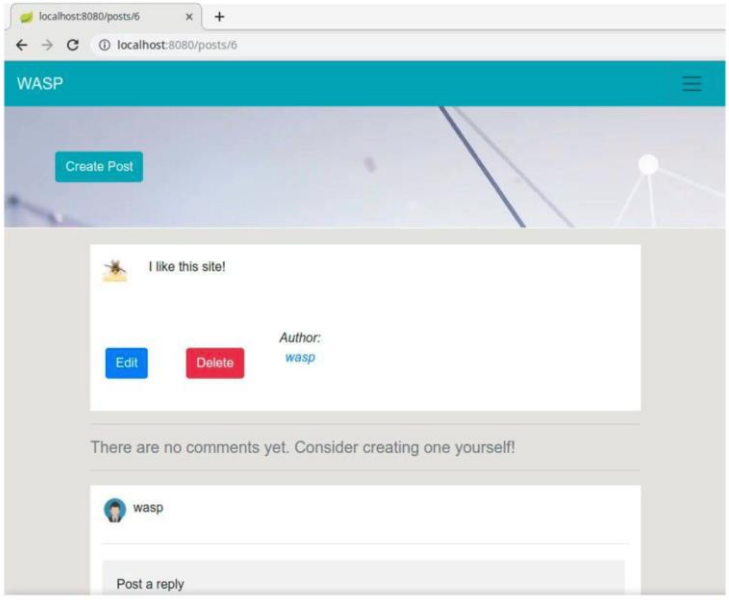


Рис. 3.18 Інтерфейс публікації, відкритої її автором (кнопки “Edit” та “Delete” доступні)

Після натискання кнопки “Edit”, користувач, який є автором поста, отримує доступ до форми редагування публікації. Ця форма відображається у вигляді спливаючого вікна, де можна змінити заголовок та вміст поста. Приклад вигляду вікна редагування представлено на рисунку 3.19 [34].

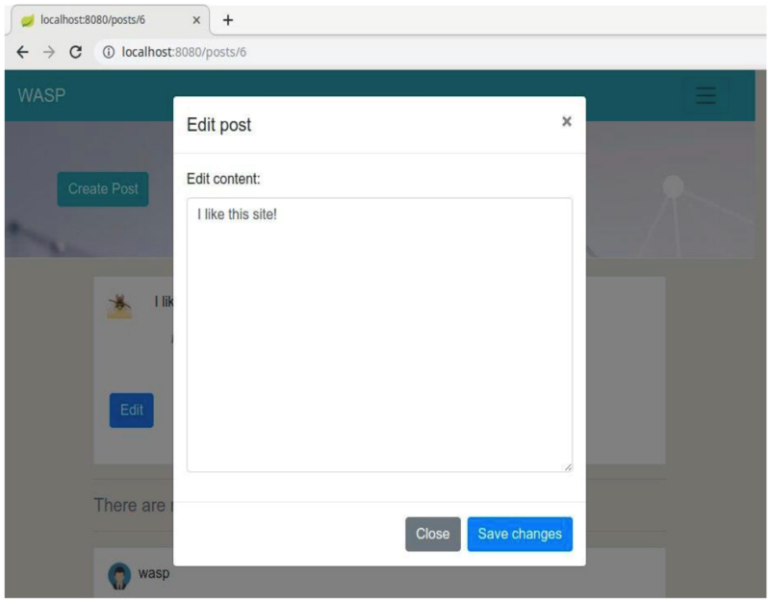


Рис. 3.19 Спливаюче вікно редагування публікації

На рисунку 3.20 представлено спливаюче вікно підтвердження видалення поста, яке з'являється після натискання кнопки "Delete". Це вікно слугує механізмом безпеки, що попереджає користувача про незворотність дії та потребує підтвердження перед остаточним видаленням публікації.

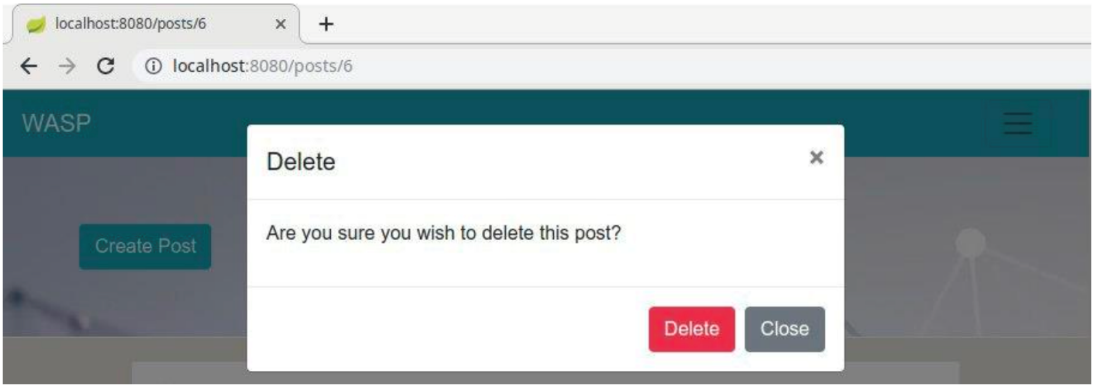


Рис. 3.20 Модальне вікно видалення поста

3.6 Порівняльний аналіз з аналогами та тестування

У процесі розробки даного додатку було проведено порівняння з рядом існуючих аналогів, які виконують схожі функції, зокрема системи для створення та обміну постами, описані в розділі 2. Основний акцент був зроблений на мінімалізм,

зручність використання та продуктивність.

Переваги розробленого додатку порівняно з конкурентами:

Компактність та швидкодія:

Додаток має невеликий розмір та оптимізовану структуру, що забезпечує швидке завантаження й мінімальне використання ресурсів системи, на відміну від важких конкурентних рішень.

Відсутність реклами:

Використання додатку не супроводжується сторонніми рекламними оголошеннями, що позитивно впливає на швидкодію, не дратує користувачів і дозволяє повністю зосередитись на функціоналі.

Відсутність нав'язливих сповіщень:

Додаток не відволікає користувача під час важливих справ, оскільки не надсилає постійних push-сповіщень, на відміну від багатьох конкурентів, які активно використовують цей канал для утримання аудиторії [34].

