Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота магістра

на тему: «Розробка веб-застосунку нотатку з використанням Incremental Static Regeneration»

Виконав: студент групи К23-1М

Спеціальність 122 Комп'ютерні науки

Пирогов В.І.
(прізвище та ініціали)

Керівник к.т.н., доц. Ульяновська Ю. В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент Університет митної справи та фінансів
(місце роботи)

Доцент кафедри кібербезпеки та інфомармаційних технологій
(посада)

к.т.н., доц. Клим В.Ю.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Пирогов В.І. Розробка веб-застосунку нотатку з використанням Incremental Static Regeneration.

Дипломна робота на здобуття освітнього ступеня магістр за спеціальністю 122 «Комп’ютерні наук». – Університет митної справи та фінансів, Дніпро, 2024.

Об’єктом дослідження є процес управління нотатками у цифровому середовищі.

Предмет дослідження – розробка веб-застосунку для ведення нотаток із використанням технології Incremental Static Regeneration.
Метою роботи є створення веб-застосунку, який забезпечує швидке завантаження контенту, зручність користування та ефективну організацію текстових даних за допомогою технології ISR.

Дана робота присвячена розробці веб-застосунку для управління нотатками, що дозволяє користувачам створювати, редагувати, видаляти та шукати нотатки. У процесі роботи проведено аналіз методів ведення нотаток у цифровому середовищі, обґрунтовано вибір технологічного стеку (Next.js, React.js, Node.js, TypeScript) та розглянуто архітектурні рішення, включаючи реалізацію клієнтської та серверної частин застосунку.
Особливу увагу приділено технології Incremental Static Regeneration, яка забезпечує оновлення контенту без повного перезавантаження сторінки, підвищуючи продуктивність веб-застосунку. Практична цінність роботи полягає у створенні універсального інструменту для управління нотатками, який може бути використаний у навчальних закладах, бізнес-середовищі та для особистих цілей.

Ключові слова: веб-застосунок, нотатки, Incremental Static Regeneration, ISR, Next.js, React.js, Node.js, TypeScript.

ABSTRACT

Pirogov V.I. Development of a Web Application for Note Management Using Incremental Static Regeneration

Diploma thesis (project) for obtaining a master's degree in speciality 122 "Computer Science." - University of Customs and Finance, Dnipro, 2024.

Object of the study: The process of managing notes in a digital environment. Subject of the study: The development of a web application for note management using Incremental Static Regeneration technology. Purpose of the study: To create a web application that ensures fast content loading, ease of use, and effective organization of textual data using ISR technology.

This thesis is dedicated to the development of a web application for managing notes, enabling users to create, edit, delete, and search for notes. The study includes an analysis of note-taking methods in the digital environment, a justification for the chosen technology stack (Next.js, React.js, Node.js, TypeScript), and the consideration of architectural solutions, including the implementation of both the client-side and server-side components of the application. Special attention is given to the Incremental Static Regeneration (ISR) technology, which allows content updates without fully reloading the page, enhancing the application's performance. The practical significance of the work lies in the creation of a universal tool for note management, which can be used in educational institutions, business environments, and for personal purposes.

Keywords: web application, notes, Incremental Static Regeneration, ISR, Next.js, React.js, Node.js, TypeScript.

ЗМІСТ

ВСТУП

Актуальність дослідження. У сучасних умовах розвитку інформаційних технологій важливим аспектом є створення ефективних інструментів для збору, обробки та аналізу даних, що дозволяють приймати обґрунтовані рішення. Ведення нотаток є важливим елементом управління інформацією, особливо в освітній та професійній діяльності. Водночас традиційні підходи мають обмеження у швидкості доступу до даних та їх обробки.

Інноваційність даної роботи полягає у використанні технології Incremental Static Regeneration, що дозволяє поєднувати переваги статичного та динамічного рендерингу веб-сторінок. Технологія забезпечує оновлення контенту без повного перезавантаження сторінок, що сприяє підвищенню продуктивності та зручності користувача.

У сучасну цифрову епоху на процес ведення нотаток дедалі більше впливають нові фактори. Впровадження планшетів, застосунків для нотаток (наприклад, OneNote, Notion, Evernote) та інших освітніх технологій трансформує підходи до фіксації, сприйняття та обробки інформації. Цей технологічний прогрес пропонує нові можливості, як-от зберігання великої кількості даних, зручність у пошуку та спільний доступ до нотаток. Водночас він створює нові виклики:

- Цифрова перевантаженість – надлишок ресурсів може ускладнювати фокусування на ключових ідеях.
- Технічна залежність – потреба в пристроях і доступі до мережі.
- Зниження залученості – дослідження показують, що ручне конспектування може бути ефективнішим для запам'ятовування через залучення моторики [1].

Таким чином, сучасні дослідження спрямовані на пошук збалансованих підходів, що поєднують традиційні методи з цифровими інструментами. Це може включати, наприклад, використання планшетів для рукописних нотаток

або оцифрування рукописних матеріалів для подальшого аналізу та зберігання.

Впровадження цифрових технологій в освітній процес зумовило появу спеціалізованих застосунків для ведення нотаток, які значно трансформували підходи до конспектування. Завдяки широкому функціоналу, таким як підтримка текстових, графічних і мультимедійних записів, ці програми стали важливими інструментами для студентів, дослідників і професіоналів у різних сферах діяльності. Однак ефективність їх використання безпосередньо залежить від функціональних можливостей, когнітивних переваг та потенційних обмежень, які вони створюють у процесі навчання.

Сучасні цифрові застосунки для ведення нотаток орієнтовані на полегшення процесу збору, обробки та зберігання інформації. Їхні ключові функціональні можливості включають:

1. Текстове нотування.
2. Мультимедійна інтеграція.
3. Анотування та виділення ключової інформації.
4. Організація нотаток.
5. Пошук за ключовими словами.
6. Синхронізація та хмарне зберігання.
7. Можливості для співпраці.

Деякі додатки дозволяють декільком користувачам одночасно редагувати документ, що є особливо корисним для групових проєктів та дослідницьких колаборацій.

Мета роботи – розробка веб-застосунку для управління нотатками, який використовує Incremental Static Regeneration для швидкої генерації контенту.

Методи дослідження – методи проектування, розробки програмного забезпечення, метод теорії інформації, обробка та аналіз інформації.

У відповідності до поставленої мети у кваліфікаційній роботі вирішувались наступні завдання:

1. Провести аналіз існуючих методів ведення нотаток;
2. Обґрунтувати вибір стеку технологій для реалізації;
3. Розробити архітектуру веб-застосунку;
4. Реалізувати основний функціонал веб-застосунку із використанням Next.js, React.js, Node.js та TypeScript;
5. Перевірити відповідність програмного продукту функціональним та нефункціональним вимогам.

Об'єкт дослідження – процес управління нотатками у цифровому середовищі.

Предмет дослідження – веб-застосунок для ведення нотаток із застосуванням технології Incremental Static Regeneration.

Практичне значення результатів роботи полягає у створенні ефективного інструменту для управління нотатками, що може бути використаний у навчальних закладах, бізнесі або особистих цілях.

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Обсяг роботи становить 60 сторінок основного тексту, 38 рисунків та 1 таблиці.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Конспектування у цифрову епоху

Конспектування є важливим елементом формального навчання як в академічному, так і в професійному середовищі. Воно сприяє кращому засвоєнню матеріалу, структуризації знань та розвитку аналітичних навичок. Особи, які активно ведуть нотатки, зазвичай демонструють вищі результати у навчанні, проте ефективність цього процесу залежить від багатьох змінних. Серед них — структура лекції, знання про майбутні тести, сприйняття актуальності матеріалу, а також наявність та типи роздаткових матеріалів. Саме ці фактори ускладнюють визначення найефективніших методів ведення нотаток, що робить результати досліджень у цій сфері неоднозначними.

Конспектування – це індивідуальний процес, у якому важливу роль відіграють особисті відмінності: рівень когнітивних здібностей, стилі навчання, мотивація та здатність до концентрації. Відмінності між студентами у підходах до занотовування ускладнюють інтерпретацію результатів досліджень. Протягом багатьох років конспектування залишалося предметом наукових пошуків, спрямованих на виявлення чинників, що сприяють підвищенню продуктивності та успішності навчання. Проте через численні варіації підходів до нотування важко сформулювати однозначні висновки.

Традиційні методи конспектування включають:

- Лінійне конспектування – послідовний запис основних тез без додаткових візуальних елементів.
- Метод Корнелла – поділ сторінки на зони для ключових ідей, деталей та підсумків.
- Ментальні карти – візуалізація інформації у вигляді схематичних діаграм, які показують зв'язки між поняттями.

- Метод контурів (Outline method) – створення ієрархічної структури нотаток з рівнями деталізації.

Однак у сучасну цифрову епоху на процес ведення нотаток дедалі більше впливають нові фактори. Впровадження планшетів, застосунків для нотаток (наприклад, OneNote, Notion, Evernote) та інших освітніх технологій трансформує підходи до фіксації, сприйняття та обробки інформації. Цей технологічний прогрес пропонує нові можливості, як-от зберігання великої кількості даних, зручність у пошуку та спільний доступ до нотаток. Водночас він створює нові виклики:

- Цифрова перевантаженість – надлишок ресурсів може ускладнювати фокусування на ключових ідеях.
- Технічна залежність – потреба в пристроях і доступі до мережі.
- Зниження залученості – дослідження показують, що ручне конспектування може бути ефективнішим для запам'ятовування через залучення моторики [1].

Таким чином, сучасні дослідження спрямовані на пошук збалансованих підходів, що поєднують традиційні методи з цифровими інструментами. Це може включати, наприклад, використання планшетів для рукописних нотаток або оцифрування рукописних матеріалів для подальшого аналізу та зберігання.

1.2 Роздаткові матеріали та прописні методи конспектування

Роздаткові матеріали є важливим допоміжним засобом у процесі навчання, оскільки сприяють кращому засвоєнню інформації під час конспектування. Вони активно використовуються в академічному, професійному та дослідницькому середовищах. Основне питання, яке постає при використанні роздаткових матеріалів, стосується їхнього обсягу та формату. Існують різні думки щодо того, чи ефективніше засвоюється матеріал, коли студенти записують лекції власними словами або користуються

підготовленими матеріалами, які дозволяють зосередитися безпосередньо на змісті лекції.

Здатність перетворювати отриману інформацію на особисто значущі нотатки є критично важливою складовою навчального процесу. Конспектування сприяє активному залученню студента до опрацювання матеріалу, допомагаючи кодувати інформацію для довгострокового зберігання в пам'яті. Водночас цей процес є когнітивно складним і залежить як від навичок конспектування, так і від стилю викладання лектора. Наприклад, якщо лектор говорить занадто швидко або нерозбірливо, слухач може відчувати когнітивне перевантаження, що ускладнює одночасне сприйняття, обробку та записування інформації.

Приймаючи рішення щодо формату роздаткових матеріалів, викладачі мають збалансувати переваги самостійного конспектування з обмеженнями когнітивного навантаження. Одним із підходів є надання студентам структурованих конспектів лекційного матеріалу замість повного набору слайдів чи текстових нотаток викладача. Такий підхід забезпечує базовий когнітивний каркас, який дозволяє студентам зосередитися на слуханні лекції та записуванні лише ключових моментів. Дослідження свідчать, що студенти, які отримують конспекти з основною структурою лекції, зазвичай демонструють кращі результати у порівнянні з тими, хто записує матеріал повністю самостійно [2].

Окрім формату роздаткових матеріалів, важливим є застосування стандартизованих стратегій конспектування. Традиційно, коли лекції були основною формою подачі матеріалу, роздаткові матеріали використовувалися рідше, а основний акцент робився на запам’ятовуванні інформації. Для допомоги у фіксації інформації було розроблено низку методів, зокрема:

Процедура формального конспектування – запис основних тез з розширенням їх деталями.

Метод Корнелла – поділ сторінки на три зони: для ключових ідей, основного змісту та підсумків.

Активний метод Бартуша – підхід, який включає створення питань для самоперевірки під час запису.

Метод дослівного конспектування – буквальний запис усіх тез лекції без обробки інформації.

Хоча ці методи демонструють високу ефективність, вони часто залишаються недооціненими через зміни у педагогічних підходах та сучасні освітні тенденції. Зокрема, розвиток технологій та перехід до більш інтерактивних моделей навчання, як-от формат "перевернутого класу", знову актуалізували важливість систематичних підходів до ведення нотаток. У цьому форматі студенти переглядають лекційні матеріали до занять, а на самих заняттях зосереджуються на обговоренні та практичному застосуванні знань, що робить навички ефективного конспектування критично важливими.

1.3 Цифрові нотатки

Впровадження цифрових технологій в освітній процес зумовило появу спеціалізованих застосунків для ведення нотаток, які значно трансформували підходи до конспектування. Завдяки широкому функціоналу, таким як підтримка текстових, графічних і мультимедійних записів, ці програми стали важливими інструментами для студентів, дослідників і професіоналів у різних сферах діяльності. Однак ефективність їх використання безпосередньо залежить від функціональних можливостей, когнітивних переваг та потенційних обмежень, які вони створюють у процесі навчання.

Основні функціональні можливості цифрових застосунків для нотаток

Сучасні цифрові застосунки для ведення нотаток орієнтовані на полегшення процесу збору, обробки та зберігання інформації. Їхні ключові функціональні можливості включають:

8. Текстове нотування.

Більшість застосунків дозволяють створювати текстові записи як за допомогою клавіатури, так і рукописного вводу, що сприяє гнучкості підходів до конспектування.

9. Мультимедійна інтеграція.

Багато програм підтримують додавання до нотаток зображень, аудіофайлів, відеоматеріалів, а також можливість малювання діаграм та схем, що є важливим для конспектування технічних дисциплін.

10. Анотування та виділення ключової інформації. Функція виділення тексту, створення підписів і коментарів допомагає структурувати матеріал та акцентувати увагу на ключових концепціях.

11. Організація нотаток.

Важливою особливістю є можливість структуризації нотаток через використання тегів, папок та міток, що полегшує доступ до інформації.

12. Пошук за ключовими словами.

Інструменти пошуку дозволяють швидко знаходити необхідні фрагменти тексту навіть у великих масивах даних, що є перевагою порівняно з традиційними рукописними нотатками.

13. Синхронізація та хмарне зберігання.

Сучасні застосунки підтримують зберігання даних у хмарних сховищах, що забезпечує доступ до нотаток із різних пристроїв та знижує ризик втрати інформації.

14. Можливості для співпраці.

Деякі додатки дозволяють декільком користувачам одночасно редагувати документ, що є особливо корисним для групових проєктів та дослідницьких колаборацій.

Популярні застосунки для цифрових нотаток та їхні особливості

На сучасному ринку представлено низку програм, що задовольняють різноманітні потреби користувачів. Серед найбільш популярних:

- Microsoft OneNote: Відзначається гнучкою системою організації нотаток із використанням вкладених секцій і сторінок, підтримкою рукописного вводу та мультимедіа.
- Evernote: Призначений для текстових записів із можливістю додавання аудіофайлів, зображень, PDF-файлів і синхронізацією через хмарне сховище.
- Notability: Особливо популярний серед користувачів пристроїв Apple, завдяки підтримці Apple Pencil для створення рукописних нотаток та анотацій до PDF-файлів.
- GoodNotes: Орієнтований на рукописне конспектування з можливістю створення багаторівневих структур нотаток та додавання зображень і схем.
- Google Keep: Простий застосунок для створення швидких текстових нотаток із підтримкою тегів, чек-листів та синхронізації через Google Drive.
- Notion: Поєднує функції текстових нотаток, баз даних і таблиць, що робить його зручним для ведення комплексних проєктів.

Вплив цифрових застосунків на когнітивні процеси під час навчання

Використання цифрових застосунків для конспектування може мати як позитивний, так і негативний вплив на процес навчання, залежно від контексту їх застосування.

