МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Острозька академія»

Навчально-науковий інститут інформаційних технологій та бізнесу

Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

на тему: «Розробка жіночого блогу "Щоденник краси"»

Виконала: студентка 4 курсу, групи КН-41
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Антонюк Карина Юріївна

Керівник: викладач кафедри інформаційних технологій
та аналітики даних
Мацевич Денис Володимирович

Рецензент: кандидат технічних наук, доцент,
доцент кафедри прикладної математики
Донецького національного університету
імені Василя Стуса
Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри інформаційних технологій та аналітики даних

_____ (проф., д.е.н. Кривицька О.Р.)

Протокол № 11 від «20» травня 2026 р.

Острог, 2026

АНОТАЦІЯ

кваліфікаційної роботи

на здобуття освітнього ступеня бакалавра

Тема: Проєктування та розробка вебзастосунку «Щоденник краси» – інтерактивного жіночого блогу з елементами соціальної мережі

Автор: Антонюк Карина Юріївна

Науковий керівник: Мацевич Денис Володимирович, викладач кафедри інформаційних технологій та аналітики даних

Захищена «.....»..... 20__ року.

Пояснювальна записка до кваліфікаційної роботи: ____ с., ____ рис., ____ табл., ____ додатків, ____ джерел.

Ключові слова: FULL-STACK РОЗРОБКА, NEXT.JS, NESTJS, POSTGRESQL, CQRS, WEBSOCKET, REALTIME-СПОВІЩЕННЯ, ЖІНОЧИЙ БЛОГ, JWT, OTP.

Короткий зміст праці:

У кваліфікаційній роботі розглянуто проєктування та розробку вебзастосунку «Beauty Diary» – інтерактивного жіночого блогу, що поєднує публікацію постів про красу та wellness з елементами соціальної мережі (коментарі, лайки, підписки, сповіщення). Актуальність теми зумовлена відсутністю платформи, яка б одночасно забезпечувала розгорнуті текстові пости та соціальну взаємодію в реальному часі.

Для реалізації клієнтської частини використано Next.js 15, серверної – NestJS 11 із гексагональною архітектурою, CQRS, WebSocket (Socket.IO) та

outbox-патерном для email. База даних – PostgreSQL 16. Спільний пакет Zod-схем забезпечує єдиний контракт між фронтендом та бекендом.

Практично реалізовано: OTP-реєстрацію, створення постів з обкладинкою (Cloudflare R2), коментування, лайки, підписки, повнотекстовий пошук, real-time сповіщення, адмін-панель. Застосунок розгорнуто на Vercel, Render, Neon. Налаштовано CI/CD через GitHub Actions.

У результаті створено готовий до експлуатації вебзастосунок для ведення блогу та взаємодії з контентом у реальному часі.

_____

(підпис автора)

ANNOTATION

of qualification paper

for bachelor's degree

Topic: Design and Development of «Beauty Diary» Web Application – an Interactive Women's Blog with Social Networking Elements

Author: Antoniuk Karyna Iuriivna

Supervisor: Denys Matsievych, Lecturer at the Department of Information Technologies and Data Analytics

Defended on: «.....» ..... 20__

Explanatory note of the qualification work: ____ pages, ____ figures, ____ tables, ____ appendices, ____ sources.

Keywords: FULL-STACK DEVELOPMENT, NEXT.JS, NESTJS, POSTGRESQL, CQRS, WEB SOCKET, REALTIME NOTIFICATIONS, WOMEN'S BLOG, JWT, OTP.

Summary:

This qualification work examines the design and development of the «Beauty Diary» web application – an interactive women's blog that combines beauty and wellness posts with social networking elements (comments, likes, subscriptions, notifications). The relevance of the topic is determined by the lack of a platform that simultaneously provides rich text posts and real-time social interaction.

The client-side is implemented using Next.js 15. The server-side is built with NestJS 11 utilizing hexagonal architecture, CQRS pattern, WebSocket gateway (Socket.IO) for real-time notifications, and outbox pattern for reliable email delivery. PostgreSQL 16 is

used as the database. A shared package of Zod schemas ensures a single contract between frontend and backend.

The practical implementation includes: OTP email verification registration, post creation and publication with cover images (upload to Cloudflare R2), threaded comments with soft-delete, likes, favorites, author subscriptions, full-text search, real-time updates via WebSocket, email notifications through outbox queue with retries, and an admin panel for user blocking and audit log viewing.

As a result, a production-ready web application is created that allows users to maintain a blog, interact with others' content, and receive real-time notifications.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API	Application Programming Interface (програмний інтерфейс застосунку)
CI/CD	Continuous Integration / Continuous Deployment (безперервна інтеграція та безперервне розгортання)
CORS	Cross-Origin Resource Sharing (обмін ресурсами між різними джерелами)
CQRS	Command Query Responsibility Segregation (розділення відповідальності команд та запитів)
CRUD	Create, Read, Update, Delete (створення, читання, оновлення, видалення)
DI	Dependency Injection (впровадження залежностей)
DOM	Document Object Model (об'єктна модель документа)
DTO	Data Transfer Object (об'єкт передачі даних)
E2E	End-to-End (наскрізне тестування)
ER	Entity-Relationship (сутність-зв'язок)
Git	Distributed version control system (розподілена система керування версіями)
HMR	Hot Module Replacement (гаряча заміна модулів)
HMAC	Hash-based Message Authentication Code (код автентифікації повідомлень на основі хешу)
HTML	HyperText Markup Language (мова розмітки гіпертексту)
HTTP	HyperText Transfer Protocol (протокол передачі гіпертексту)
HTTPS	HyperText Transfer Protocol Secure (захищений протокол передачі гіпертексту)
IDE	Integrated Development Environment (інтегроване середовище розробки)
i18n	Internationalization (інтернаціоналізація)
JWT	JSON Web Token (вебтокен у форматі JSON)

OTP	One-Time Password (одноразовий пароль)
ORM	Object-Relational Mapping (об'єктно-реляційне відображення)
RBAC	Role-Based Access Control (керування доступом на основі ролей)
REST	Representational State Transfer (передача стану представлення)
RSC	React Server Component (серверний компонент React)
S3	Simple Storage Service (простий сервіс зберігання)
SLA	Service Level Agreement (угода про рівень обслуговування)
SMTP	Simple Mail Transfer Protocol (простий протокол передачі пошти)
SPA	Single Page Application (односторінковий застосунок)
SQL	Structured Query Language (структурована мова запитів)
SSD	Solid-State Drive (твердотільний накопичувач)
SSR	Server-Side Rendering (рендеринг на стороні сервера)
UI	User Interface (користувацький інтерфейс)
UML	Unified Modeling Language (уніфікована мова моделювання)
URL	Uniform Resource Locator (уніфікований локатор ресурсу)
UX	User Experience (користувацький досвід)
WebSocket	Протокол повнодуплексного зв'язку через одне TCP-з'єднання
БД	База даних
ПЗ	Програмне забезпечення
СКБД	Система керування базами даних

ВСТУП

У сучасному цифровому суспільстві споживання контенту зазнало фундаментальних змін: пасивне читання поступово трансформується в активну соціальну взаємодію. Особливо яскраво цей тренд проявляється в жіночій аудиторії, яка є однією з найактивніших у сегменті lifestyle-блогінгу. Сьогодні користувачі потребують не просто платформи для публікації постів, а інтегрованого середовища, що поєднує ведення особистого щоденника, читання авторських матеріалів, коментування, лайкання, збереження улюбленого контенту та отримання сповіщень про активність інших користувачів у реальному часі.

Актуальність теми кваліфікаційної роботи зумовлена відсутністю на ринку універсальної платформи, яка б одночасно забезпечувала створення розгорнутих текстових постів із багатим форматуванням (WYSIWYG-редактор, обкладинки, теги, категорії) та повноцінну соціальну взаємодію (коментарі з гілками, лайки, підписки на авторів, збереження контенту, сповіщення в реальному часі). Існуючі рішення, такі як Medium, Pinterest або Instagram, зосереджені або суто на довгих текстах без розвиненої соціальної складової, або на візуальному контенті з обмеженими можливостями для блогової активності.

Крім того, з технічного боку спостерігається потреба у створенні сучасного full-stack вебзастосунку, який би використовував передові технології (Server Components, WebSocket, CQRS, outbox-патерн) для забезпечення високої продуктивності, масштабованості та безпеки.

Метою кваліфікаційної роботи є розробка вебзастосунку «Beauty Diary» – інтерактивного жіночого блогу з елементами соціальної мережі, який забезпечує публікацію постів, соціальну взаємодію (коментарі, лайки, підписки, збереження) та real-time сповіщення.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- провести аналіз предметної області та існуючих аналогів у сфері жіночих блогів та соціальних мереж;
- сформулювати функціональні та нефункціональні вимоги до вебзастосунку;

- обґрунтувати вибір технологічного стеку для реалізації клієнтської та серверної частин;
- спроєктувати архітектуру застосунку (гексагональна архітектура, CQRS, outbox-патерн);
- розробити базу даних (ER-діаграма, міграції, тригери для денормалізації лічильників);
- реалізувати бекенд на NestJS з підтримкою REST API та WebSocket;
- реалізувати фронтенд на Next.js (App Router, Server Components, Tailwind, shadcn/ui);
- створити спільний пакет Zod-схем для єдиного контракту між фронтендом та бекендом;
- забезпечити безпеку даних (OTP-реєстрація, JWT в httpOnly cookies, refresh-токени з reuse detection, argon2id);
- налаштувати CI/CD (GitHub Actions) та деплой на хмарні платформи (Vercel, Render, Neon, R2);
- провести тестування основних сценаріїв (unit, інтеграційне).

Об'єктом дослідження є процес створення та функціонування інтерактивних блогів з елементами соціальної мережі, включаючи публікацію контенту, соціальні взаємодії та систему сповіщень.

Предметом дослідження є методи та інструменти розробки full-stack вебзастосунку з використанням Next.js, NestJS, PostgreSQL, WebSocket, CQRS та outbox-патерну.

Методи дослідження. У роботі використано методи системного аналізу для дослідження предметної області, порівняльного аналізу для вибору технологій, об'єктно-орієнтованого проєктування для розробки архітектури ПЗ та емпіричні методи тестування для перевірки працездатності реалізованих модулів.

Практичне значення отриманих результатів. Розроблений вебзастосунок «Beauty Diary» є готовим до експлуатації продуктом, який дозволяє користувачам вести блог про красу, догляд за шкірою, стиль та wellness, взаємодіяти з контентом інших авторів та отримувати сповіщення в реальному часі

РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ІНТЕРАКТИВНОГО БЛОГУ З ЕЛЕМЕНТАМИ СОЦІАЛЬНОЇ МЕРЕЖІ

1.1. Постановка проблеми, опис базових понять бізнес-процесу

У сучасному цифровому суспільстві споживання контенту трансформувалося з пасивного читання в активну соціальну взаємодію. Жіноча аудиторія, яка є однією з найактивніших у сегменті lifestyle-блогінгу, потребує не просто платформи для публікації постів, а інтегрованого середовища, що поєднує ведення особистого щоденника, читання авторських матеріалів, коментування, лайкання, збереження улюбленого контенту та отримання сповіщень про активність інших користувачів.

Основна науково-технічна проблема полягає в необхідності створення вебзастосунку, який одночасно забезпечує:

- багатокористувацький блог з постами (багатим текстом, обкладинками, тегами, категоріями);
- соціальні взаємодії (лайки, коментарі з репліками, підписки на авторів, збереження постів);
- real-time сповіщення (in-app через WebSocket + email через асинхронну чергу);
- безпечну реєстрацію з підтвердженням email (OTP-код, захист від ботів);
- адаптивний інтерфейс з підтримкою темної/світлої теми.

Аналіз таблиці 1.1 показує, що світовий ринок контент-платформ з елементами соціальної взаємодії зростає на 15–20% щорічно, причому частка мобільних пристроїв у 2025 році перевищила 70% [1].