Характеристика	Розроблений додаток	Конкурент 1	Конкурент 2	Конкурент 3
Розмір додатку	Малий	Середній	Великий	Великий
Наявність реклами	Відсутня	Є	Є	Є
Наявність сповіщень	Відсутні	Є	Є	Є
Підтримка коментарів	Так	Так	Так	Частково
Швидкодія	Висока	Середня	Низька	Середня
Простота інтерфейсу	Висока	Середня	Низька	Середня
Інтеграція з базою даних	H2 (легка, вбудована)	MySQL	PostgreSQL	SQLite

Рис. 3.21 Порівняльна характеристика додатків

Після розробки всі основні модулі додатку були протестовані на коректність виконання функціоналу (див. рисунок 3.21).

Тестування проводилося вручну, а також частково із використанням автоматизованих Unit-тестів для серверної частини.

Методи тестування:

– Функціональне тестування: перевірка правильності роботи реєстрації, логіну, створення, редагування, видалення постів та коментування.

– Перевірка UX/UI: оцінка зручності інтерфейсу на різних пристроях (десктоп, мобільні телефони).

– Кросбраузерне тестування: тестування відображення інтерфейсу в популярних браузерах (Google Chrome, Mozilla Firefox, Microsoft Edge).

– Тестування безпеки: базова перевірка вразливостей (SQL-ін’єкції, XSS).

– Перевірка продуктивності: тестування часу завантаження сторінок і відповіді серверної частини.

Тип тестування	Статус виконання	Виявлені проблеми	Прийняті рішення
Реєстрація та логін	Успішно	Не виявлено	—
Створення та редагування постів	Успішно	Не виявлено	—
Додавання коментарів	Успішно	Не виявлено	—
Кросбраузерна сумісність	Успішно	Невелике зміщення кнопок у Safari	Виправлено в CSS
Перевірка безпеки	Частково	Відсутність захисту від SQL-ін’єкції на початковому етапі	Додано валідацію даних
Продуктивність	Успішно	Не виявлено	—

Рис. 3.22 Результати тестування

За результатами тестування додаток продемонстрував стабільну роботу всіх ключових функцій та відповідність заявленим вимогам (див. рисунок 3.22)[34].

ВИСНОВКИ

У результаті виконання бакалаврського проєкту було проаналізовано сучасний стан і підходи до організації цифрової комунікації з клієнтами, що дозволило окреслити ключові вимоги до побудови ефективної системи взаємодії. Було розглянуто роль CRM-систем, месенджерів та інших платформ автоматизації комунікацій, що на сьогодні є невід’ємною частиною бізнес-процесів компаній різних галузей.

В межах аналітичного етапу було розглянуто існуючі рішення в цій сфері, що дало змогу виявити їхні сильні та слабкі сторони. Це стало основою для формування вимог до майбутнього програмного засобу, серед яких ключовими є простота використання, відсутність зайвого функціоналу, висока продуктивність і незалежність від сторонніх сервісів реклами та нав'язливих сповіщень.

За допомогою проведеного аналізу був зроблений вибір технологічного стеку для реалізації системи, включаючи мови програмування, фреймворки, бібліотеки та системи управління базами даних. Було доведено доцільність застосування Java як основної мови розробки, Spring Boot як серверного фреймворку, H2 Database як легкої та зручної СУБД, а також React, Bootstrap і Thymeleaf для побудови адаптивного веб-інтерфейсу.

У розділі проєктування та реалізації програмного забезпечення було розроблено архітектуру системи, що забезпечує її стабільну роботу, масштабованість та можливість подальшого розвитку. Продемонстровано побудову клієнтської та серверної частин, їхню взаємодію через REST API, а також забезпечено реалізацію базових сценаріїв роботи користувачів із системою – від реєстрації та авторизації до створення постів, коментування та керування власними публікаціями.

Особливу увагу було приділено порівняльному аналізу розробленого додатку з аналогами, що дозволило виділити основні конкурентні переваги створеного рішення, такі як компактність, висока швидкодія, відсутність реклами та

нав’язливих сповіщень. Проведене тестування підтвердило працездатність усіх модулів системи та відповідність їх функціональності поставленим завданням.

У межах роботи було успішно розроблено, впроваджено та перевірено програмний засіб, який відповідає актуальним потребам користувачів у простій та зручній платформі для взаємодії. Результати проєкту демонструють можливість подальшого вдосконалення системи, зокрема шляхом інтеграції з популярними месенджерами та впровадження нових механізмів зворотного зв’язку.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Жеженко Є.О., Довженко Т.П. Платформа інтеграції CRM з Telegram для оперативного ведення діалогу з лідом. Матеріали VI Всеукраїнської науково-технічної конференції “Застосування програмного забезпечення в інформаційно-комунікаційних технологіях”, 24.04.2025, ДУІКТ, Київ, Україна, с. 610-614.

2. Жеженко Є.О., Довженко Т.П. Розробка програмних засобів для підтримки комунікації з клієнтами. Матеріали VII Міжнародна студентська конференція “Пріоритетні напрямки та вектори розвитку світової науки” 06.06.2025, УкрІНТЕІ, Суми, Україна (подано до друку).

ПЕРЕЛІК ПОСИЛАНЬ

1. Simone A., Kayfitz B. Google Flutter 2 Cookbook: Over 100 proven techniques and solutions to mobile development with Flutter and Dart. En, 2020. 587 p. – [Електронний ресурс] – https://www.storytel.com/tv/books/flutter-cookbook-over-100-proven-techniques-and-solutions-for-app-development-with-flutter-2-2-and-dart-5324219

2. Stankov S., Žitko B. and Grubišić A. Ontology as a Foundation for Knowledge Evaluation in Intelligent E-learning Systems. AIED'05 Workshop SW-EL'05: Applications of Semantic Web Technologies for E-Learning: Papers of 12th International Conference on Artificial Intelligence in Education (AIED 2005). Amsterdam, 2005. P. 361–367. – [Електронний ресурс] – https://www.researchgate.net/publication/234116282_Ontology_as_a_foundation_for_knowledge_evaluation_in_intelligent_e-learning_systems

3. Spring Framework – [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/blog/2025/01/23/spring-framework-7-0-0-M1-available-now

4. MSDN – Основи HTML – [Електронний ресурс] – Режим доступу до ресурсу: https://moodle.znu.edu.ua/enrol/index.php?id=80

5. IntelliJ IDEA [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/idea/?host=intellijidea.net.

6. Importance of Customer Relationship Management (CRM) – [Електронний ресурс] – Режим доступу до ресурсу: https://managementstudyguide.com/importance-of-crm.htm

7. ISO 8402:1994 Quality management and quality assurance – Vocabulary [Електронний ресурс] // ISO. – 1994. – Режим доступу до ресурсу: https://cdn.standards.iteh.ai/samples/20115/d87cc9045fac4bb483e60258c6844361/ISO-8402-1994.pdf.