Позитивний вплив:

- Полегшення доступу до інформації: Завдяки функціям пошуку та систематизації, користувачі можуть легко знаходити потрібні фрагменти тексту.
- Візуалізація даних: Можливість інтеграції графіків, схем та діаграм сприяє кращому сприйняттю складної інформації.
- Покращення організації: Структурування матеріалів за допомогою тегів, папок і міток полегшує навігацію та управління великими обсягами даних.

Негативний вплив:

- Зниження когнітивної обробки: Дослідження Мюллера і Оппенгеймера (2014) показали, що студенти, які використовують ноутбуки для нотування, частіше фіксують інформацію дослівно, що може знижувати рівень розуміння матеріалу.
- Відволікання: Використання пристроїв із доступом до Інтернету може сприяти багатозадачності, що негативно впливає на концентрацію уваги.
- Пасивне конспектування: Якщо застосунок дозволяє лише копіювати текст без можливості його редагування та перефразування, це може обмежувати активне залучення до навчального процесу.

Критерії вибору застосунку для цифрового конспектування

При виборі застосунку для ведення цифрових нотаток доцільно враховувати такі критерії:

- Функціональність: Відповідність потребам користувача щодо формату конспектування.
- Зручність використання: Наскільки інтерфейс є інтуїтивно зрозумілим та адаптованим для щоденного використання.
- Сумісність: Підтримка різних операційних систем та пристроїв.
- Синхронізація: Можливість зберігання даних у хмарних сховищах та доступу з різних пристроїв.
- Безпека даних: Забезпечення конфіденційності та захисту інформації.

Застосунки для цифрових нотаток є потужними інструментами, які можуть значно покращити ефективність навчального процесу, особливо в умовах зростання ролі дистанційного навчання та гібридних освітніх моделей. Вони забезпечують зручний доступ до матеріалів, покращують їхню структуризацію та надають можливості для мультимедійної інтеграції. Однак, ефективність використання таких інструментів залежить від усвідомленого підходу до конспектування, де важливою залишається когнітивна обробка

інформації, а не лише механічне копіювання даних. Поєднання традиційних методів з цифровими може забезпечити оптимальні результати у навчанні.

1.4 Incremental Static Regeneration

Incremental Static Regeneration є однією з ключових функцій фреймворку Next.js, яка дозволяє оновлювати статично згенеровані сторінки після їх розгортання, не потребуючи повного перезавантаження веб-застосунку. Цей підхід поєднує переваги статичної генерації контенту (SSG) та серверного рендерингу (SSR), забезпечуючи високу продуктивність застосунку поряд із можливістю динамічного оновлення даних.

Incremental Static Regeneration функціонує на основі попередньо згенерованих сторінок під час білду, які створюються на основі даних, доступних у момент збірки застосунку. Однак, на відміну від класичної статичної генерації, ISR дозволяє задавати параметр часу для автоматичного оновлення контенту. Це досягається через використання методу getStaticProps у Next.js, який дозволяє отримувати дані під час білду, а завдяки параметру revalidate задавати інтервал, через який система ініціює оновлення даних. У разі, якщо користувач здійснює запит до сторінки, яка вже була згенерована, він отримує кешовану версію. Якщо з моменту останнього оновлення пройшов визначений інтервал часу, Next.js автоматично перегенерує сторінку у фоновому режимі, забезпечуючи доступ до актуального контенту для наступних відвідувачів [10].

Функціональність Incremental Static Regeneration забезпечує низку переваг для веб-розробки:

По-перше, використання попередньо згенерованих сторінок значно підвищує швидкість завантаження застосунку, оскільки більшість запитів обслуговуються через кешований контент.

По-друге, Incremental Static Regeneration дозволяє оновлювати дані у реальному часі без необхідності перевипуску всієї програми, що особливо

корисно для платформ із часто оновлюваним контентом, таких як блоги, новинні портали або онлайн-магазини.

По-третє, цей підхід допомагає знизити навантаження на сервер, оскільки сторінки оновлюються лише тоді, коли це необхідно.

Таким чином, Incremental Static Regeneration у Next.js є потужним інструментом для сучасної веб-розробки, який забезпечує баланс між продуктивністю, актуальністю даних та зручністю підтримки контенту. Він дозволяє розробникам створювати швидкі, надійні та ефективні веб-застосунки з можливістю гнучкого оновлення контенту.

1.5 Висновок до першого розділу

У даному розділі було проведено дослідження процесу конспектування в академічному та професійному середовищі, а також впливу цифрових технологій на ефективність навчання. Було проаналізовано традиційні методи ведення нотаток, такі як лінійне конспектування, метод Корнелла, ментальні карти та метод контурів. Визначено, що кожен із підходів має свої переваги та недоліки, залежно від індивідуальних особливостей студентів і стилю викладання.

Особливу увагу приділено цифровим інструментам для конспектування, включаючи OneNote, Notion, Evernote та інші застосунки, що забезпечують текстове та мультимедійне нотування, а також функції пошуку, анотування і спільного доступу до даних. Розглянуто як переваги використання цифрових інструментів, зокрема полегшення доступу до матеріалів та покращення організації нотаток, так і потенційні недоліки, серед яких зниження когнітивної обробки інформації та ризик цифрової перевантаженості.

Метою роботи є розробка веб-застосунку керування нотатками з використанням технології Incremental Static Regeneration.

Для досягнення поставленої мети в кваліфікаційній роботі було визначено та виконано такі завдання дослідження:

1. Проведено аналіз технічних засобів, які використовуються для розробки веб-застосунків та обрано необхідні інструменти для створення веб-застосунку для ведення нотатків.

2. Сформульовано вимоги до програмного забезпечення, включаючи функціональні (створення, редагування, видалення, пошук нотаток, аутентифікація користувачів) та нефункціональні (продуктивність, безпека, зручність використання) вимоги.

3. Спроєктовано та розроблено веб-застосунок для ведення нотатків із використанням підходу Incremental Static Regeneration (ISR) для забезпечення швидкої роботи з контентом та оптимізації продуктивності.

4. Проведено тестування створеного веб-застосунку, перевірено його функціональність, працездатність, а також відповідність сформульованим вимогам.

Вхідними даними для веб-застосунку є текстові дані, що вводяться користувачем у відповідні поля інтерфейсу. Ці дані обробляються серверною частиною, зберігаються у форматі JSON-файлів та можуть бути доступні для перегляду, редагування або видалення через відповідні API-запити.

РОЗДІЛ 2. АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ ВЕБ-ЗАСТОСУНКУ

2.1 Вибір програмних засобів для реалізації проекту

Вибір платформи для створення веб-застосунку є визначальним фактором для забезпечення стабільності, продуктивності та зручності подальшої підтримки проєкту. На сучасному етапі розвитку веб-технологій існує кілька підходів до розробки веб-застосунків, кожен з яких має власні переваги та обмеження. Основними критеріями вибору платформи є: продуктивність, гнучкість у розробці, легкість підтримки, інтеграція з базами даних та підтримка серверної й клієнтської частини.

Платформи для веб-розробки поділяються на декілька основних груп, серед яких:

1) Класичні монолітні платформи (LAMP, LEMP)

Монолітні платформи, такі як LAMP (Linux, Apache, MySQL, PHP) і LEMP (Linux, Nginx, MySQL, PHP), є одним із найстаріших стеків для розробки веб-застосунків. Вони базуються на серверній логіці з використанням PHP та реляційних баз даних MySQL. Такі платформи ідеально підходять для створення статичних та динамічних сайтів із мінімальними вимогами до інтерактивності.

Переваги:

- Простота розгортання.
- Стабільність та перевірені роками підходи.
- Велика кількість готових рішень.

Недоліки:

- Обмеженість у створенні складних SPA (Single Page Applications).
- Відсутність гнучкості для масштабування та інтерактивних елементів.

2) JavaScript-стеки (MEAN, MERN, MEVN)

З появою сучасних JavaScript-фреймворків популярними стали технологічні стеки MEAN (MongoDB, Express.js, Angular, Node.js), MERN (MongoDB, Express.js, React.js, Node.js) та MEVN (MongoDB, Express.js, Vue.js, Node.js). Вони засновані на використанні JavaScript на всіх рівнях застосунку: від клієнтської частини до серверної логіки та бази даних [5].

MEAN (Angular): підходить для корпоративних застосунків із суворою архітектурою та структурою даних.
MERN (React): популярний серед розробників завдяки гнучкій компонентній архітектурі та легкому управлінню інтерфейсом.
MEVN (Vue.js): використовується для менш складних, але інтерактивних застосунків із фокусом на простоту [3].

Переваги:

- Використання єдиної мови програмування (JavaScript) для всієї архітектури.
- Гнучка компонентна архітектура.
- Добре підходять для SPA та динамічних веб-додатків.

Недоліки:

- MEAN: високий поріг входу через складність Angular.
- MERN: обмеженість серверного рендерингу та SEO-оптимізації.
- MEVN: менша екосистема у порівнянні з React.

3) Full-Stack фреймворки (Next.js, Nuxt.js, Meteor.js)

Full-Stack фреймворки поєднують можливості розробки як клієнтської, так і серверної частини в одному середовищі. Серед таких платформ виділяються Next.js (для React), Nuxt.js (для Vue.js) та Meteor.js.

Next.js забезпечує підтримку SSR (Server-Side Rendering), SSG (Static Site Generation) та ISR (Incremental Static Regeneration), що дає змогу створювати як статичні, так і динамічні сторінки з високою продуктивністю. Nuxt.js працює аналогічно з Vue.js, додаючи можливості серверного рендерингу.

Meteor.js — платформа для повного циклу розробки з мінімальними налаштуваннями.

Переваги:

- Підтримка серверного рендерингу та статичних сторінок.
- Висока продуктивність та SEO-оптимізація.
- Компонентний підхід та можливість використання TypeScript.

Недоліки:

- Вища складність налаштування порівняно зі SPA-фреймворками.
- Потреба у знаннях React або Vue.js [6].

4) JAMstack (JavaScript, APIs, Markup)

JAMstack — це підхід до створення статичних сайтів із використанням JavaScript для інтерактивності та зовнішніх API для обробки даних. Фреймворки, такі як Gatsby.js та Eleventy, використовуються для генерації статичних сторінок з API для обробки даних.

Переваги:

- Висока швидкість завантаження сторінок.
- Простота хостингу та масштабування.
- Відмінно підходить для інформаційних сайтів.

Недоліки:

- Обмежені можливості для складних додатків.
- Не підходить для рішень із великою кількістю динамічних даних.

5. Python та Java фреймворки (Django, Flask, Spring Boot)

Python- і Java-орієнтовані платформи також популярні для розробки серверних частин веб-застосунків.
Django — повноцінний Python-фреймворк із вбудованими засобами для створення веб-застосунків, ORM та адмін-панелі.
Flask — мінімалістичний Python-фреймворк для простих REST API.
Spring Boot — Java-фреймворк для створення масштабованих корпоративних систем.

Переваги:

- Потужність та безпека.
- Ідеально підходять для великих систем.

Недоліки:

- Високий поріг входу для новачків.
- Менш гнучкі для SPA-підходів.

Для детальнішого аналізу та обґрунтування вибору платформи для реалізації веб-застосунку нижче представлено порівняльну таблицю основних стеків технологій, що використовуються у сучасній веб-розробці. Таблиця демонструє ключові особливості кожного стеку, включаючи використання клієнтської та серверної частин, типи баз даних, а також переваги й недоліки, що впливають на вибір відповідного рішення для розробки веб-застосунків.

Таблиця 1.1

Порівняльна таблиця технологій

Стек техно-логій	Front-end	Back-end	База даних	Особливо-сті	Переваги	Недоліки
MEAN (MongoDB, Express.js Angular, Node.js)	Angular	Express.js, Node.js	MongoDB	Повний стек для створення SPA з використанням JavaScript на всіх рівнях.	Єдина мова (JavaScript), підтримка SPA, масштабованість	Високий поріг входу через Angular, складність налаштування
MERN (MongoDB, Express.js, React.js, Node.js)	React.js	Express.js, Node.js	MongoDB	Фокус на створення динамічних веб-застосунків з компонентною архітектурою.	Гнучкість React, JS на всіх рівнях, легкість для SPA	Відсутність готових рішень для складних додатків
MEVN (MongoDB, Express.js, Vue.js, Node.js)	Vue.js	Express.js, Node.js	MongoDB	Легший у використанні аналог MERN із спрощеною логікою Vue.	Простіший синтаксис Vue, JS на всіх рівнях	Менша екосистема порівняно з React та Angular
LAMP (Linux,	HTML, CSS, JS	PHP (на Apache)	MySQL	Класичний стек для	Простота використан-	Обмежена гнучкість

Apache, MySQL, PHP)				створення веб-сайтів із серверним рендерингом.	ня, стабільність, готовність до продакшну	для SPA, застарілі підходи
LEMP (Linux, Nginx, MySQL, PHP)	HTML, CSS, JS	PHP (на Nginx)	MySQL	Подібний до LAMP, але з кращою продуктивністю через Nginx.	Вища продуктивність порівняно з Apache	Застарілість PHP для складних SPA-додатків
JAMstack (JavaScript, APIs, Markup)	React.js, Vue.js, Angular	API (серверлес або хмарні сервіси)	Серверлесс хо-вища	Фокус на статичних сайтах з API для обробки даних.	Висока продуктивність, швидкий деплой	Підходить для статичних сайтів, складний для динаміч-них
Next.js Full-Stack (Node.js, React, TypeScript)	React.js + TypeScript	Node.js (SSR, ISR)	JSON або MongoDB	Підтримка серверного рендерингу, статичних сторінок та ISR.	Висока продуктивність, SEO, гнучкість SSR та ISR	Складніший у налашту-ванні для новачків
Ruby on Rails (RoR)	HTML, CSS, JS	Ruby on Rails	PostgreSQL, MySQL	Фреймворк з фокусом на швидку розробку та автоматизацію процесів.	Простота створення CRUD-додатків	Менша популярність у порівнянні з JS-стеками
Django + React	React.js	Django (Python)	PostgreSQL, SQLite	Поділ фронтенду і бекенду, зручний для великих проєктів.	Висока безпека, зручність для складних проєктів	Вища складність у налашту-ванні та інтеграції
Spring Boot + Angular	Angular	Spring Boot (Java)	PostgreSQL, MySQL	Корпоратив-ний стек для створення великих додатків.	Висока продуктив-ність, масштабо-ваність	Складний у налашту-ванні, високий поріг входу

Для реалізації веб-застосунку для управління нотатками було обрано Next.js Full-Stack стек у поєднанні з Node.js, React.js та TypeScript. Такий вибір обумовлений потребою у швидкому завантаженні сторінок, підтримкою серверного рендерингу (SSR) та технологією Incremental Static Regeneration

(ISR). Порівняно з іншими стеками, Next.js забезпечує високу продуктивність та гнучкість у реалізації як статичних, так і динамічних сторінок. Обраний стек також дозволяє використовувати одну мову програмування (JavaScript/TypeScript) як для клієнтської, так і для серверної частини, що значно спрощує розробку.

2.2 Вимоги до програмного забезпечення

Вимоги до програмного забезпечення є важливим етапом у процесі розробки веб-застосунку, оскільки вони визначають функціональні можливості, які повинна виконувати система, а також технічні характеристики, що впливають на продуктивність, масштабованість та безпеку продукту. В даному розділі розглянуто основні функціональні, нефункціональні вимоги, а також вимоги до інтеграції з базою даних для веб-застосунку управління нотатками.