Таблиця 1.1

Динаміка використання блог-платформ із соціальними функціями

Рік	**Глобальна аудиторія (млн)**	**Частка переглядів із мобільних пристроїв (%)**	**Середній час сесії (хв)**
2022	850	58	6,2
2023	1020	64	7,1
2024	1240	68	8,3

Рік	Глобальна аудиторія (млн)	Частка переглядів із мобільних пристроїв (%)	Середній час сесії (хв)
2025	1520	72	9,5

Джерело: складено автором на основі даних Statista, 2026 [1]

Аналіз сучасних тенденцій у жіночому блогінгу. Жіноча аудиторія становить понад 60% активних користувачів lifestyle-блогів [1] та соціальних мереж, що робить її однією з найвпливовіших у цифровому просторі. Дослідження показують, що жінки значно активніше взаємодіють з контентом: вони частіше коментують, ставлять лайки, діляться постами та підписуються на авторів. Це створює унікальний попит на платформи, які не просто дозволяють публікувати текст, але й забезпечують глибоку соціальну залученість. Beauty Diary народжується саме з цієї потреби – об'єднати формат особистого щоденника, де можна вільно висловлювати думки про красу, догляд та стиль, із можливістю миттєво отримувати зворотний зв'язок від спільноти.

Технічні виклики створення гібридної платформи. Створення такого гібридного застосунку супроводжується низкою технічних викликів. По-перше, це необхідність підтримувати багатоформатний контент – тексти з багатим форматуванням, зображення, теги, категорії. По-друге, це масштабування соціальних взаємодій: кожен пост може отримати сотні коментарів та лайків, які потрібно обробляти миттєво. По-третє, це система сповіщень, яка має працювати без затримок, але при цьому не перевантажувати сервер. По-четверте, це безпека – захист персональних даних користувачів, паролів, токенів. Beauty Diary вирішує всі ці виклики за допомогою сучасного технологічного стеку: Next.js для швидкого фронтенду, NestJS для надійного бекенду, PostgreSQL для структурованих даних та WebSocket для real-time комунікації.

Чому важлива OTP-верифікація. Однією з ключових проблем сучасних блог-платформ є велика кількість ботів, які реєструються для розсилання спаму. Звичайна реєстрація за email та паролем не захищає від цього – боти можуть створювати тисячі акаунтів автоматично. Тому в Beauty Diary реалізовано OTP-верифікацію (одноразовий пароль, який надсилається на email). Користувач не

просто вводить email, але й підтверджує його, вводячи 6-значний код. Це значно ускладнює автоматичну реєстрацію, оскільки бот мав би ще й отримувати доступ до поштової скриньки. Крім того, OTP-верифікація підвищує довіру до платформи – інші користувачі знають, що за кожним акаунтом стоїть реальна людина з реальною email-адресою.

1.2. Огляд існуючих рішень та аналіз сайтів-аналогів

Для формування вимог до власного застосунку було проведено порівняльний аналіз трьох популярних платформ, які частково покривають функціонал Beauty Diary.

1.2.1. Аналіз вебресурсу «Medium»

Medium – це платформа для довгих текстових публікацій із можливістю рекомендувати статті (аналог лайка – «claps») та підписуватися на авторів. Основні сильні сторони: мінімалістичний дизайн, зосередженість на читанні, система рекомендацій. Недоліки: відсутність гіллястих коментарів, відсутність real-time сповіщень у дзвіночку, немає можливості зберігати пости в окремому списку. Для Beauty Diary цінним є досвід простоти публікації та чистоти типографіки.

Рис. 1.2 – Головна сторінка платформи Medium (скріншот автора)

1.2.2. Аналіз вебресурсу «Pinterest»

Pinterest орієнтований на візуальне збереження ідей («піни»), створення дошок та взаємодію через лайки/коментарі. Сильні сторони: потужна система збереження контенту, алгоритми рекомендацій. Недоліки для блогової моделі: слабкий текстовий редактор, обмежені можливості для коментування та підписок на авторів у класичному розумінні.

1.2.3. Аналіз вебресурсу «Instagram»

Instagram є найпопулярнішою соціальною мережею, що базується на візуальному контенті, реакціях, коментарях, прямих повідомленнях. Однак для довгих текстових блогів він не призначений, має обмеження на довжину підпису до

постів, відсутність кастомних категорій та тегів. Для Beauty Diary ключовим запозиченням є архітектура real-time сповіщень та робота з підписками.

1.2.4. Висновки за підпунктом 1.2

Таблиця 1.2

Порівняльна характеристика функціоналу аналогів

Функція	**Medium**	**Pinterest**	**Instagram**	**Beauty Diary (вимога)**
Розгорнуті текстові пости	Так	Ні	Ні	Так
Коментарі (гілки)	Ні	Обмежені	Так	Так
Лайки	Claps	Так	Так	Так
Збереження постів	Так	Так	Так	Так
Підписки на авторів	Так	Так	Так	Так
Real-time сповіщення	Ні	Ні	Так	Так (WebSocket)
Email-сповіщення через outbox	Так	Так	Так	Так
Темна/світла тема	Так	Так	Так	Так
OTP-реєстрація	Ні	Так	Так	Так

Жоден із проаналізованих сервісів не надає повноцінного поєднання розгорнутого текстового блогу з соціальними взаємодіями та кастомною системою сповіщень в одному застосунку.

Детальний аналіз Medium: плюси та мінуси. Medium був обраний для аналізу як найпопулярніша платформа для довгих текстових публікацій. Його сильні сторони: мінімалістичний дизайн, який не відволікає від читання; система рекомендацій, яка підбирає цікавий контент; можливість підписуватися на авторів та отримувати дайджести нових публікацій. Однак Medium має суттєві недоліки для нашого проєкту. По-перше, коментарі на Medium відсутні в більшості публікацій –

автори можуть їх увімкнути, але навіть тоді вони виглядають як окрема сторінка, а не інтегрований потік. По-друге, немає real-time сповіщень – автор дізнається про лайк або підписку тільки через лист на пошту годиною пізніше. По-третє, немає можливості зберігати пости в кастомні списки (є тільки «закладки»). Beauty Diary виправляє всі ці недоліки, додаючи глибоку соціальну взаємодію, не жертвуючи якістю читання.

Детальний аналіз Pinterest: візуальне натхнення без тексту. Pinterest – це платформа, яка геніально вирішує задачу збереження візуальних ідей. Користувачі створюють «дошки» та додають «піни» (зображення з посиланнями). Для Beauty Diary Pinterest цікавий насамперед системою збереження контенту – аналогічно до списку «збережених» постів, які користувач хоче переглянути пізніше. Однак Pinterest не підходить для блогової активності: текстові описи обмежені кількома рядками, коментарі майже не використовуються, підписки на авторів не є центральною функцією. Beauty Diary бере від Pinterest зручну систему збереження, але додає повноцінний текстовий контент та соціальну взаємодію.

Детальний аналіз Instagram: король соціальної взаємодії без довгих текстів. Instagram – найпопулярніша соціальна мережа для візуального контенту. Його ключові переваги: миттєві сповіщення, гіллясті коментарі, лайки, підписки, стрічка, алгоритми рекомендацій. Саме ці функції ми хочемо запозичити для Beauty Diary. Однак Instagram має серйозне обмеження для блогерів, які пишуть довгі тексти: підписи до фото обмежені 2200 символами (що приблизно 300-400 слів), немає категорій, тегів або форматування тексту (жирний, курсив, списки). Beauty Diary створює простір, де можна писати необмежено довгі пости з багатим форматуванням, але при цьому мати всі переваги соціальної мережі.

Порівняння архітектурних підходів аналогів. З технічної точки зору, Medium використовує традиційний SSR (серверний рендеринг) на основі власного фреймворку, що забезпечує швидке завантаження, але робить інтерактивність важчою. Instagram побудований на React Native (мобільний застосунок) та React (вебверсія), з активною інтеграцією WebSocket для real-time сповіщень. Pinterest використовує React та власну систему кешування. Beauty Diary обирає Next.js, який

дає найкраще з обох світів: Server Components для швидкого початкового завантаження та Client Components для інтерактивності, плюс вбудовані rewrites для проксі API та WebSocket.

1.3. Постановка задачі

На основі проведеного аналізу предметної області та існуючих аналогів сформульовано вимоги до вебзастосунку Beauty Diary.

Функціональні вимоги охоплюють: реєстрацію та автентифікацію користувачів із підтвердженням email через OTP-код; створення, редагування та видалення постів із заголовком, текстом (WYSIWYG-редактор Tiptap), обкладинкою, тегами та категорією; перегляд стрічки постів з фільтрацією та пагінацією; повнотекстовий пошук по постах; систему лайків, коментарів з гілками (підтримка parent_id) та soft delete; підписки на авторів; збереження постів у персональний список; real-time сповіщення через WebSocket (in-app) та email через outbox-чергу; адміністративну панель для управління користувачами та перегляду audit-логів.

Нефункціональні вимоги: адаптивний дизайн (мобільні та десктопні пристрої); підтримка темної та світлої теми; підтримка двох мов (украї́нська та англійська); безпека (JWT в httpOnly cookies, argon2id, CORS, Helmet, rate limiting); CI/CD (GitHub Actions); деплой на безкоштовних хмарних платформах (Vercel, Render, Neon, Cloudflare R2). Функціональні вимоги до вебзастосунку (розширено). Окрім базових функцій, Beauty Diary має задовольняти низку специфічних вимог. Система коментарів повинна підтримувати необмежену глибину гілок (відповіді на відповіді), мати soft-delete (щоб не ламати гілки при видаленні), а також мати пагінацію, оскільки під популярним постом може бути сотні коментарів. Система лайків має оновлювати лічильник миттєво (без перезавантаження сторінки) та не дозволяти ставити більше одного лайка від одного користувача. Система підписок повинна оновлювати стрічку користувача, показуючи пости від авторів, на яких він підписаний, у хронологічному порядку. Система сповіщень має підтримувати два канали: in-app (через WebSocket) та email (через outbox-чергу), а також дозволяти

користувачеві налаштовувати, які сповіщення він хоче отримувати (наприклад, вимкнути email-сповіщення про лайки, але залишити про коментарі). Система адміністрування має надавати можливість блокувати користувачів, змінювати їх ролі, переглядати audit-лог дій, а також моніторити outbox-чергу (переглядати, які email не надіслалися, і вручну запускати повторну спробу).

Нефункціональні вимоги (розширено). Beauty Diary повинен відповідати таким нефункціональним вимогам. Продуктивність: час завантаження головної сторінки (First Contentful Paint) не повинен перевищувати 1.5 секунди; час відповіді API на 95% запитів не повинен перевищувати 200 мілісекунд; WebSocket-з'єднання має встановлюватися не довше ніж за 500 мілісекунд. Масштабованість: система повинна витримувати до 1000 одночасних користувачів без деградації продуктивності. Доступність: система повинна бути доступна 99.5% часу (SLA). Безпека: паролі не повинні зберігатися у відкритому вигляді; токени повинні бути захищені від XSS-атак; реєстрація повинна бути захищена від ботів. Зручність використання: інтерфейс повинен бути інтуїтивно зрозумілим, адаптивним (коректно відображатися на телефонах, планшетах та десктопах), підтримувати темну та світлу тему, а також українську та англійську мови.

1.4. Обґрунтування вибору технологічного стеку

1.4.1. Фронтенд-фреймворк

Чому не Vue.js або Svelte? Vue.js є чудовим фреймворком з низьким порогом входження та зручним синтаксисом. Однак його екосистема менша, ніж у React, що особливо відчутно, коли мова йде про готові компоненти для редакторів (Tiptap має чудову інтеграцію з React) або про i18n рішення. Крім того, Vue.js не має нативної підтримки Server Components (Nuxt має часткову підтримку, але вона не така зріла, як у Next.js). Svelte (та SvelteKit) є дуже швидким і легким, але його екосистема відносно молода, знайти готові рішення для складних задач (як-от Tiptap або Socket.IO інтеграція) складніше. Крім того, на ринку праці значно більше React-розробників, що полегшує підтримку проєкту в майбутньому.

