**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ**
**Кафедра програмної інженерії**

# КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ОНЛАЙН-ПЛАТФОРМИ ДЛЯ ВЕДЕННЯ БЛОГІВ З ВИКОРИСТАННЯМ REACT ТА NODE.JS»**

Виконав: студент 4 курсу, групи 6.1211-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
Б.О. Онопрієнко
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Кривохата А.Г.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

**ЗАТВЕРДЖУЮ**
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

# З А В Д А Н Н Я

## НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Онопрієнку Богдану Олександровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка онлайн-платформи для ведення блогів з використанням React та Node.Js

керівник роботи Кривохата Анастасія Григорівна, к.ф.-м.н.
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Проєктування застосунку.
3. Програмна реалізація онлайн-платформи.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання 26.12.2024 р.

## К А Л Е Н Д А Р Н И Й  П Л А Н

| № | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | Розробка плану роботи. | 13.01.2025 | |
| | | | |
| 2. | Збір вихідних даних. | 03.02.2025 | |
| | | | |
| 3. | Обробка методичних та теоретичних | 04.03.2025 | |
| | джерел. | | |
| | | | |
| 4. | Розробка першого та другого розділу. | 15.04.2025 | |
| | | | |
| 5. | Розробка третього розділу. | 19.05.2025 | |
| | | | |
| 6. | Оформлення та нормоконтроль | 26.05.2025 | |
| | кваліфікаційної роботи бакалавра. | | |
| | | | |
| 7. | Захист кваліфікаційної роботи. | 17.06.2025 | |

Студент _____ Б.О. Онопрієнко
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.Г. Кривохата
(підпис) (ініціали та прізвище)

## Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

# РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка онлайн-платформи для ведення блогів з використанням React та Node.Js»: 51 с., 25 рис., 10 джерел, 1 додаток.

БЕКЕНД, ІНТЕРФЕЙС, КЛІЄНТ, КОМПОНЕНТ, МАРШРУТ, МОДУЛЬ, СЕРВЕР, СКБД, СТЕК, API, DOM, MIDDLEWARE, SPA, UI.

Об’єкт дослідження – процес розробки онлайн-платформи для ведення блогів.

Предмет дослідження – React та Node.Js.

Мета роботи: розробка вебзастосунку для ведення блогів з використанням React та Node.Js.

Метод дослідження – методи об’єктно-орієнтовного програмування, методи конструювання програмного забезпечення.

У кваліфікаційній роботі розглядається процес розробки онлайн-платформи для ведення блогів. Для створення інтерфейсу користувача застосовується бібліотека React у поєднанні зі збирачем модулів Vite, що забезпечує високу продуктивність розробки та оптимізацію застосунку. Серверна частина реалізована за допомогою програмної платформи Node.js та фреймворку Express, який відповідає за обробку HTTP-запитів та реалізацію логіки доступу до функціоналу системи. Для реалізації системи автентифікації та авторизації використовується сервіс Auth0, який забезпечує безпечне управління користувачами.

Перевагами обраних технологій є швидкість розробки, масштабованість, гнучкість у налаштуванні, а також висока продуктивність і зручність у використанні як для розробника, так і для кінцевого користувача. Реалізована платформа є кросбраузерною.

# SUMMARY

Bachelor's qualifying paper "Development of an Online Platform for Blogging Using React and Node.js": 51 pages, 25 figures, 10 references, 1 supplement.

API, BACKEND, CLIENT, COMPONENT, DBMS, DOM, FRONTEND, INTERFACE, MIDDLEWARE, MODULE, ROUTE, SERVER, SPA, STACK, UI.

Object of the research is the process of developing an online blogging platform using React and Node.js.

The subject of the research is React and Node.js.

Purpose of the study is the development of a web application for blogging with React and Node.js.

Research methods are object-oriented programming methods and software-construction methods.

The qualification work examines the process of creating an online platform for blogging. The React library is used to build the user interface together with the Vite module bundler, which provides high development performance and application optimization. The server side is implemented with the Node.js runtime platform and the Express framework, which handles HTTP requests and realizes the access logic to the system's functionality. Authentication and authorization are implemented via the Auth0 service, ensuring secure user management.

The advantages of the chosen technologies are development speed, scalability, configuration flexibility, and high performance, offering convenience for both developers and end users. The implemented platform is cross-browser.

**ЗМІСТ**

Завдання на кваліфікаційну роботу ……..…….2
Реферат ..... 4
Summary ..... 5
Вступ..... 8
1 Аналіз предметної області..... 9
1.1 Опис предметної області ..... 9
1.2 Огляд мови програмування JavaScript ..... 10
1.3 Визначення способу відтворення застосунку ..... 11
1.4 Обґрунтування вибору бібліотеки React ..... 12
1.5 Обґрунтування вибору Node.js ..... 15
1.6 Обґрунтування вибору СКБД PostgreSQL..... 16
2 Проєктування застосунку ..... 18
2.1 Аналіз та визначення функціональних вимог системи ..... 18
2.2 Аналіз та визначення нефункціональних вимог системи ..... 20
2.3 Концептуальне проєктування системи ..... 23
2.3.1 Розробка макетів головних сторінок системи..... 23
2.3.2 Діаграма послідовності сценаріїв використання системи ..... 25
2.4 Проєктування логічної моделі бази даних ..... 28
2.5 Проєктування фізичних компонентів системи ..... 31
2.5.1 Діаграма компонентів інформаційної системи ..... 31
2.5.2 Діаграма розгортання інформаційної системи ..... 34
2.6 Висновки другого розділу ..... 35
3 Програмна реалізація застосунку ..... 36
3.1 Огляд архітектури інформаційної системи ..... 36
3.2 Огляд програмної реалізації ключових сценаріїв..... 38
3.2.1 Сценарій авторизації та автентифікації..... 38
3.2.2 Сценарій управління дописами ..... 41

Висновки ..... 48
Перелік посилань..... 49
Додаток А Код Docker Compose контейнера ..... 51

# ВСТУП

Практично кожна сучасна людина хоча б раз у житті стикалася з блогами під час користування різними онлайн-платформами. Платформа, реалізована в межах виконання кваліфікаційної роботи, може бути корисною для широкого кола користувачів із найрізноманітніших сфер, які бажають поділитися власним досвідом у певних галузях, або просто добре провести час.

Мета – розробка вебзастосунку для ведення блогів із використанням React та Node.Js.

Досягнення поставленої мети передбачає реалізацію наступних завдань:

– аналіз предметної області;
– дослідження програмних засобів розробки;
– проєктування інформаційної системи;
– розробка програмної реалізації системи.

Об’єкт дослідження – процес розробки онлайн-платформи для ведення блогів.

Предмет дослідження – фреймворки React та Node.Js.

Методи дослідження – методи об’єктно-орієнтовного програмування, методи конструювання програмного забезпечення.

Кваліфікаційна робота складається із вступу, трьох розділів, висновків, переліку посилань та додатку. Загальний обсяг роботи становить 51 сторінку. Додатки викладено на 1 сторінці. У переліку посилань 10 джерел.

Перший розділ присвячений аналізу предметної області, огляду інструментів розробки та аргументацій на користь їх вибору. Другий розділ присвячений проєктуванню системи. В третьому розділі описано програмну реалізацію системи. У додатку наведено конфігурацію Docker Compose.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У вебпросторі існує великий спектр онлайн-платформ, які надають користувачам можливість створювати та публікувати власні дописи. Серед найвідоміших:

– Twitter – орієнтований на короткі текстові повідомлення;
– Medium – платформа для детальних дописів з акцентом на журналістику;
– Reddit – майданчик для тематичних обговорень;
– StackOverflow – платформа для обміну професійними знаннями.

Попри велику кількість доступних рішень, багато з них мають певні обмеження: складні інтерфейси, комерціалізацію, наявність реклами та непрозорої системи рекомендацій контенту, обмежені можливості форматування, закритість програмного коду тощо.

Створення власної платформи дозволить уникнути вищеописаних проблем та забезпечить повний контроль над структурою, зовнішнім виглядом, функціональністю та прозорістю застосунку.

Крім того, створення такої системи є актуальним завданням для набуття навичок програмування та комплексної розробки застосунків, в цілому дозволяючи опанувати повний цикл створення інформаційної системи від аналізу предметної області, до проєктування та безпосередньо створення програмної реалізації.

## 1.1 Опис предметної області

Предметною областю в кваліфікаційній роботі є онлайн-платформа блогів. Одним із найкращих рішень для виконання поставленого завдання було обрано реалізацію у вигляді SPA (Single Page Application). SPA – це

односторінковий застосунок, в якому всі зміни відбуваються без перезавантаження сторінки як у нативних застосунках, які можна встановити на комп’ютер [1]. Така поведінка досягається завдяки управлінню базовою поведінкою браузера, створенню механізму маршрутизації та асинхронному завантаженню даних системи. Односторінкові застосунки забезпечують кращий користувацький досвід порівняно зі звичайними вебсайтами.

Основними інструментами для створення інформаційної системи було обрано мову JavaScript з бібліотекою React та середовище Node.js з використанням фреймворку Express.

Бібліотека React буде основою побудови користувацького інтерфейсу (UI), вона легко інтегрується у систему, має велику екосистему допоміжних бібліотек та надає зручний інструментарій для розробки клієнтських застосунків.

## 1.2 Огляд мови програмування JavaScript

Як основний інструмент створення застосунку було обрано мову JavaScript. Відповідно до специфікації ECMAScript [2], це інтерпретована мова загального призначення, яка може виконуватись у всіх сучасних браузерах без встановлення додаткового програмного забезпечення. Крім того з появою середовища Node.js стало можливо використовувати цю мову не лише в сфері веброзробки, а й, наприклад, для створення серверних застосунків [3]. Це значно розширило сферу використання мови JavaScript, призвело до зростання її популярності та сприяло появі технічних стеків із інструментів та технологій різної спрямованості, де JavaScript використовується як єдина мова програмування, наприклад стек MERN (MongoDB, Express, React, Node.js).

Незважаючи на існування певних альтернатив, таких як Dart або Python з використанням компіляторів для JavaScript (наприклад, Brython або Transcrypt), саме JavaScript залишається стандартом у сфері розробки систем для вебпростору й фактично не має конкурентів.

Враховуючи швидкість розробки з використанням JavaScript, швидкість роботи самої мови, а також те, що вона фактично не має аналогів у сфері створення клієнтської частини застосунків і можливість використати єдину мову для клієнтської та серверної частини застосунку, було прийнято рішення щодо використання саме JavaScript.

### 1.3 Визначення способу відтворення застосунку

Перш за все, варто визначити, що розроблювальна система є вебзастосунком, а не вебсайтом, і буде створена згідно з архітектурним підходом SPA. Такий застосунок має працювати в рамках однієї сторінки без повторного її перезавантаження у процесі використання системи. Односторінковий застосунок працює наступним чином: під час першого візиту на сторінку завантажується єдиний HTML-файл (цей файл визначає DOM-структуру сторінки в браузері), а JavaScript-код буде динамічно оновлювати цю структуру за потреби з використанням вбудованого в браузер API [1].

Також важливим етапом є визначення способу відтворення застосунку, всього існує три [4]:

- Client-Side Rendering (CSR): відтворення сторінки відбувається на стороні клієнта (у браузері);
- Server-Side Rendering (SSR): відтворення сторінки відбувається на сервері, після чого оновлені елементи надсилаються на клієнт;
- Static-Side Rendering (SSR): сервер відтворює всі сторінки заздалегідь і надсилає їх за потреби (після запиту від клієнта).

У межах кваліфікаційної роботи перевага була надана саме CSR-способу, де відтворення відбувається на стороні клієнта. Платформа онлайн-блогів має багато елементів, які динамічно оновлюються й не потребують для цього жодної взаємодії зі серверним API, лише актуальні дані системи. Тобто

вибір на користь SSR спричинив появу надлишкової кількості запитів до сервера, такі операції за умови поганої мережі будуть створювати лише затримки оновлення та погіршать якість використання системи користувачем, не надаючи ніяких переваг. Static-Side Rendering, у свою чергу, не підходить для розроблюваної системи за визначенням, він використовується у системах, де немає часто оновлюваних даних (йдеться про дані, які повинні спричинити оновлення користувацького інтерфейсу).

### 1.4 Обґрунтування вибору бібліотеки React

Для прискорення розробки та стандартизації застосунку потрібно дотримуватись певних стандартів при розробці системи, а також використовувати певні інструменти. Зараз для розробки SPA-застосунків визначаються три інструменти: бібліотека React, фреймворки Vue і Angular [5].