Функціональні вимоги описують основний набір операцій, які повинні бути доступними користувачам веб-застосунку для виконання основних завдань. У цьому проєкті застосунок для управління нотатками повинен підтримувати наступний функціонал:

- Створення нотатки: Користувач повинен мати можливість створювати нові нотатки з додаванням текстового контенту та міток.
- Редагування нотатки: Забезпечення можливості редагування існуючих нотаток для внесення змін до вмісту або міток.
- Видалення нотатки: Користувач повинен мати можливість видаляти непотрібні нотатки.
- Перегляд списку нотаток: Веб-застосунок має відображати всі нотатки у вигляді списку або карток для зручного перегляду.
- Пошук нотаток: Передбачено можливість пошуку нотаток за ключовими словами.

- Фільтрація за мітками: Користувач повинен мати змогу фільтрувати нотатки за категоріями або мітками.
- Аутентифікація користувача: Реалізація базової системи аутентифікації для управління доступом до персональних нотаток.

Дані функціональні можливості забезпечують базову функціональність для управління нотатками, спрощуючи користувачеві роботу з додатком.

Нефункціональні вимоги

1. Продуктивність – застосунок повинен забезпечувати швидке завантаження сторінок і оперативну обробку запитів до сервера.
2. Масштабованість – архітектура повинна підтримувати можливість розширення функціоналу та обробку зростаючого обсягу даних і кількості користувачів.
3. Безпека – дані користувачів мають бути захищені від несанкціонованого доступу за допомогою сучасних методів шифрування та аутентифікації.
4. Зручність використання – інтерфейс застосунку має бути зручним, інтуїтивно зрозумілим та адаптованим для різних пристроїв.
5. Надійність – програмне забезпечення має функціонувати стабільно, мінімізуючи ризики збоїв у роботі системи.

2.3. Засоби для розробки клієнтської частини

Розробка клієнтської частини веб-застосунку для управління нотатками здійснюється із застосуванням сучасних технологій, орієнтованих на гнучкість, продуктивність та зручність у підтримці коду. Основою для створення інтерфейсу було обрано бібліотеку React.js, яка є однією з найпопулярніших у сучасній веб-розробці. React.js використовує компонентний підхід, що передбачає розділення інтерфейсу на незалежні, багаторазово використовувані елементи. Це сприяє покращенню структури коду, його читабельності та полегшує підтримку й масштабування додатка.

Однією з ключових особливостей React.js є використання віртуального DOM (Virtual Document Object Model), який значно підвищує продуктивність застосунку. Віртуальний DOM дозволяє оновлювати лише ті елементи сторінки, які зазнали змін, замість повного перерендерингу. Це особливо важливо для інтерактивних застосунків, де інтерфейс змінюється динамічно у відповідь на дії користувача [14].

Для створення компонентів у React застосовується синтаксис JSX (JavaScript XML) – розширення JavaScript, яке дозволяє писати HTML-подібний код безпосередньо у файлах JavaScript. JSX забезпечує зручний спосіб визначення структури інтерфейсу, дозволяючи одночасно керувати логікою та візуальним відображенням компонентів. Це сприяє зменшенню кількості коду та полегшує розробку складних інтерфейсів.

Окрім JSX, у клієнтській частині застосунку використовуються CSS-модулі для стилізації компонентів. CSS-модулі дозволяють уникнути конфліктів між стилями різних елементів за рахунок автоматичної ізоляції класів у межах кожного компонента. Це забезпечує чистоту коду та спрощує подальше масштабування застосунку, оскільки стилі залишаються локальними для конкретних компонентів [15].

З метою підвищення якості коду та зниження ймовірності помилок у клієнтській частині було прийнято рішення використовувати TypeScript – надбудову над JavaScript, яка додає підтримку статичної типізації. Завдяки статичній типізації TypeScript дозволяє виявляти помилки вже на етапі компіляції, що значно зменшує ризик помилок під час виконання програми. Це особливо важливо для великих проєктів, де контроль за типами даних сприяє кращій підтримуваності коду та спрощує його рефакторинг.

TypeScript також сприяє підвищенню читабельності коду завдяки можливості створення інтерфейсів для об’єктів, що забезпечує чітке визначення типів даних, які можуть використовуватися в компонентах та функціях. Наприклад, для компонента нотатки можна створити тип Note, який включатиме такі поля, як title, content, tags та createdAt. Це допомагає уникнути

помилок, пов’язаних із неправильним використанням об’єктів або їх властивостей.

2.4. Засоби для розробки серверної частини

У розробці серверної частини веб-застосунку для управління нотатками використовується сучасний фреймворк Next.js, який забезпечує ефективне поєднання клієнтської та серверної логіки в одному середовищі. Next.js є фреймворком для React, що дозволяє реалізовувати серверний рендеринг (SSR), статичну генерацію сторінок (SSG) та інкрементальну статичну генерацію (ISR). Завдяки цим можливостям застосунок забезпечує високу продуктивність та зручність роботи як із клієнтською, так і з серверною частиною [8].

Основним принципом роботи Next.js є використання серверних функцій через папку /api, яка дозволяє створювати серверні ендпойнти безпосередньо в межах проєкту. Це забезпечує спрощену реалізацію обробки запитів до сервера, зокрема для створення, оновлення, видалення та перегляду нотаток. Таким чином, серверна логіка інтегрується без необхідності додаткового налаштування окремого серверного середовища, що прискорює процес розробки [4].

У даному проєкті серверна частина відповідає за такі ключові функції:

- Обробка HTTP-запитів: Використовується серверний API Next.js для обробки GET, POST, PUT та DELETE-запитів. Це дозволяє керувати нотатками безпосередньо через серверні ендпойнти, які автоматично створюються у папці pages/api.
- Робота з файловою системою: Дані зберігаються у локальному JSON-файлі, а взаємодія з ним реалізована через модуль fs (File System), вбудований у Node.js. Функції запису, зчитування та оновлення файлу виконуються за допомогою асинхронних методів, що забезпечує ефективність роботи сервера.

- Серверний рендеринг (SSR): Next.js дозволяє здійснювати серверний рендеринг сторінок у реальному часі, що покращує SEO-оптимізацію застосунку та знижує навантаження на клієнтську частину.
- Інкрементальна статична генерація (ISR): Цей підхід дозволяє оновлювати статичний контент без необхідності повного перегенерування сайту, що особливо важливо для застосунків з динамічними даними, такими як нотатки.

Для покращення якості коду серверної частини застосовується TypeScript, який додає статичну типізацію та забезпечує виявлення помилок ще на етапі компіляції. Визначення типів для об'єктів нотаток (Note) дозволяє знизити ймовірність помилок при обробці даних, а також покращує читабельність коду.

У питаннях безпеки серверної частини Next.js також пропонує зручні механізми. Для обмеження доступу до захищених ресурсів використовується базова система аутентифікації на основі JWT-токенів (JSON Web Tokens). Це дозволяє забезпечити захист персональних даних користувачів, надаючи доступ до нотаток лише після успішної аутентифікації.

2.5 Висновки до другого розділу

У другому розділі було здійснено ґрунтовний аналіз засобів та інструментів для створення веб-застосунку для ведення нотатків. Основну увагу приділено обґрунтуванню вибору технологічного стеку, здатного забезпечити стабільну роботу системи, її продуктивність, масштабованість та відповідність сучасним вимогам безпеки й зручності використання.

У процесі дослідження було розглянуто низку технологічних підходів до розробки веб-застосунків, зокрема стеків MEAN, MERN, LAMP та JAMstack. Аналіз цих рішень дозволив встановити, що для поставлених цілей найбільш оптимальним є використання технологій Next.js, React.js, Node.js та TypeScript, які забезпечують єдину мову програмування на клієнтській та серверній

частинах, підтримку серверного рендерингу (SSR) та використання інкрементальної статичної генерації (ISR).

Важливою частиною дослідження стало формування вимог до програмного забезпечення, серед яких:

- Функціональні вимоги: можливість створення, редагування, видалення нотаток, пошуку за ключовими словами, фільтрації за мітками, а також базова аутентифікація користувачів.
- Нефункціональні вимоги: забезпечення високої продуктивності системи, надійності зберігання даних, захисту інформації користувачів, а також зручності інтерфейсу для кінцевих користувачів.

Було обґрунтовано архітектурний підхід до розробки веб-застосунку. Клієнтська частина створена з використанням бібліотеки React.js, що забезпечує компонентну структуру та сприяє повторному використанню елементів інтерфейсу. Серверна частина реалізована за допомогою Next.js, що дозволяє обробляти HTTP-запити через API-ендпойнти та підтримує рендеринг на стороні сервера. Для зберігання даних обрано файлову систему у форматі JSON, що забезпечує простоту інтеграції з основними функціями застосунку.

Таким чином, у другому розділі було обґрунтовано вибір сучасного технологічного стеку для реалізації веб-застосунку, визначено функціональні та нефункціональні вимоги до системи, а також описано архітектурний підхід, який забезпечує ефективну роботу програмного продукту та створює основу для подальшої практичної реалізації.

РОЗДІЛ 3 РОЗРОБКА ВЕБ-ЗАСТОСУНКУ НОТАТКУ З ВИКОРИТАННЯМ INCREMENTAL STATIC REGENERATION (ISG)

3.1 Актуальність розробки

У сучасному світі веб-застосунки є невід'ємною частиною повсякденного життя, а зручність та швидкість їх роботи є ключовими факторами для користувачів. Розвиток технологій веб-розробки, зокрема Next.js, відкрив нові можливості для підвищення продуктивності та оптимізації користувацького досвіду [9].

Інкрементальна статична генерація (ISR) є інноваційним підходом до генерації статичного контенту, який дозволяє поєднувати переваги статичних сайтів (швидкість завантаження, безпека) із динамічністю традиційних серверних рішень. У більшості реалізацій ISR використовується для сайтів з великим обсягом статичного контенту, таких як блоги чи магазини. У роботі пропонується адаптація цієї концепції для управління динамічними даними у персональних системах, таких як нотатки. Проєкт передбачає реалізацію ефективного механізму часткової генерації сторінок, що дозволить знижувати затримки оновлення контенту та уникати зайвих запитів до сервера. Використання JSON як основного формату зберігання є додатковим внеском у гнучкість системи. Це рішення дозволяє легко інтегрувати систему з іншими технологіями або експортувати дані у зовнішні сервіси.

Інкрементальна статична генерація (ISR) — це підхід у веб-розробці, який дозволяє оновлювати статичні сторінки без необхідності повного перегенерування всього сайту. ISR працює шляхом оновлення лише тих сторінок, які були змінені або до яких здійснено запит після заданого інтервалу часу. Завдяки цьому можна поєднати високу швидкість статичних сайтів (оскільки сторінки вже згенеровані й кешовані) з можливістю динамічного оновлення контенту.

Incremental Static Regeneration (ISR) та Node.js взаємопов’язані через використання Node.js як серверного середовища, що підтримує реалізацію ISR. Node.js використовується як основа для серверного виконання JavaScript-коду, що відповідає за обробку HTTP-запитів до сторінок. Коли користувач запитує сторінку, Node.js повертає або статичну версію сторінки (з кешу), або, у разі її оновлення, створює нову у фоновому режимі. Завдяки асинхронній архітектурі Node.js забезпечує високу продуктивність під час обробки великої кількості запитів.

Node.js забезпечує гнучкість і продуктивність, які необхідні для роботи ISR. Його особливості, такі як асинхронність, підтримка веб-серверів і інтеграція з сучасними фреймворками, роблять його ідеальним вибором для створення динамічних, масштабованих застосунків. У контексті ISR Node.js виступає "двигуном", що обробляє запити, оновлює сторінки та підтримує високу продуктивність системи.

Next.js є одним із найбільш популярних фреймворків. Він забезпечує також інтеграцію з React.js та TypeScript.js для простоти розробки інтерфейсів.

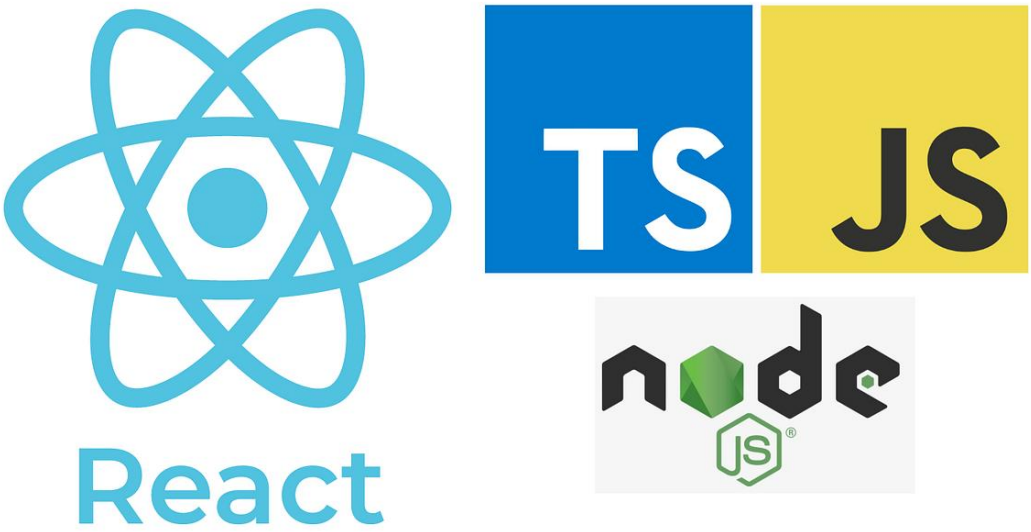

Рисунок 3.1 – Технології, що підтримує node.js

TypeScript – це мова програмування, яка є строго типізованим надбудовою над JavaScript. Вона додає до JavaScript можливості статичної типізації, що дозволяє розробникам визначати типи змінних, функцій і об'єктів. Це сприяє виявленню помилок ще на етапі написання коду, що робить розробку більш передбачуваною та знижує ризик виникнення багів у великих проєктах. TypeScript повністю сумісний із JavaScript, що дозволяє інтегрувати його в існуючі проєкти без серйозних змін. Завдяки підтримці сучасних функцій ECMAScript, TypeScript також надає інструменти для розробки масштабованих і надійних застосунків, особливо у великих командах. У Node.js TypeScript широко застосовується для створення API, серверних функцій і типізованих даних [7].

React.js – це бібліотека для створення користувацьких інтерфейсів, яка використовується для динамічного відображення контенту. У поєднанні з Node.js і Next.js React.js дозволяє створювати компоненти, які рендеряться як на стороні клієнта, так і на сервері.

3.2 Структура проекту

В якості структури використовується базова файлова система node.js, яка автоматично створюється під час ініціалізації проекту. Загалом структура проекту виглядає наступним чином:

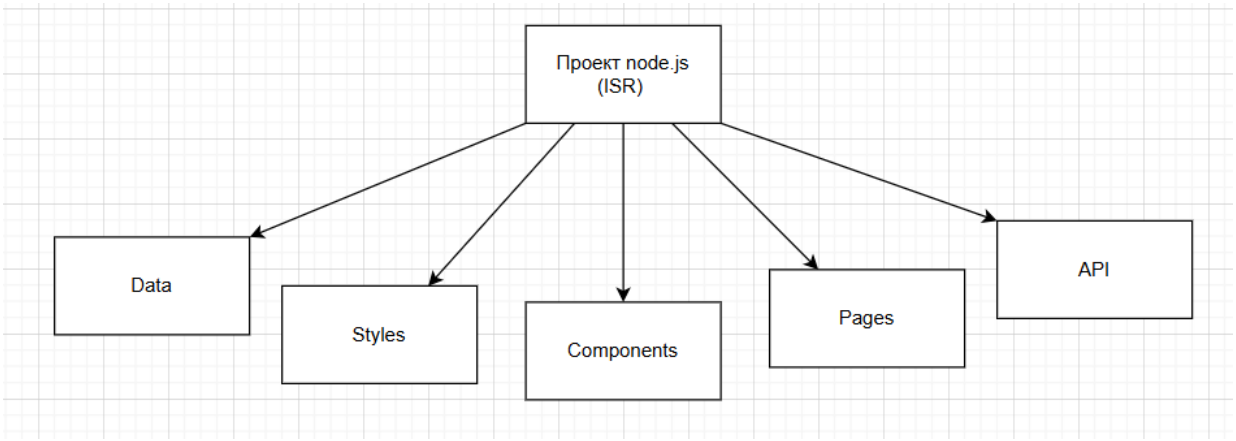

Рисунок 3.2 – Загальна структура проекту

- pages/: Це центральна частина Next.js додатка. Кожен файл JavaScript або TypeScript у каталозі pages автоматично стає маршрутом на вебсайті. Наприклад, файл index.js відповідає за головну сторінку сайту. API маршрути розміщуються в папці pages/api/.
- components/: Каталог для всіх компонентів, які можуть бути використані на кількох сторінках (наприклад, шапка сайту, кнопки, таблиці і т. д.).
- styles/: Тут знаходяться всі стилі для проєкту. Можуть бути глобальні стилі (наприклад, для всіх сторінок) або стилі, специфічні для певних компонентів.
- Api/ зберігають компоненти TypeScript які відповідають за які дозволяють обробляти HTTP запити (GET, POST, PUT, DELETE тощо). Ці маршрути можуть виконувати різні операції на сервері, наприклад, отримувати дані з бази даних, виконувати аутентифікацію, відправляти електронні листи чи працювати з іншими сервісами.

Дана структура проекту дозволяє чітко розділити різні частини проєкту на категорії, що робить код більш зрозумілим для розробника. Також Компоненти в окремих файлах дозволяють повторно використовувати їх на різних сторінках чи в різних частинах додатку. Це зменшує дублювання коду і покращує підтримку проєкту.

3.3 Розробка веб-додатку

Розробка Node.js проєкту в Visual Studio Code (VS Code) має багато переваг завдяки потужному набору функцій та розширень, які забезпечують зручну та ефективну роботу. По-перше, VS Code — це легкий, але в той же час потужний редактор з інтегрованими інструментами для розробки, що дозволяє значно пришвидшити процес кодування. Він підтримує інтерактивний інтерфейс командного рядка, що дозволяє легко запускати сервери, виконувати скрипти, перевіряти логи та тестувати код безпосередньо з

редактора. За допомогою розширень, таких як Node.js Extension Pack, можна автоматично підключати дебагери, працювати з Git, а також мати доступ до численних інструментів для тестування та профілювання. Крім того, VS Code пропонує чудову підтримку JavaScript/TypeScript, включаючи автодоповнення, перевірку синтаксису в реальному часі та інтелектуальне підсвічування коду, що значно знижує кількість помилок та покращує продуктивність розробника. Завдяки інтеграції з npm (Node Package Manager) можна легко керувати залежностями проєкту, встановлювати пакети, оновлювати їх та виконувати скрипти. Також важливою перевагою є інтеграція з Git, яка дозволяє зручно працювати з версіями коду, створювати гілки та виконувати коміти без необхідності покидати редактор. Усі ці можливості, зручний інтерфейс та підтримка широкого спектра технологій роблять VS Code ідеальним інструментом для розробки Node.js проєктів, забезпечуючи ефективність, швидкість і гнучкість на кожному етапі розробки.

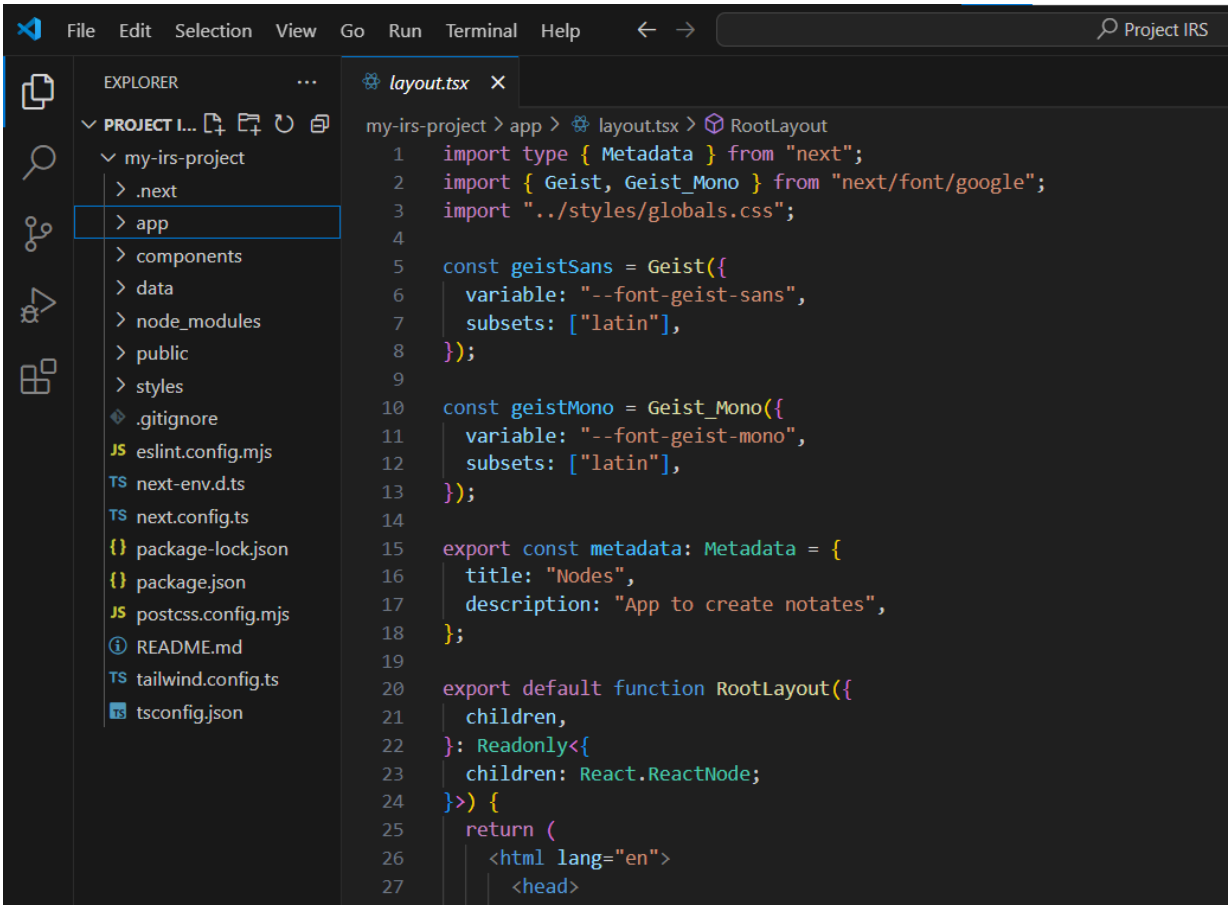

Рисунок 3.3 – Середовище розробки Visual Studio Code

3.3 Розробка додатку

Загалом основний функціонал додатку розподілений між 3 компонентами Components, API, Pages:

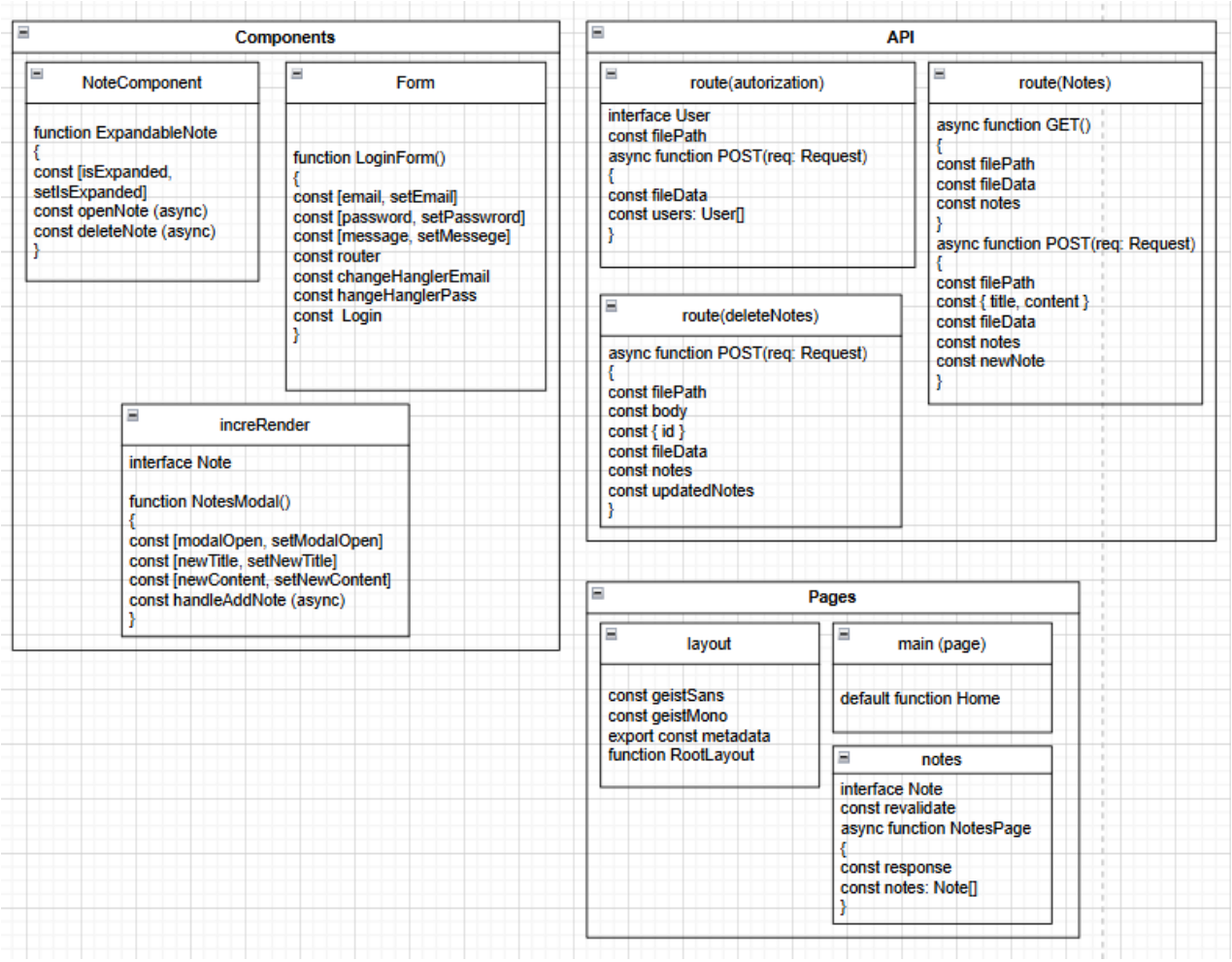

Рисунок 3.4 – Діаграма структури компонентів Components, API, Pages

Як вже зазначалося API повністю побудований на технології TypeScript, та не передбачає жодної розмітки. Дані компоненти відповідають за відправку та отримання даних.

Наступні компоненти Pages, Components взаємодіють між з собою та побудовані на синтаксисі .jsx. JSX (JavaScript XML) — це розширення синтаксису для JavaScript, яке дозволяє писати HTML-подібний код всередині JavaScript-файлів. Це є основною особливістю таких бібліотек, як React, де JSX

використовується для створення інтерфейсів користувача. Основна мета JSX — зробити код більш зручним для написання, дозволяючи розробникам описувати структуру інтерфейсу, немов це звичайний HTML, при цьому зберігаючи всю потужність JavaScript для логіки і маніпулювання даними.

Спочатку розглянемо компоненти групи Pages:

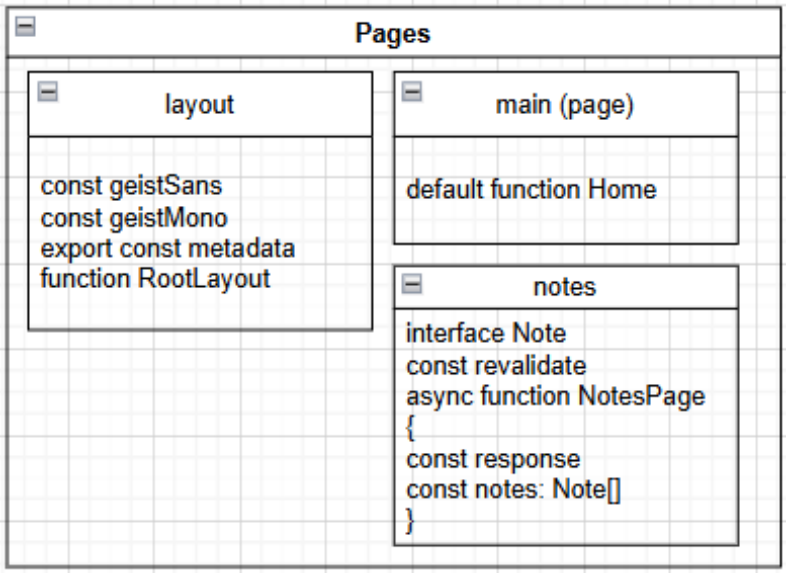

Рисунок 3.5 – Компоненти групи Pages

Layout.jsx в контексті Node.js – це файл компонента, який визначає загальну структуру та макет (layout) для різних сторінок веб-додатку. Він використовується для створення спільного шаблону, який застосовується до кількох сторінок або компонентів, що дозволяє забезпечити уніфіковану навігацію, заголовки, футери, стилі та інші елементи інтерфейсу на всіх сторінках проєкту. Основна мета такого компонента — це узгодженість інтерфейсу на різних сторінках або секціях додатку.

```tsx
export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode;
}>) {
  return (
    <html lang="en">
      <head>
      <link href='https://unpkg.com/boxicons@2.0.7/css/boxicons.min.css' rel='stylesheet'></link>
      </head>

      <body className={`${geistSans.variable} ${geistMono.variable} antialiased`}>
        <header className="navbar">
          <div className="logo"> Notes App</div>
          <ul className="link">
            <li><a href="#">Home</a></li>
            <li><a href="#">About</a></li>
          </ul>
        </header>
        <main className="main">{children}</main>
      </body>
    </html>
  );
}
```

Рисунок 3.6 – Компонент Layout.jsx

У програмуванні на JavaScript (та відповідно в Next.js, що базується на JavaScript/TypeScript), змінні, оголошені за допомогою ключового слова const, використовуються для збереження значень, які не можуть бути змінені після їхнього початкового присвоєння.

В файлі описано структуру навігації за допомогою тегу header, та вміст, що міститься в масиві children.

Наступний компоненти групи Pages, є сторінка авторизації, яка використовує так званий React-компонент, що додається за допомогою тегу <LoginForm/>:

```tsx
import { LoginForm } from "@/components/form";

export default function Home() {
  return (
    <>
      <LoginForm/>
    </>
  );
}
```

Рисунок 3.7 – Сторінка авторизації з React-компонентом

React-компонент – це незалежна, багаторазова частина інтерфейсу користувача (UI). У React компоненти є основними будівельними блоками додатків, що дозволяють розділяти інтерфейс на невеликі, керовані й багаторазові частини. Кожен компонент відповідає за рендеринг певної частини інтерфейсу на основі вхідних даних. Компоненти можуть бути функціональними або класовими. Функціональні компоненти є простішими і використовують функції для опису інтерфейсу, в той час як класові компоненти мають більш розширені можливості, зокрема для роботи зі станом. Компонент може отримувати вхідні дані через props (які передаються з батьківського компонента) та може керувати своїм внутрішнім станом за допомогою state (через useState). Після отримання даних компонент рендерить відповідну частину UI, оновлюючи її при зміні стану або вхідних параметрів. Завдяки такій модульності, React дозволяє створювати складні веб-додатки, де інтерфейс зберігає свою ефективність і гнучкість, а також легкість у підтримці і тестуванні.

Наступним компонентом page(notes) - це сторінка, в якому користувач може отримати нотатки, що були завантажені з бази даних. Робота компонентів починається з імпортування відповідних ресурсів. Для прикладу на сторінці використовується для відображення всіх нотатків <ExpandableNote key={note.id} note={note}/>. Для роботи react-компоненту необхідно виконати імпорт: import { ExpandableNote } from "@/components/NoteComponent".

Далі в основній частині сторінку використовується асинхронна функція NotesPage в для завантаження даних через fetch з API (http://localhost:3000/api/notes). Метод fetch() приймає як аргумент URL (або адресу ресурсу), до якого потрібно здійснити запит. За замовчуванням цей метод виконує GET-запит. Це API (notes) повертає список нотаток у форматі JSON.

```
export default async function NotesPage() {
  // Завантаження даних через fetch із параметром revalidate
  const response = await fetch("http://localhost:3000/api/notes", {
    next: { revalidate: 10 },
  });
  const notes: Note[] = await response.json();

  return (
    <div>
      <div className="bg-color-gr">
      <h1>Мої Нотатки</h1>
        <div className="bg-blur">
      <div className="notates">
        {notes.map((note) => (
          <ExpandableNote key={note.id} note={note}/>
        ))}
      </div>
      <NotesModal></NotesModal>
      <div className="container">
        <div className="typing-effect">
            Керуй власним життям!
        </div>
       </div>
      </div>
      </div>
    </div>
```

Рисунок 3.8 – Функція NotesPage

Після того як дані будуть завантажені, вони зберігаються у змінній notes і відображатимуться за допомогою методу map. Для кожної нотатку з масиву notes створюється компонент ExpandableNote, якому передаються наступні вхідін дані (key={note.id} та note={note}), що дозволяє рендерити окремі нотатки на сторінці.

Останнім елементом на сторінці <NotesModal></NotesModal> — компонент для відображення модального вікна, який, дозволяє додавати нотатки. Для ініціалізації функції NotesPage використовується конструкція export default function. Вона дозволяє оголосити функцію, яку можна експортувати з файлу як основну (за умовчанням), і потім імпортувати її в іншому файлі. При імпорті з модуля можна задати будь-яке ім’я для експортованої функції, що дає гнучкість при використанні в інших частинах програми. Оскільки кожен файл може мати лише один default експорт, це дозволяє чітко визначити основну функціональність модуля, що робить код більш організованим і зрозумілим.

Наступним потрібно розглянути групу Components, що містить групу react-компонентів:

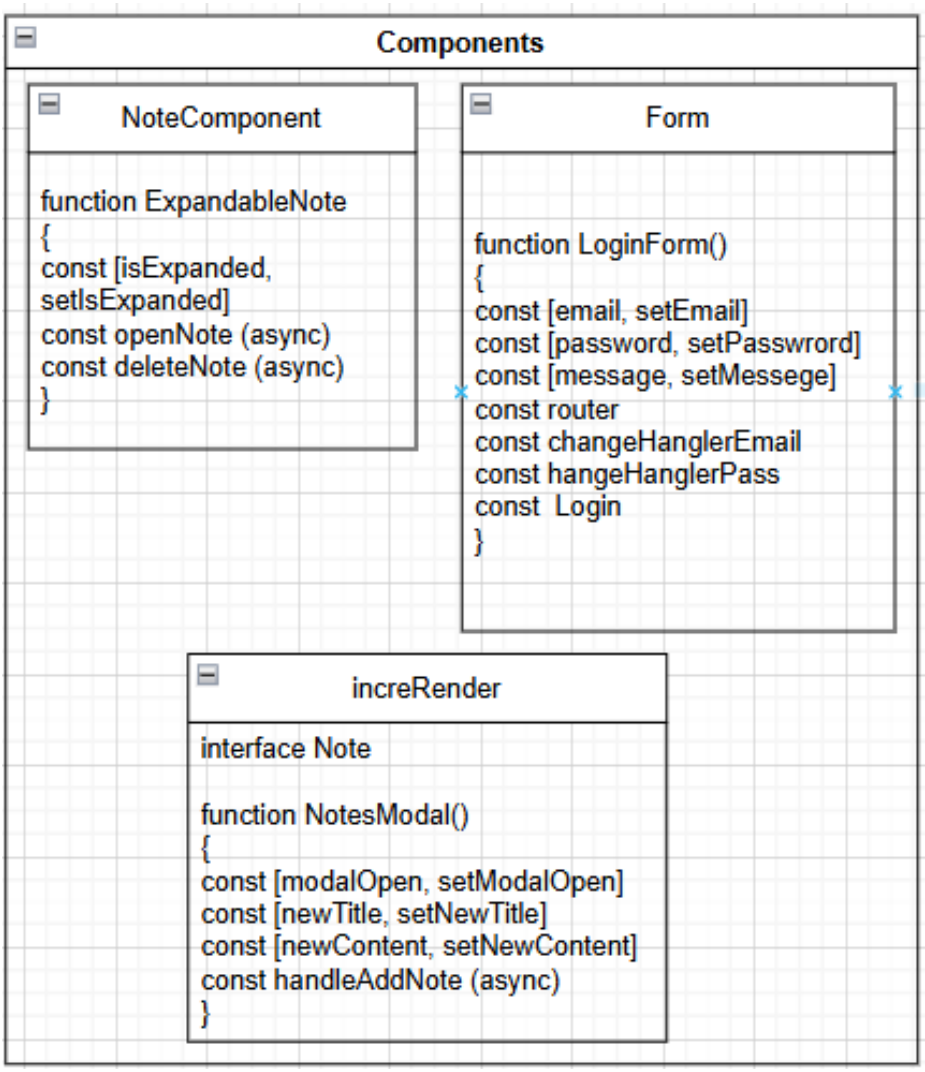

Рисунок 3.9 – Група Components

Перш за все потрібно розглянути компонент form.tsx. Це react-компонент, що використовуються для відображення та роботи форми авторизації:

```
export function LoginForm(){
    const [email, setEmail] = useState('')
    const [password, setPasswrord] = useState('')
    const [message, setMessege] = useState('')
    const router = useRouter()
    const changeHanglerEmail = (event:React.ChangeEvent<HTMLInputElement> )=>
        {
            setEmail(event.target.value)
        }
    const hangeHanglerPass = (event:React.ChangeEvent<HTMLInputElement> )=>
        {
            setPasswrord(event.target.value)
        }
    const  Login = async (event: React.FormEvent)=>{
        event.preventDefault()
        let newUser = {
            email,
            password
```

Рисунок 3.10 – Перша частина компоненту Form.tsx

У першій частині коду здійснюється ініціалізація кількох станів за допомогою useState з бібліотеки React. Стани — це механізм, що дозволяє зберігати й оновлювати дані в компоненті. Спочатку створюються три стани: email, password, і message. Кожен з них ініціалізується порожнім значенням (пустим рядком). Для кожного з цих станів визначені функції для їх оновлення: setEmail, setPasswrord і setMessege. Крім того, в компоненті ініціалізується змінна router за допомогою useRouter з Next.js, що дозволяє програмно переміщуватися між сторінками додатку.

Далі йде визначення самої форми. Вона огорнута в контейнери div з класами form та wrapper, що відповідають за стилізацію цілих блоків. Форма складається з двох основних полів вводу для електронної пошти (email) і пароля (password). Для кожного поля введення створюється окремий контейнер int-box, всередині якого знаходиться текстове поле input з атрибутом value, а також з обробником подій onChange, які оновлюють стани при введенні.

```
<>
<div className="form">
<div className="wrapper">
    <form action="" onSubmit={Login}>
        <h1>Register</h1>
        <div className="int-box">
            <input value={email} type="text"  placeholder="Username" onChange={changeHanglerEmail} required></input>
            <i className='bx bxs-user-pin'></i>
        </div>
        <div className="int-box">
            <input value={password} type="password" placeholder="Password" onChange={hangeHanglerPass} required></input>
            <i className='bx bxs-lock'></i>
        </div>
        <button type="submit" className="btn">Register</button>
        {message && <p>{message}</p>}
    </form>
</div>
</div>
</>
```

Рисунок 3.11 – HTML-розмітка форми

Форма обробляє події через функцію Login (рис. 3.12), яка викликається при натисканні на кнопку submit. Перед відправкою форми вона запобігає стандартній поведінці браузера (перезавантаженню сторінки) за допомогою event.preventDefault(). Потім створюється об'єкт newUser, який містить введені дані: email і password. Ці дані передаються через метод fetch до API-сервера на шлях /api, де вони обробляються (метод POST). Після отримання відповіді від сервера, форма оновлює стан message, виводячи повідомлення, яке повертається з сервера (повідомлення про успішний або невдалий вхід). Якщо запит успішний, користувач автоматично перенаправляється на сторінку /about за допомогою методу router.push. Якщо сталася помилка при виклику API, в консоль виводиться повідомлення про помилку.

```
const  Login = async (event: React.FormEvent)=>{
    event.preventDefault()
    let newUser = {
        email,
        password
    }
    try {
        const response = await fetch('/api', {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({ email, password }),
        });

        const data = await response.json();
        console.log(data.message);
        setMessege(data.message)
        router.push("/about")
      } catch (err) {
        console.log("Помилка при виклику API");
      }
    console.log({email, password})

}
```

Рисунок 3.12 – Функція Login

Наступним компонентом є NoteComponents. Це динамічний React-компонент, який відображає одну нотаткок з можливістю розгортання вмісту або видалення. Спочатку оголошується стан isExpanded за допомогою стану useState, який контролює, чи компонент розгорнутий (значення true) чи згорнутий (false). Основна функція openNote змінює цей стан на протилежний, дозволяючи користувачеві відкривати та закривати зміст. Усі ці механізми забезпечують гнучкість і динамічність у відображенні нотатоків.

```
  return (
    <div className="node">
      <h3>{note.title}</h3>
      <p className={isExpanded ? "expanded" : "collapsed"}>
        {note.content}
      </p>
      <i
        className={`bx bx-chevron-${isExpanded ? "up" : "right"}`}
        onClick={() => openNote(note.id)}
        role="button"
        aria-label={isExpanded ? "Згорнути" : "Розгорнути"}
      ></i>
      <i className="bx bx-trash" onClick={() => deleteNote(note.id)} role="button" aria-label="Видалити"
    ></i>
    </div>
  );
}
```

Рисунок 3.13 – HTML-розмітка компоненту NoteComponents

Компонент відображає заголовок нотатки note.title і її зміст note.content. Зміст представлено в параграфі з класами expanded (розгорнутий) або collapsed (згорнутий), залежно від стану isExpanded.

Функція deleteNote (рис. 3.14) відповідає за видалення нотатку за допомогою API-запиту. Вона надсилає POST-запит на сервер на маршрут /api/deleteNote, передаючи у тілі запиту id нотатки, яка має бути видалена. Якщо сервер відповідає успішно, сторінка перезавантажується через window.location.reload(), щоб оновити список нотаток і видалити з нього обрану. У разі виникнення помилки функція виводить повідомлення в консоль, що нотатку не вдалося видалити. Такий підхід забезпечує просту інтерактивну взаємодію з нотатками, дозволяючи користувачеві легко

керувати ними, розгортати зміст для читання або видаляти непотрібні елементи.

```
const deleteNote = async (id: number) => {
    try {
        console.log(id)
        const response = await fetch("/api/deleteNote",{
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({id}),
          });
        if (response.ok) {
          // Якщо видалення успішне, перезавантажуємо сторінку
          window.location.reload();
          }
    }catch (error){
        console.log("не вдало видалено нотаток")
    }
};
```

Рисунок 3.14 – Функція deleteNote

Компонент NotesModal є інтерактивним React-компонентом, що використовується для створення нових нотаток через модальне вікно. Використовуючи useState, оголошується кілька станів: modalOpen (відповідає за відкриття/закриття модального вікна), newTitle (для збереження введеного заголовка нотатки), і newContent (для збереження змісту нотатки). Ці стани дозволяють відстежувати дії користувача та оновлювати інтерфейс у режимі реального часу. Основна функція handleAddNote обробляє додавання нового нотатку через API-запит на сервер. Вона надсилає POST-запит на маршрут /api/notes, передаючи дані у форматі JSON. Якщо запит успішний, модальне вікно закривається, і додається новий нотаток. У разі помилки функція виводить повідомлення в консоль.

```
const handleAddNote = async () => {
  try {
    const response = await fetch("/api/notes", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        title: newTitle,
        content: newContent,
      }),
    });
    if (response.ok) {
      const newNote = await response.json();
      setModalOpen(false); // Закриваємо модальне вікно
    } else {
      console.error("Помилка при додаванні нотатки");
    }
  } catch (error) {
    console.error("Помилка при виконанні POST-запиту", error);
  }
};
```

Рисунок 3.15 – Функція handleAddNote

Компонент починається з кнопки для відкриття модального вікна, стилізованої за допомогою класу open-modal-button. Кнопка розміщена всередині контейнера button-container. При натисканні на неї викликається функція setModalOpen(true), яка відкриває модальне вікно, встановлюючи стан modalOpen значення true.

Модальне вікно складається з двох основних частин: фонового накладання (з класом modal-overlay open) і самого вікна (з класом modal open). У модальному вікні знаходяться поля для введення заголовка (input) і змісту нотатку (textarea), значення яких синхронізуються з відповідними станами newTitle та newContent. Після натискання на кнопку "Додати", яка викликає функцію handleAddNote для збереження даних. Якщо користувач вирішить закрити вікно без додавання нотатки, є кнопка "Закрити", яка змінює стан modalOpen на false.

```
<div className="modal-overlay open"></div>
<div className="modal open">
  <h2>Нова Нотаток</h2>
  <input
    type="text"
    placeholder="Заголовок"
    value={newTitle}
    onChange={(e) => setNewTitle(e.target.value)}
    style={{ display: "block", marginBottom: "10px", width: "100%" }}
  />
  <textarea
    placeholder="Зміст"
    value={newContent}
    onChange={(e) => setNewContent(e.target.value)}
    style={{ display: "block", marginBottom: "10px", width: "100%" }}
  ></textarea>
  <button
    onClick={handleAddNote}
    style={{ marginRight: "10px", padding: "10px 20px" }}
  >
    Додати
  </button>
  <button
    onClick={() => setModalOpen(false)}
    style={{ padding: "10px 20px" }}>
    Закрити </button>
```

Рисунок 3.16 – HTML-розмітка модального вікна

Наступна група компонентів це це API-компоненти, які відіграють ключову роль у взаємодії з даними, що зберігаються у форматі JSON. Вони відповідають за отримання, оновлення, створення та видалення інформації, що стосується нотатків, забезпечуючи динамічну роботу додатка. Вони працюють як посередники між клієнтським інтерфейсом та серверною частиною, дозволяючи зручно та ефективно обробляти запити.

