m-math_2025_018

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**
**ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА**

**Факультет математики та інформатики**
**кафедра математичного моделювання**

# Створення сервісу оптимізації сайтів для пошукових систем: архітектура та фронтенд

**Кваліфікаційна робота**

**Рівень вищої освіти – другий (магістерський)**

***Виконав:***
студент 6 курсу, 601 групи
**Циганаш Вадим Віталійович**
***Керівник**:*
доцент, канд. фіз.-мат. наук
Перцов А. С.

***До захисту допущено***
***на засіданні кафедри***
***протокол №5 від 26 листопада 2024 р.***
***Зав. кафедрою _____ проф. Черевко І.М.***

**Чернівці – 2024**

***Анотація***

Розроблено вебсервіс для автоматизації оптимізації пошукових систем (SEO) вебсайтів. Основна увага приділяється архітектурі та фронтенду сервісу. Сервіс включає інструменти для створення, планування та публікації статей, інтеграції з популярними CMS (Webflow, Strapi, WordPress), а також функціонал для роботи з ключовими словами. Фронтенд сервісу розроблено на основі Next.js, що також виконує роль backend для авторизації користувачів, обробки платежів через Stripe і управління проєктами. Логіка генерації контенту реалізована на окремому сервері, що взаємодіє із загальною базою даних через Prisma. Вебсервіс забезпечує інтуїтивно зрозумілий інтерфейс для створення SEO-статей, автоматичного індексування в Google Search Console і покращення пошукової видимості вебсайтів користувачів.

***Ключові слова:*** автоматизація SEO, пошукова оптимізація, генерація контенту, інтеграція з CMS, індексація в Google, управління контентом.

***Abstract***

The web service was developed to automate search engine optimization (SEO) for websites. The focus is on the service's architecture and frontend. Service includes tools for creating, scheduling, and publishing articles, integration with popular CMSs (Webflow, Strapi, WordPress), and keyword management functionality. The service's frontend is based on Next.js, which also acts as a backend for user authorization, payment processing via Stripe, and project management. The content generation logic is implemented on a separate server that interacts with a shared database using Prisma. The service enables SEO automation, improves search indexing, optimizes keyword management, and simplifies content workflows.

**Keywords:** SEO automation, search engine optimization, content generation, CMS integration, Google indexing, content workflow.

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

_____ В. В. Циганаш

# Зміст

## Вступ

### Актуальність роботи

У сучасному світі якісна оптимізація вебсайтів для пошукових систем (SEO) є одним із ключових елементів успішного функціонування бізнесу в інтернеті. Пошукові системи, такі як Google, виступають основним каналом залучення органічного трафіку, що безпосередньо впливає на популярність і прибутковість вебсайтів.

Зростаюча конкуренція у сфері SEO вимагає ефективних інструментів автоматизації. Ручне написання статей, управління ключовими словами та публікація контенту вимагають значних ресурсів і часу. Крім того, інтеграція вебсайтів із системами управління контентом (CMS) часто ускладнена через різноманітність платформ і форматів даних.

Розроблений вебсервіс спрямований на вирішення цих проблем, забезпечуючи користувачам зручні інструменти для автоматизації генерації, планування та публікації SEO-контенту, покращуючи видимість вебсайтів у пошукових системах.

**Мета.** Метою кваліфікаційної роботи є:

✔ розробка сервісу для автоматизації пошукової оптимізації (SEO) вебсайтів;

✔ створення багатофункціонального інтерфейсу для роботи з генерацією контенту, ключовими словами та публікацією статей;

✔ забезпечення інтеграції з популярними CMS (Webflow, Strapi, WordPress) та автоматизація індексації в пошукових системах.

**Завдання роботи.** Для досягнення мети необхідно вирішити такі завдання:

Під час роботи над кваліфікаційною роботою було розв’язано такі задачі:

✔ розробити архітектуру сервісу, яка включає два окремі бекенди з доступом до спільної бази даних;

✔ реалізувати фронтенд на основі Next.js з функціоналом реєстрації, авторизації, управління проєктами та інтеграції платежів через Stripe;

✔ забезпечити інтеграцію з CMS і можливість гнучкого налаштування мапінгу полів для статей;

✔ розробити функціонал для роботи з ключовими словами (планування, аналіз, створення контенту);

✔ протестувати сервіс для забезпечення його стабільності, зручності використання та продуктивності.

**Об’єкт дослідження** – Об’єктом дослідження є методи автоматизації пошукової оптимізації вебсайтів, сучасні підходи до інтеграції з CMS і технології генерації контенту на основі штучного інтелекту.

**Практичне застосування** Розроблений вебсервіс має широке практичне застосування в галузі пошукової оптимізації вебсайтів. Він може бути використаний компаніями, власниками вебсайтів, SEO-агентствами та маркетологами для автоматизації процесів створення, планування та публікації контенту.

Вебсервіс дозволяє:

1. **Автоматизувати створення контенту:**
    - Генерація SEO-оптимізованих статей за допомогою OpenAI API скорочує час і зусилля, необхідні для створення якісного матеріалу.
    - Інтеграція ключових слів та автоматичне формування метаописів покращують індексацію статей у пошукових системах.
2. **Спрощувати управління контентом:**

- Інтеграція з популярними CMS (Webflow, Strapi, WordPress) дозволяє швидко та безпечно публікувати статті на веб сайтах, незалежно від платформи.
- Функціонал мапінгу полів забезпечує адаптацію до унікальних структур даних кожного користувача.

3. **Покращувати SEO-результати:**
   - Автоматичне індексування в Google Search Console сприяє швидкому виявленню контенту пошуковими системами.
   - Аналіз ключових слів (обсяг пошуку, конкурентність) допомагає будувати ефективну SEO-стратегію.
4. **Підвищувати ефективність маркетингових кампаній:**
   - Масштабованість сервісу дозволяє працювати з великими обсягами контенту, планувати публікацію статей на тривалий період і відстежувати результати.
   - Інтеграція додаткових функцій, таких як робота з ключовими словами та аналітика, покращує стратегічний підхід до SEO.

Завдяки вебсервісу користувачі отримують потужний інструмент для автоматизації SEO, що не лише зменшує витрати часу й ресурсів, але й забезпечує стабільне зростання видимості їхніх вебсайтів у пошукових системах.

Під час написання дипломної роботи використано інтернет ресурси за посиланнями [1-17].

# Розділ 1. Використані технології

## 1.1. Основні технології для розробки фронтенду та архітектури

### 1.1.1. Next.js

Next.js — це один із найпотужніших і найпоширеніших сучасних фреймворків для розробки вебзастосунків на основі React. Він був створений компанією **Vercel** у 2016 році з метою спрощення розробки вебсайтів, які мають високі вимоги до продуктивності, SEO-оптимізації та масштабованості. Завдяки своїм інноваційним можливостям, Next.js швидко здобув популярність серед розробників по всьому світу.

Назва Next.js підкреслює його орієнтованість на створення вебзастосунків "нового покоління", які поєднують сучасний дизайн, швидкодію та адаптивність. Фреймворк забезпечує простий і гнучкий підхід до створення багатофункціональних вебсайтів.

Однією з головних особливостей Next.js є підтримка **серверного рендерингу (SSR)** і **статичної генерації сторінок (SSG)**. Це дозволяє знижувати час завантаження вебсайтів, що особливо важливо для SEO-оптимізації. Завдяки цьому вебсайти, створені на базі Next.js, отримують вищі рейтинги у пошукових системах і забезпечують покращений досвід користувача.

Ще однією важливою перевагою Next.js є вбудована підтримка **API-роутів**. Це означає, що розробники можуть створювати backend-ендпоінти в межах одного додатка, спрощуючи архітектуру проєкту. Крім того, Next.js підтримує динамічну маршрутизацію, яка дозволяє легко створювати складні структури сторінок.

Фреймворк пропонує широкі можливості для інтеграції з іншими технологіями, такими як TypeScript, Tailwind CSS і Redux. Він також чудово

працює у зв’язці з хмарними платформами, наприклад, **Vercel**, яка забезпечує автоматичне розгортання додатків, масштабування та підтримку функцій серверного рендерингу.

Next.js активно використовують великі компанії, такі як Netflix, GitHub, TikTok та Uber, що підкреслює його надійність і функціональність.

У рамках розробки вебсервісу Next.js застосовується для створення як фронтенду, так і частини бекенду. Це дозволяє реалізувати функції реєстрації, авторизації, управління проєктами, взаємодії з CMS і обробки платежів. Серверний рендеринг забезпечує високі показники продуктивності, що критично для SEO-сервісів.

Next.js — це потужний інструмент для розробки сучасних вебдодатків, який поєднує продуктивність, зручність і масштабованість. Завдяки його використанню, вебсервіс здатний забезпечити своїм користувачам швидкий і зручний доступ до функціональності сервісу.

### 1.1.2. TypeScript

**TypeScript** — це мова програмування, яка є надбудовою над JavaScript і додає до нього статичну типізацію. Створена компанією **Microsoft** у 2012 році, TypeScript швидко стала популярною серед розробників завдяки можливості створювати більш стабільні та масштабовані застосунки. Її основна мета — розширити функціональність JavaScript і зробити процес розробки зручнішим, особливо для великих проєктів.

Назва TypeScript підкреслює головну особливість мови — роботу з типами. Вона дозволяє розробникам визначати типи даних для змінних, функцій і об’єктів. Це значно зменшує кількість помилок, пов’язаних із використанням неправильних типів даних, і робить код зрозумілішим.

Однією з головних переваг TypeScript є його сумісність із JavaScript. Код, написаний на TypeScript, компілюється у звичайний JavaScript, що дозволяє використовувати його в будь-якому середовищі, де працює JavaScript, включаючи браузери та сервери. Це робить перехід на TypeScript простим і доступним для розробників, знайомих із JavaScript.

TypeScript підтримує сучасні функції JavaScript, а також додає власні можливості, такі як:

1. **Інтерфейси та типи:** дозволяють чітко визначати структуру даних.
2. **Модулі та простори імен:** спрощують організацію коду у великих проєктах.
3. **Декоратори:** забезпечують розширення можливостей класів.
4. **Автозаповнення коду:** завдяки статичній типізації, інтегровані середовища розробки (IDE) можуть пропонувати підказки та виявляти помилки в реальному часі.

TypeScript є строго типізованою мовою, що означає, що типи даних визначаються на етапі компіляції, а не виконання. Це допомагає виявляти помилки ще до запуску коду, що особливо важливо для великих командних проєктів.

Завдяки підтримці об'єктно-орієнтованого програмування (ООП) TypeScript пропонує такі функції, як класи, інтерфейси, наслідування та абстракція, що робить його привабливим для розробників зі світу традиційних мов, таких як C# чи Java.

У рамках проєкту TypeScript використовується як основна мова програмування. Завдяки статичній типізації TypeScript допомагає уникати помилок у коді, покращує його читабельність і спрощує підтримку. Наприклад, у проєкті TypeScript застосовується для роботи з компонентами Next.js, API-запитами через tRPC і Prisma, а також для валідації даних через бібліотеку Zod.

TypeScript — це вибір розробників, які прагнуть створювати надійні, масштабовані та зрозумілі вебзастосунки. Його використання в вебсервісі забезпечує стабільність коду, що є важливим для проєкту такого масштабу та функціональності.

### 1.1.3. tRPC

**tRPC** (Type-Safe Remote Procedure Call) — це інноваційна бібліотека, яка дозволяє створювати типобезпечні API між сервером і клієнтом без необхідності вручну писати REST або GraphQL ендпоінти. Вона розроблена для розробників, які працюють з TypeScript, і значно спрощує обмін даними в застосунках.

Головною ідеєю tRPC є повна інтеграція типів між сервером і клієнтом. Це означає, що розробники можуть використовувати один і той самий набір типів для опису запитів і відповідей, без потреби у додаткових контрактних бібліотеках, як у GraphQL або Swagger.

**Основні особливості tRPC:**

1. **Типобезпечність від початку до кінця:**

Типи, визначені на сервері, автоматично доступні на клієнті. Це виключає ризик типових помилок у запитах і відповідях.

2. **Простота інтеграції:**

tRPC легко інтегрується з сучасними фреймворками, такими як Next.js, завдяки своїй простоті і нативній підтримці серверних API.

3. **Відсутність зайвих залежностей:**

На відміну від REST або GraphQL, tRPC не потребує створення додаткових схем чи контрактів. Це зменшує обсяг зайвого коду та покращує продуктивність розробників.

4. **Масштабованість:**

tRPC підтримує модульний підхід, що дозволяє легко організовувати API навіть у великих застосунках.

5. **Висока продуктивність:**

Завдяки своїй легковажній архітектурі tRPC забезпечує швидку взаємодію між клієнтом і сервером.

Як працює tRPC?

- tRPC використовує **процедурні виклики** замість традиційного маршрутизаційного підходу REST або GraphQL.
- Розробник створює "роути" (API-методи) на сервері, які клієнт може викликати як звичайні функції. Типи параметрів і відповідей автоматично синхронізуються між обома сторонами.

tRPC — це сучасний інструмент для створення API, який поєднує простоту, типобезпечність і продуктивність. Його використання забезпечує ефективний обмін даними між фронтендом і бекендом, значно покращуючи загальний процес розробки.

### 1.1.4. Prisma ORM

**Prisma** — це сучасний інструмент для роботи з базами даних, який належить до категорії ORM (Object-Relational Mapping). Він дозволяє розробникам працювати з базами даних на високому рівні, використовуючи простий і зрозумілий API, замість написання складних SQL-запитів. Prisma

орієнтований на розробників, які використовують TypeScript і JavaScript, і пропонує зручний спосіб роботи з даними в застосунках.

Основні особливості Prisma

1. **Декларативна схема бази даних:**

   Розробники визначають структуру бази даних у вигляді файлу схеми (`schema.prisma`), що значно спрощує розуміння і управління базою даних.

2. **Автоматична генерація клієнта:**

   Prisma автоматично створює клієнтський API для роботи з даними, забезпечуючи простий доступ до бази через методи, що відповідають структурам таблиць.

3. **Міграції бази даних:**

   Prisma дозволяє легко створювати та виконувати міграції для оновлення структури бази даних відповідно до змін у схемі.

4. **Підтримка популярних баз даних:**

   Prisma працює з багатьма реляційними (PostgreSQL, MySQL, SQLite, SQL Server) та нереляційними (MongoDB) базами даних.

5. **Сумісність із TypeScript:**

   Завдяки інтеграції з TypeScript Prisma забезпечує типобезпечність під час роботи з даними.

6. **Простота інтеграції:**

   Prisma легко інтегрується з сучасними фреймворками, такими як Next.js, і бібліотеками для роботи з API, як-от tRPC.

Як працює Prisma?

- Розробник описує структуру бази даних у схемі.
- На основі схеми Prisma генерує клієнтський API для виконання CRUD-операцій.
- Зміни в схемі автоматично синхронізуються з базою даних за допомогою міграцій.

Використання Prisma

Prisma використовується як основна ORM для роботи з базою даних, яка обслуговує як фронтенд, так і бекенд. Основні завдання Prisma в проєкті включають:

1. **Зберігання даних:**

   Збереження даних про користувачів, їхні проєкти, статті, ключові слова та підписки.

2. **Синхронізація двох бекендів:**

   Оскільки вебсервіс має два окремих бекенди, Prisma забезпечує спільний доступ до бази даних із узгодженою структурою.

3. **Міграції:**

   Зміни у схемі бази даних легко впроваджуються за допомогою Prisma, що дозволяє безпечно оновлювати структуру бази під час розширення функціоналу.

4. **Типобезпечність:**

   Завдяки TypeScript Prisma гарантує, що запити до бази даних відповідають її структурі, що знижує ризик помилок і полегшує розробку.

Переваги Prisma

1. **Простота використання:**

Чіткий і зрозумілий API прискорює розробку.

2. **Ефективне управління базою даних:**

   Зручні міграції спрощують роботу зі складними структурами даних.

3. **Типобезпечність:**

   Забезпечує узгодженість між сервером і базою даних.

4. **Масштабованість:**

   Легко адаптується до нових вимог проєкту, таких як додавання нових таблиць чи зв’язків.

