ВСТУП

У сучасному цифровому світі веб-додатки відіграють важливу роль у забезпеченні комунікації, обміну інформацією та взаємодії користувачів. Одним із популярних форматів таких сервісів є блоги, які дозволяють користувачам публікувати контент, ділитися думками та отримувати зворотній зв’язок. Водночас для підтримки якості контенту та безпеки платформи необхідна система модерації та управління правами доступу.

Метою даної роботи є розробка блогу з можливістю подання постів користувачами, їх подальшою модерацією адміністраторами та автоматичною системою сповіщень про публікацію, що підвищує зручність і взаємодію користувачів із сервісом. Особлива увага приділена застосуванню сучасних технологій Elixir, Phoenix LiveView для створення інтерактивного інтерфейсу, а також використанню сервісів для автоматизованої обробки фонових задач та відправки електронної пошти. Окрім того, проект передбачає комплексне тестування функціоналу для забезпечення стабільності та якості додатку, а також розгортання на платформі Fly.io для досягнення високої доступності і масштабованості сервісу.

Ця робота має на меті продемонструвати ефективність інтеграції сучасних інструментів у створенні веб-додатків, що відповідають вимогам сучасного ринку та користувачів.

1 ОГЛЯД ТЕХНІЧНИХ ЗАСОБІВ ДЛЯ РОЗРОБКИ ВЕБ ЗАСТОСУНКУ НА ELIXIR

1.1 Основи функціонального програмування та особливості мови Elixir

1.1.1 FP в Elixir:

Elixir підтримує функціональну парадигму, що означає:

- *Незмінність даних:* значення після створення не змінюються (immutable).
- *Функції - першокласні об'єкти:* їх можна передавати як аргументи, зберігати у змінних тощо.
- *Чисті функції*: не мають побічних ефектів та залежать лише від вхідних параметрів.

Це сприяє кращій передачуваності коду, легшому тестуванню та безпечному використанню у багатопоточному середовищі.

Неперервність процесів та відсутність мутації стану:

У Elixir замість змінного стану використовуються процеси, які спілкуються між собою через обмін повідомленнями. Дані всередині процесу можна змінити лише шляхом створення нового стану (наприклад, за допомогою рекурсії або GenServer'a), а не мутацією змінних. Це дозволяє уникнути race conditions та забезпечити масштабованість системи.

1.1.2 Модель конкурентності в BEAM

Акторна модель:

BEAM реалізує акторну модель конкурентності, де кожен процес є незалежним актором з власним станом. Актори спілкуються виключно через повідомлення, що гарантує ізоляцію станів і відсутність спільної пам'яті.

Процеси, повідомлення, ізоляція:

- *Процеси BEAM* - це lightweight-процеси, що займають лише кілька кілобайтів пам’яті.
- *Ізоляція:* кожен процес повністю незалежний, аварія одного процесу не впливає на інші.
- *Повідомлення*: передаються асинхронно (черга повідомлень), без блокування.
- *Масштабованість*: BEAM може обробляти мільйони процесів одночасно.

Ця модель ідеально підходить для побудови реального часу додатків, таких як чати, де велика кількість користувачів взаємодіють одночасно.

1.1.3 GenServer та OTP

GenServer як основна абстракція:

GenServer - це поведінка (behaviour), яка надає стандартний інтерфейс для створення серверів з внутрішнім станом.

Він реалізує такі колбеки, як:

- init/1 - ініціалізація стану,
- handle_call/3 - обробка синхронних запитів,
- handle_cast/2 - обробка асинхронних повідомлень,
- handle_info/2 - обробка довільних повідомлень.

Це дозволяє легко інкапсулювати логіку, яка має зберігати стан або реагувати на події.

Supervisor та дерева нагляду в Elixir

Однією з ключових концепцій платформи Erlang/Elixir є модель нагляду (supervision), що забезпечує стійкість до збоїв (fault-tolerance) програмної системи.

Supervisor - це спеціальний процес, призначений для контролю за іншими процесами (так званими "дітьми") та автоматичного їх перезапуску у разі збою. За рахунок цього формується ієрархічна структура - дерево нагляду (supervision tree).

У системах на основі Elixir запуск програми розпочинається з ініціалізації головного супервізора, який виконує роль кореневого елемента дерева нагляду. До складу дерева входять як окремі робочі процеси (workers), так і інші супервізори, що керують підлеглими процесами нижчого рівня. Це дозволяє структурувати систему як набір незалежних компонентів, кожен з яких може бути відновлений без впливу на решту.

Завдяки цій архітектурі система має змогу:

- Автоматично відновлювати зламані частини без зупинки всієї програми;
- Забезпечувати ізоляцію збоїв - падіння одного процесу не впливає на інші;
- Підвищувати надійність і передбачуваність поведінки при виникненні помилок.

Принцип fault-tolerance та філософія "Let it crash"

Філософія розробки в Elixir/Erlang базується на підході "Let it crash" (укр. "Нехай падає"). Це означає, що процеси не повинні намагатися обробити всі можливі помилки самостійно. Натомість при виникненні критичної помилки процес завершує свою роботу, а його відновленням займається супервізор. Це дозволяє знизити складність коду, уникнути накопичення непередбачуваних станів і підвищити загальну стабільність системи.

Використання Supervisor у поєднанні з деревами нагляду та філософією "Let it crash" є фундаментальним підходом до побудови надійних, масштабованих і стійких

до збоїв програм в екосистемі Elixir. Цей підхід мінімізує вплив помилок, дозволяє будувати самовідновлювальні системи та спрощує обслуговування складних застосунків.

1.1.4 PubSub модель

Публікація/підписка в Elixir:

Модель *PubSub (publish-subscribe)* дозволяє процесам публікувати повідомлення в певну тему (topic), а інші процеси можуть підписатися на ці теми та отримувати повідомлення.

Це ефективний спосіб реалізувати реактивну логіку, наприклад:

- оновлення інтерфейсу при надходженні нових повідомлень,
- сповіщення клієнтів про зміну даних у системі.

Вбудований Phoenix.PubSub:

Фреймворк Phoenix має вбудовану систему Phoenix.PubSub, яка дозволяє:

- створювати власні теми (MyApp.PubSub.subscribe("chat:lobby")),
- надсилати повідомлення (Phoenix.PubSub.broadcast/3),
- використовувати PubSub у LiveView, Channel та будь-яких інших процесах.

Це ядро системи реального часу, що забезпечує миттєву доставку оновлень між користувачами в чаті, блогах, панелях керування тощо.

1.2 Огляд можливостей фреймворку Phoenix у розробці веб застосунків

1.2.1. Структура типового Phoenix-застосунку

Фреймворк Phoenix дотримується чіткої структури проєкту, яка сприяє модульності, повторному використанню коду та зручній підтримці застосунку. Основні компоненти типової структури:

- *Контексти (contexts)* - ізольовані модулі, які інкапсулюють бізнес-логіку певної частини системи. Наприклад, Accounts для роботи з користувачами або Chat для обробки повідомлень.
- *Схеми (schemas)* - визначають структуру таблиць у базі даних через модуль Ecto.Schema. Схеми також містять валідацію та зв’язки між сутностями.
- *Маршрути (routes)* - визначають шляхи до дій контролерів через файл router.ex. Phoenix використовує REST-підхід і дозволяє маршрутизувати як HTTP-запити, так і WebSocket-події для LiveView.

