МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2024 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

ВЕБЗАСТОСУНОК ДЛЯ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ ЛЮДИНИ ЗА РАХУНОК ТРЕКІНГУ ЗВИЧОК

Спеціальність 122 «Комп'ютерні науки»

122 – БКР – 402.2010320

Виконав студент 4-го курсу, групи 402
_____ М. І. Супруненко
«21» червня 2024 р.

Керівник: ст.викл. кафедри ІПЗ
_____М. В. Фаленкова
«21» червня 2024 р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____2024 р.

З А В Д А Н Н Я
на виконання кваліфікаційної роботи

Видано студенту групи 402 факультету комп'ютерних наук Супруненку Максиму Ігоровичу.

1. Тема кваліфікаційної роботи «Вебзастосунок для підвищення продуктивності людини за рахунок трекінгу звичок».

Керівник роботи Фаленкова Марина Володимирівна, ст.викл. кафедри ІПЗ
Затв. наказом Ректора ЧНУ ім. Петра Могили від «28» грудня 2023 р. № 271

2. Строк представлення кваліфікаційної роботи студентом «21» червня 2024 р.

3. Вхідні (початкові) дані до роботи: дослідження формування корисних й шкідливих звичок та їх формування.

Очікуваний результат: вебзастосунок для підвищення продуктивності за рахунок трекінгу звичок.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

- дослідження предметної області та аналіз існуючих аналогів;
- формування специфікації вимог до програмного забезпечення;
- визначення архітектури для проектування програмного забезпечення;
- моделювання та проектування програмного забезпечення;

- розробка програмного забезпечення;
- здійснення тестування роботи програмного забезпечення;
- проведення аналізу результатів розробки.

5. Перелік графічного матеріалу: презентація, 88 сторінок, містить 3 розділи, 40 ілюстрацій, 1 таблицю, 14 джерел в переліку посилань, додатки А, Б.

6. Завдання до спеціальної частини: «Аналіз охорони праці на робочих місцях фахівців з інформаційних технологій»

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	Алексєєва А.О., доцент кафедри екології	

Керівник роботи __ст.викл. кафедри ІПЗ Фаленкова М. В._____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання ___Супруненко М. І._____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «_14_» _____січня____ 2024 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: Вебзастосунок для підвищення продуктивності за рахунок трекінгу звичок

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників КРБ	10.11.2023	15.11.2023	Виконано
2	Отримання завдання на виконання КРБ	10.01.2024	15.01.2024	Виконано
3	Складання календарного плану роботи на весь період виконання КРБ	16.01.2024	30.01.2024	Виконано
4	Отримання завдання на переддипломну практику	15.04.2024	29.04.2024	Виконано
5	Проходження переддипломної практики, збір та аналіз матеріалів до КРБ	29.04.2024	11.05.2024	Виконано
6	Розробка звіту з переддипломної практики	12.05.2024	15.05.2024	Виконано
7	Виконання КРБ: аналіз утворення звичок, огляд існуючих технологій, розробка ПЗ	13.05.2024	22.06.2024	Виконано
8	Перший попередній захист КРБ на засіданні комісії кафедри	27.05.2024	27.05.2024	Виконано
9	Доробка та остаточне оформлення КРБ	28.05.2024	09.06.2024	Виконано
10	Другий попередній захист КРБ на засіданні комісії кафедри	10.06.2024	10.06.2024	Виконано
11	Подання КРБ рецензенту	13.06.2024	13.06.2024	Виконано
11	Подання КРБ, її електронної копії та інших документів (відгуку, рецензії) до захисту	17.06.2024	21.06.2024	Виконано
12	Захист БКР перед екзаменаційною комісією (ЕК)	24.06.2024	28.06.2024	Виконано

Розробив студент __Супруненко М. І._____
(прізвище, ім'я, по батькові студента) (*підпис*)

Керівник роботи __ст.викл. кафедри ІПЗ Фаленкова М. В.

(*посада, прізвище, ім'я, по батькові)* *(підпис)*

«_29_» _____01_____ 2024 р.

АНОТАЦІЯ

кваліфікаційної роботи студента групи 402 ЧНУ ім. Петра Могили

Супруненко Максима Ігоровича

Тема: «Вебзастосунок для підвищення продуктивності людини за рахунок трекінгу звичок»

Дана робота присвячена розробці вебзастосунку для підвищення продуктивності людини за рахунок створення й відслідковування звичок, які будуть знаходитися на вебсторінці.

Об’єкт роботи – процес розробки вебзастосунку.

Предмет роботи – алгоритм створення вебзастосунку для підвищення обізнаності ширшої аудиторії про формування звичок та підвищення продуктивності за рахунок трекінгу звичок.

Метою роботи є підвищення обізнаності більш широкої аудиторії про формування шкідлвих та корисних звичок, що спрямовано на підвищення продуктивності людини за рахунок їх трекінгу.

Кваліфікаційна робота складається з вступу, 3 розділів, висновків та переліку джерел посилань.

У вступі визначається актуальність теми, мета, предмет та об’єкт дослідження.

У першому розділі проведено аналіз існуючих застосунків-аналогів, визначення функціоналу, переваг та недоліків.

У другому розділі описується огляд мов, технологій та бібліотек, що використовуються для розробки, проєктування програмного забезпечення.

У третьому розділі описано процес розробки програмного забезпечення та його тестування.

Кваліфікаційна робота бакалавра викладена на 88 сторінок, містить 3 розділи, 40 ілюстрацій, 1 таблицю, 14 джерел в переліку посилань, додатки А, Б.

Ключові слова: *створення вебзастосунку, адаптивний інтерфейс, розробка на React, користь звичок, подразник, прагнення, відгук, винагорода.*

ABSTRACT

qualification work of the student of group 402

Black Sea National University named after Petro Mohyla

Maksym Suprunenko

Theme: “Web application for increasing human productivity by tracking habits”

This work is devoted to the development of a web application for increasing productivity by creating and tracking habits that will be on a web page.

Object of work - the process of developing a web application to increase productivity through habit tracking.

The subject of the work is the technology of creating a web application to raise awareness of the wider audience about habit formation and increase productivity through habit tracking.

The purpose of the work is to raise awareness of a wider audience about the formation of bad and good habits, which is aimed at increasing human productivity through habit tracking.

The qualification work consists of an introduction, 3 chapters, conclusions and a list of references.

The introduction defines the relevance of the topic, the purpose, subject and object of the study.

The first chapter analyzes existing analog applications, identifies the functionality, advantages, disadvantages.

The second section describes an overview of the languages, technologies, and libraries used for development and the software design process.

The third chapter describes the process of software development and testing.

The bachelor's thesis is 88 pages long, contains 3 chapters, 40 illustrations, 1 table, 14 sources in the list of references, and Appendix A, B.

Keywords: *creating a web application, adaptive interface, React development, benefit of habits, cue, craving, response, reward*.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД	–	База Даних
ПЗ	–	Програмне Забезпечення
ОС	–	Операційна система
CRUD	–	Create, Read, Update, Delete
CSS	–	Cascading Style Sheet
JS	–	JavaScript
JIT	–	Just-in-Time
AWS	–	Amazon Web Services
RDS	–	Relational Database Service

ВСТУП

Актуальність теми. На сьогоднішній день життя складається з безмежної кількості подразників, які в свою чергу викликають в нас прагнення до дії й винагороди по її завершенню. Всі ці етапи формують в нас звички, які потім впливають на наше життя кожного дня, навіть якщо ми не приділяємо цьому уваги. Відслідковування звичок, їх формування й відмова від негативних може значно підвищити продуктивність. Займатись цим можно, використовуючи безліч джерел: блокноти, програмні застосунки на смартфон, групи та канали у мессенджерах, та вебсайти. Спосіб можна обрати за своїм власним вподобанням, незважаючи на переваги легкої доступності застосунку у смартфоні, вебсайт звільняє нас від потреби позбуватись ще однієї шкідливої звички - постійної перевірки свого телефону. Вебсайти відрізняються від всіх інших тим, що для отримання актуальної інформації потрібні пристрій с браузером та доступ до інтернету. Вебсайти забезпечують наступне:

- отримування інформації цілодобово;
- вільний вибір інформації;
- великий спектр тем, для обрання;
- легкодоступність інформації.

Об’єкт роботи – процес розробки вебзастосунку для підвищення продуктивності за рахунок трекінгу звичок.

Предмет роботи – технології створення вебзастосунку для підвищення обізнаності більш широко аудиторії про звички, їх формування, користь і шкоду.

Метою кваліфікаційної роботи є підвищення обізнаності більш широкої аудиторії про силу маленьких змін, які мають значення, шляхом створення вебзастосунку для підвищення продуктивності за рахунок трекінгу звичок, що спрямовано на покращення рівня життя, позбавлення від зайвого стресу й зосередженості на головному.

Для досягнення поставленої **мети** необхідно виконати наступні завдання:

- аналіз предметної сфери;
- огляд застосунків-аналогів для трекінгу звичок;
- обговорення деталей проєкту;
- розробка backend-частини;
- розробка адмін-панелі;
- тестування вебзастосунку;
- завантаження готового продукту на сервер.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд застосунків-аналогів

Одним з важливих етапів аналізу вимог до програмного заезпечення є аналіз вимог. Оскільки це дозволяє визначити, чи існують на ринку такі програмні засоби, які можуть задовольнити потреби користувачів, або які можуть бути використані в якості основи для створення нового ПЗ. Також це допомагає визначитись у критеріях функціональності, інтерфейсу користувача, швидкодії, маштабованості та інших.

Було розглянуто аналоги до розроблюваного вебзастосунку, для виявлення основних функцій, недоліків, що потрібно буде вирішити для покращення розроблюваного застосунку. Були розглянуті наступні аналоги відслідковування звичок Avocation – трекер корисних звичок , Everyday - трекер звичок & щоденної рутини Productive – Трекер звичок і планувальник задач`.

Avocation – трекер корисних звичок

Якщо ви самі не можете себе змусити працювати над корисними звичками, то це за вас зробить Авокадо! Головний персонаж застосунку допоможе вам покращити різні сфери життя, серед яких: здоров’я, продуктивність,стосунки, фізична форма та креативність.

Ваше завдання – обрати оптимальну кількість часу, яку ви можете щоденно приділяти формуванню звички. Авокадо запропонує вам декілька напрямків роботи над собою. Наприклад, для того, щоб стати більш креативним, Авокадо радить вести блог, навчитися пояснювати або досліджувати усе навколо.

Відповідно до обраної поради застосунок сформує ваш щоденний план. Кожного дня вам потрібно вводити свої результати, які будуть зображатися у статистиці за тиждень/місяць/рік.

Everyday - трекер звичок & щоденної рутини

Головна перевага додатку – ви можете самі придумати собі звичку, описавши її при реєстрації. Окрім цього, вам варто обрати час, частоту виконання та колір позначки виконаної звички.

Попри те, що у застосунку Everyday мінімалістичний інтерфейс, ви зможете з легкістю розібратися у його роботі. Ви відмічаєте щоденний результат, а додаток формує вашу статистику, встановлюючи рейтинг формування звички. Також у застосунку є окремий розділ з корисними матеріалами: відео, статті та подкасти.

Productive – Трекер звичок і планувальник задач

Productive – це універсальний інструмент для формування цілей та звичок. При реєстрації вас запитають про годину, о якій ви прокидаєтесь, частоту вашої прокрастинації та рівень фокусування на завданнях.

Окрім цього, додаток запропонує вам чотири звички, над якими ви будете працювати: регулярне споживання води, спорт, медитація чи читання книги.

Встановлюєте щоденну ціль і кожного вечора відмічаєте свій результат. Тим паче, що сповіщення не дозволять вам розслабитись. Застосунок регулярно нагадує про вашу звичку та щоденний план.

Перевага додатку – анімаційний інтерфейс, який приваблює своєю зручністю та лаконічністю.

Серед застосунків для смартфону знайти якісний продукт не є проблемою, але при пошуці вебзастосунків ця задача стає або занадто тяжкою за рахунок використовування таких велечезних проектів як notion, або занадто непривабливаю через малу кількість аналогів або потреби завантажити програму на пристрій.

Веб-застосунок вирішує цю проблему доступом з будь-якого місця й пристроя, зберігає данні за рахунок прив’язки до облікового запису та надає доступ до свого прогресу.

1.2 Специфікація вимог

Призначенням застосунку є використання трекінгу звичок для підвищення продуктивності за рахунок розробки вебзастосунку. Було погоджено, що для створення ПЗ було використано додаткову бібліотеку – React. Сферою застосування є те, що вебзастосунок призначений для підвищення продуктивності за рахунок трекінгу звичок.

Основні характеристики користувачів: наявність пристрою (комп'ютер, ноутбук, телефон тощо), доступ до Інтернету.

Система складається з наступних частин:

1) Клієнтська частина (Front-end):
   - дизайн та інтерфейс вебсайту;
   - графічний контент та мультимедійні елементи;
   - клієнтська програмна логіка (Next.js 13, React, Convex, Tailwind);
2) Серверна частина (Back-end):
   - серверний хостинг та база даних;
   - серверна логіка (TypeScript);
   - інтерфейс взаємодії з базою даних;
3) Користувацька панель:
   - інтерфейс користовуча;
   - функціонал керування контентом (додавання, редагування, видалення);
4) База даних:
   - збереження прогресу звичок;
   - взаємодія з БД.

Основне обмеження у використанні застосунку – доступ до інтернету.

Функція допомагає користувача додавати нові звички для покращення своєї ефекивності, а також позбуття від негативних.

Вхідна інформація – заголовок звички українською та англійською, опис англійською і українською та, можливо, зображення.

Вихідна інформація – згенерована сторінка шаблону, який готовий для друку за потребую.

В цьому ПЗ вхідні данні отримуються від імені користовуча, що, заповнюючи свої активні звички, наповнює застосунок вмістом.

Обмін даними відбувається через обробку запитів у контроллері та використанням Convex. БД – AWS RDS.

Вимоги до технічного забезпечення. для розробки програмного забезпечення немає значних технічних обмежень. Але для підтримки необхідних програм та комфортної розробки бажано мати комп’ютер або ноутбук з оперативної пам’яті місткістю не менше 8 Гбайт.

Архітектура програмної системи. Система складається з: клієнтської частини, серверної частини та БД. Застосунок побудований з використанням Next.js 13, React, Convex, Tailwind. В якості БД для застосунку обрано AWS RDS.

Мережне програмне забезпечення/ Підчас розробки було використано ОС Windows 11, для написання коду було використано IDE VS Code, для перегляду застосунку веб браузер Google Chrome й Opera GX.

Програмне забезпечення ведення інформаційної бази. CRUD-операцій виконується маніпуляції з БД.

Мова і технологія розробки ПЗ. Програмне забезпечення має розроблюватися з використанням React. Мови розробки – TypeScript.

Вимоги до зовнішніх інтерфейсів. Інтерфейс має бути інтуїтивно зрозумілим користувачеві, для цього були використані стандартні рішення при верстці.

Програмний застосунок має бути доступним на будь-якому пристрої з будь-яким розширенням екрану, тому інтерфейс має бути адаптивним.

React – це потужна бібліотека JavaScript для створення користувацьких інтерфейсів, яка надає велику кількість інструментів для зручної та швидкої розробки.

Загалом, React є потужним інструментом для розробки веб-інтерфейсів, який надає зручність, продуктивність та високу швидкість розробки.

Програмне забезпечення має ряд ключових властивостей, які визначають його якість та ефективність. Ось основні з них:

– функціональність програмного забезпечення виконувати заплановані функції та завдання відповідно до специфікацій. Вона включає правильність, точність, придатність для використання та взаємодію з іншими системами;

– надійність вдображає стабільність роботи програмного забезпечення за різних умов, його здатність виконувати функції без збоїв та помилок протягом певного часу;

– зручність використання, яка відображає легкість, з якою користувачі можуть навчитися і ефективно використовувати програмне забезпечення. Сюди входять зручний інтерфейс, зрозуміла навігація та підтримка користувач;

– сумісність програмного забезпечення працювати разом з іншими системами та програмами без конфліктів. Це включає можливість обміну даними та інтеграції з іншими системами;

– мобільність, за рахунок якої програмне забезпечення може бути перенесеним з однієї платформи на іншу без значних змін. Це особливо важливо для програм, які повинні працювати на різних типах обладнання та операційних систем;

– підтримуваність і ростота, з якою програмне забезпечення може бути модифіковане для виправлення помилок, оновлення функціональності або адаптації до змін у навколишньому середовищі. Це включає добре документований код та структуру програмного забезпечення;

– безпека, яка є здатністю програмного забезпечення захищати дані та функції від несанкціонованого доступу, зловживань та атак. Це включає шифрування, автентифікацію та управління правами доступу.

Кожна з цих властивостей є важливою для створення якісного програмного забезпечення, яке відповідає потребам користувачів і забезпечує надійну та ефективну роботу системи.

1.3 Дослідження впливу звичок

Звички відіграють ключову роль у формуванні нашого життя. Це повторювані дії, які ми виконуємо автоматично, часто без особливої свідомості, і вони можуть мати як позитивний, так і негативний вплив на наше здоров'я, продуктивність, стосунки та загальне благополуччя. У цьому розділі ми розглянемо, як звички впливають на людину в довгостроковій перспективі, зокрема враховуючи негативні звички.

Позитивні звички можуть значно покращити якість життя, сприяючи досягненню особистих і професійних цілей, зміцненню здоров'я та покращенню емоційного стану. Ось декілька ключових аспектів позитивного впливу звичок:

– здоров'я та фізична форма, регулярна фізична активність, здорове харчування та достатній сон – це звички, які сприяють довголіттю та запобігають багатьом хронічним захворюванням. Наприклад, звичка робити ранкову зарядку може не лише покращити фізичний стан, а й підвищити рівень енергії та продуктивність протягом дня;

– продуктивність та досягнення. Систематичний підхід до роботи, організація часу та постійне навчання – це звички, що допомагають досягати високих результатів у професійній сфері. В книзі «Atomic Habits» [1] Джеймс Клір описує принцип 1%, згідно з яким щоденне покращення на 1% може призвести до значних успіхів з часом;

– соціальні зв'язки: Звичка підтримувати регулярні контакти з друзями та родиною, висловлювати вдячність і допомагати іншим сприяє зміцненню соціальних зв'язків і покращенню емоційного стану.

Негативні звички можуть мати руйнівний вплив на життя людини, ведучи до погіршення здоров'я, зниження продуктивності та проблем у стосунках.

Розглянемо декілька прикладів:

– шкідливі звички для здоров'я. Куріння, зловживання алкоголем та нездорове харчування можуть призвести до серйозних медичних проблем, таких як хвороби серця, рак та ожиріння. Наприклад, звичка перекушувати на ніч може спричинити набір ваги та порушення сну;

– втрата часу та продуктивності Звичка прокрастинувати, тобто відкладати важливі завдання на потім, може значно знижувати ефективність роботи та призводити до стресу. Джеймс Клір в "Atomic Habits" зазначає, що невеликі негативні звички можуть накопичуватися і створювати великі проблеми в майбутньому;

– проблеми в стосунках. Нездатність контролювати гнів або постійне невдоволення можуть руйнувати стосунки з оточуючими. Постійна критика, зневажливе ставлення або байдужість – це звички, що можуть призвести до розриву соціальних зв'язків і самотності.

Процес формування нових звичок та зміни старих вимагає часу, наполегливості та самоконтролю. Ключові принципи, які допомагають у цьому процесі, включають:

– маленькі кроки, найефективніший спосіб формування нових звичок – це починати з маленьких, легко досяжних кроків. Цей підхід дозволяє знизити психологічний бар'єр та поступово вбудувати нову поведінку в щоденний ритуал;

– чіткі наміри: Важливо мати чітке уявлення про те, яку звичку ви хочете розвинути та чому вона важлива. Наприклад, замість загального наміру «хочу займатися спортом» краще визначити конкретну дію, як «робити зарядку щоранку протягом 15 хвилин»;

– середовище та тригери. Змінюючи своє середовище, можна значно полегшити процес формування нових звичок. Наприклад, якщо ви хочете почати їсти здоровіше, наповніть свій холодильник корисними продуктами і усуньте спокуси;

– моніторинг прогресу. Ведення щоденника або використання мобільних додатків для відстеження прогресу може мотивувати і допомогти зберегти регулярність у виконанні нових звичок.

Звички мають потужний вплив на наше життя в довгостроковій перспективі. Позитивні звички можуть значно покращити здоров'я, підвищити продуктивність і зміцнити стосунки, тоді як негативні звички можуть мати протилежний ефект. Усвідомлення важливості звичок і вміння керувати ними – ключ до успішного та щасливого життя.

Висновки до першого розділу

В першому розділі кваліфікаційної роботи бакалавра було оглянуто та проведено аналізовано існуючі застосунки-аналоги, було зазначено наступні функції та властивості ПЗ:

- призначення проєкту;
- загальний опис;
- функції системи;
- вимоги до інформаційного забезпечення;
- вимоги до технічного забезпечення;
- вимоги до програмного забезпечення;
- вимоги до зовнішніх інтерфейсів;
- властивості програмного забезпечення.

2 ТЕХНІЧНІ ПРОГРАМНІ ЗАСОБИ

2.1 Огляд технологій

При розробці вебзастосунку «Вебзастосунок для підвищення продуктивності за рахунок трекінгу звичок» використано наступні технології:

- мови програмування – TypeScript;
- система управління базою данних – MySQL;
- інструмент для контролю версій – Git;
- бібліотеки для реалізацій деяких функцій – Next.js 13, React, Convex, Tailwind.

Обрання стеку технологій грає важливу роль у процессі розробки програмного забезпечення, так як він визначає основі інструменти, що будуть використовуватися для створення проєкта.