Prisma ORM — це основа для ефективної та безпечної роботи з даними. Завдяки своїм можливостям він забезпечує стабільність і продуктивність бази даних, що критично важливо для сервісу, орієнтованого на великий обсяг інформації.

### 1.1.5. NextAuth.js

**NextAuth.js** — це потужна бібліотека для реалізації аутентифікації у застосунках на базі Next.js. Вона підтримує безпечний вхід користувачів через різні провайдери (Google, GitHub, Facebook тощо), а також через кастомні методи, такі як email і пароль.

Розроблена для спрощення впровадження автентифікації, NextAuth.js пропонує простий і гнучкий підхід до інтеграції з API, базами даних та інструментами керування сесіями.

**Основні особливості NextAuth.js:**

1. **Підтримка кількох провайдерів:**

NextAuth.js дозволяє використовувати популярні OAuth2-провайдери (Google, Facebook, Discord тощо) або налаштувати власний провайдер через `CredentialsProvider`.

2. **Гнучка конфігурація:**

Розробники можуть визначати, як обробляються сесії, JWT, callback-функції та інтеграція з базами даних.

3. **JWT і керування сесіями:**

Бібліотека підтримує як JSON Web Tokens (JWT), так і серверні сесії для збереження автентифікаційного стану.

4. **Інтеграція з базою даних:**

Через адаптери, наприклад, PrismaAdapter, NextAuth.js легко підключається до баз даних для збереження інформації про користувачів, акаунти та сесії.

5. **Підтримка TypeScript:**

Завдяки типовій сумісності, NextAuth.js чудово інтегрується у TypeScript-проєкти, забезпечуючи безпеку під час написання коду.

**Переваги NextAuth.js у вебсервісі:**

1. **Швидка інтеграція:**

Простий у налаштуванні й легко інтегрується з іншими технологіями.

2. **Безпека:**

Використання сучасних стандартів аутентифікації, таких як OAuth2 і JWT.

3. **Масштабованість:**

Підтримка кількох провайдерів і кастомних методів дозволяє адаптувати рішення до потреб проєкту.

NextAuth.js забезпечує надійну аутентифікацію, роблячи вебсервіс безпечним і зручним для користувачів. Його використання спрощує обробку входу та збереження сесій, що є критично важливим для масштабних і сучасних вебпроєктів.

## 1.2. Бібліотеки для роботи з UI та UX

### 1.2.1. Tailwind CSS

**Tailwind CSS** — це сучасний утилітарний CSS-фреймворк, який надає набір класів для швидкого створення адаптивного та кастомізованого дизайну інтерфейсу. Він розроблений для спрощення роботи з CSS та забезпечення високої продуктивності розробки.

Замість створення окремих стилів, як це робиться в традиційних CSS або SCSS, Tailwind CSS дозволяє використовувати готові класи безпосередньо в HTML. Це забезпечує гнучкість і швидкість роботи, оскільки розробник може зосередитися на функціоналі та структурі сайту, не витрачаючи час на написання зайвого коду.

**Основні особливості Tailwind CSS:**

1. **Утилітарний підхід:**

Кожен клас у Tailwind CSS відповідає за одну функцію, наприклад, встановлення кольору, розміру шрифту чи відступів. Це дозволяє швидко створювати складні дизайни, комбінуючи класи.

2. **Адаптивність за замовчуванням:**

Фреймворк має вбудовану підтримку адаптивного дизайну завдяки зручній системі брейкпоінтів (наприклад, `sm:`, `md:`, `lg:`).

3. **Конфігурованість:**

Tailwind CSS дозволяє розробникам кастомізувати стилі через файл конфігурації `tailwind.config.js`, додаючи унікальні кольори, шрифти, розміри тощо.

4. **Зменшення розміру CSS:**

За допомогою інструменту PurgeCSS Tailwind CSS видаляє невикористані класи у продакшн-збірці, що значно зменшує розмір фінального файлу стилів.

5. **Екосистема компонентів:**

Tailwind підтримує інтеграцію з бібліотеками готових компонентів, такими як Headless UI чи Shadcn/UI, для швидкого створення кнопок, модальних вікон, форм тощо.

**Переваги Tailwind CSS:**

1. **Швидкість розробки:**

Замість написання складного CSS коду, розробники використовують готові утилітарні класи безпосередньо в HTML.

2. **Легкість у підтримці:**

Завдяки декларативному підходу стильові зміни виконуються швидко та з мінімальними помилками.

3. **Зменшення залежності від зовнішніх бібліотек:**

Tailwind CSS дозволяє створювати унікальний дизайн без використання громіздких бібліотек, таких як Bootstrap.

**Чому Tailwind CSS ідеально підходить для вебсервісу:**

- **Швидке налаштування:** дозволяє створювати стильний і функціональний дизайн з мінімальними зусиллями.
- **Гнучкість:** адаптується до будь-якого стилю проєкту, завдяки чому інтерфейс виглядає сучасно та професійно.
- **Продуктивність:** мінімізує обсяг коду та покращує швидкість розробки.

Tailwind CSS забезпечує елегантний і адаптивний інтерфейс, що робить роботу з вебсервісом приємною та зрозумілою для користувачів.

### 1.2.2. @shadcn/ui

**@shadcn/ui** — це набір готових компонентів користувацького інтерфейсу (UI), створений для спрощення розробки інтерфейсів у проєктах. Він базується на **Radix UI**, що забезпечує доступність, адаптивність і високу якість базових компонентів. Основна перевага @shadcn/ui — це поєднання функціональності Radix UI з гнучкими стилями, які легко інтегруються з фреймворками, такими як Tailwind CSS.

**Radix UI як основа:**

Radix UI — це набір компонентів із доступністю за замовчуванням (ARIA-комплаєнтні), які можна кастомізувати під будь-який проєкт. Він надає фундаментальні компоненти, такі як діалоги, випадаючі меню, слайдери та інші інтерактивні елементи, що відповідають сучасним стандартам UX.

@shadcn/ui використовує Radix UI як базовий шар функціональності, додаючи до нього стилізацію, яка підходить для Tailwind CSS, що значно прискорює розробку.

**Основні особливості @shadcn/ui:**

1. **Підтримка Tailwind CSS:**

Компоненти стилізовані для роботи з Tailwind CSS, що дозволяє швидко та легко налаштовувати їх зовнішній вигляд.

2. **Базування на Radix UI:**

Використовує перевірену часом функціональність Radix UI, яка забезпечує доступність і високу якість інтерактивних елементів.

3. **Готовність до кастомізації:**

Кожен компонент легко адаптується під специфічні потреби дизайну.

4. **Зручність використання:**

Компоненти мають простий API, що забезпечує швидку інтеграцію навіть у складні проєкти.

5. **Відкритий код:**

Набір компонентів доступний як open-source, що дозволяє розробникам розширювати його функціональність або використовувати як базу для власних компонентів.

**Типові компоненти в @shadcn/ui:**

1. **Dialogs:** модальні вікна для відображення додаткової інформації або підтвердження дій.
2. **Dropdown Menus:** випадаючі меню для роботи з інтерактивними списками.
3. **Tooltips:** підказки для елементів інтерфейсу.
4. **Tabs:** панелі вкладок для організації контенту.
5. **Sliders:** слайдери для вибору значень у діапазоні.

**Переваги @shadcn/ui:**

1. **Швидкість розробки:**

Завдяки готовим компонентам розробники можуть зосередитися на функціональності, а не на дизайні.

2. **Доступність:**

Компоненти Radix UI забезпечують відповідність стандартам ARIA, що покращує UX для користувачів з обмеженими можливостями.

3. **Гнучкість:**

Висока кастомізація дозволяє легко адаптувати компоненти під унікальні вимоги проєкту.

4. **Інтеграція з Tailwind CSS:**

Спрощує стилізацію, дозволяючи створювати сучасні та адаптивні інтерфейси.

### 1.2.3. @dnd-kit

**@dnd-kit** — це сучасна бібліотека для реалізації функціоналу drag-and-drop у React-застосунках. Вона забезпечує гнучкість, високу продуктивність і простоту використання, дозволяючи створювати інтерактивні елементи, які користувач може переміщувати на екрані. Завдяки модульному підходу, @dnd-kit дозволяє налаштовувати поведінку drag-and-drop відповідно до потреб проєкту, включаючи визначення зон перетягування, обмеження областей, до яких можна переміщувати елементи, і багато іншого.

Однією з ключових переваг @dnd-kit є її доступність для користувачів із обмеженими можливостями. Вона дотримується стандартів доступності ARIA та забезпечує плавну інтеграцію з іншими бібліотеками для створення складних

і кастомізованих інтерфейсів. Завдяки легкості налаштувань і широкому набору функцій, @dnd-kit ідеально підходить для створення адаптивних і інтерактивних інтерфейсів, які потребують перетягування, сортування чи групування елементів.

### 1.2.4. Framer Motion

**Framer Motion** — це потужна бібліотека для створення плавних і високоякісних анімацій у React-застосунках. Завдяки простому синтаксису та багатому функціоналу, вона дозволяє розробникам створювати як базові, так і складні анімації для покращення користувацького досвіду. Framer Motion підтримує інтерактивність, наприклад, анімацію при наведенні, кліку чи перетягуванні елементів, що додає живості інтерфейсам.

Однією з головних переваг Framer Motion є його інтеграція з React, що дозволяє використовувати анімації як частину компонентів із динамічним оновленням стану. Також бібліотека підтримує фізично коректні анімації, такі як пружні ефекти чи затухання, що надає реалістичності рухам. Завдяки своїй простоті та гнучкості Framer Motion є вибором багатьох розробників для створення сучасних і привабливих інтерфейсів.

## 1.3. Інструменти для управління станом і формами

### 1.3.1. React Hook Form

**React Hook Form** — це популярна бібліотека для управління станом форм у React-застосунках. Вона забезпечує простий і ефективний спосіб обробки форм із мінімальним впливом на продуктивність. Основною перевагою React Hook Form є використання вбудованих React-хуків, що дозволяє уникнути

зайвого рендерингу компонентів і спрощує роботу з формами будь-якої складності.

Бібліотека підтримує інтеграцію з багатьма інструментами для валідації, включаючи Zod, Joi та Yup. Вона також має інтуїтивний API для обробки полів, валідації та управління помилками. React Hook Form підходить як для невеликих форм, так і для масштабних проєктів, що робить її універсальним вибором для розробників.

### 1.3.2. Zod

**Zod** — це потужна TypeScript-бібліотека для валідації даних і створення типобезпечних схем. Вона дозволяє легко визначати структури даних, перевіряти їх коректність і автоматично генерувати TypeScript-типи на основі описаних схем. Це робить Zod ідеальним інструментом для забезпечення безпеки даних як на фронтенді, так і на бекенді.

На відміну від багатьох інших бібліотек валідації, Zod має простий і декларативний синтаксис, який легко розширюється та інтегрується з іншими інструментами. Завдяки цьому розробники можуть уникнути поширених помилок, пов’язаних із некоректними даними, та спростити обмін інформацією між різними частинами застосунку.

### 1.3.3. Zustand

**Zustand** — це легка бібліотека для управління глобальним станом у React-застосунках. Її головною особливістю є простота та продуктивність: Zustand використовує мінімалістичний API і уникає зайвих рендерингів компонентів, що дозволяє зберігати високу швидкість роботи застосунку.

Бібліотека підтримує використання з TypeScript і має модульну архітектуру, що дозволяє організовувати код у великих проєктах. Zustand підходить як для невеликих проєктів, де потрібен простий глобальний стан, так і для масштабних систем із великою кількістю взаємодій між компонентами.

### 1.4. Бібліотеки для роботи з запитами та таблицями

#### 1.4.1. @tanstack/react-query

**@tanstack/react-query** — це потужна бібліотека для управління серверними запитами в React-застосунках. Вона забезпечує зручний інтерфейс для кешування, синхронізації та оновлення даних, що дозволяє оптимізувати роботу з API. Однією з ключових особливостей є автоматичне оновлення даних у реальному часі, наприклад, після змін у базі даних або помилок під час запитів.

Бібліотека надає розробникам можливості для роботи з фоновими завантаженнями, управління станом завантаження та обробки помилок, що значно спрощує побудову сучасних застосунків. @tanstack/react-query підтримує інтеграцію з TypeScript і легко масштабується, роблячи її ідеальним вибором для складних і динамічних систем.

#### 1.4.2. @tanstack/react-table

**@tanstack/react-table** — це високопродуктивна бібліотека для створення динамічних таблиць у React-застосунках. Вона побудована за принципом "голий API", тобто забезпечує лише функціональність таблиць, залишаючи розробникам повну свободу у стилізації. Бібліотека підтримує сортування, фільтрацію, пагінацію, групування даних та інші функції, які можна легко налаштувати.

Завдяки своїй гнучкості, @tanstack/react-table підходить для побудови як простих, так і складних таблиць із великою кількістю даних. Вона добре інтегрується з іншими інструментами, такими як React Query, і забезпечує високий рівень продуктивності навіть у масштабних проєктах.

## 1.5. Інтеграція платежів

### 1.5.1. Stripe

**Stripe** — це популярна платформа для обробки онлайн-платежів, яка забезпечує інструменти для інтеграції платіжних систем у вебзастосунки. Розроблена у 2010 році, Stripe швидко стала одним із лідерів ринку завдяки своїй простоті використання, високій безпеці та підтримці широкого спектра платіжних методів, включаючи кредитні картки, електронні гаманці та банківські перекази. Stripe пропонує API, який дозволяє розробникам інтегрувати платіжний функціонал без необхідності глибоких знань у фінансових системах.

Однією з головних переваг Stripe є її підтримка глобального ринку. Платформа працює більш ніж у 40 країнах і підтримує понад 135 валют, що дозволяє легко обробляти міжнародні платежі. Крім того, Stripe забезпечує сучасний інтерфейс для відстеження транзакцій, управління підписками, створення рахунків-фактур і навіть запровадження динамічного ціноутворення. Усе це робить її ідеальним вибором як для стартапів, так і для великих компаній.

Stripe також приділяє велику увагу безпеці, застосовуючи інструменти для виявлення шахрайства та дотримуючись стандартів PCI DSS (Payment Card Industry Data Security Standard). Завдяки функціям автоматичної верифікації платежів і підтримці 3D Secure платформа забезпечує безпечний обмін фінансовими даними між користувачами й компаніями. Простий API Stripe у

поєднанні з гнучкими можливостями налаштування робить цю платформу одним із найбільш надійних і зручних рішень для інтеграції платіжних систем у вебзастосунки.

### 1.6. Додаткові інструменти

#### 1.6.1. Vercel

**Vercel** — це хмарна платформа для розгортання вебзастосунків, спеціалізована на проєктах, побудованих на фреймворках JavaScript, таких як Next.js. Заснована у 2015 році, Vercel здобула популярність завдяки простоті використання, швидкому деплою та оптимізованій інфраструктурі для сучасних вебдодатків. Платформа автоматично обробляє складні серверні процеси, такі як масштабування, CDN-оптимізація та кешування, що дозволяє розробникам зосередитися на створенні функціональності проєкту.

Однією з ключових особливостей Vercel є її глибока інтеграція з **Next.js**, фреймворком, який також розроблений цією компанією. Це дозволяє Vercel повною мірою підтримувати функції Next.js, включаючи серверний рендеринг (SSR), статичну генерацію сторінок (SSG) і API-роути. Процес деплою максимально спрощений: Vercel синхронізується з GitHub, GitLab або Bitbucket, автоматично виявляє зміни у коді та запускає процес збірки й публікації, що робить кожен деплой швидким і безболісним.

Ще однією важливою перевагою є вбудована система попереднього перегляду. Під час деплою кожна гілка в репозиторії отримує унікальний URL для попереднього тестування, що забезпечує зручність у командній роботі. Крім того, Vercel автоматично оптимізує ресурси через CDN, забезпечуючи високу продуктивність і швидкість завантаження сторінок для користувачів по всьому світу. Це робить Vercel ідеальним вибором для розгортання вебзастосунків, створених на Next.js, завдяки його швидкості, простоті та безшовній інтеграції.

### 1.6.2. Sentry

**Sentry** — це платформа для моніторингу помилок і продуктивності, яка допомагає розробникам виявляти, аналізувати та усувати проблеми у застосунках. Вона підтримує широкий спектр технологій, включаючи вебзастосунки, мобільні додатки та серверні системи. Sentry надає інструменти для збору детальної інформації про помилки, таких як стек викликів, дані про користувачів, середовище виконання та багато іншого. Це дозволяє розробникам оперативно реагувати на проблеми та мінімізувати їх вплив на кінцевих користувачів.

Одна з найбільших переваг Sentry — це можливість відстежувати помилки в режимі реального часу. Платформа автоматично фіксує виключення та логічні помилки у вашому застосунку, надсилаючи повідомлення про інциденти через електронну пошту, Slack або інші інтегровані сервіси. Крім того, Sentry підтримує розумну групування помилок, що дозволяє уникнути дублювання й сфокусуватися на ключових проблемах. Завдяки динамічним дашбордам і звітам, розробники отримують повну картину про стан застосунку та можуть приймати ефективні рішення для його покращення.

Sentry має інтеграцію з **Next.js**, яка дозволяє легко налаштувати відстеження помилок і продуктивності. Вона забезпечує автоматичний збір інформації про помилки під час серверного рендерингу (SSR), клієнтського виконання та API-запитів. Це особливо важливо для проєктів, де велика частина функціональності виконується як на сервері, так і на клієнті. Простота налаштування та підтримка TypeScript роблять Sentry незамінним інструментом для забезпечення стабільності та якості сучасних вебзастосунків.

# Розділ 2. Програмна реалізація

## 2.1. Архітектура вебсервісу

### 2.1.1. Загальна структура

Архітектура вебсервісу базується на сучасному стеку технологій T3, який включає Next.js, tRPC, Prisma, NextAuth.js, Tailwind CSS та shadcn/ui. Цей підхід забезпечує ефективність розробки, високу продуктивність і гнучкість у розширенні функціоналу. Основним компонентом є Next.js, який одночасно виконує роль фронтенду та бекенду завдяки інтеграції з API-роутами. tRPC забезпечує типобезпечну комунікацію між клієнтом і сервером, зменшуючи ризик помилок у запитах та відповідях.

Архітектура побудована з урахуванням мікросервісного підходу, де основний бекенд відповідає за авторизацію, управління даними користувачів і підписками, а мікросервіс обробляє генерацію статей через API. Це дозволяє розділити логіку роботи вебсервісу, підвищуючи його масштабованість і спрощуючи підтримку.

### 2.1.2. Взаємодія між компонентами

Фронтенд реалізований на Next.js і надає зручний інтерфейс для користувачів. За допомогою tRPC фронтенд взаємодіє з бекендом для виконання завдань, таких як управління проєктами, статтями та підписками. Завдяки типобезпечності tRPC, обмін даними відбувається швидко та без помилок, а Tailwind CSS разом із shadcn/ui забезпечують сучасний і адаптивний дизайн інтерфейсу.

Бекенд реалізований через API-роути Next.js. Він обробляє запити від фронтенда, використовуючи Prisma ORM для взаємодії з базою даних PostgreSQL. Бекенд також виконує роль посередника для комунікації з

мікросервісом, відправляючи запити на генерацію статей та отримуючи результат у форматі JSON.

### 2.1.3. База даних

У вебсервісі використовується реляційна база даних PostgreSQL, яка забезпечує надійне зберігання даних. Prisma ORM спрощує роботу з базою, дозволяючи швидко виконувати міграції, працювати зі складними запитами та підтримувати типобезпечність у коді. Основні моделі бази даних включають `User` для зберігання інформації про користувачів, `Project` для управління проєктами, `Subscription` для обліку підписок, а також `Article` і `ArticleTask` для роботи зі статтями.

База даних є центральною точкою для обробки інформації. Всі компоненти вебсервісу, включаючи бекенд і мікросервіс, мають доступ до неї через Prisma. Це дозволяє підтримувати консистентність даних та уникати дублювання, а також забезпечує зручне масштабування у разі збільшення обсягу даних.

### 2.1.4. Комунікація з мікросервісом

Мікросервіс, який відповідає за генерацію контенту, реалізований як окремий компонент вебсервісу. Основний бекенд передає запити до мікросервісу через API, надсилаючи необхідні параметри, наприклад, структуру статті, ключові слова чи часові обмеження. Мікросервіс обробляє ці запити та повертає результат у вигляді згенерованого контенту.

Комунікація виконується у форматі JSON через HTTP-запити, що забезпечує простоту та універсальність інтеграції. Завдяки цьому підходу можна легко

модифікувати чи масштабувати мікросервіс, не впливаючи на основний функціонал вебсервісу.

### 2.1.5. Масштабованість

Архітектура вебсервісу побудована з урахуванням можливості масштабування. Поділ функціоналу між основним бекендом і мікросервісом дозволяє розподіляти навантаження, забезпечуючи стабільність роботи навіть при значному зростанні кількості користувачів чи даних. Використання PostgreSQL у поєднанні з Prisma дозволяє ефективно обробляти великі обсяги запитів та оновлень.

Розгортання на платформі Vercel автоматично забезпечує балансування навантаження та оптимізацію продуктивності. Завдяки цьому вебсервіс залишається швидким і стабільним незалежно від масштабів використання.

## 2.2. Реалізація функцій аутентифікації

### 2.2.1. Використання NextAuth.js

NextAuth.js було обрано як основну бібліотеку для реалізації аутентифікації завдяки її гнучкості та простоті інтеграції з Next.js. Бібліотека надає широкий набір функцій для роботи з OAuth-провайдерами, кастомними методами аутентифікації та управління сесіями. Основною перевагою NextAuth.js є її підтримка різних стратегій зберігання сесій, включаючи використання JWT, що робить її ідеальним вибором для сучасних вебзастосунків.

Використання NextAuth.js дозволяє легко інтегрувати кілька методів аутентифікації без необхідності написання великої кількості власного коду. Це

також забезпечує можливість розширення функціоналу завдяки підтримці callback-функцій, що дозволяють налаштовувати поведінку сесій, обробку токенів та взаємодію з базою даних.

### 2.2.2. Провайдери аутентифікації (Google, email + пароль)

Для OAuth-аутентифікації був інтегрований Google як провайдер. Цей метод дозволяє користувачам швидко входити до вебсервісу, використовуючи свої облікові записи Google, забезпечуючи зручність та безпеку. Інтеграція реалізована через Google Provider, де клієнтський ID та секрет отримуються з Google Cloud Console. У разі входу нового користувача його інформація автоматично додається до бази даних, а для існуючих оновлюються відповідні дані.

Кастомна аутентифікація реалізована через email і пароль за допомогою `CredentialsProvider`. Цей підхід дозволяє забезпечити користувачів можливістю традиційного входу. Для зберігання паролів застосовується хешування за допомогою сучасних алгоритмів, таких як bcrypt, що гарантує їх безпеку. У разі успішної автентифікації користувач отримує токен доступу, який додається до сесії.

### 2.2.3. Управління сесіями та JWT

NextAuth.js забезпечує зручний механізм управління сесіями через використання JWT. Сесії зберігаються у вигляді токенів, що дозволяє мінімізувати навантаження на сервер і уникати необхідності зберігання даних сесій у базі даних. JWT містить інформацію про користувача, таку як ID, email та токен доступу, який використовується для авторизації в подальших запитах.

Callback-функції NextAuth.js, такі як `session` і `jwt`, дозволяють додавати до токенів додаткові поля, наприклад, роль користувача чи токен доступу до сторонніх сервісів. Це робить сесії гнучкими та придатними для складних сценаріїв аутентифікації. Для забезпечення безпеки всі токени підписуються за допомогою секретного ключа (`NEXTAUTH_SECRET`), що гарантує їхню цілісність і захист від підробки.

### 2.3. Інтеграція з базою даних

#### 2.3.1. Використання Prisma ORM

Prisma ORM було обрано як основний інструмент для роботи з базою даних завдяки його потужності та зручності. Prisma забезпечує типобезпечну взаємодію між застосунком і базою даних, дозволяючи легко виконувати операції створення, читання, оновлення та видалення даних (CRUD). Крім того, Prisma автоматично генерує клієнтський код для роботи з моделями, що значно спрощує інтеграцію з TypeScript-застосунками.

Особливістю Prisma є система міграцій, яка дозволяє підтримувати синхронізацію структури бази даних із кодом проєкту. Це забезпечує просте додавання або зміну таблиць без ризику пошкодження існуючих даних. Використання PostgreSQL як бази даних дозволяє ефективно працювати з великими обсягами структурованої інформації, яку обробляє вебсервіс.

#### 2.3.2. Структура бази даних

Основна структура бази даних складається з кількох ключових моделей, таких як `User`, `Project`, `Subscription`, `Article` та `ArticleTask`.

- **Модель `User`** зберігає інформацію про користувачів, включаючи їхні облікові дані, статуси, історію сесій та пов’язані проєкти.
- **Модель `Project`** відображає інформацію про кожен проєкт користувача, включаючи параметри інтеграції з CMS та API.
- **Модель `Subscription`** дозволяє відстежувати активні підписки користувачів, обмеження за токенами та відповідні тарифи.
- **Моделі `Article` і `ArticleTask`** використовуються для планування, генерації та публікації статей, а також для відстеження їх статусів.

Всі моделі мають чітко визначені зв’язки між собою. Наприклад, модель `User` пов’язана з моделлю `Project` через поле `userId`, що забезпечує доступ користувача лише до своїх проєктів. Подібні зв’язки гарантують цілісність даних у базі.

### 2.3.3. Доступ до бази даних

Обидва бекенди мають доступ до однієї бази даних через Prisma. Основний бекенд забезпечує роботу з даними користувачів, управління проєктами та підписками. Мікросервіс, у свою чергу, працює з моделями статей, генерації та завдань. Завдяки чітко визначеній структурі бази й автоматичній генерації клієнтського коду Prisma, взаємодія між бекендами та базою є безпечною й ефективною.

Типові операції, що виконуються через Prisma, включають створення нових користувачів і проєктів, оновлення підписок, отримання списків статей та планування завдань на їхню публікацію. Завдяки типобезпечності Prisma помилки, пов’язані з некоректними запитами, мінімізуються, що забезпечує стабільність роботи всього вебсервісу.

## 2.4. Реалізація користувацького інтерфейсу

### 2.4.1. Використання Tailwind CSS і shadcn/ui

Для стилізації інтерфейсу було обрано комбінацію **Tailwind CSS** і компонентів **shadcn/ui**. Tailwind CSS забезпечує гнучкий утилітарний підхід до створення стилів, що дозволяє швидко адаптувати інтерфейс під потреби вебсервісу. Його інтеграція з компонентами shadcn/ui спрощує роботу над складними елементами інтерфейсу, такими як діалоги, кнопки чи випадаючі меню.

Компоненти shadcn/ui побудовані на основі Radix UI, що гарантує відповідність стандартам доступності (ARIA). Їхня готова стилізація за допомогою Tailwind CSS дозволяє не тільки швидко розробляти інтерактивні елементи, але й забезпечує легкість у кастомізації. Це сприяє створенню сучасного, адаптивного та функціонального дизайну вебсервісу.

### 2.4.2. Інтерактивні компоненти

Для підвищення зручності користувачів у вебсервісі інтегровані інструменти для роботи з drag-and-drop через бібліотеку **@dnd-kit**. Ця бібліотека дозволяє створювати інтуїтивно зрозумілий функціонал, наприклад, переміщення елементів, таких як ключові слова, між наборами. Її модульність та гнучкість дозволяють легко адаптувати функціонал під потреби вебсервісу.

Для анімацій інтерфейсу використовується **Framer Motion**, який забезпечує плавний перехід між станами елементів, покращуючи користувацький досвід. Наприклад, завдяки анімації при відкритті модальних вікон чи наведенні на інтерактивні елементи інтерфейс виглядає динамічніше та професійніше.

### 2.4.3. Таблиці та фільтри

Таблиці у вебсервісі реалізовані за допомогою **@tanstack/react-table**, яка є високопродуктивною бібліотекою для роботи з великими наборами даних. Таблиці підтримують базові функції, такі як сортування, фільтрація та пагінація, що робить їх зручними для користувачів.

Окрім цього, таблиці мають адаптивний дизайн і легко інтегруються з Tailwind CSS, забезпечуючи стильний вигляд навіть для складних наборів даних. Це рішення дозволяє ефективно працювати зі статтями, ключовими словами та іншими об'єктами вебсервісу.

## 2.5. Функціонал мікросервісу з генерації контенту

Мікросервіс у вебсервісі відповідає за складні обчислювальні задачі, пов’язані з генерацією контенту, плануванням публікацій та інтеграцією з платформами керування контентом (CMS). Весь функціонал мікросервісу реалізовано як автономний компонент, який працює незалежно від основного бекенду, взаємодіючи з ним через API.

Однією з ключових функцій мікросервісу є **генерація статей** за допомогою OpenAI API. Мікросервіс обробляє вхідні дані, такі як ключові слова та структури статей, і повертає готові тексти, зображення та інші елементи, необхідні для публікації.

Мікросервіс також виконує функцію **планування публікацій**. Завдяки інтеграції з Google Search Console, він забезпечує автоматичну індексацію створеного контенту, що сприяє оптимізації вебсайтів для пошукових систем.

Крім того, мікросервіс реалізує **інтеграцію з CMS**, такими як Webflow, Strapi та WordPress. Він автоматизує процес передачі контенту до вибраної платформи, забезпечуючи відповідність між полями CMS і створеними даними.

Завдяки гнучкому налаштуванню мапування, користувачі можуть адаптувати цей процес під свої потреби.

Цей автономний підхід дозволяє зменшити навантаження на основний бекенд та забезпечити гнучкість у розширенні функціоналу мікросервісу без впливу на роботу інших компонентів вебсервісу.

## 2.6. Обробка платежів

### 2.6.1. Інтеграція Stripe

Для обробки платежів і керування підписками у вебсервісі використовується платформа **Stripe**. Це потужне рішення, яке забезпечує зручну інтеграцію з вебзастосунками, підтримує різні способи оплати та відповідає найвищим стандартам безпеки. Stripe дозволяє автоматизувати весь процес від створення підписок до обробки транзакцій і оновлення статусу підписки в базі даних.

Інтеграція реалізована через Stripe API, що забезпечує гнучкість і можливість налаштування. Зокрема, Stripe дозволяє відстежувати різні події, такі як створення або видалення підписок, через **webhooks**, які автоматично надсилають відповідні запити на сервер вебсервісу. Ця автоматизація значно спрощує управління фінансовими операціями та знижує ризик помилок.

### 2.6.2. Підписки та тарифи

У вебсервісі доступні три основні тарифні плани: **Starter**, **Growth** та **Full-Service**, кожен з яких відповідає різним потребам користувачів.

- **Starter Plan** ($299 на місяць) включає 30 статей, автоматичну генерацію метаописів, інтеграцію з Google Search Console та можливість додаткових послуг, таких як фактчекінг і редагування.
- **Growth Plan** ($499 на місяць) пропонує все з Starter Plan, але дозволяє публікувати до 60 статей на місяць і включає підтримку SEO-менеджера.
- **Full-Service** ($700 на місяць) забезпечує комплексну підтримку з підбором ключових слів, керуванням кампаніями, а також повним набором додаткових послуг.

Окрім місячних планів, користувачі можуть обрати річні підписки, які забезпечують значну економію. Усі підписки пов'язані з доступом до токенів, які є внутрішнім ресурсом для генерації статей. Кількість токенів залежить від обраного плану.

### 2.6.3. Управління платежами

Stripe реалізує обробку платежів через **webhooks**, які повідомляють вебсервіс про стан підписок та транзакцій. Наприклад, подія `"customer.subscription.created"` автоматично створює запис у базі даних, пов'язаний із проєктом користувача, і встановлює кількість доступних токенів на основі тарифного плану. У разі завершення сесії оплати (`"checkout.session.completed"`), система оновлює поточний баланс токенів у відповідності до замовлення.

У разі невдалих платежів чи видалення підписки (`"customer.subscription.deleted"`) вебсервіс автоматично деактивує пов'язані послуги. Завдяки цьому користувачі отримують прозорий механізм доступу до функцій вебсервісу, а розробники можуть легко керувати станом підписок через централізовану обробку подій.

### 2.6.4. Логіка реалізації Webhooks

Webhooks Stripe є центральним механізмом для синхронізації стану підписок із базою даних вебсервісу. Кожна подія проходить через функцію-обробник, яка перевіряє тип події (`event.type`) і виконує відповідну дію. Наприклад:

- Для події `"customer.subscription.created"` створюється новий запис у моделі `Subscription`, в якому вказано план, дати початку й завершення, кількість токенів та зв'язок із проєктом.
- Подія `"checkout.session.completed"` дозволяє додавати токени до вже існуючої підписки, збільшуючи баланс користувача.
- У випадку `"customer.subscription.deleted"`, пов'язані записи видаляються з бази даних.

Завдяки цьому підходу система залишається прозорою для користувачів, а всі фінансові операції чітко синхронізуються з функціоналом вебсервісу.

### 2.6.5. Переваги інтеграції Stripe

Використання Stripe дозволяє вебсервісу забезпечити високий рівень безпеки та прозорості в обробці платежів. Усі транзакції шифруються й проходять валідацію через підписаний секрет (`STRIPE_WEBHOOK_SECRET`), що унеможливлює підробку запитів. Інтеграція Stripe також підтримує автоматичне масштабування, дозволяючи обробляти збільшені обсяги платежів без втрати продуктивності.

Stripe надає користувачам простий і зрозумілий інтерфейс для оплати, водночас забезпечуючи вебсервіс усім необхідним для управління підписками й

фінансовими операціями. Такий підхід забезпечує якісний користувацький досвід і надійну роботу платіжної системи.

## 2.7. Моніторинг і помилки

### 2.7.1. Інструмент для моніторингу помилок

Для моніторингу стану вебсервісу та відстеження помилок використовується базовий функціонал **Sentry SDK** для **Next.js**. Ця платформа дозволяє автоматично збирати інформацію про винятки, логічні помилки та проблеми продуктивності, що виникають як на клієнтській, так і на серверній стороні. Інтеграція забезпечує реєстрацію помилок у режимі реального часу, що дозволяє швидко реагувати на них.

Основна перевага Sentry полягає в його універсальності: платформа підтримує всі ключові компоненти вебсервісу, включаючи фронтенд (React) та бекенд (API-роути Next.js). Завдяки цьому можна відстежувати весь ланцюжок виконання запитів і оперативно визначати джерело проблем.

### 2.7.2. Відстеження в реальному часі

Інтеграція з Sentry забезпечує сповіщення про помилки в режимі реального часу. Усі події надсилаються до дашборду Sentry, де розробники отримують доступ до детального опису, включаючи стек викликів, середовище виконання та дії, які призвели до помилки.

Цей функціонал мінімізує час реакції на критичні проблеми. Наприклад, якщо помилка виникає у процесі генерації статті чи обробки запитів, Sentry відразу фіксує її й відправляє повідомлення через інтегровані сервіси, такі як Slack чи email.

### 2.7.3. Групування та аналіз помилок

Sentry автоматично групує схожі помилки, зменшуючи дублювання записів у дашборді. Це дозволяє розробникам зосередитися на основних проблемах, а не на їх окремих проявах. Кожна група містить аналітику щодо частоти виникнення помилок, що допомагає виявляти пріоритетні завдання для виправлення.

Окрім цього, інструмент дозволяє створювати звіти про стан продуктивності. Наприклад, якщо певний API-роут або компонент працює повільніше за очікуване, це можна легко визначити й оптимізувати.

### 2.7.4. Вплив на стабільність вебсервісу

Моніторинг через Sentry допомагає підтримувати стабільність роботи вебсервісу. Завдяки автоматичній реєстрації помилок та швидкому повідомленню розробників, ризик тривалого впливу помилок на користувачів мінімізується. Це особливо важливо для функціоналу, що стосується обробки платежів або генерації контенту, де будь-які проблеми можуть вплинути на бізнес-процеси.

Sentry забезпечує базовий рівень моніторингу, достатній для підтримки якісної роботи вебсервісу без значних витрат часу на налаштування або складну інтеграцію. Це робить його ефективним рішенням для сучасних застосунків, побудованих на Next.js.

## 2.8. Розгортання вебсервісу

### 2.8.1. Платформа для розгортання

Для розгортання вебсервісу використовується **Vercel**, платформа, що забезпечує швидке й зручне розгортання застосунків, розроблених на Next.js. Інтеграція з Vercel дозволяє автоматизувати процеси розгортання завдяки прямому зв’язку з репозиторієм GitHub. При кожному новому коміті у вибраній гілці розгортання оновлюється автоматично, що значно спрощує процес тестування і впровадження змін.

Vercel забезпечує високопродуктивну інфраструктуру, яка підтримує серверний рендеринг (SSR), статичні файли та API-роути. Це робить її ідеальним вибором для вебсервісів, побудованих на Next.js, адже платформа оптимізована саме для цього фреймворку.

### 2.8.2. Процес розгортання

Процес розгортання включає кілька основних етапів:

1. **Інтеграція з репозиторієм GitHub**: підключення до Vercel дозволяє вибрати репозиторій і гілку для автоматичного розгортання. Це забезпечує безперервну інтеграцію (CI/CD).
2. **Підготовка конфігурацій**: налаштування змінних середовища, таких як API-ключі (наприклад, Stripe, Sentry), у панелі керування Vercel.
3. **Автоматичне розгортання**: після кожного нового коміту у вибраній гілці Vercel автоматично запускає процес побудови застосунку, перевіряє його на наявність помилок і публікує на сервері.

### 2.8.3. Оптимізація продуктивності

Vercel забезпечує автоматичне масштабування серверів відповідно до трафіку, що дозволяє підтримувати стабільну роботу вебсервісу навіть за умов високого навантаження. Окрім цього, платформа автоматично оптимізує завантаження зображень і забезпечує кешування статичних ресурсів, що покращує швидкість завантаження сторінок.

Також платформа підтримує інструменти моніторингу та аналізу, такі як **Vercel Analytics**, що допомагають відстежувати продуктивність застосунку та виявляти можливі "вузькі місця". Завдяки цьому розробники можуть швидко реагувати на проблеми та вдосконалювати вебсервіс.

### 2.8.4. Переваги розгортання на Vercel

- **Автоматизація**: процес побудови, тестування та публікації застосунку автоматизовано, що зменшує кількість ручної роботи.
- **Стабільність і масштабованість**: платформа адаптується до навантажень, забезпечуючи безперервну роботу навіть при значному зростанні трафіку.
- **Простота інтеграції**: налаштування змінних середовища та API-ключів виконується у кілька кліків, що спрощує управління конфігураціями

# Розділ 3. Інструкція користувача

## 3.1. Реєстрація у вебсервісі

Користувач переходить на сторінку реєстрації

- Сторінка реєстрації доступна через головне меню або кнопку "Зареєструватися" на стартовій сторінці.

На сторінці реєстрації користувач заповнює форму:

- **Ім'я**: Вводить своє ім’я (обов’язкове поле).
- **Прізвище**: Вводить своє прізвище (обов’язкове поле).
- **Електронна пошта**: Вказує робочу електронну пошту (обов’язкове поле).
- **Пароль**: Придумує пароль, який відповідає вимогам безпеки (мінімум 8 символів).
- **Підтвердження пароля**: Повторює введений пароль для перевірки коректності.

Заповнена форма повинна виглядати приблизно як на зображенні (див. 3.1).

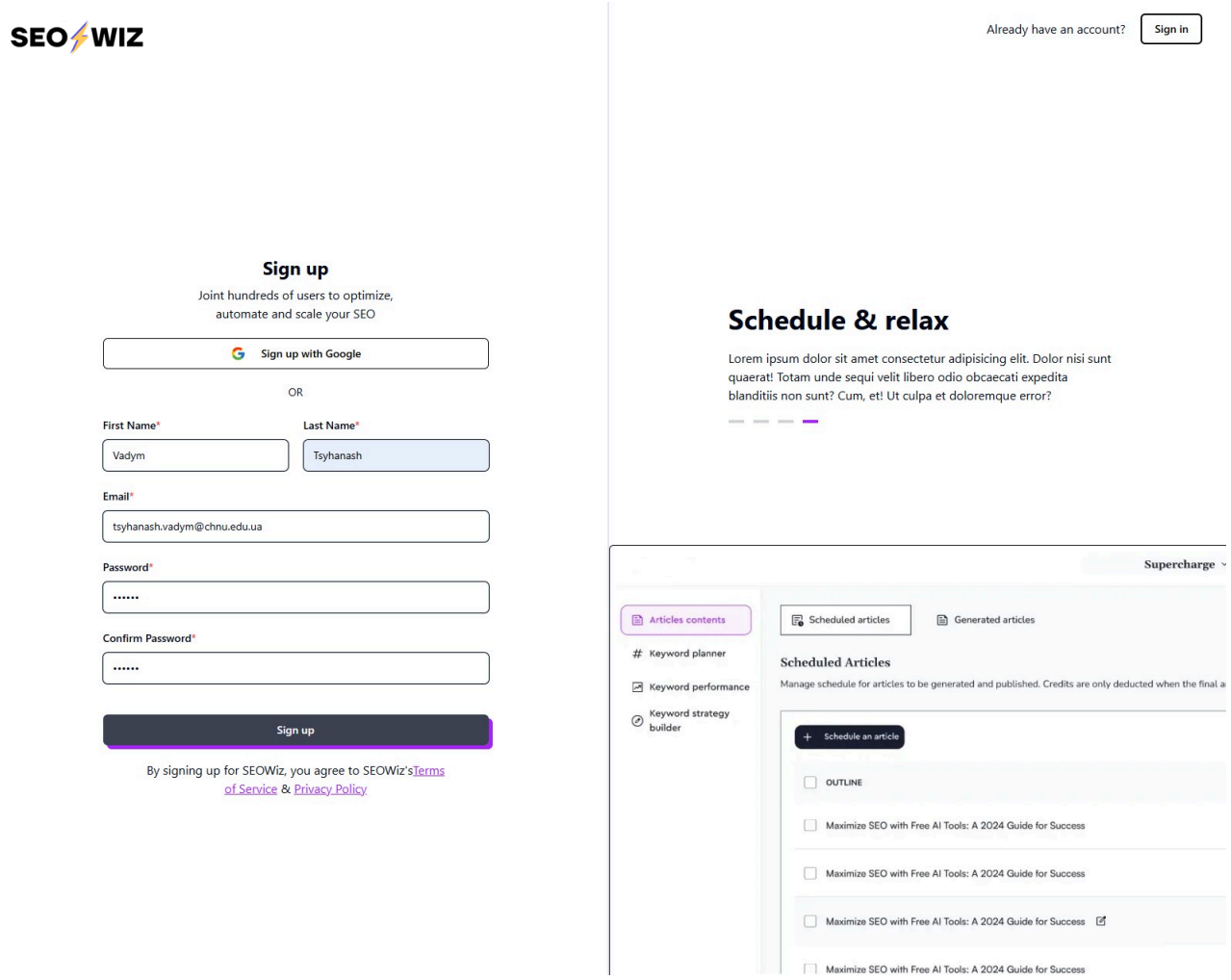


*Рис. 3.1 сторінка реєстрації*

Альтернативно користувач може скористатися входом через Google:

- Для цього натискає кнопку "Sign up with Google".
- Авторизується через обліковий запис Google і підтверджує дозвіл на використання свого профілю.

Після заповнення всіх обов’язкових полів користувач натискає кнопку "Зареєструватися".

- У разі некоректного заповнення форми з’являється повідомлення про помилку (наприклад, "Паролі не співпадають" або "Ця електронна пошта вже зареєстрована").

Після успішної реєстрації користувач автоматично перенаправляється на сторінку панелі керування, де він може почати створення нового проєкту.

### 3.2. Створення нового проєкту

Користувач переходить на панель керування (Dashboard), де відображається список його проєктів (див. 3.2).

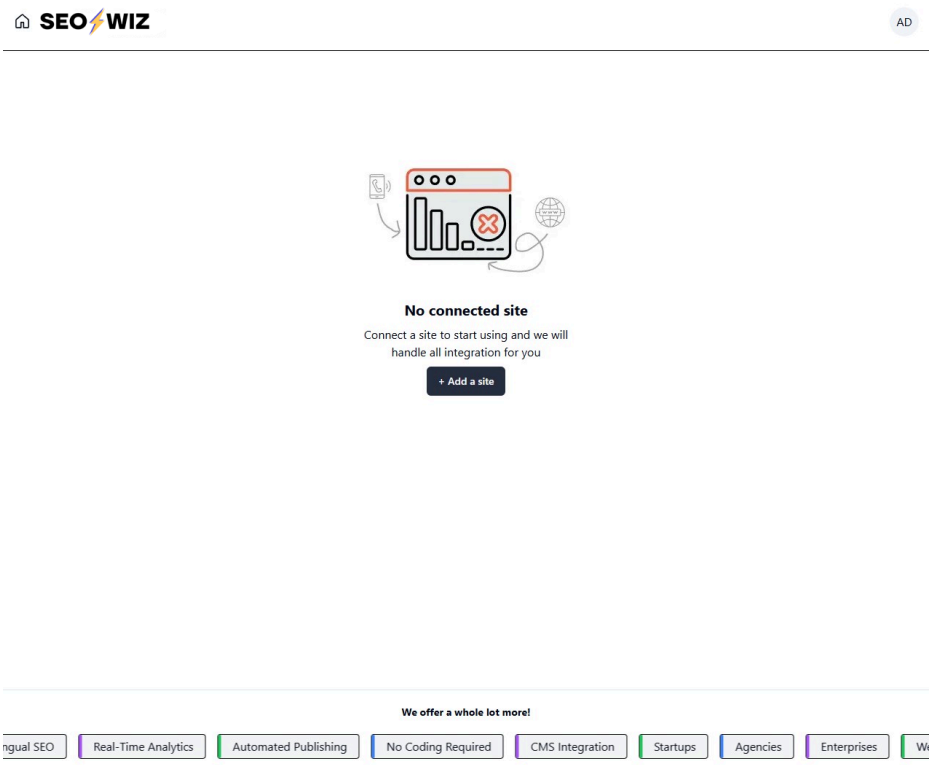


*Рис. 3.2 сторінка панелі керування*

- Якщо це перший вхід у вебсервіс, список буде порожнім.

На панелі керування користувач натискає кнопку **"Створити проєкт"**.

- Кнопка розташована у верхньому правому куті сторінки та виділена для зручності.

Користувач вибирає тип CMS із доступних варіантів (див. 3.3):

- **Webflow**, **Strapi**, або **WordPress**.
- Після вибору CMS користувач натискає кнопку **"Продовжити"**.