Кожна із перелічених технологій використовується для прискорення створення користувацького інтерфейсу, та стандартизації розроблювальної системи, надаючи певну абстракцію над логікою оновлення сторінки браузера.

React – це бібліотека для JavaScript [6], яка допомагає створювати користувацькі інтерфейси, була розроблена компанією Meta у 2013 році. Бібліотека не нав’язує конкретну архітектуру розроблювального застосунку й дозволяє визначати її самотужки. React лише надає інструментарій для швидкої стандартизованої розробки та масштабування, дозволяє повторно використовувати раніше написаний код, і керувати великою кількістю динамічних елементів сторінки. Вона легко інтегрується з іншими бібліотеками й має обширну екосистему з додаткових бібліотек. Наприклад, для створення маршрутизації часто використовують бібліотеку React Router, а для стилізації елементів UI використовують Material UI.

Із недоліків варто відзначити велику вагу бібліотеки, значну частину якої займає механізм Virtual DOM, це може відштовхувати від React якщо

потрібно створити інформаційну систему невеликого розміру. Virtual DOM також потребує дещо більшого використання пам’яті для роботи застосунку, проте при створенні систем середнього рівня розміру та правильному підходу до оптимізації системи це не є суттєвим недоліком щоб відмовлятись від React [5].

Vue є фреймворком, який був розроблений у 2014 році, його головна мета – надати простий та гнучкий інструментарій для створення SPA-застосунків. Цей фреймворк передбачає певні обмеження при розробці, на відміну від React, він використовує шаблонний підхід до створення компонентів. Основна перевага Vue – це простота використання (компоненти створюються за допомогою базового синтаксису HTML, CSS та JavaScript) та досить невелика вага фреймворку. Невелика вага досягається компактністю ядра фреймворку, саме тому він має гарну продуктивність та швидкість роботи. Проте при розробці важких інформаційних систем, компактного ядра скоріш за все, буде замало й його доведеться розширювати за допомогою модулів (наприклад інструменти маршрутизації або керування станом), таким чином початкова компактність буде втрачена.

Із недоліків окрім можливої втрати компактності ядра варто відзначити досить молоду екосистему, внаслідок чого може відчуватись нестача готових рішень у вигляді бібліотек тощо. Також при створенні складних систем, може страждати продуктивність застосунку через надмірну кількість реактивних залежностей, або необґрунтовану реактивність компонентів. Для уникнення цих проблем потрібно буде впроваджувати продумані механізми оптимізації.

Vue – це хороший вибір для розробки невеликих або середніх за розміром систем, проте з розвитком цієї системи може виникати потреба у додатковому інструментарії, якого може або не бути, або він лишить фреймворк початкової компактності [5].

Angular – це фреймворк який розробила компанія Google. Його плюсами вважається модульний підхід до формування архітектури застосунку, що в свою чергу сприяє високому рівню масштабованості. Також Angular визначає

жорстку структуру модулів Model-View-Controller (MVC), це може бути як плюсом так і мінусом залежно від інформаційної системи та кількості розробників.

Із суттєвих недоліків варто відзначити велику вагу фреймворку, а також складнощі з SEO-оптимізацією. Погано налаштована SEO-оптимізація позбавить систему можливостей «спливати» при звичайному пошуку через пошукові запити.

Angular – це ідеальне рішення для розробки високопродуктивних корпоративних застосунків, які розробляють у великих командах, проте при індивідуальній розробці в рамках виконання кваліфікаційної роботи, надмірна «важковаговість» може стати суттєвим недоліком [5].

З усіх розглянутих інструментів саме бібліотека React буде найбільш доцільним та збалансованим вибором для створення інформаційної системи “InkFlow”. Вона поєднує в собі: гнучкість розробки, продуктивність, простоту масштабування та широкий спектр додаткових бібліотек.

Ключовими особливостями React є: компонентний підхід до створення інтерфейсу, декларативний опис компонентів, реактивність системи, механізми узгодження (Reconciliation та DOM Reconciliation), механізм Virtual DOM [7].

Основною структурною одиницею побудови клієнтського застосунку буде виступати React-компонент. Існують два типи компонентів: класові та функціональні. Згідно до офіційної документації React [8], наразі використання функціональних компонентів є пріоритетним.

Функціональні компоненти забезпечують простоту та лаконічність розроблювальної системи, кожний компонент це звичайна функція. Також вони створюють високу продуктивність застосунку, оскільки мають високу швидкість відтворення порівняно з класовими.

В ході виконання кваліфікаційної роботи функціональні компоненти стануть основою для побудови користувацького інтерфейсу програми.

## 1.5 Обґрунтування вибору Node.js

Node.js – це середовище виконання JavaScript коду на сервері, створене на основі рушія V8 від Google Chrome [3].

Головною особливістю Node.js є неблокуюча модель введення та виведення даних Event Loop. Цей механізм дозволяє ефективно оброблювати велику кількість одночасних операцій без створення окремих потоків. Середовище Node.js ідеально підходить для створення REST API застосунків з великою кількістю користувачів системи, як наприклад, у межах виконання кваліфікаційної роботи. Проте для об’єктивного обґрунтування вибору слід розглянути й інші можливі аналоги для створення серверного API.

Одна із найпоширеніших мов програмування для створення серверних застосунків це PHP. Вона досить довго існує і добре працює у поєднанні з класичними вебсайтами, має великий обсяг доступної інформації для вивчення, а також обширну екосистему з бібліотек та фреймворків тощо. Проте вибір на користь PHP закриває можливість дотриматися єдиного технологічного стеку при створенні системи, додає додаткової складності при організації передачі та обробки даних між клієнтом та сервером, і в цілому зменшить швидкість розробки всієї системи та її уніфікованість, не надаючи ніяких переваг [9].

Для створення серверних застосунків також поширеними мовами є Java та C# (.NET), це потужні рішення з високою продуктивністю, мають сильну типізацію й в цілому ідеально підходять для побудови великих інформаційних систем корпоративного рівня. Проте, як недолік, вони потребують додаткового якісного засвоєння, також як і з PHP, використовуючи Java або C# не вдасться дотриматися єдиного технологічного стеку розроблювального застосунку. Інфраструктура інформаційної системи буде перенавантаженою у порівнянні з використанням Node.js, що призведе до більшого її розміру й не надасть ніяких вагомих переваг. Тому вибір на користь Java або C# буде недоцільним із урахуванням масштабу розроблювальної інформаційної системи.

Отже, незважаючи на те, що для створення серверного API може бути використано багато мов програмування, перевага була надана саме Node.js. Це швидкодійне, оптимізоване для подібних задач середовище, використання Node.js додатково забезпечить єдність технологічного стеку, та зменшить складність у координації між клієнтом та сервером сприяючи простішому масштабуванню додатку у майбутньому.

### 1.6 Обґрунтування вибору СКБД PostgreSQL

PostgreSQL – це потужна реляційна система керування базами даних (БД) з відкритим вихідним кодом. Вона активно використовується у проєктах різного масштабу та рівня складності. PostgreSQL підтримує ACID-транзакції, має сильну типізацію, референтну цілісність, розвинуту систему індексацій, можливість створення складних SQL-запитів тощо [10]. Також ця СКБД відзначається високою стабільністю роботи, розширюваністю функціоналу за допомогою плагінів, та обширною екосистемою.

Для визначення коректності вибору на користь PostgreSQL потрібно порівняти її з іншими поширеними аналогами, особливо у сфері веброзробки.

Перш за все, як найбільш інтуїтивно-очікуваний вибір, слід розглянути MongoDB. Це документоорієнтована БД яка добре підходить для швидкої розробки та гнучкого зберігання даних у форматі JSON. Гнучкість MongoDB корисна, коли інформаційна система не має чіткої ієрархії сутностей, або ж просто коли система не вимагає більш складних та організованих операцій аніж просто зберігання даних.

MongoDB, як і всі документорієнтовні СКБД не забезпечують строгих взаємозв’язків між сутностями, що може ускладнювати підтримку систем, де зв’язки мають бути чітко структуровані. Свобода використання, яку надають документорієнтовні СКБД може бути як їх перевагою так і їх недоліком, залежно від розроблювальної системи.

Інша можлива альтернатива це – MySQL, реляційна БД яка часто використовується у сфері веброзробки, особливо в поєднанні з PHP. У порівнянні з вищеописаною PostgreSQL має менш гнучку типізацію, наприклад обмежену підтримку складних типів, таких як JSONB, масивів та користувацьких типів; менш виражену підтримку плагінів і тригерів. В контексті REST API архітектури ці обмеження можуть ускладнювати реалізацію складної логіки додатку, а також подальший розвиток системи й створення нового функціоналу.

Таким чином, доцільно обирати між MongoDB як частиною MERN стеку, та PostgreSQL як класичним реляційним рішенням. Попри те, що MongoDB на перший погляд здається зручнішою, саме PostgreSQL є кращім вибором. Інформаційна система формату онлайн-блогу має чітко визначену структуру сутностей та їх взаємозв’язків, наприклад: користувач та дописи, дописи та коментарі, тощо. Тож система має ієрархічну структуру, де дані чітко зберігаються структуровано, а сутності мають визначені взаємозв’язки.

Отже, PostgreSQL було обрано як найкращій варіант для створення платформи онлайн-блогу, яка передбачає чітку визначену структуру даних, а також як стабільне та перевірене часом рішення з обширним функціоналом і гарною продуктивністю [10]. Задля спрощення взаємодії з БД буде використано шар абстракції, бібліотека PrismaORM. Комбінація PostgreSQL та PrismaORM забезпечить надійність, масштабованість та легку підтримку застосунку.

# 2 ПРОЄКТУВАННЯ ЗАСТОСУНКУ

Перед початком розробки інформаційної системи доцільно пройти етап проєктування системи, він формує вимоги до системи, визначає способи та інструменти, середовище для реалізації поставленого завдання.

Етап проєктування дає змогу ще до початку розробки бути впевненим, як повинна функціонувати розроблена система. Крім того, етап проєктування дозволяє оцінити складність поставленого завдання та строки його реалізації, що особливо важливо у комерційній розробці інформаційних систем.

## 2.1 Аналіз та визначення функціональних вимог системи

Функціональні вимоги визначають набір можливостей застосунку та дозволяють ще на етапі проєктування зрозуміти, який функціонал системи буде створений для користувачів. Для інформаційної системи “InkFlow” висуваються такі функціональні вимоги:

– авторизація та автентифікація у системі;
– перегляд існуючих дописів;
– фільтрація контенту застосунку;
– створення власних дописів;
– можливість оцінювати дописи;
– можливість зберегти вподобаний допис;
– можливість написати коментар;
– можливість управляти власними дописами.

Розглянемо детальніше кожний пункт. Авторизація та автентифікація – користувач повинен мати можливість створити обліковий запис у системі. Для цього буде реалізовано процедуру реєстрації за допомогою сервісу Auth0. Таке рішення дозволить делегувати всі процеси обробки та зберігання чутливих даних користувача окремому надійному сервісу. Використання Auth0

додатково забезпечить певний рівень зручності для користувачів, оскільки дозволить реєструватися за допомогою будь-якого соціального логіну, наприклад, Google, Github тощо.

Перегляд існуючих дописів – користувачі повинні мати змогу зручно переглядати існуючі дописи без створення облікового запису. Для забезпечення додаткової зручності повинен бути реалізований механізм фільтрації контенту сторінки.

Створення власного допису – кожен зареєстрований у системі користувач повинен мати змогу створити власний допис. Для створення функціоналу редактору буде використана бібліотека Editor.js. Додатково для забезпечення зручності використання системи буде реалізовано можливість зберегти чернетку допису до його публікації. Авторизований користувач може видалити свій допис за бажанням.

Оцінювання дописів – користувач повинен мати змогу оцінити допис, оцінювати дописи можуть лише зареєстровані користувачі системи. Система оцінювання також буде надавати можливість зберегти вподобаний допис (по аналогії, як це зроблено, наприклад, у сервісі Github).

Написання коментарів – кожен авторизований у системі користувач повинен мати змогу залишити коментар до допису. Система коментарів повинна бути каскадною, для створення можливості обговорювати дописи з іншими користувачами системи.

Перелічені функціональні вимоги продемонстровані у вигляді UML-діаграми варіантів використання (див. рис. 2.1). Діаграма відображає актора «Користувач» (User) та вищеописані варіанти використання системи.

Можливості користувача по взаємодії із системою поділяються на дві основні групи залежно від його статусу. Не авторизований у системі користувач може лише переглядати, сортувати та фільтрувати дописи. Авторизований у системі користувач має такі самі можливості, як неавторизований, і, у доповнення, можливість управління власними дописами та оцінками, управління власними коментарями, чернетками тощо.