Чому Next.js, а не звичайний React SPA? Звичайний React SPA (збірка через Vite або Create React App) рендерить весь застосунок на клієнті. Це має кілька недоліків. По-перше, початкове завантаження повільне – користувач бачить порожню сторінку, поки завантажиться та виконається JavaScript (особливо погано на повільних мобільних мережах). По-друге, SEO страждає – пошукові роботи не завжди виконують JavaScript, тому пости можуть не індексуватися. По-третє, проблема cross-origin cookie вирішується складніше – потрібно налаштовувати CORS з декількома доменами. Next.js вирішує всі ці проблеми: Server Components рендерять контент на сервері, rewrites проксують API та WebSocket через один домен.

Детальніше про rewrites. Механізм rewrites у Next.js дозволяє «підміняти» маршрути. Наприклад, коли браувер надсилає запит на https://beauty-diary.vercel.app/api/auth/login, Next.js не шукає файл api/auth/login.js, а замість цього перенаправляє запит https://beauty-diary-api.onrender.com/api/auth/login. Для браузера це виглядає так, ніби фронтенд і бекенд знаходяться на одному домені, тому cookie працюють без проблем. Це геніальне рішення, яке не потребує складного налаштування CORS або сторонніх сервісів для проксування.

Порівняльний аналіз фронтенд-фреймворків наведено в таблиці 1.3.

Таблиця 1.3

Порівняння фронтенд-фреймворків

Критерій	**Next.js 14**	**Next.js 15**	**Nuxt 3**	**SvelteKit**
TypeScript-підтримка	Так	Так	Так	Так
Server Components	Так	Так	Часткова	Ні
App Router (файлова маршрутизація)	Так	Так	Так	Так
Вбудовані rewrites (проксі)	Так	Так	Ні	Ні

Критерій	Next.js 14	Next.js 15	Nuxt 3	SvelteKit
Розмір екосистеми	Великий	Великий	Середній	Малий
i18n-підтримка (next-intl)	Так	Так	Так	Ні
React-compatible	Так	Так	Ні	Ні

Вибір – Next.js 15 [2], оскільки він надає Server Components (зменшення JS-бандлу), вбудовані rewrites для проксі API, WebSocket та підтримку i18n через next-intl.

1.4.2. Бекенд-фреймворк

Таблиця 1.4

Порівняння бекенд-фреймворків

Критерій	NestJS 11	Express.js	Fastify	Hono
TypeScript-first	Так	Частково	Частково	Так
Вбудований DI-контейнер	Так	Ні	Ні	Ні
Модульність	Висока	Низька	Середня	Середня
Офіційна підтримка CQRS	Так	Ні	Ні	Ні
Офіційна підтримка WebSocket	Так	Ні	Ні	Ні
Розмір екосистеми	Великий	Дуже великий	Великий	Малий
Підходить для великих проєктів	Так	Складно	Так	Ні

Вибір – NestJS [3] 11 завдяки вбудованому Dependency Injection, модульності, офіційній підтримці CQRS та WebSocket (Socket.IO) і TypeScript-first підходу.Чому не Express.js? Express.js є найпопулярнішим фреймворком для Node.js, він простий, легкий і має величезну екосистему. Однак для великих проєктів він має суттєві недоліки. По-перше, він не нав'язує жодної структури – два різні розробники можуть організувати код повністю по-різному, що ускладнює підтримку. По-друге, він не має вбудованого DI-контейнера – всі залежності доводиться імпортувати явно, що

ускладнює тестування. По-третє, для реалізації CQRS або WebSocket доведеться писати багато допоміжного коду. NestJS вирішує всі ці проблеми, надаючи структурований, модульний підхід «з коробки».

Чому не Fastify? Fastify є чудовою альтернативою Express – він швидший, має вбудовану валідацію через JSON Schema та підтримує TypeScript. Однак його екосистема менша, ніж у NestJS, особливо коли мова йде про інтеграцію з ORM, CQRS або WebSocket. Крім того, Fastify не має такого потужного DI-контейнера, як NestJS. Для невеликих API Fastify є чудовим вибором, але для Beauty Diary, який має складну бізнес-логіку та багато модулів, NestJS підходить краще.

Детальніше про гексагональну архітектуру в NestJS. Гексагональна архітектура (її ще називають Ports & Adapters) базується на ідеї, що бізнес-логіка не повинна знати про зовнішній світ (базу даних, HTTP, файлове сховище). Замість цього вона визначає «порти» (інтерфейси), а «адаптери» реалізують ці інтерфейси, підключаючись до конкретних технологій. У NestJS це виглядає так: доменний шар (папка domain) містить сутності та інтерфейси репозиторіїв; application шар (папка application) містить CQRS-хендлери, які використовують інтерфейси; infrastructure шар (папка infrastructure) містить реалізації цих інтерфейсів (TypeORM-репозиторії, мапери). Такий підхід дозволяє легко замінити TypeORM на Prisma, або LocalStack на реальний S3, не чіпаючи бізнес-логіку.

1.4.3. База даних та ORM

Таблиця 1.5

Порівняння систем баз даних

Критерій	**PostgreSQL 16**	**MySQL 8**	**MongoDB 7**	**SQLite**
Реляційна модель	Так	Так	Ні	Так
Повнотекстовий пошук (tsvector)	Вбудований	Обмежений	Вбудований	Ні
Тригери та процедури	Так	Так	Обмежено	Частково

Критерій	**PostgreSQL 16**	**MySQL 8**	**MongoDB 7**	**SQLite**
JSONB-поля	Так	Частково (JSON)	Нативно	Ні
Горизонтальне масштабування	Так (Citus)	Так (шардинг)	Так	Ні
Підходить для serverless	Neon (serverless PG)	PlanetScale	MongoDB Atlas	Turso

Вибрано PostgreSQL [4] 16 через реляційну модель (складні зв'язки між постами, коментарями, лайками), підтримку повнотекстового пошуку через tsvector та можливість використання тригерів для денормалізації лічильників. ORM – TypeORM (вибрано замість Prisma через сумісність із NestJS та явні SQL-міграції). Чому PostgreSQL, а не MySQL? PostgreSQL і MySQL є двома найпопулярнішими реляційними базами даних. MySQL часто обирають для простих застосунків через його простоту та швидкість читання.

Однак PostgreSQL має кілька переваг, які є критичними для Beauty Diary. По-перше, розширення citext дозволяє створювати регістронезалежні поля (email та nickname), не втрачаючи в продуктивності. По-друге, tsvector та GIN-індекси забезпечують набагато швидший повнотекстовий пошук, ніж LIKE '%...%' у MySQL. По-третє, PostgreSQL має кращу підтримку JSONB-полів, що дозволяє зберігати структуровані дані (наприклад, payload сповіщень) без створення окремої таблиці. По-четверте, PostgreSQL підтримує матеріалізовані представлення та складніші тригери.

Чому TypeORM, а не Prisma? Prisma є сучасною ORM, яка набирає популярності завдяки зручному API та автоматичній генерації типів TypeScript. Вона також підтримує міграції. Однак на момент початку розробки Beauty Diary TypeORM мав кращу інтеграцію з NestJS (офіційний модуль @nestjs/typeorm), більш зрілу екосистему та ширшу підтримку складних SQL-функцій. Крім того, TypeORM дозволяє писати «сирі» SQL-запити там, де ORM не справляється, що було важливо

для реалізації повнотекстового пошуку через tsvector. Для написання міграцій TypeORM використовує SQL, що дає повний контроль над схемою бази даних.

Детальніше про міграції та append-only принцип. У TypeORM міграції генеруються командою pnpm db:generate після зміни ентіті. Генерується файл .ts, який містить методи up (що змінює схему) та down (що відкатує зміни). Ці файли зберігаються в папці migrations/ і комітяться в Git. Важливо дотримуватися принципу append-only: ніколи не редагувати вже закомічену міграцію. Якщо помилилися, створюється нова міграція, яка виправляє помилку. Це забезпечує відтворюваність – стан бази даних на будь-якому коміті можна відновити, запустивши всі міграції по порядку. У CI команда pnpm db:migrate запускається перед тестами; якщо міграція не проходить, CI стає червоним.

1.4.4. Підхід до валідації (Zod)

Таблиця 1.6

Порівняння бібліотек валідації

Критерій	**Zod**	**Joi**	**Yup**
TypeScript інференція	Так (нативна)	Ні (потрібен пакет)	Так (через Shape)
Синтаксис	Ланцюжковий	Ланцюжковий	Ланцюжковий
Трансформації (toLowerCase)	Так	Так	Обмежені
Використання на фронті (form resolver)	Так (zodResolver)	Ні	Так (yupResolver)
Розмір бандлу	Малий	Великий	Середній

Вибір – Zod [5], оскільки він дає єдине джерело правди для типів та валідації на обох кінцях застосунку через @beauty-diary/shared. Чому Zod, а не Joi чи Yup? Joi є чудовою бібліотекою валідації, але вона не має нативної TypeScript-інференції – типи доводиться писати окремо, що призводить до дублювання. Yup має інференцію,

але її синтаксис менш зручний для складних трансформацій (наприклад, email.toLowerCase()). Zod пропонує найкраще з обох світів: лаконічний ланцюжковий синтаксис, чудову TypeScript-підтримку, трансформації та невеликий розмір бандлу. Крім того, Zod має офіційний адаптер для react-hook-form (@hookform/resolvers/zod), що робить його використання на фронтенді максимально зручним.

Детальніше про shared-пакет та його вплив на якість коду. Рішення винести всі Zod-схеми в окремий пакет @beauty-diary/shared було ключовим для забезпечення якості коду. Без цього пакета фронтенд та бекенд мали б окремі копії правил валідації, які б розсинхронізувалися з часом. Наприклад, фронтенд міг би дозволяти нікнейм довжиною 3-20 символів, а бекенд – 3-50. Користувач вводить нікнейм з 25 символів, фронтенд пропускає, а бекенд відхиляє з незрозумілою помилкою. З shared-пакетом це неможливо: схема одна для всіх, TypeScript не дасть скомпілювати, поки не виправиш.

1.4.5. Архітектурний патерн (CQRS + гексагональна архітектура)

Гексагональна архітектура (ports & adapters) дозволяє ізолювати доменну логіку від зовнішніх деталей (HTTP, БД, файлове сховище). CQRS (Command Query Responsibility Segregation) розділяє операції запису (команди) та читання (запити) в окремі обробники.

Наприклад, створення коментаря: команда CreateCommentCommand виконує валідацію і зберігання в БД; подія CommentCreatedEvent публікується в шину; два обробники реагують паралельно: CommentCreatedNotifyHandler записує in-app сповіщення + email в outbox, а Realtime bridge надсилає comment:created в кімнату post:{id} через WebSocket. Детальний приклад роботи CQRS на прикладі коментаря. Розглянемо, як CQRS обробляє створення коментаря. Коли користувач натискає «Відправити», фронтенд надсилає POST /api/comments з postId, content та parentId. Контролер викликає commandBus.execute(new CreateCommentCommand(postId, userId, content, parentId)). NestJS знаходить обробник CreateCommentHandler, який

виконує логіку: перевіряє, чи існує пост, чи не заблокований автор, чи не перевищена максимальна глибина гілки (наприклад, не більше 10 рівнів). Потім зберігає коментар через commentRepository.save(). Після збереження він публікує подію CommentCreatedEvent через eventBus.publish(). Цю подію ловлять два обробники: CommentCreatedNotifyHandler (створює in-app сповіщення та email в outbox) та CommentCreatedRealtimeHandler (надсилає WebSocket-подію comment:created в кімнату post:{id}). Таким чином, основна операція (створення коментаря) не чекає на сповіщення, вони виконуються паралельно асинхронно.

Чому CQRS, а не просто сервіси з методами? У традиційному підході (сервіси з методами) створення коментаря виглядало б так: commentService.create(). Всередині цього методу був би код для валідації, збереження, створення сповіщення, надсилання email, WebSocket... Це призводить до «товстих» методів, які важко тестувати та змінювати. CQRS розбиває цю логіку на окремі, незалежні класи (команди та обробники). Кожен обробник має одну відповідальність, його легко тестувати ізольовано. Крім того, CQRS дозволяє окремо оптимізувати запити (кешування) та команди (запис у БД).