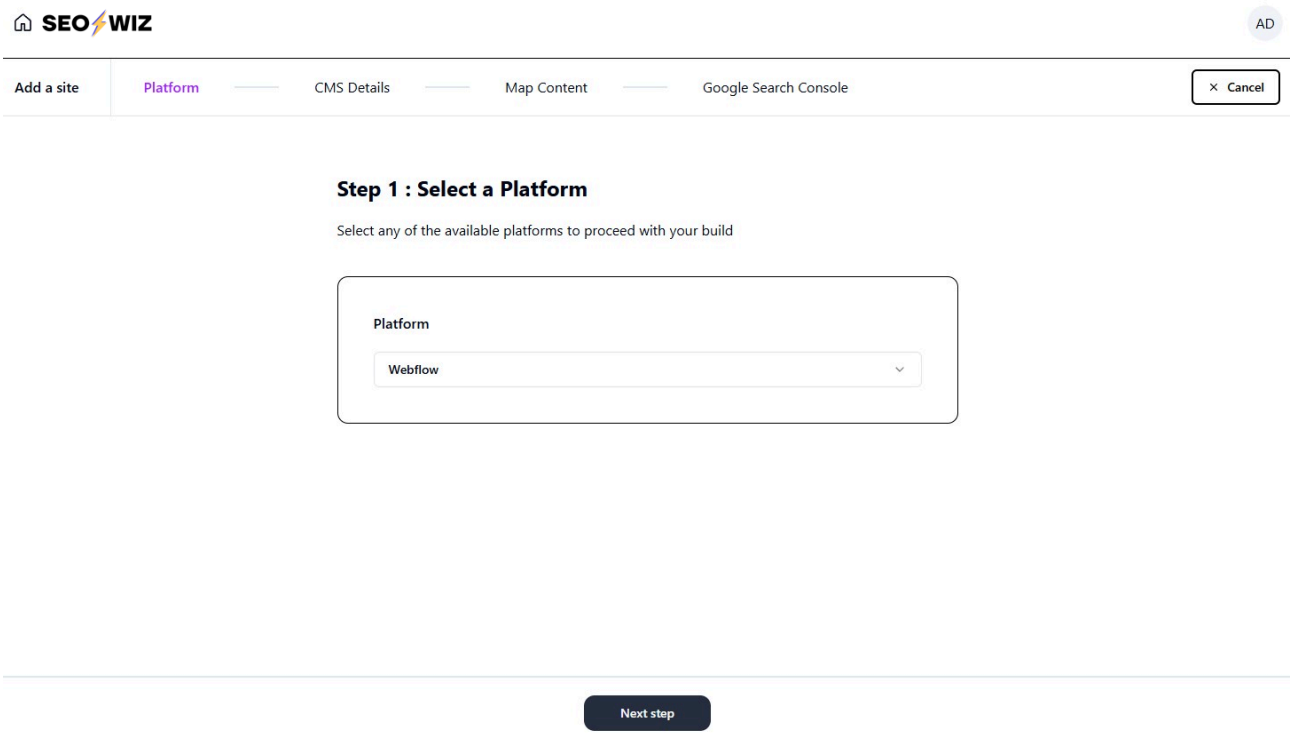


*Рис. 3.3 вибір платформи для створення нового проєкту*

На наступному кроці з’являється форма для введення параметрів проєкту (див. 3.4):

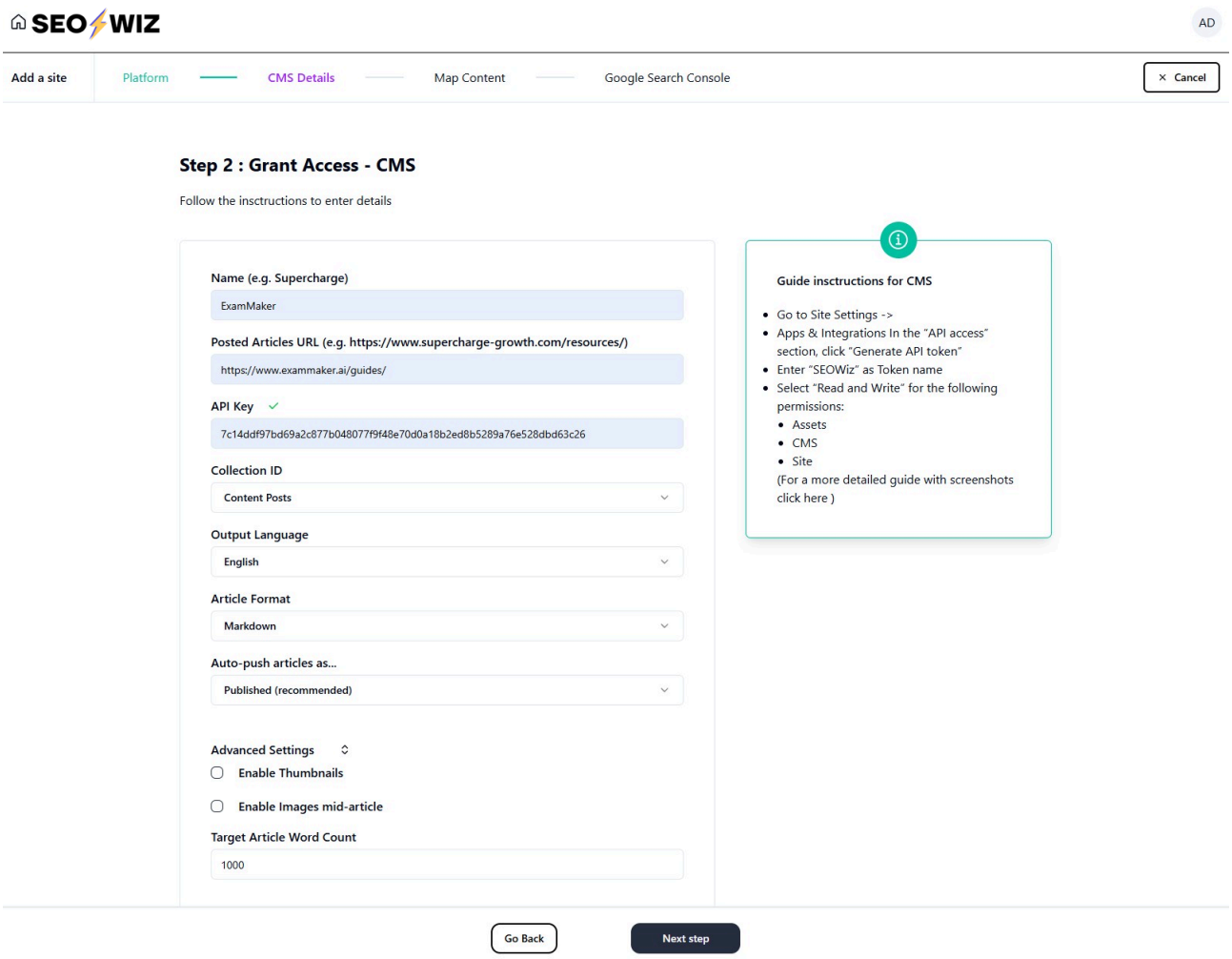


*Рис. 3.4 налаштування нового проєкту*

- **Назва проєкту**: Унікальна назва, яка допоможе ідентифікувати проєкт.
- **Посилання на статті**: URL, де будуть публікуватися статті (для налаштування автоматичного лінкування).
- **API-ключ**: Ключ для доступу до CMS (наприклад, для Webflow).
- **Колекція статей**: Назва або ID колекції в CMS, де будуть розміщуватися статті.
- **Мова статей**: Мова, на якій будуть генеруватися статті.
- **Формат статей**: Вибір між Markdown та Rich Text.
- **Автоматична публікація**: Опція для налаштування автоматичного розміщення статей як чернеток або опублікованих.

Після заповнення всіх полів користувач натискає кнопку **"Далі"**.

На наступному кроці налаштовується **мапування полів CMS** (див. 3.5):

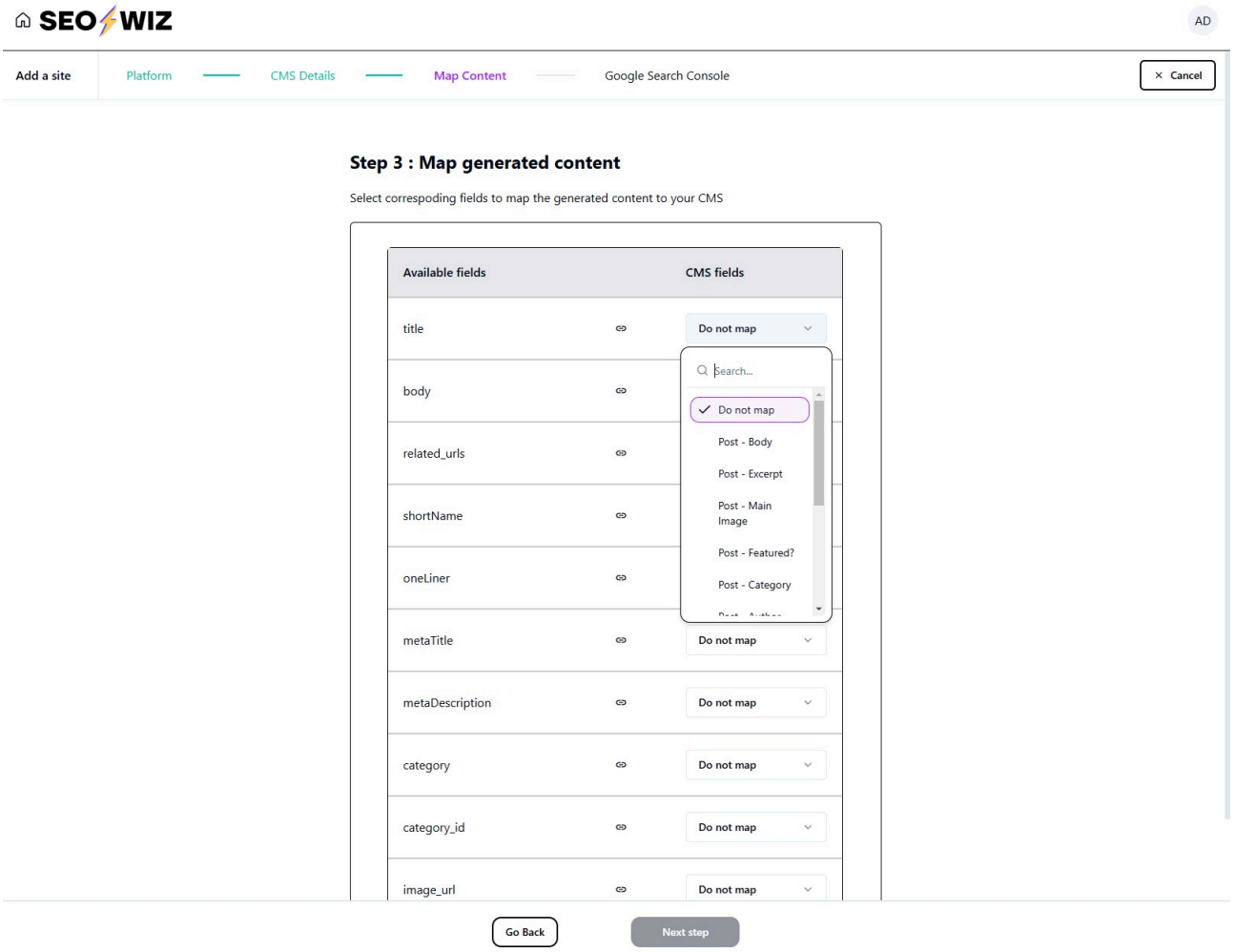


*Рис. 3.5 налаштування мапування полів для нового проєкту*

- Вебсервіс відображає список полів, які мають бути синхронізовані між вебсервісом та CMS (наприклад, "Title", "Body", "Meta Description").
- Користувач співвідносить кожне поле вебсервісу з відповідним полем у CMS за допомогою випадаючих меню.
- У разі потреби можна налаштувати значення за замовчуванням для деяких полів.

Останнім етапом є підтвердження створення проєкту:

- Користувач перевіряє всі введені дані та натискає кнопку **"Створити проєкт"**.
- Після успішного створення проєкт відображається у списку на панелі керування (див. 3.6), і користувач може перейти до його налаштування або планування статей.

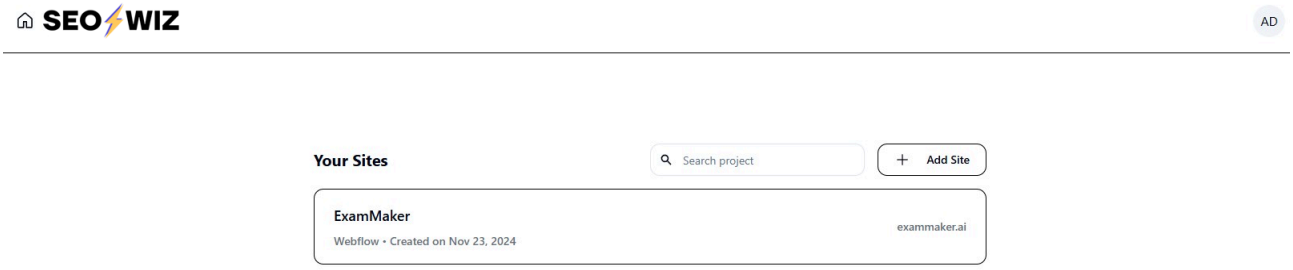


*Рис. 3.6 список проєктів на панелі керування*

### 3.3. Оформлення підписки

Користувач переходить до налаштувань обраного проєкту (див. 3.7):

- У списку проєктів на панелі керування користувач натискає на назву проєкту або кнопку **"Налаштування"**.

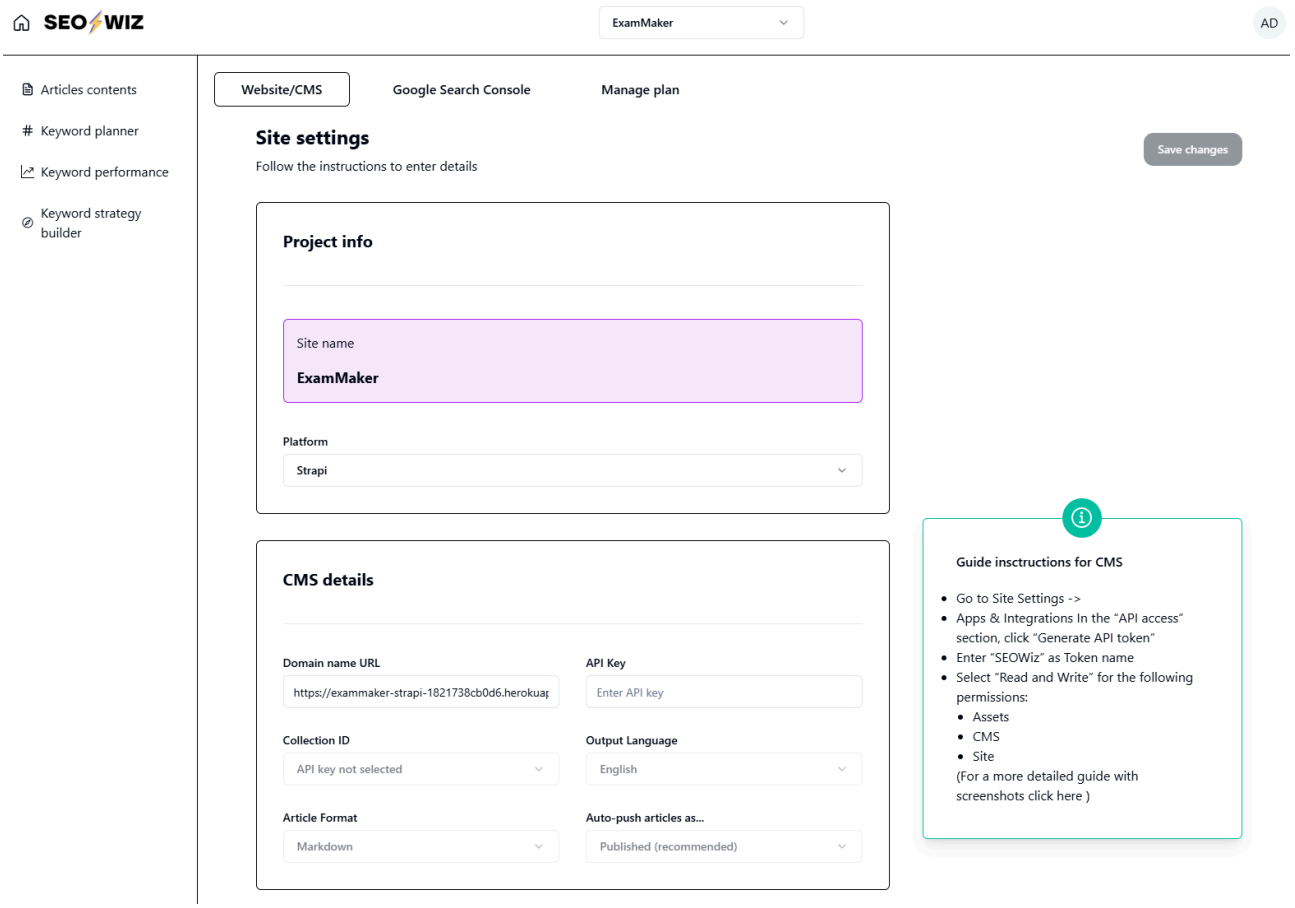


*Рис. 3.7 сторінка налаштувань проєкта*

У налаштуваннях проєкту відкривається вкладка **"Підписка"**:

- Тут користувач бачить опис доступних тарифних планів, таких як **Starter Plan**, **Growth Plan** і **Full-Service**, разом із інформацією про доступні функції та кількість статей, які можна генерувати за місяць.
- Якщо користувач уже має активну підписку, відображається її статус і термін дії  (див. 3.8).