8. Програми лояльності або як зробити покупця вашим союзником? послуг [Електронний ресурс] // Арт-група IMXO. – 2021. – Режим доступу до ресурсу: http://ukr.art-imxo.com.ua/category/article.

9. Як залучити клієнтів: топ 5 способів [Електронний ресурс] // Kyivstar business hub. – 2021. – Режим доступу до ресурсу: https://hub.kyivstar.ua/news/yak-zaluchiti-kliyentiv-top-5-sposobiv/.

10. Капелюшна Т. В. Експертна оцінка якості надання телекомунікаційних послуг / Т. В. Капелюшна, Р. А. Дименко [Електронний ресурс] // Ефективна економіка. – 2021. – №8. – Режим доступу до ресурсу: http://www.economy.nayka.com.ua/pdf/8_2021/96.pdf.

11. The official web-site of The Heritage Foundation [Електронний ресурс] // Index of Economic Freedom. – 2021. – Режим доступу до ресурсу: https://www.heritage.org/index/visualize?cnts=ukraine&type=2.

12. Zhang, Y., & Ali, B. (2019). A comprehensive study of web application development frameworks. Journal of Systems and Software, 158, 110-128. – [Електронний ресурс] – https://old.resourcepanel.org/sites/default/files/documents/document/media/resource_efficiency_and_climate_change_full_report.pdf

13. Rahman, M. M., & Kumar, A. (2020). A comparative study of front-end web development frameworks. International Journal of Computer Applications, 975, 1-5. – [Електронний ресурс] – https://www.researchgate.net/publication/328958566_Comparison_of_front-end_frameworks_for_web_applications_development

14. Gaur, S., & Choudhary, A. (2018). A review of popular web development frameworks and their comparison. International Journal of Advanced Research in Computer Science, 9(4), 174-178. – [Електронний ресурс] – https://www.researchgate.net/publication/336143161_A_Comparative_Analysis_on_Widely_used_Web_Frameworks_to_Choose_the_Requirement_based_Development_Technology

15. MySQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.mysql.com/products/workbench/.

16. What is PostgreSQL? [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresqltutorial.com/what-is-postgresql/.

17. IBM Db2 – Data Management Software [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/analytics/db2.

18. H2 Database [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/H2_(DBMS).

19. HyperText Markup Language [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/HTML.

20. Freeman, A., Robson, E., & Bates, B. (2020). Head First JavaScript Programming: A Brain-Friendly Guide (2nd ed.). O'Reilly Media. [Електронний ресурс] – https://nibmehub.com/opac-service/pdf/read/Head%20First%20JavaScript%20Programming%20_%20a%20learner's%20guide%20to%20JavaScript%20programming-compressed.pdf

21. Gamble, J., Helm, R., Johnson, R., & Vlissides, J. (2021). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional. [Електронний ресурс] – https://www.javier8a.com/itc/bd1/articulo.pdf

22. Elliott, J., & Freeman, E. (2020). Head First JavaScript: A Brain-Friendly Guide (2nd ed.). O'Reilly Media. [Електронний ресурс] – https://theswissbay.ch/pdf/Gentoomen%20Library/Programming/JavaScript/O%27Reilly%20Head%20First%20Javascript.pdf

23. Brown, J. (2020). Fullstack React: The Complete Guide to ReactJS and Friends. Independently published. [Електронний ресурс] – https://demo.smarttrainerlms.com/uploads/0003/trainings/course/45/modules/fullstack-react-book-r30_1510302324482009603.pdf

24. Simpson, K. (2021). You Don't Know JS Yet: Get Started. O'Reilly Media. [Електронний ресурс] – https://dokumen.pub/you-dont-know-js-yet-get-started-2nbsped-9781647862008.html

25. Mead, A. (2018). The Complete Node.js Developer Course (3rd ed.). Udemy. [Електронний ресурс] – https://libcatalog.usc.edu/discovery/fulldisplay?docid=alma9910432470356037 31&context=L&vid=01USC_INST:01USC&lang=en&search_scope=MyInst_and_CI&adaptor=Local%20Search%20Engine&tab=Everything&query=sub,exact,Node.js,AND&mode=

advanced&offset=30

26. Stack Overflow Developer Survey 2020. URL: https://insights.stackoverflow.com/survey/2020 (дата звернення: 20.02.2021).

27. Integration with DRF. https://djangofilter.readthedocs.io/en/stable/guide/rest_framework.html (дата звернення: 20.03.2025).

28. Ian Lewis. Mixins and Python. URL: https://www.ianlewis.org/en/mixins-andpython (дата звернення: 21.03.2025).

29. Internet Engineering Task Force. RFC-7519. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 10.03.2025).

30. MobX Readme. URL: https://mobx.js.org/README.html (дата звернення: 15.03.2025).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмних засобів для підтримки комунікації з клієнтами

Виконала: студентка групи ПД-42
Жеженко Єлизавета Олександрівна
Керівник: к.т.н., доцент кафедри
Довженко Тимур Павлович

Київ - 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: оптимізувати обробку клієнтських звернень через цифрові інструменти.

Об'єкт дослідження: процес взаємодії між організацією та клієнтами за допомогою цифрових каналів комунікації.

Предмет дослідження: веб-застосунок для підтримки комунікації з клієнтами, що забезпечує централізовану обробку звернень, інтеграцію з Telegram, WhatsApp та інструментами для аналітики взаємодій.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Дослідження та аналіз предметної галузі, існуючих програмних засобів для підтримки комунікації з клієнтами.
2. Визначення вимог до розробки програмних засобів.
3. Створення архітектури програмного забезпечення.
4. Реалізація фронтенду.
5. Розгортання серверної частини і забезпечення безперервної роботи.
6. Інтеграція зі сторонніми сервісами.
7. Тестування програмного засобу.