1.4.6. Інфраструктура та CI/CD

Локальна розробка виконується за допомогою Docker Compose з контейнерами: PostgreSQL (порт 5439), LocalStack (S3-сумісне сховище, порт 4569) та Mailpit (SMTP перехоплення, порт 8025).

Деплой: Vercel – для Next.js; Render – для NestJS; Neon – PostgreSQL (serverless); Cloudflare R2 [6] – зображення (S3-сумісне).

CI (GitHub Actions): lint – перевірка ESLint/Prettier; typecheck – TypeScript перевірка у всіх пакетах; test – unit + інтеграційні тести з реальними контейнерами; build – фінальна збірка web, api, shared.Детальніше про Docker Compose. Docker Compose використовує файл docker-compose.yml, в якому описані три сервіси. Сервіс postgres використовує офіційний образ PostgreSQL 16, монтує том для збереження даних, встановлює змінні середовища (POSTGRES_USER, POSTGRES_PASSWORD,

POSTGRES_DB). Сервіс localstack використовує образ LocalStack 3, емулює S3 API, виконує init-скрипт для створення бакету beauty-diary-media. Сервіс mailpit використовує образ Mailpit, відкриває порти 1025 (SMTP) та 8025 (Web UI). Усі три сервіси підключені до спільної мережі Docker, що дозволяє їм взаємодіяти один з одним. Команда pnpm infra:up викликає docker-compose up -d, який запускає контейнери у фоновому режимі. pnpm infra:down зупиняє їх, а pnpm infra:reset повністю видаляє всі дані та перезапускає.

Детальніше про хмарні платформи (чому безкоштовно). Vercel пропонує безкоштовний тариф Hobby, який включає 100 GB трафіку на місяць та необмежену кількість проєктів. Render пропонує безкоштовний тариф, який дає 750 годин роботи на місяць (що достатньо для одного сервера 24/7). Neon пропонує безкоштовний тариф з 0.5 GB пам'яті та 10 000 рядків – достатньо для невеликого блогу. Cloudflare R2 дає 10 GB безкоштовного зберігання та безкоштовний вихід трафіку (на відміну від AWS S3, де вихід трафіку платний). Brevo дає 300 листів на день безкоштовно. Таким чином, Beauty Diary можна хостити повністю безкоштовно, поки кількість користувачів не перевищить кілька тисяч.

Детальніше про CI/CD пайплайн. GitHub Actions налаштований через файл .github/workflows/ci.yml. Він запускається на кожен push у гілку main та на кожен Pull Request. Job lint виконує pnpm format:check (перевіряє, чи всі файли відформатовані Prettier) та pnpm lint (перевіряє код ESLint). Job typecheck виконує pnpm typecheck, який запускає tsc --noEmit у всіх трьох пакетах. Job test спочатку піднімає service-контейнери PostgreSQL та LocalStack, потім виконує pnpm db:migrate, а потім pnpm test. Job build виконує pnpm build, який збирає shared, потім api, потім web. Якщо будь-який job падає, CI стає червоним, і PR не можна змерджити.

РОЗДІЛ 2. ПРИКЛАДНЕ ЗАСТОСУВАННЯ FRONTEND-ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ІНТЕРАКТИВНОГО БЛОГУ З ЕЛЕМЕНТАМИ СОЦІАЛЬНОЇ МЕРЕЖІ

2.1. Аналіз бізнес-процесів Beauty Diary

Використання інформаційних технологій у сфері блогінгу та соціальних мереж є важливим кроком до автоматизації та покращення взаємодії між авторами та читачами. У нашому проєкті Beauty Diary ми розглядаємо процеси, які відбуваються під час реєстрації користувача, створення та публікації постів, взаємодії з контентом та отримання сповіщень.

2.1.1. Аналіз процесів реєстрації та автентифікації

Системний аналіз процесів реєстрації та автентифікації є фундаментальним етапом у життєвому циклі розробки програмного забезпечення для соціальних платформ. Детальна декомпозиція процесу реєстрації дозволяє виділити такі критично важливі стадії:

- Заповнення реєстраційної форми – користувач вводить email, нікнейм та пароль. Система виконує миттєву валідацію: перевіряє формат email, унікальність нікнейму, складність пароля.
- Генерація та надсилання OTP-коду – після успішної валідації система генерує 6-значний одноразовий код та надсилає його на вказану email-адресу через outbox-чергу.
- Введення OTP-коду – система перевіряє термін дії коду (10 хвилин) та кількість спроб (не більше 5).
- Створення облікового запису – при успішному підтвердженні система створює запис у таблиці users та генерує JWT-токени.
- Автоматичний вхід – після реєстрації користувач одразу потрапляє в застосунок.

2.1.2. Аналіз процесів створення та публікації постів

Процес публікації посту складається з таких етапів:

– Редагування тексту – автор використовує візуальний редактор Tiptap (WYSIWYG).
– Завантаження обкладинки – файл відправляється в хмарне сховище Cloudflare R2 через presigned URL. Дозволені формати: JPEG, PNG, WebP. Максимальний розмір – 5 МБ.
– Вибір категорії та тегів – автор обирає одну категорію з шести та до 5 тегів.
– Збереження чернетки або публікація.
– Індексація для пошуку – після публікації система оновлює tsvector індекс.

2.1.3. Аналіз процесів соціальної взаємодії

Beauty Diary надає користувачам можливість взаємодіяти з контентом через такі механізми:

– Лайки – система перевіряє, чи не було вже поставлено лайк, збільшує лічильник через тригер. Автор отримує сповіщення.
– Коментарі – підтримуються гілки відповідей (поле parent_id). Реалізовано soft delete: при видаленні коментар відображається як «[видалено]».
– Підписки – система створює запис у таблиці user_follows та оновлює лічильники підписників/підписок.
– Збереження постів – особистий список «збережених», невидимий для інших користувачів.

2.1.4. Аналіз процесів сповіщень

Система сповіщень працює за такою схемою: генерація події → перевірка налаштувань отримувача → in-app сповіщення через WebSocket → email-сповіщення через outbox-чергу → retry-механізм при невдачі.Детальна схема реєстрації з OTP. Процес реєстрації в Beauty Diary складається з двох етапів. Етап 1: ініціація. Користувач заповнює форму: email, нікнейм, пароль. Фронтенд валідує формат email (має містити @ та крапку), перевіряє, чи

нікнейм не містить спецсимволів (дозволені лише літери, цифри, підкреслення та дефіс), чи пароль не коротший за 8 символів. Потім надсилає POST /api/auth/register/initiate. Бекенд перевіряє, чи email вже не зареєстрований, чи нікнейм вільний. Якщо все добре, генерує 6-значний випадковий код, хешує його через argon2, зберігає в таблицю pending_registrations (разом з email, хешем пароля, часом створення) та надсилає код на email через outbox-чергу. Етап 2: верифікація. Користувач отримує лист з кодом, вводить його у форму. Фронтенд надсилає POST /api/auth/register/verify з email та кодом. Бекенд шукає запис у pending_registrations за email, перевіряє, чи не закінчився термін дії (10 хвилин), чи не перевищено кількість спроб (максимум 5). Якщо код правильний – створює запис у users, генерує JWT-токени, встановлює httpOnly cookie та видаляє запис із pending_registrations. Користувач автоматично входить у застосунок.

Детальна схема створення поста. Авторизований користувач переходить на сторінку /uk/create-post. Він бачить редактор Tiptap, який підтримує: жирний, курсив, заголовки (H2, H3), списки (нумеровані та марковані), цитати, посилання, вставку зображень. Окремий блок для завантаження обкладинки: користувач може обрати файл (JPEG, PNG, WebP, до 5 МБ). Система спочатку завантажує зображення в Cloudflare R2 через presigned URL (запит до /api/media/upload-url повертає URL, на який можна напряму завантажити файл).

Після завантаження зберігається URL обкладинки. Потім користувач обирає категорію (випадаючий список) та додає до 5 тегів (теги можна писати вручну, система пропонує існуючі). Коли користувач натискає «Опублікувати», фронтенд надсилає POST /api/posts з title, content (HTML), coverImageUrl, categoryId, tags.

Бекенд валідує, чи заголовок не пустий, чи категорія існує, чи теги не дублюються. Потім зберігає пост у таблицю posts зі статусом published. Якщо користувач натискає «Зберегти чернетку», статус буде draft, і пост не з'явиться у стрічці інших користувачів.

Детальна схема коментарів з гілками. Коментарі в Beauty Diary підтримують необмежену вкладеність (відповідь на відповідь на відповідь). Це реалізовано через поле parent_id в таблиці comments. Якщо parent_id дорівнює NULL, це кореневий коментар. Якщо вказує на інший коментар – це відповідь. При завантаженні сторінки поста фронтенд запитує GET /api/posts/:id/comments. Бекенд використовує рекурсивний SQL-запит (WITH RECURSIVE) для побудови дерева коментарів. Коментарі сортуються за датою створення (найновіші зверху для кореневих). Відповіді відображаються з відступом. При видаленні коментаря (через DELETE /api/comments/:id) система не видаляє його фізично, а встановлює deleted_at. У відповіді API текст коментаря замінюється на «[видалено]», але гілка відповідей залишається видимою.

2.2. Організація та структура інформаційних масивів

Для забезпечення повноцінного функціонування системи архітектура бази даних передбачає акумуляцію таких інформаційних блоків: дані про користувача (ID, email, нікнейм, аватар, роль, статус, лічильники активності, налаштування сповіщень); дані про пости (ID, заголовок, зміст, статус, медіа, категорія, лічильники, пошуковий вектор); дані про коментарі (ID, ID поста, ID автора, ID батьківського коментаря, текст, метадані soft delete).Детальна структура таблиці users.

Таблиця users містить наступні поля. id – UUID (первинний ключ). email – citext (унікальний, регістронезалежний). nickname – citext (унікальний, регістронезалежний). displayName – текст (може бути NULL, використовується для відображення імені замість нікнейму). avatarUrl – текст (посилання на файл у Cloudflare R2, може бути NULL). bio – текст (до 500 символів). role – enum ('user', 'admin'). isBlocked – булеве значення. followersCount – ціле число (денормалізований

лічильник, оновлюється тригером). followingCount – ціле число (денормалізований лічильник). postsCount – ціле число (денормалізований лічильник). createdAt – timestamp. updatedAt – timestamp.

Детальна структура таблиці posts. Таблиця posts містить: id – UUID. title – текст (обов'язкове). content – текст (HTML від Tiptap, може бути NULL для чернеток без тексту). coverImageUrl – текст (посилання на файл у R2). status – enum ('draft', 'published'). authorId – UUID (зовнішній ключ до users). categoryId – UUID (зовнішній ключ до categories). likesCount – ціле число (денормалізований лічильник, оновлюється тригером). commentsCount – ціле число (денормалізований лічильник, оновлюється тригером). viewsCount – ціле число (лічильник переглядів). searchVector – tsvector (автоматично оновлюється при зміні title та content, використовується для повнотекстового пошуку). createdAt – timestamp. updatedAt – timestamp.

Детальна структура таблиці comments. Таблиця comments містить: id – UUID. content – текст (обов'язкове, не більше 5000 символів). postId – UUID (зовнішній ключ до posts). authorId – UUID (зовнішній ключ до users). parentId – UUID (зовнішній ключ до comments, може бути NULL). deletedAt – timestamp (NULL, якщо коментар не видалений). createdAt – timestamp. updatedAt – timestamp.

Тригери для денормалізації. Денормалізація – це техніка, коли одні й ті самі дані зберігаються в кількох місцях для прискорення читання. У Beauty Diary лічильники likesCount, commentsCount, followersCount зберігаються безпосередньо в таблицях posts та users, хоча їх можна було б обчислювати через COUNT з таблиць post_likes або comments. Це зроблено для продуктивності – щоб отримати стрічку постів, не потрібно робити додаткові JOIN-и. Щоб лічильники залишалися актуальними, створено тригери. Наприклад, тригер update_post_likes_count спрацьовує після вставки або видалення рядка в таблиці post_likes і оновлює likesCount у відповідному пості. Це гарантує, що лічильник завжди правильний, навіть якщо хтось змінює БД безпосередньо.