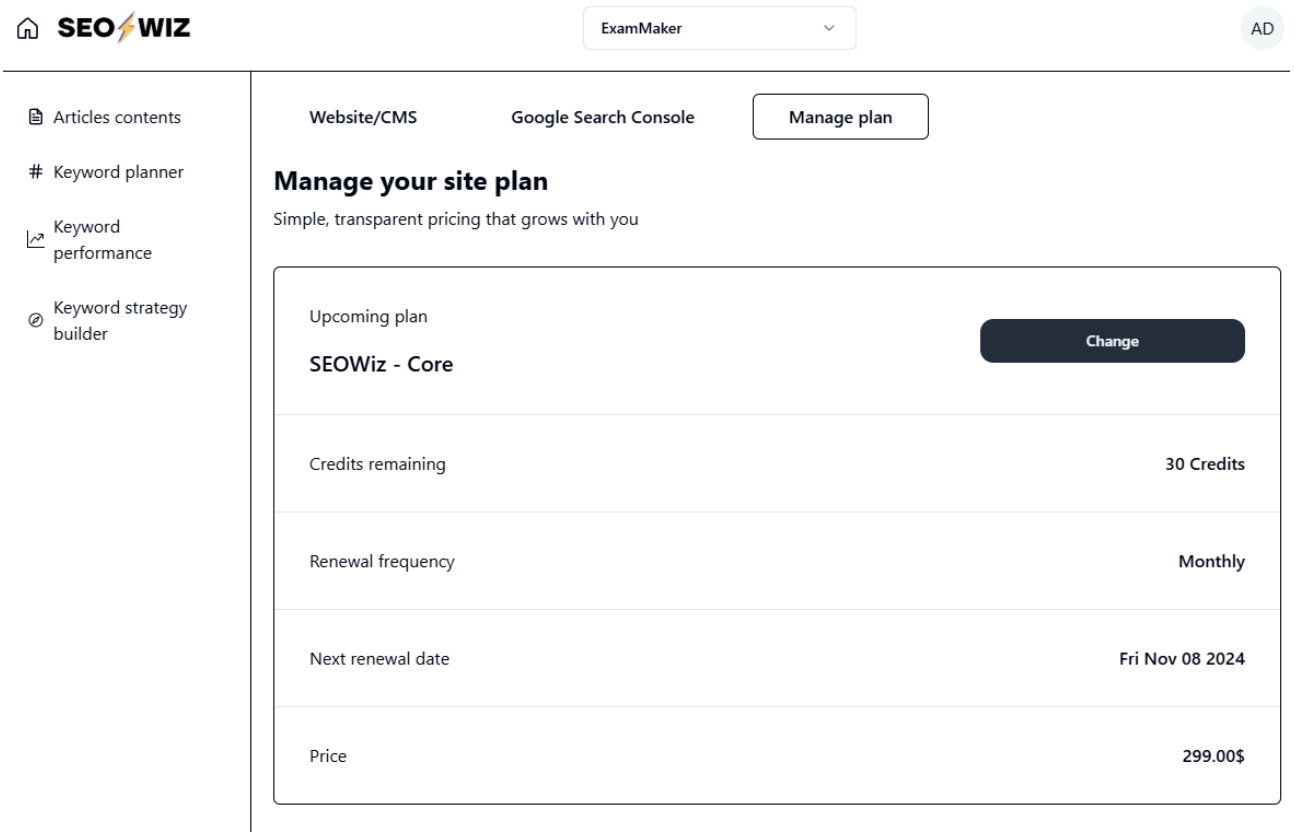


*Рис. 3.8 сторінка налаштування підписки проєкту*

Користувач обирає бажаний тарифний план (див. 3.9):

- Для цього він натискає кнопку **"Оформити підписку"** або **"Змінити план"** (для активних підписок).
- У разі потреби користувач може вибрати додаткові послуги, такі як фактчекінг, редагування або підбір зображень.

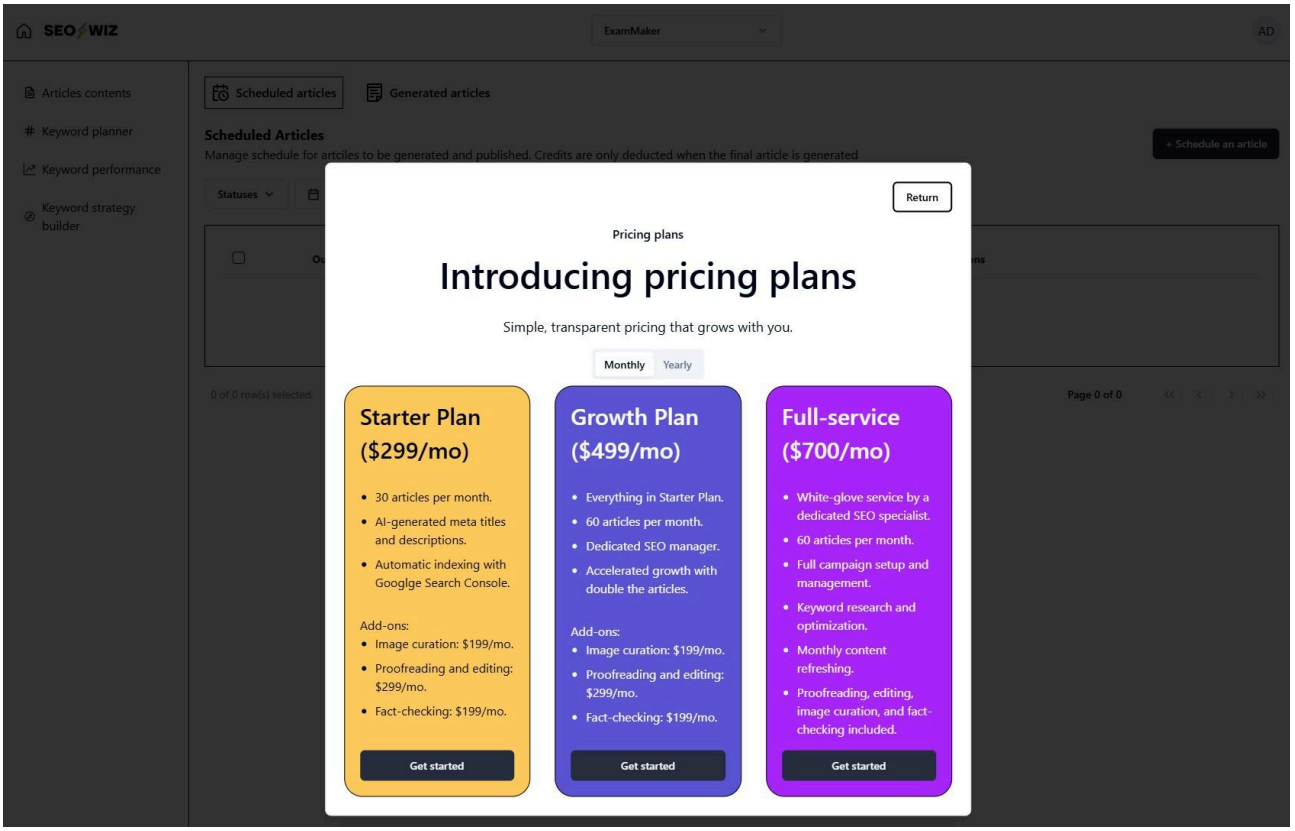


*Рис. 3.9 модальне діалогове вікно із тарифами*

Вебсервіс перенаправляє користувача на сторінку оплати Stripe (див. 3.10):

- На сторінці оплати користувач вводить свої платіжні дані, такі як номер кредитної картки, термін дії та CVV-код.
- У разі успішного введення даних користувач натискає кнопку **"Підтвердити оплату"**.

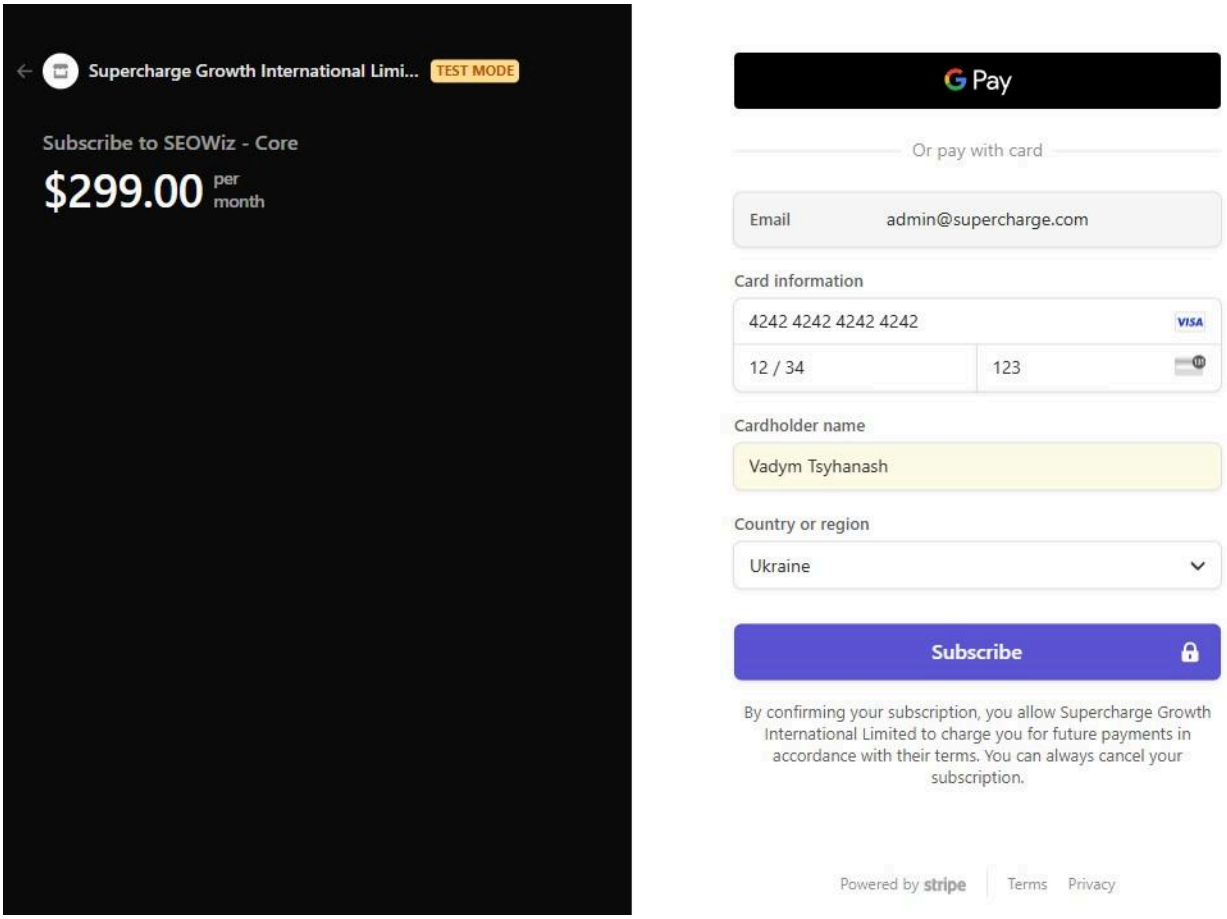


*Рис. 3.10 сторінка оплати на Stripe*

Після завершення оплати користувача перенаправляє назад до вебсервісу:

- На сторінці налаштувань відображається оновлена інформація про підписку, зокрема обраний тарифний план, кількість доступних токенів і дата завершення підписки.

У разі помилки оплати або скасування транзакції користувач залишається на сторінці вебсервісу та отримує відповідне повідомлення:

- Наприклад: **"Оплату не вдалося провести. Будь ласка, перевірте платіжні дані або спробуйте ще раз."**

### 3.4. Налаштування проєкту

Користувач переходить до сторінки налаштувань обраного проєкту:

На сторінці налаштувань доступні наступні опції:

**Основна інформація про проєкт**:

- Назва проєкту.
- Тип CMS (Webflow, Strapi, WordPress).
- Посилання на колекцію статей у CMS.
- Формат статей (Markdown або Rich Text).
- Мова статей.

Користувач може редагувати ці дані за потреби та натиснути **"Зберегти зміни"**, щоб оновити інформацію.

**Мапування полів**:

Відображаються всі поля, налаштовані під час створення проєкту, із можливістю їх зміни.

Користувач може додати нові поля або видалити існуючі.

**Підписка**:

Відображається інформація про активну підписку, включаючи тарифний план, кількість доступних токенів і дату закінчення підписки.

Користувач може змінити план або скасувати підписку, якщо це необхідно.

**Видалення проєкту**:

У нижній частині сторінки знаходиться кнопка **"Видалити проєкт"**.

Перед видаленням користувач отримує попередження, яке підтверджує, що вся пов’язана з проєктом інформація буде втрачена.

### 3.5. Планування статей

Користувач переходить на сторінку **"Планування статей"**:

- У меню проєкту, розташованому в панелі керування, вибирає вкладку **"Планування статей"**.
- Відкривається сторінка з інструментами для створення та планування контенту.

Додавання ключових слів (див. 3.11):

- Користувач може вручну ввести ключові слова або завантажити їх через CSV-файл.
- Для цього натискає кнопку **"Завантажити CSV"** та обирає файл із ключовими словами у відповідному форматі.
- Альтернативно, користувач може скористатися інструментом генерації пропозицій для ключових слів, що відображає рекомендовані варіанти для вибраного проєкту.

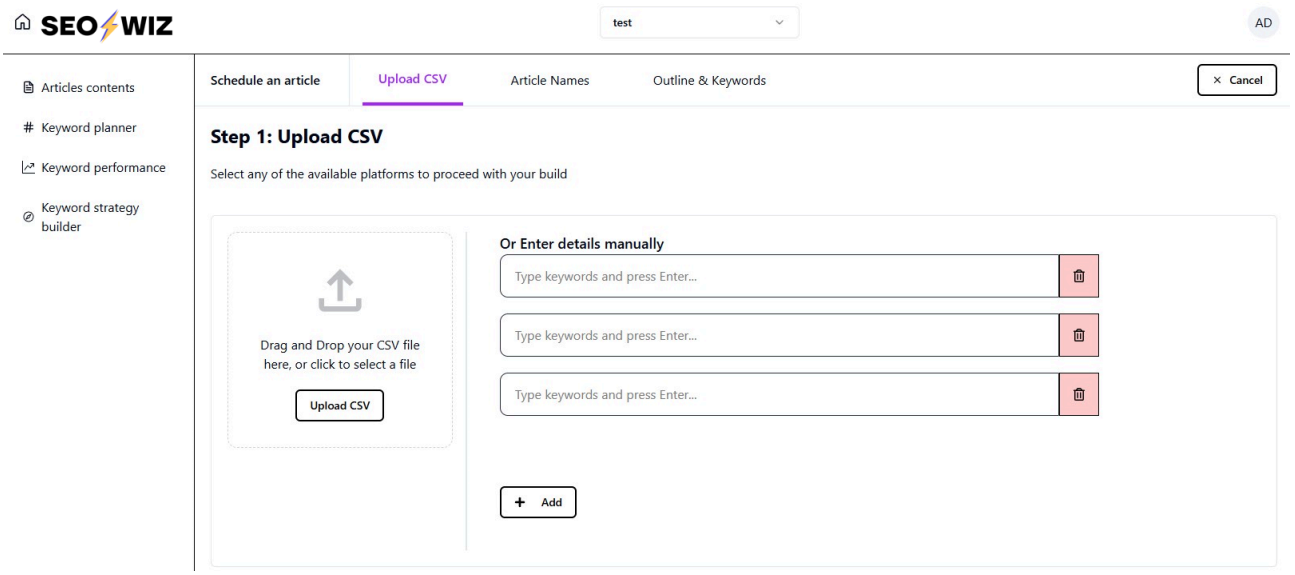


*Рис. 3.11 додавання ключових слів*

Після додавання ключових слів вебсервіс пропонує **назви статей**, які користувач може переглянути та відредагувати (див. 3.12):

- У таблиці відображаються автоматично згенеровані назви, кожну з яких можна редагувати, натиснувши на відповідний рядок.
- За потреби можна видалити назву або створити нову, натиснувши кнопку **"Додати назву"**.