Завдяки такій структурі Phoenix-додатки легко масштабуються та підтримуються. Контексти дозволяють розділити логіку по відповідальності, а схема маршрутизації чітко розділяє запити до різних частин системи.

1.2.2. Підключення бази даних через Ecto

використовує бібліотеку Ecto для взаємодії з базами даних, зокрема PostgreSQL. Phoenix

Ecto забезпечує:

- опис схем (структура таблиць),
- проведення міграцій (зміни структури БД),
- виконання запитів (через Ecto.Query),
- асоціації між таблицями.

Міграції

Міграції у Phoenix створюються через CLI-команди (наприклад, mix Ecto.gen.migration). Вони містять функції change/0, up/0, down/0 і дозволяють вносити зміни в БД поступово й контрольовано. Приклад створення таблиці:

Зв'язки між таблицями

Ecto підтримує асоціації між моделями:

- has_many - один до багатьох (наприклад, користувач має багато повідомлень),
- belongs_to - належить до (наприклад, повідомлення належить користувачу).

Це дозволяє будувати складні зв’язки між сутностями з використанням preload, join тощо.

1.2.3. Механізми аутентифікації

Phoenix пропонує генератор автентифікації через бібліотеку phx.gen.auth, який дозволяє швидко додати повноцінну систему входу, реєстрації та скидання пароля. Система базується на сесіях (cookie-based session auth) і підтримує базові функції:

- реєстрація нового користувача;
- вхід через форму з валідацією;
- скидання пароля через email;
- захист приватних маршрутів.

Створюється окремий модуль (наприклад, MyApp.Accounts) з функціями роботи з користувачами (get_user_by_email/1, register_user/1, authenticate_user/2). Дані користувача зберігаються в сесії, і через LiveView можуть бути доступні у будь-якому компоненті.

Для розмежування доступу можна реалізувати ролі користувачів, наприклад:

- *адміністратор* - має доступ до адмін-панелі, управління статтями, надісланими повідомленнями;
- *звичайний користувач* - може переглядати контент і користуватись чатом.

1.2.4. Інтеграція LiveView у Phoenix

Phoenix LiveView - це інструмент, який дозволяє створювати динамічні, інтерактивні вебінтерфейси без потреби в JavaScript з боку клієнта. Замість того, щоб логіка виконувалась у браузері, більшість змін і обробок подій відбуваються на сервері через постійне з'єднання WebSocket.

Як працює LiveView під капотом

1. *Ініціалізація:* Коли користувач відкриває сторінку з LiveView-компонентом, Phoenix спочатку повертає статичний HTML (серверний рендеринг).
2. *Встановлення WebSocket:* Після завантаження сторінки, LiveView на клієнті встановлює WebSocket-з'єднання з сервером.
3. *Передача стану*: Сервер відправляє клієнту поточний стан компонента (у вигляді змінних), який потім LiveView застосовує до DOM.
4. *Обробка подій:* Коли користувач взаємодіє з інтерфейсом (наприклад, клікає на кнопку або змінює форму), події phx-click, phx-submit, phx-change тощо надсилаються на сервер.
5. *Оновлення DOM*: Сервер обробляє подію, оновлює стан компонента і надсилає назад лише змінені частини HTML. LiveView оновлює DOM на клієнті через диференціальний патчинг (без повного перерендерингу сторінки).

Основні можливості LiveView:

- *Реактивність:* Зміни стану на сервері автоматично оновлюють відповідні частини DOM на клієнті.
- *Становість:* Компоненти можуть зберігати стан між подіями та навіть оновленнями сторінки.
- *Безперервність:* LiveView може зберігати дані під час тимчасових розривів з'єднання та відновлювати їх при повторному підключенні.
- *Універсальність:* Підходить як для невеликих віджетів (лайків, фільтрів), так і для складних інтерфейсів, як-от чат, панелі адміністрування тощо.

Компоненти:

- *LiveView* - повноцінна жива сторінка або велика частина сторінки.
- *LiveComponent* - дрібніша частина, яку можна вбудовувати в інші LiveView або LiveComponent. Підтримує незалежне оновлення та перевикористання.

Приклади використання в проєкті:

- *Валідація форм*: поля валідуються в реальному часі при введенні без перезавантаження.
- *Чат*: нові повідомлення з'являються автоматично завдяки WebSocket-з'єднанню.
- *Фільтрація контенту*: списки оновлюються миттєво при зміні фільтрів.
- *Керування контентом в адмінці*: дозволяє змінювати дані, бачити зміни в реальному часі, а також відображати системні повідомлення (flash).

1.3 Архітектура Elixir веб застосунку

1.3.1. Загальний опис архітектури

Архітектура системи базується на мікросервісах і розділена на кілька основних модулів:

1. *Chat* - цей модуль відповідає за функціональність чат-румів. Він забезпечує можливість створювати нові чатруми, приєднуватися до існуючих, а також публікувати повідомлення. Для цього використовуються механізми реального часу, які дозволяють миттєво доставляти нові повідомлення всім підключеним користувачам.
2. *Accounts* - цей модуль здійснює управління користувачами. Він містить функції для реєстрації нових користувачів, а також для аутентифікації та авторизації. Зокрема, цей модуль дозволяє отримувати доступ до даних користувачів, таких як їх імена, електронна пошта і роль у системі.
3. *Mailer* - цей модуль призначений для відправки електронних листів користувачам. Він використовується для підтвердження реєстрації, відправлення запрошень до чат-румів та інших повідомлень, які можуть бути важливими для користувача.
4. *Blog* - це модуль для управління блогами, який дозволяє користувачам публікувати статті. Цей модуль може бути інтегрований з чат-румами для взаємодії між користувачами, наприклад, через обговорення або обмін зображеннями.

Основним механізмом взаємодії між усіма компонентами є *Phoenix.PubSub* - система публікації та підписки, що дозволяє передавати дані в реальному часі. Це означає, що користувачі, підключені до одного і того ж чатруму, отримуватимуть нові повідомлення миттєво, без необхідності перезавантаження сторінки. Для того, щоб це працювало, кожен чатрум підписується на відповідний канал, і повідомлення передаються усім підписаним клієнтам.

Система реалізована через LiveView, що дозволяє взаємодіяти з даними без потреби в JavaScript. Компоненти LiveView, такі як Sidebar для навігації та Messages для відображення повідомлень, оновлюються в реальному часі без необхідності перезавантажувати сторінку. Це робить інтерфейс зручним і швидким.

1.3.2. Робота з повідомленнями

У системі реалізовано обробку повідомлень у реальному часі за допомогою технології Phoenix.PubSub. Це дозволяє миттєво обмінюватися повідомленнями між користувачами, що перебувають в одному чатрумі.

1. *Підписка на канал:* Кожен користувач, який підключається до чату, автоматично підписується на канал через *Phoenix.PubSub*. Це дозволяє йому отримувати всі повідомлення, що надходять у цей чатрум, а також брати участь у його обговореннях.
2. *Обробка нових повідомлень*: Коли один з користувачів надсилає повідомлення в чат, система публікує це повідомлення через канал PubSub. Якщо інші користувачі знаходяться в цьому ж чатрумі, вони отримують повідомлення в реальному часі. Це означає, що немає потреби перезавантажувати сторінку - повідомлення з’являється без затримок.
3. *Оновлення компонентів*: Після того, як нове повідомлення було отримано через PubSub, відповідні компоненти інтерфейсу, такі як Messages, оновлюються автоматично. Це дозволяє користувачам бачити нові повідомлення миттєво, без необхідності вручну оновлювати сторінку. Оновлення компонентів у LiveView забезпечується через механізм send_update, який передає нові дані в компонент для оновлення.