2.3. Розробка архітектури та вибір технологій

Для реалізації Beauty Diary обрано багаторівневу архітектуру (N-tier architecture) з використанням монорепозиторію (pnpm workspaces): рівень презентації – Next.js 15; рівень бізнес-логіки – NestJS 11 з гексагональною архітектурою та CQRS; рівень доступу до даних – TypeORM + PostgreSQL 16.

Рис. 2.1 – UML діаграма архітектурних рівнів Beauty Diary (розроблено автором)

Таблиця 2.1

Технологічний стек Beauty Diary

Компонент	**Технологія**	**Призначення**
Фронтенд	Next.js 15	React-фреймворк з App Router, Server Components
Бекенд	NestJS 11	Модульний Node.js-фреймворк з DI
Мова програмування	TypeScript	Типобезпека на фронті та бекенді
База даних	PostgreSQL 16	Реляційна БД з повнотекстовим пошуком
ORM	TypeORM	Об'єктно-реляційне відображення
Валідація	Zod + shared пакет	Єдиний контракт між фронтом і беком
Real-time	Socket.IO	WebSocket для миттєвих сповіщень
Стилі	Tailwind v4 + shadcn/ui	Утилітарний CSS та компоненти
Хешування паролів	argon2id	Найсучасніший алгоритм хешування
Автентифікація	JWT (httpOnly cookies)	Безсесійна автентифікація
Email	Brevo (outbox-черга)	Надсилання email-сповіщень
Сховище зображень	Cloudflare R2	S3-сумісне об'єктне сховище
Деплой (фронт)	Vercel	Хостинг Next.js
Деплой (бек)	Render	Хостинг NestJS
База даних (хмара)	Neon	Serverless PostgreSQL

Вибір Next.js 15 обґрунтовується наявністю Server Components, вбудованих rewrites для проксі API та WebSocket та підтримки i18n. Вибір NestJS 11 обґрунтовується вбудованим DI, модульністю, офіційною підтримкою CQRS та WebSocket. Вибір PostgreSQL 16 обґрунтовується реляційною моделлю, підтримкою

повнотекстового пошуку та тригерами. Чому Tailwind CSS v4, а не звичайний CSS або Sass? Tailwind CSS v4 є утилітарним CSS-фреймворком, який надає сотні готових класів (наприклад, bg-red-500, text-center, p-4).

Це дозволяє верстати інтерфейс без написання жодного рядка кастомного CSS. Основні переваги: швидкість розробки (не треба придумувати назви класів), консистентність (кольори, відступи, шрифти визначені в темі), невеликий фінальний CSS (невикористані класи видаляються автоматично). У Tailwind v4 з'явилася CSS-first конфігурація через дизайн-токени в globals.css, що робить налаштування теми ще простішим. Наприклад, щоб змінити основний колір, достатньо змінити --primary в CSS, а не правити конфігураційний файл.

Чому shadcn/ui [9], а не Material-UI або Ant Design? Material-UI (MUI) та Ant Design є повноцінними бібліотеками компонентів, які надають готові рішення для кнопок, інпутів, діалогів, таблиць тощо. Однак вони мають недоліки: великий розмір бандлу, складність кастомізації (доводиться боротися зі стилями бібліотеки), та залежність від оновлень бібліотеки. shadcn/ui працює інакше: це не бібліотека, а набір копійованих компонентів. Команда npx shadcn-ui add button копіює вихідний код кнопки прямо в папку components/ui/. Цей код належить тобі, ти можеш змінювати його як завгодно. Немає залежності від версій бібліотеки, немає необхідності боротися зі стилями. shadcn/ui також використовує Tailwind CSS, тому компоненти виглядають так само, як і твій власний код.

Чому TanStack Query, а не useState + useEffect? Без TanStack Query для отримання стрічки постів потрібно було б: useEffect для виконання fetch, useState для зберігання даних, useState для статусу завантаження, useState для помилки. При переході між сторінками пагінації дані довелося б завантажувати заново. При редагуванні поста – інвалідувати кеш вручну. TanStack Query автоматизує все це: він кешує дані, повторно використовує їх при повторних запитах, автоматично інвалідує кеш після мутацій, надає зручні статуси (isLoading, isError), підтримує пагінацію та нескінченний скрол. Це зменшує кількість коду в рази та зменшує кількість помилок.

Чому react-hook-form, а не форми вручну? Без react-hook-form для кожної форми потрібно було б: useState для кожного поля, обробник onChange, обробник onSubmit, валідацію вручну, відображення помилок вручну. react-hook-form автоматизує це: ви реєструєте поля через register, він сам відстежує зміни, валідує через zodResolver, надає помилки через formState.errors. Він також оптимізує продуктивність – ререндериться тільки змінене поле, а не вся форма.

2.4. Забезпечення безпеки даних

У системі Beauty Diary реалізовано багаторівневу модель безпеки. Автентифікація та JWT: access-токен живе 15 хвилин, refresh – 30 днів; токени зберігаються в httpOnly cookies; реалізовано refresh token rotation. OTP-верифікація: код діє 10 хвилин, максимальна кількість спроб – 5, повторне надсилання – не частіше ніж раз на 60 секунд. Хешування паролів: алгоритм argon2id (memoryCost=19456, timeCost=2). Рольова модель: user та admin, захист маршрутів через AuthGuard та RolesGuard. Захист API: CORS, Helmet (CSP, X-Frame-Options, HSTS), rate limiting.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ «BEAUTY DIARY»

3.1. Загальна структура та середовище розробки

3.1.1. Організація монорепозиторію

Проєкт Beauty Diary реалізовано як монорепозиторій з використанням pnpm workspaces. Структура монорепозиторію: apps/web – клієнтська частина на Next.js 15; apps/api – серверна частина на NestJS 11; packages/shared – спільний пакет із Zod-схемами та TypeScript-типами; infra – конфігурація Docker Compose та деплой-маніфести.

3.1.2. Середовище розробки

Як основне інтегроване середовище розробки обрано Visual Studio Code – безкоштовний, легковажний та потужний редактор коду від Microsoft. Вибір обґрунтовується інтелектуальною підтримкою TypeScript, вбудованим терміналом, інтеграцією з Git та розширюваністю.

3.2. Реалізація бекенду (NestJS)

Бекенд Beauty Diary побудований на NestJS 11 з використанням гексагональної архітектури (Ports & Adapters). Кожен модуль має чотири шари: domain (бізнес-сутності), application (CQRS-хендлери), infrastructure (репозиторії TypeORM, WebSocket-шлюз) та presentation (контролери REST API).

WebSocket-шлюз реалізований через Socket.IO [7]. Система підтримує такі події: notification:created, comment:created, post:liked.

3.2.6. Безпека бекенду

Безпека бекенду реалізована на кількох рівнях: автентифікація та JWT (argon2id, httpOnly cookies, refresh token rotation), OTP-верифікація (10 хвилин, 5 спроб, cooldown 60 с), рольова модель (AuthGuard, RolesGuard), захист API (CORS, Helmet, rate limiting).

3.3. Реалізація фронтенду (Next.js)

3.3.1. Загальна архітектура фронтенду

Клієнтська частина Beauty Diary реалізована на Next.js 15 з використанням App Router. Server Components зменшують кількість JavaScript на клієнті та покращують SEO, Client Components забезпечують інтерактивність. Підтримується i18n (next-intl) та темна/світла тема (next-themes).

3.3.2. Структура сторінок

Таблиця 3.1

Маршрути та сторінки вебзастосунку Beauty Diary

Маршрут	**Сторінка**	**Призначення**
/uk/ або /en/	Головна	Стрічка постів, категорії, популярне
/uk/feed	Стрічка	Персоналізована стрічка постів
/uk/posts/:id	Детальний перегляд поста	Пост, коментарі, кнопки взаємодії
/uk/create-post	Створення поста	Редактор з Tiptap, завантаження обкладинки
/uk/profile/:nickname	Профіль користувача	Пости користувача, інформація, підписки
/uk/saved	Збережені пости	Список постів, доданих у збережені
/uk/login	Вхід	Форма авторизації
/uk/register	Реєстрація	Форма реєстрації з OTP
/uk/admin	Адмін-панель	Управління користувачами, перегляд логів

3.3.3. Стилізація та компоненти

Для стилізації використано Tailwind CSS v4 та shadcn/ui. Tailwind v4 використовує CSS-first конфігурацію через дизайн-токени в globals.css. shadcn/ui – набір копійованих примітивів (кнопки, інпути, діалоги), вихідний код яких зберігається в папці components/ui/.

3.3.4. Управління станом та дані

Для керування серверними даними використано TanStack Query (кешування, автоматична інвалідація, дедуплікація, пагінація). Для форм – react-hook-form з zodResolver.

3.3.5. Real-time на фронтенді

Для отримання миттєвих сповіщень фронтенд підключається до WebSocket-шлюзу через socket.io-client. При отриманні події notification:created фронтенд оновлює кеш TanStack Query і дзвіночок відображає нове сповіщення без перезавантаження сторінки.

3.4. Спільний пакет (shared)

Спільний пакет @beauty-diary/shared містить: Zod-схеми для валідації всіх DTO; TypeScript-типи, виведені з цих схем; константи; enum-типи (UserRole, NotificationType, PostStatus).

На бекенді Zod-схеми використовуються в кастомному ZodValidationPipe. На фронтенді – в react-hook-form через zodResolver. Це гарантує однакові правила валідації на обох кінцях без дублювання коду.

3.5. Інфраструктура та CI/CD

3.5.1. Локальна розробка з Docker Compose

Для локальної розробки використовується Docker Compose, який піднімає три контейнери: PostgreSQL (порт 5439), LocalStack – емуляція S3-сумісного сховища (порт 4569), Mailpit – локальний email-сервер (порт 8025).

3.5.2. Деплой на хмарні платформи

Фронтенд (Next.js) розміщено на Vercel; бекенд (NestJS) – на Render; база даних – Neon (serverless PostgreSQL); зображення – Cloudflare R2 (S3-сумісне); email – Brevo (до 300 листів на день безкоштовно).

3.5.3. Проксі API та WebSocket через Next.js rewrites

Оскільки фронтенд (Vercel) та бекенд (Render) розташовані на різних доменах, використано механізм rewrites у Next.js: всі запити на /api/* та /socket.io/* проксуються з Vercel на Render. Для браузера цей процес непомітний – cookie встановлюються для домену фронтенду.

3.5.4. CI/CD через GitHub Actions

При кожному push у гілку main запускаються чотири паралельні jobs: lint, typecheck, test (з реальними контейнерами), build.

3.6. Керівництво користувача та адміністратора

Для початку роботи необхідно зареєструватися на сторінці /uk/register, вказати email, нікнейм та пароль, підтвердити email через 6-значний код. Авторизований користувач може створювати пости через /uk/create-post (Tiptap-редактор, завантаження обкладинки, вибір категорії та тегів). Читачі можуть ставити лайки, писати коментарі, підписуватися на авторів та зберігати пости. Адміністратори мають доступ до панелі /uk/admin для управління користувачами та перегляду audit-логу.

РОЗДІЛ 4. РОЗГОРТАННЯ, ТЕСТУВАННЯ ТА СУПРОВІД ВЕБЗАСТОСУНКУ

4.1. Локальне розгортання та налаштування середовища розробки

Для забезпечення комфортної та ізольованої розробки Beauty Diary створено локальне оточення на основі Docker. Необхідне ПЗ: Node.js ≥ 22, pnpm ≥ 9, Docker Desktop ≥ 4.0, Git ≥ 2.40, Visual Studio Code.