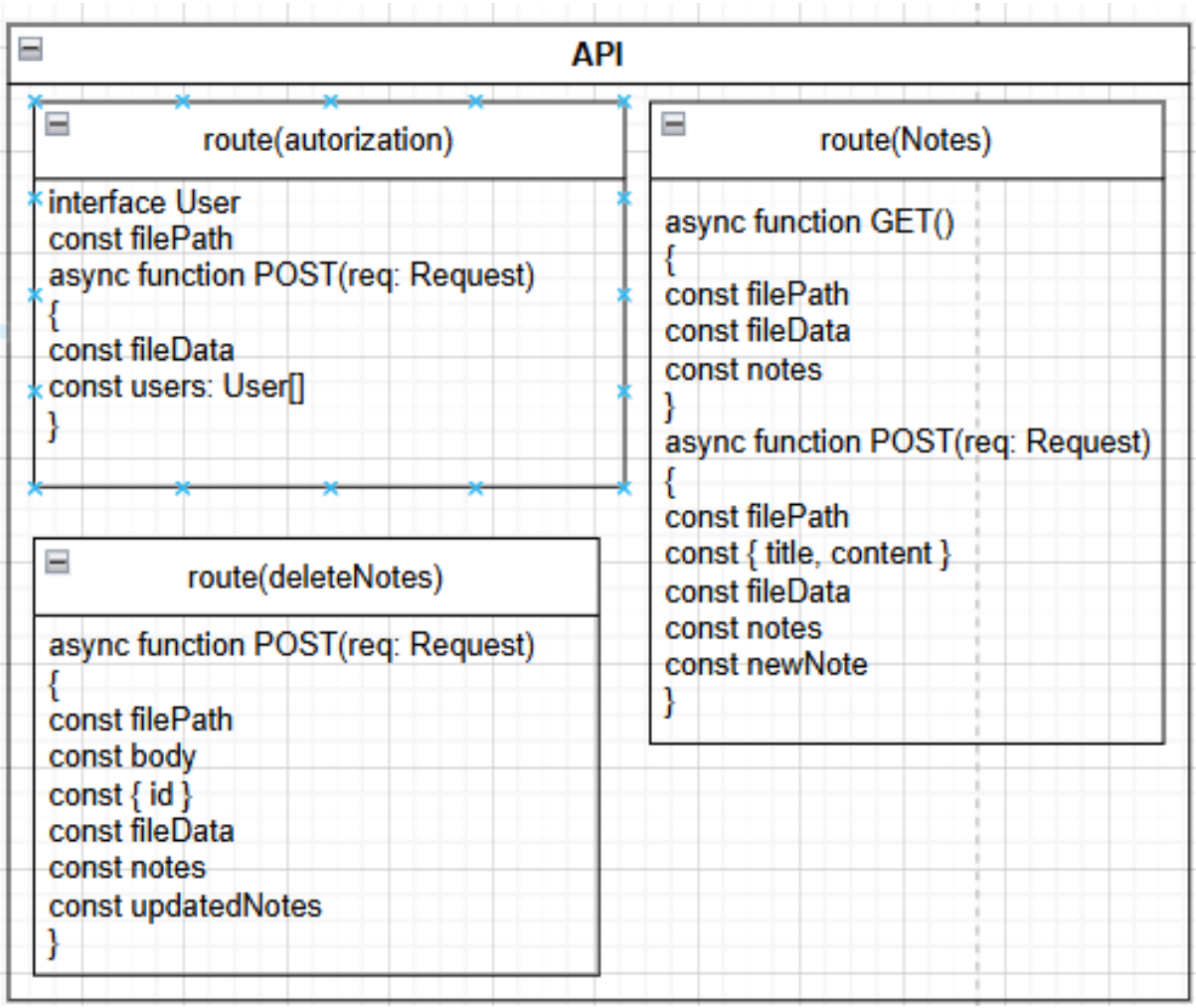

Рисунок 3.17 – Група компонентів API

Виконуються тільки на стороні сервера під час запитів fetch. Зокрема мають виключно функціональну роль та не передають react-компоненти для відображення додаткових елементів.

Перший компонент відповідає за відправлення даних з форми в json-файл – route(authorization). На початку коду імпортуються необхідні модулі: fs для роботи з файловою системою та path для створення абсолютного шляху до файлу з даними. Додатково оголошується інтерфейс User, який визначає структуру об’єкта користувача з полями email і password. Це гарантує, що всі операції над даними користувачів будуть типізовані. Для збереження даних використовується файл users.json, розташований у директорії data, до якого створюється абсолютний шлях за допомогою функції path.join, а process.cwd повертає шлях до поточної робочої директорії це зазвичай коренева

директорія, де зберігається файл package.json. Ця конструкція використовується для надійного і динамічного доступу до файлів, що знаходяться у проєкті: const filePath = path.join(process.cwd(), "data", "users.json").

```
import fs from "fs";
import path from "path";
import { NextApiRequest, NextApiResponse } from "next";

interface User {
  email: string;
  password: string;
}

const filePath = path.join(process.cwd(), "data", "users.json");
```

Рисунок 3.18 – Ініціалізація бібліотеки для роботи з файлами

Функція POST перевіряє тип запиту. Якщо метод запиту правильний, зчитуються дані з тіла запиту за допомогою req.json(), а потім вони розділяються на змінні email і password. Проводиться базова перевірка наявності цих даних: якщо один із параметрів порожній, повертається помилка з кодом 400 і відповідним повідомленням про необхідність заповнення полів.

Далі файл users.json зчитується за допомогою fs.readFileSync, і дані перетворюються на масив об’єктів users за допомогою JSON.parse. До масиву додається новий об’єкт користувача з полями email та password. Після цього оновлений масив записується назад у файл через fs.writeFileSync, зберігаючи дані у форматованому вигляді. Якщо все проходить успішно, повертається відповідь із кодом 200 та повідомленням про успішне додавання користувача. У разі помилок у запиті чи роботі з файлами програма повертає відповідну відповідь, полегшуючи діагностику проблем.

Метод POST використовується для передачі даних із клієнта на сервер, зазвичай для створення нових ресурсів (наприклад, створення запису в базі даних). Коли клієнт надсилає запит POST, дані передаються в тілі запиту (body), що дозволяє передавати великі обсяги інформації у форматі JSON,

XML або навіть у формі файлів. Вданому прикладі метод POST використовується для надсилання інформації про користувача на сервер, де ці дані зберігаються у файлі users.json. Основна перевага POST — це можливість безпечно передавати інформацію, адже дані не відображаються у URL-адресі, як у випадку з GET.

```
export async function POST(req: Request) {
    if (req.method === "POST") {
      const body = await req.json();
      const { email, password } = body;
      console.log({email, password});
      if (!email || !password) {
        return Response.json(
          { message: "Email та пароль обов'язкові" },
          { status: 400 }
        );
      }


      const fileData = fs.readFileSync(filePath, "utf-8");
      const users: User[] = JSON.parse(fileData);

      users.push({ email, password });

      fs.writeFileSync(filePath, JSON.stringify(users, null, 2));

      return Response.json(
          { message: "користувача успішно додано" },
          { status: 200 }
        );
```

Рисунок 3.19 – Метод POST

Метод GET використовується для отримання інформації з сервера, яка зазвичай повертається у форматі JSON, HTML або тексту. У GET-запитах параметри можуть бути передані через URL (у вигляді рядка запиту — query string). GET-запити є ідеальними для операцій, які не змінюють стан сервера, для отримання списку нотатків, користувачів чи іншої інформації.

Наступний API-компонент route(notes) використовує обидва ці методи. Метод GET у цьому API використовується для отримання списку нотаток, які зберігаються у файлі notes.json. Отриманий рядок з fileData парсується у JavaScript-об'єкт за допомогою JSON.parse. На завершення дані повертаються у вигляді JSON-відповіді через NextResponse.json(). Таким чином, цей метод дозволяє клієнту отримати всі існуючі нотатки.

```
export async function GET() {
  const filePath = path.join(process.cwd(), "data", "notes.json");
  const fileData = await fs.readFile(filePath, "utf-8");
  const notes = JSON.parse(fileData);

  return NextResponse.json(notes);
}
```

Рисунок 3.20 – Метод GET в route(notes)

Метод POST використовується для додавання нового нотатку до файлу notes.json. Спочатку визначається шлях до файлу аналогічно до методу GET. Далі з тіла запиту (через req.json()) отримуються дані, які містять title (заголовок) і content (зміст). Якщо ці поля порожні, сервер повертає відповідь із кодом статусу 400 (поганий запит) і повідомленням про помилку. Після цього файл notes.json зчитується, щоб завантажити існуючі нотатки. Новий запис створюється як об'єкт із унікальним ідентифікатором (id), який визначається на основі кількості існуючих нотатків (довжини масиву). Новий нотаток додається до масиву, вже потім зберігається у файлі через fs.writeFile. У разі успіху сервер повертає JSON-об'єкт нової нотатки з кодом статусу 201 (створено).

```
export async function POST(req: Request) {
    try {
      const filePath = path.join(process.cwd(), "data", "notes.json");
      const { title, content } = await req.json();

      if (!title || !content) {
        return Response.json(
          { message: "Заголовок і зміст є обов'язковими" },
          { status: 400 }
        );
      }

      // Читання існуючих нотатків
      const fileData = await fs.readFile(filePath, "utf-8");
      const notes = JSON.parse(fileData);

      // Додавання нової нотатки
      const newNote = {
        id: notes.length + 1,
        title,
        content,
      };
      notes.push(newNote);

      // Збереження оновленого масиву нотатків у файл
      await fs.writeFile(filePath, JSON.stringify(notes, null, 2));
```

Рисунок 3.21 – Метод POST в route(notes)

Останій компонент route(deleteNotes), відповідає за видалення нотатку з json-файлу за допомогою filter видаляється елемент, у якого id збігається із отриманим з тіла запиту. Оновлений масив нотатків зберігається назад у файл через fs.writeFile, причому використовується форматування JSON для зручності читання. Якщо всі кроки виконані успішно, сервер повертає JSON-відповідь із повідомленням «Нотаток успішно видалено».

```
import fs from "fs/promises";
import path from "path";

export async function POST(req: Request) {
    try {
        const filePath = path.join(process.cwd(), "data", "notes.json");
        if (req.method === "POST") {
            const body = await req.json();
            const { id } = body;
            const fileData = await fs.readFile(filePath, "utf-8");
            const notes = JSON.parse(fileData);
            const updatedNotes = notes.filter((note: { id: number }) => note.id !== id);
            await fs.writeFile(filePath, JSON.stringify(updatedNotes, null, 2));

        return Response.json({ message: "Нотаток успішно видалено" }, { status: 200 });
        }
    } catch (error) {
        return Response.json(
          { message: "Помилка при обробці запиту" },
          { status: 500 }
        );
    }
}
```

Рисунок 3.22 – Метод POST route(deleteNotes)

3.4 Стилізація веб-додатку

Для дотримання одного стилю всіх сторінок, використовується імпортування import "../styles/globals.css". Даний стиль використовується в компоненті layout, що відповідає за глобальне налаштування вигляду веб додатку. Стилі використовуються для відображення елементу інтерфейсу NavBar.

Навігаційний елемент інтерфейсу NavBar із класом className="navbar" стилізується за допомогою низки CSS-правил, які забезпечують сучасний і зручний дизайн. Загальні налаштування «*» встановлюють універсальну

модель розрахунку розмірів елементів box-sizing: border-box і переформатовують стандартні відступи, а margin: auto в класі .main та встановлює по центрує основний контейнер. Для очищення списків і посилань використовуються стилі «li». Hover додає інтерактивності, змінюючи колір тексту при наведенні. Фон заголовку header створює градієнт додаючи візуальної привабливості, а також має бічні відступи для дотримання пропорцій.

Сам NavBar оформлений як гнучкий контейнер із властивістю display: flex, що забезпечує горизонтальне розташування елементів, вирівняних по вертикалі (align-items: center) і розподілених на весь простір (justify-content: space-between).

```
header{
    background: rgb(170,177,177);
    background: linear-gradient(90deg, rgba(170,177,177,1) 0%,
     rgba(247,233,146,1) 52%, rgba(157,239,255,1) 100%);
    position: relative;
    padding: 0 2rem;
}
.navbar{
    width: 100%;
    height: 60px;
    display: flex;
    align-items: center;
    justify-content: space-between;
}
.logo{
    font-size: 1.5rem;
    font-weight: bold;
}
.link{
    display: flex;
    gap: 2rem;
}
```

Рисунок 3.23 – Стилізація верхнього меню сторінки

Клас .logo задає стиль для логотипу, роблячи його помітним завдяки збільшеному розміру шрифту (font-size: 1.5rem) та жирному накресленню (font-weight: bold), тоді як .link розташовує посилання рівномірно, задаючи між ними проміжок у «2rem» за допомогою властивості gap. Загальний дизайн

передбачає легкість, читабельність та естетичний вигляд, що підходить для використання у сучасних веб-інтерфейсах.

Рисунок 3.24 – Дизайн елементу NavBar

Наступний дизайн потрібно розглянути, це стилізація форми авторизації. Для цього використовується наступні стилі: globals.css (частково) та formstyle.css.

Контейнер форми з класом .form займає всю висоту вікна браузера (min-height: 100vh), вирівнюється по центру завдяки властивостям display: flex, justify-content: center та align-items: center, а фоном слугує зображення, яке розташовується по центру (background-position: center) та розтягується на весь екран (background-size: cover). Внутрішній блок .wrapper створює ефект прозорості завдяки backdrop-filter: blur(20px) і має округлі кути (border-radius: 10px), що додає м'якості дизайну. Його напівпрозорий фон та контур (border: 2px solid rgba(255, 255, 255, .2)) разом із текстом білого кольору (color: #fff) створюють стильний контраст із фоновим зображенням.

```css
.form{
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
    background: url(../public/notatki-photo.jpeg) no-repeat;
    background-size: cover;
    background-position: center;
}

.wrapper{
    width: 420px;
    background: transparent;
    border: 2px solid rgba(255, 255, 255, .2);
    backdrop-filter: blur(20px);
    color: #fff;
    border-radius: 10px;
    padding: 30px 40px;
}
```

Рисунок 3.25 – Стилізація форми в formstyle.css

Заголовок h1 у .wrapper вирівняний по центру (text-align: center) і має великий розмір шрифту (font-size: 36px). Поля вводу стилізовані в блоці .int-box: вони мають округлу форму (border-radius: 40px), напівпрозору рамку (border: 2px solid rgba(255, 255, 255, .2)), та відступи для зручності введення тексту. Користувач бачить текст-підказку завдяки стилізації input::placeholder, яка відображає текст білого кольору. Іконки, розташовані праворуч від полів вводу (.int-box i), вирівнюються по вертикалі за допомогою transform: translateY(-50%), додаючи візуальної ідентифікації для користувача.

Кнопка форми з класом .btn має округлу форму (border-radius: 40px), заповнений білий фон (background: #fff) і стиль без обведення (border: none), що забезпечує мінімалістичний вигляд. Додатково, форма забезпечує інтерактивність за рахунок динамічного відображення повідомлень ({message && {message} }) під час роботи з формою.

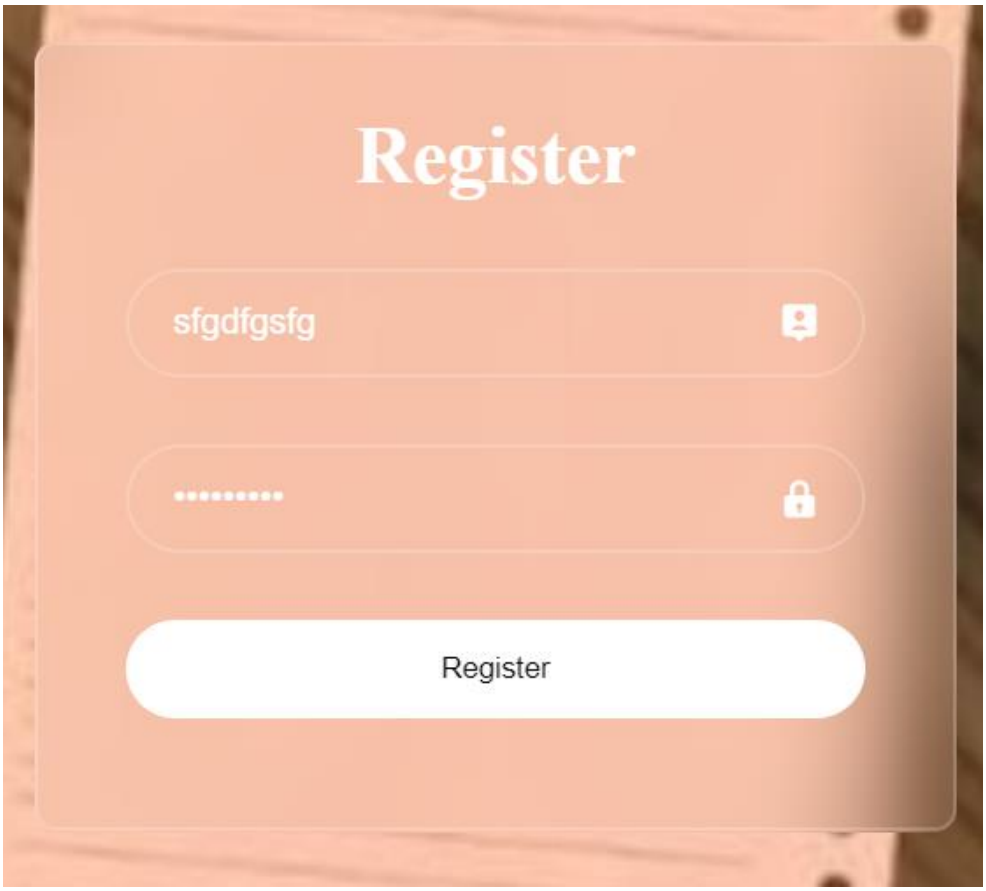

Рисунок 3.26 – Дизайн форми

Більш складнішу стилізацію має сторінка нотатків, яка має декілька react-компонентів, які також мають власні стилі. Спочатку потрібно розглянути основу сторінки нотатків. Для стилізації використовується notes.css.