Цей механізм взаємодії забезпечує зручність і швидкість обміну повідомленнями в реальному часі, що є основною перевагою для чат-систем.

1.3.3 Теоретичні основи маршрутизації

У сучасних веб-застосунках маршрутизація визначає, як саме HTTP-запити обробляються всередині програми, до яких контролерів вони надсилаються та які дії викликаються. У фреймворку Phoenix (Elixir) маршрути задаються у файлі router.ex і організовані через так звані пайплайни та scope-області.

Пайплайни дозволяють групувати маршрути за типом запиту: наприклад, HTML-сторінки обробляє пайплайн :browser, а JSON-запити - :api. Це дозволяє чітко розділяти маршрути для веб-інтерфейсу та API.

Також важливими концепціями є:

- використання scope для групування маршрутів за префіксом (/api, /admin тощо),
- умовний доступ до маршрутів залежно від ролі користувача (адміністратор, оператор, звичайний користувач),
- підтримка LiveView для динамічного оновлення інтерфейсу без перезавантаження сторінки,
- розмежування доступу до сторінок за допомогою аутентифікації та авторизації.

Таким чином, маршрутизація є критично важливою частиною додатку, яка напряму впливає на зручність використання та безпеку системи.

1.3.4 Обробка фонових задач через Oban

У цьому додатку для обробки фонових задач використовується бібліотека Oban. Oban є потужним інструментом для обробки асинхронних задач у фоновому режимі, таких як надсилання електронних листів, планування задач та інші довготривалі процеси.

Надсилання листів

Однією з задач, яку обробляє Oban, є надсилання електронних листів користувачам. Коли користувач публікує пост на платформі, йому автоматично відправляється підтвердження на пошту про успішне публікування. Для цього створено фонову задачу, яка виконується через Oban.

Планування задач

Oban забезпечує надійне планування задач, таких як повторні спроби надсилання листів. У випадку, якщо лист не вдалося надіслати через тимчасову проблему з мережею або API поштового сервісу, Oban автоматично спробує надіслати лист знову, використовуючи механізм повторних спроб. Це дозволяє зменшити ймовірність того, що користувач не отримає повідомлення.

Обробка задач через Oban включає такі етапи:

1. Коли пост публікується, в системі створюється джоб (завдання) для надсилання листа.
2. Це завдання ставиться в чергу на виконання.
3. Через обробник фонових задач Oban, лист надсилається через Mailjet API.
4. Якщо з якихось причин лист не вдалося надіслати, Oban буде автоматично намагатись відправити лист ще кілька разів відповідно до налаштувань для повторних спроб.

Використання Oban для фонових задач дозволяє ефективно обробляти великий обсяг запитів, не блокуючи основний процес додатку.

1.3.5 Адмін-панель

Адміністративна панель в додатку надає спеціальні можливості для управління контентом та користувачами. Адміністратори можуть виконувати важливі операції, які вимагають підвищених прав доступу.

Доступ лише для адміністраторів

Адміністратори мають спеціальний доступ до цього розділу додатку. Інші користувачі без відповідних прав не можуть переглядати або змінювати дані в адміністративній панелі. Для цього реалізовано механізм контролю доступу, що перевіряє роль користувача перед наданням доступу до адмін-панелі.

Можливість додавання зображень до статей

Адміністратори можуть додавати зображення до статей через інтерфейс адмін-панелі. Це дозволяє покращити вигляд контенту та зробити його більш привабливим для користувачів.

Можливість давати ролі іншим користувачам

В адмін-панелі є функціонал для присвоєння ролей користувачам. Адміністратори можуть змінювати ролі користувачів, надаючи їм доступ до певних функцій платформи. Це дозволяє гнучко налаштовувати доступ користувачів до різних частин додатку.

Можливість переглядати інформацію по іншим користувачам

Адміністратори мають доступ до повної інформації про інших користувачів. Це включає перегляд профілів, історії активності та інших даних, що допомагає адміністратору ефективно управляти користувачами платформи.

1.4 Використання Phoenix LiveView для створення динамічного інтерфейсу

1.4.1. Принципи роботи LiveView

Phoenix LiveView забезпечує можливість створювати інтерактивні інтерфейси, при цьому логіка та оновлення відбуваються повністю на стороні сервера. Замість JavaScript'у - HTML і події оновлюються через WebSocket.

Основні принципи:

- *WebSocket-з'єднання*: LiveView встановлює постійне з'єднання з сервером після завантаження сторінки. Це дозволяє передавати події, обробляти їх на сервері та повертати лише змінені частини DOM.
- *Без JavaScript (майже):* Більшість динаміки - без власного JavaScript. Прості дії, як-от перевірка форми, взаємодія з кнопками, списками, вкладками, реалізуються лише з HTML-атрибутами типу phx-click, phx-change, phx-submit.
- *Серверна реактивність*: Зміни в стані на сервері автоматично оновлюють HTML на клієнті через "диференційовані патчі", без перезавантаження сторінки чи перерендеру всього DOM.

Життєвий цикл LiveView:

1. Користувач відкриває сторінку → отримує статичний HTML.
2. Автоматично встановлюється WebSocket-з'єднання.
3. Клієнтський LiveView отримує стан з сервера.
4. При події (наприклад, кліку чи введенні) вона надсилається на сервер.
5. Сервер змінює стан та надсилає лише оновлення DOM.
6. LiveView оновлює інтерфейс без повного перерендерингу.

Переваги:

- Не потрібно писати JavaScript для більшості динамічних сценаріїв.
- Підтримка масштабованості через розподілені PubSub.

- Просте використання стану між оновленнями компонента.
- Надійне відновлення після переривання WebSocket-з’єднання.

1.4.2. Використання LiveView у чаті

У реалізованому чаті LiveView відіграє ключову роль для забезпечення динамічності та "живого" інтерфейсу, особливо без складного фронтенду.

Реалізовані можливості:

- *Оновлення повідомлень в реальному часі*
  При створенні нового повідомлення, воно миттєво відображається у всіх підключених користувачів завдяки комбінації Phoenix.PubSub та LiveView. Компонент чату підписується на тему (наприклад, chat:room_id) і реагує на надходження нових повідомлень.
- *Валідація форм*
  Повідомлення перевіряється ще до надсилання. Наприклад, перевірка на порожнє поле (validate_required) - миттєва, без оновлення сторінки. Валідація реалізована через phx-change, яка викликає callback на сервері й повертає зміни у вигляді помилок для LiveView-шаблону.
- *Інтерактивність без перезавантаження*
  Повідомлення додаються, видаляються, фільтруються або групуються в режимі реального часу без жодного перезавантаження. Також, користувачі бачать повідомлення одне одного одразу, навіть без оновлення сторінки.
- *Обробка подій у LiveView*
  Події типу phx-submit чи phx-click обробляються у handle_event/3. Наприклад, при надсиланні повідомлення викликається подія, яка:
  1. Створює нове повідомлення в базі.

2. Публікує його через PubSub.
3. Вставляє HTML повідомлення на сторінку.