Важливість обрання стеку технологій пов’язана з декількома факторами:

- функціональність, бо кожна технологія має свої особливості та можливості і вибір стекутехнологій повинен відповідати вимогам проєкту;
- продуктивність і вибір стеку технологій може суттєво впливати на продуктивність застосунку. Правильний вибір технологій може допомогти оптимізувати продуктивність та прискорити роботи системи;
- вибір технологій також може впливати на швидкість та комфортність розроки. Деякі технології та фреймворки можуть значно спростити процес розробки та прискорити процес створення застосунку;
- спільнота та підтримка, вибір популярних та широко використоуваних технологій може забезпечити наявність розвиненої спільноти розробників та ресурсів підтримки, що може значно полегшити розробку та підтримку системи;
- вибір стеку технологій може впливати на рівень безпеки системи. Деякі технології та фрейморки маюит механізм захисту від уразливостей та атак, що може підвищити безпеку програми.

Вцілому правильний вибір стеку технологій може допомогти створити високоякісний та продуктивний застосунок, а також полегшити процес розробки та підтримки.

2.2 Next.js

Next.js - це фреймворк для React, який забезпечує можливість швидко створювати масштабовані веб-застосунки з урахуванням найкращих практик розробки. Основні переваги Next.js полягають у вбудованій підтримці серверного та статичного рендерінгу, автоматичному код-сплітінгу та оптимізації проекту для продуктивногорозгортання (рис. 2.1).

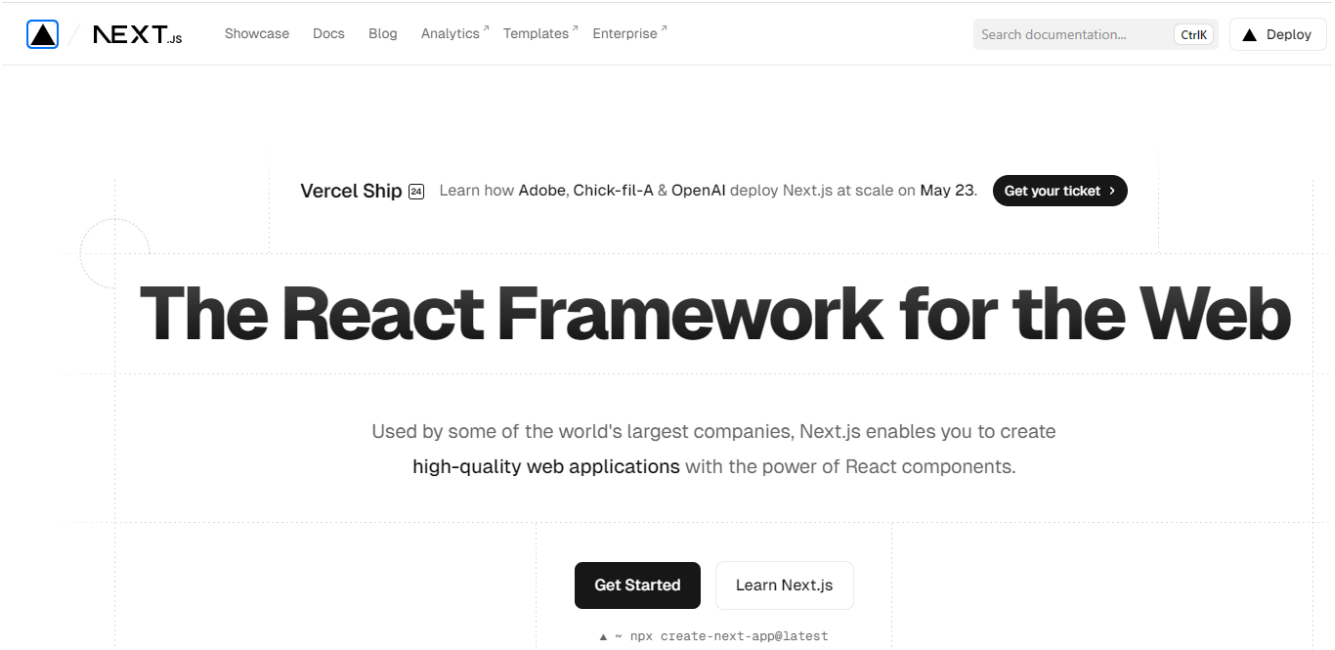

Рисунок 2.1 – Next.js

Характеристики Next.js наведені нижче.

1. Рендеринг на стороні сервера (SSR). Next.js дозволяє виконувати рендеринг React-компонентів на сервері перед відправкою їх до клієнта. Це покращує SEO та прискорює завантаження сторінок.

2. Статичний рендеринг (SSG). Next.js підтримує генерацію статичних сторінок під час збірки (Static Site Generation), що забезпечує дуже високу швидкість завантаження та безпеку.

3. Автоматичне розбиття коду. Код розбивається автоматично, що зменшує розмір файлів JavaScript, які потрібно завантажити браузеру, та покращує продуктивність.

4. Маршрутизація на основі файлів. Next.js використовує файлову систему для визначення маршрутів, що робить створення нових сторінок простим та інтуїтивно зрозумілим.

5. API-роути. Next.js дозволяє створювати API-ендпойнти поряд з вашим фронтенд-кодом. Це корисно для створення бекенду або обробки серверних функцій без необхідності в окремому сервері.

6. Hot Module Replacement (HMR). Ця функція дозволяє оновлювати модулі під час роботи застосунку без повного перезавантаження сторінки, що прискорює розробку.

7. Інтеграція з популярними CSS-методами. Next.js підтримує CSS-in-JS, CSS-модулі, а також інші бібліотеки стилізації, такі як styled-components.

Загалом, Next.js 13 є потужним інструментом для розробки сучасних веб-додатків, який надає розробникам широкі можливості та зручні інструменти для створення високоякісних додатків з урахуванням найновіших технологій та практик розробки.

2.3 Мова програмування - TypeScript

TypeScript – це мова програмування, розроблена компанією Microsoft, яка розширює стандартний JavaScript додаванням статичної типізації та деяких інших сучасних функцій. Вона призначена для розробки великих та складних веб-додатків, де важлива безпека, стабільність та масштабованість коду.

Наведена частина кода демонструє можливість створювати і вкладати компоненти один в одного, фактично маємо основну сторінку, яка складається з наступних частин: блоки контейнери, всередині header, основна частинаб footer (рис. 2.2).

```
import { Heading } from "./_components/heading";
import Heroes from "./_components/heroes";
import { Footer } from "./_components/footer";

const MarketingPage = () => {
  return (
    <div className="min-h-full flex flex-col">
      <div className="flex flex-col items-center justify-center md:justify-start text-center gap-y-8 flex-1 px-6 pb-10">
        <Heading></Heading>
        <Heroes></Heroes>
        <Footer></Footer>
      </div>
    </div>
  );
};
export default MarketingPage;
```

Рисунок 2.2 – Фрагмент TS коду

TypeScript є потужним інструментом для розробки сучасних веб-додатків, який надає розробникам широкі можливості та зручні інструменти для створення безпечного, стабільного та масштабованого коду. Основні особливості приведені нижче.

1. Статична типізація. TypeScript додає статичні типи до JavaScript, що дозволяє виявляти помилки на етапі компіляції, а не під час виконання.

2. Сумісність з JavaScript. TypeScript є надбудовою JavaScript, що означає, що будь-який коректний JavaScript-код є також коректним TypeScript-кодом. Це дозволяє поступово переходити на TypeScript в існуючих проектах, використовуючи його переваги лише там, де це необхідно.

3. Класова Система та Об’єктно-орієнтоване Програмування (ООП). TypeScript підтримує класи, наслідування, інтерфейси та модифікатори доступу (public, private, protected), що полегшує розробку великих та складних додатків.

4. Модулі. TypeScript підтримує модульну архітектуру, що дозволяє розділяти код на незалежні частини, які можна імпортувати та експортувати між файлами.

5. Інтерфейси. Інтерфейси дозволяють визначати контракти для об’єктів, що сприяє чіткішій структурі коду та полегшує його підтримку та рефакторинг.

6. Розширення синтаксису. TypeScript включає деякі функції, які ще не були реалізовані в стандарті ECMAScript, такі як декоратори, що дозволяє використовувати нові можливості мови раніше.

7. Інтеграція з Розробницькими Інструментами. TypeScript забезпечує покращену інтеграцію з IDE та редакторами коду, такими як Visual Studio Code, надаючи автозаповнення, перевірку типів в реальному часі та інші корисні інструменти.

Використання TypeScript надає купу переваг:

- виявлення помилок на етапі компіляції. Завдяки статичній типізації, багато помилок можна виявити до виконання коду, що значно зменшує кількість багів у продакшені;
- покращена підтримка IDE. Багато сучасних IDE підтримують TypeScript, забезпечуючи покращену навігацію по коду, автозаповнення та інші інструменти, що підвищують продуктивність розробки;
- зрозуміла документація та читабельність коду. Статичні типи виступають як додаткова документація, що полегшує розуміння коду іншими розробниками та зменшує кількість помилок при зміні та підтримці коду;
- легкий перехід з JavaScript. Оскільки TypeScript є надбудовою JavaScript, перехід на нього не вимагає повного переписування існуючого коду, що робить його зручним вибором для покращення якості великих проектів.

TypeScript широко використовується в розробці фронтенд та бекенд додатків. Деякі популярні фреймворки та бібліотеки, такі як Angular, розроблені з використанням TypeScript. Інші фреймворки, такі як React і Vue, також мають хорошу підтримку TypeScript.

2.4 Середа розробки IDE – Visual Studio Code

Для написання коду використовувалось безкоштовне програмне забезпечення **VS Code** з приємним для використання інтерфейсом (рис. 2.3).

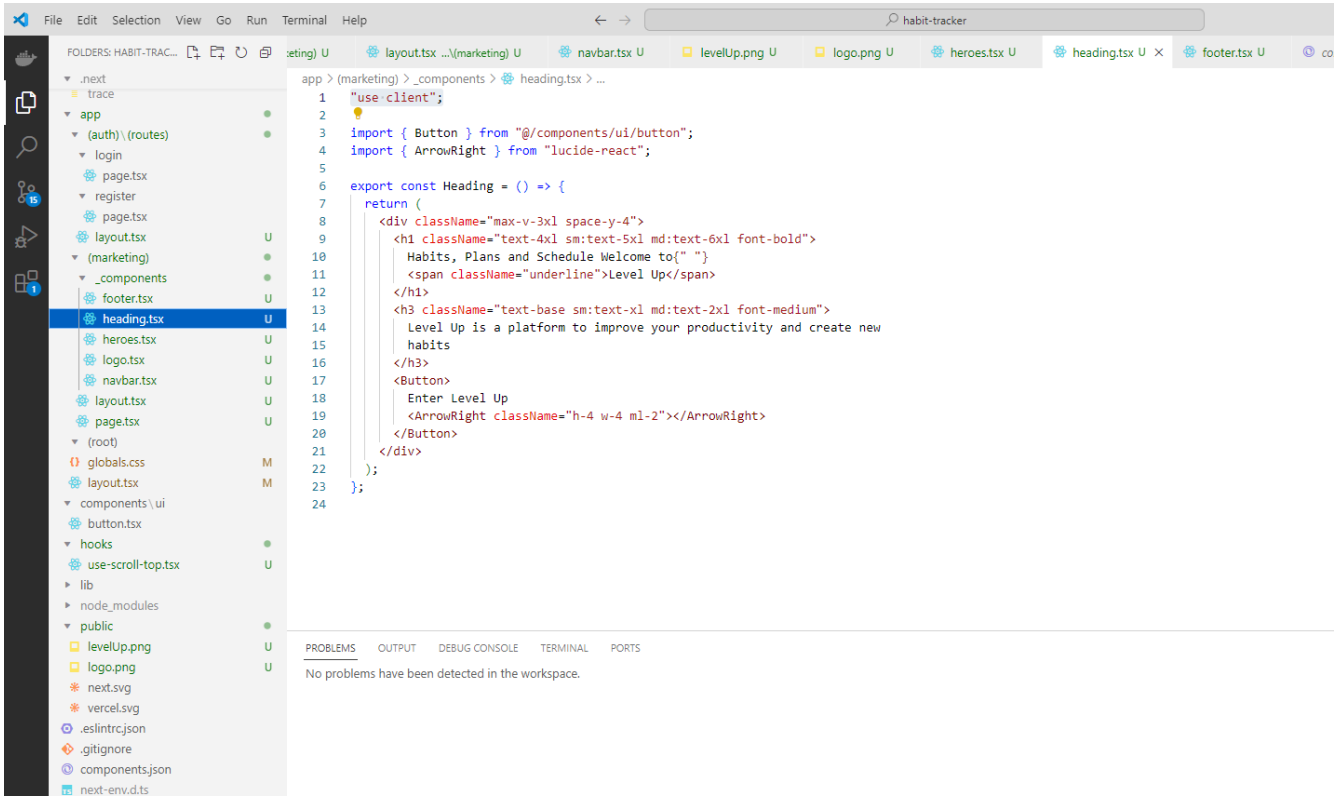

Рисунок 2.3 – Зображення інтерфейсу **VS Code**

Visual Studio Code — засіб для створення, редагування та зневадження сучасних вебзастосунків і програм для хмарних систем. Visual Studio Code розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux і OS X.

VS Code став настільки популярним тому що:

– крос-платформний VS Code працює на різних операційних системах, таких як Windows, macOS та Linux, що робить його зручним для широкого кола користувачів;

– підтримує багато мов програмування, завдяки розширенням, VS Code підтримує велику кількість мов програмування, включаючи JavaScript, Python, Java, C++, HTML, CSS, та багато інших;

– ви можете змінити мову для вибраного файлу, що дозволяє використовувати його для різних типів проектів;

– надає мовну документацію;

– вбудована інтеграція Git, вбудована підтримка системи контролю версій Git дозволяє легко виконувати команди, такі як commit, pull, push, і переглядати історію змін безпосередньо з редактора;

– розширення та налаштування: Магазин розширень (Visual Studio Code Marketplace) пропонує тисячі розширень, які додають нові функції або інтеграції, такі як підтримка мов програмування, теми, інструменти для розробки, налаштування продуктивності та інші;

– інтелектуальне автозавершення та IntelliSense: VS Code має потужні функції автозавершення коду, синтаксичний аналіз, підказки та документацію в реальному часі. IntelliSense надає контекстні пропозиції для функцій та методів;

– налагодження (debugging): Редактор має вбудовані засоби для налагодження коду, що дозволяють встановлювати точки зупинки, крокувати по коду, переглядати змінні та виконувати інші налагоджувальні дії;

– термінал: Вбудований термінал дозволяє працювати з командною строкою безпосередньо в редакторі, що зручно для виконання команд, запуску скриптів або управління проектом;

– живий сервер (Live Server): За допомогою розширення Live Server можна запускати локальний сервер і автоматично оновлювати веб-сторінки при збереженні файлів, що значно спрощує розробку веб-додатків;

– синхронізація налаштувань: Можливість синхронізувати налаштування, розширення та інші параметри між різними пристроями, використовуючи обліковий запис Microsoft або GitHubю;

– інтеграція з Docker та Kubernetes: Підтримка Docker та Kubernetes дозволяє розробникам легко працювати з контейнерами та керувати ними безпосередньо з редактора;

– висока продуктивність та легкість: VS Code відомий своєю швидкістю та невеликим споживанням ресурсів, що робить його ефективним навіть на менш потужних комп’ютерах.

2.5 Система контролю версій Git

Також кінцева версія була завантажена на безкоштовний вебсервіс **GitHub –** найбільший вебсервіс для хостингу IT-проєктів і їх спільної розробки. (див. рис. 2.4).

Коли розробники створюють новий проект, вони продовжують вносити оновлення до коду. Навіть після запуску проектів їм все одно потрібно оновлювати версії, виправляти помилки, додавати нові функції тощо.

Git — це розподілена система керування версіями, створена Лінусом Торвальдсом у 2005 році. Вона використовується для відстеження змін у файлах, координації роботи кількох людей над одним проектом, та збереження історії змін. Ось основні аспекти Git:

– розподілена архітектура: Кожен розробник має повну копію всього репозиторію, включаючи всю історію змін. Це забезпечує незалежність від центрального сервера та підвищує надійність;

– швидкість і ефективність: Git оптимізований для високої швидкості роботи, що дозволяє швидко виконувати операції, такі як commit, merge та push;

– гіілки та злиття: Git підтримує легке створення та об'єднання гілок (branches), що дозволяє паралельно працювати над різними функціями або виправленнями. Механізм злиття (merge) забезпечує об'єднання змін з різних гілок;

– знімки (snapshots): Git зберігає стан файлів як знімки, а не просто різниці (diff). Це дозволяє швидко перемикатися між різними версіями файлів;

– історія комітів: Кожна зміна в Git зберігається як коміт з унікальним ідентифікатором (хешем). Це дозволяє точно відстежувати історію змін та авторів кожної зміни;

– віддалені репозиторії: Git дозволяє працювати з віддаленими репозиторіями (наприклад, на GitHub), що полегшує спільну роботу над проектами.

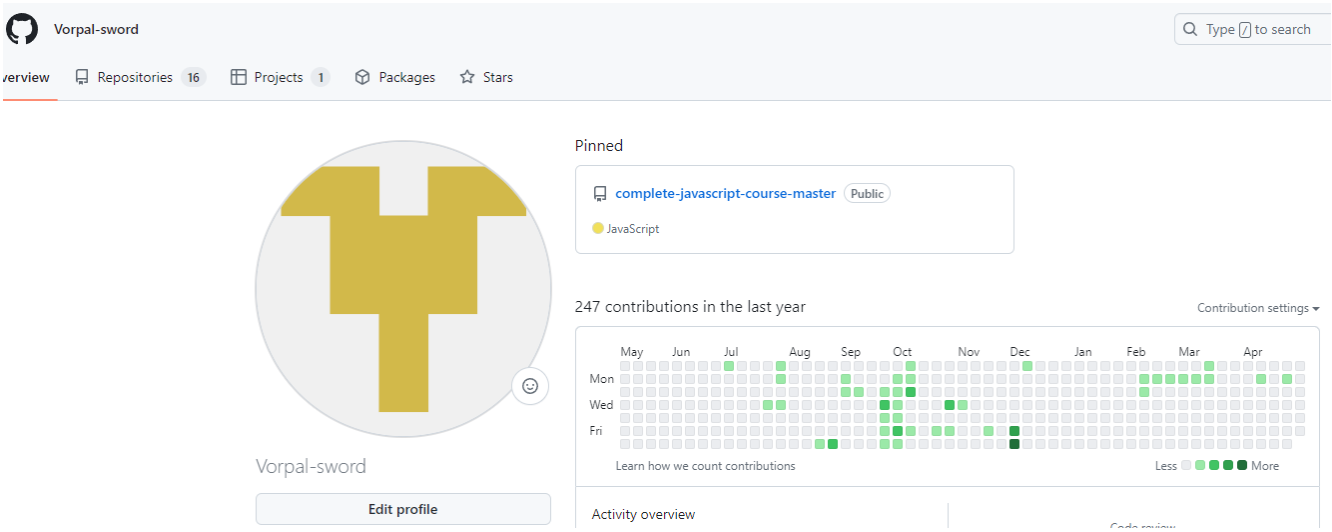

Рисунок 2.4 – Зображення інтерфейсу GitHub

Система контролю версій допомагає відстежувати зміни, внесені до бази коду. Більше того, він записує, хто вніс зміни та може відновити стертий або змінений код.

Перезаписаних кодів не існує, оскільки Git зберігає кілька копій у сховищі

GitHub є одним з найпопулярніших сервісів для розробників, що використовують Git, і надає потужний набір інструментів для управління проектами, співпраці та автоматизації процесів розробки.

2.6 Розробка інтерфейсів у Figma

Figma – це популярний інструмент для дизайну інтерфейсів користувача та спільної роботи над проєктами. Він має ряд особливостей та можливостей, які зробили його одним із найпопулярніших виборів серед дизайнерів та команд (див. рис. 2.5).

У Figma дві ключові особливості: доступ до макету прямо з вікна браузера і можливість спільної роботи над документами.

Figma використовується у різних сферах, таких як дизайн UI/UX, брендінг та графічний дизайн, протитипування.

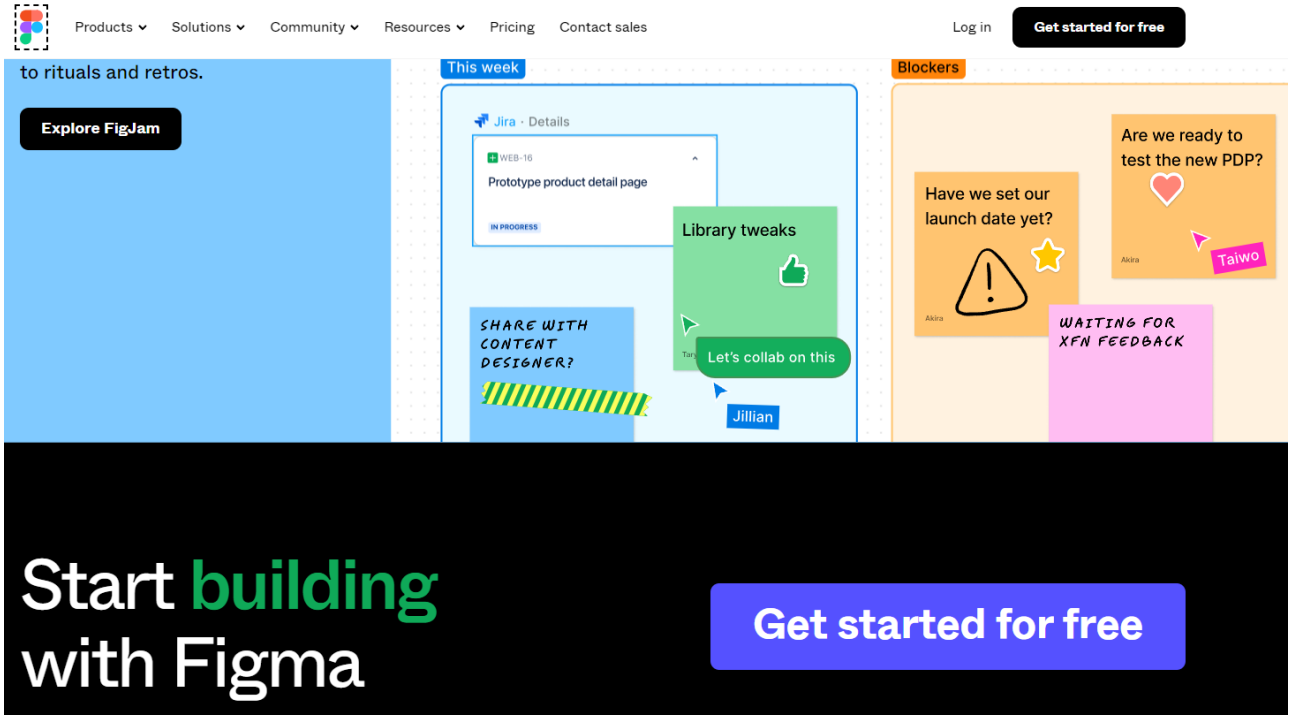

Рисунок 2.5 – Інтервейс у Figma

Figma має ряд вагомих плюсів:

– безкоштовна версія з лімітом за кількістю користувачів та проектів;

– робота у хмарі та через програму з хорошою продуктивністю;

– завантаження своєї бібліотеки компонентів, необмежене хмарне зберігання файлів;

– фрейми (полотна в полотнах) із встановленими розмірами для різних пристроїв - телефонів, планшетів, комп'ютерів;

– інтеграції – для тестування прототипів (Maze), для продуктивності (Slack, Asana), для розробників (Zeplin, Avocode, zeroheight);

– плагіни - для роботи з графікою (Blobs та Get Waves, Image Palette), для анімації (GiffyCanvas, LottieFiles), для організації роботи (LilGrid, Design Lint).

Додатковою перевагою Figma є:

– колаборативне середовище: Figma дозволяє кільком користувачам одночасно працювати над проектом, спільно редагувати макети, коментувати елементи та взаємодіяти в реальному часі. Це робить спільну роботу команди над проектом більш ефективною і зручною;

– прототипування та анімація: Figma надає зручні інструменти для створення прототипів взаємодії та анімації. Ви можете створювати інтерактивні переходи між екранами, додавати анімацію до об'єктів та ефекти переходу. Це допомагає презентувати та тестувати дизайн-концепції перед реалізацією;

– бібліотеки компонентів: Figma дозволяє створювати бібліотеки з повторюваними компонентами, такими як кнопки, поля вводу, меню тощо. Це спрощує роботу з дизайну, дозволяє швидко створювати нові екрани та забезпечує єдність стилів у всьому проекті;

– інтеграції з іншими інструментами: Figma має широкий вибір інтеграцій з іншими популярними інструментами для дизайну та розробки. Це дозволяє зручно обмінюватися даними між Figma та іншими сервісами, спрощує роботу з командою та забезпечує безперебійний робочий процес;

– доступність з різних платформ: Figma є кросс-платформеним і доступним як для роботи у веб-браузері, так і для використання в мобільних пристроях з встановленою спеціальною програмою. Це дає можливість працювати над проектами з будь-якого пристрою з доступом.

2.7 Бібліотека React

React - це бібліотека JavaScript, розроблена компанією Facebook, яка використовується для створення користувацьких інтерфейсів веб-додатків. Вона відома своєю ефективністю, зручністю використання та широкою популярністю серед розробників (див. рис. 2.6).

Компонентна архітектура: React базується на компонентній архітектурі, де інтерфейс розбивається на невеликі, незалежні компоненти. Компоненти можуть бути вкладені один в одного, що спрощує розробку, тестування та підтримку коду.

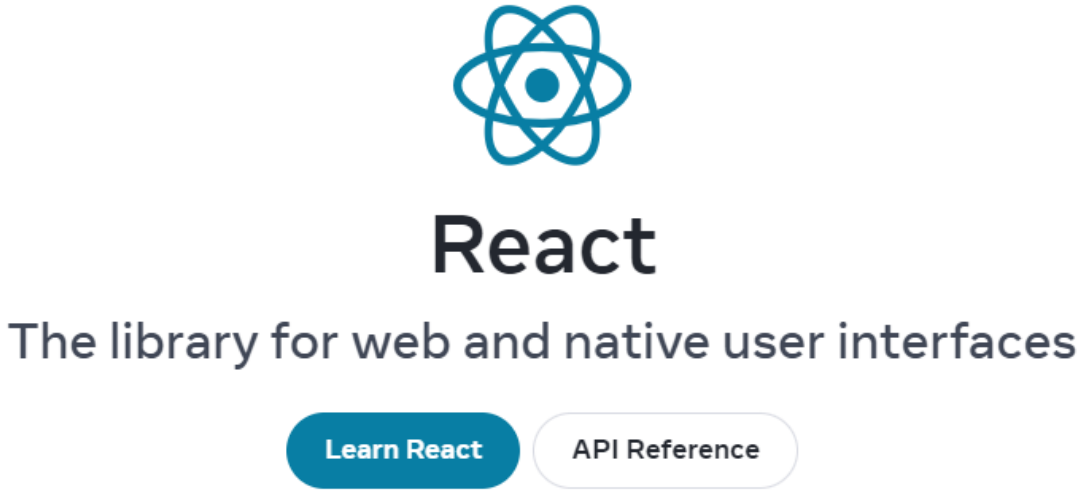

Рисунок 2.6 – Зображення інтерфейсу GitHub

Віртуальний DOM: Віртуальний DOM - це концепція, яка дозволяє React ефективно оновлювати сторінки та компоненти, шляхом оновлення лише тих елементів, які змінилися. Це забезпечує високу швидкість рендерингу та покращує продуктивність додатків.

JSX: JSX - це розширення синтаксису JavaScript, яке дозволяє вбудовувати HTML-подібний код безпосередньо в JavaScript. Це робить написання компонентів більш зручним та зрозумілим, а також сприяє розширенню можливостей розробки.

Статичний та серверний рендеринг: React підтримує як статичний, так і серверний рендеринг, що дозволяє створювати швидкі та SEO-оптимізовані веб-додатки. Це особливо важливо для розробки інтернет-магазинів, блогів та інших веб-додатків, де важлива доступність та індексація контенту пошуковими системами.

Велике співтовариство та екосистема: React має велике та активне співтовариство розробників, яке надає безліч корисних бібліотек, інструментів та

ресурсів для підтримки розробки. Це робить React одним з найпопулярніших і найбільш підтримуваних фреймворків для створення веб-додатків.

2.8 Convex

Convex - це бібліотека для керування станом в React, яка спрощує роботу зі станом додатку. Вона використовує контекст та реактивну програмну архітектуру для забезпечення ефективного управління станом та його оновлення (див. рис. 2.7).

Convex - це бекенд з відкритим вихідним кодом для розробників додатків. Універсальна платформа з продуманими, орієнтованими на продукт API. Все, на що заслуговує ваш продукт для створення, запуску та масштабування.

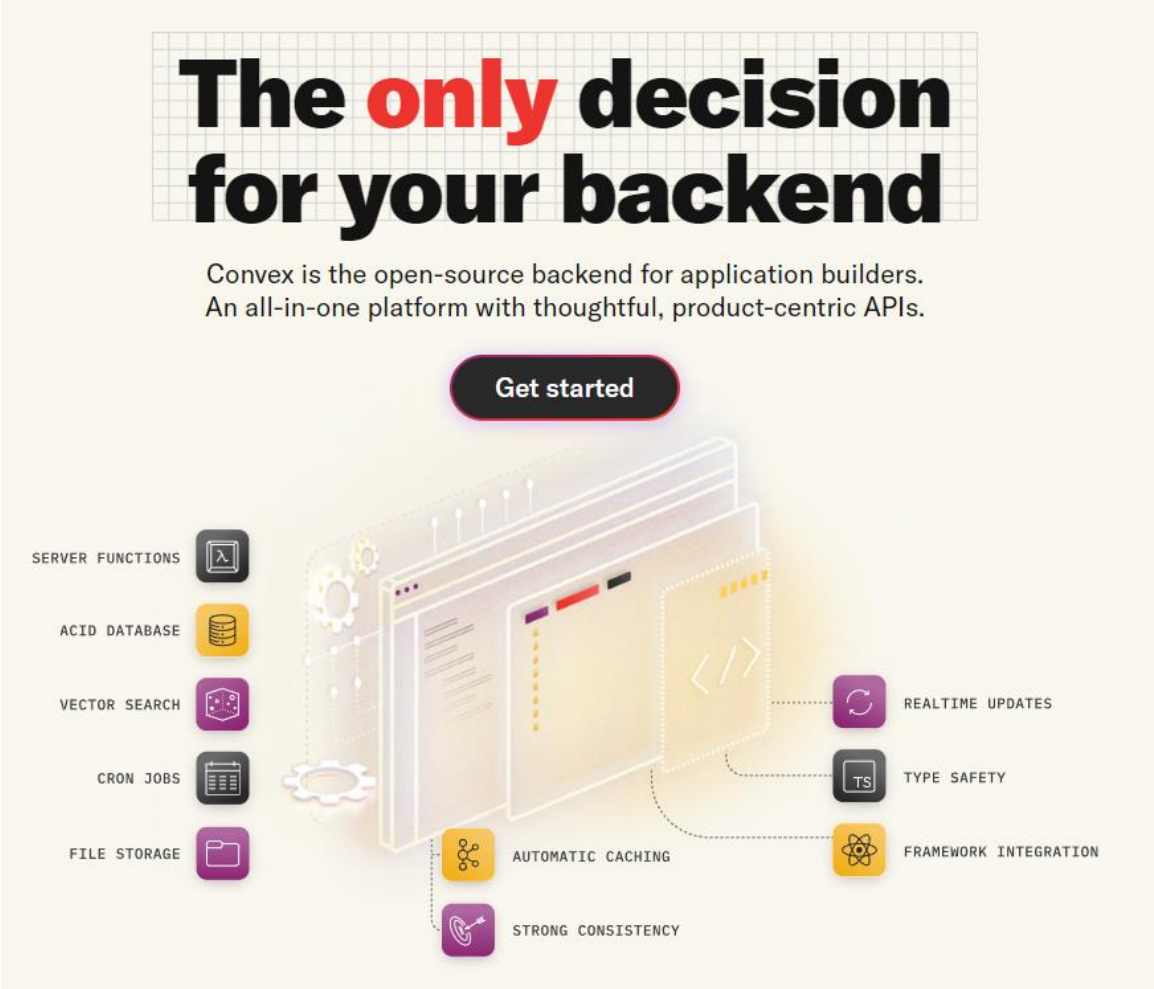

Рисунок 2.7 – Рішення для серверної частини

База даних Convex в режимі реального часу відстежує всі залежності для кожної функції запиту. Щоразу, коли змінюється будь-яка залежність, включаючи будь-які рядки бази даних, Convex повторно запускає функцію запиту і запускає оновлення всіх активних підписок на клієнті.

Функції Convex виконуються на стороні сервера в ізольованому середовищі виконання в самій базі даних Convex. Вони забезпечують прямий ефективний доступ до основних даних, а також до планування, зберігання, дій загального призначення тощо.

Реляційні бази даних, такі як MySQL і PostgreSQL, були основою веб-додатків з моменту появи перших інтерактивних веб-додатків. Вони лежать в основі багатьох найпопулярніших веб-додатків в Інтернеті і мають перевірену репутацію надійності. Але при всьому цьому, основною перевагою є те, що за рахунок Convex маємо можливість витрачати свій час на створення своїх продуктів, а не на управління інфраструктурою чи суперечки з SQL.

2.9 Tailwind CSS

Tailwind CSS: Tailwind CSS - це інструмент для створення користувацьких дизайнів, який працює на основі набору класів. Замість написання CSS вручну, розробники можуть використовувати класи Tailwind для стилізації елементів.

Tailwind відрізняється від традиційних CSS-фреймворків тим, що він не надає готових компонентів (як Bootstrap чи Foundation), а натомість забезпечує низькорівневі утиліти для побудови власних унікальних дизайнів. Це дозволяє прискорити процес розробки і зменшити кількість коду (див. рис. 2.8).

```
return (
  <div className="pl-[54px] group relative">
    {!!initialData.icon && !preview && (
      <div className="flex items-center gap-x-2 group/icon pt-6">
        <IconPicker onChange={onIconSelect}>
          <p className="text-6-xl hover:opacity-75 transition">
            {initialData.icon}
          </p>
          <Button
            onClick={onRemoveIcon}
            className="rounded-full opacity-0 group-hover/icon:opacity-100 transition"
```

Рисунок 2.8 – Приклад використання у коді

Основні особливості та функції Tailwind CSS включають в себе утилітрані css-класи, адаптивний дизайн, теми і налаштування, jit-компіляцію, плагіни та розширейння, простіше обслуговавання та масштабованість:

– tailwind підтримує адаптивні класи, які дозволяють змінювати стилі в залежності від розміру екрану. Наприклад, md:text-lg означає, що текст буде великим на середніх екранах і більших;

– власні breakpoints легко налаштовуються у конфігураційному файлі;

– всі налаштування, як-от кольори, шрифти, відступи, можна налаштовувати у конфігураційному файлі tailwind.config.js;

– ви можете визначати власні теми та розширювати базову конфігурацію Tailwind для своїх потреб;

– JIT-режим дозволяє значно скоротити розмір кінцевого CSS-файлу, генеруючи тільки ті класи, які використовуються у вашому проекті. Це призводить до швидшого завантаження сторінок і кращої продуктивності;

– tailwind підтримує плагіни, які дозволяють додавати нові утиліти та компоненти. Існує багато офіційних та сторонніх плагінів для Tailwind, які можна легко інтегрувати;

– утилітарний підхід робить код більш читабельним і зрозумілим, оскільки стилі прописуються безпосередньо в HTML;

– це спрощує процес обслуговування і зменшує кількість повторюваного коду.

Висновки до другого розділу

В другому розділі кваліфікаційної роботи бакалавра було оглянуто та проведено всі штатні та технічні програмні засоби. Детально надано опис вище зазначених технологій, які були використані при розробці вебзастосунку для підвищення продуктивності за рахунок трекінгу звичок. Зокрема, було розглянуто мови програмування, систему управління базою даних, інструмент для контролю версій та бібліотеки для реалізації функцій застосунку.

Вибір стеку технологій відіграє критичну роль у процесі розробки програмного забезпечення. Правильно підібрані технології дозволяють оптимізувати роботу команди розробників, забезпечують стабільність та продуктивність кінцевого продукту, а також спрощують подальшу підтримку та масштабування. У нашому випадку, використання TypeScript як основної мови програмування дало можливість забезпечити строгий контроль типів, що значно зменшує кількість помилок та покращує якість коду.

Таким чином, обрані технології забезпечили ефективний та зручний процес розробки вебзастосунку для підвищення продуктивності за рахунок трекінгу звичок. Вони дозволили створити стабільний, продуктивний та легко підтримуваний продукт, який відповідає всім сучасним вимогам веброзробки. Це демонструє важливість та вплив правильного вибору технологічного стеку на успішність проекту в цілому.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ ТА ПРОВЕДЕННЯ ТЕСТУВАННЯ

3.1 Налаштування середовища проекту

Для того, щоб створити Next.js проєкт створена окрема папка, в якій за допомогою, вбудованої консолі було створено проєкт (рис. 3.1).

```
PS D:\Learning\4_YearOfUniversity\Diplomas\habit-tracker> npx create-next-app@latest habit-tracker
```

Рисунок 3.1 – Створення проєкту

Ця команда використовує create-next-app, інструмент інтерфейсу командного рядка (CLI), який створить для вас додаток Next.js.

Проєкт має таку структуру папок (рис. 3.2):

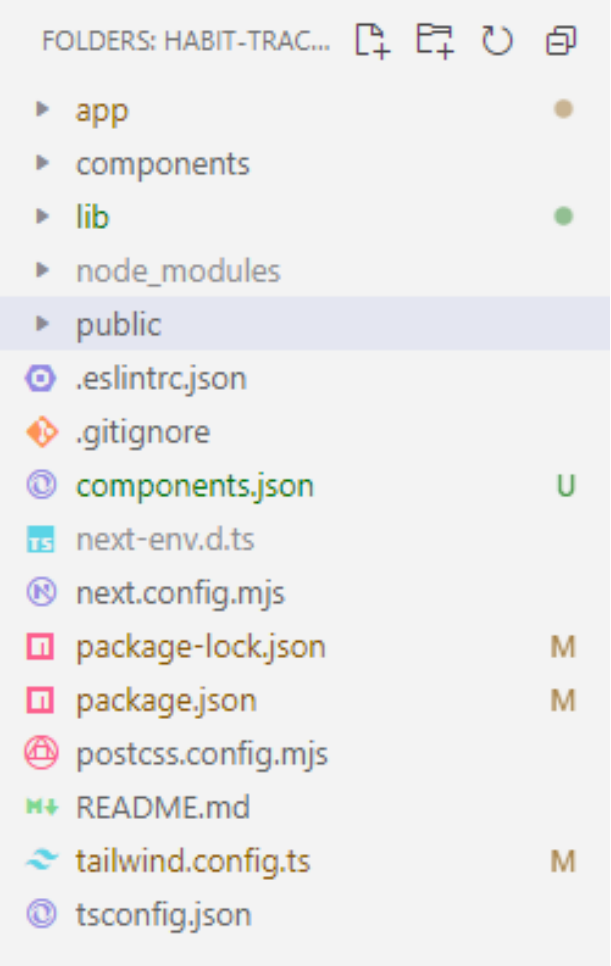

Рисунок 3.2 – Структура папок проєкту

Структура папок проекту дашборду, що показує основні папки та файли app, public та config:

– /app: Містить всі маршрути, компоненти та логіку для вашого додатку, саме звідси ви будете працювати в основному;

– /app/lib: Містить функції, що використовуються у вашій програмі, такі як утиліти багаторазового використання та функції збору даних;

– /app/ui: Містить всі компоненти інтерфейсу користувача для вашого додатку, такі як картки, таблиці та форми. Щоб заощадити час, ми попередньо оформили ці компоненти для вас;

– /public: Містить усі статичні ресурси вашого додатку, такі як зображення;

– /scripts: Містить скрипт заповнення, який ви будете використовувати для заповнення бази даних у наступному розділі;

– файли конфігурації: Ви також помітите конфігураційні файли, такі як next.config.js, в корені вашого додатку. Більшість з цих файлів створюються і попередньо налаштовуються, коли ви починаєте новий проект за допомогою create-next-app.

Для комфортного провадження компонентів замість створення кожної кнопки, менюшки, основних позначок було завантажено бібліотеку і використано shadcn/ui (див. рис. 3.3). За її допомогою стає можливо сворювати красиво оформлені компоненти, які ви можете копіювати та вставляти у свої програми. Вони є доступними, їх частина кода знаходиться в проекті, яку також дуже легко налаштувати чи доповнювати за потребую.

```
PS D:\Learning\4_YearOfUniversity\Diplomas\habit-tracker> npx shadcn-ui@latest init
Need to install the following packages:
shadcn-ui@0.8.0
Ok to proceed? (y) y
√ Which style would you like to use? » Default
√ Which color would you like to use as base color? » Neutral
√ Would you like to use CSS variables for colors? ... no / yes

✓ Writing components.json...
✓ Initializing project...
✓ Installing dependencies...

Success! Project initialization completed. You may now add components.
```

Рисунок 3.3 – Завантаження бібліотеки багаторазових компонентів

Shadcn/ui - це колекція багаторазових компонентів, які ви можете копіювати та вставляти у свої програми (рис. 3.3).

Я маю на увазі, що ви не встановлюєте його як залежність. Він не доступний і не поширюється через npm.

Виберіть потрібні вам компоненти. Скопіюйте та вставте код у свій проект і налаштуйте його відповідно до ваших потреб. Код належить вам.

Використовуйте його як зразок для створення власних бібліотек компонентів.

Ідея полягає в тому, щоб надати вам право власності та контролю над кодом, дозволяючи вам вирішувати, як будувати та стилізувати компоненти.

Почніть з деяких розумних значень за замовчуванням, а потім налаштуйте компоненти відповідно до ваших потреб.

Одним з недоліків пакування компонентів в npm-пакунок є те, що стиль поєднується з реалізацією. Дизайн ваших компонентів має бути відокремлений від їхньої реалізації.

Для більш зручної розробки звісно ж було використано ситстему контролю версій Git, а саме її функціонал гілок (branches), які надають змогу використовувати й працювати над декількома різними варіантами версій в один проміжок часу.

Основною гілкою є master або ж main. Це основна гілка, яка містить стабільний код. Часто використовується для випуску релізів. Вона є створеную за замовченням, ви маєте можливість переймінувати її, але на це немає потреби й не рекомендується.

Основною для розробки ж є гілка develop. Зазвичай використовується для інтеграції нових функцій перед їх додаванням до основної гілки. Ця гілка може бути менш стабільною і буде основною для розробки нашого вебзастосунку, також при розробці великого масштабу проєктів створюються додаткові, окреми гілки, такі, як:

– feature branches: Гілки для розробки нових функцій. Називаються зазвичай за функцією, над якою ведеться робота (наприклад, feature/login-form);

– bugfix branches: Гілки для виправлення помилок (наприклад, bugfix/fix-login-error);

– release branches: Використовуються для підготовки нового релізу. Наприклад, release/1.0.0.

Але в нашому випадку для економії часу й за відсутністю потреби в поділенні на велику купу гілок проекту використано тільки дві основні гілки для розробки й тестування (рис. 3.4).

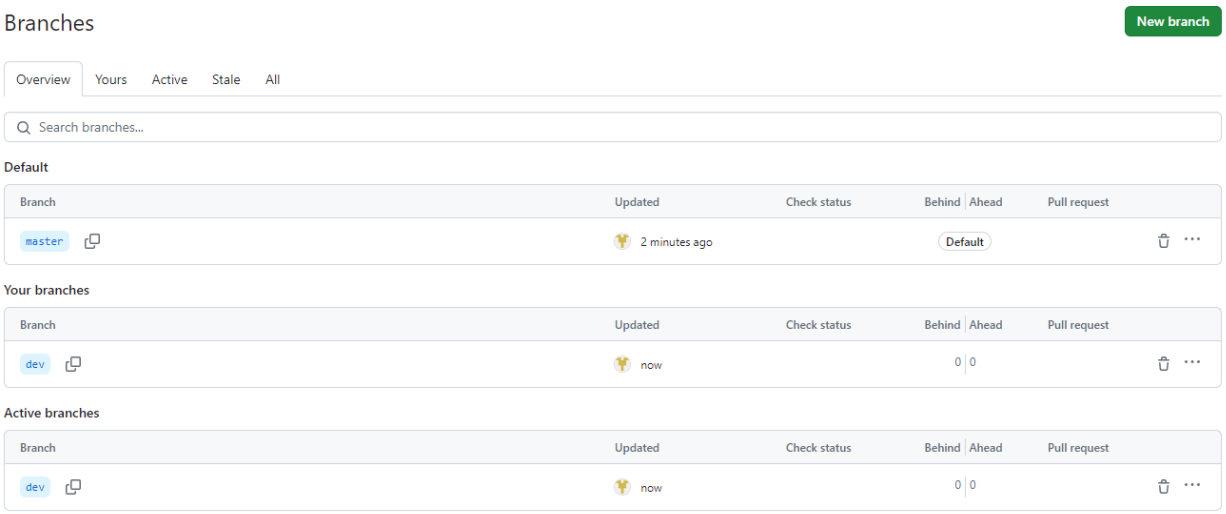

Рисунок 3.4 – Створення основних гілок для розробки й тестування.

Для зручності використання технології tailwind було завантажено додаткове розширення vs code під назвоню Tailwind CSS IntelliSense (рис. 3.5).

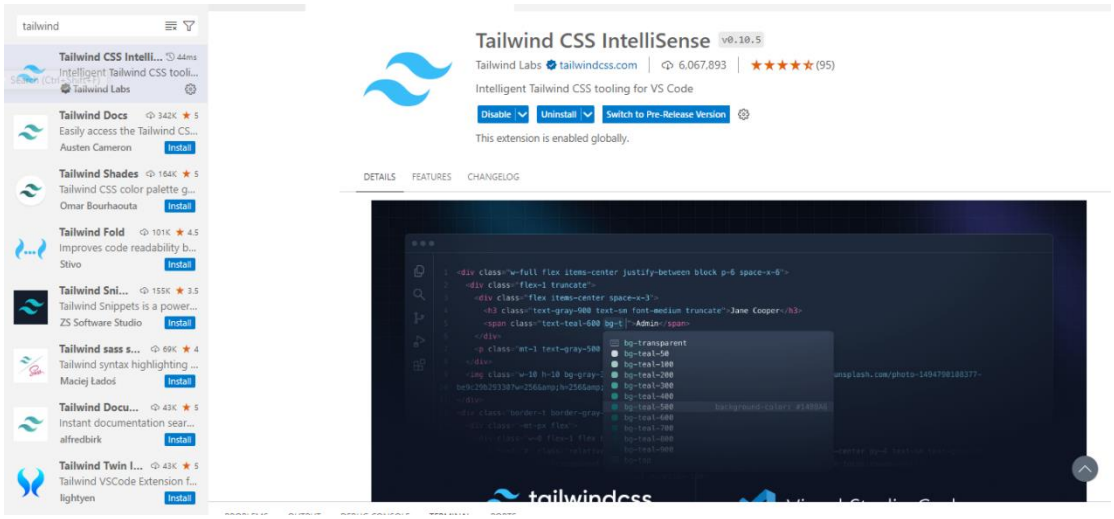

Рисунок 3.5 – Додавання розширення

Tailwind CSS IntelliSense покращує досвід розробки Tailwind, надаючи користувачам Visual Studio Code розширені можливості, такі як автозаповнення, підсвічування синтаксису та лінтування.

Одним з корисних розширень VS Code в нашій реалізації стало Simple React Snippets (рис. 3.6).

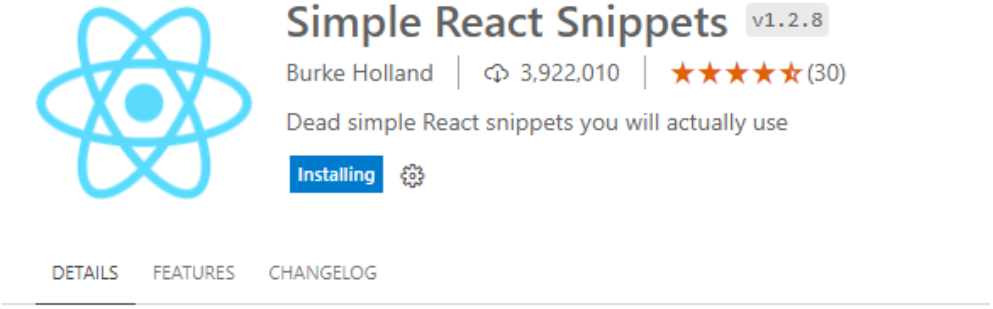

Рисунок 3.6 – Додавання розширення

Сніпет в практиці програмування — невеликий фрагмент початкового коду або тексту, придатний для повторного використання. Сніпети не є заміною процедур, функцій або інших подібних понять структурного програмування. Вони зазвичай використовуються для більш легкого читання коду функцій, які без їх використання виглядають занадто перевантаженими деталями, або для усунення повторення окремої частини ділянки коду (рис. 3.7).

```
app > test > page.tsx > ...
1   const  = () => {
2       return ( );
3   }
4
5   export default ;
```

Рисунок 3.7 – Приклад використання сніпетів

3.2 Темна тема та складності її провадження

Темна тема стала дуже популярною в сучасних веб та мобільних застосунках. Вона не лише знижує навантаження на очі користувачів у темних умовах, але й може зменшити споживання енергії на пристроях з OLED-екранами. Однак, впровадження темної теми в проєкти, написані на React, може бути складним завданням. Цей розділ розглядає основні аспекти та виклики, пов'язані з реалізацією темної теми у React-застосунках. Перемикач тем є важливою частиною застосунку (рис. 3.8).

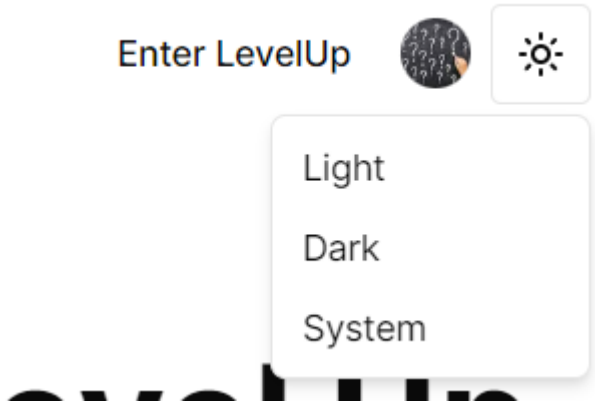

Рисунок 3.8 – Перемикач тем

Одна з головних складнощів - забезпечення сумісності стилів між темною та світлою темами. Це вимагає ретельного планування та тестування, щоб уникнути проблем з контрастністю та читабельністю.

Реалізація плавного та безшовного перемикання між темами може бути викликом, особливо якщо застосунок містить багато динамічних елементів та компонентів.

Також зміна картинок, потрібно мати всі зображення в темному й світлому стилях. Це наглядно видно на (рис. 3.9 та рис 3.10).

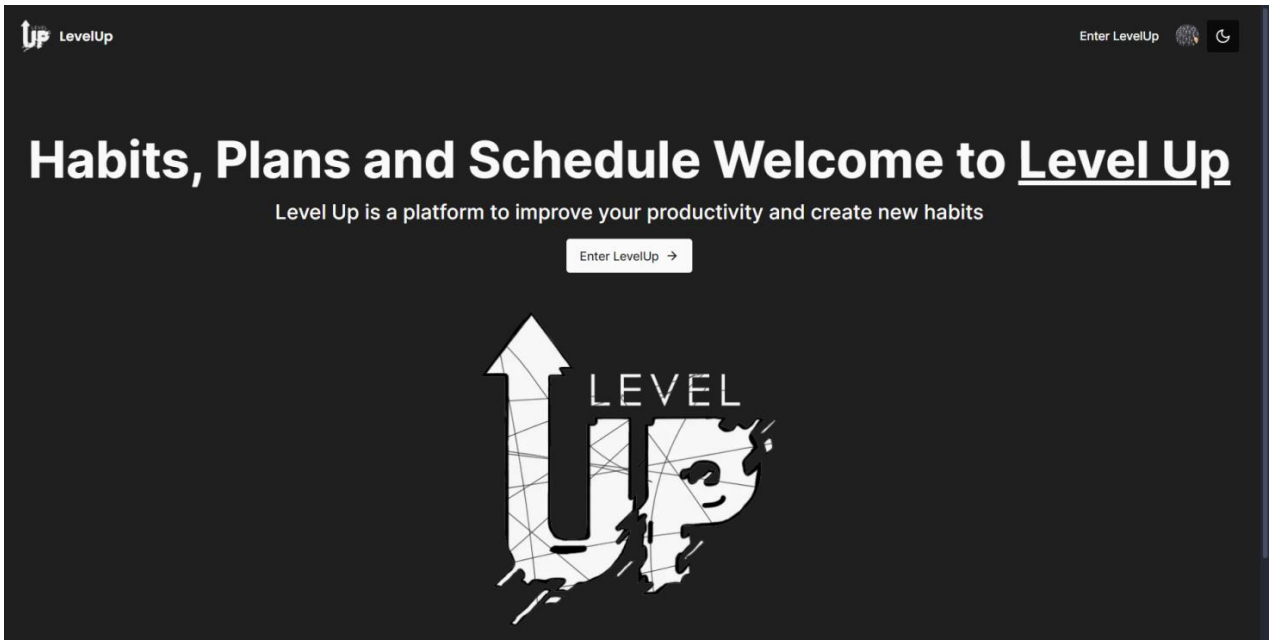

Рисунок 3.9 – Темна тема

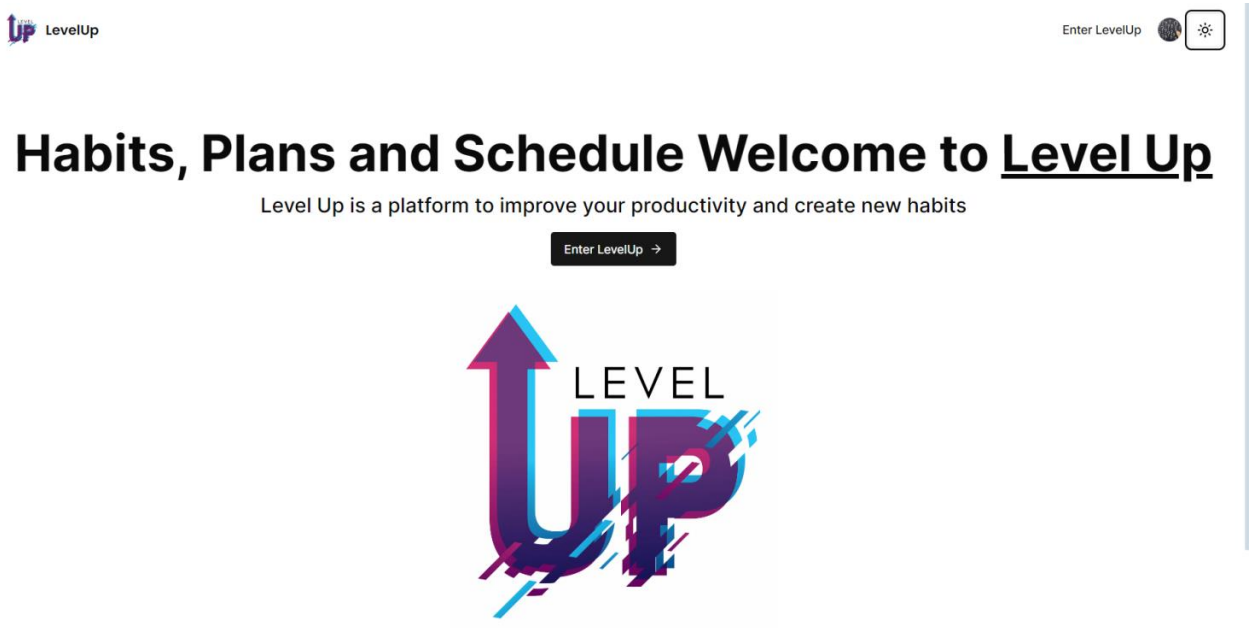

Рисунок 3.10 – Світла тема

Впровадження темної теми у React-застосунках є важливим завданням, яке вимагає ретельного планування та тестування. Використання сучасних підходів, таких

як CSS-змінні, Context API та styled-components, дозволяє значно спростити цей процес. Однак, необхідно враховувати сумісність стилів, забезпечення плавного перемикання тем, збереження стану теми та підтримку сторонніх бібліотек. Завдяки правильному підходу можна створити комфортний для користувачів інтерфейс, який відповідатиме їхнім потребам та вподобанням. Реалізація у коді включає в себе перемикач за допомогою tailwind (рис. 3.11). Дизайн перемикачу (див. рис. 3.12).

```
export const Logo = () => {
  return (
    <div className="hidden md:flex items-center gap-x-2">
      <Image
        src="/logo.svg"
        height="40"
        width="40"
        alt={""}
        className="dark:hidden"
      ></Image>
      <Image
        src="/levelUp-white.png"
        height="40"
        width="40"
        alt={""}
        className="hidden dark:block"
      ></Image>
      <p className={cn("font-semibold", font.className)}>LevelUp</p>
    </div>
```

Рисунок 3.11 – Приклад реалізації у коді

В майбутньому також розглядається варіант розробки додаткових тематичних тем в різних кольорових гамах, або навіть повна зміна кольорів і кастомізація користувачем.

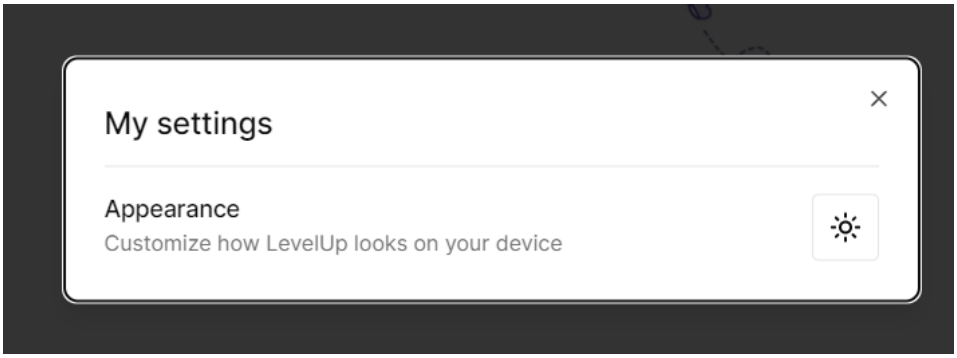

Рисунок 3.12 – Перемикач на сторінці звичок

Крім цього, перемикач теми повинен бути на кожній сторінці для того, щоб користувач не повинен був поверататись на головну сторінку й не мав проблеми зі зміною теми при користуванні вебзастосунка.

3.3 Реалізації аутентифікації

Аутентифікація - це процес підтвердження ідентичності користувача, який намагається отримати доступ до системи, сервісу або ресурсу. У сучасному цифровому світі, де інформація зберігається та обмінюється через мережу Інтернет, захист доступу до цієї інформації стає критично важливим аспектом. Мінімалістичний дизайн з імортуванням підходящих іконок (рис. 3.13).

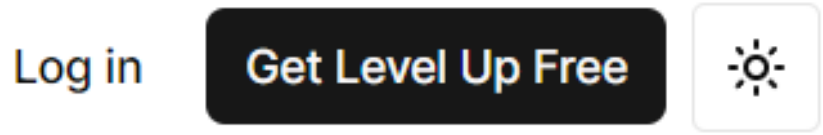

Рисунок 3.13 – Вигляд навігації на сторінці до аутентифікації

Аутентифікація забезпечує впевненість у тому, що тільки правомірні користувачі мають доступ до конфіденційних даних, особистих аккаунтів чи інших обмежених ресурсів.

Важливість аутентифікації полягає в тому, що вона допомагає захистити інформацію від несанкціонованого доступу, зловживання та крадіжки. Вона дозволяє користувачам впевнено користуватися сервісами та ресурсами в Інтернеті, знаючи, що їхні дані захищені від несанкціонованого використання.

Без аутентифікації інформація користувачів може бути легко підвернута ризику зловживання та крадіжки, що може призвести до серйозних наслідків, таких як фінансові втрати, порушення конфіденційності, втрата довіри користувачів та збитки для бізнесу. Вигляд навігації на сторінці аутентифікації включає в себе поля для введення та можливість швидкого входу за допомогою інтеграції з Google та Githgub (див. рис . 3.14).

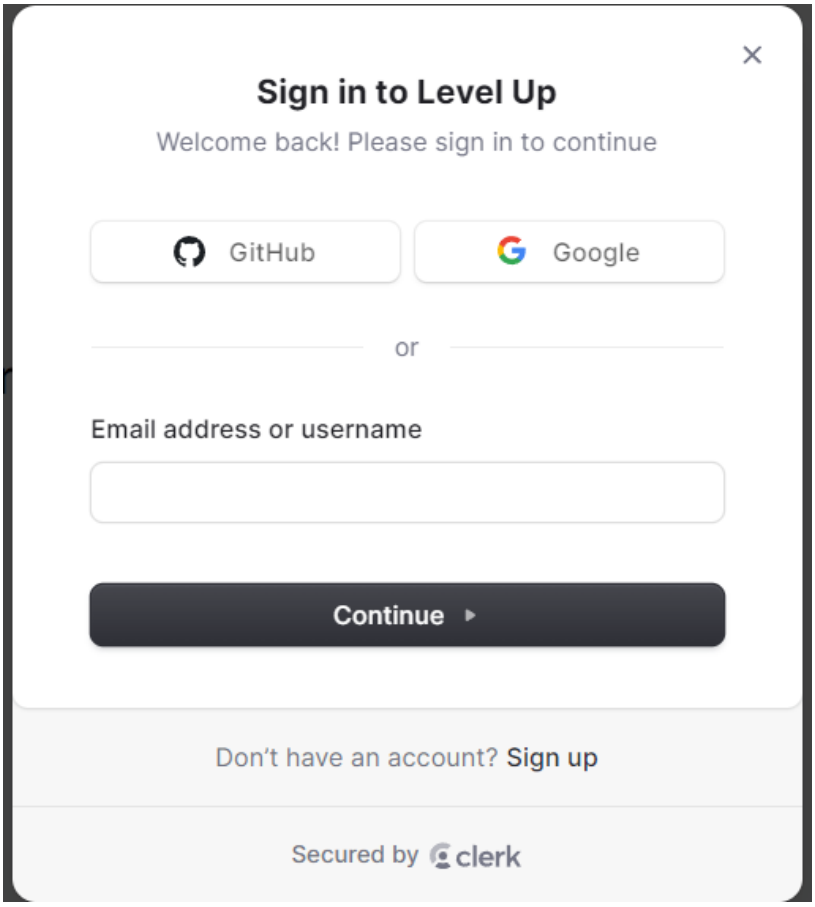

Рисунок 3.14 – Вигляд навігації на сторінці до аутентифікації

Аутентифікація була реалізована за допомогою Clerk. Clerk підтримує кілька стратегій автентифікації, щоб ви могли впровадити стратегію, яка має сенс для ваших користувачів.

Налаштування конфігурації Clerk's впливають на те, як користувачі вашого додатку можуть реєструватися і входити в систему, а також на те, які властивості можна редагувати в їхньому профілі користувача. Ви також можете керувати сеансами користувачів, контролювати, хто отримує доступ до вашого додатку, і налаштовувати електронні та SMS-повідомлення, які надсилаються Clerk під час потоків автентифікації. Всі ці налаштування можна знайти в розділі «Користувачі та автентифікація» на інформаційній панелі Clerk.

Після завантаження за допомогою npm (рис. 3.15) наступним кроком стане додавання змінних оточення у env. файлі для збереження конфеданційності.

```
PS D:\Learning\4_YearOfUniversity\Diplomas\habit-tracker> npm install @clerk/nextjs
```

Рисунок 3.15 – Команда для завантаження Clerk

Далі створемо Convex провайдер в якому використаємо Clerk та його аутентифіцкацію (рис. 3.16).

components > providers > convex-provider.tsx > ...

```tsx
"use client";

import { ReactNode } from "react";
import { ConvexReactClient } from "convex/react";
import { ConvexProviderWithClerk } from "convex/react-clerk";
import { ClerkProvider, useAuth } from "@clerk/clerk-react";

const convex = new ConvexReactClient(process.env.NEXT_PUBLIC_CONVEX_URL!);

export const ConvexClientProvider = ({ children }: { children: ReactNode }) => {
  return (
    <ClerkProvider
      publishableKey={process.env.NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY!}
    >
      <ConvexProviderWithClerk useAuth={useAuth} client={convex}>
        {children}
      </ConvexProviderWithClerk>
    </ClerkProvider>
  );
};
```

Рисунок 3.16 – Використання Clerk при аутентифікації

Після аутентифікації ми отримуємо можливість налаштовувати наш аккаунт. Видалити, змінити іконку профілю чи ім’я, додати пошту або прив’язати інший спосіб для входження до застосунку. (рис. 3.17)

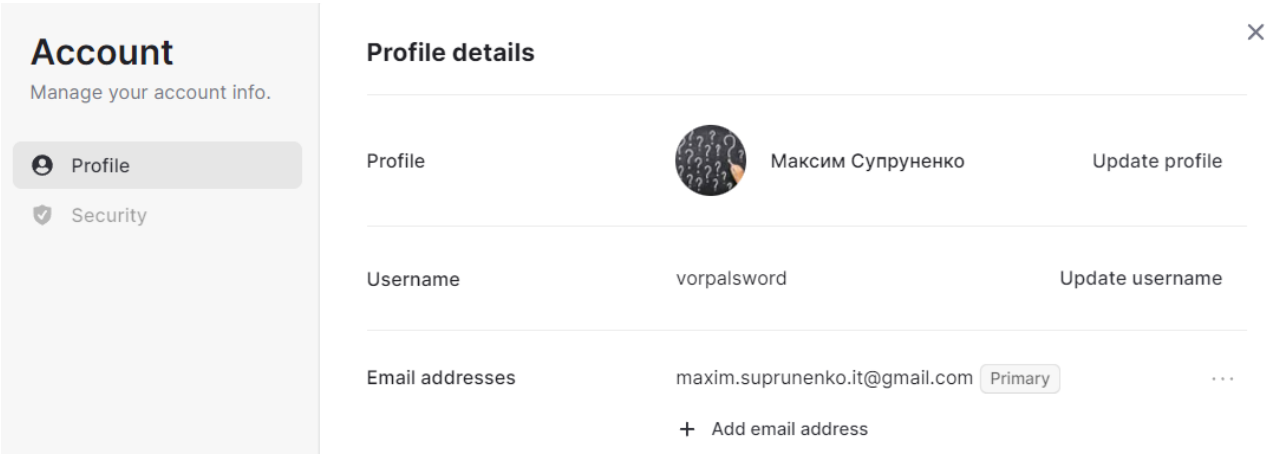

Рисунок 3.17 – Модальне вікно налаштування

3.4 Огляд та підключення БД

Існують різні типи баз даних, кожен з яких має свої переваги і недоліки, і вибір конкретного типу зазвичай залежить від потреб проекту та його характеристик. Ось кілька типів баз даних та їх переваги й недоліки (див. табл. 1.1).

Таблиця 1.1 – Переваги й недоліки різних типів БД

	RDBMS	**NoSQL бази даних**	**Ключ-значення**
Переваги	Стандартний та широко використовуваний тип баз даних	Гнучкіша схема даних, що дозволяє зберігати неструктуровані дані.	Проста модель даних, що дозволяє швидко зберігати та отримувати дані.
	Забезпечує цілісність даних через зв'язки між таблицями.	Висока масштабовність та продуктивність при обробці великих обсягів даних.	Висока масштабовність та продуктивність при роботі з великими обсягами даних.
	Можливість використання мови SQL для запитів та маніпуляції даними.	Підтримка розподіленого збереження даних та горизонтального масштабування.	Підтримка горизонтального масштабування та розподіленого збереження даних.
Недоліки	Схильність до низької масштабовності при великому обсязі даних.	Відсутність стандартів та відкритих інтерфейсів, що може ускладнити розробку та інтеграцію.	Обмежена можливість виконання складних операцій або запитів.
	Потреба в складній структурі таблиць та зв'язків, що може бути важко розробити та підтримувати.	Обмежена підтримка складних операцій, таких як транзакції та JOIN-операції.	Відсутність можливості для взаємодії між даними з різних ключів.
	Відносно повільні операції при великому обсязі даних.	Відносно низька надійність та можливість втрати даних через асинхронні реплікації та розділення.	Недостатня підтримка для аналітики або складних структур даних.

Після аналізу усіх можливих варіантів було вибрано Convex який базується на основі AWS RDS. Relational Database Service (RDS) - це простий в управлінні сервіс реляційних баз даних, оптимізований за сукупною вартістю володіння. Його легко налаштовувати, експлуатувати та масштабувати відповідно до попиту. RDS автоматизує недиференційовані завдання управління базами даних, такі як

забезпечення, конфігурація, резервне копіювання та виправлення. RDS дозволяє клієнтам створювати нову базу даних за лічені хвилини та пропонує гнучкість у налаштуванні баз даних відповідно до їхніх потреб за допомогою 8 механізмів та 2 варіантів розгортання. Структура наведена нижче (рис. 3.18).

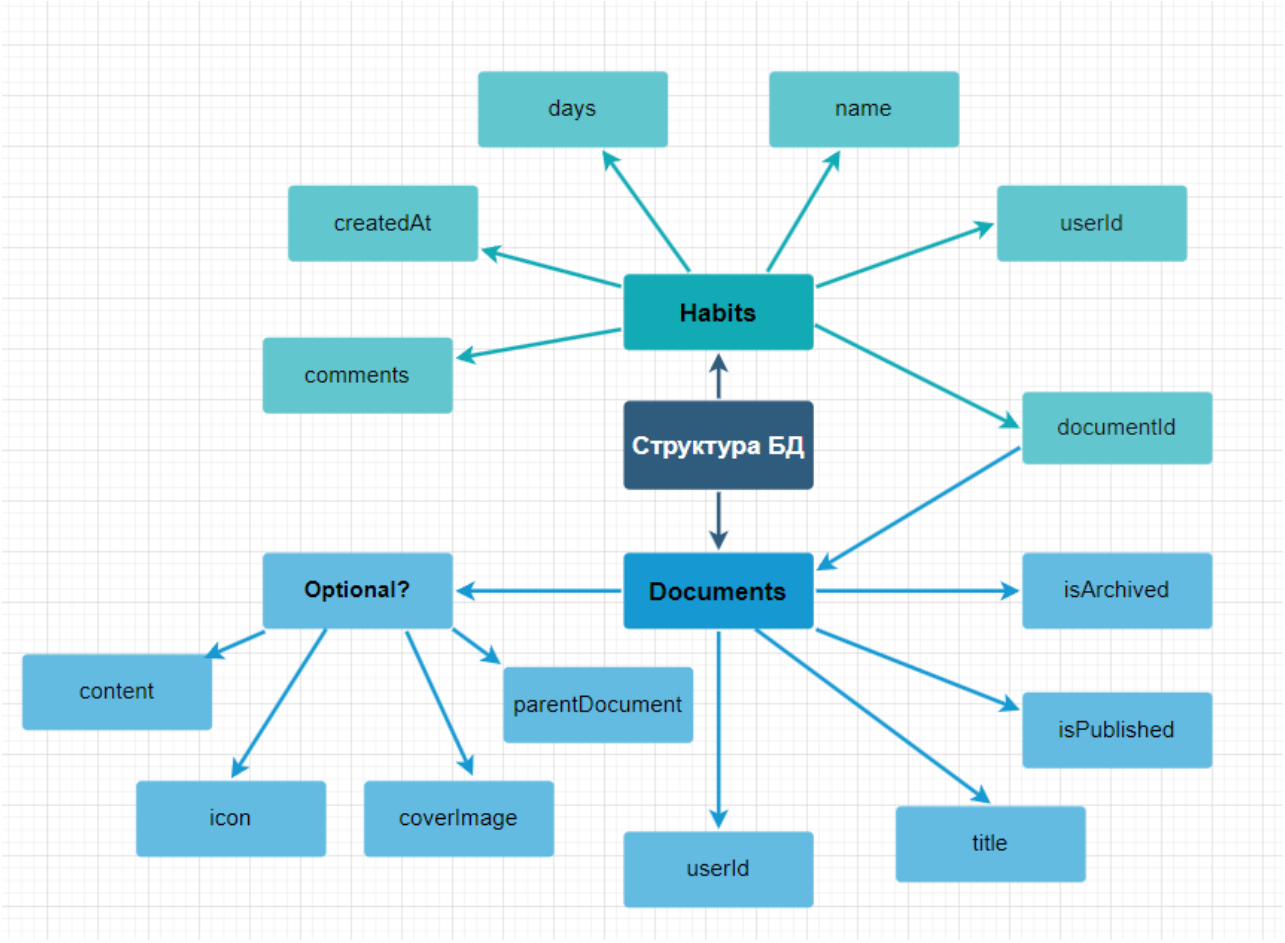

Рисунок 3.18 – Структура БД

Основними є чотири операції CRUD (Create, Read, Update, Delete), які виконуються з даними в базах даних. Вони відображають базові функціональні можливості, необхідні для роботи з будь-якою системою управління базами даних (СУБД) (рис. 3.19).

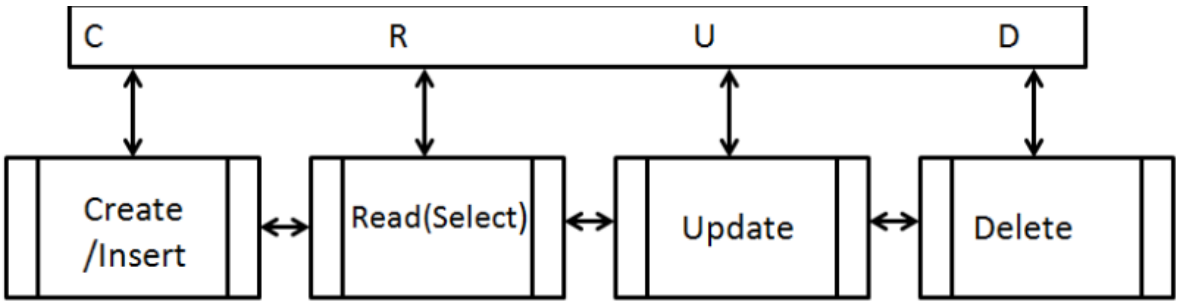

Рисунок 3.19 – Концепт CRUD операцій

Create – це операція створення нових записів в базі даних. Вона використовується для додавання нової інформації в таблиці. Операція дозволяє додавати нові записи в базу даних, зберігаючи нові дані і є важливою для початкового наповнення бази даних та внесення нової інформації у процесі її роботи.

Read – це операція читання даних з бази даних. Вона використовується для отримання інформації з таблиць. Операція забезпечує доступ до збережених даних для аналізу, відображення та використання. Користувачі можуть здійснювати запити до бази даних для отримання потрібної їм інформації.

Update – це операція оновлення існуючих записів в базі даних. Вона використовується для зміни даних у таблицях. Операція дозволяє змінювати існуючі записи, зберігаючи актуальність даних. Можливість виправлення помилок або зміни даних, які вже зберігаються в базі.

Delete – це операція видалення записів з бази даних. Вона використовується для знищення даних з таблиць. Операція дозволяє видаляти застарілі або непотрібні дані, підтримуючи чистоту бази даних. Видалення непотрібних записів звільняє місце для зберігання нових даних.

CRUD-операції є основними елементами роботи з базами даних. Вони забезпечують базову функціональність для створення, читання, оновлення та видалення даних, що є критично важливим для ефективного управління інформацією. Кожна з цих операцій виконує специфічні завдання, необхідні для підтримки цілісності, актуальності та організованості даних в базі даних.

3.5 Вкладені сторінки звичок

Для відслідковування звичок було обрано функціонал створення сторінок (рис. 3.20), але так, як певні звички можуть бути зв’язанами один між одними вирішено додати можливість вкладання їх і створення ієрархії.

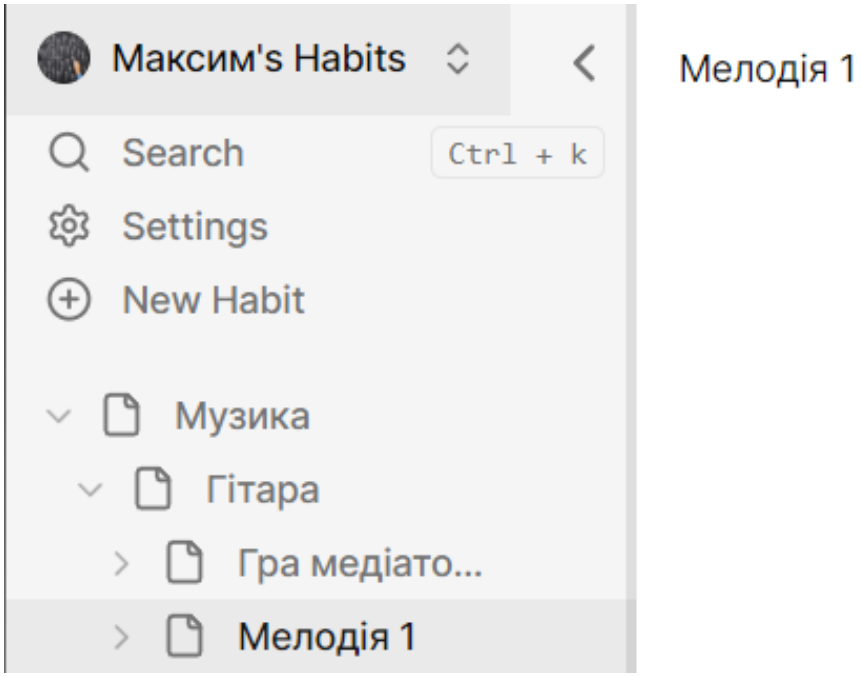

Рисунок 3.20 – Ієрархія створених сторінок звичок

Основної проблемою було вирішення питання видалення й відновлення цієї вкладеної структури. Через складність реалізації було витрачено багато часу на продумку самого алгоритма і принципу відновлення подібних звичок. Проблема була вирішена за допомогою використання рекурсовної архівації (див. рис. 3.21).

```
const recursiveArchive = async (habitId: Id<"habits">) => {
  const children = await ctx.db
    .query("habits")
    .withIndex("by_user_parent", (q) =>
      q.eq("userId", userId).eq("parentHabit", habitId)
    )
    .collect();

  for (const child of children) {
    await ctx.db.patch(child._id, { isArchived: true });

    await recursiveArchive(child._id);
  }
};
```

Рисунок 3.21 – Функція рекурсивної архівації

Рекурсивна архівація - це процес створення резервних копій або архівів, що включає всі підкаталоги та файли в заданій директорії, тобто виконується рекурсивно для всіх вкладених структур.

Це дозволяє зберігати повну структуру даних, включаючи всі підкаталоги та файли, що в них містяться.

Рекурсивна архівація забезпечує створення повної копії директорії з усіма її підкаталогами та файлами, що гарантує збереження всієї структури даних. Процес рекурсивної архівації може бути автоматизований, що зменшує ймовірність людських помилок і забезпечує регулярне резервне копіювання даних.

При відновленні даних з рекурсивного архіву, структура файлів і каталогів буде відновлена в точності до початкового стану, що важливо для збереження цілісності даних.

Переваги рекурсивної архівації:

– повне резервне копіювання;
– автоматизація процесу;
– відновлення повної структури.

Рекурсивна архівація є важливою технікою для забезпечення повного резервного копіювання даних. Вона гарантує, що всі файли та підкаталоги в обраній директорії будуть збережені, що є критично важливим для відновлення даних у випадку їх втрати. Автоматизація цього процесу підвищує ефективність і надійність архівації, знижуючи ризики втрат важливої інформації. У сайдбарі відображення наведено за рахунок використання map, який проходиться по БД й відправляє всі звички з правильною ієрархією (рис. 3.22).

```
<>
  <p
    style={{ paddingLeft: level ? `${level * 12 + 25}px` : undefined }}
    className={cn(
      "hidden text-sm font-medium text-muted-foreground/80",
      expanded && "last:block",
      level === 0 && "hidden"
    )}
  >
    No habits inside
  </p>
  {habits.map((habit) => (
    <div key={habit._id}>
      <Item
        id={habit._id}
        onClick={() => onRedirect(habit._id)}
        label={habit.title}
        icon={FileIcon}
        habitIcon={habit.icon}
        active={params.habitId === habit._id}
        level={level}
        onExpand={() => onExpand(habit._id)}
        expanded={expanded[habit._id]}
      ></Item>
      {expanded[habit._id] && (
        <HabitList parentHabitId={habit._id} level={level + 1}></HabitList>
      )}
    </div>
  ))}
</>
```

Рисунок 3.22 – Відображення сторінок звичок у сайдбарі

3.5 Архівація, відновлення, видалення

Архівація, відновлення та видалення даних є важливими аспектами управління даними в будь-якій організації. Кожен з цих процесів має своє значення та функції, що забезпечують ефективність, безпеку та відповідність нормативним вимогам.

Архівація - це процес переміщення даних, які не потрібні для поточного використання, до спеціального місця зберігання для зменшення навантаження на основні системи та звільнення місця для активних даних.

В нашому проекті будь-яку окрему сторінку можнливо архівувати, щоб мати можливість її відновити у майбутньому (рис. 3.23).

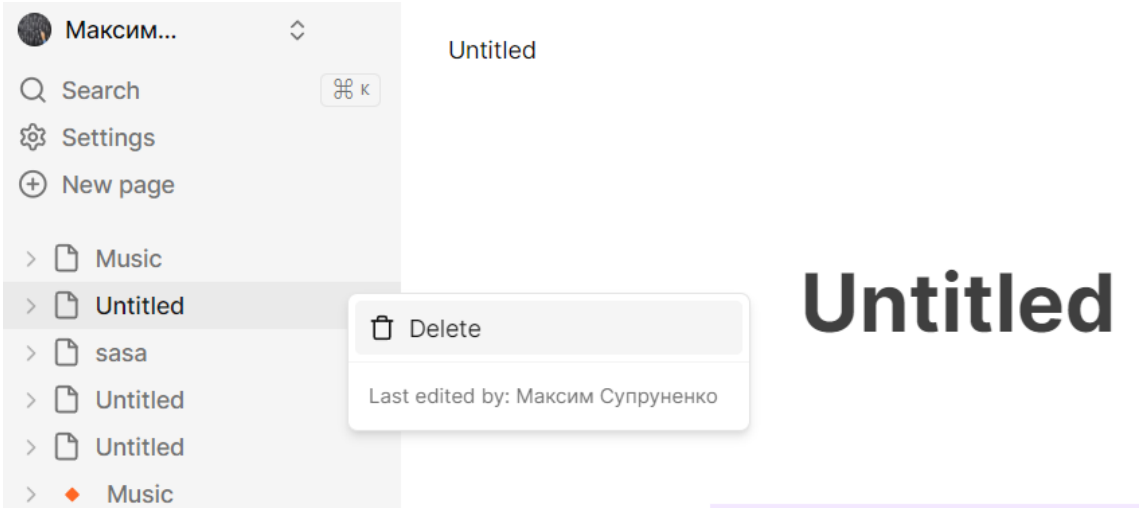

Рисунок 3.23 – Архівація сторінки звичок

Важливість архівації:

– оптимізація продуктивності, архівація зменшує обсяг активних даних, що підвищує швидкість доступу до потрібної інформації;

– економія ресурсів, бо зберігання великих обсягів даних може бути дорогим. Архівація дозволяє зменшити витрати на зберігання;

– забезпечення відповідності через те, що багато галузей мають нормативні вимоги щодо зберігання даних. Архівація допомагає виконувати ці вимоги;

– захист від втрати даних, архіви часто зберігаються в безпечних місцях і можуть служити резервними копіями для відновлення в разі втрати основних даних.

При спробі перейти до архівованої звички, користувачу буде виведено банер про те, що сторінка знаходиться в корзині з двома опціями: видаленням назавжди й відновленням для подальшого користування (рис. 3.24).

```
app > (main) > _components > banner.tsx > Banner
16  export default function Banner({ documentId }: BannerProps) {
44    return (
45      <div className="w-full bg-rose-500 text-center text-sm p-2 text-white flex items-center gap-x-2 justify-center">
46        <p>This page is in the Trash.</p>
47        <Button
48          size="sm"
49          onClick={onRestore}
50          variant="outline"
51          className="border-white bg-transparent hover:bg-primary/5 text-white hover:text-white p-1 px-2 h-auto font-normal"
52        >
53          Restore page
54        </Button>
55        <ConfirmModal onConfirm={onRemove}>
56          <Button
57            size="sm"
58            variant="outline"
59            className="border-white bg-transparent hover:bg-primary/5 text-white hover:text-white p-1 px-2 h-auto font-normal"
60          >
61            Delete forever
62          </Button>
63        </ConfirmModal>
```

Рисунок 3.24 – Код роботи компонента банер

При спробі перейти до архівованої звички, користувачу буде виведено банер про те, що сторінка знаходиться в корзині з двома опціями: видаленням назавжди й відновленням для подальшого користування. Якщо архівування ніби закриває сторінку і переносить її до спеціального резервуару в нашому випадку корзини, то видалення назавжди взаємодіє з бд й видаляє звичку назавжди, без можливості повернення (рис. 3.25).

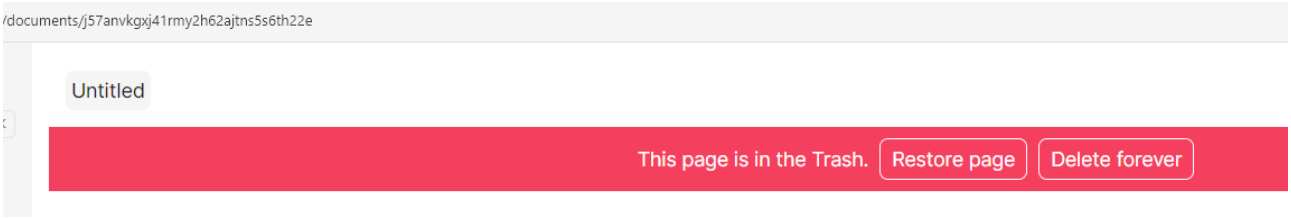

Рисунок 3.25 – Банер з опціями

Відновлення - це процес повернення даних з резервних копій до стану, в якому вони можуть бути використані після втрати або пошкодження оригінальних даних. Для цього було використано модальне вікно з функціоналом, який надає змогу відновляти, видаляти назавжди звички. (рис. 3.26).

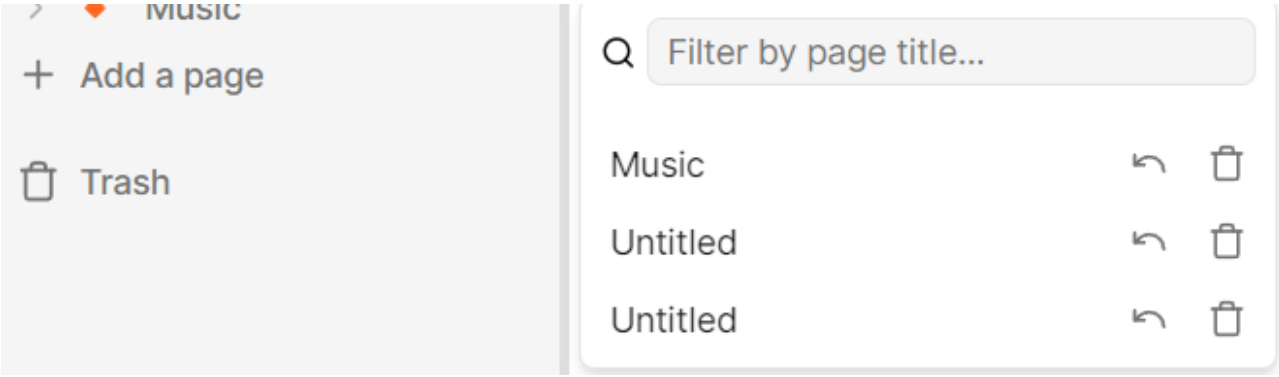

Рисунок 3.26 – Модальне вікно взаємодії з корзиною

Насправді, величезна кількість даних втрачається кожного дня з різних причин, включаючи технічні збої, людські помилки, кібератаки та природні катастрофи. Відновлення даних є критично важливим для мінімізації збитків і швидкого відновлення роботи організації. Тому наявність надійної системи резервного копіювання та відновлення є обов'язковою умовою для будь-якого сучасного бізнесу, що прагне забезпечити безперервність своєї діяльності та захистити свої інформаційні ресурси.

Важливість відновлення:

– забезпечення безперервності бізнесу, ідновлення дозволяє швидко повернутися до нормальної роботи після збоїв, аварій або кібератак;

– захист від втрати даних, системи відновлення даних забезпечують захист від втрати цінної інформації через випадкові видалення або пошкодження;

– виконання нормативних вимог, деякі галузі вимагають збереження копій даних для аудиту та інших цілей. Відновлення забезпечує виконання цих вимог;

– забезпечення довіри клієнтів: швидке і ефективне відновлення даних допомагає підтримувати довіру клієнтів, показуючи, що організація має механізми захисту їхньої інформації.

При виборі відновлення спрацьовує функція, яка приймає id документа й повертає його (рис. 3.27).

```
<span className="truncate pl-2">{document.title}</span>
<div className="flex items-center">
  <div
    onClick={(e) => onRestore(e, document._id)}
    role="button"
    className="rounded-sm p-2 hover:bg-neutral-200 dark:hover:bg-neutral-600"
  >
    <Undo className="h-4 w-4 text-muted-foreground" />
  </div>
  <ConfirmModal onConfirm={() => onRemove(document._id)}>
    <div
      role="button"
      className="rounded-sm p-2 hover:bg-neutral-200 dark:hover:bg-neutral-600"
    >
      <Trash className="h-4 w-4 text-muted-foreground" />
    </div>
  </ConfirmModal>
```

Рисунок 3.27 – Реалізація відновлення та видалення

Якщо казати про видалення, мо можемо термінологізувати це, як процес безповоротного знищення даних, які більше не потрібні або мають бути видалені відповідно до політик безпеки чи нормативних вимог.

Важливість видалення:

– захист конфіденційності, видалення чутливих даних допомагає захистити конфіденційну інформацію від несанкціонованого доступу;

– зменшення ризиків за тією причиною, що видалення старих або непотрібних даних знижує ризики, пов'язані з витоками інформації або кібератакам;

– оптимізація ресурсів, бо знищення застарілих даних звільняє ресурси, що можуть бути використані для зберігання та обробки актуальної інформації.

Постає питання, що робити, якщо користувач випадково натисне на кнопку видалення назавжди? Для запобігання подібної ситуації було прийняте рішення додати модальне вікно, яке уточнить наміри людини.

Модальні вікна є важливою частиною будь-якого веб-застосунку. Вони використовуються для привернення уваги користувачів до важливої інформації або для забезпечення взаємодії з додатковими функціями без покидання основної сторінки.

Використання модальних вікон дозволяє уникнути перевантаження основного інтерфейсу великою кількістю елементів, зберігаючи його простим та зрозумілим для користувачів.

Модальні вікна допомагають забезпечити послідовність роботи, особливо коли необхідно підтвердити важливі дії, наприклад, видалення даних або здійснення покупки. Це зменшує ймовірність випадкових помилок (рис 3.28)

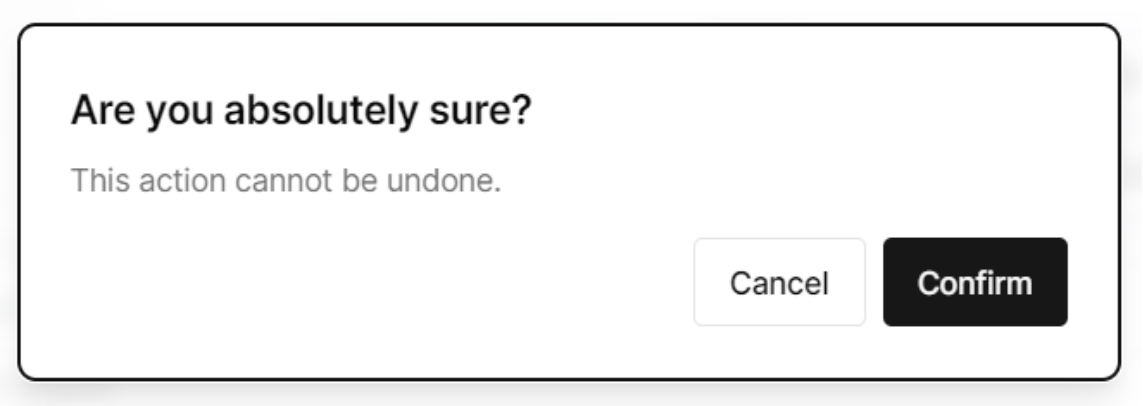

Рисунок 3.28 – Модальне вікно перевірки намірів остаточного видалення

Модальні вікна є важливим інструментом в арсеналі веб-розробників та дизайнерів. Вони забезпечують ефективну взаємодію з користувачами, підвищують зручність використання і допомагають вирішувати конкретні задачі без відволікання від основного контексту. Правильне використання модальних вікон може значно покращити загальний досвід користувача, зробивши взаємодію з додатком чи сайтом більш приємною та інтуїтивно зрозумілою.

Підводячи висновки, архівація, відновлення та видалення даних є ключовими складовими управління даними. Вони допомагають забезпечити ефективність, безпеку та відповідність нормативним вимогам, що в свою чергу підтримує стабільність та розвиток організації.

3.6 Пошук по назві

Скільки разів, ви стикалися з несправним або повільним пошуком, який просто не дає вам змоги отримати необхідну інформацію на запит? Ця проблема доволі поширена, десь просто не має пошуку, а він мав там бути.

Пошук по назві є однією з найважливіших функцій в інформаційних системах, веб-сайтах та додатках. Він дозволяє користувачам швидко знаходити потрібну інформацію, товари чи контент, вводячи ключові слова або назви (рис. 3.29).

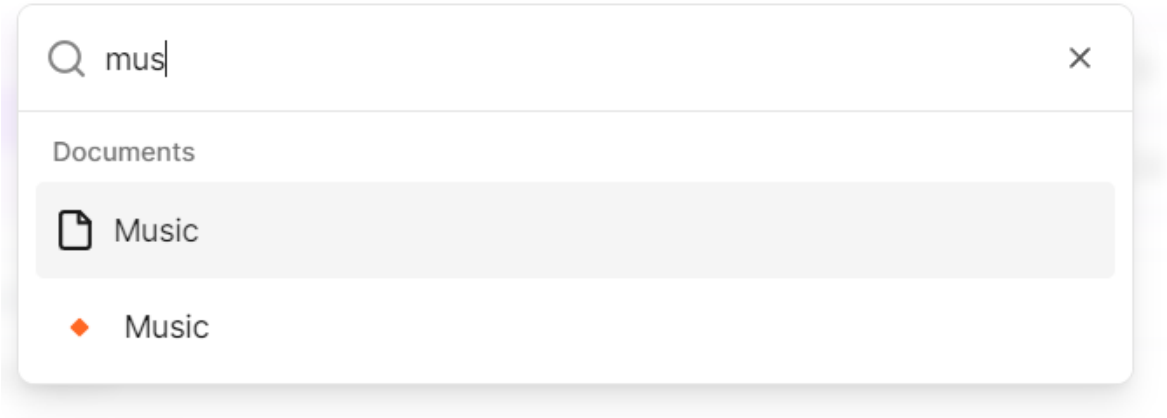

Рисунок 3.29 – Реалізація пошуку сторінок звичок

Важливість цієї функції полягає в наступних аспектах.

1. Підвищення зручності користувачів, які можуть швидко знаходити потрібні товари, статті, документи або інші об'єкти, вводячи їх назви. Це значно покращує користувацький досвід, зменшуючи час, витрачений на пошук.

2. Ефективність навігації. Пошук по назві дозволяє легко орієнтуватися у великій кількості контенту чи продуктів. Це особливо важливо для інтернет-магазинів, бібліотек, архівів та інших систем з великим обсягом даних.

3. Підвищення продуктивності. Можливість швидкого знаходження потрібної інформації або продуктів підвищує загальну продуктивність користувачів, особливо в бізнес-середовищі, де час є критичним фактором.

4. Зменшення навантаження на підтримку. Інтуїтивно зрозумілий та ефективний пошук по назві зменшує кількість запитів до служби підтримки. Користувачі можуть самостійно знаходити необхідну інформацію без допомоги технічної підтримки.

5. Покращення взаємодії з клієнтами. У випадку інтернет-магазинів, пошук по назві може збільшити продажі, забезпечуючи клієнтам легкий доступ до продуктів, які вони шукають. Це підвищує задоволеність клієнтів та їхню лояльність.

6. Аналіз поведінки користувачів. Дані, зібрані під час використання пошуку, можуть бути використані для аналізу поведінки користувачів. Це дозволяє компаніям краще розуміти потреби своїх клієнтів і вдосконалювати свої продукти та послуги.

7. Покращення SEO. Пошук по назві допомагає покращити видимість контенту в пошукових системах, таких як Google. Це може збільшити трафік на сайт та залучити нових користувачів або клієнтів.

Пошук по назві є критично важливою функцією для будь-якої інформаційної системи, сайту чи додатку. Він забезпечує зручність користувачів, підвищує ефективність навігації, продуктивність та покращує взаємодію з клієнтами. Інтуїтивно зрозумілий та швидкий пошук допомагає користувачам знаходити потрібну інформацію, зменшуючи навантаження на підтримку та підвищуючи загальний рівень задоволеності користувачів.

3.7 Стилізація

Стилізація даних — це процес оформлення та представлення даних у зрозумілій і візуально привабливій формі. Вона включає використання різних методів і інструментів для поліпшення читабельності, сприйняття та ефективності комунікації інформації. Стилізація може застосовуватися до текстових даних, таблиць, графіків, діаграм та інших візуальних елементів. (рис. 3.30 та 3.31).

Існує некінченна купа ідей для покращення стилю, тобто зовнішього вигляду, який є доволі суб’єктивною характеристикою. Наш стиль є доволі мінімалістичним на данному етапі, і як на мене ніяк не можнливо обійстись без емодзі й картинок. Вони доповнюють і надають певної унікальності, а також допомогаються швидше орієнтуватися через асоціаційні зв’язки у нашому мозку.

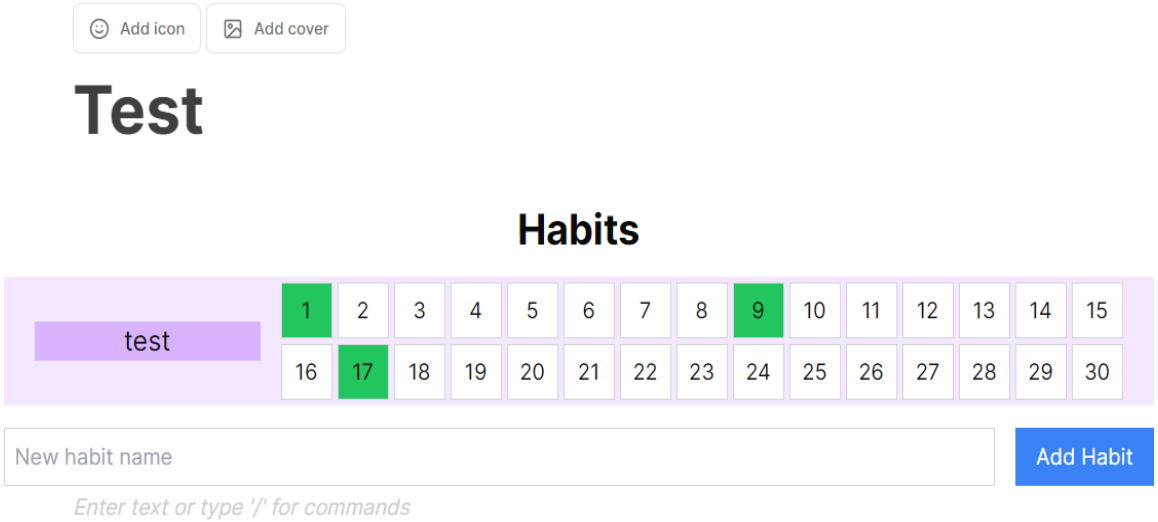

Рисунок 3.30 – Додавання зображення й емодзі по ховеру

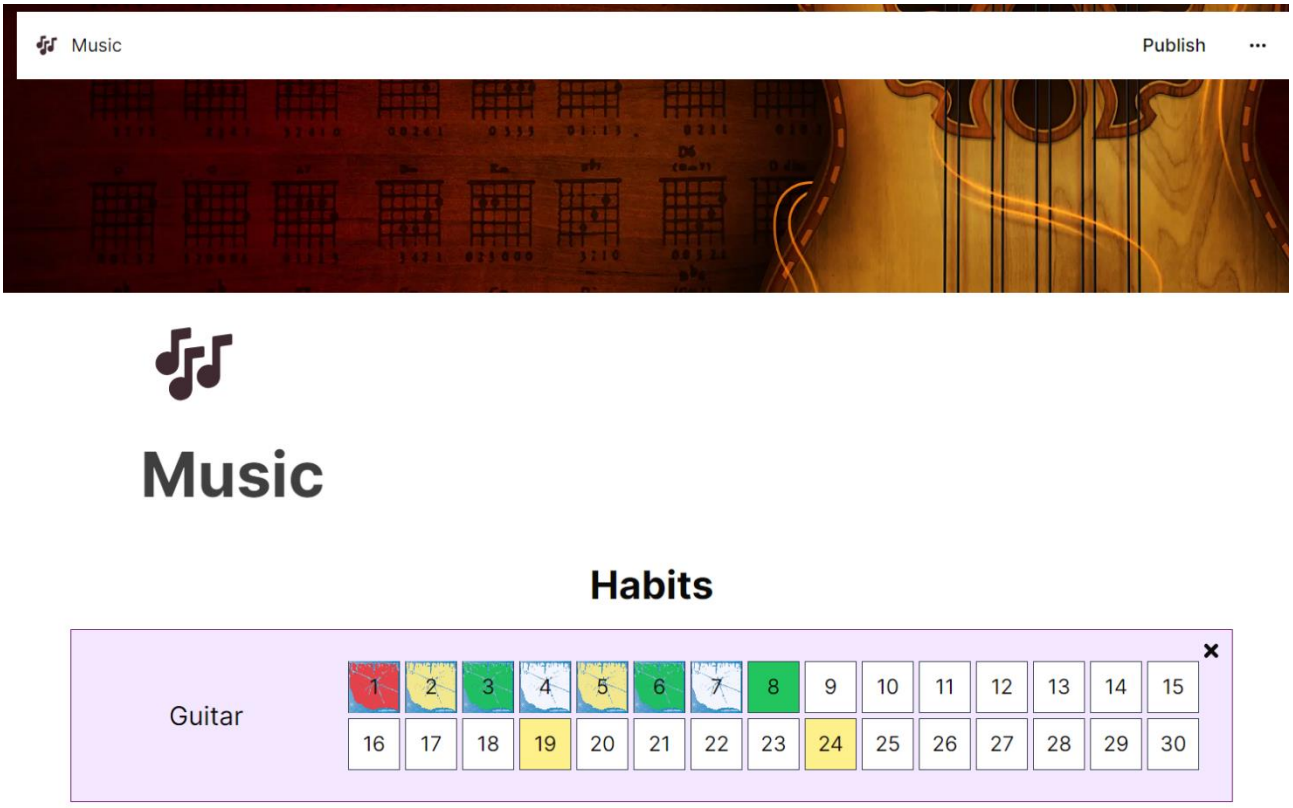

Рисунок 3.31 – Вигляд сторінки після стилізації

Основні аспекти стилізації даних:

– вибір шрифтів. Використання відповідних шрифтів може значно покращити читабельність даних. Наприклад, прості та сучасні шрифти, такі як Arial або Helvetica, можуть бути зручнішими для читання, ніж декоративні або складні шрифти. Використовуйте прості та добре читабельні шрифти. Уникайте надмірно декоративних шрифтів, які можуть ускладнити читання;

– колірні схеми. Використання кольорів для виділення важливих частин даних або для створення візуальних груп може допомогти швидко зрозуміти інформацію. Важливо дотримуватися принципів кольорової теорії, щоб забезпечити гармонійність та уникнути надмірного навантаження на зір. Використовуйте контрастні кольори для виділення важливих елементів. Це допомагає швидко зосередити увагу на ключових частинах даних;

– форматування тексту. Використання заголовків, підзаголовків, маркованих списків, таблиць та інших елементів форматування допомагає структурувати інформацію і зробити її більш зрозумілою;

– візуалізація даних. Графіки, діаграми, інфографіка та інші візуальні елементи дозволяють представляти складну інформацію в більш зрозумілій формі. Вибір відповідного типу візуалізації залежить від природи даних і цілей презентації;

– відповідність стандартам. Використання загальноприйнятих стандартів і стилів у стилізації даних допомагає забезпечити зрозумілість і узгодженість інформації, особливо коли вона представлена широкій аудиторії.

Стилізація даних є критично важливою для ефективної комунікації інформації. Вона допомагає зробити дані зрозумілішими, читабельнішими та візуально привабливішими, що в свою чергу сприяє кращому розумінню та прийняттю рішень на їх основі. Використання відповідних інструментів і методів стилізації дозволяє максимально ефективно представляти інформацію для різних аудиторій.

Висновки до третього розділу

У третьому розділі кваліфікаційної роботи бакалавра було оглянуто різноманітні аспекти налаштування серверного середовища та розглянуто важливі теми, такі як темна та світла теми, реалізація аутентифікації, вибір та огляд баз даних, а також створення вкладених сторінок із віджетами. Також були досліджені питання архівації, відновлення та видалення даних, а також пошук по назві. Завершився розділ розглядом процесу стилізації даних, що є важливим етапом у покращенні зрозумілості та привабливості представлення інформації.

У результаті дослідження цих аспектів було виявлено, що правильне налаштування серверного середовища, реалізація аутентифікації та вибір підходящих баз даних є ключовими для забезпечення надійності та ефективності системи. Крім того, реалізація відновлення та видалення даних, а також пошук по назві дозволяє забезпечити зручний та швидкий доступ до інформації.

Створення вкладених сторінок з віджетами та належна стилізація даних є важливими елементами для забезпечення користувацького зручного інтерфейсу та поліпшення загального враження від використання програмного продукту. Тема світлої та темної тем також відіграє важливу роль у комфортному користуванні системою, надаючи можливість вибору того варіанту, який відповідає особистим вподобанням користувача.

Отже, третій розділ кваліфікаційної роботи бакалавра виявився досить різноманітним і включав в себе багато важливих аспектів розробки програмного забезпечення, що відображає важливість і комплексність подібних досліджень у сучасному інформаційному середовищі.

ВИСНОВОКИ

У кваліфікаційній роботі бакалавра було проведено всебічне дослідження розробки вебзастосунку для підвищення продуктивності за рахунок трекінгу звичок. Огляд здійснювався в трьох розділах, кожен з яких охоплював різні аспекти проєкту.

У першому розділі було проаналізовано існуючі застосунки-аналоги. Було детально розглянуто призначення проєкту й провдено дослідження впливу звичок. Це дозволило визначити основні характеристики, які повинен мати вебзастосунок для ефективного трекінгу звичок.

Другий розділ присвячений детальному огляду штатних та технічних програмних засобів, що використовуються при розробці вебзастосунку. Було розглянуто мови програмування, систему управління базою даних, інструмент для контролю версій та бібліотеки для реалізації функцій застосунку. Особлива увага приділялася вибору стеку технологій, що відіграє критичну роль у процесі розробки програмного забезпечення.

Третій розділ охоплював різноманітні аспекти налаштування серверного середовища та реалізації функціональних можливостей вебзастосунку. Було розглянуто темну та світлу теми, реалізацію аутентифікації, вибір та огляд баз даних, створення вкладених сторінок із віджетами, а також питання архівації, відновлення та видалення даних, пошуку по назві. Створення вкладених сторінок з віджетами та належна стилізація даних сприяють зручному користувацькому інтерфейсу.

Таким чином, дипломна робота охоплює всебічний аналіз, планування та реалізацію вебзастосунку для трекінгу звичок, підвищення продуктивності. Робота демонструє важливість правильного вибору технологічного стеку, налаштування серверного середовища та розробки інтуїтивного інтерфейсу, що разом забезпечують успішність проєкту в сучасному інформаційному середовищі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Clear, James. Atomic Habits: An Easy & Proven Way to Build Good Habits & Break Bad Ones. Avery, 2018. 320 pages.
2. Next.js. URL: https://nextjs.org (дата звернення: 15.05.2024).
3. GitHub docs. URL: https://docs.github.com/en/get-started (дата звернення: 31.03.2024).
4. React learn. URL: https://react.dev/learn (дата звернення: 18.04.2024).
5. Tailwind. URL: https://www.tailwindapp.com (дата звернення: 10.06.2024).
6. Figma. URL: https://www.figma.com (дата звернення: 05.04.2024).
7. Convex. URL: https://www.convex.dev (дата звернення: 22.03.2024).
8. VS Code. URL: https://code.visualstudio.com (дата звернення: 17.03.2024).
9. Moodle. URL: https://moodle3.chmnu.edu.ua (дата звернення: 29.02.2024).
10. Printable Templates. URL: https://freeorganizingprintables.com/habit-tracker-printable/ (дата звернення: 10.02.2024).
11. Clear, James. "The Habits Academy." URL: https://habitsacademy.com (дата звернення: 14.01.2024).
12. Duhigg, Charles. The Power of Habit: Why We Do What We Do in Life and Business. Random House Trade Paperbacks, 2014. 371 pages.
13. Habitica. URL: https://habitica.com (дата звернення: 29.12.2023).
14. Google Firebase. URL: https://firebase.google.com (дата звернення: 21.12.2023).
15. MongoDB. URL: https://www.mongodb.com (дата звернення: 15.12.2023).
16. Stack Overflow. URL: https://stackoverflow.com (дата звернення: 05.12.2023).
17. Heroku. URL: https://www.heroku.com (дата звернення: 27.11.2023).
18. Amazon Web Services. URL: https://aws.amazon.com (дата звернення: 27.03.2024).
19. Node.js. URL: https://nodejs.org (дата звернення: 15.11.2023).

20. Atlassian Jira. URL: https://www.atlassian.com/software/jira (дата звернення: 27.10.2023).

21. Knapp, Jake, et al. Sprint: How to Solve Big Problems and Test New Ideas in Just Five Days. Simon & Schuster, 2016. 288 pages.

22. Cohn, Mike. Succeeding with Agile: Software Development Using Scrum. Addison-Wesley Professional, 2009. 504 pages.

23. Jeff Sutherland, Ken Schwaber. "The Scrum Guide." URL: https://scrumguides.org/scrum-guide.html (дата звернення: 20.10.2023).

24. "The Importance of Tracking Habits for Personal Growth." Psychology Today. URL: https://www.psychologytoday.com/us/blog/the-wise-open-mind/202110/the-importance-tracking-habits-personal-growth (дата звернення: 15.10.2023).

25. "Building a Habit Tracker App: A Step-by-Step Guide." Smashing Magazine. URL: https://www.smashingmagazine.com/2021/08/building-habit-tracker-app/ (дата звернення: 10.10.2023).

ДОДАТОК А

Лістинг коду

page.tsx

```
"use client";
import { useMutation, useQuery } from "convex/react";
import dynamic from "next/dynamic";
import { useMemo, useState } from "react";

import { api } from "@/convex/_generated/api";
import { Id } from "@/convex/_generated/dataModel";
import { Skeleton } from "@/components/ui/skeleton";
import Toolbar from "@/components/toolbar";
import Cover from "@/components/cover";
import ConfirmModal from "@/components/modals/confirm-modal";
import { FaTimes } from "react-icons/fa";

interface Habit {
  _id: Id<"habits">;
  name: string;
  userId: string;
  documentId: Id<"documents">;
  days: string[];
  comments: string[];
  createdAt: string;
}

interface DocumentIdPageProps {
  params: {
    documentId: Id<"documents">;
  };
}

export default function DocumentIdPage({ params }: DocumentIdPageProps) {
  const Editor = useMemo(
    () => dynamic(() => import("@/components/editor"), { ssr: false }),
    []
  );

  const document = useQuery(api.documents.getById, {
    documentId: params.documentId,
  });

  const update = useMutation(api.documents.update);

  const habits = useQuery(api.habits.getHabitsByDocumentId, {
    documentId: params.documentId,
  });

  const updateHabitDays = useMutation(api.habits.updateHabitDays);
  const createHabit = useMutation(api.habits.createHabit);
  const updateHabit = useMutation(api.habits.updateHabit);
  const deleteHabit = useMutation(api.habits.deleteHabit);

  const [newHabitName, setNewHabitName] = useState("");
  const [newHabitDays, setNewHabitDays] = useState(30); // Default to 30 days
```

```
const handleDayClick = (habit: Habit, dayIndex: number) => {
  const currentDate = new Date();
  const habitStartDate = new Date(habit.createdAt);
  const diffDays = Math.floor(
    (currentDate.getTime() - habitStartDate.getTime()) / (1000 * 60 * 60 * 24)
  );

  if (dayIndex <= diffDays) {
    if (dayIndex === diffDays) {
      const updatedDays = [...habit.days];
      updatedDays[dayIndex] =
        updatedDays[dayIndex] === "green" ? "red" : "green";
      updateHabitDays({
        habitId: habit._id,
        days: updatedDays,
        comments: habit.comments,
      });
    }
  } else {
    const updatedComments = [...habit.comments];
    updatedComments[dayIndex] =
      prompt("Enter your comment for this day:") || "";
    updateHabitDays({
      habitId: habit._id,
      days: habit.days,
      comments: updatedComments,
    });
  }
};

const addHabit = () => {
  if (!newHabitName.trim()) {
    alert("Please enter a habit name.");
    return;
  }

  createHabit({
    name: newHabitName,
    documentId: params.documentId,
    days: newHabitDays,
  });
  setNewHabitName("");
  setNewHabitDays(30);
};

const [editedHabitId, setEditedHabitId] = useState<Id<"habits"> | null>(null);
const [editedHabitName, setEditedHabitName] = useState("");

const startEditingHabit = (habit: Habit) => {
  setEditedHabitId(habit._id);
  setEditedHabitName(habit.name);
};

const saveHabitName = (habitId: Id<"habits">) => {
  updateHabit({ habitId, name: editedHabitName });
  setEditedHabitId(null);
  setEditedHabitName("");
};
```

```
const removeHabit = (habitId: Id<"habits">) => {
  deleteHabit({ habitId });
};

if (document === undefined || habits === undefined) {
  return (
    <div>
      <Cover.Skeleton />
      <div className="md:max-w-3xl lg:max-w-4xl mx-auto mt-10">
        <div className="space-y-4 pl-8 pt-4">
          <Skeleton className="h-14 w-[50%]" />
          <Skeleton className="h-4 w-[80%]" />
          <Skeleton className="h-4 w-[40%]" />
          <Skeleton className="h-4 w-[60%]" />
        </div>
      </div>
    </div>
  );
}

if (document === null) {
  return <div>Not found</div>;
}

return (
  <div className="pb-40">
    <Cover url={document.coverImage} />
    <div className="md:max-w-3xl lg:max-w-4xl mx-auto">
      <Toolbar initialData={document} />

      <div className="mt-8">
        <h2 className="text-3xl font-bold mb-4 flex justify-center items-center">
          Habits
        </h2>
        <div className="space-y-4">
          {habits.map((habit: Habit) => (
            <div
              key={habit._id}
              className="relative flex items-center justify-between p-4 bg-purple-100 border border-fuchsia-900"
            >
              <div className="flex items-center space-x-4 ">
                {editedHabitId === habit._id ? (
                  <input
                    className="flex-shrink-0 w-44 p-2 bg-white border border-fuchsia-900"
                    value={editedHabitName}
                    onChange={(e) => setEditedHabitName(e.target.value)}
                    onBlur={() => saveHabitName(habit._id)}
                  />
                ) : (
                  <div
                    className="flex-shrink-0 w-44 items-center flex justify-center  cursor-pointer"
                    onClick={() => startEditingHabit(habit)}
                  >
                    <span className="text-xl break-words w-full items-center text-center ">
                      {habit.name}
                    </span>
                  </div>
                )}
```

```
    <div className="my-1">
      <div className="flex space-x-1 mt-1 ">
        <div className="flex flex-wrap items-center justify-center">
          {habit.days.map((day, index) => (
            <div
              key={index}
              className={`w-10 h-10 mb-1 mr-1 flex items-center justify-center ${
                index <
                Math.floor(
                  (new Date().getTime() -
                    new Date(habit.createdAt).getTime()) /
                    (1000 * 60 * 60 * 24)
                )
                  ? "border-gray-100"
                  : ""
              } ${
                day === "green"
                  ? "bg-green-500"
                  : day === "red"
                  ? "bg-red-500"
                  : habit.comments[index]
                  ? "bg-yellow-200"
                  : "bg-white"
              } border border-gray-700 cursor-pointer`}
              onClick={() => handleDayClick(habit, index)}
            >
              <div
                key={index}
                className={`${
                  index <
                  Math.floor(
                    (new Date().getTime() -
                      new Date(habit.createdAt).getTime()) /
                      (1000 * 60 * 60 * 24)
                  )
                    ? "frozen "
                    : ""
                }`}
              >
                {index + 1}
              </div>
            </div>
          ))}
        </div>
      </div>
    </div>

    <ConfirmModal onConfirm={() => removeHabit(habit._id)}>
      <FaTimes className="absolute top-2 right-2 cursor-pointer" />
    </ConfirmModal>
  </div>
))}
<div className="flex space-x-4">
  <input
    className="flex-1 border border-gray-300 p-2"
    type="text"
    placeholder="New habit name"
    value={newHabitName}
```

```
            onChange={(e) => setNewHabitName(e.target.value)}
          />
          <input
            className="flex-1 border border-gray-300 p-2"
            type="number"
            placeholder="Number of days"
            value={newHabitDays}
            onChange={(e) => setNewHabitDays(Number(e.target.value))}
          />
          <button
            className="px-4 py-2 bg-green-500 text-white"
            onClick={addHabit}
          >
            Add Habit
          </button>
        </div>
      </div>
    </div>
    <Editor
      onChange={(content) => update({ id: params.documentId, content })}
      initialContent={document.content}
    />
  </div>
 </div>
);
}
```

spinner.tsx

```
import { Loader } from 'lucide-react';
import { cva, type VariantProps } from 'class-variance-authority';
import { cn } from '@/lib/utils';

const spinnerVariants = cva('text-muted-foreground animate-spin', {
  variants: {
    size: {
      default: 'h-4 w-4',
      sm: 'h-2 w-2',
      lg: 'h-6 w-6',
      icon: 'h-10 w-10',
    },
  },
  defaultVariants: {
    size: 'default',
  },
});

interface SpinnerProps extends VariantProps<typeof spinnerVariants> {}

export const Spinner = ({ size }: SpinnerProps) => {
  return <Loader className={cn(spinnerVariants({ size }))} />;
};
```

toolbar.tsx

```
'use client';
```

```
import { ElementRef, useRef, useState } from 'react';
import { ImageIcon, Smile, X } from 'lucide-react';
import { useMutation } from 'convex/react';
import TextareaAutosize from 'react-textarea-autosize';

import { useCoverImage } from '@/hooks/use-cover-image';
import { Doc } from '@/convex/_generated/dataModel';
import { Button } from '@/components/ui/button';
import { api } from '@/convex/_generated/api';
import IconPicker from './icon-picker';

interface ToolbarProps {
  initialData: Doc<'documents'>;
  preview?: boolean;
}

export default function Toolbar({ initialData, preview }: ToolbarProps) {
  const inputRef = useRef<ElementRef<'textarea'>>(null);
  const [isEditing, setIsEditing] = useState(false);
  const [value, setValue] = useState(initialData.title);

  const update = useMutation(api.documents.update);
  const removeIcon = useMutation(api.documents.removeIcon);

  const coverImage = useCoverImage();

  const enableInput = () => {
    if (preview) return;

    setIsEditing(true);
    setTimeout(() => {
      setValue(initialData.title);
      inputRef.current?.focus();
    }, 0);
  };

  const disableInput = () => setIsEditing(false);

  const onInput = (value: string) => {
    setValue(value);
    update({
      id: initialData._id,
      title: value || 'Untitled',
    });
  };

  const onKeyDown = (event: React.KeyboardEvent<HTMLTextAreaElement>) => {
    if (event.key === 'Enter') {
      event.preventDefault();
      disableInput();
    }
  };

  const onIconSelect = (icon: string) => {
    update({
      id: initialData._id,
      icon,
    });
```

```
};

const onRemoveIcon = () => {
  removeIcon({
    id: initialData._id,
  });
};

return (
  <div className="pl-[54px] group relative">
    {!!initialData.icon && !preview && (
      <div className="flex items-center gap-x-2 group/icon pt-6">
        <IconPicker onChange={onIconSelect}>
          <p className="text-6xl hover:opacity-75 transition">
            {initialData.icon}
          </p>
        </IconPicker>
        <Button
          onClick={onRemoveIcon}
          className="rounded-full opacity-0 group-hover/icon:opacity-100 transition text-muted-foreground text-xs"
          variant="outline"
          size="icon"
        >
          <X className="h-4 w-4" />
        </Button>
      </div>
    )}
    {!!initialData.icon && preview && (
      <p className="text-6xl pt-6">{initialData.icon}</p>
    )}
    <div className="opacity-0 group-hover:opacity-100 flex items-center gap-x-1 py-4">
      {!initialData.icon && !preview && (
        <IconPicker asChild onChange={onIconSelect}>
          <Button
            className="text-muted-foreground text-xs"
            variant="outline"
            size="sm"
          >
            <Smile className="h-4 w-4 mr-2" />
            Add icon
          </Button>
        </IconPicker>
      )}
      {!initialData.coverImage && !preview && (
        <Button
          onClick={coverImage.onOpen}
          className="text-muted-foreground text-xs"
          variant="outline"
          size="sm"
        >
          <ImageIcon className="h-4 w-4 mr-2" />
          Add cover
        </Button>
      )}
    </div>
    {isEditing && !preview ? (
      <TextareaAutosize
        ref={inputRef}
        onBlur={disableInput}
```

```
        onKeyDown={onKeyDown}
        value={value}
        onChange={(e) => onInput(e.target.value)}
          className="text-5xl bg-transparent font-bold break-words outline-none text-[#3F3F3F] dark:text-[#CFCFCF] resize-
none"
       />
     ) : (
       <div
        onClick={enableInput}
        className="pb-[11.5px] text-5xl font-bold break-words outline-none text-[#3F3F3F] dark:text-[#CFCFCF]"
       >
        {initialData.title}
       </div>
     )}
   </div>
  );
}
```

schema.ts

```
import { defineSchema, defineTable } from "convex/server";
import { v } from "convex/values";

export default defineSchema({
  documents: defineTable({
    title: v.string(),
    userId: v.string(),
    isArchived: v.boolean(),
    parentDocument: v.optional(v.id("documents")),
    content: v.optional(v.string()),
    coverImage: v.optional(v.string()),
    icon: v.optional(v.string()),
    isPublished: v.boolean(),
  })
    .index("by_user", ["userId"])
    .index("by_user_parent", ["userId", "parentDocument"]),

  habits: defineTable({
    name: v.string(),
    userId: v.string(),
    documentId: v.id("documents"),
    days: v.array(v.string()),
    comments: v.array(v.string()),
    createdAt: v.string(),
  })
    .index("by_user", ["userId"])
    .index("by_user_document", ["userId", "documentId"]),
});
```

habits.ts

```
import { v } from "convex/values";
import { mutation, query } from "./_generated/server";
import { Doc, Id } from "./_generated/dataModel";

export const createHabit = mutation({
  args: {
    name: v.string(),
    documentId: v.id("documents"),
    days: v.number(), // Number of days for the habit
```

```
  },
  handler: async (ctx, args) => {
    const identity = await ctx.auth.getUserIdentity();

    if (!identity) {
      throw new Error("Not authenticated");
    }

    const userId = identity.subject;

    const habit = await ctx.db.insert("habits", {
      name: args.name,
      userId,
      documentId: args.documentId,
      days: Array(args.days).fill("white"),
      comments: Array(args.days).fill(""),
      createdAt: new Date().toISOString(),
    });

    return habit;
  },
});

export const getHabitsByDocumentId = query({
  args: { documentId: v.id("documents") },
  handler: async (ctx, args) => {
    const identity = await ctx.auth.getUserIdentity();

    if (!identity) {
      throw new Error("Not authenticated");
    }

    const userId = identity.subject;

    const habits = await ctx.db
      .query("habits")
      .withIndex("by_user_document", (q) =>
        q.eq("userId", userId).eq("documentId", args.documentId)
      )
      .collect();

    return habits;
  },
});

export const updateHabitDays = mutation({
  args: {
    habitId: v.id("habits"),
    days: v.array(v.string()),
    comments: v.array(v.string()),
  },
  handler: async (ctx, args) => {
    const identity = await ctx.auth.getUserIdentity();

    if (!identity) {
      throw new Error("Not authenticated");
    }

    const userId = identity.subject;
```

```
    const existingHabit = await ctx.db.get(args.habitId);

    if (!existingHabit) {
      throw new Error("Not found");
    }

    if (existingHabit.userId !== userId) {
      throw new Error("Unauthorized");
    }

    const habit = await ctx.db.patch(args.habitId, {
      days: args.days,
      comments: args.comments,
    });

    return habit;
  },
});

export const updateHabit = mutation({
  args: { habitId: v.id("habits"), name: v.string() },
  handler: async (ctx, args) => {
    const identity = await ctx.auth.getUserIdentity();

    if (!identity) {
      throw new Error("Not authenticated");
    }

    const userId = identity.subject;
    const existingHabit = await ctx.db.get(args.habitId);

    if (!existingHabit) {
      throw new Error("Not found");
    }

    if (existingHabit.userId !== userId) {
      throw new Error("Unauthorized");
    }

    const habit = await ctx.db.patch(args.habitId, {
      name: args.name,
    });

    return habit;
  },
});

export const deleteHabit = mutation({
  args: { habitId: v.id("habits") },
  handler: async (ctx, args) => {
    const identity = await ctx.auth.getUserIdentity();

    if (!identity) {
      throw new Error("Not authenticated");
    }

    const userId = identity.subject;
    const existingHabit = await ctx.db.get(args.habitId);
```

```
    if (!existingHabit) {
      throw new Error("Not found");
    }

    if (existingHabit.userId !== userId) {
      throw new Error("Unauthorized");
    }

    await ctx.db.delete(args.habitId);
  },
});
```

documents.tsx

```
import { v } from "convex/values";

import { mutation, query } from "./_generated/server";
import { Doc, Id } from "./_generated/dataModel";

export const archive = mutation({
    args: { id: v.id("documents") },
    handler: async (ctx, args) => {
        const identity = await ctx.auth.getUserIdentity();

        if (!identity) {
            throw new Error("Not authenticated");
        }

        const userId = identity.subject;

        const existingDocument = await ctx.db.get(args.id);

        if (!existingDocument) {
            throw new Error("Not found");
        }

        if (existingDocument.userId !== userId) {
            throw new Error("Unauthorized");
        }

        const recursiveArchive = async (documentId: Id<"documents">) => {
            const children = await ctx.db
                .query("documents")
                .withIndex("by_user_parent", (q) => (
                    q
                        .eq("userId", userId)
                        .eq("parentDocument", documentId)
                ))
                .collect();

            /* The code block is recursively archiving child documents. */
            for (const child of children) {
                await ctx.db.patch(child._id, {
                    isArchived: true,
                });

                await recursiveArchive(child._id);
            }
        }
```

```
        const document = await ctx.db.patch(args.id, {
            isArchived: true,
        });

        recursiveArchive(args.id);

        return document;
    }
})

export const getSidebar = query({
    args: {
        parentDocument: v.optional(v.id("documents"))
    },
    handler: async (ctx, args) => {
        const identity = await ctx.auth.getUserIdentity();

        if (!identity) {
            throw new Error("Not authenticated");
        }

        const userId = identity.subject;

        const documents = await ctx.db
            .query("documents")
            .withIndex("by_user_parent", (q) =>
                q
                    .eq("userId", userId)
                    .eq("parentDocument", args.parentDocument)
            )
            .filter((q) =>
                q.eq(q.field("isArchived"), false)
            )
            .order("desc").collect();

        return documents;
    }
})

export const create = mutation({
    args: {
        title: v.string(),
        parentDocument: v.optional(v.id("documents"))
    },
    handler: async (ctx, args) => {
        const identity = await ctx.auth.getUserIdentity();

        if (!identity) {
            throw new Error("Not authenticated");
        }

        const userId = identity.subject;

        const document = await ctx.db.insert("documents", {
            title: args.title,
            parentDocument: args.parentDocument,
            userId,
            isArchived: false,
```

```
        isPublished: false,
      });

      return document;
    }
});

export const getTrash = query({
    handler: async (ctx) => {
      const identity = await ctx.auth.getUserIdentity();

      if (!identity) {
        throw new Error("Not authenticated");
      }

      const userId = identity.subject;

      const documents = await ctx.db
        .query("documents")
        .withIndex("by_user", (q) => q.eq("userId", userId))
        .filter((q) =>
          q.eq(q.field("isArchived"), true),
        )
        .order("desc")
        .collect();

      return documents;
    }
});

export const restore = mutation({
    args: { id: v.id("documents") },
    handler: async (ctx, args) => {
      const identity = await ctx.auth.getUserIdentity();

      if (!identity) {
        throw new Error("Not authenticated");
      }

      const userId = identity.subject;

      const existingDocument = await ctx.db.get(args.id);

      if (!existingDocument) {
        throw new Error("Not found");
      }

      if (existingDocument.userId !== userId) {
        throw new Error("Unauthorized");
      }

      const recursiveRestore = async (documentId: Id<"documents">) => {
        const children = await ctx.db
          .query("documents")
          .withIndex("by_user_parent", (q) => (
            q
              .eq("userId", userId)
              .eq("parentDocument", documentId)
          ))
```

```
            .collect();

        for (const child of children) {
            await ctx.db.patch(child._id, {
                isArchived: false,
            });

            await recursiveRestore(child._id);
        }
    }

    const options: Partial<Doc<"documents">> = {
        isArchived: false,
    };

    if (existingDocument.parentDocument) {
        const parent = await ctx.db.get(existingDocument.parentDocument);
        if (parent?.isArchived) {
            options.parentDocument = undefined;
        }
    }

    const document = await ctx.db.patch(args.id, options);

    recursiveRestore(args.id);

    return document;
  }
})

export const remove = mutation({
  args: { id: v.id("documents") },
  handler: async (ctx, args) => {
    const identity = await ctx.auth.getUserIdentity();

    if (!identity) {
        throw new Error("Not authenticated");
    }

    const userId = identity.subject;

    const existingDocument = await ctx.db.get(args.id);

    if (!existingDocument) {
        throw new Error("Not found");
    }

    if (existingDocument.userId !== userId) {
        throw new Error("Unauthorized");
    }

    const document = await ctx.db.delete(args.id);

    return document;
  }
});

export const getSearch = query({
  handler: async (ctx) => {
```

```
    const identity = await ctx.auth.getUserIdentity();

    if (!identity) {
      throw new Error("Not authenticated");
    }

    const userId = identity.subject;

    const documents = await ctx.db
      .query("documents")
      .withIndex("by_user", (q) => q.eq("userId", userId))
      .filter((q) =>
        q.eq(q.field("isArchived"), false),
      )
      .order("desc")
      .collect()

    return documents;
  }
})

export const getById = query({
  args: { documentId: v.id("documents") },
  handler: async (ctx, args) => {
    const identity = await ctx.auth.getUserIdentity();

    const document = await ctx.db.get(args.documentId);

    if (!document) {
      throw new Error("Not found");
    }

    if (document.isPublished && !document.isArchived) {
      return document;
    }

    if (!identity) {
      throw new Error("Not authenticated");
    }

    const userId = identity.subject;

    if (document.userId !== userId) {
      throw new Error("Unauthorized");
    }

    return document;
  }
});

export const update = mutation({
  args: {
    id: v.id("documents"),
    title: v.optional(v.string()),
    content: v.optional(v.string()),
    coverImage: v.optional(v.string()),
    icon: v.optional(v.string()),
    isPublished: v.optional(v.boolean())
  },
```

```
    handler: async (ctx, args) => {
        const identity = await ctx.auth.getUserIdentity();

        if (!identity) {
            throw new Error("Unauthenticated");
        }

        const userId = identity.subject;

        const { id, ...rest } = args;

        const existingDocument = await ctx.db.get(args.id);

        if (!existingDocument) {
            throw new Error("Not found");
        }

        if (existingDocument.userId !== userId) {
            throw new Error("Unauthorized");
        }

        const document = await ctx.db.patch(args.id, {
            ...rest,
        });

        return document;
    }
})

export const removeIcon = mutation({
    args: { id: v.id("documents") },
    handler: async (ctx, args) => {
        const identity = await ctx.auth.getUserIdentity();

        if (!identity) {
            throw new Error("Unauthenticated");
        }

        const userId = identity.subject;

        const existingDocument = await ctx.db.get(args.id);

        if (!existingDocument) {
            throw new Error("Not found");
        }

        if (existingDocument.userId !== userId) {
            throw new Error("Unauthorized");
        }

        const document = await ctx.db.patch(args.id, {
            icon: undefined
        });

        return document;
    }
});

export const removeCoverImage = mutation({
```

```
    args: { id: v.id("documents") },
    handler: async (ctx, args) => {
      const identity = await ctx.auth.getUserIdentity();

      if (!identity) {
        throw new Error("Unauthenticated");
      }

      const userId = identity.subject;

      const existingDocument = await ctx.db.get(args.id);

      if (!existingDocument) {
        throw new Error("Not found");
      }

      if (existingDocument.userId !== userId) {
        throw new Error("Unauthorized");
      }

      const document = await ctx.db.patch(args.id, {
        coverImage: undefined,
      });

      return document;
    }
});
```