3

АНАЛІЗ АНАЛОГІВ

Інструмент	Автоматизація	Легкість використання	Можливість кастомізації	Інтеграції	Аналітика	Підтримка/ Спільнота	Локалізація
Zoho CRM	+	-	-	+	+	+	-
Chatfuel	-	+	-	+	-	+	+
ManyChat	-	+	-	+	-	-	+
SendPulse	+	+	-	+	+	+	-
Messenger	+	+	+	+	+	+	+

4

Вимоги до програмного забезпечення

Функціональні вимоги:

1. Створення облікових записів, вхід через логін/пароль
2. Збереження, редагування даних клієнтів
3. Використання двостороннього чату та історії листування
4. Інтеграція з месенджерами та email, автоматичне створення тікетів
5. Створення заявок, зміна статусу, призначення на операторів
6. Розробка статистики звернень, час відповіді, оцінка якості обслуговування

Нефункціональні вимоги:

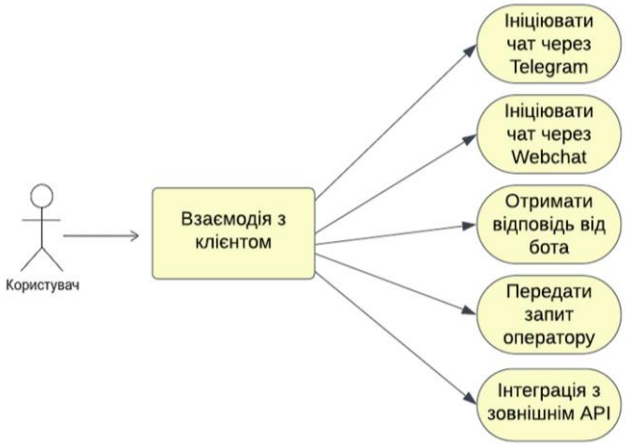
1. Швидкість завантаження не більше 1 секунди, автоматичне резервне копіювання
2. Шифрування паролів
3. Адаптація під мобільні пристрої

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

6

Взаємодія з клієнтом через месенджери



7

ПРОЦЕС ІНТЕГРАЦІЇ

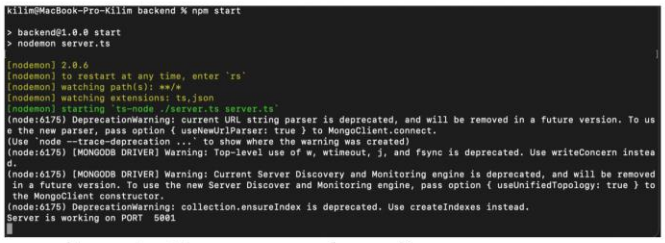


Рисунок 1 — Процесу запуску та інтеграції серверного застосунку з використанням nodemon та підключенням до MongoDB.

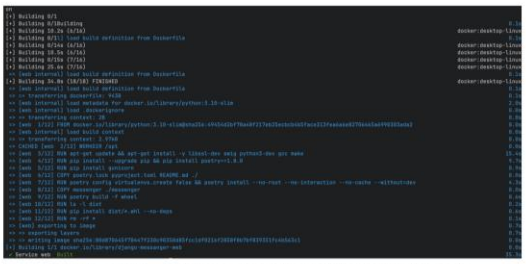


Рисунок 1.3 – Збірка докер образу (Docker Compose)

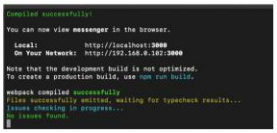


Рисунок 1.2 — Консольний вивід успішної компіляції frontend-застосунку Messenger.

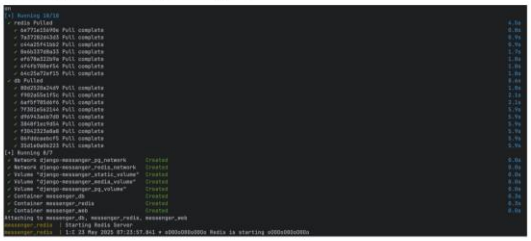


Рисунок 1.4 – Запуск контейнеру

Рисунок 1.5 – Конфігурація кластеру Mongo DB

8

ЕКРАННІ ФОРМИ

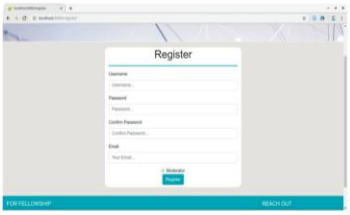


Рисунок 1 – Вигляд сторінки реєстрації

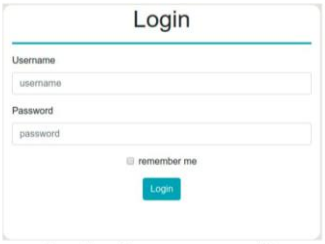


Рисунок 2 – Сторінка авторизації користувача перед введенням даних

Рисунок 3 – Приклад заповненої форми авторизації користувача

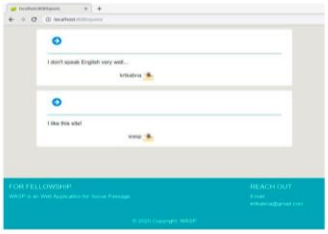


Рисунок 4– Головна сторінка з відображенням останніх публікацій

Рисунок 5 – Форма додавання нового допису користувачем

9

ЕКРАННІ ФОРМИ

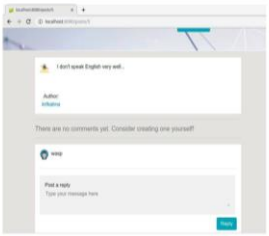


Рисунок 6 – Сторінка перегляду поста без коментарів

Рисунок 7 – Введення тексту коментаря та надсилання відгуку

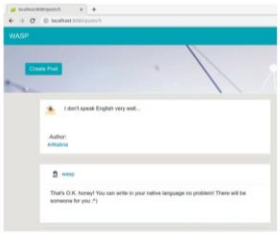


Рисунок 8 – Інтерфейс поста після додавання коментаря

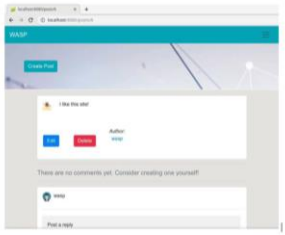


Рисунок 9 – Інтерфейс публікації, відкритої її автором (кнопки “Edit” та “Delete” доступні)