- *Сортування або фільтрація повідомлень (опціонально)*

Якщо чат підтримує сортування за датою або фільтрацію по ключовому слову - це теж реалізовано через LiveView без окремих запитів чи перезавантажень.

2 ДОСЛІДЖЕННЯ ПІДХОДІВ ДО ПРОЄКТУВАННЯ ВЕБ ЗАСТОСУНКУ

2.1. Аналіз підходів до побудови систем з розмежуванням прав доступу

2.1.1. Необхідність розмежування доступу

У будь-якій інформаційній системі, що має кілька типів користувачів, важливо організувати контроль доступу до функціоналу відповідно до ролі користувача. Це забезпечує:

- *Безпеку даних* – лише уповноважені особи можуть переглядати або змінювати критичну інформацію.
- *Зручність використання* – користувачі бачать лише ті функції, які їм дійсно потрібні.
- *Масштабованість* – легше додавати нові ролі та повноваження в майбутньому.

2.2.2. Ролі користувачів у системі

У рамках розробленої системи було впроваджено кілька ролей користувачів:

- *Адміністратор* – має повний доступ до всієї системи: керування користувачами, редагування даних, перегляд аналітики тощо.
- *Оператор* – має доступ до обмеженого адміністративного функціоналу, наприклад, модерації контенту чи підтвердження дій.
- *Звичайний користувач* – має доступ лише до основних функцій системи: перегляд власної інформації, використання загальнодоступного функціоналу.
- *Гість* – може переглядати лише частину публічної інформації без можливості змін.

2.2.3. Реалізація ролей у системі

Реалізацію було здійснено за допомогою:

- *Рольової моделі*: кожному користувачеві призначається роль (role), яка визначає рівень доступу.
- *Фільтрації доступу на рівні контролерів* (у випадку з Elixir – через plug або Phoenix.Controller):

Система розмежування прав доступу є критично важливою частиною розробленої платформи. Вона забезпечує безпечне, ефективне та кероване використання програмного продукту. Рольова модель дозволяє чітко контролювати дії кожного типу користувача та забезпечує основу для подальшого масштабування функціоналу.

2.2 Архітектурні рішення для інтеграції чату та блогу в одному застосунку

Інтеграція кількох функціональних модулів у межах одного застосунку завжди вимагає ретельного архітектурного планування. У випадку даного проєкту було прийнято рішення поєднати дві ключові частини - чат та блог - в одному середовищі, забезпечуючи при цьому незалежність кожного з модулів і водночас їхню тісну взаємодію на рівні інтерфейсу та бази даних.

Для цього було обрано модульну архітектуру з чітким поділом на контексти. Контекст чату відповідає за створення, збереження та відображення повідомлень і кімнат, тоді як контекст блогу - за управління публікаціями користувачів, включаючи створення, модерацію та відображення постів. Такий підхід дозволяє розробникам працювати над окремими частинами системи незалежно, не створюючи взаємних конфліктів, та дає змогу у майбутньому масштабувати систему, додаючи нові модулі, не змінюючи існуючу логіку.

Крім того, у рамках єдиної архітектури обидва модулі розміщуються у спільному веб-інтерфейсі, що забезпечує цілісне користувацьке враження. Загальні компоненти, такі як автентифікація, авторизація, управління користувачами, були винесені в окремий контекст, який обслуговує як чат, так і блог. Це дозволяє повторно використовувати код, зменшує дублювання та спрощує супровід.

Комунікація між модулями відбувається на рівні подій - наприклад, якщо користувач створює новий пост у блозі, інші частини системи можуть бути повідомлені про це через систему публікацій і підписок, що створює умови для подальшої інтеграції функціональностей, таких як нотифікації або відображення активності користувачів.

Таким чином, архітектурне рішення базується на принципах розділення відповідальності (Separation of Concerns), повторного використання коду (Code Reusability) та масштабованості (Scalability), що в сукупності сприяє ефективній реалізації проєкту з багатьма інтерактивними модулями.

2.3 Вибір технологій для реалізації користувацької взаємодії в режимі реального часу

Важливою особливістю розроблюваного застосунку є високий рівень інтерактивності. Основною вимогою було забезпечити користувачам можливість взаємодіяти з системою в режимі реального часу - зокрема, це стосується як миттєвого отримання нових повідомлень у чаті, так і динамічного оновлення інтерфейсу без потреби ручного перезавантаження сторінки.

Для досягнення цього було обрано бібліотеку Phoenix LiveView, яка надає зручний інструментарій для створення реактивних інтерфейсів на основі WebSocket-з'єднань. На відміну від традиційних підходів із застосуванням сторонніх JavaScript-фреймворків (таких як React чи Vue), LiveView дозволяє реалізувати інтерактивність,

залишаючи логіку управління станом на сервері. Це не лише спрощує архітектуру, а й зменшує обсяг клієнтського коду, що підвищує безпеку та полегшує супровід.

Завдяки LiveView стало можливим реалізувати наступні елементи взаємодії в реальному часі:

- динамічне оновлення повідомлень у чаті при надходженні нових даних;
- плавне перемикання між чат-кімнатами без перезавантаження сторінки;
- обробка подій на стороні клієнта (наприклад, натискання кнопок або введення тексту) з миттєвим відгуком з боку сервера;
- оновлення частин інтерфейсу без зміни загального DOM-дерева, що дозволяє економити ресурси.

Окрім LiveView, для організації передачі даних у режимі реального часу використовується вбудована в Phoenix система публікацій і підписок - Phoenix PubSub. Вона дозволяє налагодити масштабовану систему повідомлень, у якій сервер транслює події на відповідні топіки (наприклад, "chatrooms:123"), а підписані на них клієнти автоматично отримують актуальні дані.

Завдяки цим інструментам було досягнуто головної мети - забезпечення безперебійної та швидкої взаємодії користувачів з інтерфейсом платформи, що особливо важливо для чат-функціоналу. А в майбутньому ці ж технології можуть бути використані й у блозі, наприклад, для реалізації функціоналу коментування постів у реальному часі чи сповіщення про нові публікації.

3 РЕАЛІЗАЦІЯ ЗАСТОСУНКУ З ОНЛАЙН ЧАТОМ

3.1 Реалізація базової структури чату з використанням Phoenix

3.1.1 Реалізація функціональності чату

Функціональність чату реалізована з використанням бібліотеки Phoenix LiveView, яка дозволяє створювати інтерактивні інтерфейси без використання зовнішніх JavaScript-фреймворків. LiveView забезпечує постійне з'єднання з сервером через WebSocket, що дозволяє динамічно оновлювати інтерфейс користувача у відповідь на серверні події.

Ключовою сторінкою є ChatRoomPage, яка завантажується як LiveView-компонент і містить два підкомпоненти: бічну панель із доступними кімнатами (Sidebar) та область перегляду повідомлень (Messages). Обидва компоненти є LiveComponent'ами, які можуть оновлюватися незалежно один від одного.

Також реалізовано обробку подій через функцію handle_event, яка реагує на вибір кімнати чи створення нової, а також підписку на повідомлення за допомогою Phoenix.PubSub. Коли користувач відкриває сторінку чату, застосунок автоматично підписується на тему "chatrooms", що дозволяє отримувати нові повідомлення в реальному часі.