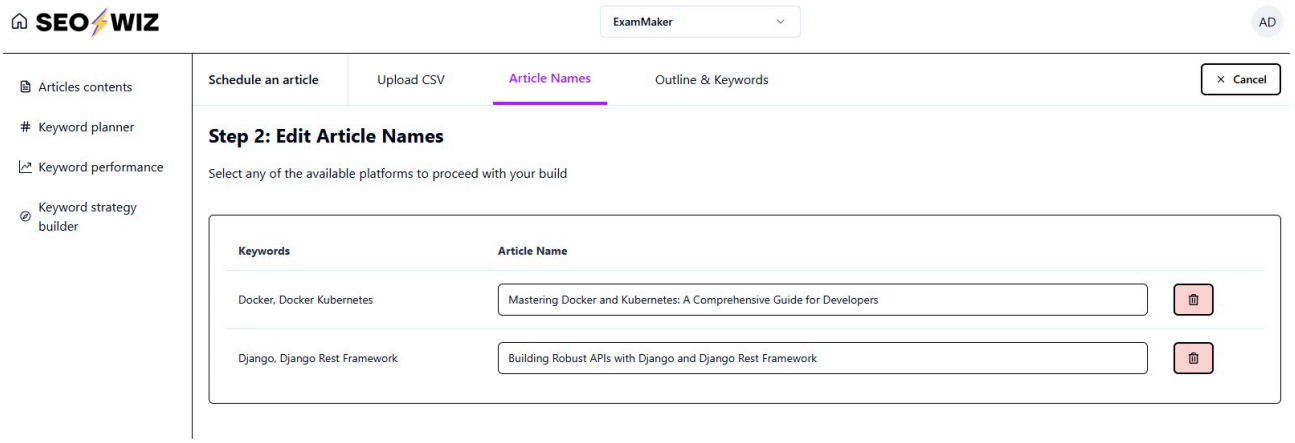


*Рис. 3.12 пропоновані назви статей*

Формування структури статей (див. 3.13):

- Для кожної статті вебсервіс автоматично генерує нариси, що включають основні заголовки, ключові слова та пропозиції щодо зображень.
- Користувач може переглянути запропоновану структуру, відредагувати заголовки або додати нові елементи, використовуючи простий інтерфейс.

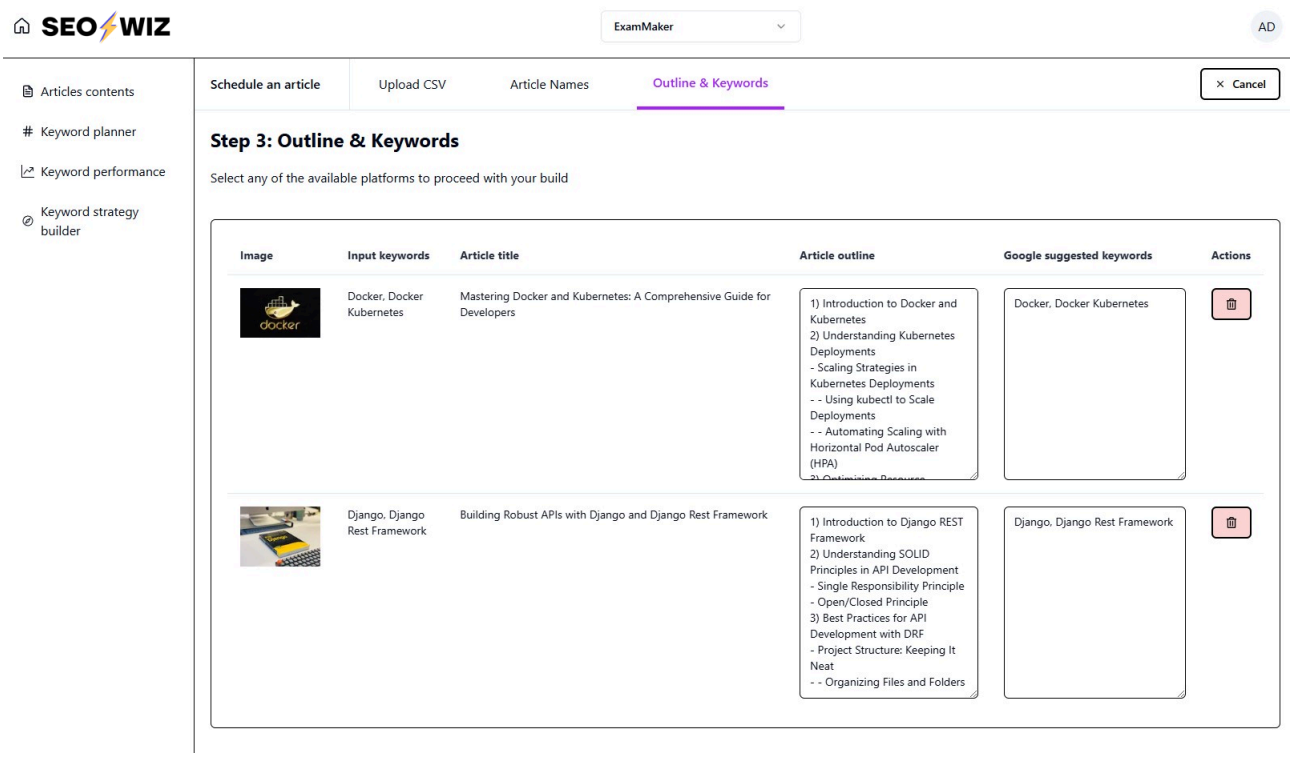


*Рис. 3.13 пропонована структура статей*

Планування публікації статей (див. 3.14):

- Користувач обирає бажану дату та час для публікації кожної статті або залишає її в статусі **"Чернетка"**.
- Для планування публікації натискає кнопку **"Запланувати"**.
- У разі потреби є можливість налаштувати періодичність публікацій, наприклад, 2 статті на тиждень.

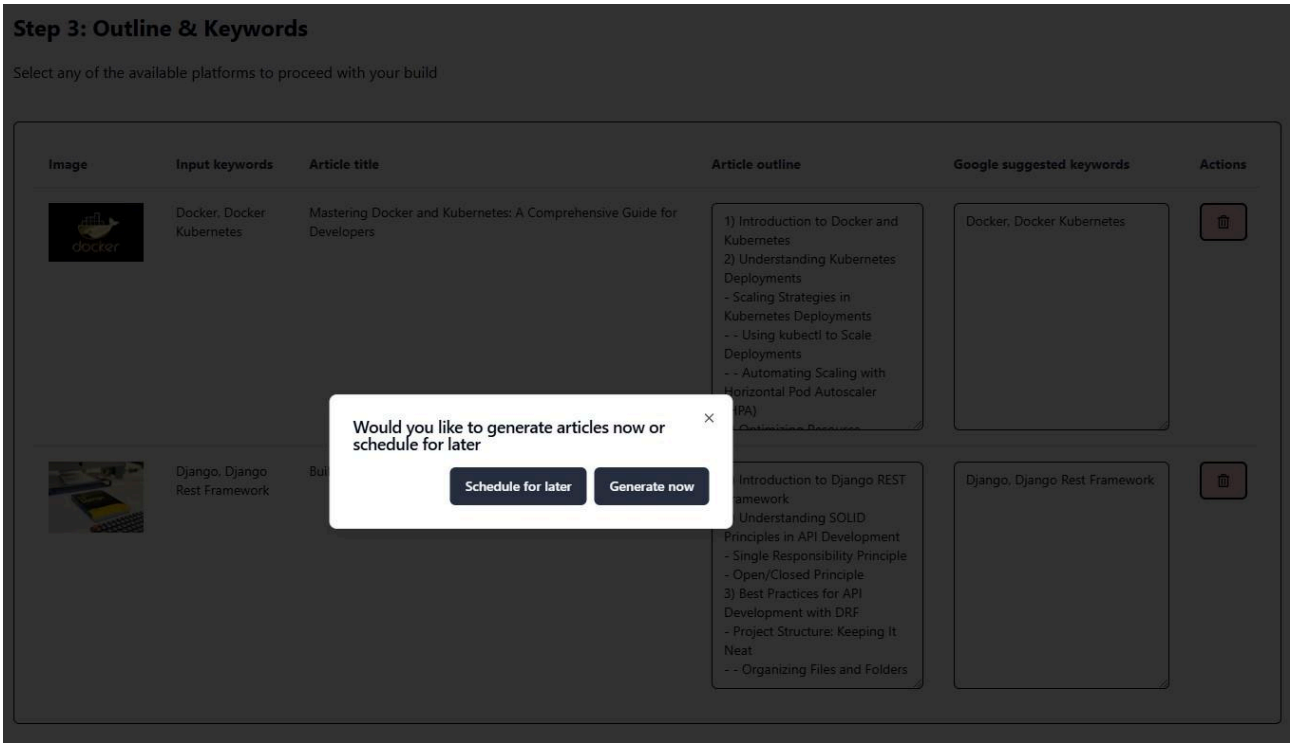


*Рис. 3.14 модальне діалогове вікно з вибором часу публікації*

Завершення планування (див. 3.15):

- Після завершення процесу всі статті з’являються у списку запланованих, де відображається їхній статус (заплановано, чернетка, опубліковано).
- За необхідності користувач може повернутися до планування, щоб внести зміни або видалити заплановані статті.

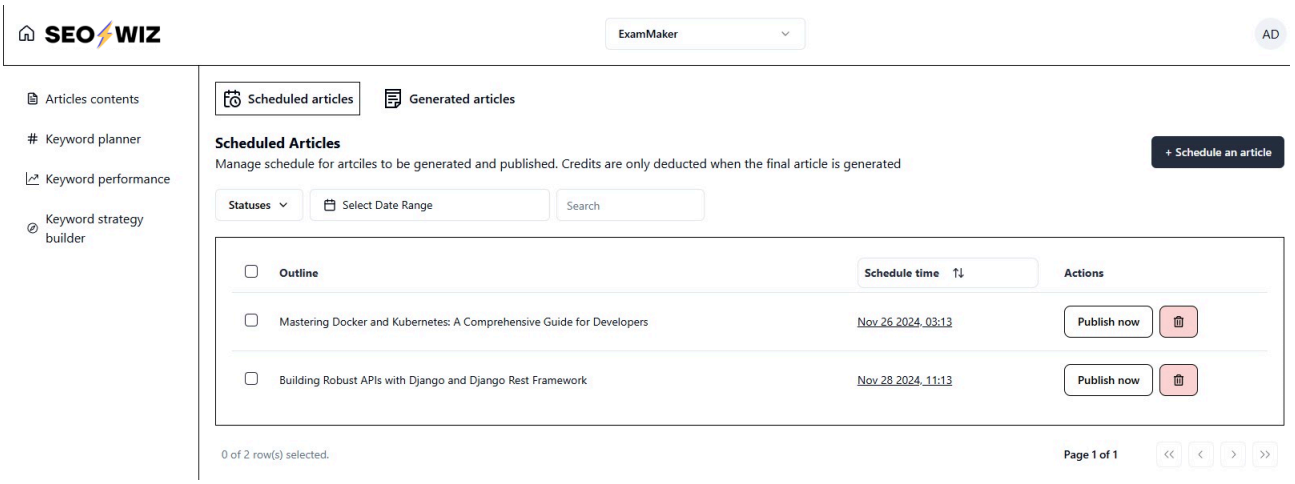


*Рис. 3.15 сторінка із запланованими статтями*

# Розділ 4. Результати покращення SEO

## Аналіз результатів

Завдяки використанню вебсервісу для автоматизації створення та оптимізації контенту, було досягнуто значного покращення видимості сайту в пошукових системах. На знімку з **Google Search Console** (див. 4.1) можна побачити зростання ключових показників SEO:

- **Total Clicks (Загальна кількість кліків)**: кількість переходів на сайт з пошукових систем.
- **Total Impressions (Загальна кількість показів)**: кількість разів, коли сторінка сайту з'явилася у результатах пошуку.

На зображенні (див. 4.1) чітко видно позитивну динаміку зростання: кількість кліків і показів значно збільшилась у порівнянні з початковими значеннями.

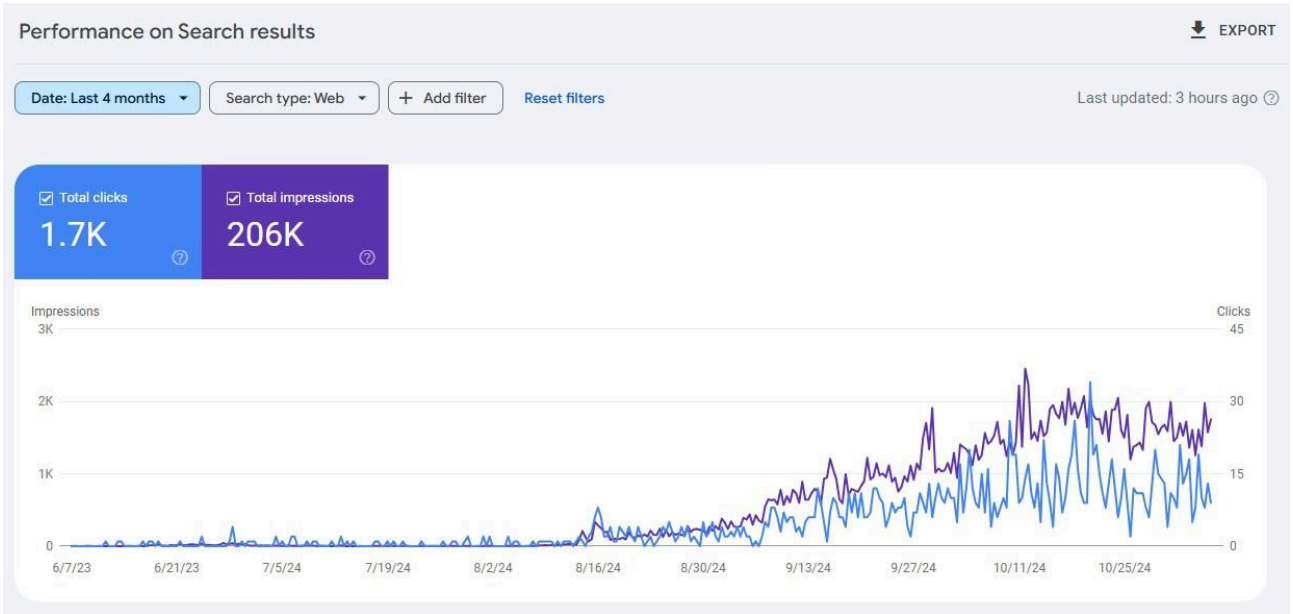


*Рис. 4.1 показники продуктивності в Google Search Console*

## Що таке Clicks і Impressions?

- **Clicks (Кліки):**

  Цей показник вказує на кількість разів, коли користувачі натискали на посилання, яке веде на ваш сайт, у результатах пошуку Google. Збільшення цього показника свідчить про те, що ваш контент відповідає потребам користувачів і має привабливі заголовки та опис.
- **Impressions (Покази):**

  Це кількість разів, коли сторінка вашого сайту з'являлась у результатах пошуку. Навіть якщо користувач не натискає на ваше посилання, наявність великої кількості показів означає, що ваш сайт ранжується за релевантними ключовими словами.

**Як вебсервіс сприяє зростанню SEO?**

1. **Автоматична оптимізація статей**:

   Генерація статей із правильно підібраними ключовими словами та метаописами підвищує релевантність контенту для пошукових систем.
2. **Міжстатейна лінковка (Internal Linking):**

   Автоматичне створення посилань між статтями підвищує рейтинг сторінок у межах сайту.
3. **Швидка індексація:**

   Завдяки інтеграції з Google Search Console новий контент індексується швидше, що дозволяє залучати органічний трафік у найкоротші терміни.
4. **Оптимізація ключових слів:**

   Використання Google Ads API для генерації релевантних ключових слів дозволяє краще націлювати контент на запити користувачів.

## Висновок

Отже, розроблений вебсервіс забезпечує користувачам зручний та ефективний інструмент для автоматизації створення, оптимізації та публікації статей, що дозволяє значно покращити видимість їхніх вебсайтів у пошукових системах. Завдяки інтуїтивно зрозумілому інтерфейсу та широкому функціоналу, користувачі можуть створювати нові проєкти, інтегрувати їх із популярними CMS, планувати статті та керувати своїми підписками з мінімальними витратами часу та зусиль.