cover.tsx

```
"use client";

import Image from "next/image";
import { ImageIcon, X } from "lucide-react";
import { useMutation } from "convex/react";
import { useParams } from "next/navigation";

import { Skeleton } from "@/components/ui/skeleton";
import { cn } from "@/lib/utils";
import { Button } from "@/components/ui/button";
import { useCoverImage } from "@/hooks/use-cover-image";
import { api } from "@/convex/_generated/api";
import { Id } from "@/convex/_generated/dataModel";
import { useEdgeStore } from "@/lib/edgestore";

interface CoverImageProps {
  url?: string;
  preview?: boolean;
}

export default function Cover({ url, preview }: CoverImageProps) {
  const { edgestore } = useEdgeStore();
  const params = useParams();
  const coverImage = useCoverImage();
  const removeCoverImage = useMutation(api.documents.removeCoverImage);

  const onRemove = async () => {
    if (url) {
      await edgestore.publicFiles.delete({
```

```
      url: url,
    });
  }
  removeCoverImage({
    id: params.documentId as Id<"documents">,
  });
 };

 return (
  <div
   className={cn(
    "relative w-full h-[35vh] group",
    !url && "h-[12vh]",
    url && "bg-muted"
   )}
  >
   {!!url && <Image src={url} fill alt="Cover" className="object-cover" />}
   {url && !preview && (
    <div className="opacity-0 group-hover:opacity-100 absolute bottom-5 right-5 flex items-center gap-x-2">
     <Button
      onClick={() => coverImage.onReplace(url)}
      className="text-muted-foreground text-xs"
      variant="outline"
      size="sm"
     >
      <ImageIcon className="h-4 w-4 mr-2" />
      Change cover
     </Button>
     <Button
      onClick={onRemove}
      className="text-muted-foreground text-xs"
      variant="outline"
      size="sm"
     >
      <X className="h-4 w-4 mr-2" />
      Remove
     </Button>
    </div>
   )}
  </div>
 );
}

Cover.Skeleton = function CoverSkeleton() {
 return <Skeleton className="w-full h-[12vh]" />;
};
```

editor.tsx

```
"use client";

import { useTheme } from "next-themes";
import { BlockNoteEditor, PartialBlock } from "@blocknote/core";
import { BlockNoteView, useBlockNote } from "@blocknote/react";
import "@blocknote/core/style.css";

import { useEdgeStore } from "@/lib/edgestore";

interface EditorProps {
 onChange: (value: string) => void;
```

```
  initialContent?: string;
  editable?: boolean;
}

export default function Editor({
  onChange,
  initialContent,
  editable,
}: EditorProps) {
  const { resolvedTheme } = useTheme();
  const { edgestore } = useEdgeStore();

  const handleUpload = async (file: File) => {
    const response = await edgestore.publicFiles.upload({
      file,
    });

    return response.url;
  };

  const editor: BlockNoteEditor = useBlockNote({
    editable,
    initialContent: initialContent
      ? (JSON.parse(initialContent) as PartialBlock[])
      : undefined,
    onEditorContentChange: (editor) => {
      onChange(JSON.stringify(editor.topLevelBlocks, null, 2));
    },
    uploadFile: handleUpload,
  });

  return (
    <div>
      <BlockNoteView
        editor={editor}
        theme={resolvedTheme === "dark" ? "dark" : "light"}
      />
    </div>
  );
}
```

icon-picker.tsx

```
'use client';

import EmojiPicker, { Theme } from 'emoji-picker-react';
import { useTheme } from 'next-themes';

import {
  Popover,
  PopoverContent,
  PopoverTrigger,
} from '@/components/ui/popover';

interface IconPickerProps {
  onChange: (icon: string) => void;
  children: React.ReactNode;
  asChild?: boolean;
}
```

```
export default function IconPicker({
  onChange,
  children,
  asChild,
}: IconPickerProps) {
  const { resolvedTheme } = useTheme();
  const currentTheme = (resolvedTheme || 'light') as keyof typeof themeMap;

  const themeMap = {
    dark: Theme.DARK,
    light: Theme.LIGHT,
  };

  const theme = themeMap[currentTheme];

  return (
    <Popover>
      <PopoverTrigger asChild={asChild}>{children}</PopoverTrigger>
      <PopoverContent className="p-0 w-full border-none shadow-none">
        <EmojiPicker
          height={350}
          theme={theme}
          onEmojiClick={(data) => onChange(data.emoji)}
        />
      </PopoverContent>
    </Popover>
  );
}
```

mode-toggle.tsx

```
'use client';

import * as React from 'react';
import { Moon, Sun } from 'lucide-react';
import { useTheme } from 'next-themes';

import { Button } from '@/components/ui/button';
import {
  DropdownMenu,
  DropdownMenuContent,
  DropdownMenuItem,
  DropdownMenuTrigger,
} from '@/components/ui/dropdown-menu';

export default function ModeToggle() {
  const { setTheme } = useTheme();

  return (
    <DropdownMenu>
      <DropdownMenuTrigger asChild>
        <Button variant="outline" size="icon">
          <Sun className="h-[1.2rem] w-[1.2rem] rotate-0 scale-100 transition-all dark:-rotate-90 dark:scale-0" />
          <Moon className="absolute h-[1.2rem] w-[1.2rem] rotate-90 scale-0 transition-all dark:rotate-0 dark:scale-100" />
          <span className="sr-only">Toggle theme</span>
        </Button>
      </DropdownMenuTrigger>
      <DropdownMenuContent align="end">
```

```
      <DropdownMenuItem onClick={() => setTheme('light')}>
        Light
      </DropdownMenuItem>
      <DropdownMenuItem onClick={() => setTheme('dark')}>
        Dark
      </DropdownMenuItem>
      <DropdownMenuItem onClick={() => setTheme('system')}>
        System
      </DropdownMenuItem>
    </DropdownMenuContent>
  </DropdownMenu>
 );
}
```

search-command.tsx

```
'use client';

import { useEffect, useState } from 'react';
import { File } from 'lucide-react';
import { useQuery } from 'convex/react';
import { useRouter } from 'next/navigation';
import { useUser } from '@clerk/clerk-react';

import {
 CommandDialog,
 CommandEmpty,
 CommandGroup,
 CommandInput,
 CommandItem,
 CommandList,
} from '@/components/ui/command';
import { useSearch } from '@/hooks/use-search';
import { api } from '@/convex/_generated/api';

export default function SearchCommand() {
 const { user } = useUser();
 const router = useRouter();
 const documents = useQuery(api.documents.getSearch);
 const [isMounted, setIsMounted] = useState(false);

 const toggle = useSearch((store) => store.toggle);
 const isOpen = useSearch((store) => store.isOpen);
 const onClose = useSearch((store) => store.onClose);

 useEffect(() => {
  setIsMounted(true);
 }, []);

 useEffect(() => {
  const down = (e: KeyboardEvent) => {
   if (e.key === 'k' && (e.metaKey || e.ctrlKey)) {
    e.preventDefault();
    toggle();
   }
  };

  document.addEventListener('keydown', down);
  return () => document.removeEventListener('keydown', down);
 }, [toggle]);
```

```
  const onSelect = (id: string) => {
    router.push(`/documents/${id}`);
    onClose();
  };

  if (!isMounted) {
    return null;
  }

  return (
    <CommandDialog open={isOpen} onOpenChange={onClose}>
      <CommandInput placeholder={`Search ${user?.fullName}'s Jotion...`} />
      <CommandList>
        <CommandEmpty>No results found.</CommandEmpty>
        <CommandGroup heading="Documents">
          {documents?.map((document) => (
            <CommandItem
              key={document._id}
              value={`${document._id}-${document.title}`}
              title={document.title}
              onSelect={() => onSelect(document._id)}
            >
              {document.icon ? (
                <p className="mr-2 text-[18px]">{document.icon}</p>
              ) : (
                <File className="mr-2 h-4 w-4" />
              )}
              <span>{document.title}</span>
            </CommandItem>
          ))}
        </CommandGroup>
      </CommandList>
    </CommandDialog>
  );
}
```

single-image-dropzone.tsx

```
'use client';

import { UploadCloudIcon, X } from 'lucide-react';
import * as React from 'react';
import { useDropzone, type DropzoneOptions } from 'react-dropzone';
import { twMerge } from 'tailwind-merge';

import { Spinner } from './spinner';

const variants = {
  base: 'relative rounded-md flex justify-center items-center flex-col cursor-pointer min-h-[150px] min-w-[200px] border border-dashed border-gray-400 dark:border-gray-300 transition-colors duration-200 ease-in-out',
  image:
    'border-0 p-0 min-h-0 min-w-0 relative shadow-md bg-slate-200 dark:bg-slate-900 rounded-md',
  active: 'border-2',
  disabled:
    'bg-gray-200 border-gray-300 cursor-default pointer-events-none bg-opacity-30 dark:bg-gray-700',
  accept: 'border border-blue-500 bg-blue-500 bg-opacity-10',
  reject: 'border border-red-700 bg-red-700 bg-opacity-10',
};
```

```
type InputProps = {
  width?: number;
  height?: number;
  className?: string;
  value?: File | string;
  onChange?: (file?: File) => void | Promise<void>;
  disabled?: boolean;
  dropzoneOptions?: Omit<DropzoneOptions, 'disabled'>;
};

const ERROR_MESSAGES = {
  fileTooLarge(maxSize: number) {
    return `The file is too large. Max size is ${formatFileSize(maxSize)}.`;
  },
  fileInvalidType() {
    return 'Invalid file type.';
  },
  tooManyFiles(maxFiles: number) {
    return `You can only add ${maxFiles} file(s).`;
  },
  fileNotSupported() {
    return 'The file is not supported.';
  },
};

const SingleImageDropzone = React.forwardRef<HTMLInputElement, InputProps>(
  (
    { dropzoneOptions, width, height, value, className, disabled, onChange },
    ref
  ) => {
    const imageUrl = React.useMemo(() => {
      if (typeof value === 'string') {
        // in case a url is passed in, use it to display the image
        return value;
      } else if (value) {
        // in case a file is passed in, create a base64 url to display the image
        return URL.createObjectURL(value);
      }
      return null;
    }, [value]);

    // dropzone configuration
    const {
      getRootProps,
      getInputProps,
      acceptedFiles,
      fileRejections,
      isFocused,
      isDragAccept,
      isDragReject,
    } = useDropzone({
      accept: { 'image/*': [] },
      multiple: false,
      disabled,
      onDrop: (acceptedFiles) => {
        const file = acceptedFiles[0];
        if (file) {
          void onChange?.(file);
        }
```

```
  },
  ...dropzoneOptions,
});

// styling
const dropZoneClassName = React.useMemo(
  () =>
    twMerge(
      variants.base,
      isFocused && variants.active,
      disabled && variants.disabled,
      imageUrl && variants.image,
      (isDragReject ?? fileRejections[0]) && variants.reject,
      isDragAccept && variants.accept,
      className
    ).trim(),
  [
    isFocused,
    imageUrl,
    fileRejections,
    isDragAccept,
    isDragReject,
    disabled,
    className,
  ]
);

// error validation messages
const errorMessage = React.useMemo(() => {
  if (fileRejections[0]) {
    const { errors } = fileRejections[0];
    if (errors[0]?.code === 'file-too-large') {
      return ERROR_MESSAGES.fileTooLarge(dropzoneOptions?.maxSize ?? 0);
    } else if (errors[0]?.code === 'file-invalid-type') {
      return ERROR_MESSAGES.fileInvalidType();
    } else if (errors[0]?.code === 'too-many-files') {
      return ERROR_MESSAGES.tooManyFiles(dropzoneOptions?.maxFiles ?? 0);
    } else {
      return ERROR_MESSAGES.fileNotSupported();
    }
  }
  return undefined;
}, [fileRejections, dropzoneOptions]);

return (
  <div className="relative">
    {disabled && (
      <div className="flex items-center justify-center absolute inset-y-0 h-full w-full bg-background/80 z-50">
        <Spinner />
      </div>
    )}
    <div
      {...getRootProps({
        className: dropZoneClassName,
        style: {
          width,
          height,
        },
      })}
```

```
      >
        {/* Main File Input */}
        <input ref={ref} {...getInputProps()} />

        {imageUrl ? (
          // Image Preview
          <img
            className="h-full w-full rounded-md object-cover"
            src={imageUrl}
            alt={acceptedFiles[0]?.name}
          />
        ) : (
          // Upload Icon
          <div className="flex flex-col items-center justify-center text-xs text-gray-400">
            <UploadCloudIcon className="mb-2 h-7 w-7" />
            <div className="text-gray-400">
              Click or drag to this area to upload
            </div>
          </div>
        )}

        {/* Remove Image Icon */}
        {imageUrl && !disabled && (
          <div
            className="group absolute right-0 top-0 -translate-y-1/4 translate-x-1/4 transform"
            onClick={(e) => {
              e.stopPropagation();
              void onChange?.(undefined);
            }}
          >
            <div className="flex h-5 w-5 items-center justify-center rounded-md border border-solid border-gray-500 bg-white
transition-all duration-300 hover:h-6 hover:w-6 dark:border-gray-400 dark:bg-black">
              <X
                className="text-gray-500 dark:text-gray-400"
                width={16}
                height={16}
              />
            </div>
          </div>
        )}
      </div>

      {/* Error Text */}
      <div className="mt-1 text-xs text-red-500">{errorMessage}</div>
    </div>
  );
 }
);
SingleImageDropzone.displayName = 'SingleImageDropzone';

const Button = React.forwardRef<
 HTMLButtonElement,
 React.ButtonHTMLAttributes<HTMLButtonElement>
>(({ className, ...props }, ref) => {
 return (
   <button
     className={twMerge(
       // base
```

```
        'focus-visible:ring-ring inline-flex cursor-pointer items-center justify-center rounded-md text-sm font-medium transition-colors focus-visible:outline-none focus-visible:ring-1 disabled:pointer-events-none disabled:opacity-50',
        // color
        'border border-gray-400 text-gray-400 shadow hover:bg-gray-100 hover:text-gray-500 dark:border-gray-600 dark:text-gray-100 dark:hover:bg-gray-700',
        // size
        'h-6 rounded-md px-2 text-xs',
        className
      )}
      ref={ref}
      {...props}
    />
  );
});
Button.displayName = 'Button';

function formatFileSize(bytes?: number) {
  if (!bytes) {
    return '0 Bytes';
  }
  bytes = Number(bytes);
  if (bytes === 0) {
    return '0 Bytes';
  }
  const k = 1024;
  const dm = 2;
  const sizes = ['Bytes', 'KB', 'MB', 'GB', 'TB', 'PB', 'EB', 'ZB', 'YB'];
  const i = Math.floor(Math.log(bytes) / Math.log(k));
  return `${parseFloat((bytes / Math.pow(k, i)).toFixed(dm))} ${sizes[i]}`;
}

export { SingleImageDropzone };
```

ДОДАТОК Б

Блок-схема алгоритму роботи застосунку від лиця користувача

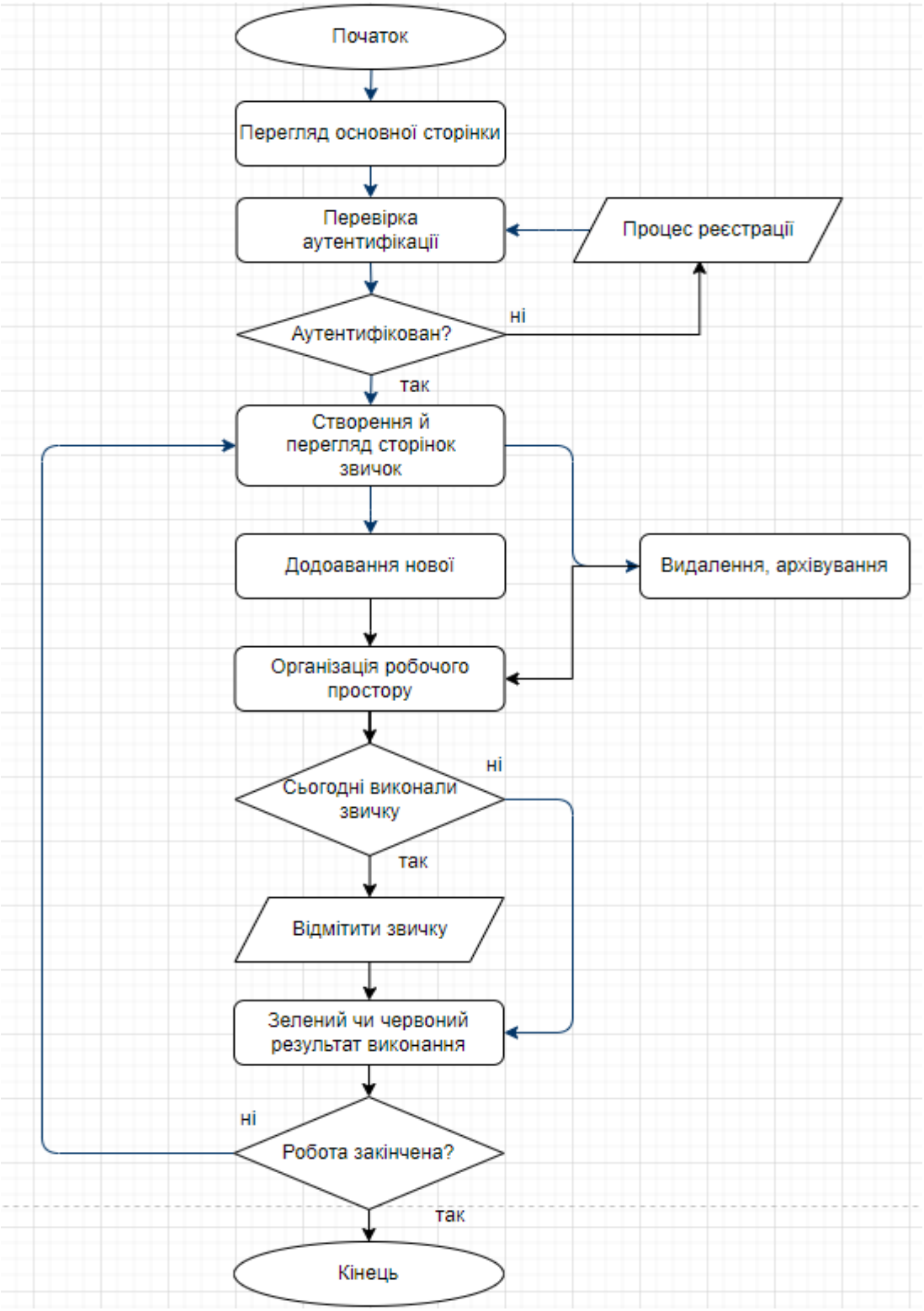

Рисунок Б.1 – Блок-схема