3.1.2 Вибір і створення чат-кімнат

У межах чату користувач має змогу вибрати існуючу чат-кімнату або створити нову. Для цього реалізовано механізм обробки подій через LiveView. При створенні нової кімнати використовується псевдовипадкове ім’я, яке генерується за допомогою бібліотеки Faker, а також фіксований опис. Нова кімната прив’язується до користувача, що її створив.

Вибір кімнати здійснюється через обробку події "enter-chatroom", в якій у сокет збережується current_chatroom_id. Це дозволяє компоненту повідомлень (Messages) оновлювати відображення відповідно до вибраної кімнати.

3.1.3 Передача та відображення повідомлень

Передача повідомлень між користувачами реалізована через систему публікацій та підписок Phoenix PubSub. Коли надходить нове повідомлення, сервер транслює його на відповідну тему (наприклад, "chatrooms"), після чого LiveView перевіряє, чи поточна чат-кімната збігається з тією, що зазначена в повідомленні. Якщо збігається - повідомлення передається у компонент Messages через send_update.

Це дозволяє уникнути оновлення всієї сторінки або повторної завантаження повідомлень - оновлюється лише потрібна частина інтерфейсу. Такий підхід забезпечує високу інтерактивність та ефективність роботи чату.

3.1.4 Приєднання до чат-кімнати через інвайт-код

Для забезпечення приватності та гнучкості в організації комунікації реалізовано можливість приєднання до чат-кімнати за допомогою інвайт-коду. Ця функція обробляється окремим LiveView-компонентом - JoinChatRoomView.

Після переходу користувача за спеціальним посиланням з інвайт-кодом система виконує пошук кімнати за кодом, а потім додає поточного користувача до списку учасників цієї кімнати. Успішне приєднання супроводжується повідомленням у flash-повідомленні та автоматичним перенаправленням на основну сторінку чату.

3.2 Реалізація блогу з можливістю модерації публікацій користувачів

3.2.1. Форма подачі постів користувачами

У межах платформи був реалізований окремий функціональний модуль - блог. Цей розділ призначений для опублікування статей, оголошень, новин та інших матеріалів, що можуть бути цікаві всім користувачам системи. Особливістю реалізованого блогу є можливість створення постів не лише адміністраторами, а й звичайними користувачами. При цьому публікація відбувається лише після модерації. Такий підхід дозволяє зберегти контроль над контентом, забезпечити його відповідність правилам спільноти та водночас заохотити активну участь користувачів у житті платформи.

Інтерфейс користувача містить розділ "Запропонувати публікацію", який доступний кожному зареєстрованому користувачу. У цій формі користувач заповнює поля:

- Назва публікації (обов’язкове)
- Текстовий зміст (обов’язковий)
- Теги (необов’язкові)
- Зображення (за потреби)

Після натискання кнопки "Надіслати", публікація не стає відразу доступною для всіх. Вона зберігається у базі даних зі статусом pending, а дата подачі фіксується автоматично. Усі пости, що перебувають у цьому стані, недоступні для публічного перегляду.

3.2.2. Адміністративна модерація публікацій

У адміністративній панелі системи реалізований окремий інтерфейс для перегляду усіх користувацьких пропозицій постів. Для кожного з них вказується:

- Автор (тобто користувач, який подав публікацію)

- Дата створення
- Заголовок і короткий опис
- Поточний статус

Модератор може ознайомитися з повним текстом публікації та прийняти рішення про її подальшу долю. При натисканні кнопки "Схвалити", статус поста змінюється на published, і він стає доступним для всіх користувачів на головній сторінці блогу. Якщо публікація не відповідає вимогам або має сумнівний зміст, модератор може натиснути "Відхилити", що залишає її у статусі rejected або видаляє остаточно, залежно від реалізованої політики.

Процес модерації є ключовим етапом у забезпеченні якості контенту та уникненні спаму або порушення правил користування платформою.

3.2.3. Надсилання сповіщень користувачам про статус публікації

У разі схвалення запропонованого поста, система автоматично надсилає листа на електронну пошту користувача, під якою він був авторизований у момент створення публікації. Повідомлення містить підтвердження публікації та посилання на опублікований матеріал.

Цей функціонал реалізовано за допомогою зовнішнього сервісу Mailjet, що забезпечує надійне й масштабоване надсилання електронних листів. З метою уникнення затримок у відповіді на користувацькі дії, надсилання відбувається у фоновому режимі за допомогою бібліотеки Oban. Oban дозволяє ставити задачі у чергу, повторювати їх у разі помилок, логувати результати та налаштовувати пріоритети.

У модулі надсилання формується лист у такому вигляді:

- Адресат: електронна пошта користувача

- Тема: "Ваш пост успішно опубліковано"
- Тіло листа: коротке повідомлення про успішну публікацію

Така інтеграція дозволяє значно покращити зворотний зв’язок із користувачами та підвищити рівень залученості.

3.2.4. Відображення публікацій у відкритій частині платформи

Після підтвердження поста модератором і зміни його статусу на published, публікація автоматично з’являється у загальному списку постів на головній сторінці блогу. Там реалізовано пагінацію, сортування за датою та пошук за назвою або тегами. Кожен пост можна відкрити для перегляду повного тексту.

Для неавторизованих користувачів доступна функція читання публікацій, однак створювати нові або залишати коментарі (якщо така функція активна) можуть лише зареєстровані користувачі. Це дозволяє забезпечити відкритість блогу для перегляду, зберігаючи при цьому контроль над внеском контенту.

3.3 Впровадження системи автентифікації, авторизації та маршрутизації

У процесі розробки було реалізовано чітко структуровану маршрутизацію, яка тісно інтегрована з системами аутентифікації та авторизації. Нижче детально описано, як це було реалізовано:

Пайплайни

Для обробки різних типів запитів у файлі router.ex було створено два основні пайплайни:

- :browser - для HTML-запитів, із сесіями, CSRF-захистом та макетом сторінки.
- :api - для обробки JSON-запитів, що використовується при роботі з API.

API-маршрути

У межах scope "/api" реалізовано всі основні CRUD-операції для ресурсів:

- пости, зображення, користувачі, ролі, сайти - з підтримкою create, index, show, update, delete;
- спеціальний маршрут /pictures/by_post/:post_id дозволяє отримати зображення для конкретного поста.

Маршрути веб-сторінок (LiveView)

Було додано ряд сторінок на LiveView, що забезпечують реальний час без перезавантаження:

- Головна (/), меню (/menu), вкладки (/tabs), акордеон (/Accordion);
- /contribution - сторінка для внесення даних.

Сторінки з обмеженим доступом

Доступ до деяких сторінок обмежено залежно від автентифікації:

- /users - доступна тільки автентифікованим користувачам;
- /permission_error - показується при спробі доступу без прав;
- /posts, /posts/new, /posts/:id/edit, /chatroom - лише для користувачів, які успішно увійшли в систему.

Адміністративні маршрути

Для адміністраторів і операторів передбачено окремі області:

- /admin - головна панель адміністрування;
- /Account-managers, /policy - тільки для супер-адміністраторів;
- /site-configs, /roles - для операторів.

Маршрути для розробки

Для потреб розробки додано:

- /dashboard - LiveDashboard для моніторингу;
- /mailbox - перегляд пошти через Plug.Swoosh.MailboxPreview.

Маршрути автентифікації

Для нових або неавторизованих користувачів:

- /users/register, /users/log_in - реєстрація та вхід;
- /users/reset_password - відновлення пароля.

3.4. Тестування основного функціоналу застосунку

Одним з важливих етапів розробки програмного забезпечення є тестування. Воно дозволяє перевірити працездатність окремих частин системи, уникнути помилок ще на етапі розробки та забезпечити стабільну роботу програми після оновлень. У рамках цього проєкту було реалізовано модульне тестування бекенд-частини, яка написана на мові програмування Elixir з використанням фреймворку Phoenix, а також інтеграційне тестування для інтерфейсу, реалізованого за допомогою Phoenix LiveView.

Модульне тестування на Elixir

У Phoenix-проєктах для тестування за замовчуванням використовується вбудований інструмент ExUnit. Це потужна система модульного тестування, що дозволяє створювати окремі тести для функцій, контролерів, API, контекстів та іншої логіки.

Також використовуються додаткові бібліотеки, такі як:

- Phoenix.ConnTest - для тестування HTTP-запитів та відповідей контролерів;
- Phoenix.LiveViewTest - для LiveView інтерфейсів;

- Ecto.Adapters.SQL.Sandbox - для ізоляції роботи з базою даних у кожному тесті.

Тестування охоплює такі ключові частини бекенду:

- *Контексти* - модулі, які безпосередньо взаємодіють із базою даних PostgreSQL. Було протестовано основні CRUD-операції (створення, читання, оновлення, видалення), перевірка валідності полів, робота з асоціаціями, а також спеціальні фільтри та логіка, реалізована в рамках проєкту.
- *Контролери API* - для кожного RESTful контролера (наприклад, для PostController, UserController, RoleController) були написані тести, що перевіряють відповідність HTTP-відповідей очікуваним результатам. Це включає: перевірку списку ресурсів (GET), створення нового ресурсу (POST), оновлення існуючого ресурсу (PUT/PATCH), видалення (DELETE), обробку невалідних даних (400, 404).

Окрему увагу було приділено перевірці помилок, таких як спроба оновлення або видалення неіснуючого ресурсу, неправильні поля, або порожні значення. Також перевірялась відповідність структури JSON-даних у відповідях.

Інтеграційне тестування LiveView інтерфейсу

Phoenix LiveView дозволяє створювати інтерактивні веб-сторінки без використання JavaScript. У цьому проєкті значна частина інтерфейсу - зокрема адмін-панель - реалізована саме за допомогою LiveView.

Для перевірки таких сторінок були реалізовані LiveView-тести. Вони емулюють поведінку реального користувача у браузері та дозволяють перевірити:

- чи коректно відображається контент на сторінці;
- чи працює фільтрація, сортування та динамічне оновлення вмісту;

- як поводиться інтерфейс при взаємодії з елементами (випадаючими списками, формами, кнопками);
- чи спрацьовують повідомлення про помилки або успішні дії (наприклад, підтвердження видалення);
- як поводиться система при переключенні станів (відкриття/закриття модальних вікон, зміна ролей тощо).

LiveView тести охоплюють різні сторінки проєкту - від списку користувачів до форм редагування, керування ролями, фільтрації за умовами, перевірки вмісту таблиць тощо. Завдяки цьому вдається досягти впевненості у правильній роботі інтерфейсу без використання браузерних тестів.

Підхід до написання тестів

Усі тести організовані за принципами:

- ізоляції - кожен тест працює незалежно від інших;
- повторюваності - результати тестів завжди однакові;
- швидкості - за рахунок використання Ecto.Sandbox тести з базою даних працюють без потреби очищення вручну;
- наочності - кожен модуль тестується окремо з чіткими перевірками очікуваних результатів.

Всі тести регулярно проганяються під час локальної розробки, а також можуть бути інтегровані в CI/CD процеси (наприклад, за допомогою GitHub Actions або GitLab CI).

3.5 Деплой додатку на fly.io

Для деплою веб-додатку, написаного на Elixir з фреймворком Phoenix, було обрано платформу Fly.io. Fly.io - це хмарна платформа, яка дозволяє легко розгортати додатки по всьому світу, забезпечуючи низьку затримку та масштабованість.

Переваги Fly.io для Elixir-проєкту

- *Проста інтеграція з Elixir/Phoenix* - Fly.io підтримує контейнеризацію додатків та дозволяє швидко запускати їх у глобально розподіленому середовищі.
- *Автоматичне масштабування* - платформа автоматично регулює ресурси залежно від навантаження.
- *Налаштовуване розташування серверів* - можна вибирати регіони для розгортання, що підвищує швидкодію для користувачів.
- *Вбудована підтримка баз даних PostgreSQL* - Fly.io пропонує керовані бази даних, що спрощує управління сховищем даних.

Процес деплою

1. *Підготовка додатку*
   У проєкті налаштовано Dockerfile для збірки Elixir/Phoenix додатку. Важливо, щоб додаток міг запускатися у контейнері з усіма залежностями.
2. *Реєстрація та налаштування Fly.io*
   Спочатку створюється обліковий запис на fly.io. За допомогою CLI інструментів виконуються команди для ініціалізації проєкту (fly launch), вибору регіону і створення додатку у Fly.io.
3. *Налаштування бази даних*
   Для роботи з PostgreSQL на Fly.io можна скористатися їхньою керованою базою або підключитися до зовнішньої бази даних. У конфігурації додатку вказуються відповідні параметри підключення.

4. *Розгортання*

   Виконується команда fly deploy, яка збирає контейнер і завантажує його на платформу. Після успішного деплою додаток автоматично запускається у вибраному регіоні.

5. *Моніторинг та підтримка*

   Fly.io надає інструменти для моніторингу стану додатку, логів, а також можливість масштабування вручну або автоматично.

Особливості налаштувань

- *Змінні середовища* - секрети (наприклад, ключі API для Mailjet, налаштування бази даних) зберігаються у змінних оточення Fly.io, що підвищує безпеку.
- *Міграції бази даних* - перед запуском додатку на сервері виконується застосування міграцій Ecto для створення необхідних таблиць та оновлення схеми.
- *Обробка фонових задач* - такі сервіси як Oban для обробки фонових джобів також коректно працюють у середовищі Fly.io.

3.6 Огляд веб застосунку з інтегрованим онлайн чатом та блогом

Рисунок 3.6.1: Головна сторінка веб-застосунку

На рисунку 3.6.1 зображено головну сторінку мого веб-застосунку, яка є центральним пунктом для доступу до всіх основних функцій. Не зареєстровний користувач має доступ тільки до цієї сторінки. Сторінка має сучасний та мінімалістичний дизайн, що забезпечує зручність користування. У верхній частині розташоване головне меню з наступними пунктами:

- Home - повертає на головну сторінку.
- Users Page - сторінка для управління користувачами.
- ChatRoom - доступ до онлайн-чату.
- Blog - перехід до блогу.
- Register/Log in - реєстрація або вхід для зареєстрованих користувачів.

У центральній частині сторінки розміщено заголовок "This is Potrap's Project" та блоки з описом основних функцій застосунку:

1. Chat Room - інтерактивний чат для спілкування в реальному часі. Користувачі можуть обмінюватися повідомленнями, ідеями та новинами з друзями, родиною або колегами. Чат забезпечує безпеку та зручність використання.
2. Admin Page - інструмент для адміністраторів, який дозволяє керувати користувачами, контентом та налаштуваннями сайту. Ця функція надає повний контроль над веб-застосунком.
3. Blog Page - платформа для публікації статей, думок та оновлень. Блог підходить як для професійних авторів, так і для початківців, завдяки зручному інтерфейсу та можливостям кастомізації.

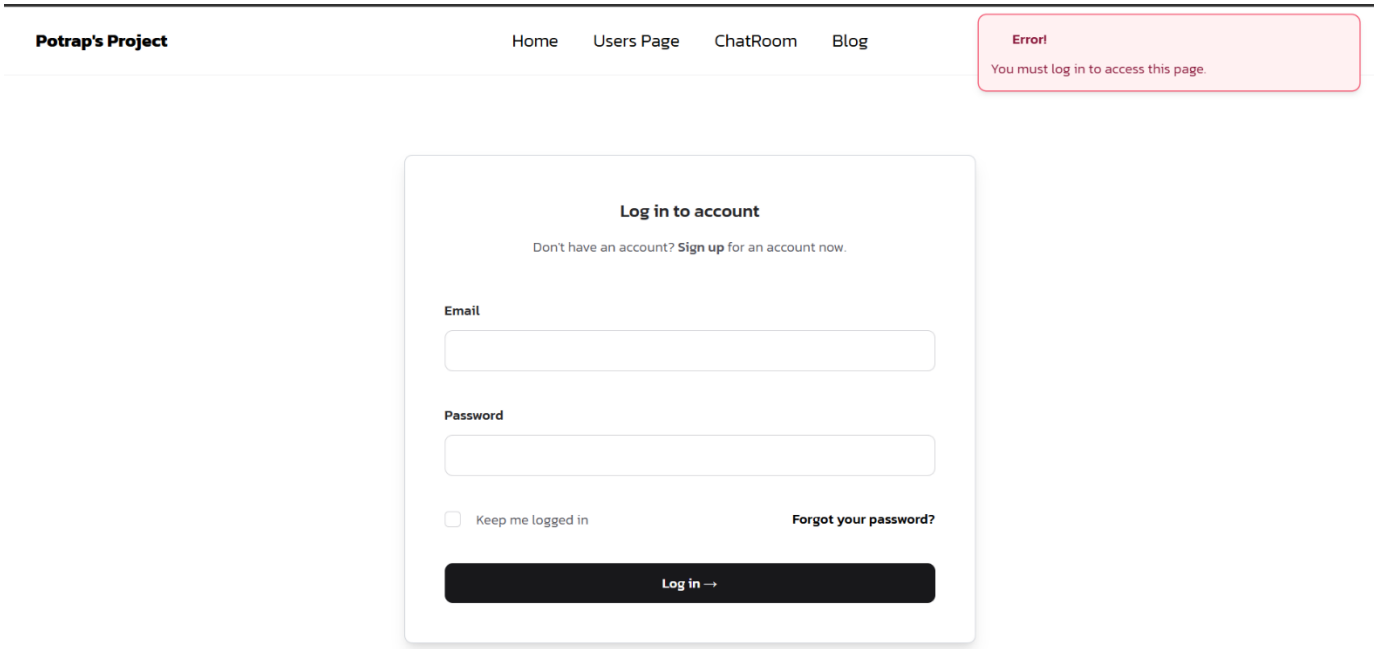

Рисунок 3.6.2: Сторінка авторизації веб-застосунку

На рисунку 10.2 зображено сторінку авторизації, яка з’являється при спробі перейти на будь-яку сторінку веб-застосунку (окрім головної). Система виводить повідомлення "Error! You must log in to Access this page", після чого пропонує користувачу увійти в обліковий запис або зареєструватися.

Основні елементи сторінки:

- Форма входу з полями для введення електронної пошти та пароля.
- Опція "Keep me logged in" для зручного продовження сеансу.
- Кнопка "Login" для підтвердження авторизації.
- Посилання на реєстрацію ("Sign up for an Account now").
- Посилання на відновлення пароля ("Forget your password?").

Сторінка забезпечує захист даних, обмежуючи доступ до функцій застосунку (чату, блогу, панелі адміністратора) лише для авторизованих користувачів. Простий інтерфейс дозволяє швидко увійти або створити новий обліковий запис.

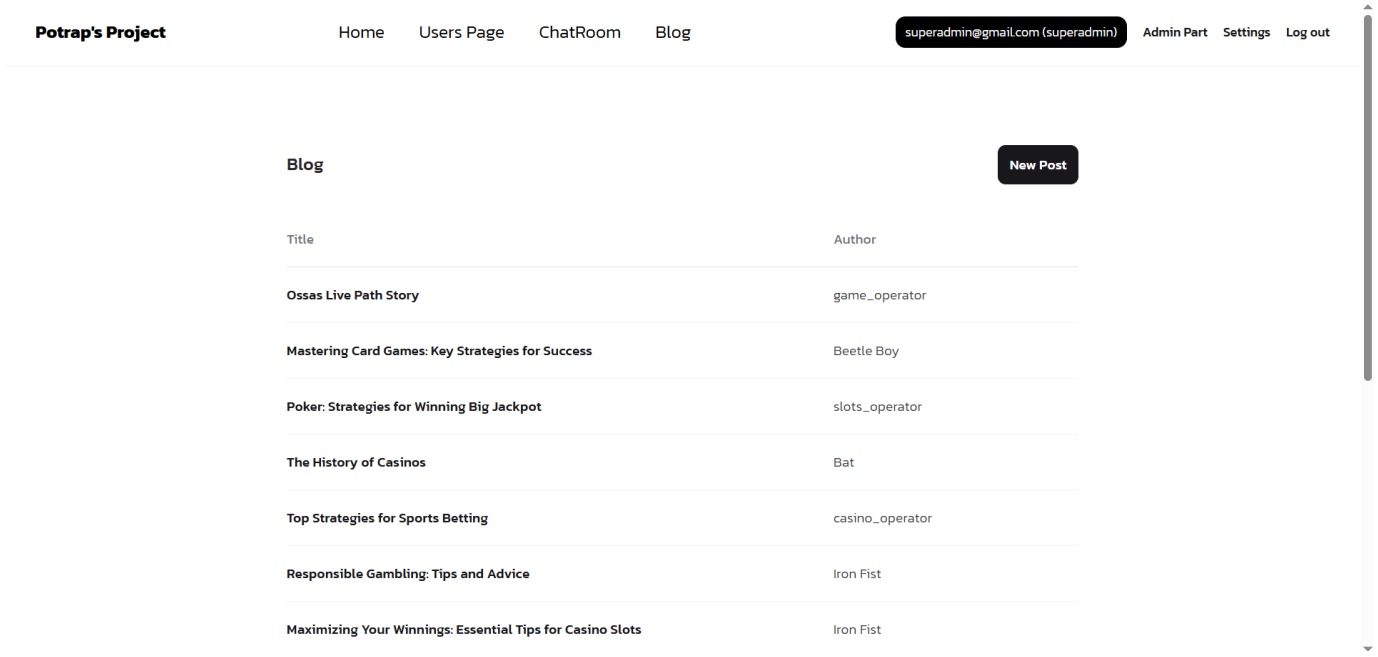

Рисунок 3.6.3: Сторінка блогу веб-застосунку

На рисунку 3.6.3 показано сторінку блогу, де відображаються публікації. Користувачі можуть створювати нові пости, які потім проходять модерацію через адмін-панель. Після підтвердження адміністратором автор отримує повідомлення на електронну пошту про успішну публікацію.