Інтеграція зі Stripe забезпечує прозору обробку платежів і керування підписками, тоді як система моніторингу помилок на основі Sentry гарантує стабільність роботи сервісу. Розгортання на платформі Vercel дозволяє автоматизувати процеси CI/CD, забезпечуючи надійність та масштабованість вебсервісу навіть при зростанні кількості користувачів.

Для мене ця робота стала важливим етапом професійного розвитку. Вона дозволила мені реалізувати сучасні підходи до проєктування вебзастосунків, удосконалити знання у сфері фронтенд- і бекенд-розробки, баз даних та інтеграції сторонніх сервісів. Розв'язання реальних задач допомогло мені зрозуміти значущість своєї роботи як розробника, а також підвищити свої навички організації, планування та вирішення технічних проблем.

Ця розробка дала мені можливість зосередитися на ключових аспектах своєї професійної діяльності, навчила ефективно застосовувати здобуті знання та використовувати досвід у майбутніх проєктах. У результаті я отримав не лише практичний досвід розробки складних систем, а й усвідомлення важливості створення програмних продуктів, які вирішують реальні проблеми користувачів.

## Список використаної літератури


1. Next.js Documentation – [Електронний ресурс]. Режим доступу : https://nextjs.org/docs
2. TypeScript Documentation – [Електронний ресурс]. Режим доступу : https://www.typescriptlang.org/docs/
3. tRPC Documentation – [Електронний ресурс]. Режим доступу : https://trpc.io/docs/
4. Prisma Documentation – [Електронний ресурс]. Режим доступу : https://www.prisma.io/docs
5. NextAuth.js Documentation – [Електронний ресурс]. Режим доступу : https://next-auth.js.org/getting-started/introduction
6. Tailwind CSS Documentation – [Електронний ресурс]. Режим доступу : https://tailwindcss.com/docs/
7. Radix UI Documentation – [Електронний ресурс]. Режим доступу : https://www.radix-ui.com/primitives/docs/overview/introduction
8. shadcn/ui Documentation – [Електронний ресурс]. Режим доступу : https://ui.shadcn.com/docs
9. @dnd-kit Documentation – [Електронний ресурс]. Режим доступу : https://docs.dndkit.com/
10. Framer Motion Documentation – [Електронний ресурс]. Режим доступу : https://motion.dev/docs
11. React Hook Form Documentation – [Електронний ресурс]. Режим доступу : https://react-hook-form.com/get-started
12. Zod Documentation – [Електронний ресурс]. Режим доступу : https://zod.dev/
13. Zustand Documentation – [Електронний ресурс]. Режим доступу : https://zustand.docs.pmnd.rs/getting-started/introduction
14. @tanstack/react-query Documentation – [Електронний ресурс]. Режим доступу : https://tanstack.com/query/v5/docs/framework/react/overview

15. Stripe Documentation – [Електронний ресурс]. Режим доступу : https://docs.stripe.com/
16. Sentry Documentation – [Електронний ресурс]. Режим доступу : https://docs.sentry.io/
17. Vercel Documentation – [Електронний ресурс]. Режим доступу : https://vercel.com/docs

## Додатки

dashboard/page.tsx

```
import { api } from "~/trpc/server";
import { projectService } from "~/services/projectService";
import { TopBar } from "~/components/top-bar";
import { NoProjects } from "./_components/no-projects";
import { Projects } from "./_components/projects";
import { Separator } from "~/components/ui/separator";
import Marquee from "react-fast-marquee";
import { cn } from "~/lib/utils";


type MarqueeItemProps = React.HtmlHTMLAttributes<"div"> & {
  label: string;
  color: "green" | "blue" | "purple";
};
function MarqueeItem({ label, color, className }: MarqueeItemProps) {
  return (
    <div
      className={cn(
        "flex rounded-sm border border-black bg-gray-100",
        className,
      )}
    >
      <div
        className={cn(
          "rounded-sm border-l-4 bg-gray-100",
          color === "green" && "border-green-500",
          color === "blue" && "border-blue-500",
```

```
            color === "purple" && "border-purple-500",
          )}
        />
        <div className="px-4 py-1">{label}</div>
      </div>
    );
}

const marqueeItems = [
  "Webflow",
  "Strapi",
  "Wordpress",
  "SEO Audits",
  "Multilingual SEO",
  "Real-Time Analytics",
  "Automated Publishing",
  "No Coding Required",
  "CMS Integration",
  "Startups",
  "Agencies",
  "Enterprises",
];

export default async function Page() {
  const user = await api.user.get();
  const projects = await projectService.getProjects(user.id);

  const hasProjects = projects && projects.length > 0;
```

```
const getMarqueeColor = (index: number) => {
  if (index % 3 === 0) return "green";
  if (index % 3 === 1) return "blue";
  return "purple";
};


return (
  <>
    <TopBar showProjectSelector={false} />
    <div className="flex h-[calc(100vh-80px)] flex-col justify-between">
      {hasProjects ? <Projects projects={projects} /> : <NoProjects />}
      <div className="my-5 flex flex-col gap-5">
        <Separator />
        <div className="flex justify-center">
          <div className="text-sm font-bold">
            SEOWiz offers a whole lot more!
          </div>
        </div>
        <Marquee autoFill speed={35}>
          {marqueeItems.map((item, index) => (
            <MarqueeItem
              className="mx-2"
              key={index}
              label={item}
              color={getMarqueeColor(index)}
            />
          ))}
```

```
          </Marquee>
        </div>
      </div>
    </>
  );
}
```

projects/create/page.tsx

```
"use client";

import { TopBar } from "~/components/top-bar";

import { ProjectCreationStep as Step } from "~/types/components/steps";

import { SelectProjectTypeStep } from
"./_components/steps/select-project-type-step";
import { ConfigProjectDetailsStep } from
"./_components/steps/config-project-details-step";
import { MapFieldsStep } from "./_components/steps/map-fields-step";
import { useEffect } from "react";
import { StepsDisplay } from "./_components/steps-display";
import { Separator } from "~/components/ui/separator";
import { StepsControl } from "./_components/steps-control";
import { useStepsControl } from "./_hooks/use-steps-control";
import { ScrollArea } from "~/components/ui/scroll-area";
import { GoogleSearchConsoleStep } from
"./_components/steps/google-search-console-step";

const stepComponentMap = {
```

```
  [Step.SelectProjectType]: SelectProjectTypeStep,
  [Step.ConfigProjectDetails]: ConfigProjectDetailsStep,
  [Step.MapFields]: MapFieldsStep,
  [Step.GoogleSearchConsole]: GoogleSearchConsoleStep,
};

type StepComponentProps = React.HTMLAttributes<"div"> & {
  step: Step;
};
function StepComponent({ step, ...props }: StepComponentProps) {
  const Component = stepComponentMap[step];
  if (!Component) {
    return null;
  }

  return <Component {...props} />;
}

export default function Page() {
  const { currentStep, resetStepsStore } = useStepsControl();

  // Reset the store when the component is unmounted
  useEffect(() => {
    return () => {
      resetStepsStore();
    };
  }, []);
```

```
  return (
    <>
      <TopBar showProjectSelector={false} />
      <div className="relative h-[calc(100vh-80px)]">
        <ScrollArea className="h-full">
          <StepsDisplay currentStep={currentStep} />
          <Separator />
          <StepComponent step={currentStep} className="pb-20" />
          <StepsControl className="absolute bottom-0 z-10 w-full border-t-2
bg-white" />
        </ScrollArea>
      </div>
    </>
  );
}
```

articles/page.tsx

```
"use client";

import { NavBar } from "~/components/new-nav-bar";
import { TopBar } from "~/components/top-bar";
import { ScheduledArticles } from "./_components/scheduled-articles";
import { GrNotes as Notes } from "react-icons/gr";
import { LuCalendarClock as CalendarClock } from "react-icons/lu";
import { useState } from "react";
import { GeneratedArticles } from "./_components/generated-articles";
import { cn } from "~/lib/utils";
```

```
enum SelectedArticlesTable {
  Scheduled = "scheduled",
  Generated = "generated",
}

type ArticlesTableProps = React.HtmlHTMLAttributes<"div"> & {
  selectedTable: SelectedArticlesTable;
};
const ArticlesTable = ({ className, selectedTable }: ArticlesTableProps) =>
{
  if (selectedTable === SelectedArticlesTable.Scheduled) {
    return <ScheduledArticles className={className} />;
  }
  if (selectedTable === SelectedArticlesTable.Generated) {
    return <GeneratedArticles className={className} />;
  }
  return null;
};

export default function Page() {
  const [selectedTable, setSelectedTable] = useState<SelectedArticlesTable>(
    SelectedArticlesTable.Scheduled,
  );

  const handleSelectTable = (table: SelectedArticlesTable) => {
    setSelectedTable(table);
  };
```

```
  return (
    <>
      <TopBar />
      <NavBar>
        <div className="m-5 flex gap-5">
          <button
            className={cn(
              "flex w-fit gap-2 border border-transparent p-2",
              selectedTable === SelectedArticlesTable.Scheduled
                ? "border-black"
                : "hover:border hover:border-solid hover:border-black",
            )}
            onClick={() =>
handleSelectTable(SelectedArticlesTable.Scheduled)}
          >
            <CalendarClock className="h-6 w-6" />
            <div className="font-semibold">Scheduled articles</div>
          </button>
          <button
            className={cn(
              "flex w-fit gap-2 border border-transparent p-2",
              selectedTable === SelectedArticlesTable.Generated
                ? "border-black"
                : "hover:border hover:border-solid hover:border-black",
            )}
            onClick={() =>
handleSelectTable(SelectedArticlesTable.Generated)}
          >
```

```
            <Notes className="h-6 w-6" />
            <div className="font-semibold">Generated articles</div>
          </button>
        </div>

        <ArticlesTable selectedTable={selectedTable} className="m-5" />
      </NavBar>
    </>
  );
}
```

articles/schedule/page.tsx

```
"use client";

import { NavBar } from "~/components/new-nav-bar";
import { TopBar } from "~/components/top-bar";
import { Separator } from "~/components/ui/separator";
import { StepsDisplay } from "./_components/steps-display";
import { StepsControl } from "./_components/steps-control";
import { useStepsControl } from "./_hooks/use-steps-control";

import { UploadFileStep } from "./_components/steps/upload-file-step";
import { ArticleGenerationStep as Step } from "~/types/components/steps";
import { ArticleNamesStep } from "./_components/steps/article-names-step";
import { ArticleOutlinesStep } from
"./_components/steps/article-outlines-step";
import { useEffect } from "react";
import { useSchedulerStore } from "~/stores/scheduler-store";
```

```
import { KeywordsStoreProvider } from "~/stores/keywords-store";

const stepComponentMap = {
  [Step.UploadFile]: UploadFileStep,
  [Step.Articles]: ArticleNamesStep,
  [Step.EditArticles]: ArticleOutlinesStep,
  [Step.ScheduledArticles]: null,
};

type StepComponentProps = React.HTMLAttributes<"div"> & {
  step: Step;
};
function StepComponent({ step, ...props }: StepComponentProps) {
  const Component = stepComponentMap[step];
  if (!Component) {
    return null;
  }

  return <Component {...props} />;
}

export default function Page() {
  const { resetSchedulerStore } = useSchedulerStore();
  const { currentStep, resetStepsStore } = useStepsControl();

  useEffect(() => {
    return () => {
      resetStepsStore();
```

```
      resetSchedulerStore();
    };
  }, []);

  // Might need to just pass the go back and go next functions to the step function
  return (
    <>
      <TopBar />
      <NavBar>
        <KeywordsStoreProvider>
          <StepsDisplay currentStep={currentStep} />
          <Separator />
          <StepComponent step={currentStep} className="m-5 pb-20" />
          <StepsControl className="absolute bottom-0 z-10 w-full border-t-2 bg-white" />
        </KeywordsStoreProvider>
      </NavBar>
    </>
  );
}
```

server/auth.ts

```
import { PrismaAdapter } from "@auth/prisma-adapter";
import {
  getServerSession,
  type User,
  type DefaultSession,
```

```
  type NextAuthOptions,
} from "next-auth";
import { type Adapter } from "next-auth/adapters";
import CredentialsProvider from "next-auth/providers/credentials";
import GoogleProvider, { type GoogleProfile } from
"next-auth/providers/google";


import { env } from "~/env";
import jwt from "jsonwebtoken";
import { db } from "~/server/db";
import { passwordHasher } from "./services/passwordHasher";


/**
 * Module augmentation for `next-auth` types. Allows us to add custom
properties to the `session`
 * object and keep type safety.
 *
 * @see
https://next-auth.js.org/getting-started/typescript#module-augmentation
 */
declare module "next-auth" {
  interface Session extends DefaultSession {
    user: {
      id: string;
      token?: string;
    } & DefaultSession["user"];
  }


  interface User {
```

```
    id: string;
    name?: string | null;
    email?: string | null;
    token?: string;
  }
}

/**
 * Options for NextAuth.js used to configure adapters, providers, callbacks,
etc.
 *
 * @see https://next-auth.js.org/configuration/options
 */
export const authOptions: NextAuthOptions = {
  callbacks: {
    signIn: async ({ account, profile }) => {
      if (account?.provider === "google" && profile) {
        const googleProfile = profile as GoogleProfile;
        const existingUser = await db.user.findUnique({
          where: { email: googleProfile.email },
        });

        if (!existingUser) {
          await db.user.create({
            data: {
              email: googleProfile.email,
              firstName: googleProfile.given_name,
              lastName: googleProfile.family_name,
```

```
            name: googleProfile.name,
            image: googleProfile.picture,
            emailVerified: new Date(),
            accounts: {
              create: {
                ...account,
              },
            },
          },
        });
      } else {
        await db.user.update({
          where: {
            email: profile.email,
          },
          data: {
            firstName: googleProfile.given_name,
            lastName: googleProfile.family_name,
          },
        });
      }
    }

    return true;
  },
  session: ({ session, token }) => {
    return {
      ...session,
```

```
      user: {
        ...session.user,
        id: token.sub,
        accessToken: token.accessToken,
      },
    };
  },
  jwt: ({ token, user }) => {
    if (user) {
      token.accessToken = user.token;
      token.id = user.id;
    }
    return token;
  },
},
jwt: {
  maxAge: 7 * 24 * 60 * 60, // 7 days
},
session: {
  strategy: "jwt",
},
adapter: PrismaAdapter(db) as Adapter,
providers: [
  GoogleProvider({
    clientId: env.GOOGLE_AUTH_CLIENT_ID,
    clientSecret: env.GOOGLE_AUTH_CLIENT_SECRET,
  }),
  CredentialsProvider({
```

```
id: "credentials",
name: "credentials",
credentials: {
  email: {
    label: "Email",
    type: "email",
    placeholder: "Enter email",
  },
  password: {
    label: "Password",
    type: "password",
    placeholder: "Enter password",
  },
},
async authorize(credentials): Promise<User | null> {
  if (!credentials) return null;
  const user = await db.user.findUnique({
    where: {
      email: credentials.email,
    },
  });

  if (!user) return null;

  const isPasswordValid = await passwordHasher.verifyPassword(
    credentials.password,
    user.password ?? "",
  );
```

```
        if (!isPasswordValid) return null;


        const userJson = JSON.stringify({
          id: user.id,
          email: user.email,
          name: user.name,
        });
        const token = jwt.sign(userJson, env.NEXTAUTH_SECRET!, {
          algorithm: "HS256",
        });


        return {
          id: user.id,
          email: user.email,
          name: user.name,
          token: token,
        };
      },
    }),
    /**
     * ...add more providers here.
     *
     * Most other providers require a bit more work than the Discord
provider. For example, the
     * GitHub provider requires you to add the `refresh_token_expires_in`
field to the Account
     * model. Refer to the NextAuth.js docs for the provider you want to
use. Example:
```

```
     *
     * @see https://next-auth.js.org/providers/github
     */
  ],
  pages: {
    signIn: "/login",
  },
};


/**
 * Wrapper for `getServerSession` so that you don't need to import the
`authOptions` in every file.
 *
 * @see https://next-auth.js.org/configuration/nextjs
 */
export const getServerAuthSession = () => getServerSession(authOptions);
```