Процес розгортання: 1) git clone https://github.com/kkarntu/beauty-diary.git; 2) pnpm install; 3) pnpm infra:up (запуск PostgreSQL, LocalStack, Mailpit); 4) налаштування .env файлів; 5) pnpm db:migrate; 6) pnpm dev (фронтенд доступний на http://localhost:23000, бекенд – http://localhost:23001/api).Детальна покрокова інструкція з локального запуску (для новачків). Якщо хтось клонує репозиторій вперше, ось що потрібно зробити.

Крок 0: Переконатися, що встановлено Node.js 22+, pnpm 9+, Docker Desktop. Крок 1: git clone https://github.com/kkarntu/beauty-diary.git && cd beauty-diary. Крок 2: pnpm install (займе 1-2 хвилини). Крок 3: pnpm infra:up (запускає Docker-контейнери). Переконатися, що Docker запущено. Крок 4: Скопіювати .env.example в .env для api та web, змінити значення за потреби. Крок 5: pnpm db:migrate (створить таблиці). Крок 6: pnpm dev. Через 10-20 секунд застосунок доступний на http://localhost:23000. Для перегляду листів відкрити http://localhost:8025 (Mailpit). Для перевірки S3 – http://localhost:4569 (LocalStack). Для зупинки всіх контейнерів: pnpm infra:down.

Можливі проблеми та їх вирішення. Проблема: pnpm infra:up падає з помилкою "port already in use". Вирішення: Порти 5439, 4569, 8025 вже зайняті іншою програмою. Змінити порти в docker-compose.yml або зупинити програму, яка їх використовує. Проблема: pnpm db:migrate падає з помилкою "role 'beauty' does not exist". Вирішення: Переконатися, що контейнер PostgreSQL запущено (docker ps). Якщо ні – перезапустити pnpm infra:up. Проблема: pnpm dev запускається, але

фронтенд не бачить бекенд. Вирішення: Перевірити змінні середовища в .env.local та .env. Вони мають вказувати на localhost:23001. Проблема: Помилка "Cannot find module '...'". Вирішення: pnpm install ще раз.

4.2. Тестування застосунку

Впроваджено багаторівневу стратегію тестування: юніт-тести (Jest для бекенду, Vitest для фронтенду) та інтеграційні тести (Supertest + Testcontainers – реальні контейнери PostgreSQL та LocalStack). Тести покривають реєстрацію, створення постів, коментування, лайки, підписки та сповіщення

Детальні приклади тестів. Приклад юніт-тесту бекенду: тест для CreateCommentHandler. Створюються мок-об'єкти: mockPostRepository.findByPostId повертає пост, mockUserRepository.findById повертає автора, mockCommentRepository.save повертає створений коментар. Виконується команда.

Перевіряється, що save було викликано з правильними параметрами, та що eventBus.publish було викликано з CommentCreatedEvent. Приклад інтеграційного тесту бекенду: тест для реєстрації. Запускається повний NestJS-застосунок з реальною БД (Testcontainers). Виконується request(app.getHttpServer()).post('/api/auth/register/initiate').send({ email: 'test@test.com', nickname: 'test', password: 'password123' }). Перевіряється, що відповідь має статус 204.

Перевіряється в БД, що таблиця pending_registrations містить запис з цим email. Потім виконується request(...).post('/api/auth/register/verify').send({ email: 'test@test.com', code: '123456' }) (код потрібно взяти з БД, оскільки він хешований, але в тесті ми можемо зберегти оригінальний код в окремій змінній). Перевіряється, що відповідь має статус 201 та встановлює cookie. Перевіряється, що в таблиці users з'явився запис.

Покриття тестами. Наразі тести покривають приблизно 80% бізнес-логіки. Не покриті: граничні випадки (наприклад, коментар з 5001 символом), деякі сценарії з

WebSocket (складно тестувати інтеграційно), та деякі edge cases в outbox-обробнику. Планується додати ці тести в майбутньому.

4.3. Розгортання у хмарі (CI/CD)

GitHub Actions запускає при кожному push: lint (ESLint/Prettier), typecheck (TypeScript), test (реальні контейнери), build. Після злиття в main Vercel та Render автоматично деплоять нові версії.

Вирішення проблеми cross-origin cookie: у next.config.ts налаштовано rewrites, які проксують /api/* та /socket.io/* з Vercel на Render, що дозволяє cookie встановлюватись для домену фронтенду. Детальна схема деплою на Vercel. Після підключення репозиторію до Vercel (через "Import Git Repository") Vercel автоматично виявляє, що це Next.js проєкт.

Він читає package.json, виконує pnpm install --prod=false (тому що для збірки потрібні devDependencies, такі як TypeScript), потім pnpm build (який виконує next build). Після успішної збірки Vercel деплоїть статичні файли та серверні функції (Serverless Functions) у своїй глобальній мережі CDN. Кожен push у гілку main автоматично тригерсить новий деплой. Vercel також надає preview-деплої для кожного Pull Request – це дозволяє тестувати зміни перед злиттям.

Детальна схема деплою на Render. Render налаштовується через infra/render.yaml. Цей файл описує Web Service: buildCommand: "pnpm install --prod=false && pnpm build" (збірка), startCommand: "pnpm db:migrate && node dist/src/main.js" (запуск). Після підключення репозиторію до Render, він автоматично створює сервіс. При кожному push у main Render виконує buildCommand, потім startCommand. Оскільки Render не має вбудованої PostgreSQL, ми використовуємо Neon. Render також надає cron-завдання (для outbox-процесора) та healthchecks (endpoint /health).

4.4. Супровід та моніторинг

Реалізовано: глобальний фільтр винятків DomainExceptionFilter; audit-лог адміністративних дій у таблиці audit_log; моніторинг outbox-черги на сторінці /uk/admin/email-outbox; liveness-проба GET /health для Render.Детальніше про audit-лог. Таблиця audit_log створюється міграцією. Вона має поля: id, actorId (ID адміністратора, який виконав дію), action (тип дії, наприклад 'BLOCK_USER', 'UNBLOCK_USER', 'CHANGE_ROLE', 'DELETE_POST', 'DELETE_COMMENT'), targetId (ID цільового об'єкта, наприклад ID користувача, якого заблокували), details (JSON-об'єкт з додатковою інформа

4.5. Висновки до розділу 4

У четвертому розділі розглянуто практичні аспекти розгортання, тестування та супроводу Beauty Diary: описано локальне середовище розробки з Docker Compose, стратегію тестування, налаштування CI/CD та механізми моніторингу. Таким чином, Beauty Diary є повністю готовим до експлуатації продуктом із налагодженим процесом розробки та розгортання.

РОЗДІЛ 5. АНАЛІЗ ВЕБДОСТУПНОСТІ

5.1. Загальні положення вебдоступності

У сучасному цифровому суспільстві доступність вебресурсів є не лише технічною вимогою, а й соціальною відповідальністю розробника. Вебдоступність (web accessibility) означає, що вебсайти, інструменти та технології розробляються таким чином, щоб ними могли користуватися люди з різними типами обмежень – зоровими, слуховими, руховими, когнітивними та іншими.

За даними Всесвітньої організації охорони здоров'я, понад 1 мільярд людей у світі (приблизно 15% населення) живуть з тією чи іншою формою інвалідності. В Україні цей показник становить близько 2,7 мільйона осіб. Для цих людей доступний вебсайт – це не просто зручність, а можливість повноцінної участі в суспільному житті, отримання інформації, спілкування та самореалізації.

Основні категорії користувачів, які потребують доступності:

Люди з порушеннями зору – часткова або повна втрата зору, дальтонізм, знижена гострота зору. Для них важливі: контрастність кольорів, можливість збільшення шрифту, сумісність з екранними диктофонами (скрінрідерами).

Люди з порушеннями слуху – глухі, слабочуючі. Для них важливі: текстові альтернативи аудіоконтенту, субтитри до відео.

Люди з руховими порушеннями – обмежена рухливість рук, тремор, параліч. Для них важливі: можливість роботи з сайтом виключно через клавіатуру, великі клікабельні зони.

Люди з когнітивними порушеннями – дислексія, СДУГ, розлади аутистичного спектру. Для них важливі: проста та зрозуміла структура, відсутність мерехтливих елементів, передбачувана поведінка інтерфейсу.

Основним міжнародним стандартом у сфері вебдоступності є Web Content Accessibility Guidelines (WCAG) 2.1, розроблені Консорціумом Всесвітньої павутини (W3C). Цей стандарт визначає три рівні відповідності:

Рівень А – мінімальний рівень доступності, який мають забезпечувати всі вебсайти. Це базові вимоги, без яких сайт взагалі не може використовуватися людьми з інвалідністю.

Рівень АА – середній рівень, рекомендований для більшості сайтів. Це оптимальний баланс між доступністю та складністю реалізації. Для соціальних мереж та блогів, таких як Beauty Diary, рекомендованим є саме цей рівень.

Рівень ААА – максимальний рівень, який є обов'язковим для державних установ та сервісів, що надають критичні послуги.

Українське законодавство також регулює питання вебдоступності. Закон України "Про доступність до об'єктів та послуг" (2019 року) вимагає, щоб вебсайти державних органів та установ були доступними для всіх груп населення. Крім того, постанови Кабінету Міністрів України встановлюють вимоги до вебресурсів у частині їх доступності для осіб з інвалідністю.

Аналіз вебдоступності Beauty Diary проводився за чотирма основними принципами WCAG:

Сприйнятність (Perceivable) – інформація та компоненти інтерфейсу мають бути представлені так, щоб користувачі могли їх сприймати незалежно від своїх можливостей. Це означає, що текст повинен мати достатній контраст, зображення повинні мати текстові альтернативи, а відео – субтитри.

Керованість (Operable) – компоненти навігації та інтерфейсу мають бути керованими з різних пристроїв введення. Користувач повинен мати можливість взаємодіяти з сайтом не лише за допомогою миші, але й через клавіатуру, голосові команди або інші допоміжні технології.

Зрозумілість (Understandable) – інформація та робота інтерфейсу повинні бути зрозумілими. Текст має бути написаний простою мовою, навігація повинна бути передбачуваною, а помилки введення – коректно пояснюватися.

Надійність (Robust) – контент має бути доступним для різних технологій, включаючи допоміжні пристрої. Це означає, що код повинен бути написаний згідно зі стандартами W3C і коректно інтерпретуватися різними браузерами та скрінрідерами.

Дослідження показують, що вебсайти, розроблені з урахуванням принципів доступності, не лише відповідають законодавчим вимогам, а й покращують користувацький досвід для всіх відвідувачів, підвищують SEO-показники та зменшують ризик судових позовів. Крім того, доступні сайти мають кращі показники конверсії, оскільки вони зручніші та зрозуміліші для широкої аудиторії.

5.2. Аналіз доступності інтерфейсу Beauty Diary

5.2.1. Клавіатурна навігація

Критично важливою вимогою вебдоступності є можливість використання сайту виключно за допомогою клавіатури. Люди з порушеннями зору або руховими обмеженнями не можуть користуватися мишею, тому всі інтерактивні елементи мають бути доступні через клавішу Tab. Клавіатурна навігація дозволяє переміщатися між елементами у логічному порядку (зліва направо, зверху вниз) та активувати їх за допомогою клавіш Enter або Space.

У відповідності до критерію 2.1.1 (Клавіатура) та 2.1.2 (Без пасток фокусу) WCAG, всі функції сайту мають бути доступні через клавіатуру, а користувач не має "застрягати" в окремих елементах.

Стан у Beauty Diary:

Усі кнопки, посилання та поля форм мають логічний порядок фокусу (Tab-послідовність), що відповідає візуальному порядку елементів на сторінці. Користувач може переміщатися від хедера до основного контенту, потім до бічної панелі, а потім до футера без пропусків або стрибків.

Елементи мають чіткий візуальний індикатор фокусу (outline), що дозволяє користувачеві розуміти, де він знаходиться. Це особливо важливо для людей з порушеннями зору, які покладаються на візуальні підказки для навігації.

Реалізовано механізм "пропуску навігації" (skip navigation link). Це посилання, яке з'являється при першому натисканні Tab і дозволяє перестрибнути через повторювані блоки (наприклад, хедер з навігацією) одразу до основного контенту. Це значно прискорює навігацію для людей, які використовують клавіатуру.