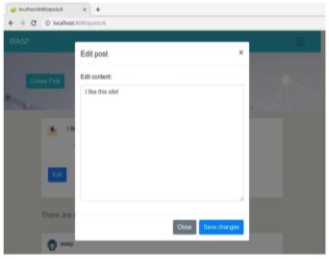


Рисунок 10 – Спливаюче вікно редагування публікації

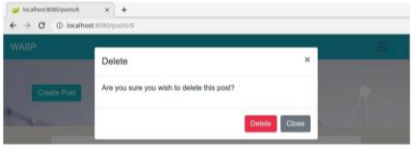


Рисунок 11 – Модальне вікно видалення поста

10

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Жеженко Є.О., Довженко Т.П. Платформа інтеграції CRM з Telegram для оперативного ведення діалогу з лідом. Матеріали VI Всеукраїнської науково-технічної конференції “Застосування програмного забезпечення в інформаційно-комунікаційних технологіях”, 24.04.2025, ДУІКТ, Київ, Україна, с. 610-614
2. Жеженко Є.О., Довженко Т.П. Розробка програмних засобів для підтримки комунікації з клієнтами. Матеріали VII Міжнародна студентська конференція “Пріоритетні напрямки та вектори розвитку світової науки” 06.06.2025, УкрІНТЕІ, Суми, Україна (подано до друку)

ВИСНОВКИ

1. Проведено аналіз сучасних підходів до цифрової взаємодії з користувачами у малому бізнесі, що підтвердив значення інтеграції месенджерів, CRM-систем та автоматизованих чат-ботів для покращення клієнтської підтримки.
2. На основі дослідження було здійснено вибір сучасних технологій і фреймворків для розробки, які забезпечують високу продуктивність, масштабованість та інтерактивність.
3. Спроєктовано клієнт-серверну архітектуру веб-застосунку, яка дозволяє рівномірно розподіляти функціональні обов’язки між фронтендом і бекендом, забезпечувати гнучкість і можливість подальшого розвитку системи.
4. Реалізовано основні функціональні вимоги.
5. Налаштовано збереження історії комунікацій у базі даних та обробку запитів, інтеграцію з зовнішніми сервісами, що гарантує цілісність даних і надійність роботи системи.
6. Проведено тестування розробленого веб-застосунку, яке підтвердило його функціональність, стабільність роботи та зручність використання кінцевими користувачами.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Клас DiplomaApplication - точка входу у програму

```
package com.wasp.diploma;
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplicatio
n;
@SpringBootApplication
public class DiplomaApplication {
public static void main(String[] args) {
SpringApplication.run(DiplomaApplication.class,
args);
}
}
```

ENTITIES

Клас User

```
package com.wasp.diploma;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import java.util.Objects;
@Entity
public class User {@Id @GeneratedValue(strategy =
GenerationType.IDENTITY)
private long id;
private String username;
public User() {
}
public User(String username) {
this.username = username;
}
public String getUsername() {
return username;
}
public void setUsername(String username) {
this.username = username;
}
@Override
public boolean equals(Object o) {
if (this == o) {
return true;
}
if (o == null || getClass() != o.getClass()) {
return false;
}
User user = (User) o;return id == user.id &&
Objects.equals(username, user.username);
}
@Override
public int hashCode() {
return Objects.hash(id, username);
}
@Override
public String toString() {
return "User{" +
"id=" + id +
", username='" + username + '\'' +
'}';
}
}
```

Клас Post

```
package com.wasp.diploma;
import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;import
java.time.LocalDateTime;
import java.util.Objects;
import java.util.Set;
@Entity
public class Post {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
private String title;
private String content;
private LocalDateTime publishDate;
@ManyToOne
private User user;
@OneToMany(
mappedBy = "post",
cascade = CascadeType.ALL,
orphanRemoval = true,
fetch = FetchType.EAGER
)
private Set<Comment> comments;
public Post() {
}public String getTitle() {
return title;
}
public void setTitle(String title) {
this.title = title;
}
public String getContent() {
return content;
}
public void setContent(String content) {
this.content = content;
}
public LocalDateTime getPublishDate() {
return publishDate;
}
public void setPublishDate(LocalDateTime publishDate) {
this.publishDate = publishDate;
}
public User getUser() {
return user;
}
public void setUser(User user) {this.user = user;
}
public Set<Comment> getComments() {
return comments;
}
public void setComments(Set<Comment> comments) {
this.comments = comments;
}
@Override
public boolean equals(Object o) {
if (this == o) return true;
if (!(o instanceof Post)) return false;
Post post = (Post) o;
return Objects.equals(getTitle(), post.getTitle()) &&
```

```
Objects.equals(getContent(),
post.getContent()) &&
Objects.equals(getPublishDate(),
post.getPublishDate()) &&
Objects.equals(getUser(), post.getUser()) &&
Objects.equals(getComments(),
post.getComments());
}
@Override
public int hashCode() {
return Objects.hash(getTitle(), getContent(),
getPublishDate(), getUser(), getComments());
}@Override
public String toString() {
return "Post{" +
"id=" + id +
", title='" + title + '\'' +
", content='" + content + '\'' +
", publishDate=" + publishDate +
", user=" + user +
", comments=" + comments +
'}';
}
}
```

Клас Comment

```
package com.wasp.diploma.model;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import java.time.LocalDateTime;
import java.util.Objects;
@Entity
public class Comment {@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
private String content;
private LocalDateTime publishDate;
@ManyToOne(fetch = FetchType.EAGER)
private Post post;
@ManyToOne(fetch = FetchType.EAGER)
private User user;
public Comment() {
}
public Comment(String content) {
this.content = content;
}
public String getContent() {
return content;
}
public void setContent(String content) {
this.content = content;
}public LocalDateTime getPublishDate() {
return publishDate;
}
public void setPublishDate(LocalDateTime
publishDate) {
this.publishDate = publishDate;
}
public Post getPost() {
return post;
}
public void setPost(Post post) {
this.post = post;
}
public User getUser() {
return user;
}
public void setUser(User user) {
this.user = user;
}
@Override
public boolean equals(Object o) {
if (this == o) return true;
if (!(o instanceof Comment)) return false;
Comment comment = (Comment) o;return
Objects.equals(getContent(),
comment.getContent()) &&
Objects.equals(getPublishDate(),
comment.getPublishDate()) &&
Objects.equals(getPost(), comment.getPost())
&&
Objects.equals(getUser(), comment.getUser());
}
@Override
public int hashCode() {
return Objects.hash(getContent(), getPublishDate(),
getPost(), getUser());
}
@Override
public String toString() {
return "Comment{" +
"id=" + id +
", content='" + content + '\'' +
", publishDate=" + publishDate +
", post=" + post +
", user=" + user +
'}';
}
}
```

Клас Role

```
package com.wasp.diploma.model;
import
org.springframework.security.core.GrantedAuthority;public
enum Role implements GrantedAuthority {
MODERATOR, AUTHENTICATED, UNAUTHENTICATED;
public String getAuthority() {
return name();
}
}
```

REPOSITORIES

Клас UserRepository

```
package com.wasp.diploma.repository;
import com.wasp.diploma.model.User;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;
@Repository
public interface UserRepository extends CrudRepository<User,
Long> {
}
```

Клас PostRepository

```
package com.wasp.diploma.repository;
import com.wasp.diploma.model.Post;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;
@Repositorypublic interface PostRepository extends
CrudRepository<Post,
Long> {
}
```

Клас CommentRepository

```
package com.wasp.diploma.repository;
import com.wasp.diploma.model.Comment;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;
@Repository
public interface CommentRepository extends
```

```
CrudRepository<Comment, Long> {
}
Клас UserService
package com.wasp.diploma.service;
import com.wasp.diploma.model.User;
public interface UserService {
User createUser(User user);
User readUser(User user);
}
SERVICES
Клас UserServiceImplpackage com.wasp.diploma.service;
import com.wasp.diploma.model.User;
import com.wasp.diploma.model.Role;
public class UserServiceImpl implements UserService {
private final UserRepository userRepository;
private final RoleService roleService;
@Autowired
public UserServiceImpl(UserRepository userRepository,
RoleService roleService) {
this.userRepository = userRepository;
this.roleService = roleService;
}
@Override
public void createUser(User user) {
RoleServiceModel roleServiceModel =
this.roleService.findByAuthority(userServiceModel.isModerato
r
() ? "MODERATOR" : "USER");
Role role = this.modelMapper.map(roleServiceModel,
Role.class);
user.addRole(role);
this.userRepository.save(user);
}
@Override
public void readUser() {this.postRepository.delete(post); }
}
Клас PostService
package com.wasp.diploma.service;
import com.wasp.diploma.model.Post;
public interface PostService {
void createPost();
void editPost(Post post, String changes);
void deletePost(Post post);
}
Клас PostServiceImpl
package com.wasp.diploma.service;
import com.wasp.diploma.model.Post;
import com.wasp.diploma.repository.PostRepository;
import
org.springframework.beans.factory.annotation.Autowired;
public class PostServiceImpl implements PostService {
private final PostRepository postRepository;
public PostServiceImpl() {}
@Autowired
public PostServiceImpl(PostRepository postRepository) {
this.postRepository = postRepository;
}
@Override
public Post editPost(Post post, String changes) {
post.setContent(changes);
this.postRepository.save(post);
}
@Override
public void deletePost(Post post) {
this.postRepository.delete(post);
}
}
Клас CommentService
package com.wasp.diploma.service;
import com.wasp.diploma.model.Comment;
public interface CommentService {
void createComment();
}Клас CommentServiceImpl
package com.wasp.diploma.service;
import com.wasp.diploma.model.Comment;
import com.wasp.diploma.repository.CommentRepository;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.time.LocalDateTime;
@Service
public class CommentServiceImpl implements
CommentService {
private final CommentRepository commentRepository;
@Autowired
public CommentServiceImpl(CommentRepository
commentRepository) {
this.commentRepository = commentRepository;
}
@Override
public void create(Comment comment) {
commentEntity.setPublishDate(LocalDateTime.now());
this.commentRepository.save(commentEntity);
}
}CONTROLLERS
Клас WebController
package com.wasp.diploma.controller;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class WebController {
@RequestMapping(value = "/")
public String index() {
return "index";
}
@GetMapping(value = "/login")
public String getLogin() {
return "login";
}
@GetMapping(value = "/signup")
public String getSignUp() {
return "signup";
}
@GetMapping(value = "/about")
public String getAbout() {
return "login";
}}
Клас UserController
package com.wasp.diploma.controller;
import com.wasp.diploma.service.UserService;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import
org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.servlet.ModelAndView;
import javax.servlet.http.HttpServletRequest;
import javax.validation.Valid;
public class UserController {
private final UserService userService;
private final ModelMapper modelMapper;
@Autowired
```

```
public UserController(UserService userService,
ModelMapper modelMapper) {
this.userService = userService;
this.modelMapper = modelMapper;
}@GetMapping("/register")
public ModelAndView register(@ModelAttribute
UserRegisterBindingModel userRegisterBindingModel) {
return super.view("views/users/register",
"Register");
}
@PostMapping("/register")
public ModelAndView registerConfirm(@Valid
@ModelAttribute UserRegisterBindingModel
userRegisterBindingModel,
BindingResult
bindingResult,
HttpServletRequest
request) {
if (bindingResult.hasErrors()) {
return super.view("views/users/register",
"Register");
}
UserServiceModel userServiceModel =
this.modelMapper.map(userRegisterBindingModel,
UserServiceModel.class);
this.userService.createUser(userServiceModel);
return super.redirect("/login");
}
@GetMapping("/users/{username}")
public ModelAndView userProfile(@PathVariable String
username) {
UserServiceModel userServiceModel =
this.userService.findByUsername(username);return
super.view("/views/users/profile",
userViewModel);
}
@GetMapping("/login")
public ModelAndView login(String error, ModelAndView
mav)
{
mav.addObject("viewName", "/views/users/login");
mav.setViewName("layout");
if (error != null) {
mav.addObject("error", "Wrong username or
password");
}
return mav;
}
}
```

Клас PostController

```
package com.wasp.diploma.controller;
import com.wasp.diploma.model.Post;
import com.wasp.diploma.service.CommentService;
import com.wasp.diploma.service.PostService;
import com.wasp.diploma.service.UserService;
import org.modelmapper.ModelMapper;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageImpl;
import org.springframework.data.domain.Pageable;import
org.springframework.data.web.PageableDefault;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.ModelAndView;
import javax.validation.Valid;
import java.util.ArrayList;
@Controller
@RequestMapping("/posts")
public class PostController extends BaseController {
private final PostService postService;
private final UserService userService;
private final CommentService commentService;
private final ModelMapper modelMapper;
@Autowired
public PostController(PostService postService,
UserService userService, CommentService createCommentDto,
ModelMapper modelMapper) {
this.postService = postService;
this.userService = userService;
this.commentService = createCommentDto;
this.modelMapper = modelMapper;
}@GetMapping("")
public ModelAndView allPosts(@PageableDefault(size = 10)
Pageable pageable, @RequestParam(value = "search", required
=
false) String searchGetParameter) {
Page<PostServiceModel> postServiceModels;
if (searchGetParameter != null) {
postServiceModels =
this.postService.findAllByName(searchGetParameter,
pageable);
} else {
postServiceModels =
this.postService.findAll(pageable);
}
List<PostViewModel> postViewModelList = new
ArrayList<>();
postServiceModels.forEach(postServiceModel -> {
PostViewModel postViewModel =
this.modelMapper.map(postServiceModel,
PostViewModel.class);
postViewModelList.add(postViewModel);
});
Page<PostViewModel> postViewModelPages = new
PageImpl<PostViewModel>(postViewModelList, pageable,
postServiceModels .getTotalElements());
return super.view("views/posts/all",
postViewModelPages);
}
@GetMapping("/create")
public ModelAndView createPost(@ModelAttribute
CreatePostBindingModel createPostBindingModel) {
List<PostViewModel> postViewModels = new
ArrayList<>();this.postService.findAll().forEach(postServiceM
odel
-> {
PostViewModel postViewModel =
this.modelMapper.map(postServiceModel,
Post.class);
postViewModels.add(postViewModel);
});
postViewModels);
}
return super.view("views/posts/create",
@PostMapping("/create")
public ModelAndView storePost(@Valid @ModelAttribute
CreatePostBindingModel createPostBindingModel,
BindingResult
bindingResult, Authentication authentication) {
PostServiceModel postServiceModel =
this.modelMapper.map(createPostBindingModel,
PostServiceModel.class);
UserServiceModel userServiceModel =
this.userService.findByUsername(authentication.getName());
postServiceModel.setUser(userServiceModel);
```

```
this.postService.create(postServiceModel);
return super.redirect("/posts");
}
@PostMapping("/{id}")
public ModelAndView storeAnswer(@Valid @ModelAttribute
CommentBindingModel commentBindingModel,
BindingResult
bindingResult, Authentication authentication,
@PathVariable(value =
"id", required = true) Long postId) {CommentServiceModel
commentServiceModel = new
CommentServiceModel();
commentServiceModel.setContent(commentBindingModel.get
Comment
Content());
PostServiceModel postServiceModel =
this.postService.findById(postId);
commentServiceModel.setPost(postServiceModel);
UserServiceModel userServiceModel =
this.userService.findByUsername(authentication.getName());
commentServiceModel.setUser(userServiceModel);
this.commentService.create(commentServiceModel);
return super.redirect("/posts/" + postId);
}
@PostMapping("/{id}/edit")
public ModelAndView editConfirm(@Valid @ModelAttribute
EditPostBindingModel editPostBindingModel, BindingResult
bindingResult, @PathVariable Long id) {
if (bindingResult.hasErrors()) {
return super.redirect("/posts/" + id);
}
PostServiceModel postServiceModel =
this.modelMapper.map(editPostBindingModel,
PostServiceModel.class);
this.postService.edit(postServiceModel);
return redirect("/posts/" + id);
}
@PostMapping("/{id}/delete")
public ModelAndView deleteConfirm(@PathVariable Long id)
{
this.postService.deleteById(id);
return redirect("/posts");}
}
```

DATALOADER

Клас DataLoader

```
package com.wasp.diploma;
import com.wasp.diploma.model.User;
import com.wasp.diploma.repository.UserRepository;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
@Component
public class DataLoader implements CommandLineRunner {
private final UserRepository repository;
@Autowired public DataLoader(UserRepository repository) {
this.repository = repository;
}
@Override
public void run(String... args) throws Exception {
this.repository.save(new User(“VespulaVulgaris”));
}
}
```

WEB INTERFACE

Клас app.jsconst *React* = require('react');

```
const ReactDOM = require('react-dom');
class App extends React.Component {
constructor(props) {
super(props);
this.state = {users: []};
}
componentDidMount() {
fetch('http://localhost:8080/wasp/users').then(response
=> {
this.setState({users:
response.entity._embedded.users});
});
}
render() {
return (
<UserList users={this.state.users}/>
)
}
}
class UserList extends React.Component {
render() {
const users = this.props.users.map(user =>
<User key={user._links.self.href} user={user}/>
);return (
<table>
<tbody>
<tr>
<th>User Name</th>
</tr>
{users}
</tbody>
</table>
)
}
}
class User extends React.Component{
render() {
return (
<tr>
<td>{this.props.user.username}</td>
</tr>
)
}
}
ReactDOM.render(
<App />,
document.getElementById('react')
)
```

Файл application.properties

```
spring.data.rest.base-path=/waspFRAGMENTS
```

Фрагмент header.html

```
<div class="header-nav p-5"
xmlns:th="http://www.w3.org/1999/
xhtml">
<div class="container-fluid row">
<form class="form-inline offset-sm-2 col-sm-6"
method="get" action="/posts">
<input name="search" class="form-control mr-sm-2"
type="search" th:placeholder="#{header.search}" aria-
label="Search">
<button class="btn btn-outline-info my-2 my-sm-0"
type="submit" th:text="#{header.search.button}"></button>
</form>
<a class="nav-link text-white col-sm-4" th:href="@{/
posts/create}">
<button class="btn btn-info header-button"
type="button" th:text="#{header.create.post}">
</button>
</a>
</div>
</div>
```

Фрагмент footer.html

```
<footer class="page-footer font-small text-white border-top
```

```
border-white bg-info" xmlns:th="http://www.w3.org/1999/
xhtml">
<div class="container-fluid text-center text-md-left
pt-3">
<div class="row">
<div class="col-md-6 mt-md-0 mt-3">
<h5 class="text-uppercase">For fellowship.</
h5><p th:text="#{footer.description}"></p>
</div>
<hr class="clearfix w-100 d-md-none pb-3">
<div class="col-md-3 mb-md-0 mb-3"></div>
<div class="col-md-3 mb-md-0 mb-3">
<h5 class="text-uppercase">Reach out</h5>
<ul class="list-unstyled">
<li>
<p>Email: krtkabna@gmail.com</p>
</li>
</ul>
</div>
</div>
</div>
<div class="footer-copyright text-center py-3">© 2020
Copyright:
<a class="text-white" th:href="@{/}">WASP</a>
</div>
</footer>
```

PAGES

Сторінка index.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<link href="css/bootstrap.min.css" rel="stylesheet"/>
<title>Home</title>
</head>
<body><!--
=====================================
========================
=============== -->
<!-- This content is only used for static prototyping
purposes (natural templates)-->
<!-- and is therefore entirely optional, as this markup
fragment will be included -->
<!-- from "fragments/header.html" at runtime.
-->
<!--
=====================================
========================
=============== -->
<div th:insert="fragments/general.html :: header"/>
<h1>Hi beech!</h1>
<div id="react"></div>
<script src="bundle.js"></script>
<p>
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Praesent scelerisque neque neque, ac elementum quam
dignissim interdum.
Phasellus et placerat elit. Lorem ipsum dolor sit amet,
consectetur adipiscing elit.
Praesent scelerisque neque neque, ac elementum quam
dignissim interdum.
Phasellus et placerat elit.
</p>
<footer th:insert="fragments/general.html :: footer"></
footer>
</body>
</html>
```

Сторінка login.html

```
<!DOCTYPE html>
<html lang="en"
xmlns:th="http://www.thymeleaf.org"><head>
<meta charset="UTF-8">
<link href="css/bootstrap.min.css" rel="stylesheet"/>
<title>Log In</title>
</head>
<body>
<th:block th:replace="fragments/general.html :: header"/>
<div class="container">
<div class="row justify-content-center">
<form>
<div class="form-group">
<label for="exampleInputUsername1">Username</label>
<input aria-describedby="usernameHelp" class="form-
control" id="exampleInputUsername1"
placeholder="username" type="text">
</div>
<div class="form-group">
<label for="exampleInputPassword1">Password</label>
<input class="form-control"
id="exampleInputPassword1" placeholder="password"
type="password">
</div>
<button class="btn btn-primary" type="submit">Submit</
button>
</form>
</div>
<div class="row"></div>
<p id="login-greeting">This content will be replaced by
Spring Security.</p>
<th:block th:replace="fragments/general.html :: footer"/>
</div>
</body></html>
```

Сторінка signup.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<link href="css/bootstrap.min.css" rel="stylesheet"/>
<title>Sign Up</title>
</head>
<body>
<th:block th:replace="fragments/general.html :: header"/>
<div class="container">
<div class="row justify-content-center">
<form>
<div class="form-group">
<label for="exampleInputEmail1">Email address</label>
<input aria-describedby="emailHelp" class="form-
control" id="exampleInputEmail1"
placeholder="Enter email"
type="text">
<small class="form-text text-muted"
id="emailHelp">We'll never share your email with anyone
else.</small>
</div>
<div class="form-group">
<label for="exampleInputUsername1">Username</label>
<input aria-describedby="usernameHelp" class="form-
control" id="exampleInputUsername1"
placeholder="username" type="text">
</div><div class="form-group">
<label for="exampleInputPassword1">Password</label>
<input class="form-control"
id="exampleInputPassword1" placeholder="Password"
type="password">
</div>
<button class="btn btn-primary" type="submit">Submit</
button>
```

```
</form>
</div>
</div>
<th:block th:replace="fragments/general.html :: footer"/>
</body>
</html>
```

Сторінка posts.html

```
<div class="container" xmlns:th="http://www.thymeleaf.org">
<div class="row">
<div class="col-lg-8">
<!-- RENDER POSTS -->
<th:block th:each="post : ${viewModel}">
<th:block th:include="~{fragments/posts/all/
single-post}"/>
</th:block>
<!-- PAGINATION -->
<div th:if="${viewModel.getTotalElements()} > 0">
<th:block th:include="~{fragments/posts/all/
pagination}"/>
</div><!-- MESSAGE FOR MISSING POSTS-->
<th:block th:include="~{fragments/posts/all/
message}"/>
</div>
<th:block th:include="~{fragments/sidebar}"/>
</div>
</div>
```

Сторінка post.html

```
<div class="bg-white p-3">
<div class="row">
<div class="avatar col-sm-1 col-md-1">
<img src="/img/wasp1.jpg" height="512"
width="512"/></div>
<div class="col-xs-11 col-sm-11 col-md-11 text-">
<div class="row">
<h4 class="col-12 post-title"
th:text="*{title}"></h4>
<i class="text-left col-12 text-secondary
mb-4"
th:text="|Published on
*{#temporals.format(publishDate, 'dd-MM-yyyy')} at
*{#temporals.format(publishDate, 'HH:mm')}|"></i>
</div>
<p class="text-wrap" th:text="*{content}"></p>
</div>
</div>
<div class="col-md-6 text-center row">
<th:block th:if="${#authentication.name} ==
*{user.username}">
<div class="col-md-8 d-flex justify-content-
center"><button data-toggle="modal" data-
target="#editModal"
class="btn m-4 text-white bg-
primary text-center">
Edit
</button>
<button class="btn m-4 btn-danger text-
center"
data-toggle="modal" data-
target="#deleteModal">
Delete
</button>
</div>
</th:block>
<th:block th:unless="${#authentication.name} ==
*{user.username}">
<div
sec:authorize="hasAuthority('MODERATOR')">
<div class="col-md-8 d-flex justify-
content-center">
<button class="btn m-1 btn-danger
text-center"
data-toggle="modal" data-
target="#deleteModal">
Delete
</button>
</div>
</div>
</th:block>
<th:block th:unless="${#authentication.name} ==
*{user.username}">
<div class="offset-md-8"></div>
</th:block>
<div class="text-center col-md-4 text-center"><i>
Author: <a th:href="@{/users/{username}
(username=*{user.username})}"
th:text="*{user.username}"></
a>
</i>
</div>
</div>
</div>
</div>
```