Ключові елементи:

- Список опублікованих статей
- Розділ "New Posts" для нових дописів, що очікують перевірки.
- Панель адміністратора (доступна для обліковки superadmin).

Функціонал:

- Автори подають пости через інтерфейс блогу.
- Адміністратор перевіряє та підтверджує публікацію.
- Автор отримує email-сповіщення про статус поста.

Рисунок 3.6.4: Приклад створеного посту в блозі

На рисунку 3.6.4 зображено сторінку з повним текстом публікованого посту в блозі веб-застосунку. Пост містить заголовок, основний текст, ім’я автора (Beatle Boy) та теги (Cards).

Ключові елементи:

- Заголовок: "Mastering Card Games: Key Strategies for Success".

- Контент: Детальний опис стратегій для карткових ігор (покер, блекджек), включаючи аналіз ймовірностей, позиціювання та психологічні аспекти (блеф).
- Автор та теги: Вказано для організації та пошуку.

Функціонал:

- Користувачі можуть читати готові пости.
- Автори (як Beatle Boy) публікують власні матеріали після модерації.
- Теги допомагають у категорізації контенту.

Пост є прикладом інформативного матеріалу, який може бути створений у системі.

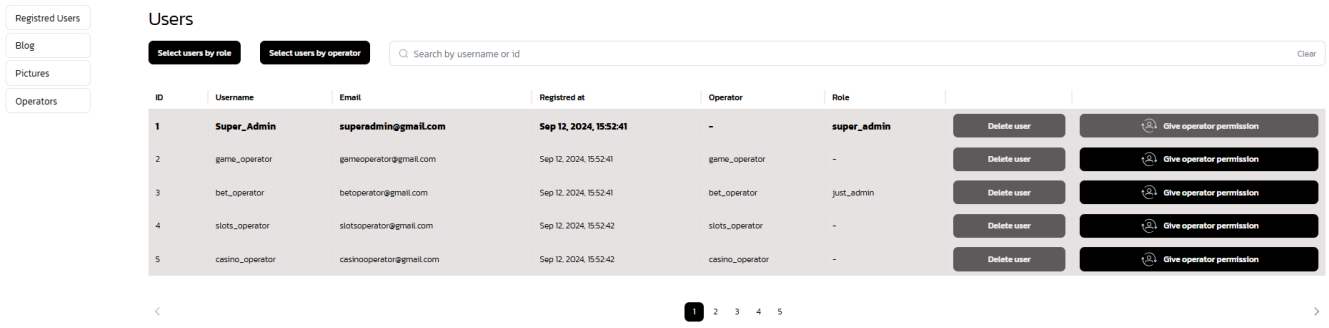

Рисунок 3.6.5: Панель адміністратора - управління користувачами

Сторінка на рисунку 3.6.5 відображає інструменти адміністратора для керування користувачами веб-застосунку. Вона містить таблицю з переліком усіх зареєстрованих користувачів та їх основними даними.

Ключові функції:

- Перегляд списку користувачів з детальною інформацією (ID, ім'я, email, роль)
- Пошук користувачів за іменем або ID

- Можливість змінювати ролі користувачів (наприклад, надавати права адміністратора)
- Фільтрація користувачів за ролями (super_admin, just_admin тощо)

Особливості:

- Система відображає час останньої активності користувача ("Reptime of")
- Можна ідентифікувати операторів за допомогою поля "Operator"
- Простий інтерфейс для швидкого управління правами доступу

Ця панель дає адміністраторам повний контроль над користувацькою базою, забезпечуючи безпеку та ефективне управління системою.

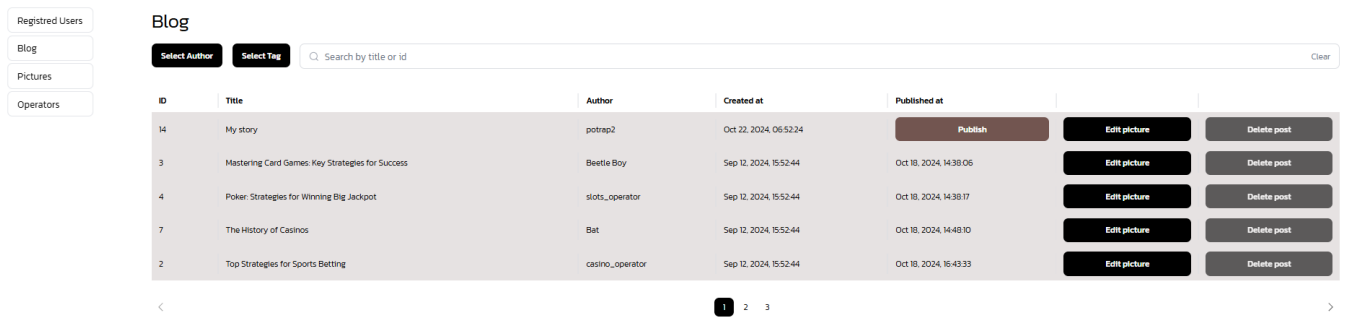

Рисунок 3.6.6: Панель адміністратора - керування блогом

Сторінка на рисунку 3.6.6 адміністратора для управління контентом блогу. Містить таблицю з усіма постами (опублікованими та очікуваними), інструменти для їх редагування та модерації.

Функціонал адміна:

- Перегляд списку постів (ID, заголовок, автор, дата створення)
- Фільтрація (пошук за назвою або ID)

- Редагування (зміна тексту, додавання зображень)
- Публікація/видалення (кнопки "Publish" та "Delete post")
- Контакт з авторами (через "Email address")

Особливості:

- Пости очікують підтвердження адміна перед публікацією.
- Адмін може швидко знайти пост через пошук або фільтрацію.
- Простий інтерфейс для масованих дій (наприклад, видалення кількох постів).

Ця панель забезпечує повний контроль над контентом блогу, дозволяючи підтримувати якість та актуальність публікацій.

ВИСНОВОК

У ході виконання дипломної роботи на тему «Розробка блогу з модерацією постів, системою повідомлень та деплоєм на Fly.io» було реалізовано повнофункціональний блог, який дозволяє користувачам пропонувати нові пости, а модераторам - перевіряти їх та публікувати після затвердження. Для покращення взаємодії з користувачами впроваджено систему сповіщень, яка надсилає листи на електронну пошту автора поста після його публікації, використовуючи інтеграцію з Mailjet та фонову обробку через Oban.

Значну увагу було приділено якісному тестуванню додатку: написано модульні тести для контекстів, що працюють з базою даних PostgreSQL, а також проведено тестування API контролерів для перевірки коректності CRUD операцій з постами. Крім того, було створено фронтенд-тести для LiveView сторінок, які забезпечують правильне відображення даних та роботу інтерактивних елементів. Для розгортання проекту обрана платформа Fly.io, що дозволяє забезпечити стабільне, масштабоване та безпечне середовище з керованою базою даних та налаштованими змінними середовища.

Використання Elixir і Phoenix LiveView надало можливість створити високопродуктивний та зручний у підтримці веб-додаток, а застосування сучасних інструментів і технологій забезпечило гнучкість розвитку проекту та високу якість кінцевого продукту. Таким чином, реалізований проект підтвердив ефективність поєднання сучасних підходів у веб-розробці для створення надійних і масштабованих сервісів.