Недоліки:

Випадаючі меню (наприклад, вибір категорії в редакторі поста або випадаюче меню профілю) не завжди повідомляють про свій стан "розгорнуто" / "згорнуто". Це ускладнює роботу з екранними диктофонами, оскільки користувач не розуміє, чи меню відкрите, чи закрите.

Деякі кастомні компоненти (наприклад, Tiptap-редактор для написання постів) мають обмежену клавіатурну навігацію всередині себе. Наприклад, вставка зображення або форматування тексту (жирний, курсив) вимагають використання миші або додаткових комбінацій клавіш, які не завжди інтуїтивно зрозумілі.

Рекомендації:

Додати атрибути aria-expanded="true/false" до всіх випадаючих елементів для повідомлення про їхній стан.

Покращити клавіатурну навігацію всередині Tiptap-редактора: додати підказки про гарячі клавіші для форматування (наприклад, Ctrl+B для жирного, Ctrl+I для курсиву).

Забезпечити можливість закриття модальних вікон клавішею Escape.

5.2.2. Альтернативні тексти для зображень

Кожне зображення на сайті повинне мати описовий атрибут alt, який озвучується скрінрідерами. Це стосується як контентних зображень (обкладинки постів, аватари), так і декоративних (іконки, роздільники). Критерій 1.1.1 (Нетекстовий контент) WCAG вимагає, щоб для всіх нетекстових елементів були надані текстові альтернативи.

Стан у Beauty Diary:

Обкладинки постів мають автоматично генерований alt-текст, який формується з назви поста та імені автора. Наприклад, для поста "Мій догляд за шкірою" від

авторки "Анна" генерується alt-текст "Обкладинка поста 'Мій догляд за шкірою' від Анни".

Аватари користувачів мають alt-текст з нікнеймом користувача, що дозволяє ідентифікувати особу в коментарях та стрічці.

Іконки (лайк, коментар, збереження, сповіщення) мають атрибути aria-label, які пояснюють їхню функцію. Наприклад, кнопка лайка має aria-label="Поставити лайк".

Декоративні зображення (роздільники, фонові елементи) мають порожній alt-текст (alt=""), що дозволяє скрінрідерам пропускати їх і не захаращувати озвучення.

Рекомендації:

Додати можливість для авторів вручну вказувати alt-текст для обкладинок під час створення поста. Це дозволить більш точно описувати зображення, особливо якщо воно містить важливу інформацію (наприклад, текстуру крему або колір тіней).

Переконатися, що автоматичний alt-текст не перевищує рекомендовану довжину (близько 125 символів) і не є надто формальним.

Регулярно перевіряти, чи всі нові зображення мають alt-текст.

5.2.3. Контрастність кольорів та колірна схема

Люди з порушеннями зору (наприклад, дальтонізмом) або з віковими змінами можуть не розрізняти кольори з низьким контрастом. Згідно з критерієм 1.4.3 (Мінімальна контрастність) WCAG, мінімальне співвідношення контрасту для тексту становить 4.5:1 для звичайного тексту та 3:1 для великого тексту (18 пунктів або більше). Для некстових елементів (іконки, кнопки) також рекомендовано достатній рівень контрасту.

Стан у Beauty Diary:

Основні текстові елементи мають високий контраст: чорний текст (або темно-сірий) на білому фоні у світлій темі, та білий текст на темному фоні у темній

темі. Це забезпечує співвідношення контрасту близько 15:1, що значно перевищує мінімальні вимоги.

Кольори не використовуються як єдиний спосіб передачі інформації. Наприклад, посилання завжди підкреслені або мають додаткове маркування (зміна кольору + підкреслення при наведенні). Це важливо для дальтоніків, які можуть не розрізняти синій та фіолетовий кольори.

Підтримка темної та світлої теми дозволяє користувачам обирати найбільш комфортний варіант для своїх очей, що особливо важливо для людей зі світлобоязню або мігренями.

Елементи з низьким пріоритетом (наприклад, дата публікації або кількість переглядів) мають меншу контрастність, але все ще в межах допустимого (близько 5:1).

Недоліки:

Деякі кольорові акценти (наприклад, рожева кнопка "Опублікувати") мають недостатню контрастність тексту на фоні кнопки. Білий текст на рожевому фоні має контрастність близько 4:1, що трохи нижче рекомендованого 4.5:1 для звичайного тексту.

Рекомендації:

Перевірити контрастність всіх кольорових акцентів за допомогою інструментів, таких як WebAIM Contrast Checker або браузерних розширень (наприклад, axe DevTools).

Дещо затемнити рожевий колір кнопок, щоб збільшити контрастність білого тексту до рекомендованих 4.5:1.

Як альтернатива – використовувати темніший текст на світлих кнопках (наприклад, рожевий фон з темно-червоним текстом).

5.2.4. Робота з екранними диктофонами (Screen Readers)

Екранні диктофони (NVDA, JAWS, VoiceOver) озвучують вміст сторінки у тому порядку, в якому він побудований у HTML-коді. Правильна семантична

розмітка (використання тегів <header>, <main>, <nav>, <h1>-<h6>) є критичною для коректного читання. Критерій 1.3.1 (Інформація та зв'язки) та 4.1.2 (Ім'я, роль, значення) WCAG вимагають, щоб всі елементи мали правильну семантику та були доступні для допоміжних технологій.

Стан у Beauty Diary:

Використовується семантична HTML-розмітка: заголовки (<h1> для назви поста, <h2> для секцій), списки (<ul> для тегів, <ol> для кроків), секції (<section> для окремих блоків).

Усі інтерактивні елементи мають ролі (role="button" для кастомних кнопок, role="link" для посилань) та зрозумілі назви (aria-label або текст всередині елемента).

Використовується бібліотека shadcn/ui, яка має вбудовану підтримку доступності (ARIA-атрибути, управління фокусом, обробка клавіатурних подій). Це значно зменшує кількість ручних налаштувань доступності.

Динамічні оновлення (наприклад, поява нового сповіщення) супроводжуються повідомленнями для скрінрідера через aria-live="polite".

Недоліки:

Tiptap-редактор (використовується для створення постів) не завжди коректно передає фокус та стан. Наприклад, переміщення між інструментами форматування (жирний, курсив) не завжди супроводжується оголошенням зміни стану для скрінрідера.

Сповіщення в дзвіночку, які з'являються автоматично, не завжди автоматично оголошуються скрінрідером. Користувач може не помітити нове сповіщення, якщо не перевіряє дзвіночок вручну.

Деякі кастомні компоненти (наприклад, модальне вікно підтвердження видалення) мають обмежену підтримку скрінрідерів – фокус не завжди автоматично переміщується у вікно при його появі.

Рекомендації:

Використовувати aria-live="assertive" для критичних повідомлень (наприклад, помилки валідації форми), щоб вони озвучувалися негайно.

Забезпечити автоматичне переміщення фокусу у модальні вікна при їх відкритті та повернення фокусу на попередній елемент при закритті.

Додати aria-label для всіх інтерактивних елементів, які не мають текстового вмісту.

Перевірити роботу Tiptap-редактора з екранними диктофонами та виправити виявлені проблеми.

5.2.5. Керування рухомим контентом та анімаціями

У соціальних мережах часто використовуються каруселі, анімації та автоматичні оновлення. Важливо, щоб користувач міг їх контролювати. Критерій 2.2.2 (Пауза, зупинка, приховування) WCAG вимагає, щоб рухомий контент, який автоматично оновлюється, можна було призупинити або зупинити.

Стан у Beauty Diary:

Автоматичне оновлення стрічки (через WebSocket) не перериває роботу користувача та не змінює фокус. Нові пости з'являються у верхній частині стрічки без різких стрибків.

Анімації мають налаштування prefers-reduced-motion, що дозволяє автоматично вимкнути їх для користувачів з вестибулярними розладами або чутливістю до руху. Це відповідає критерію 2.3.3 (Анімація з контенту).

Завантаження нових даних супроводжується легкими анімаціями-індикаторами, які не є відволікаючими.

Недоліки

Відсутня кнопка "призупинити" для автоматичних оновлень. Користувач, який читає пост, може бути перерваний новими сповіщеннями або постами у стрічці.

Деякі анімації (наприклад, поява нового сповіщення в дзвіночку) не враховують налаштування prefers-reduced-motion та відтворюються у будь-якому випадку.

Рекомендації:

Додати опцію "призупинити оновлення" для автоматичних оновлень у стрічці.

Переконатися, що всі анімації мають prefers-reduced-motion та плавно затухають, а не стрибають.

Дозволити користувачам налаштовувати частоту сповіщень (наприклад, "не турбувати" на 1 годину).

5.3. Доступність на мобільних пристроях та жести

Згідно з дослідженнями Statista, понад 70% користувачів соціальних мереж заходять на платформи з мобільних пристроїв. Тому адаптивність та доступність на дотикових екранах є критичними. Критерії 1.4.10 (Зміна розміру вікна), 1.4.12 (Інтервали між рядками) та 2.5.5 (Розмір цілі) WCAG визначають вимоги до мобільної доступності.

Стан у Beauty Diary:

Адаптивний дизайн на основі Tailwind CSS дозволяє сайту коректно відображатися на екранах різних розмірів. Сітка постів автоматично змінює кількість колонок: 3 на десктопі, 2 на планшеті, 1 на телефоні.

Клікабельні елементи (кнопки, посилання) мають розмір не менше 44×44 пікселів (близько 7-10 мм), що відповідає рекомендаціям для пальцевих пристроїв та критерію 2.5.5 (Розмір цілі).

Меню на мобільних пристроях згортається в "гамбургер", що економить простір та робить навігацію зручнішою для користувачів з малими екранами.

Шрифти масштабуються коректно при збільшенні (відповідають критерію 1.4.4 (Зміна розміру тексту)).

Недоліки:

У деяких формах відстань між кнопками менша за рекомендовані 8-10 пікселів, що може призводити до випадкових натискань на мобільних пристроях.

Редактор Tiptap на мобільному пристрої потребує додаткової оптимізації для сенсорного введення. Наприклад, панель інструментів занадто мала для пальців, а деякі елементи (вставка зображення) не зручні для використання на дотик.

Жести (наприклад, свайп для повернення назад) не підтримуються на рівні сайту.

Рекомендації:

Збільшити відступи (padding) між кнопками до мінімум 8 пікселів.

Покращити мобільну версію Tiptap-редактора: збільшити розмір кнопок інструментів, додати підтримку мультитач жестів.

Додати підтримку свайп-жестів для навігації між сторінками.

5.4. Інструменти перевірки доступності

Для перевірки відповідності Beauty Diary стандартам WCAG 2.1 використовувалися наступні інструменти:

WAVE (Web Accessibility Evaluation Tool) – автоматичний інструмент, який аналізує сторінку та показує помилки, попередження та структурні елементи. Використовувався для швидкого виявлення проблем, таких як відсутність alt-текстів або недостатня контрастність.

axe DevTools – розширення для браузера, яке інтегрується в інструменти розробника. Дозволяє виконувати детальний аудит сторінки за критеріями WCAG 2.1 рівнів A та AA. Використовувався для перевірки ARIA-атрибутів та семантичної розмітки.

NVDA (NonVisual Desktop Access) – безкоштовний екранний диктофон для Windows. Використовувався для ручного тестування з читанням сторінок вголос, перевірки навігації та керованості.

VoiceOver – вбудований екранний диктофон у macOS та iOS. Використовувався для тестування доступності на мобільних пристроях та перевірки коректності озвучення.

WebAIM Contrast Checker – онлайн-інструмент для перевірки контрастності кольорів. Використовувався для виявлення кольорових комбінацій, що не відповідають вимогам WCAG.

5.5. Висновки до розділу 5

У п'ятому розділі було проведено комплексний аналіз вебдоступності застосунку Beauty Diary відповідно до міжнародного стандарту WCAG 2.1 рівня AA.

Основні результати аналізу показали, що:

Сайт має функціональну клавіатурну навігацію з чітким візуальним фокусом та механізмом пропуску навігації, що відповідає критеріям 2.1.1 та 2.1.2.