Дизайн основи сторінки нотатків побудований із використанням динамічного градієнтного фону, інтерактивних елементів і ефектів для створення візуально привабливого інтерфейсу. Основний контейнер сторінки, позначений класом .bg-color-gr, має повноекранний розмір (height: 100vh, width: 100%) і фон із плавним градієнтом, який анімується (background: linear-gradient(...) та @keyframes gradientAnimation). Анімація змінює позиції кольорів, створюючи ефект руху градієнта кожні 10 секунд, додаючи динаміки та сучасності дизайну. Елемент ::before має блюр (filter: blur(20px)) і розташовується позаду контенту завдяки z-index: -1, створюючи враження глибини.

Блок із класом container містить елемент typing-effect, який, реалізує ефект машинопису (анімований текст "Керуй власним життям!"). Це додає інтерактивності та оживляє сторінку.

```css
.bg-color-gr{
    position: relative;
    height: 100vh;
    width: 100%;
    background: linear-gradient(45deg, #bc98ae, #8d8259, #e6fc98, #7cabfb);
    background-size: 400% 400%;
    animation: gradientAnimation 10s ease infinite;
}
.bg-color-gr::before{
    content: "";
    position: absolute;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background: inherit;
    filter: blur(20px); /* Рівень блюру */
    z-index: -1; /* Для того, щоб блюр був позаду контенту */
}
/* Анімація для фонового градієнта */
@keyframes gradientAnimation {
    0% {
      background-position: 0% 20%;
    }
    20%{
        background-position: 20% 40%;
    }
```

Рисунок 3.27 – Стилізація основи сторінки notes.css

На даній сторінці присутні 2 react-компоненти ExpandableNote та NotesModal та мають імпортовані стилі з notes.css

Блоки нотатків групуються у контейнері з класом .notates, який використовує flexbox для розташування нотатків у ряд із автоматичним перенесенням (flex-wrap: wrap), рівномірним розподілом (justify-content: center) та проміжками між ними (gap: 20px), забезпечуючи адаптивність дизайну.

Кожен нотаток (.node) має білий фон (background-color: #fff), округлі кути (border-radius: 10px) і привабливий ефект тіні (box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1)), що додає глибини. Ефекти ховера на нотатках підкреслюють інтерактивність: при наведенні піднімається (transform: translateY(-5px)) і тінь стає виразнішою (box-shadow: 0 6px 12px rgba(0, 0, 0, 0.15)).

Основна функціональність нотатку реалізована через динамічне відображення тексту: зміст (<p>) за замовчуванням обмежений трьома рядками (-webkit-line-clamp: 3, overflow: hidden), що забезпечує компактний вигляд нотатку. При розгортанні (.expanded) текст відображається повністю (display: block). Іконки дій (<i>) інтерактивні: одна відповідає за розгортання/згортання нотатку, а друга — за його видалення. Вони мають позицію absolute, розташовані у верхньому правому куті нотатку, а їх колір (color: #007bff) і плавний ефект збільшення при ховері (transform: scale(1.2)) забезпечують зручність взаємодії.

```
.notates{
    display: flex;
    flex-wrap: wrap;
    gap: 20px;
    justify-content: center;
    margin-top: 20px;
}
.node {
    position: relative;
    border: 1px solid #ddd;
    border-radius: 10px;
    padding: 20px;
    width: 250px;
    background-color: #fff;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    transition: transform 0.3s ease, box-shadow 0.3s ease;
    word-wrap: break-word;
    overflow-wrap: break-word;
}
```

Рисунок 3.28 – Стилізація нотатків

Модальне вікно NotesModal створює сучасний та інтуїтивний інтерфейс для додавання нотаток. Кнопка відкриття модального вікна (.open-modal-button) має привабливий вигляд із закругленими краями, насиченим синім кольором та інтерактивним ефектом підйому при ховері. При відкритті з'являється напівпрозорий фон (.modal-overlay), який приглушує основний контент за допомогою затемненого шару з плавним переходом (opacity), привертаючи увагу до центрального елемента. Саме вікно (.modal) розташовується посередині екрана з використанням position: fixed та трансформації для вирівнювання, має білий фон. У вікні присутні текстові поля (input і textarea) для введення заголовку та змісту нотатку, які мають мінімалістичний дизайн із заокругленими краями та м'яким кольором фону, а також підсвічуються синім при фокусі для акценту на активному полі. Два інтерактивні кнопкові елементи дозволяють додати новий або закрити модальне вікно.

```css
.modal{
    position: fixed;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    background-color: #fff;
    border-radius: 10px;
    padding: 30px;
    border-radius: 5px;
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
    z-index: 1000;
    max-width: 90%;
    opacity: 0;
    transition: opacity 0.3s ease-in-out;
}
.modal.open {
    opacity: 1;
  }

.modal input,
.modal textarea {
  width: 100%;
  padding: 10px;
```

Рисунок 3.29 – Стилізація модального вікна

3.5 Тестування додатку

Для тестування додатку використовується термінал в Visual Studio Code. За допомогою команди npm run dev, завантажуємо сервер для відображення веб додатку в браузері:

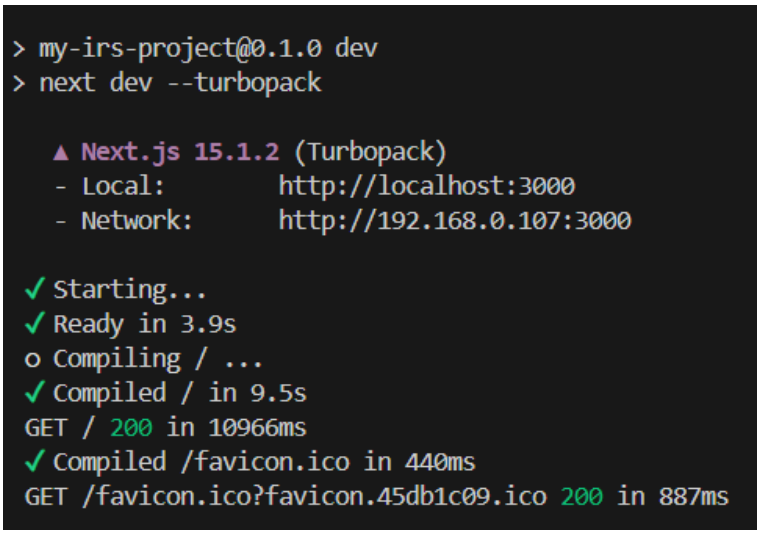

Рисунок 3.30 – Завантаження проекту

Під час завантаження термінал надає url-адрес, завдяки якому користувач може відкрити додаток у будь-якому браузері: http://localhost:3000.

Після завантаження додатку з’явиться сторінка авторизації, де розташоване верхнє меню для навігації в середині додатку та форма авторизації, яка має поля для введення даних.

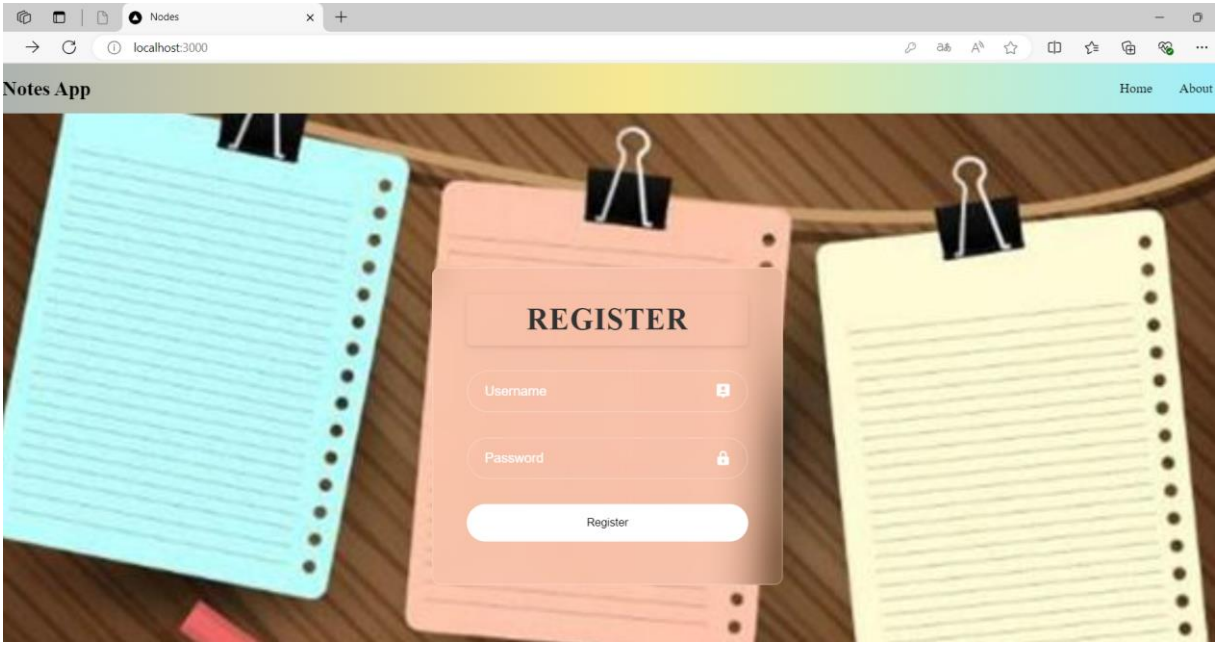

Рисунок 3.31 – Сторінка авторизації

Щоб перейти на основну сторінку додатку потрібно ввести відповідні дані користувача:

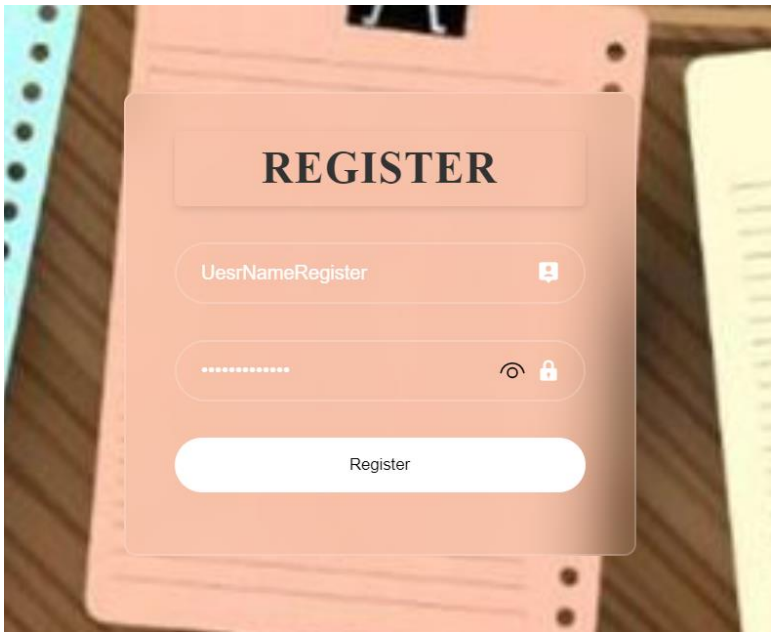

Рисунок 3.32 – Введення даних користувача

Після введення даних користувач потрапляє на сторінку де розташовані нотатки, саме для неї використовується алгоритм ISR.

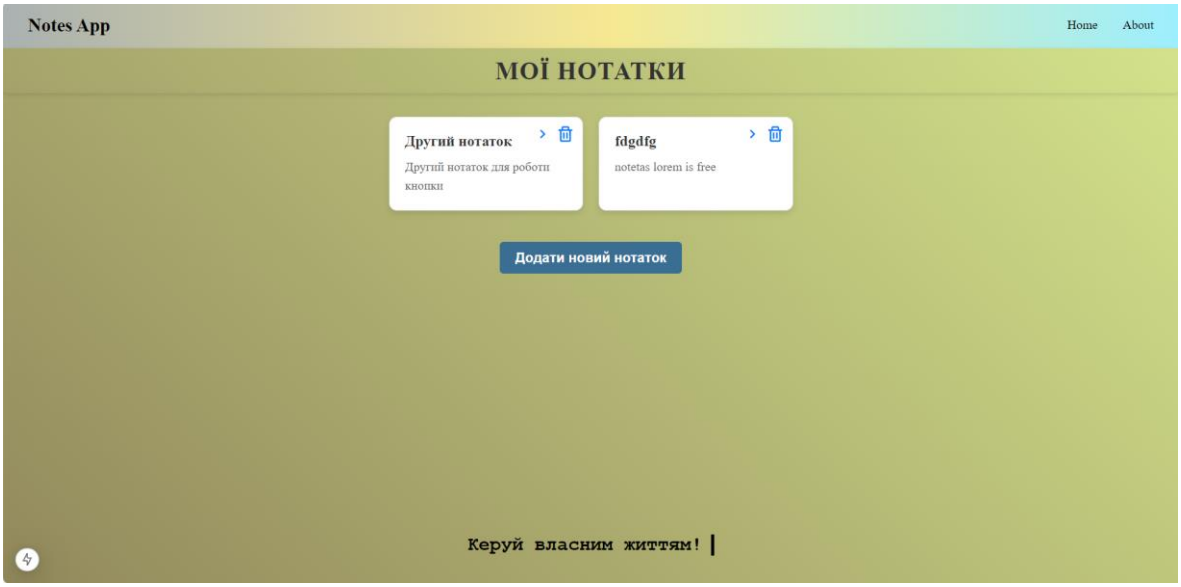

Рисунок 3.33 – Сторінка нотатків

На даній сторінці користувач може взаємодіяти з нотатками, отримати повний зміст запису та видалити. Для тестування цього функціоналу, спочатку створимо новий нотаток, для цього потрібно натиснути на кнопку «Додати новий нотаток». Після цього з’явиться модальне вікно, в якому користувач може створити новий нотаток, заповнивши відповідні поля:

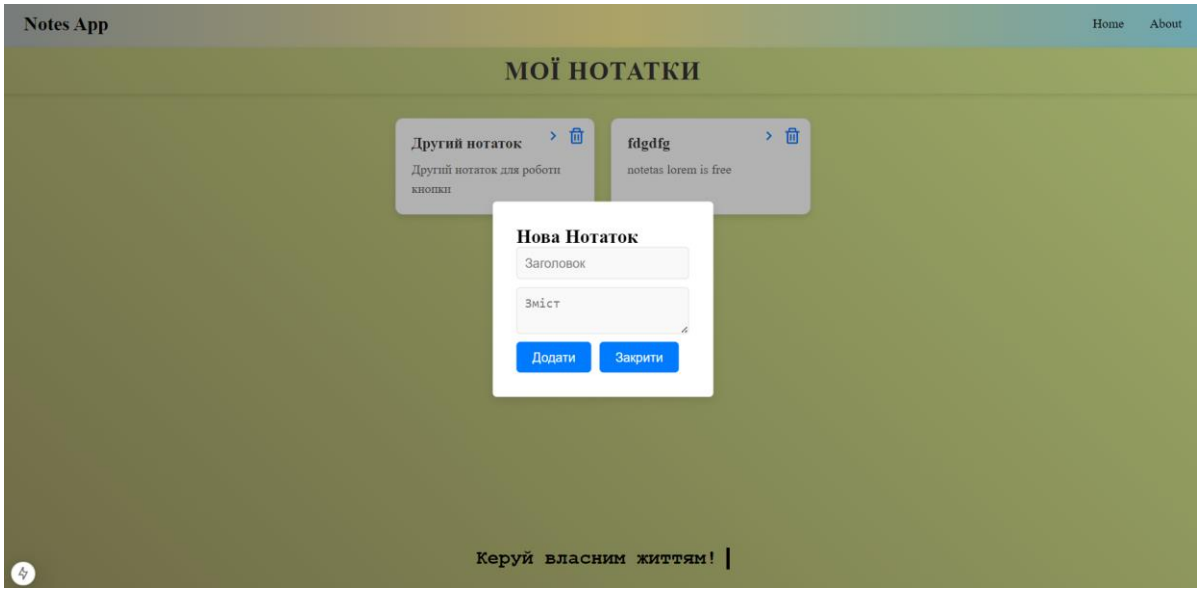

Рисунок 3.34 – Поява модального вікна

За бажанням користувача, це вікно можна розтягнути, для великого обсягу тексту:

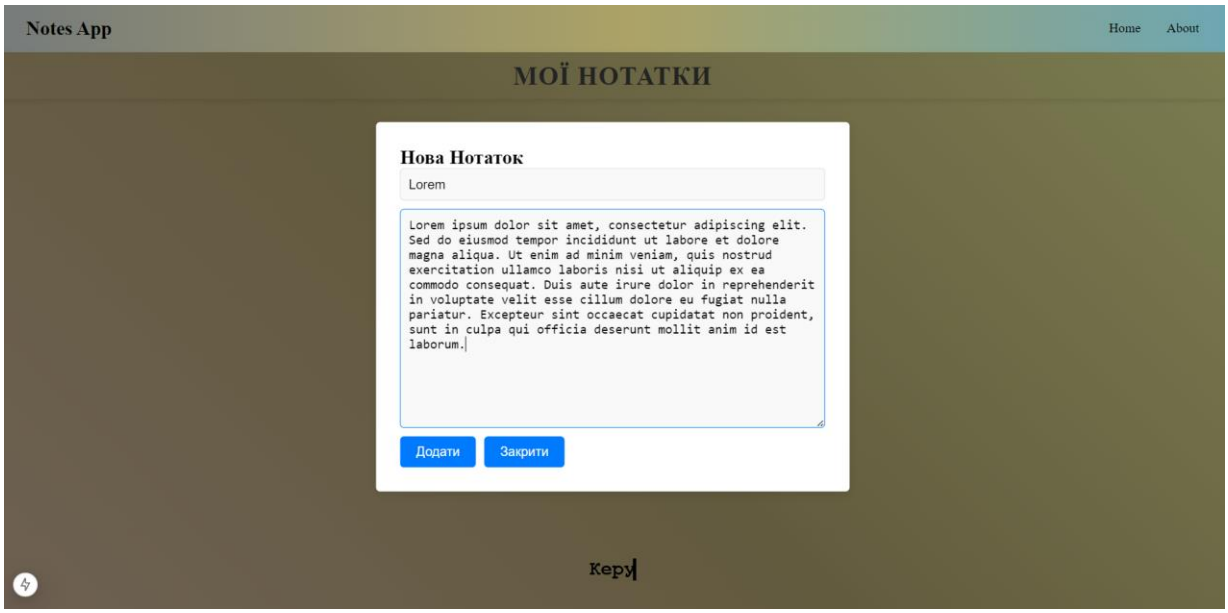

Рисунок 3.35 – Заповнення полів

Далі потрібно натиснути кнопку «Додати», щоб новий нотаток потрапив в базу даних та з’явився на основній сторінці.

Рисунок 3.36 – Поява нового нотатку

Користувач за бажанням може переглянути повний вміст нотатку, натиснувши на відповідну кнопку з верхнього-правого нотатку:

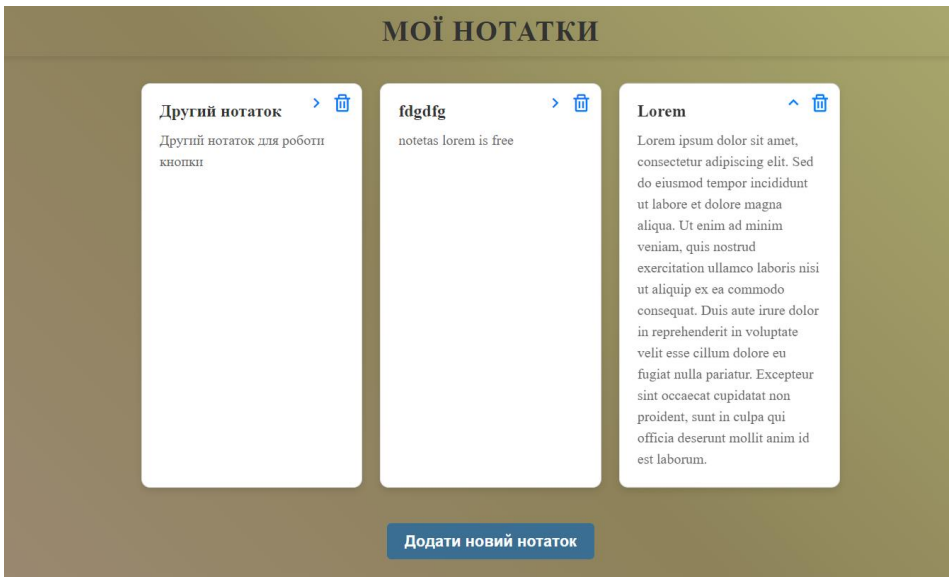

Рисунок 3.37 – Відображення повного змісту всіх нотатків

При бажанні користувач може видалити не потрібний додаток, теж натиснувши на відповідну кнопку біля певного нотатку:

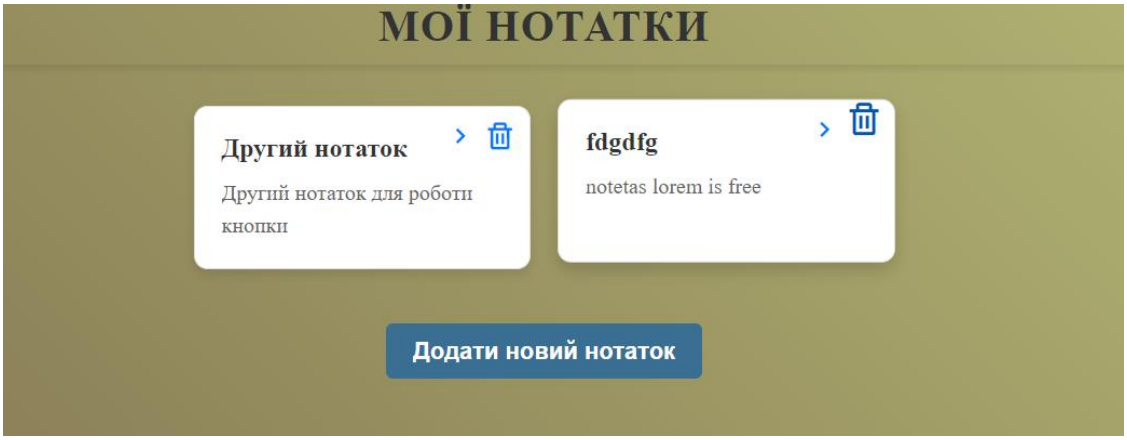

Рисунок 3.38 – Видалення нотатку

3.6 Висновок до третього розділу

У третьому розділі було здійснено розробку веб-застосунку для управління нотатками з використанням технології Incremental Static Regeneration (ISR). Було детально описано процес створення архітектури застосунку, включаючи клієнтську та серверну частини.

Клієнтська частина була реалізована за допомогою бібліотеки React.js з використанням компонентного підходу, що забезпечило модульність, зручність у підтримці та розширюваність коду. Для покращення читабельності та зниження кількості помилок було використано TypeScript, який забезпечив статичну типізацію даних.

Серверна частина була реалізована на базі фреймворку Next.js, який забезпечив підтримку серверного рендерингу (SSR) та інкрементальної статичної генерації (ISR). Було впроваджено систему API для обробки HTTP-запитів, включаючи створення, редагування, видалення та пошук нотаток.

Особливу увагу було приділено технології Incremental Static Regeneration, яка дозволила поєднати переваги статичного рендерингу з можливістю оновлення контенту в реальному часі, що значно підвищило продуктивність веб-застосунку.

ВИСНОВОК

В процесі розробки веб-застосунку для управління нотатками з використанням технології Incremental Static Regeneration (ISR) було виконано комплексну роботу, що включала створення архітектури проєкту, розробку дизайну, реалізацію клієнтської та серверної частин, а також тестування функціональності застосунку. Проєкт передбачав створення ефективної системи для зберігання, обробки та управління нотатками, яка забезпечує високу продуктивність і зручність користувача.

Основними завданнями, які були вирішені під час виконання роботи, стали:

1. Проведення аналізу сучасних підходів до ведення нотаток, як традиційних, так і цифрових.
2. Обґрунтування вибору стеку технологій для реалізації (Next.js, React.js, Node.js, TypeScript).
3. Розробка вимог до програмного забезпечення, включаючи функціональні (створення, редагування, видалення нотаток, пошук за ключовими словами, фільтрація за мітками, аутентифікація) та нефункціональні (продуктивність, безпека, зручність використання).
4. Створення клієнтської частини із використанням бібліотеки React.js, що забезпечило гнучкий та модульний підхід до побудови інтерфейсу.
5. Реалізація серверної частини на основі Next.js із підтримкою ISR, що дозволило значно покращити швидкість завантаження сторінок.
6. Тестування програмного продукту на відповідність функціональним вимогам та критеріям продуктивності.

Було проведено дослідження процесу конспектування в академічному та професійному середовищі, а також впливу цифрових технологій на ефективність навчання. Було проаналізовано традиційні методи ведення нотаток, такі як лінійне конспектування, метод Корнелла, ментальні карти та метод контурів. Визначено, що кожен із підходів має свої переваги та

недоліки, залежно від індивідуальних особливостей студентів і стилю викладання.

Особливу увагу приділено цифровим інструментам для конспектування, включаючи OneNote, Notion, Evernote та інші застосунки, що забезпечують текстове та мультимедійне нотування, а також функції пошуку, анотування і спільного доступу до даних. Розглянуто як переваги використання цифрових інструментів, зокрема полегшення доступу до матеріалів та покращення організації нотаток, так і потенційні недоліки, серед яких зниження когнітивної обробки інформації та ризик цифрової перевантаженості.

Дизайн застосунку було створено із застосуванням сучасних підходів, що забезпечують зручний доступ до функціональності системи та покращують користувацький досвід. Використання технології ISR сприяло досягненню високої швидкості завантаження контенту та забезпечило динамічне оновлення даних без повного перезавантаження сторінки.

Здійснено ґрунтовний аналіз засобів та інструментів для створення веб-застосунку для ведення нотатків. Основну увагу приділено обґрунтуванню вибору технологічного стеку, здатного забезпечити стабільну роботу системи, її продуктивність, масштабованість та відповідність сучасним вимогам безпеки й зручності використання.

У процесі дослідження було розглянуто низку технологічних підходів до розробки веб-застосунків, зокрема стеків MEAN, MERN, LAMP та JAMstack. Аналіз цих рішень дозволив встановити, що для поставлених цілей найбільш оптимальним є використання технологій Next.js, React.js, Node.js та TypeScript, які забезпечують єдину мову програмування на клієнтській та серверній частинах, підтримку серверного рендерингу (SSR) та використання інкрементальної статичної генерації (ISR).

Важливою частиною дослідження стало формування вимог до програмного забезпечення, серед яких:

- Функціональні вимоги: можливість створення, редагування, видалення нотаток, пошуку за ключовими словами, фільтрації за мітками, а також базова аутентифікація користувачів.
- Нефункціональні вимоги: забезпечення високої продуктивності системи, надійності зберігання даних, захисту інформації користувачів, а також зручності інтерфейсу для кінцевих користувачів.

Було обґрунтовано архітектурний підхід до розробки веб-застосунку. Клієнтська частина створена з використанням бібліотеки React.js, що забезпечує компонентну структуру та сприяє повторному використанню елементів інтерфейсу. Серверна частина реалізована за допомогою Next.js, що дозволяє обробляти HTTP-запити через API-ендпойнти та підтримує рендеринг на стороні сервера. Для зберігання даних обрано файлову систему у форматі JSON, що забезпечує простоту інтеграції з основними функціями застосунку.

Здійснено вибір сучасного технологічного стеку для реалізації веб-застосунку, визначено функціональні та нефункціональні вимоги до системи, а також описано архітектурний підхід, який забезпечує ефективну роботу програмного продукту та створює основу для подальшої практичної реалізації.

Тестування показало стабільну роботу застосунку та відповідність всім заявленим функціональним та нефункціональним вимогам. Проєкт є готовим до використання в освітньому та професійному середовищі для управління нотатками та організації інформації.

Отже, поставлена мета щодо розробки веб-застосунку для управління нотатками була досягнута, а розроблений програмний продукт відповідає сучасним вимогам щодо ефективності, зручності та продуктивності.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Мосій, Л.Є., Струтинська, І.В., & Козбур, Г.В. (2023). Цифрова трансформація: успішний досвід Європи. У Матеріали LII Науково-технічної конференції підрозділів ВНТУ (стор. [вказати номери сторінок]). Вінниця: ВНТУ.

2. Piolat A, Olive T, Kellogg RT. Cognitive effort during note-taking. Appl Cognit Psychol. 2005; 19:291-312.

3. What’s the Difference Between Frontend and Backend in Application Development? – [Електронний ресурс] – Режим доступу: http://surl.li/udvcr

4. Next js Introduction. – [Електронний ресурс] – Режим доступу: https://nextjs.org/docs.

5. What is Node.js (Node)? – [Електронний ресурс] – Режим доступу: https://www.techtarget.com/whatis/definition/Nodejs#:~:text=js%20(Node)%20is%20an%20Open,to%20learn%20an%20additional%20language..

6. Legacy Reactjs. – [Електронний ресурс] – Режим доступу: https://legacy.reactjs.org.

7. TypeScript for JavaScript Programmers. – [Електронний ресурс] – Режим доступу: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html.

8. Web Application Architecture in 2024: Moving in the Right Direction. – [Електронний ресурс] – Режим доступу: https://mobidev.biz/blog/web-application-architecture-types.

9. What is Client-Server Architecture? Explained in Detail. – [Електронний ресурс] – Режим доступу: https://www.theknowledgeacademy.com/blog/client-server-architecture/.

10. Incremental Static Regeneration (ISR) in Next.js 15 with App Router – [Електронний ресурс] – Режим доступу: https://slateblog.netlify.app/articles/incremental-static-regeneration-isr-in-next-js-15-with-app-router?utm_source=chatgpt.com

11. Incremental Static Regeneration (ISR) – [Електронний ресурс] – Режим доступу: https://nextjs.org/docs/pages/building-your-application/data-fetching/incremental-static-regeneration?utm_source=chatgpt.com

12. Introducing JSON – [Електронний ресурс] – Режим доступу: https://www.json.org/json-en.html

13. JSON – [Електронний ресурс] – Режим доступу: http://surl.li/uvjkba

14. Dynamic scripting with JavaScript – [Електронний ресурс] – Режим доступу: http://surl.li/vrgeqi

15. CSS styling basics – [Електронний ресурс] – Режим доступу: http://surl.li/vaxkpu