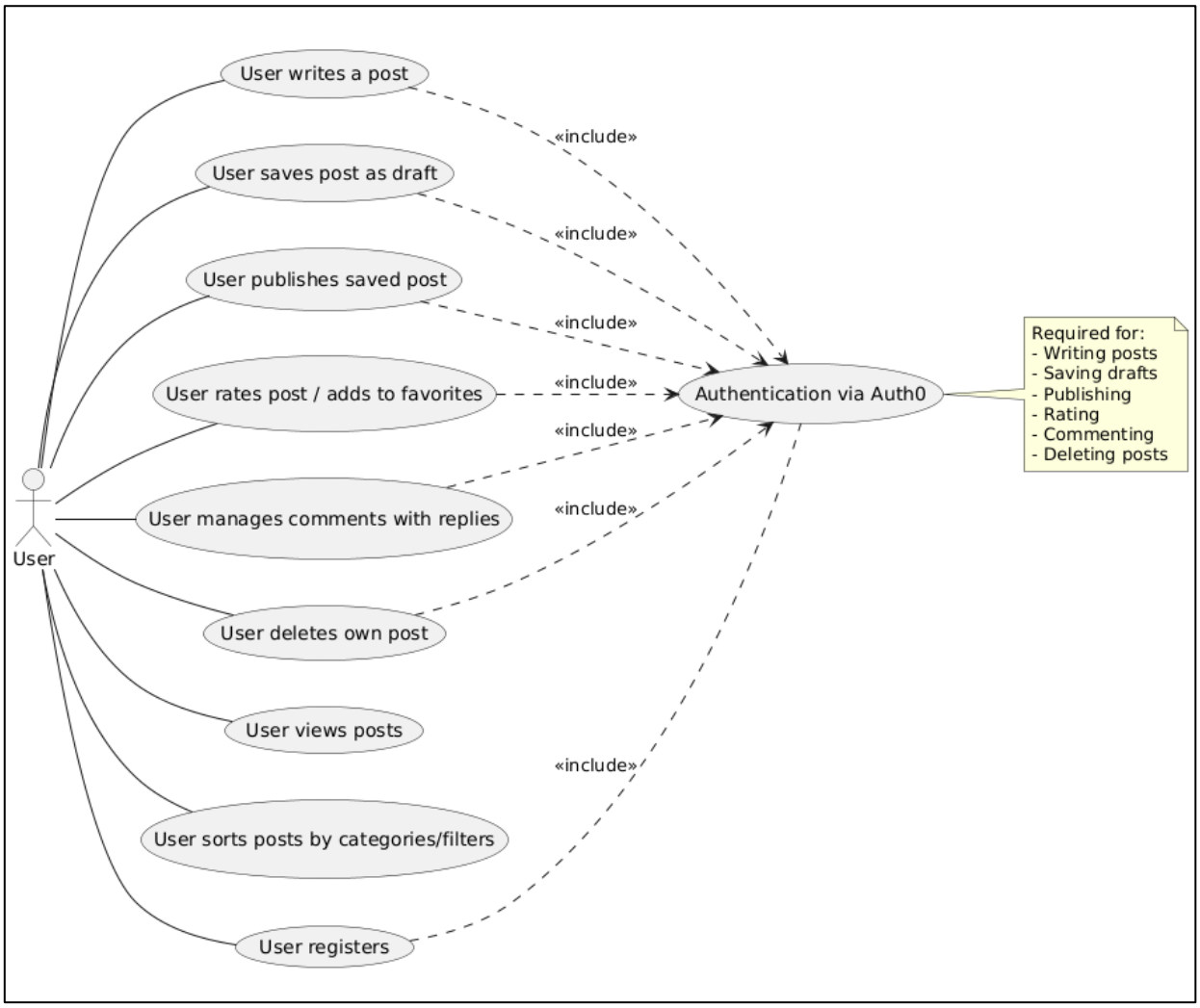



Рисунок 2.1 – Діаграма використання інформаційної системи “InkFlow”

## 2.2 Аналіз та визначення нефункціональних вимог системи

Нефункціональні вимоги дозволяють визначити якісні характеристики, які повинна задовольняти система, такі як: ефективність та швидкість розробки, зручність використання системи, безпечність чутливих даних користувача, відмовостійкість та можливість масштабування системи тощо.

До інформаційної розроблювальної інформаційної системи висуваються такі нефункціональні вимоги:

– система повинна швидко працювати;
– система повинна мати можливості щодо масштабування;
– система повинна забезпечити надійність даних користувача;

– система повинна мати внутрішні механізми її захисту;
– система повинна бути інтуїтивною у використанні.

Розглянемо детально кожен пункт вимог. Швидкість роботи – система повинна завантажуватись якомога швидше, всі дії користувача мають оброблятися миттєво або з мінімальною затримкою. Виключенням є ситуація, коли користувач уперше заходить на сторінку застосунку, оскільки Virtual DOM може потребувати декілька секунд на перше завантаження.

Для досягнення поставленої умови, застосунок буде створено за принципом Client-Side відтворення і технологією SPA на основі бібліотеки React. Таке рішення дозволить користувачам працювати із застосунком без повторного перезавантаження сторінки.

Для роботи з контентом сторінки та редактором дописів буде використана бібліотека Editor.js, вона формує готові для використання як клієнтом так і сервером дані у форматі JSON; це значно ефективніше аніж передача всього HTML-коду сторінки допису. Editor.js дозволяє мінімізувати обсяг трафіку та спрощує взаємодію із даними, залишаючи лише необхідні дані для обробки застосунком.

При створенні компонентів будуть задіяні хуки та визначені чіткі залежності, зміна яких призведе до оновлення компоненту, таким чином вдасться уникнути надлишкової реактивності застосунку.

Масштабованість застосунку – програмна реалізація повинна передбачати можливість бути масштабованою у майбутньому.

Для досягнення цієї умови серверний API буде реалізовано за архітектурою модульного моноліту. Кожен модуль буде відповідати тільки за свою частину бізнес-логіки додатку. Крім цього, кожний модуль всередині буде розбито на три файли: роутер, сервіс та контролер – таке рішення створює слабку зв’язаність коду та дозволяє за потреби швидко доповнити наявний код.

Проміжні обробники (middleware) на сервері також будуть винесені в окремі модулі, за потреби їх можна буде замінити без зміни основної логіки роботи серверу. Базу даних буде розгорнуто всередині Docker-контейнеру, це

дозволить швидко змінювати модель даних, створювати тестові данні або ж винести БД на окремий сервер чи окремий сторонній сервіс. Усі ці кроки сприяють створенню слабкої зв’язаності коду застосунку й легкому масштабуванню його у майбутньому.

Безпека чутливих даних користувача – система повинна надійно зберігати дані користувача, ця вимога буде реалізована з використанням сервісу Auth0. Він безпечно зберігає дані та надає певний інструментарій для взаємодії системи з ними.

Система повинна мати внутрішні механізми захисту – для досягнення цієї вимоги, у застосунку буде впроваджено використання змінних оточень для даних, які не повинні бути відомі користувачам. Змінні оточення – це системні данні, наприклад: URL підключення до БД або ключ доступу для перевірки Auth0 токенів тощо.

На стороні клієнта буде використана бібліотека AXIOS, вона дозволить зручно керувати HTTP-запитами до серверного API, а також надає проміжний обробник для перевірки певної умови, яка повинна бути виконаною щоб запит відправився. Такий проміжний обробник називається інтерцептор, він буде перевіряти валідність токену доступу користувача. Якщо інтерцептор виявить, що токен користувача не дійсний – запит просто не буде надісланий, таким чином буде забезпечено захист від несанкціонованих звернень до серверу.

На боці серверу також будуть використовуватись проміжні обробники (middleware) для перевірки токену доступу користувача. Middleware як і інтерцептор запобігає несанкціонованому доступу до функціоналу застосунку. Якщо middleware виявить, що токен не дійсний, сервер навіть не почне обробляти запит, а просто поверне статус 401 (Unauthorized). Задля запобігання помилки роботи додатку, блоки коду, які потенційно можуть зламати застосунок, будуть огорнені в конструкцію “try/catch”.

Система повинна бути інтуїтивно зрозумілою у використанні – для досягнення цієї вимоги буде розроблено зручний інтерфейс системи, а також впроваджено сповіщення, модальні вікна для неочевидних дій.

## 2.3 Концептуальне проєктування системи

Концептуальне проєктування системи – це один із ключових етапів створення інформаційної системи, оскільки саме на цьому етапі формується загальне бачення архітектури майбутнього застосунку, логіка його роботи та принципи взаємодії користувача та системи.

Головним завданням етапу концептуального проєктування є перехід від поставлених вимог до абстрактної моделі системи. Цей етап формує розуміння того як повинна виглядати та функціонувати розроблювальна система.

### 2.3.1 Розробка макетів головних сторінок системи

Для інформаційної системи “InkFlow” було розроблено макети головних сторінок застосунку, ці макети слугують візуальним представленням користувацького інтерфейсу системи.

Кожна сторінка відображає заголовок та нижню частину застосунку. Заголовок виконує роль навігаційного меню й містить: назву інформаційної системи, навігаційне меню системи, профіль користувача. Нижня частина містить інформацію про автора.

Головна сторінка відображає основний блок контенту, який містить: поле з дописами, фільтри по категоріям та фільтр сортування (див. рис. 2.2).

Сторінка перегляду допису містить заголовок допису, автора допису, дату написання та рейтинг допису; нижче відображається основний контент допису та секція коментарів (див. рис. 2.3).

Наступний макет присвячений сторінці створення власного допису, – вона надає можливість авторизованому в системі користувачу створити власний допис (див. рис. 2.4). Допис містить: заголовок, вибір категорії, та власне контент. У ролі контенту може бути: текст, зображення, посилання, списки, чекліст тощо. Редактор підтримує всі стандартні можливості форматування та вирівнювання тексту.

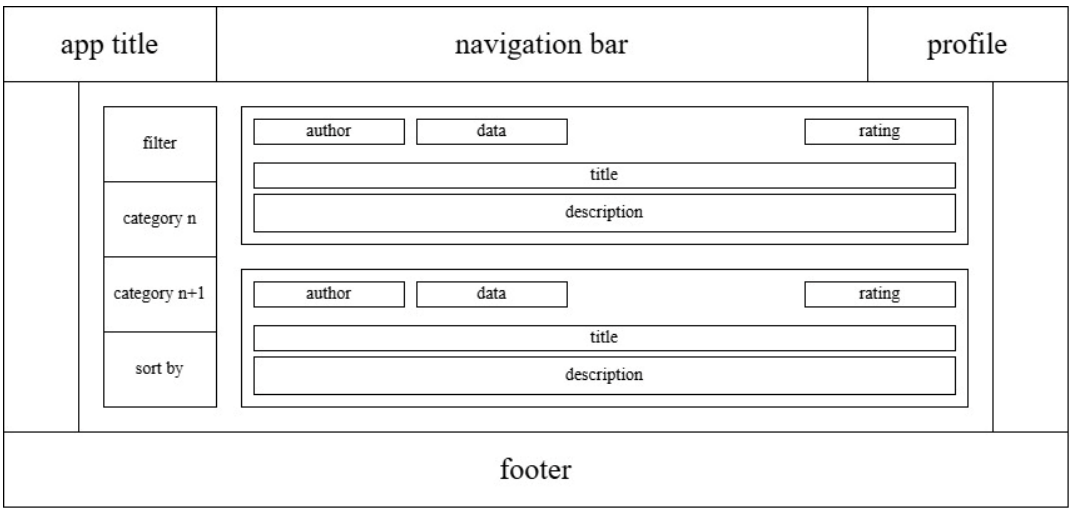



Рисунок 2.2 – Головна сторінка застосунку

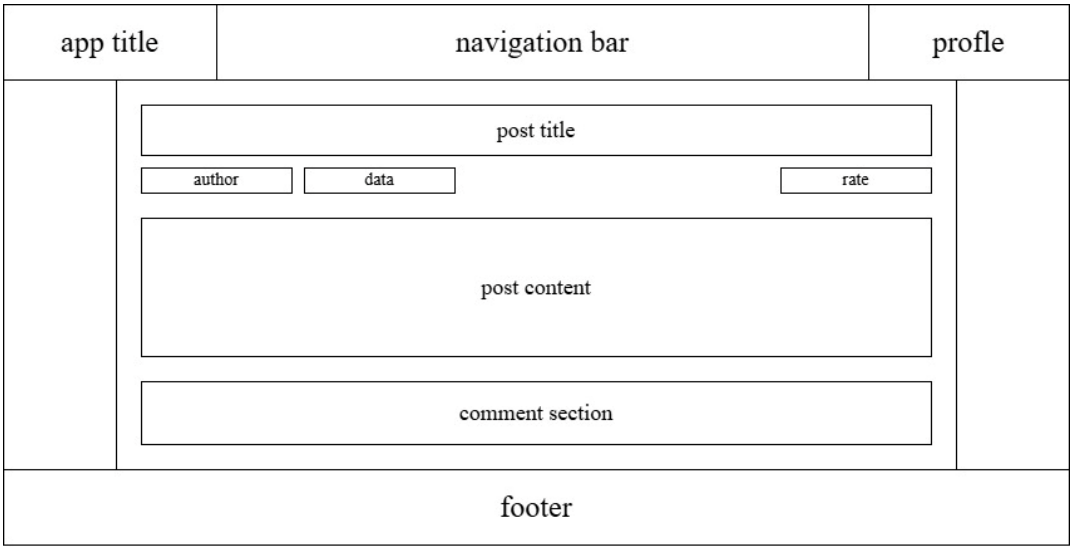



Рисунок 2.3 – Сторінка перегляду обраного допису

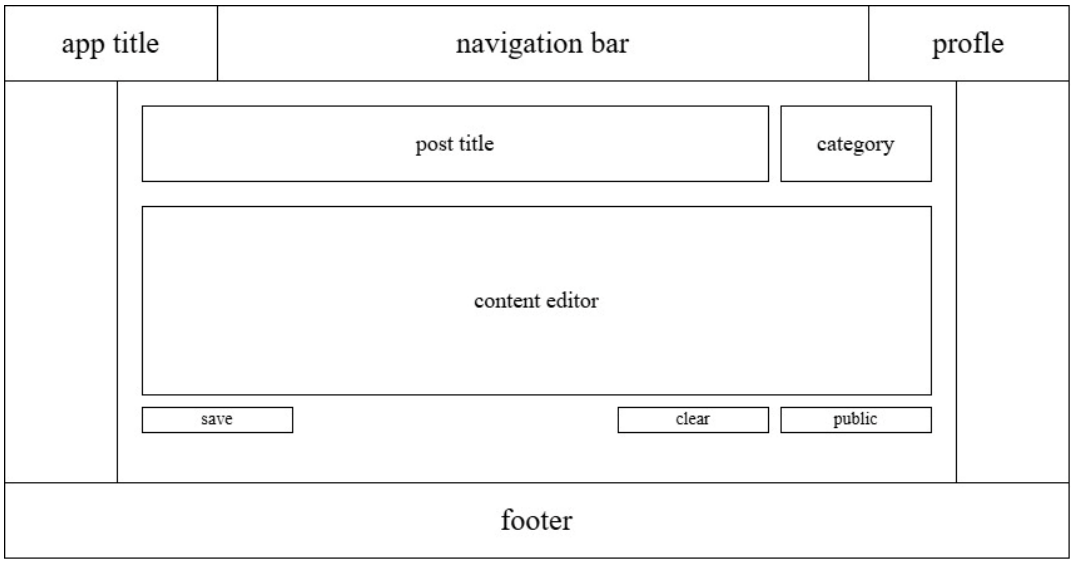



Рисунок 2.4 – Сторінка створення допису

Також сторінка створення допису підтримує автоматичне (за заданим інтервалом), та ручне збереження у чернетки.

Сторінка створення допису також буде наново використаною для створення сторінки редагування чернетки.

Наступна сторінка – профіль користувача, містить інформацію про користувача, та статистику. Має фільтр для перегляду: власних дописів, улюблених дописів, чернетки (див. рис. 2.5).

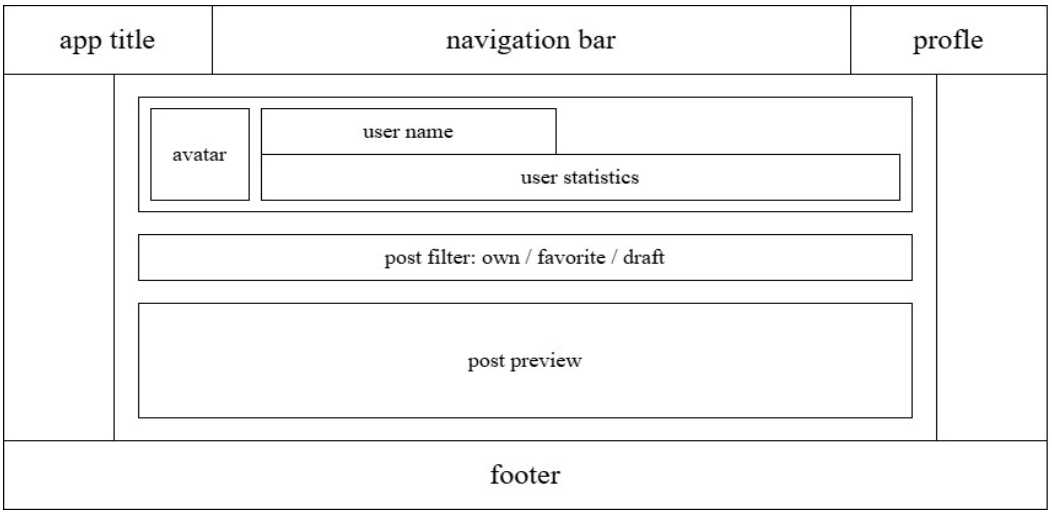



Рисунок 2.5 – Сторінка профілю користувача

Кожна сторінка отримує відповідні данні за допомогою HTTP-запиту на сервер та використовує отримані дані для створення UI.

### 2.3.2 Діаграма послідовності сценаріїв використання системи

Важливим етапом проєктування системи є створення діаграми послідовності, вона дозволяє відобразити взаємодію користувача та системи у часі. Для інформаційної системи “InkFlow” було розроблено діаграму послідовності, вона відображає основні сценарії взаємодії користувача та системи. Кожний представлений фрагмент відповідає за свій сценарій, показує як взаємодіють різні компоненти системи між собою, як система реагує на дії користувача, відображає послідовність запитів та відповідей (див. рис. 2.6).

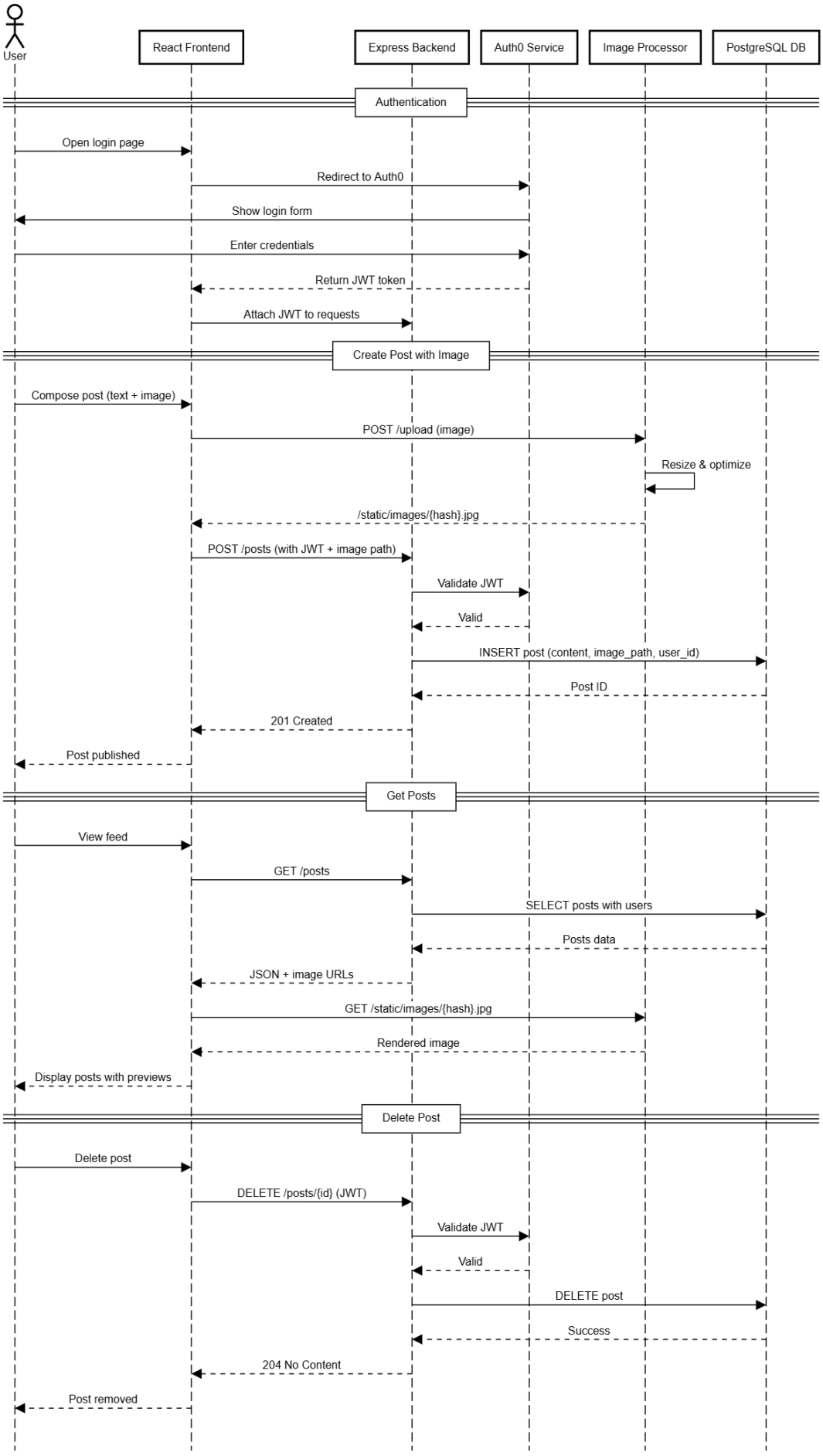



Рисунок 2.6 – Діаграма послідовності

Розроблена діаграма послідовності забезпечує узгодженість між клієнтським застосунком та серверним API, що особливо важливо в межах виконання кваліфікаційної роботи оскільки в розроблювальному SPA-застосунку значна частина бізнес-логіки знаходиться саме на боці клієнта (у браузері).

Фрагмент “Create Post” демонструє алгоритм створення допису – користувач переходить на сторінку створення допису, заповняє необхідні поля та відправляє запит на серверний API разом із своїм токеном доступу. Якщо в процесі створення допису користувач завантажує зображення, – відправляється POST-запит на модуль сервера, який оброблює та зберігає зображення. Зображення перевіряється, йому присвоюється унікальне ім’я та зберігається локально на сервері. На клієнт відправляється готове посилання на це зображення. Тобто, коли користувач публікує допис, у таблицю БД дописів, записується вже готові до використання дані. Варто відзначити, що алгоритм роботи з зображеннями аналогічний будь-якому CDN сервісу.

Фрагмент “Delete Post” демонструє, як працює видалення допису: авторизований у системі користувач заходить на сторінку перегляду профілю, обирає допис та натискає кнопку “Delete”, після чого надсилається DELETE-запит. Разом із запитом надсилається токен доступу користувача. Сервер перевіряє токен, і, якщо все добре, видаляє пост.

Фрагмент “Authentication” демонструє процес роботи механізму авторизації та автентифікації: як відбувається реєстрація та надання токену доступу для користувача. Токен доступу потрібен, щоб виконувати автентифікацію користувача. Алгоритм виглядає наступним чином: користувач заходить на сторінку застосунку й нажимає кнопку “Sign Up” або “Login”, система перенаправляє його на сторінку Auth0. Сервіс Auth0 надає користувачу форму для реєстрації, користувач вводить необхідні дані, якщо все добре, Auth0 перенаправляє користувача на головну сторінку застосунку й надає йому токен доступу.

Фрагмент “Get Post” демонструє, як працює перегляд дописів. Цей функціонал не передбачає створення облікового запису у системі. Коли

користувач заходить на головну сторінку, надсилається GET-запит, сервер його оброблює і надсилає дописи у форматі JSON.

### 2.4 Проєктування логічної моделі бази даних

Модель БД описує які система матиме сутності, та як ці сутності будуть взаємопов’язані. Розроблена модель містить не лише опис існуючих сутностей, а й атрибути які відповідають типам PostgreSQL, також на моделі були зазначені первинні й зовнішні ключі (див. рис. 2.7).

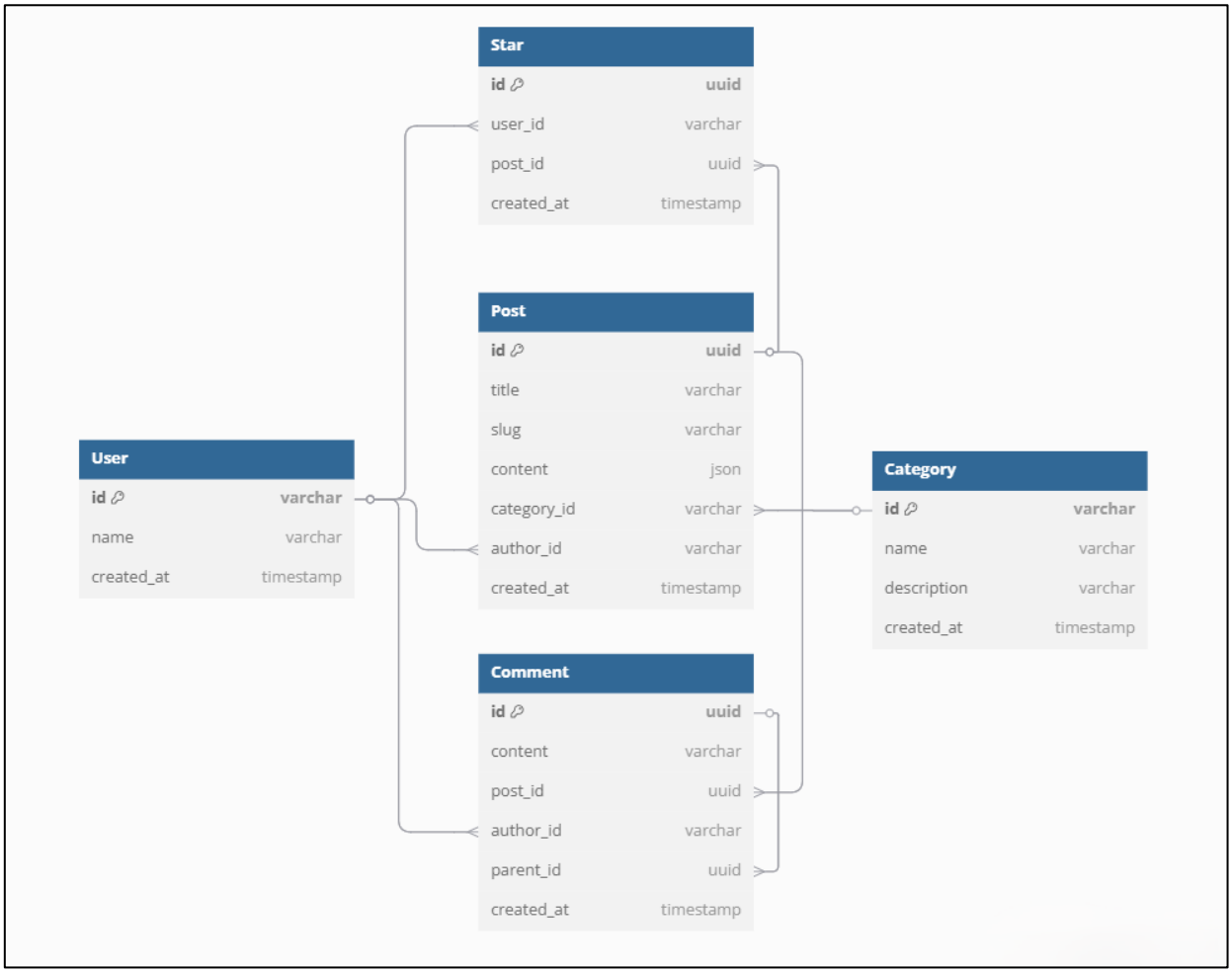



Рисунок 2.7 – Модель бази даних для інформаційної системи “InkFlow”

Варто зазначити, що моделях сутностей PrismaORM зовнішні ключі описуються через спеціальне поле “relation”, а зв’язки реалізуються зі

зворотного боку відношення. Наприклад модель User має зв’язок «один-до-багатьох» з моделлю Post, однак сама таблиця User не буде містити атрибут “posts”, це відношення визначається атрибутом “author_id” (зовнішній ключ на “id” User) у таблиці Post. Розглянемо кожну сутність, її атрибути та зв’язки з іншими сутностями.

Сутність User – відображає зареєстрованого користувача системи, використовується для зберігання мінімального набору інформації про користувача, якого буде достатньо для взаємодії з іншими сутностями. Всі додаткові дані користувача, зберігає сервіс Auth0.

Атрибути сутності User:

– id: унікальний ідентифікатор користувача, відповідає властивості Auth0 “user.sub”, використовується як первинний ключ;
– name: зберігає ім’я користувача;
– created_at: зберігає дату реєстрації користувача.

Відношення сутності User з іншими сутностями:

– posts: зв’язок один-до-багатьох;
– comments: зв’язок один-до-багатьох;
– stars: зв’язок один-до-багатьох.

Сутність Post – використовується для зберігання дописів. Атрибути сутності Post:

– id: унікальний ідентифікатор, автоматично генерується uuid;
– title: заголовок поста;
– slug: використовується для створення читабельних URL;
– content: зберігає вміст допису;
– category_id: зовнішній ключ, який вказує категорію допису;
– author_id: зовнішній ключ, вказує автора допису;
– created_at: дата створення поста, генерується автоматично Prisma.

Відношення сутності Post з іншими сутностями:

– author: зв’язок багато-до-одного;
– category: зв’язок багато-до-одного;

– comments: зв’язок один-до-багатьох;
– stars: зв’язок один-до-багатьох.

Сутність Comment – зберігає коментарі для дописів, підтримує каскадну структуру коментарів. Атрибути сутності Comment:

– id: унікальний ідентифікатор коментаря, генерується uuid;
– content: текстове поле, що зберігає вміст коментаря;
– post_id: зовнішній ключ на пост до якого належить коментар;
– author_id: зовнішній ключ на користувача, який залишив коментар;
– parent_id: необов’язковий зовнішній ключ, що вказує на батьківський коментар;
– created_at: поле дати створення коментаря.

Відношення сутності Comment з іншими:

– post: зв’язок багато-до-одного;
– author: зв’язок багато-до-одного;
– parent: рекурсивний зв’язок один-до-багатьох, дозволяє коментарям мати дочірні коментарі й будувати каскадну структуру;
– children: зворотна сторона рекурсивного зв’язку, список дочірніх коментарів для коментаря.

Сутність Star – зберігає інформацію про оцінки (зірочки), які користувачі ставлять постам. Користувач може поставити лише одну зірочку лише одному посту.

Атрибути сутності Star:

– id: унікальний ідентифікатор, генерується у форматі uuid;
– user_id: зовнішній ключ на користувача, який поставив зірочку;
– post_id: зовнішній ключ на пост, якому поставлено зірочку;
– created_at: поле дати створення оцінки, генерується автоматично.

Для цієї моделі існує унікальне обмеження “[ user_id, post_id ]”, воно забороняє користувачу ставити зірочку одному й тому самому посту більше одного разу (прибрати й поставити зірочку знову, – можна).

Відношення сутності Star з іншими:

– user: зв’язок багато-до-одного;
– post: зв’язок багато-до-одного.

Сутність Category – використовується для фільтрації постів за категорією.

Атрибути сутності Category:

– id: унікальний ідентифікатор категорії;
– name: унікальне текстове поле, що зберігає назву категорії;
– description: необов’язкове текстове поле з описом категорії;
– created_at: поле дати створення категорії.

Відношення сутності Category з іншими:

– posts: зв’язок один-до-багатьох.

## 2.5 Проєктування фізичних компонентів системи

Етап фізичного проєктування зазвичай є завершальним, його суть полягає у перенесенні абстрактної моделі системи у технічну реалізацію з урахуванням: обраних інструментів та технологій, платформ розгортання, розглянутих зовнішніх сервісів тощо. На попередніх етапах проєктування основна увага приділялась моделюванню, що повинна роботи система, як вона повинна це робити, й якими якісними характеристиками володіти. На етапі фізичного проєктування моделюється, як система повинна бути побудованою, як будуть взаємопов’язані вузли системи та, де система буде розгорнута.

### 2.5.1 Діаграма компонентів інформаційної системи

Для інформаційної систему було розроблено діаграму компонентів, вона наочно відображає фізичну структуру застосунку: основні компоненти системи, їх взаємозв’язки та взаємодію системи із зовнішніми сервісами.

На відміну від діаграми прецедентів або діаграми послідовності, які моделювали можливості системи та її поведінку, – діаграма компонентів моделює як система побудована та як взаємодіють вузли системи. Діаграма компонентів зображена на рисунку 2.8.

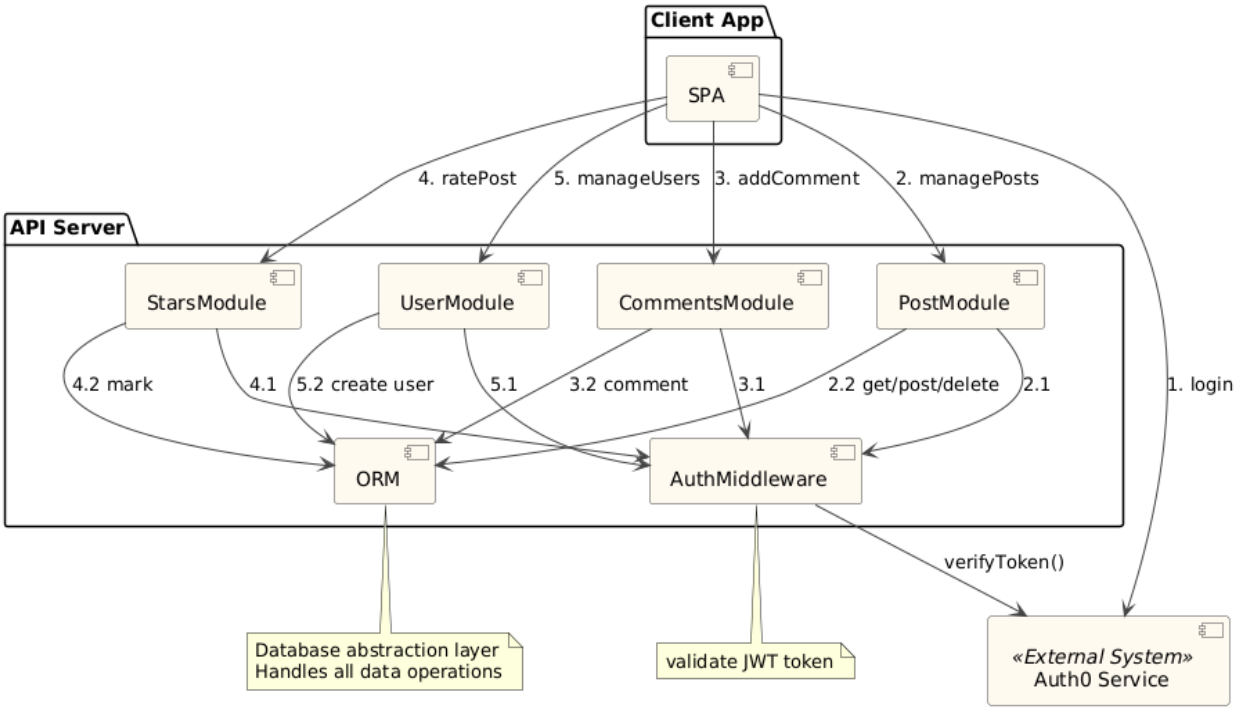



Рисунок 2.8 – Діаграма компонентів системи “InkFlow”

На діаграмі зображені клієнтський та серверний частини інформаційної системи, а також зовнішній сервіс Auth0 який відповідає за зберігання чутливих даних користувача.

Користувацький застосунок (клієнт) буде реалізовано як SPA-застосунок, користувачу буде надано функціональні можливості: управління дописами, управління коментарями, управління оцінками тощо.

При створенні допису, коментаря, оцінки тощо: клієнт перевіряє дані, формує запит, перевіряє токен доступу користувача й надсилає запит на серверний API. Перевірка вхідних даних від користувача (наприклад коли користувач створює допис) виконується за допомогою форм, запит до серверу формується із використанням бібліотеки AXIOS, вона надає зручний функціонал створення запитів а також інструменти інтерцептор, за допомогою

якого виконується перевірка токену доступу користувача на клієнті. Якщо токен не дійсний і його не вдасться оновити, запит на сервер не буде виконаний. Таким чином, буде забезпечені три нефункціональні вимоги: AXIOS дозволить шаблонувати запити, полегшуючи масштабування системи; забезпечує безпеку внутрішніх механізмів системи; SPA-застосунок матиме гарну швидкість роботи та інтуїтивний інтерфейс.

Серверна частина застосунку буде розроблена з дотриманням модульного підходу та використанням середовища Node.js у поєднанні з фреймворком Express. Кожен модуль сервера відповідає лише за свою частину логіки роботи системи, наприклад: модуль “post” відповідає тільки за створення, відправку, видалення дописів тощо. Також кожен модуль буде розбитий на три окремі сутності: роутер, контролер, сервіс. Таке рішення створює слабку зв’язаність коду та надає можливості для зручного масштабування системи або рефакторингу наявного коду. Фреймворк Express надає зручні інструменти для створення та обробки маршрутів, наприклад, проміжні обробники, які будуть використані для валідації токена доступу користувача. Проміжний обробник для перевірки токена буде виокремлений у відповідний модуль (AuthMiddleware), таке рішення дозволить за потреби дуже швидко замінити його на інший не змінюючи всієї логіки роботи маршруту.

Крім обробки запитів користувача, сервер також взаємодіє із БД за допомогою PrismaORM. Одним із найголовніших плюсів використання ORM (або OEM) є використання єдиної мови для розробки інформаційної системи. ORM мають власні механізми які забезпечують: надійність роботи з даними застосунку підтримуючи єдність типів; служать захисними механізмами, наприклад запобігають SQL-ін’єкціям тощо. Також вони надають зручні інструменти для роботи з БД, наприклад, у контексті кваліфікаційної роботи, було використано seed-файли які дозволяють швидко заповнити БД тестовими даними.

Таким чином, серверна частина системи забезпечує такі нефункціональні вимоги: масштабованість (модульна структура), безпеку

(використання middleware), безпеку даних системи та користувача (використання PrismaORM) та єдність технологічного стеку (використання ORM).

### 2.5.2 Діаграма розгортання інформаційної системи

Завершальним етапом проєктування інформаційної системи, як правило, є побудова діаграми розгортання. Ця діаграма відображає фізичне розгортання артефактів на вузлах, тобто відображає, які апаратні компоненти (вузли) існують, та які програмні компоненти (артефакти) працюють у кожному з вузлів. Діаграма розгортання поєднує в собі діаграму послідовності (див. рис. 2.6) й діаграму компонентів (див. рис. 2.8), та показує, як буде розгорнута система.

На рисунку 2.9 наведена діаграма розгортання для інформаційної системи, вона містить три вузли: User PC, Server API, DB (Docker).

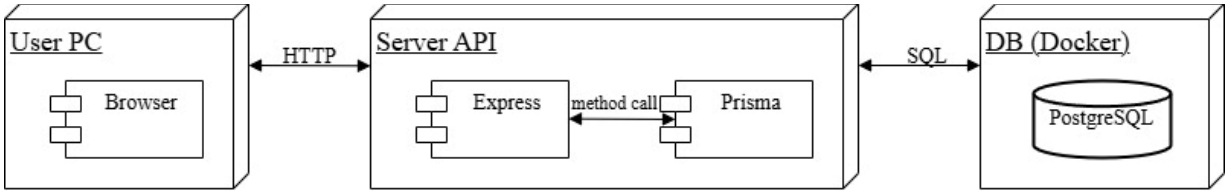



Рисунок 2.9 – Діаграма розгортання

Вузол User PC є клієнтом, тобто браузер у якому працює застосунок. Вузол Server API це серверна частина застосунку, до якої звертається клієнт.

Сервер надає маршрути для звернення клієнта, оброблює запити, за потреби взаємодіє з БД за допомогою Prisma. Вузол побудований з дотриманням монолітної архітектури, це знижує початкову складність розгортання системи. Також монолітна архітектура знижує час обробки даних різними модулями сервера. При збільшенні навантаження на систему, сервер можна масштабувати як вертикально, – шляхом апгрейду фізичного пристрою

на якому він працює (додаванням CPU/RAM); так і горизонтально, – шляхом створення екземплярів сервера на декількох машинах і розподілу навантаження між екземплярами за допомогою балансувальника навантажень.

База даних запускається у Docker-контейнері, на тому ж фізичному пристрої, що й сервер. Таке рішення забезпечує паритет оточень, мінімізує можливість конфліктів середовища, максимізує швидкість виконання операцій із БД. Використання Docker-контейнера дозволяє легко та швидко керувати БД, налаштовувати резервне копіювання даних, тестувати різні початкові дані тощо. За потреби для масштабування системи БД можна буде виокремити на інший сервер або сторонній сервіс без модифікування коду, просто змінивши рядок, який відповідає за підключення системи до БД.

## 2.6 Висновки другого розділу

В результаті проєктування системи були проаналізовані вимоги до системи, пройдений етап концептуального та фізичного проєктування.

На етапі аналізу вимог були висунуті функціональні й нефункціональні вимоги. На етапі концептуального проєктування був розроблений користувацький інтерфейс та змодельована діаграма послідовності, таким чином було продемонстровано як користувач взаємодіє із системою. На етапі фізичного проєктування було розроблено: діаграму компонентів, модель БД та діаграму розгортання. Таким чином, було розглянуто як система функціонує, як зберігає дані та як система буде розгорнена.

Сукупність пройдених етапів проєктування забезпечує готовність до початку розробки програмної реалізації інформаційної системи “InkFlow”.

# 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

Розроблений застосунок можна поділити на дві основні частини: серверний API та клієнтський вебзастосунок. Клієнтська частина – надає користувачу функціонал для управління системою. Серверна частина – обробляє запити, отримані від клієнта дані, взаємодіє із БД.

## 3.1 Огляд архітектури інформаційної системи

Клієнтська частина відповідає за маршрутизацію в середині SPA-застосунку, UI, процеси авторизації та автентифікації, відправку запитів та оновлення UI з урахуванням актуальних даних системи.

Технологічний стек виглядає наступним чином:

– JavaScript та React (UI);
– Editor.js (текстовий редактор);
– React Router v6 (маршрутизація);
– Auth0 (авторизація та автентифікація);
– Vite (збірка).

Розробка цієї частини системи починається із визначення маршрутів сторінок у конфігураційному файлі React Router. Ця бібліотека забезпечує декларативне описання маршрутів, асинхронне завантаження даних сторінки й зручне повторне використання створених сторінок. Конфігурація зображена на рисунку 3.1.

При розробці застосунку не використовувався менеджер загального стану, оскільки локального стану компонентів достатньо для розроблювальної системи, а використання додаткової бібліотеки не надало б суттєвих переваг.

Взаємодія з серверним API відбувається із використанням HTTP-запитів за технологією REST API (формат даних JSON).

```
const router = createBrowserRouter([
   {
      path: "/",
      Component: App,
      children: [
         { index: true, Component: HomePage },
         {
            path: "createpost",
            Component: CreatePost,
         },
         {
            path: "post/:id",
            Component: PostView,
         },
         {
            path: "profile",
            Component: ProfilePage,
         },
         {
            path: "drafts",
            Component: DraftsPage,
         },
         {
            path: "*",
            Component: HomePage,
         },
      ]
   }
])
```

Рисунок 3.1 – Описання маршрутів застосунку

Серверна частина застосунку побудована за модульною архітектурою. Вона сприяє кращій масштабованості застосунку у майбутньому. Сервер надає послуги обробки запитів клієнта, працює з БД, перевіряє токен користувача, виконує збір даних за посиланням, а також зберігає зображення.

Сервер розроблено як моноліт, усі модулі виконуються в межах єдиного серверного процесу. Така система добре підтримує як горизонтальне, так і вертикальне масштабування, тому що не зберігає дані сесії, а отримує їх із кожним запитом клієнта (є stateless системою). Таким чином, можна розгорнути декілька екземплярів сервера й підключити балансувальник навантажень для рівномірного розподілу навантаження.

## 3.2 Огляд програмної реалізації ключових сценаріїв

Цей пункт буде присвячений безпосередньо огляду програмної реалізації розробленої інформаційної системи: будуть розглянуті основні сценарії використання системи користувачем, продемонстрований розроблений інтерфейс користувача й програмний код розробленого застосунку.

### 3.2.1 Сценарій авторизації та автентифікації

Розроблена інформаційна система, згідно до функціональних вимог, надає можливість створення облікового запису користувача. Процес авторизації та автентифікації був делегований надійному сервісу Auth0.

Сценарій реєстрації виглядає наступним чином: користувач уперше заходить на сторінку. Без облікового запису йому доступний перегляд дописів, фільтрація та сортування. Якщо користувач хочу зареєструватися, йому потрібно скористатися кнопкою “Sign Up” у верхньому куту шапки застосунку (див. рис. 3.2).

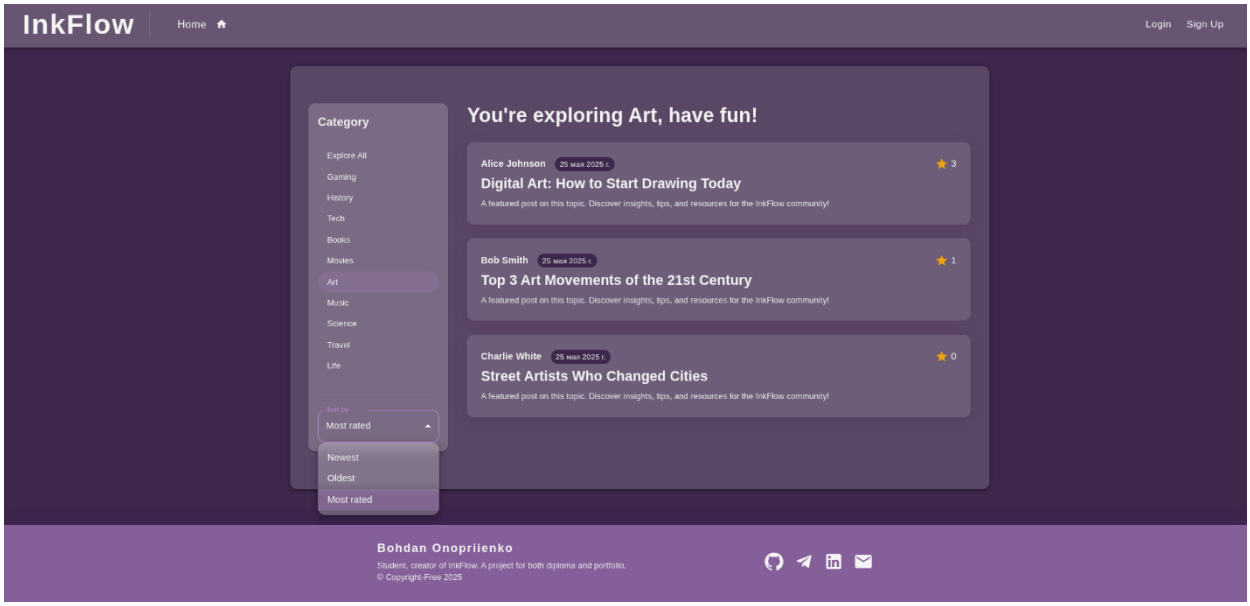



Рисунок 3.2 – Головний екран застосунку

Після чого, користувач буде перенаправлений на сторінку реєстрації Auth0. Програмна реалізація шапки й футеру сайту знаходиться у головному файлі застосунку App.jsx, це один із способів відобразити їх на кожній сторінці застосунку. Кнопки реєстрації або логіну використовують надані сервісом інструменти: стан користувача й хук, який автоматично перенаправляє на сторінку авторизації Auth0 (див. рис. 3.3).

```
{!isAuthenticated ? ( <Stack direction="row" spacing={1} sx={{ "& button": { fontSize: 17, fontWeight: 500 } }}>
<Button
onClick={() => loginWithRedirect()}
color="inherit" > Login </Button>
<Button
onClick={() => loginWithRedirect()}
color="inherit"> Sign Up </Button>
</Stack> )
```

Рисунок 3.3 – Використання стану та хука Auth0

Стан isAuthenticated виступає як умова, залежно від нього для користувача будуть відображені або кнопки логіну, або меню профіля користувача. Хук loginWithRedirect надається сервісом і працює як звичайна функція. Після того, як користувач пройде авторизацію у сервісі Auth0, він буде перенаправлений на головну сторінку застосунку, система отримає токен доступу й збереже його в localStorage.

Для збереження та оновлення токену доступу була розроблена окрема функція, вона дозволяє оновлювати токен доступу користувача асинхронно коли це потрібно (див. рис. 3.4).

Функція викликається у різних частинах застосунку за потреби, якщо токен більше не дійсний, вона використає хук getAccessTokenSilently для асинхронного оновлення, у випадку якщо це з якоїсь причини не вдасться, користувач буде автоматично перенаправлений для повторної авторизації.

```
import { useAuth0 } from "@auth0/auth0-react";
import { useEffect } from "react";
const AuthTokenHandler = () => {
   const { isAuthenticated, getAccessTokenSilently } = useAuth0();
   useEffect(() => {
      const saveToken = async () => {
         try {
            if (isAuthenticated) {
               const token = await getAccessTokenSilently();
               localStorage.setItem("access_token", token);
            } else {
               localStorage.removeItem("access_token");
            }
         } catch (error) {
            console.error("Error while saving token:", error);
         }
      };
      saveToken();
   }, [isAuthenticated, getAccessTokenSilently]);
   return null;
};
export default AuthTokenHandler;
```

Рисунок 3.4 – Функція AuthTokenHandler для оновлення токену користувача

Після авторизації у системі, користувач отримує доступ до всього функціоналу застосунку. Також тепер стан isAuthenticated повертає “true” й інтерфейс застосунку відображає додаткові елементи системи: кнопка профілю користувача, навігаційне меню (див. рис. 3.5).

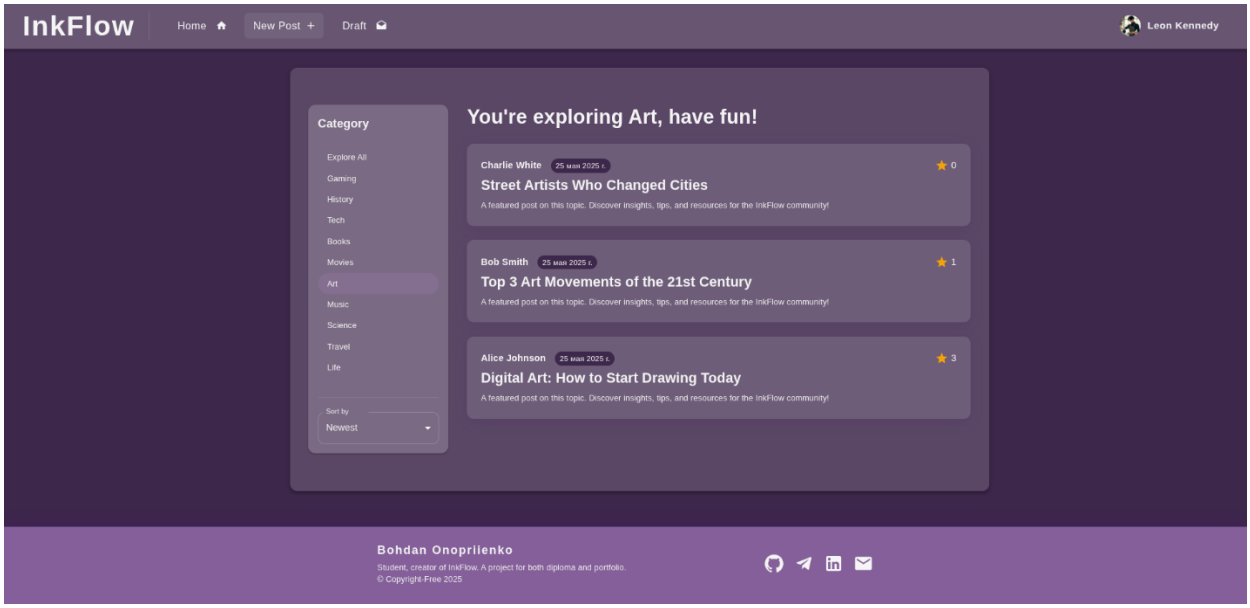



Рисунок 3.5 – Головна сторінка після авторизації

### 3.2.2 Сценарій управління дописами

Для авторизованого в системі користувача стає доступним функціонал управління дописами: він виглядає наступним чином користувач натискає кнопку “New Post” у навігаційному меню, після чого спрацьовує подія onClick, і за допомогою React Router система відтворює сторінку CreatePost (див. рис. 3.6).

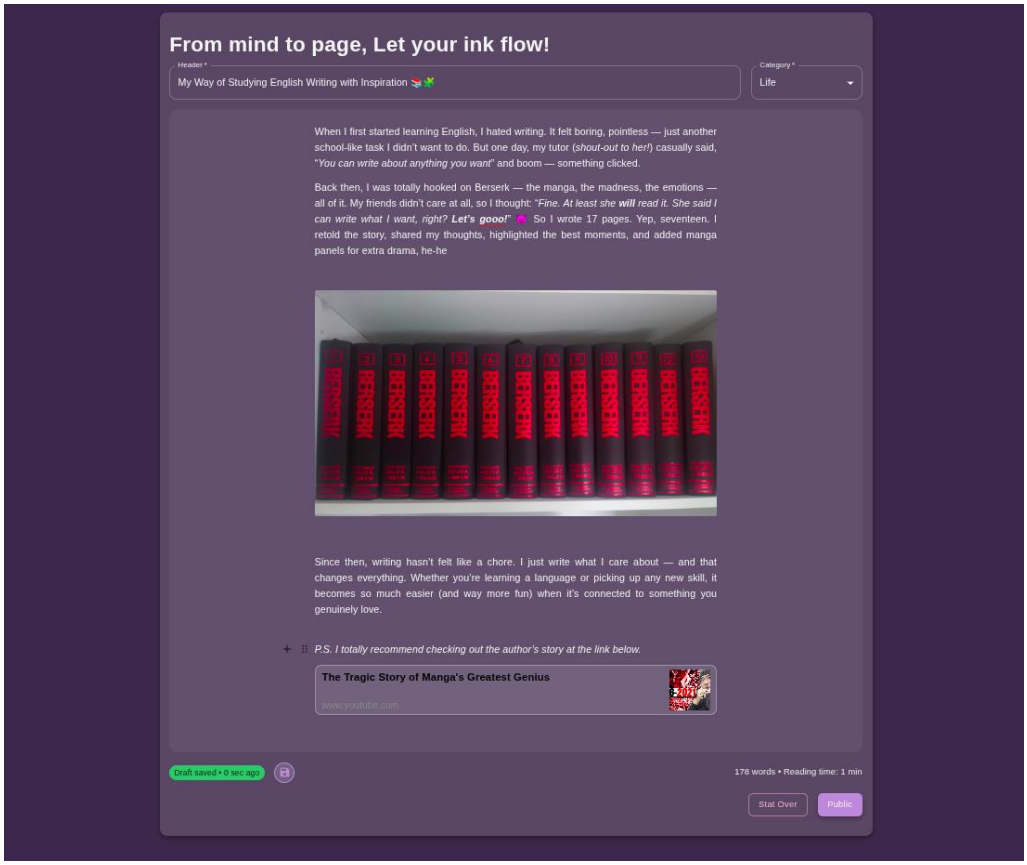



Рисунок 3.6 – Приклад створення допису

На сторінці користувачу доступна форма створення допису: вона має такі поля: текстове поле заголовок, меню вибору категорії, та основне поле редактору. Він підтримує семантичний розподіл контенту: параграф, заголовок, список, посилання, зображення тощо. Редактор підтримує усі класичні способи форматування та вирівнювання тексту.

Для створення кращого користувацького досвіду була розроблена система збереження чернетки, зберігання відбувається у localStorage за

таймером або після натискання відповідної кнопки UI. Також присутня кнопка очистки всіх полей “Start Over”.

В процесі створення допису, користувач може завантажувати зображення зі свого пристрою, а також додавати посилання на сторонні ресурси.

Для можливості завантаження зображень та створення прев’ю посилань на сервері були розроблені відповідні модулі.

Процес обробки та збереження зображення виглядає наступним чином: на клієнті в редакторі підключається додатковий модуль та визначається маршрут на який відправиться зображення (див. рис. 3.7).

```
image: {
  class: ImageTool, config: {
    field: 'image', endpoints: { byFile: 'http://localhost:4000/api/upload',}, },
  }
```

Рисунок 3.7 – Модуль редактора ImageTool

Сервер приймає це зображення, перевіряє розширення файлу, надає йому унікальне ім’я та зберігає у папку public/static. Після чого, повертає на клієнт URL для використання зображення. Цей URL, після того, як користувач опублікує допис, збережеться у БД разом із усією інформацією про допис. На рисунку 3.8 зображено сервіс, який відповідає за завантажені зображення.

```
class FileService {
    saveFile(file) {
        const allowedTypes = ['.jpg', '.jpeg', '.png', '.webp'];
        const ext = path.extname(file.name).toLowerCase();
        if (!allowedTypes.includes(ext)) throw new Error('Unsupported file type');
        const fileName = uuidv4() + ext;
        const filePath = path.resolve('public/static', fileName);
        fs.mkdirSync(path.dirname(filePath), { recursive: true });
        return new Promise((resolve, reject) =>  file.mv(filePath, err => err ? reject(err) :
        resolve(fileName)) );
```

Рисунок 3.8 – Сервіс обробки зображень

Обробка посилань відбувається схожим чином, – у редакторі Editor.js підключається модуль посилань та визначається маршрут на серверний API. Після завантаження посилання відбувається запит на сервер, у модуль який збирає meta-дані за наданим посиланням (див. рис. 3.9).

```
const html = await fetch(url).then(r => r.text())
const root = parse(html)
const getMeta = (name) =>
   root.querySelector(`meta[property="${name}"]`)?.getAttribute('content') ||
   root.querySelector(`meta[name="${name}"]`)?.getAttribute('content')
const getFavicon = () => {
const iconLink = root.querySelector('link[rel="icon"]')?.getAttribute('href') ||
   root.querySelector('link[rel="shortcut icon"]')?.getAttribute('href') || '/favicon.ico'
   try { return new URL(iconLink, url).href } catch { return '' } }
```

Рисунок 3.9 – Сервіс обробки посилання

Після збору meta-даних модуль повертає на клієнт відповідь у форматі який очікує редактор Editor.js (див. рис. 3.10).

```
res.json({
        success: 1,
        meta: {
           title: getMeta('og:title') || root.querySelector('title')?.text || url,
           image: {
              url: getMeta('og:image') || '',
           },
           favicon: getFavicon()
        },
      })
```

Рисунок 3.10 – Відповідь модуля обробки посилань на клієнт

Коли користувач готовий опублікувати допис і натискає “Public”, запит обробляється та надсилається на сервер за допомогою AXIOS. Для

забезпечення безпеки системи було створено інтерцептор, він виконує перевірку токену доступу на клієнті, тільки якщо з токеном усе добре, – надсилає запит на сервер (сервер також перевіряє токен). Код інтерцептора продемонстрований на рисунку 3.11.

```
import axios from "axios";
const api = axios.create({ baseURL: "http://localhost:4000/api" });
api.interceptors.request.use( (config) => {
      const token = localStorage.getItem("access_token");
      if (token && config.requiresAuth !== false) {
         config.headers.Authorization = `Bearer ${token}`;
      } return config; },
   (error) => return Promise.reject(error););
export default api;
```

Рисунок 3.11 – Інтерцептор захищених запитів

Інтерцептор розроблений так, щоб перевіряти наявність токену для кожного запиту, якщо у запиті явно не вказано, що це не потрібно робити. Запити також винесені в окремі модулі для зручного керування, вони описані у файлі api/axios/.

Після надсилання запиту його оброблює відповідний модуль “post”: сервіс створення допису зображено на рисунку 3.12. Сервіс перевіряє наявність усіх необхідних даних для створення запису у БД, після чого створює його.

```
export const createPost = async ({ title, content, categoryId, authorId, userName }) => {
   if (!title || !content || !categoryId || !authorId || !userName) {
      throw new Error('Missing required fields');
   }
   await ensureUserExists(authorId, userName);

   const slug = slugify(title, {
      lower: true,
      strict: true,
      trim: true,
   });
```

Рисунок 3.12 – Свервіс створення допису, аркуш 1

```
    return prisma.post.create({
        data: {
            title,
            content,
            slug,
            category: {
                connect: { id: categoryId },
            },
            author: {
                connect: { id: authorId },
            },
        },
    });
};
```

Рисунок 3.12 – Свервіс створення допису, аркуш 2

На клієнті після надсилання запиту на створення запису, й за умови, якщо створення пройшло успішно, система перенаправляє користувача на сторінку тільки що створеного допису, приклад зображено на рисунку 3.13.

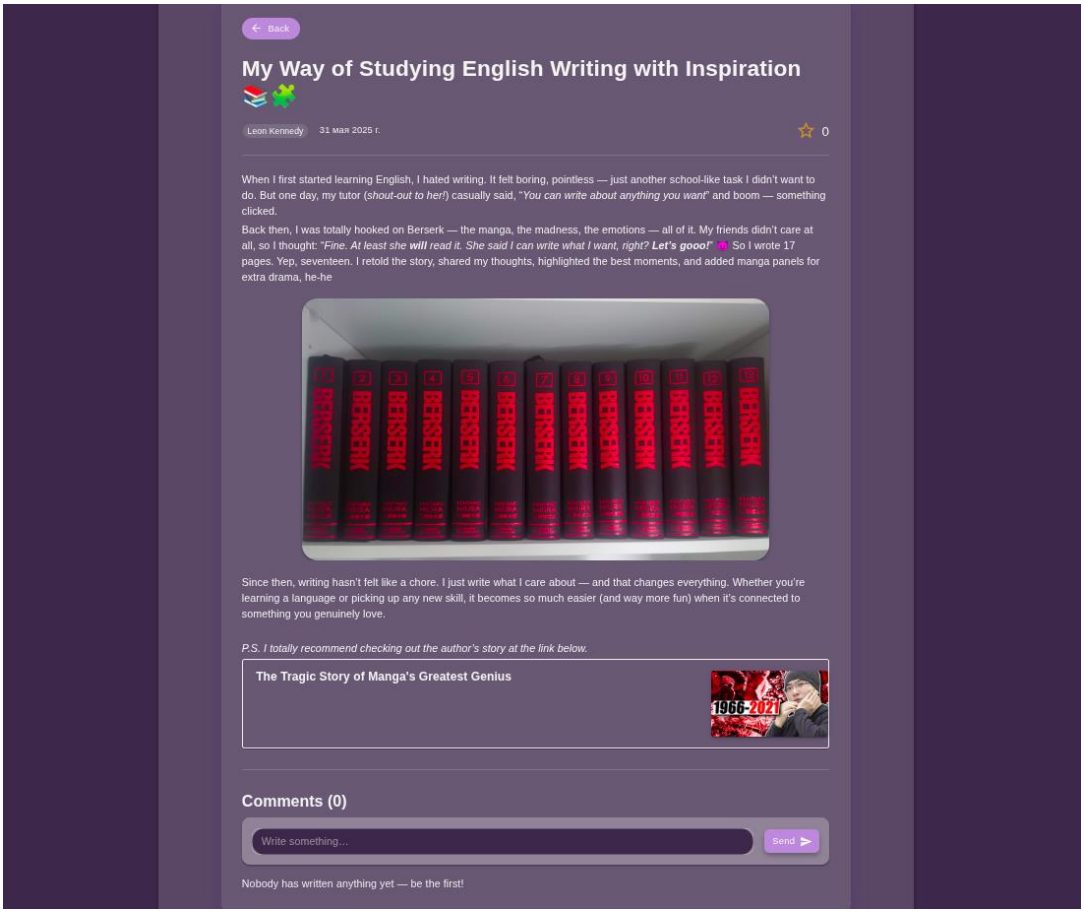



Рисунок 3.13 – Сторінка перегляду допису

На цій сторінці користувач може переглянути допис, дату публікації та автора, за бажанням поставити оцінку. Також нижче допису зроблено секцію коментарів. Вони підтримують систему відповідей (див. рис. 3.14).

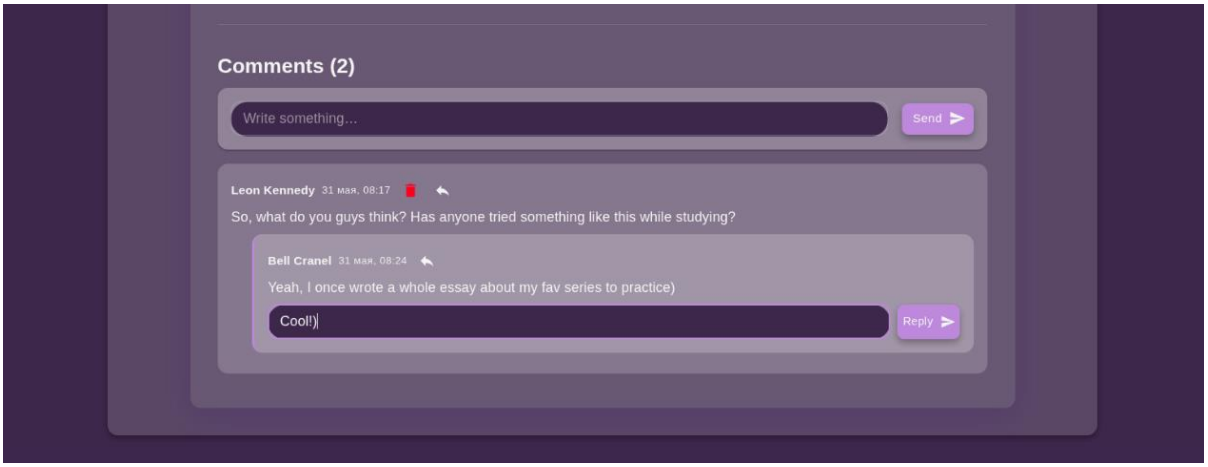



Рисунок 3.14 – Секція коментарів

Також у користувача є можливість переглядати свою статистику та керувати дописами, переглядати чернетку тощо. Для цього було створено сторінку ProfilePage; вона зображена на рисунку 3.15.

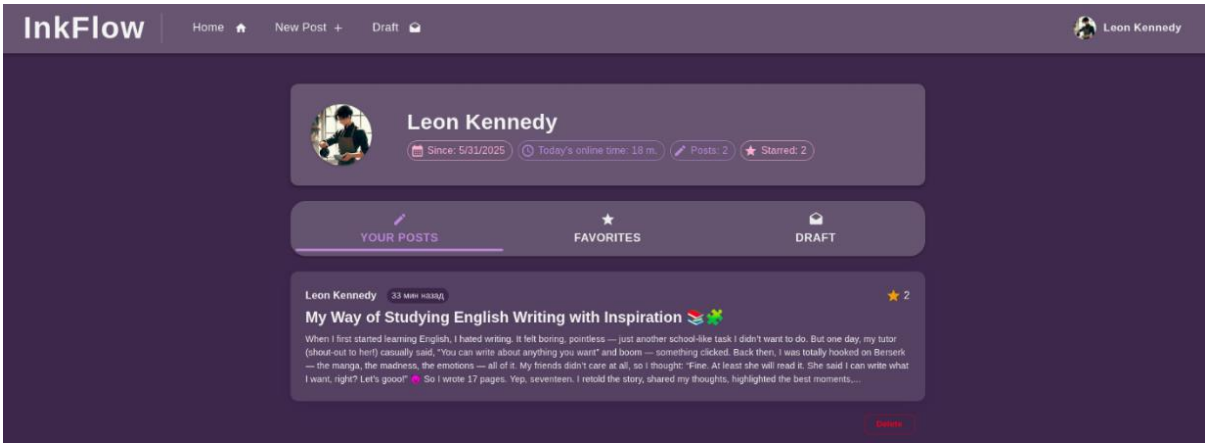



Рисунок 3.15 – Сторінка перегляду профілю

Користувач може переглянути дату реєстрації, тривалість поточної сесії, кількість написаних та обраних дописів. Також на цій сторінці користувач може видаляти власні дописи. Для забезпечення випадкових натискань було розроблено вікно підтвердження видалення (див. рис. 3.16).

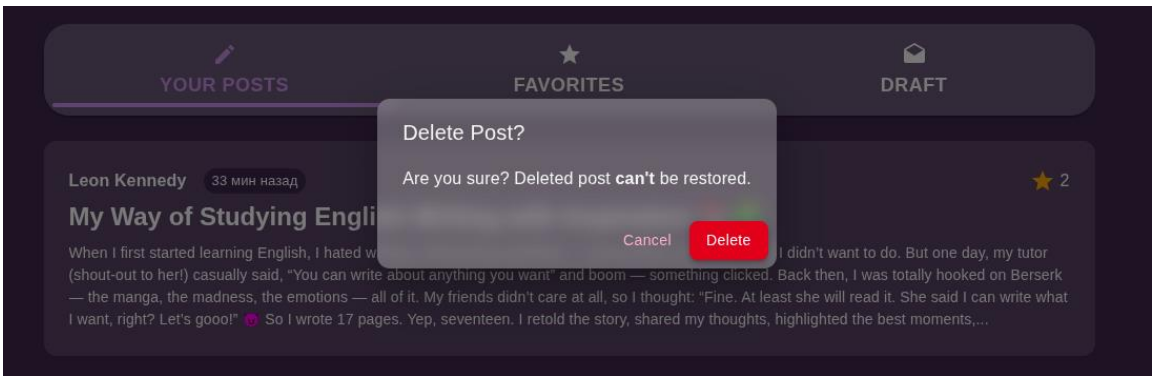



Рисунок 3.16 – Вікно підтвердження видалення допису

# ВИСНОВКИ

У кваліфікаційній роботі було розроблено онлайн-платформу для ведення блогів “InkFlow”. Для розробки вебзастосунку було використано мову програмування JavaScript та бібліотеку React для побудови інтерфейсу, середовище Node.js з фреймворком Express для реалізації серверної частини. Додатково використано бібліотеку React Router для маршрутизації та сервіс Auth0 для авторизації та автентифікації. Зберігання даних відбувалося за допомогою СКБД PostgreSQL разом із PrismaORM. База даних була розгорнута за допомогою Docker Compose контейнера, для забезпечення гнучкого керування середовищем.

У роботі було проведено аналіз предметної області, визначено вимоги до системи, здійснено порівняння програмних засобів для розробки та обґрунтовано вибір технологій. Структуру системи та варіанти використання продемонстровано на діаграмі акторів та прецедентів, послідовності, схемі даних бази даних, діаграмі компонентів та розгортання, також розроблено макети інтерфейсу користувача. У третьому розділі наведено опис розробленої системи, продемонстровано її архітектуру, а також програмну реалізацію ключових сценаріїв використання.

Серед переваг розробленої інформаційної системи слід відзначити високий ступінь гнучкості та можливість подальшого масштабування. Завдяки модульній структурі застосунку можна легко впроваджувати новий функціонал, а наявність інтерцепторів та проміжних обробників забезпечує високий рівень безпеки.

Як приклад подальшого розвитку, можливо швидко виокремити обробку зображень на CDN-сервіс, а зі збільшенням навантаження на застосунок можна легко запустити ще один екземпляр серверного застосунку.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Gavrilă V., Băjenaru L., Dobre C. Modern Single Page Application Architecture: A Case Study. *Studies in Informatics and Control*. 2019. Vol. 28, no. 2. P. 231-238 URL: https://doi.org/10.24846/v28i2y201911 (дата звернення: 15.04.2025).

2. ECMAScript Language Specification. *ECMA Internetional*. URL: https://tc39.es/ecma262/ (дата звернення: 15.04.2025).

3. Node.js v24.2.0 documentation. *Node.js*. URL: https://nodejs.org/en/docs (дата звернення: 18.04.2025).

4. Vivek J. Server-Side Rendering vs. Client-Side Rendering: A Comprehensive Analysis. *International Journal of Innovative Research and Creative Technology*. 2021. Vol. 7, Issue 2. P. 1-5. URL: https://doi.org/10.5281/zenodo.14752604 (дата звернення: 18.04.2025).

5. Sekhar Emmanni P. Comparative Analysis of Angular, React, and Vue.js in Single Page Application Development. *International Journal of Science and Research (IJSR).* 2023. Vol. 12, Issue 6. P. 2971-2974. URL: https://doi.org/10.21275/sr24401230015 (дата звернення: 23.04.2025).

6. React – The library for web and native user interfaces. *React*. URL: https://reactjs.org/ (дата звернення: 23.04.2025).

7. Kubal S. M. React.js: A Premier Frontend Solution. *International Research Journal of Modernization in Engineering Technology and Science*. 2024. Vol. 6, Issue 7. P. 3559-3561. URL: https://www.doi.org/10.56726/IRJMETS60634 (дата звернення: 24.04.2025).

8. Defining a class component. *Component – React*. URL: https://react.dev/reference/react/Component#defining-a-class-component (дата звернення: 25.04.2025).

9. Das K. H. Node JS vs PHP: Which Backend is Right for Your Next Project in 2025? *Webandcrafts*. URL: https://webandcrafts.com/blog/node-js-vs-php (дата звернення: 05.05.2025).

10. Churcher C. Beginning Database Design. New York : Apress, 2012. 221 p.

# ДОДАТОК А

## Код Docker Compose контейнера

```
services:
  postgresql:
    image: postgres:16
    container_name: psql-app
    restart: always
    environment:
      POSTGRES_DB: ${POSTGRES_DB}
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    ports:
      - ${DB_PORT}:5432
    volumes:
      - inkflow_postgres_data:/var/lib/postgresql/data
    networks:
      - inkFlow

networks:
  inkFlow:

volumes:
  inkflow_postgres_data:
```