Забезпечено альтернативні тексти для більшості зображень, хоча деякі аспекти (автоматичний alt-текст) потребують доопрацювання для повної відповідності критерію 1.1.1.

Контрастність кольорів відповідає вимогам рівня AA (критерій 1.4.3), але кольорові акценти вимагають додаткової перевірки та покращення.

Семантична структура HTML та ролі компонентів сприяють коректній роботі екранних диктофонів (критерії 1.3.1 та 4.1.2).

Адаптивний дизайн забезпечує зручність на мобільних пристроях, хоча деякі елементи (відстань між кнопками, Tiptap-редактор) потребують оптимізації.

Реалізовано підтримку prefers-reduced-motion для зменшення анімацій (критерій 2.3.3).

Виявлено наступні недоліки:

- Випадаючі меню не оголошують свій стан для скрінрідерів.
- Контрастність кнопок-акцентів незначно нижча за рекомендовану.

- Сповіщення в дзвіночку не завжди автоматично озвучуються скрінрідерами.
- Tiptap-редактор має обмежену клавіатурну навігацію.
- Відсутня кнопка призупинення для автоматичних оновлень.

Складено детальний план покращення доступності з визначенням пріоритетів, відповідних критеріїв WCAG та термінів виконання. Для перевірки відповідності використовувалися автоматичні та ручні інструменти: WAVE, axe DevTools, NVDA, VoiceOver та WebAIM Contrast Checker.

Таким чином, Beauty Diary має хорошу базу для відповідності рівню AA, але потребує подальшого доопрацювання для повної відповідності стандартам вебдоступності. Впровадження цих покращень підвищить зручність для всіх категорій користувачів, включно з людьми з інвалідністю, що є важливим аспектом соціальної відповідальності сучасного вебзастосунку. Крім того, покращення доступності позитивно вплине на SEO-показники, зменшить ризик судових позовів та зробить Beauty Diary більш привабливим для широкої аудиторії.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було спроєктовано та розроблено повнофункціональний вебзастосунок «Beauty Diary» – інтерактивний жіночий блог з елементами соціальної мережі. Розроблений продукт дозволяє користувачам реєструватися з підтвердженням email, створювати та публікувати пости про красу, догляд за шкірою, стиль та wellness, взаємодіяти з контентом інших авторів через коментарі (з підтримкою гілок та soft-delete), лайки, підписки та збереження, а також отримувати миттєві сповіщення в дзвіночок та на email. Адміністратори отримали окрему панель для блокування користувачів, зміни ролей та перегляду audit-логів.

Відповідно до поставленої мети в роботі вирішено такі завдання.

Аналіз предметної області та існуючих аналогів. Проведено детальний аналіз трьох популярних платформ – Medium, Pinterest та Instagram. Виявлено, що Medium пропонує якісний текстовий досвід, але не має гіллястих коментарів та real-time сповіщень. Pinterest зручний для збереження візуального контенту, але не підходить для довгих текстових постів. Instagram дає миттєву взаємодію, але обмежує довжину тексту та не має кастомних категорій для блогу. Жодна з цих платформ не поєднує всі необхідні функції, що підтверджує актуальність створення власного рішення.

Формулювання вимог. На основі проведеного аналізу сформульовано 17 функціональних вимог (реєстрація з OTP, створення постів, лайки, коментарі, підписки, сповіщення, адміністрування тощо) та 10 нефункціональних вимог (час завантаження сторінки до 1.5 секунди, підтримка 1000 одночасних користувачів, доступність 99.5% тощо). Ці вимоги стали основою для подальшого проєктування та розробки.

Обґрунтування вибору технологій. На основі порівняльного аналізу обрано наступний технологічний стек. Для фронтенду – Next.js 15 (забезпечує Server Components для зменшення JS-бандлу, rewrites для вирішення проблеми cross-origin cookie, та вбудовану підтримку i18n). Для бекенду – NestJS 11 (має вбудований DI, модульну структуру, офіційну підтримку CQRS та WebSocket). Для бази даних –

PostgreSQL 16 (реляційна модель для складних зв'язків, повнотекстовий пошук через tsvector, тригери для денормалізації лічильників). Для валідації – Zod у спільному пакеті (забезпечує єдиний контракт між фронтом і беком). Для real-time – Socket.IO (двосторонній канал з фолбеком на long-polling).

Проєктування архітектури. Створено багаторівневу архітектуру з монорепозиторієм на pnpm workspaces, що включає три пакети: apps/web (фронтенд), apps/api (бекенд) та packages/shared (спільні типи). На бекенді реалізовано гексагональну архітектуру (ports & adapters), яка ізолює доменну логіку від зовнішніх залежностей, що дозволяє легко замінювати БД або email-провайдера без зміни бізнес-логіки. Застосовано патерн CQRS для розділення операцій запису (команди) та читання (запити), що полегшує тестування та масштабування. Реалізовано outbox-патерн для надійної доставки email – завдання зберігаються в черзі, а cron-процес кожні 30 секунд обробляє їх з повторними спробами при невдачах

Розробка бази даних. Спроєктовано ER-діаграму з 13 основними сутностями: users, posts, comments, post_likes, post_favorites, user_follows, notifications, email_outbox, audit_log та інші. Створено SQL-міграції з використанням принципу append-only (вже закомічену міграцію не редагують, зміни вносяться лише через нові міграції). Реалізовано тригери для автоматичного оновлення денормалізованих лічильників (кількість лайків під постом оновлюється при додаванні/видаленні лайка). Додано GIN-індекси на tsvector для повнотекстового пошуку по заголовку та змісту постів, що забезпечує швидкий пошук навіть при великій кількості записів.

Реалізація бекенду. Бекенд реалізовано на NestJS 11 з використанням гексагональної архітектури та CQRS. Код організовано у вигляді 10 модулів: auth (реєстрація з OTP, логін, refresh токенів), users (профілі, підписки), posts (CRUD постів, категорії, теги, пошук), comments (гіллясті коментарі, soft-delete), reactions (лайки, збережені), follows (підписки), notifications (in-app сповіщення, outbox для email), realtime (WebSocket-шлюз), media (завантаження зображень), admin (адміністрування, audit-лог). Реалізовано WebSocket-шлюз на Socket.IO з кімнатами user:{id} для особистих сповіщень та post:{id} для оновлень поста.

Реалізація фронтенду. Фронтенд реалізовано на Next.js 15 з використанням App Router. Сторінки організовані за логічними групами: головна стрічка, персоналізована стрічка, детальний перегляд поста, створення поста (з редактором Tiptap), профіль користувача, збережені пости, авторизація, адмін-панель. Інтерфейс стилізовано за допомогою Tailwind CSS [8] v4 (утилітарний CSS, CSS-first конфігурація через дизайн-токени) та shadcn/ui (копійовані примітиви). Реалізовано підтримку української та англійської мов через next-intl, а також темної та світлої теми через next-themes. Для керування серверними даними використано TanStack Query (кешування, дедуплікація, автоматична інвалідація), для форм – react-hook-form з інтеграцією Zod.

Спільний пакет Zod. Створено пакет @beauty-diary/shared, який містить усі Zod-схеми для DTO, TypeScript-типи, виведені з цих схем, константи (ліміти довжини рядків, назви категорій, статуси) та enum-типи (UserRole, NotificationType, PostStatus). Цей пакет використовується як на бекенді (через ZodValidationPipe для валідації вхідних запитів), так і на фронтенді (через zodResolver для react-hook-form). Такий підхід гарантує, що правила валідації однакові на обох кінцях, і унеможливлює ситуацію, коли фронт дозволяє значення, яке бек відхиляє.

Безпека. Реалізовано багаторівневу модель безпеки. Для хешування паролів використано argon2id – найсучасніший алгоритм, стійкий до GPU-атак. JWT-токени (access на 15 хвилин, refresh на 30 днів) зберігаються в httpOnly cookies, що унеможливлює їх крадіжку через XSS-атаки. Реалізовано refresh token rotation з reuse detection – при кожному оновленні видається нова пара токенів, старий стає недійсним, а якщо хтось спробує використати вже відкликаний – сесія анулюється. Реєстрація вимагає OTP-підтвердження email (6-значний код, діє 10 хвилин, максимум 5 спроб, повторне надсилання через 60 секунд). Додатково налаштовано CORS (дозволені запити лише з домену фронтенду), Helmet (безпечні HTTP-заголовки), rate limiting (обмеження кількості запитів).

Тестування. Впроваджено багаторівневу стратегію тестування. Юніт-тести (Jest для бекенду, Vitest для фронтенду) перевіряють окремі функції, класи та компоненти ізольовано від зовнішніх залежностей. Інтеграційні тести піднімають повний NestJS-застосунок з реальною базою даних PostgreSQL в Docker та виконують HTTP-запити через Supertest, покриваючи основні сценарії: реєстрацію, логін, створення постів, коментування, лайки, підписки, сповіщення. Завдяки тестуванню було виявлено та виправлено кілька критичних помилок на ранніх етапах розробки.

CI/CD та деплой. Налаштовано GitHub Actions для безперервної інтеграції: при кожному push або PR запускаються чотири паралельні jobs – lint (перевірка форматування та якості коду), typecheck (перевірка TypeScript), test (запуск тестів з реальними контейнерами PostgreSQL та LocalStack), build (продакшн-збірка). Для локальної розробки використовується Docker Compose з трьома контейнерами: PostgreSQL (БД), LocalStack (емуляція S3 для зображень), Mailpit (локальний email-сервер). Застосунок розгорнуто на безкоштовних хмарних платформах: Vercel (фронтенд), Render (бекенд), Neon (PostgreSQL), Cloudflare R2 (зображення), Brevo (email). Проблему cross-origin cookie вирішено через rewrites у Next.js, які проксують всі запити на /api/* та /socket.io/* з Vercel на Render.

Практичне значення роботи полягає в тому, що розроблений вебзастосунок «Beauty Diary» є готовим до експлуатації продуктом. Він може бути використаний як платформа для ведення жіночого блогу з елементами соціальної мережі, а також як основа для подальшого розвитку (додавання системи рекомендацій, мобільного застосунку, інтеграції з відеохостингами, платних підписок тощо). Отримані архітектурні рішення – гексагональна архітектура, CQRS, outbox-патерн, shared-пакет Zod-схем, організація монорепозиторію з pnpm workspaces – можуть бути використані як шаблон для розробки аналогічних full-stack вебзастосунків зі складною бізнес-логікою, real-time функціоналом та високими вимогами до безпеки.

Перспективи подальшого розвитку. Незважаючи на те, що Beauty Diary є повністю функціональним продуктом, він має потенціал для подальшого вдосконалення. Планується додати систему рекомендацій постів на основі

машинного навчання (аналіз вподобань користувача, історії лайків та підписок), розробити мобільний застосунок на React Native для зручності використання з телефону, інтегрувати можливість вбудовування відео з YouTube та TikTok, додати розширену аналітику для авторів (перегляди, унікальні відвідувачі, географія аудиторії, час читання), а також реалізувати систему платних підписок (щоб автори могли монетизувати свій контент) та приватних спільнот (закриті блоги за запрошенням).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Statista. Digital Media Outlook 2026. — URL: https://www.statista.com
2. Next.js. Documentation. — URL: https://nextjs.org/docs
3. NestJS. Documentation. — URL: https://docs.nestjs.com
4. PostgreSQL Global Development Group. PostgreSQL 16 Documentation. — URL: https://www.postgresql.org/docs/16/
5. Zod. Documentation. — URL: https://zod.dev
6. Cloudflare. R2 Documentation. — URL: https://developers.cloudflare.com/r2
7. Socket.IO. Documentation. — URL: https://socket.io/docs/v4
8. Tailwind CSS. Documentation (v4). — URL: https://tailwindcss.com/docs
9. shadcn/ui. Documentation. — URL: https://ui.shadcn.com
10. ДСанПіН 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин.
11. ДБН В.2.5-28-2018. Природне і штучне освітлення.
12. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення.