КВАЛІФІКАЦІЙНА РОБОТА

Група ІПЗс-20-1
Мартиновський А.Р.

2024

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

на правах рукопису

Мартиновський Артур Русланович

УДК 004.4

Розробка веб-сайту для організації розкладу подій з включенням інформації про погоду

Спеціальність 121 – «Інженерія програмного забезпечення»

Кваліфікаційна робота на здобуття кваліфікації бакалавр

Нормоконтроль	Студент
_____Стисло О.В.	_____Мартиновський А.Р.
(підпис, дата, розшифрування підпису)	(підпис, дата, розшифрування підпису)
Допускається до захисту	Керівник роботи
Завідувач кафедри	
	к.т.н., проф. каф. ІТ
_____к.т.н., доц. Ващишак С.П.	_____ Пашкевич О.П.
(підпис, дата, розшифрування підпису)	(підпис, дата, розшифрування підпису)

Івано-Франківськ – 2024

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

Освітній ступінь: «бакалавр»

Спеціальність: 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____

«____ » _____ 2024 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Мартиновський Артур Русланович

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:

Сайт для планування справ з інформацією про погоду

керівник роботи:

Пашкевич Олег Петрович, к.т.н., проф. каф. ІТ

затверджена наказом вищого навчального закладу від « 12 » березня 2024 року № 19/1

2. Термін подання студентом роботи 05.06.2024
3. Вихідні дані роботи: мова програмування JavaScript, ReactJs,NextJs, HTML
4. Зміст кваліфікаційної роботи (перелік питань, які потрібно розробити)
   1. Аналіз рішень в області розробки веб додатків
   2. Обґрунтування вибору інструментів розробки
   3. Реалізація веб додатку
5. Дата видачі завдання 14.03.2024

КОНСУЛЬТАНТИ РОЗДІЛІВ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Розділ	Консультант (прізвище, ініціали та посада)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз і вивчення аналогів	16.03.2024	Виконано
2	Вибір мови програмування	20.03.2024	Виконано
3	Вибір платформи для розробки	22.03.2024	Виконано
4	Розроблення архітектури додатку	03.04.2024	Виконано
5	Розробка веб додатку	28.04.2024	Виконано
6	Перевірка та виправлення помилок	23.05.2024	Виконано
7	Підготовка до захисту	24.05.2024	Виконано

Студент _____ Мартиновський А.Р.

(підпис) (прізвище та ініціали)

Керівник роботи _____ Пашкевич О.П.

(підпис) (прізвище та ініціали)

Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

АНОТАЦІЯ

Проект "TodoWeth" - це веб-сайт, розроблений за допомогою NextJS, який об'єднує у собі функціональність todo-списку та інформацію про погоду. Завдяки зручному інтерфейсу та розширеній функціональності, користувач може легко керувати своїми завданнями та одночасно відслідковувати погодні умови. Веб-сайт має дві складові: клієнтську та серверну частини. Серверна частина використовує три API-шляхи: для авторизації користувачів, отримання інформації про користувача та завдання. Користувач може зареєструватися або увійти в систему, використовуючи електронну пошту або обліковий запис Google або Facebook. На головній сторінці користувачу доступні сторінки Weather та Tasks. На сторінці Weather він може переглянути погоду на день та прогноз на наступні 7 днів. Сторінка Tasks надає можливість створювати, відстежувати та редагувати завдання. Для забезпечення кращого користувацького досвіду, веб-сайт має анімаційні ефекти та оптимізовані переходи між сторінками. Крім того, для керування формами використовуються бібліотеки zod та react-hook-form, а для автентифікації - next-auth.

Загалом, "TodoWeth" - це інноваційний та зручний інструмент для керування завданнями та відстеження погодних умов, що забезпечує зручність та ефективність в щоденному житті користувачів.

КЛЮЧОВІ СЛОВА: TODO, ПОГОДА, NEXTJS, ВЕБ-САЙТ, СПИСОК СПРАВ, API, АВТОРИЗАЦІЯ, ПРОГНОЗ ПОГОДИ, СТОРІНКА ЗАВДАНЬ, КЕРУВАННЯ ЧАСОМ, ZOD, REACT-HOOK-FORM, NEXT-AUTH, АУТЕНТИФІКАЦІЯ, ЗРУЧНІСТЬ, ЕФЕКТИВНІСТЬ.

SUMMARY

The "TodoWeth" project is a web application developed using NextJS, which combines the functionality of a to-do list and weather information. With a user-friendly interface and enhanced functionality, users can easily manage their tasks while simultaneously tracking weather conditions. The website consists of both client-side and server-side components. The server-side utilizes three API routes: for user authentication, fetching user information, and managing tasks. Users can register or log in using their email or through Google or Facebook accounts. On the homepage, users have access to the Weather and Tasks pages. On the Weather page, they can view the weather for the day and a forecast for the next 7 days. The Tasks page allows users to create, track, and edit tasks. To enhance the user experience, the website features animated effects and optimized transitions between pages. Additionally, zod and react-hook-form libraries are used for form management, while next-auth handles authentication.

Overall, "TodoWeth" is an innovative and convenient tool for task management and weather tracking, providing users with convenience and efficiency in their daily lives.

KEYWORDS: TODO, WEATHER, NEXTJS, WEB-SITE, TODO LIST, WEATHER, API, AUTHORIZATION, WEATHER FORECAST, TRACK TASKS, ZOD, REACT-HOOK-FORM, NEXT-AUTH, AUTHENTICATION, CONVENIENCE, EFFICIENC

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Program Interface

IDE – інтегроване середовище розробки

ОС – операційна система

ПЗ – програмне забезпечення

VSC – Visual Studio Code

DB – база даних

ВСТУП

Актуальність теми. В сучасному світі розвиток інформаційних технологій впливає на усі сфери життя, включаючи організацію робочого та особистого часу. Виникає потреба у зручних та ефективних інструментах для планування задач та контролю за ними. Однією з найпоширеніших проблем є неефективне керування часом та завданнями, що може призводити до стресу та невиконання поставлених цілей. У зв'язку з цим, розробка веб-додатку, який поєднує в собі функції календаря, списку завдань та інформації про погоду, має велику актуальність для сучасного користувача. Такий додаток дозволить ефективніше планувати робочий та відпочинковий час, отримуючи всю необхідну інформацію на одному ресурсі. Таким чином, дана кваліфікаційна робота має важливе значення для практичного використання та подальшого розвитку в галузі управління часом та завданнями.

Мета і завдання дослідження. Метою цієї кваліфікаційної роботи є розробка та реалізація веб-додатку для керування завданнями з інтеграцією погодних даних, який дозволить користувачам ефективно планувати свій час та задачі, враховуючи погодні умови. З метою досягнення поставленої мети необхідно вирішити наступні завдання:

- проаналізувати потреби користувачів та визначити основні функціональні вимоги до додатку;
- обрати оптимальні технології та інструменти для розробки клієнтської та серверної частин додатку;
- розробити інтуїтивний інтерфейс користувача, що включатиме функції календаря, списку завдань та інформації про погоду;
- забезпечити взаємодію з базою даних для зберігання інформації про користувачів, їх завдання та налаштування;

– інтегрувати зовнішні сервіси для отримання актуальної погодної інформації;

– провести тестування та валідацію розробленого додатку для забезпечення його надійності та ефективності в реальних умовах використання;

– підготувати документацію та забезпечити можливість подальшого розвитку та підтримки додатку.

Об’єкт та предмет дослідження. Об'єктом дослідження є процес керування завданнями та планування робочих задач користувачів у веб-додатку. Цей процес включає в себе управління різноманітними завданнями, визначення їх пріоритетності та дедлайнів, а також організацію робочого часу.

Предметом дослідження є розробка та реалізація веб-додатку для керування завданнями з інтеграцією погодних даних. В рамках цього предмету дослідження досліджується процес розробки програмного забезпечення, а також його можливе використання для підвищення продуктивності та ефективності користувачів у керуванні своїми завданнями.

Методи дослідження. Для досягнення поставлених цілей та вирішення завдань дослідження використовувалися наступні методи:

1. Аналіз літературних джерел та аналогів: проведений огляд наукової літератури, публікацій, та аналогічних веб-додатків для вивчення сучасних підходів до управління завданнями та погодними даними.

2. Проектування і розробка програмного продукту: використання методів системного аналізу та проектування для створення архітектури програмного забезпечення.

3. Розробка клієнтської та серверної частини додатку: Використання реалізації клієнтської частини засобами React та Next.js, а також серверної частини з використанням Next.js API routes для забезпечення комунікації між клієнтом та сервером.

4. Валідація та тестування програмного забезпечення: Проведення тестування програмного забезпечення з використанням різних методів, таких як модульне тестування, функціональне тестування та тестування інтеграцій.

5. Оцінка та аналіз результатів: Проведення аналізу отриманих результатів, який включав оцінку продуктивності, функціональності та зручності використання додатку, а також збирання та аналіз даних про його використання користувачами.

Ці методи були використані для забезпечення якісної розробки та ефективного вирішення поставлених завдань дослідження.

Практичне значення одержаних результатів. Отримані результати мають наступне практичне значення:

Розробка ефективного todo-вебсайту: Розроблений веб-додаток надає користувачам зручний інструмент для управління своїми завданнями та отримання актуальної погодної інформації на одній платформі.

Підвищення продуктивності користувачів: Додаток дозволяє користувачам ефективно організовувати свої робочі та особисті завдання, що може призвести до підвищення їх продуктивності та ефективності.

Зручний доступ до погодної інформації: Інтеграція погодного API дозволяє користувачам швидко та легко отримувати актуальну погодну інформацію без необхідності переходити на інші веб-сайти чи додатки.

Використання сучасних технологій: Використання сучасних фреймворків та інструментів розробки, таких як React, Next.js, MongoDB, дозволяє створити додаток з високою продуктивністю та надійністю.

Можливість розширення та модифікації: Архітектурний підхід до розробки дозволяє легко розширювати та модифікувати функціональність додатку для відповіді на потреби користувачів.

Отримані результати відображають практичне застосування розробленого веб-додатку та його важливість для кінцевих користувачів.

Апробація результатів дослідження. Матеріали кваліфікаційної роботи були представлені на XI Міжнародній науковій конференції «IT екосистема: цифровізація бізнес-процесів в умовах війни», яка відбулася 23-24 листопада 2023 року в Університеті Короля Данила.

Структура. Розділи – 3. Обсяг основної частини – 48 сторінки. Список використаних джерел – 20.

РОЗДІЛ 1. ЗАГАЛЬНІ ТЕРЕОТИЧНІ ВІДОМОСТІ

1.1 Проблема управління задачами

Сутність досліджуваної проблеми в даному проекті полягає в розробці todo-вебсайту з інтеграцією погодових даних. Ця проблема виникає з необхідності управління задачами та одночасного отримання актуальної інформації про погоду для зручності користувача.

Досліджувана проблема включає в себе декілька аспектів:

1. Управління задачами: користувачам потрібен інструмент, який дозволить їм створювати, переглядати, редагувати та видаляти свої завдання. Це важливо для ефективного планування робочого часу та досягнення поставлених цілей.

2. Інтеграція погодових даних: знання погоди є важливим елементом для багатьох користувачів при плануванні своїх дій. Інформація про температуру, вологість, вітряність та інші погодні умови допомагає приймати обгрунтовані рішення щодо проведення часу, подорожей або занять спортом.

3. Інтеграція аутентифікації: для забезпечення безпеки та ідентифікації користувачів важливо мати систему аутентифікації, яка дозволяє їм створювати облікові записи, входити в них та зберігати персональні налаштування.

4. Оптимізація та зручність використання: важливо, щоб веб-сайт був легким у використанні та забезпечував приємний користувацький досвід. Це включає в себе швидкий доступ до інформації, чіткий інтерфейс користувача та можливість швидкої навігації між різними функціями.

Розробка веб-сайту, яка комбінує управління задачами з інформацією про погоду та системою аутентифікації, відповідає цим вимогам, надаючи користувачам зручний інструмент для планування їхнього часу та дій.

Цей підрозділ є важливим етапом в дослідженні, оскільки від нього залежить розуміння контексту проблеми та визначення його місця.

1.2 Об'єкт та предмет дослідження

У цьому підрозділі визначається об'єкт та предмет дослідження з метою забезпечення чіткого розуміння обсягу та меж роботи.

Об'єктом дослідження у даній кваліфікаційній роботі є процес створення та функціонування todo-вебсайту з інтеграцією погоди, реалізованого за допомогою Next.js. Це широке поняття, що включає в себе всі аспекти проекту, від проектування та розробки до реалізації та ефективного функціонування в реальному середовищі.

Предметом дослідження є конкретні компоненти, технології та методи програмування, які використовуються в процесі створення вебсайту. Це охоплює клієнтські та серверні компоненти, їх взаємодію, архітектурні рішення, підходи до безпеки та оптимізації, а також інтеграцію з зовнішніми сервісами, такими як погодний API (Open Weather API).

У цьому контексті, важливо розглянути деталі інтеграції погодного API з вебсайтом, включаючи процес отримання та відображення погодних даних, розробку відповідних компонентів для їх відображення, а також забезпечення коректної реакції вебсайту на зміни погодних умов.

Крім того, в рамках предмету дослідження слід розглянути вибір та використання технологій, таких як MongoDB для зберігання даних, next-auth для автентифікації користувачів, а також Tailwind CSS для стилізації та респонсивного дизайну. Цей підрозділ допомагає чітко визначити параметри дослідження, визначити обсяг роботи та встановити межі аналізу. Це дозволяє зосередитися на конкретних аспектах проекту, враховуючи його загальну мету та цілі.

1.3 Вибір методів дослідження

У даному розділі визначається методологія та методи, які використовувалися для проведення дослідження та реалізації поставлених завдань.

У контексті цієї кваліфікаційної роботи для дослідження та реалізації вебсайту з інтеграцією погоди були використані наступні методи:

1. Аналіз літературних джерел та вивчення документації: Для ознайомлення з сучасними підходами та технологіями у розробці веб-додатків, а також з особливостями роботи з NextJS, MongoDB, та іншими використаними технологіями було проведено аналіз доступних джерел і документації.

2. Емпіричні методи дослідження: Включали в себе експериментальне тестування розробленого вебсайту, а також апробацію його функціональності та користувацького досвіду.

3. Тестування програмного забезпечення: Використання тестових наборів даних для перевірки роботи програмного продукту, виявлення помилок та виправлення їх.

Вибір цих методів дослідження був обумовлений необхідністю отримання комплексного та об'єктивного уявлення про функціональність та ефективність розробленого вебсайту. Комбінація дослідницьких методів дозволила вивчити проблему з різних сторін, забезпечивши повноту та об'єктивність отриманих результатів.

1.4 Аналіз та порівняння сервісів-аналогів

На ринку існує кілька аналогічних сервісів, які комбінують управління задачами з інтеграцією погодових даних та системою аутентифікації. Деякі з них варто розглянути для порівняння з проектом:

1. Todoist: Todoist - це популярний сервіс управлінням задачами, який дозволяє користувачам створювати завдання, надавати їм терміни та пріоритети. Хоча Todoist не має вбудованої функціональності погоди, він пропонує інтеграцію з різними додатками та сервісами, включаючи погодні додатки. Однак, для використання цієї функції, користувачам доводиться встановлювати додаткові розширення або інтегрувати сторонні сервіси (рис. 1.1).

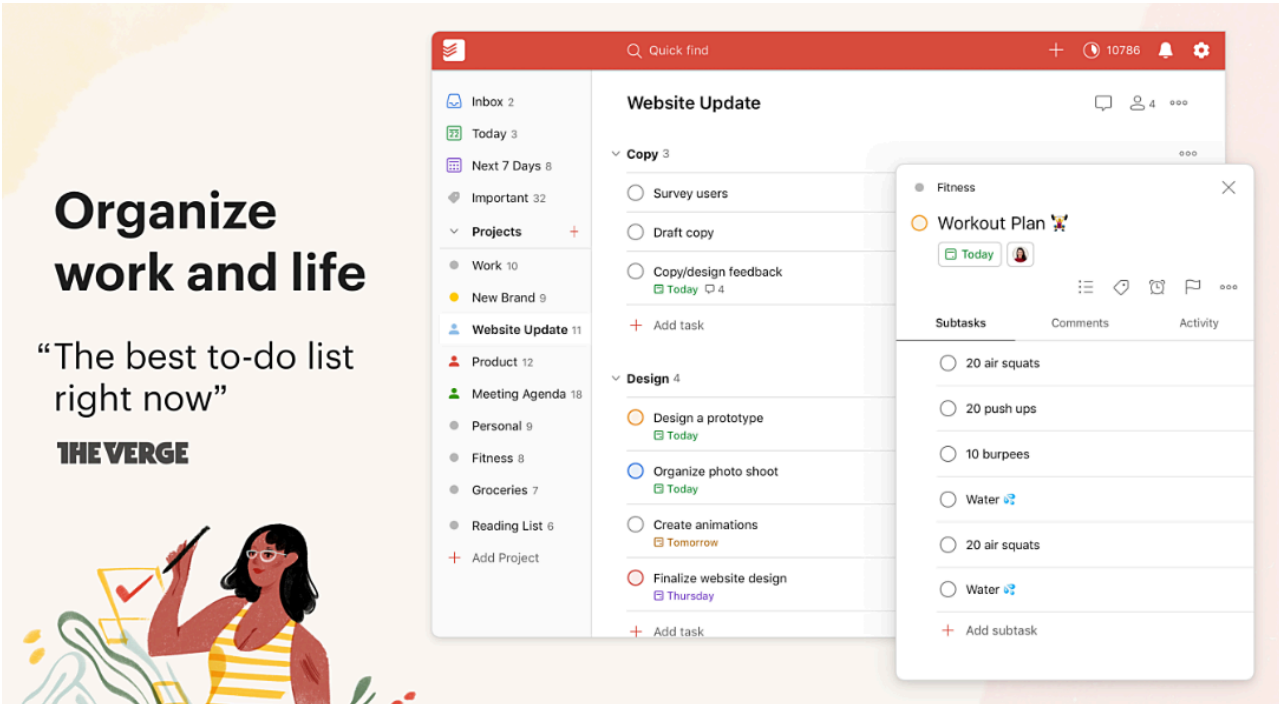


Рисунок 1.1 – Todoist

2. Microsoft To Do: Іншим аналогом є Microsoft To Do, який є частиною екосистеми продуктів Microsoft. Microsoft To Do дозволяє користувачам створювати, впорядковувати та відстежувати свої завдання. Хоча в сервісі немає вбудованого модуля погоди, він має різні інтеграції з іншими сервісами Microsoft, такими як Outlook, які можуть надавати додаткову інформацію про погоду, зображено рисунку 1.2.

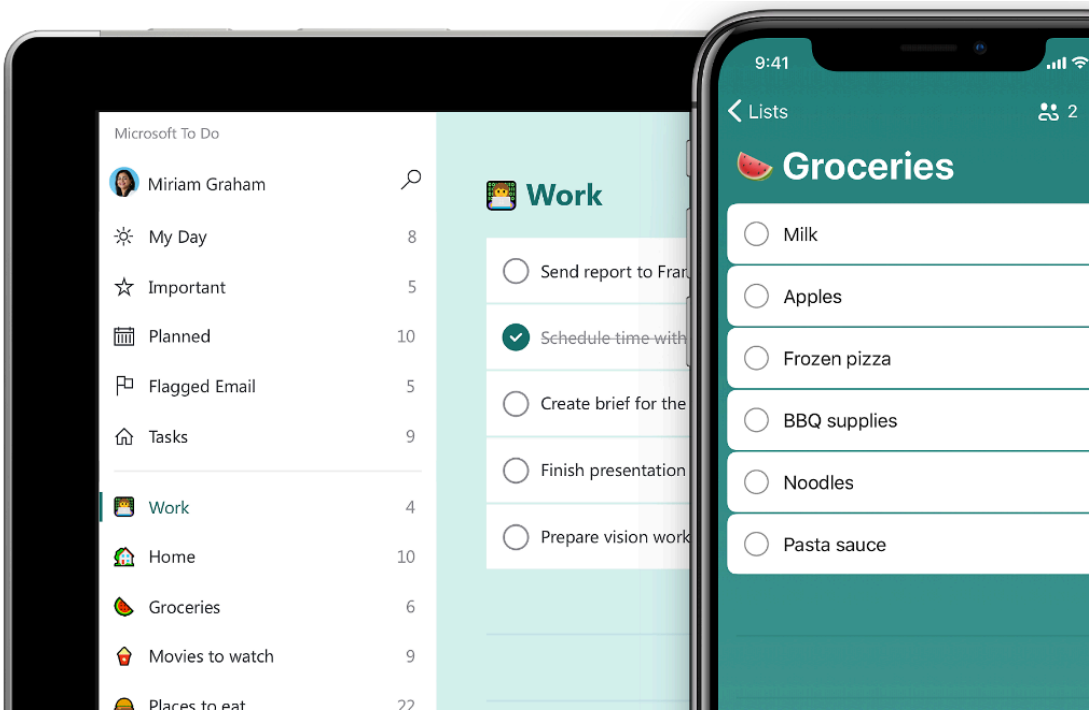


Рисунок 1.2 – Microsoft To Do

3. Google Tasks: Google Tasks - це ще один відомий сервіс управління завданнями, який інтегрується з іншими продуктами Google, такими як Gmail та Календар Google. Він дозволяє створювати прості списки завдань і встановлювати терміни, але не має вбудованої підтримки для погоди. Однак, користувачі можуть використовувати сторонні розширення або інтеграції для отримання погодової інформації (рис. 1.3).

Порівнюючи розроблюваний проект з аналогами, можна відзначити, що він вирізняється вбудованою підтримкою погоди та інтеграцією з системою аутентифікації. Такий функціонал не передбачений у сервісах-аналогах, що робить розроблюваний додаток більш вузькопрофільним та зручним у використанні для конкретно поставлених цілей. Це дозволяє користувачам не лише керувати своїми завданнями, але й отримувати актуальну інформацію про погоду, що може значно полегшити їхнє планування та прийняття рішень.

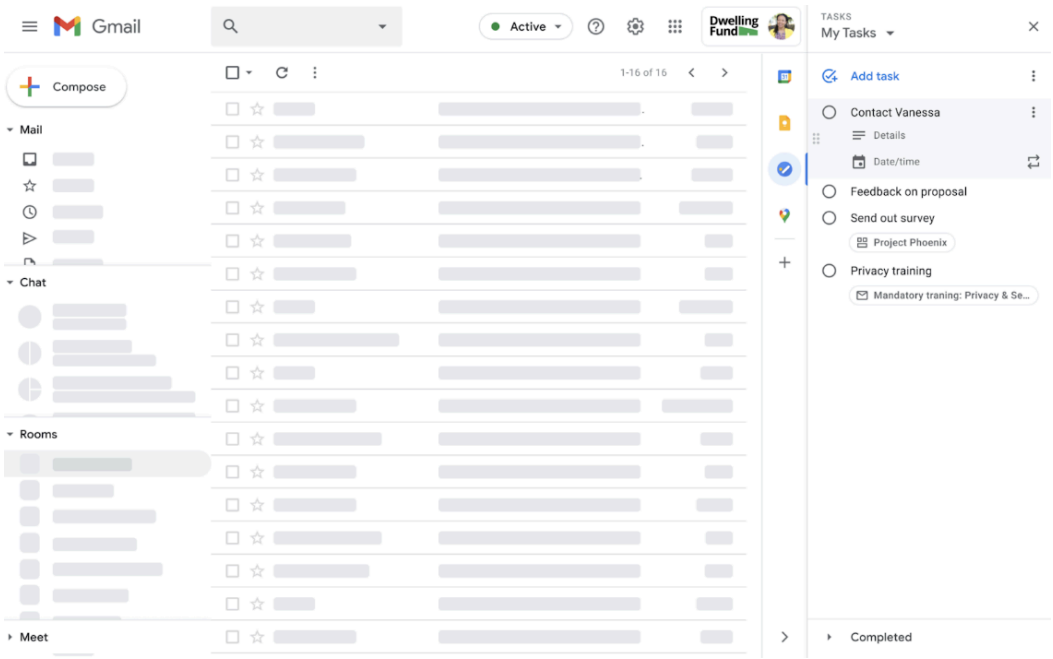


Рисунок. 1.3 – Google Tasks

Висновки до розділу 1

У першому розділі пояснювальної записки розглянуто теоретичні аспекти, які є основою для розробки веб-додатку управління задачами з інтеграцією погоди. Управління задачами є важливою складовою організації роботи як окремих людей, так і команд. Ефективне управління дозволяє підвищити продуктивність, оптимізувати робочий процес та забезпечити своєчасне виконання завдань. Однак, багато існуючих рішень можуть бути обмежені у функціоналі або не інтегрувати необхідні додаткові функції, такі як відстеження погодних умов. Об'єктом дослідження є системи управління задачами, а предметом - розробка інтегрованого веб-додатку, який не лише забезпечує функціонал планування та моніторингу задач, але й надає користувачам актуальну інформацію про погодні умови. Така інтеграція дозволяє користувачам краще планувати свої завдання, зважаючи на зовнішні фактори. Для досягнення поставлених цілей було використано методи аналізу та порівняння існуючих рішень, що дозволило визначити їх сильні та

слабкі сторони. Це, у свою чергу, допомогло сформувати вимоги до розробки власного додатку, який би вирішував виявлені проблеми та задовольняв потреби користувачів. Проведений аналіз сервісів-аналогів виявив, що більшість з них зосереджені на базовому управлінні задачами без врахування зовнішніх факторів, таких як погода. Натомість розроблюваний додаток пропонує унікальну інтеграцію з OpenWeather API, що надає користувачам додаткову цінність та можливість планувати задачі більш ефективно. Проведений теоретичний аналіз підтвердив актуальність та необхідність розробки нового веб-додатку, який поєднує управління задачами з інтеграцією погодних умов. Це дозволяє користувачам більш раціонально підходити до планування та виконання своїх завдань, враховуючи реальні умови навколишнього середовища.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ДОДАТКУ

2.1 Визначення завдань додатку

У цьому розділі надається загальна оцінка проблеми, яка вирішувалася під час розробки веб-додатку з інтеграцією погоди. Розглядаються ключові аспекти, що вплинули на вибір теми та визначення завдань дослідження.

В контексті цього проекту завдання полягатиме у створенні вебсайту, який би поєднував у собі функціональність спостереження за погодою з можливістю керування списком завдань (todo list), який також передбачатиме інтеграцію аутентифікації, що дозволить користувачам зберігати індивідуальні дані. Основними аспектами, які вплинули на вибір цієї проблеми, є:

1. Потреба у зручному способі керування задачами: у сучасному світі люди часто стикаються з великою кількістю завдань та обов'язків, і використання електронного todo-списку може допомогти їм краще організувати свій час та ресурси. Такий список вирішено реалізувати у вигляді веб-ресурсу, що забезпечть користувачам доступ до додатку за наявністю будь-якого браузера без необхідності встановлювати додаток на свій десктопний або мобільний пристрій.

2. Популярність погодних додатків: спостереження за погодою є важливою частиною повсякденного життя для багатьох людей. Велика кількість людей, які шукають можливість покращити свою продуктивність слідкують паралельно за погодними даними та планують свої задачі. Об’єднання та інтеграція цієї функціональності btodo-додаток може зробити його більш привабливим для таких користувачів.

3. Розвиток веб-технологій: За останні роки веб-технології значно покращилися, що відкрило широкі можливості для створення високопродуктивних

та функціональних веб-додатків. Хоч на ринку і існує декілька сервісів-аналогів, які передбачають схожий функціонал, проте вони не мають безпосередньої інтеграції з погодними даними та не використовують велику кількість нових технологій (наприклад, SSR), що дозволяють прискорити швидкодію додатку.

4. Потенційний комерційний інтерес: Здатність пропонувати цікавий та корисний продукт може мати комерційний потенціал, зокрема шляхом рекламних партнерств або платних підписок. Проте в рамках даної кваліфікаційної роботи такого інтересу немає.

Проект вирішено розробляти у вигляді веб-додатку, що дозволить користувачам відвідати застосунок з будь-якого пристрою. Це обумовлено популярністю такого підходу та тенденцією до інтеграції всіх ресурсів та сервісів у браузер, що спрощує роботу з додатками. Фундаментом для всього було обрано Next.js - фреймворк для реактивних веб-додатків, побудований на основі Node.js та розроблений для платформи React. Він надає простий та потужний інструментарій для розробки універсальних (іноді відомих як ізоморфні) веб-додатків, що рендеруються як на сервері, так і на клієнтському боці. Next.js дозволяє створювати універсальні додатки, які можуть виконуватися як на сервері, так і на клієнтському боці. Це дозволяє використовувати підходи SSR (Server-Side Rendering) та SSG (Static Site Generation) для побудови додатків з високою продуктивністю та SEO-вигодами. Next.js має вбудований маршрутизатор, який дозволяє легко визначати маршрути вашого додатку та пов'язані з ними обробники. Next.js підтримує розширення за допомогою сторонніх модулів та плагінів, що дозволяє додавати новий функціонал та використовувати інші бібліотеки для поліпшення розробки. Next.js дозволяє легко взаємодіяти з серверними та клієнтськими даними, використовуючи різні підходи, такі як SSR, SSG, або загальний стан додатку (наприклад, за допомогою контексту React). Next.js автоматично оптимізує додатки шляхом пакетування та мінімізації коду, а також

генерації кешу для статичних сторінок, що покращує продуктивність та швидкість завантаження. Next.js має вбудовану підтримку CSS модулів, а також можливість використання різних препроцесорів CSS, таких як Sass чи Less.

Next.js – фреймворк, побудований на базі React - бібліотеки для розробки інтерфейсів користувача, створена компанією Facebook. Вона дозволяє розробникам будувати динамічні, інтерактивні та швидкі веб-додатки з використанням компонентного підходу. React базується на ідеї компонентів - малих, незалежних будівельних блоків, які можна повторно використовувати та комбінувати для створення складних інтерфейсів. Це спрощує розробку та підтримку коду, а також забезпечує чітку структуру додатку. React використовує віртуальний DOM для ефективного оновлення інтерфейсу користувача. Він відслідковує зміни у стані компонентів та оновлює лише ті частини DOM, які потребують змін, забезпечуючи високу продуктивність додатків. React дозволяє легко створювати SPA, де весь контент завантажується за один раз, а переходи між сторінками відбуваються без повного перезавантаження сторінки. React використовує декларативний підхід до розробки, що означає, що ви описуєте, як повинен виглядати ваш інтерфейс у певний момент часу, а не програмуєте кроки для досягнення бажаного результату. React дозволяє розширювати можливості за допомогою додаткових бібліотек і фреймворків, таких як Redux для управління станом, React Router для маршрутизації, або Material-UI для створення стильних інтерфейсів.

Next.js побудований на основі React та додає додатковий шар функціональності для розробки універсальних веб-додатків з використанням SSR, SSG, розширеної системи маршрутизації та іншого. Використовуючи обидві ці технології разом, розробники можуть створювати потужні та ефективні додатки для вебу.

Загалом, Next.js є потужним інструментом для розробки веб-додатків, який поєднує в собі можливості Node.js та React, дозволяючи створювати швидкі, масштабовані та універсальні додатки з високою продуктивністю.

Розуміння цих аспектів дозволило зорієнтуватися в розробці додатку та визначити його основні функціональні вимоги.

2.2 Створення макету в Figma

У цьому підрозділі детально описується процес створення макету в графічному редакторі Figma. Макет відображає вигляд та функціональні можливості веб-додатку перед його розробкою. Важливо врахувати візуальний дизайн і ергономіку інтерфейсу для зручного користування (рис. 2.1).

Основні кроки створення макету в Figma включали:

1. Визначення структури сторінок: визначення основних елементів і розташування їх на сторінці. Це включає заголовок, навігаційне меню, область відображення погоди, todo-список та інші елементи. Під час проектування структури сторінок був визначений найбільш оптимальний варіант, за якого додаток має три основні функціональні сторінки. На кожну зі сторінок користувач легко може потрапити, скориставшись функціоналом навігаційного меню, яке також зручно розміщене у шапці веб-сайту.

2. Вибір кольорової схеми і шрифтів: були обрані кольори, які відповідають строгому стилю і тим самим надають відповідний настрій додатку. На рисунку 2.1 можна побачити обрану для додатку кольорову гаму, в якій основний колір – світло-сірий, а додатковй – блакитний. Він використовується для підкреслення певних деталей, такі, як, наприклад, кнопки або рамки блоків. Також встановлюються шрифти для текстових елементів. Було обрано строгий шрифт Latin, який також підкреслює основний настрій додатку та не відволікає

користувача від основного функціоналу додатку, даючи йому при цьому змогу легко читати текст і користуватись усіма функціями додатку без додаткових складнощів.

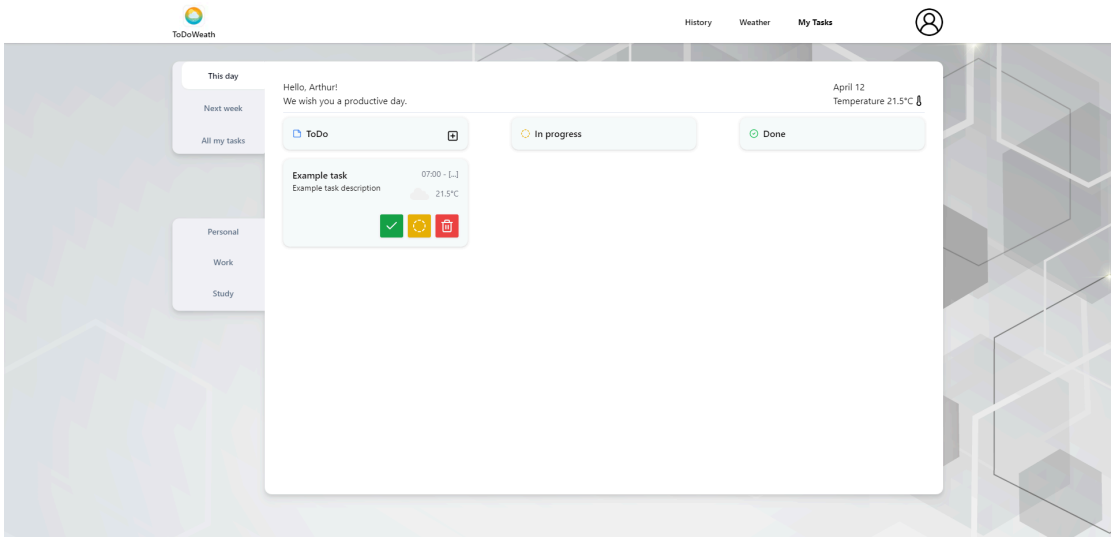


Рисунок. 2.1 – Створення макету в Figma

3. Створення макетів елементів інтерфейсу: розробка вигляду окремих елементів, таких як кнопки, поля введення, списки тощо. Враховується їх розміщення та взаємодія з користувачем. Усі кнопки мають яскраву фонову заливку, щоб користувачеві було легше розпізнавати їх на фоні інших об'єктів на сторінці. Якщо кнопка щось видаляє – вона забарвлена в червоний для кращого інтуїтивного розуміння її призначення. Кнопка виконання завдань забарвлена зеленим, що також допомагає користувачеві легко зорієнтуватись і зрозуміти що ця кнопка робить. Якщо інтуїтивний колір складно визначити – кнопка забарвлена додатковим кольором кольорової схеми додатку, а саме – блакитним кольором. Усі інтерактивні елементи веб-сайту мають блакитний аутлайн для кращого розуміння користувачем того, з яким об'єктом він взаємодіє в даний момент.

4. Прототипування: створення інтерактивних прототипів, які демонструють роботу різних елементів інтерфейсу та навігації в додатку. Це допомагає краще зрозуміти, як користувачі будуть взаємодіяти з додатком.

5. Перегляд і тестування макету: перегляд і тестування макету з потенційними користувачами для отримання зворотного зв'язку та виявлення можливих проблем (рис. 2.2). Таке тестування допомогло значно покращити юзабіліті інтерфейсу.

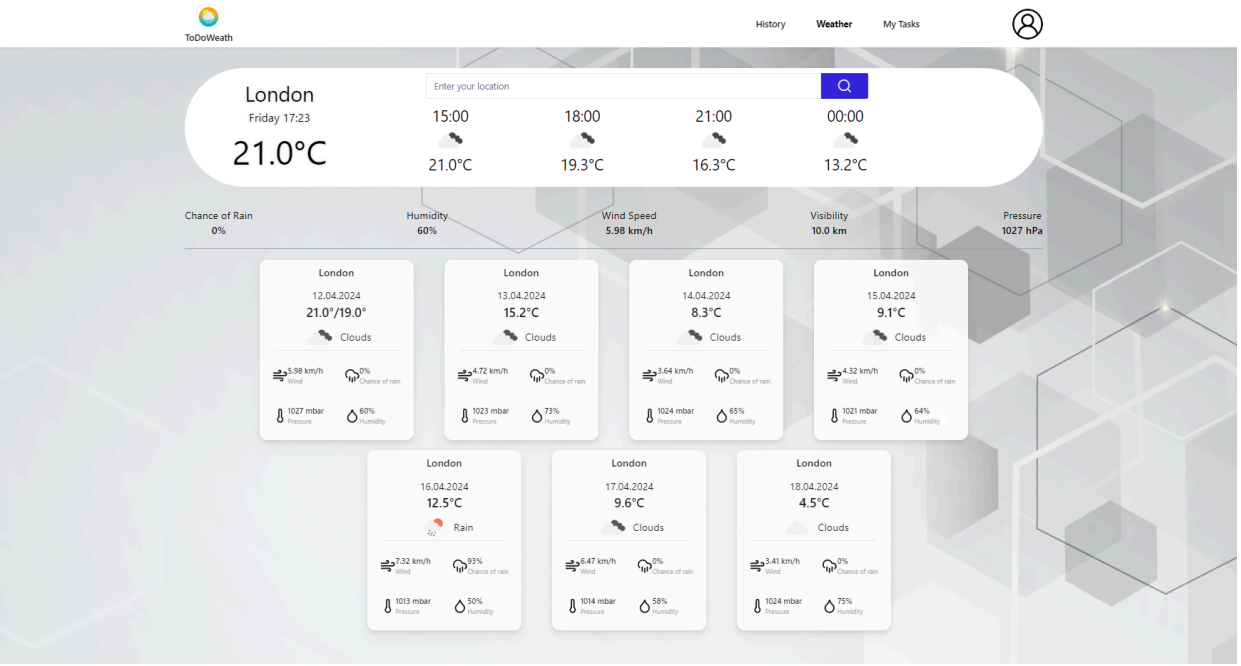


Рисунок. 2.2 – Головна сторінка сайту

Створення макету в Figma є важливим етапом перед розробкою реального додатку, оскільки це дозволяє визначити вигляд та функціонал продукту перед початком програмування. Даному етапу була приділена значна кількість часу, що в подальшому допомогло прискорити розробку та зробити додаток більш клієнтоорієнтовним.

2.3 Обґрунтування вибору технологій

У цьому розділі роботи розглядається обґрунтування вибору конкретних технологій для реалізації проекту. Це включає вибір мов програмування, фреймворків, бібліотек, баз даних та інших інструментів, які використовуються для створення програмного продукту.

Основні аспекти обґрунтування вибору технологій:

1. Відповідність вимогам проекту: Розглядається, наскільки обрані технології відповідають вимогам і функціональним можливостям проекту. Наприклад, якщо потрібно створити веб-додаток з динамічною взаємодією, вибір фронтенд-фреймворку, такого як React або Angular, може бути обґрунтованим.

2. Досвід розробника: Враховується досвід команди розробників з обраною технологією. Якщо команда має досвід роботи з певною мовою програмування або інструментом, то це може стати вагомим аргументом для їх вибору.

3. Швидкість розробки та підтримки: Оцінюється швидкість розробки за допомогою конкретних технологій, а також можливість підтримки та розвитку програмного продукту в майбутньому. Наприклад, використання популярного фреймворку може сприяти швидкій розробці, а використання активно розвиваючоїся технології може забезпечити підтримку продукту у майбутньому.

4. Екосистема та спільнота розробників: Оцінюється наявність розширень, бібліотек, плагінів та іншої екосистеми, яка може полегшити розробку та підтримку проекту. Також важливо враховувати активність спільноти розробників та наявність документації та ресурсів для вивчення обраної технології.

Обрані технології для розробки додатку:

1. Next.js: обрано Next.js як фреймворк для реалізації веб-сайту з урахуванням його можливостей у роботі як з серверною, так і з клієнтською частинами додатку. Next.js забезпечує швидку розробку завдяки своїй серверній

рендерінгу, статичній генерації та можливості простого розгортання на платформах, таких як Vercel. Екосистема Next.js в свою чергу також надала деякі бібліотеки, що дозволили прискорити розробку інтерфейсу та функціоналу, як, наприклад ShadcnUI, який надає велику кількість готових функціональних компонентів, які гнучко можуть стилізуватись засобами TailwindCSS та інтегруватись у логіку додатку, або, наприклад, react-hook-form у комбінації із zod resolver, що дозволили прискорити та оптимізувати роботу з формами та даними, а також передбачити потрапляння в базу даних помилкових даних.

2. Vercel: Vercel являється розробником Next.js, та його було обрано для розгортання веб-додатку на сервері. Зручні інструменти для моніторингу даних та серверних логів дозволяють легко відслідковувати помилки або проблеми швидкодії, що значно спрощує тестування додатку. Легка інтеграція з GitHub також робить розгортання ще простішим та зручним.

3. MongoDB: вибір MongoDB як бази даних обумовлений її гнучкістю та можливістю працювати з документами у форматі JSON, що відповідає потребам проекту. Крім того, MongoDB є популярним рішенням у сфері розробки веб-додатків, що забезпечує наявність широкої підтримки та ресурсів для вивчення. MongoDB є кращим в певних сценаріях порівняно з табличними базами даних завдяки декільком особливостям. По-перше, MongoDB використовує гнучку схему документів, дозволяючи кожному документу мати свою власну структуру без необхідності перекомпіляції таблиць. Це дозволяє легко змінювати структуру даних. Крім того, MongoDB надає підтримку для складених об'єктів, що спрощує моделювання складних даних. Для додатків з великим обсягом даних MongoDB також є привабливим вибором, оскільки він добре масштабується горизонтально. Це означає, що можна легко додавати нові сервери для розподілення навантаження. Нарешті, вбудована підтримка для реплікації у MongoDB дозволяє

автоматично реплікувати дані, що забезпечує більш високу доступність та надійність системи.

4. React: використання React для клієнтської частини додатку обґрунтовано його популярністю, широким спектром бібліотек та плагінів, а також зручним синтаксисом, що спрощує розробку і підтримку коду. React має численні переваги, які визначають його популярність серед розробників. По-перше, React пропонує компонентний підхід до розробки, що спрощує розділення складних інтерфейсів на невеликі і повторно використовувані частини, що полегшує розробку та підтримку коду. Далі, React використовує віртуальний DOM, що дозволяє ефективно оновлювати та маніпулювати інтерфейсом, що призводить до високої продуктивності додатків. Крім того, React має велику та активну спільноту, яка надає безліч ресурсів, документації та сторонніх бібліотек для розширення його можливостей. Однак у React також є деякі недоліки. Наприклад, React може вимагати додаткових бібліотек або рішень для деяких завдань, таких як управління станом додатку або маршрутизація. Також, хоча віртуальний DOM допомагає досягти високої продуктивності, він також може призводити до перевищення вимог щодо використання пам'яті, особливо в додатках з великим обсягом даних або складною логікою відображення.

5. Tailwind CSS: вибір Tailwind CSS для стилізації веб-сайту обумовлений його можливістю швидкої розробки і адаптації до різних вимог дизайну. Tailwind CSS надає готові компоненти та класи, що спрощує роботу зі стилями та забезпечує їх консистентність. Tailwind CSS пропонує швидкий та простий спосіб створення стилів, оскільки можна додавати класи безпосередньо до HTML-коду, не потрібно писати або управляти CSS-файлами. Це також полегшує візуальне тестування та налагодження стилів прямо в браузері. Крім того, Tailwind CSS надає широкий набір стандартних класів, які покривають більшість основних потреб стилізації, що дозволяє швидко створювати стильований вигляд веб-сайту

або додатку. Однак, використання Tailwind CSS також має деякі недоліки. Наприклад, великий обсяг HTML-коду через використання класів безпосередньо в розмітці, що може зробити код менш читабельним, особливо на великих проектах. Крім того, можуть виникнути проблеми з переносом або змінами стилів, оскільки стилі прив'язані безпосередньо до HTML-елементів через класи, що може призвести до зміни в коді багатьох сторінок для однієї зміни стилю.

6. NextAuth.js: обрано NextAuth.js для реалізації системи авторизації та аутентифікації через соціальні мережі, такі як Google та Facebook. NextAuth.js забезпечує зручний інтерфейс для роботи з авторизацією, а також підтримує різні стратегії аутентифікації. Підтримка різних постачальників дозволяє вибирати найкращий варіант для додатку. Також важливо відзначити, що NextAuth.js ідеально підходить для інтеграції з фреймворком Next.js, що робить його зручним вибором для даного проекту.

7. OpenWeather API: вибір OpenWeather API для отримання погодних даних обумовлений його доступністю та можливістю отримати широкий спектр погодних параметрів, що відповідає потребам для інтеграції погоди у веб-сайт.

Аналіз існуючих аналогів обраних технологій:

1. React vs. Angular vs. Vue.js: React - це бібліотека JavaScript для побудови користувацьких інтерфейсів, вона відрізняється простотою вивчення та використання, широкою спільнотою та активною підтримкою. Angular - це фреймворк, який надає більше заздалегідь визначених рішень, тому він може викликати більше викликів для великих команд і проектів. Vue.js - це фреймворк, який поєднує в собі переваги React і Angular, він є більш простим у використанні, ніж Angular, і має більше заздалегідь визначених рішень, ніж React.

2. Tailwind CSS vs. Bootstrap vs. Material-UI: Tailwind CSS - це CSS-фреймворк, який надає низку стилів і компонентів, які можна використовувати для швидкої і зручної розробки інтерфейсу. Bootstrap і Material-UI

також надають компоненти та стилі, проте вони мають власний унікальний дизайн та функціонал. Tailwind CSS відрізняється від інших тим, що він дозволяє створювати стилі прямо в HTML, що полегшує роботу зі стилями.

3. NextAuth.js vs. Passport.js: NextAuth.js - це бібліотека аутентифікації для Next.js, яка спрощує реалізацію автентифікації в вашому додатку. Вона підтримує різні стратегії аутентифікації, такі як OAuth, JWT, email/password, і має готові рішення для роботи з різними провайдерами, такими як Google, Facebook, Twitter і т.д. Passport.js - це також бібліотека аутентифікації для Node.js, проте вона не включає в себе готові рішення для Next.js і потребує додаткового налаштування для інтеграції з цим фреймворком.

4. OpenWeather API vs. Weatherstack vs. Dark Sky API: OpenWeather API - це популярне API для отримання погодних даних, воно надає широкий спектр функціоналу, включаючи поточну погоду, прогноз, погодні картинки та інше. Weatherstack - це інше популярне API для отримання погодних даних, воно також надає широкий спектр функціоналу, проте може відрізнятися в якості даних та доступності за деякими параметрами. Dark Sky API - це API, яке спеціалізується на точних погодних прогнозах, особливо в області прогнозів до години. Він також надає різноманітні погодні дані та функціонал.

5. MongoDB vs. MySQL vs. PostgreSQL: MongoDB - це NoSQL база даних, яка працює з документами у форматі JSON. Вона відрізняється від реляційних баз даних тим, що не вимагає строго визначеної схеми даних і дозволяє легко масштабуватися. MySQL і PostgreSQL - це реляційні бази даних, які використовують SQL для роботи з даними. Вони відрізняються в деяких деталях реалізації і функціоналу, таких як підтримка виразів, транзакцій та індексів. Вибір між ними залежить від потреб та вимог до бази даних.

Обрані технології були узгоджені з вимогами проекту та враховують досвід розробників, швидкість розробки та підтримки, а також наявність екосистеми та спільноти розробників для кращої реалізації програмного продукту.

2.4 Вибір архітектурного шаблону

У цьому підрозділі описується процес проектування програмного продукту з урахуванням вибраних технологій та вимог проекту. Основні аспекти проектування включають:

Архітектура системи:

1. Для розробки веб-сайту була обрана архітектура з використанням Next.js, що забезпечує розподіл функціональності між клієнтською та серверною частинами додатку.
2. Компоненти сайту (наприклад, Header, History, Weather, Tasks, Auth) розподілені для забезпечення чіткої структури та повторного використання коду.
3. MVC, MVVM і MVP - це архітектурні шаблони, які використовуються в програмуванні для розділення коду програми на логічні компоненти і полегшення розробки та управління кодом.

MVC (Model-View-Controller):

1. Модель (Model): представляє дані та бізнес-логіку програми. Модель не знає про представлення або контролер, вона просто забезпечує доступ до даних.
2. Представлення (View): відображає дані моделі користувачеві і приймає від нього введення. Вона не має прямого доступу до даних, а отримує їх через модель.
3. Контролер (Controller): приймає введення від користувача, обробляє його і взаємодіє з моделлю та представленням. Контролер також може виконувати додаткову логіку обробки даних перед їх передачею в модель або представлення.

MVVM (Model-View-ViewModel):

1. Модель (Model): той самий, що і в MVC.
2. Представлення (View): також той самий, що і в MVC.

ViewModel: це проміжний шар між моделлю і представленням. ViewModel використовується для зберігання та обробки даних, які відображаються в представленні. Він відокремлює логіку відображення даних від логіки моделі, що дозволяє легше тестувати і підтримувати код.

MVP (Model-View-Presenter):

1. Модель (Model): такий самий, як і в інших архітектурних шаблонах.
2. Представлення (View): відображає дані та взаємодіє з користувачем. У MVP представлення може взаємодіяти лише з презентером.
3. Презентер (Presenter): містить бізнес-логіку, взаємодіє з моделлю та відображенням. Презентер отримує дані від моделі, обробляє їх та оновлює представлення. У відміну від контролера у MVC, презентер виконує всю логіку, пов'язану з відображенням даних.

Архітектурним шаблоном для розроблюваного додатку було обрано шаблон MVP (Model-View-Presenter) з кількох причин.

Розділення відповідальностей: MVP дозволяє чітко розділити логіку додатку на три компоненти: модель, представлення і презентер. Це полегшить розробку, тестування та підтримку коду, оскільки кожен компонент відповідає лише за свої конкретні функції.

Тестування: оскільки логіка відображення даних розміщується в презентері, це дозволяє легше тестувати представлення без необхідності взаємодії з реальною моделлю даних або фреймворками GUI. Це спрощує автоматизоване тестування, хоча воно і не було використане в процесі розробки.

Фокус на бізнес-логіці: MVP дозволяє виокремити бізнес-логіку додатку від його відображення, що полегшує розуміння та управління кодом. Презентер

відповідає за обробку даних і взаємодію з моделлю, тоді як представлення відповідає лише за відображення даних.

Масштабованість: завдяки розділенню відповідальностей, MVP дозволяє легше масштабувати додаток. Кожен компонент може бути розроблений та підтриманий незалежно, що значно полегшує роботу над проектом.

Гнучкість і адаптабельність: MVP дозволяє змінювати відображення даних без впливу на бізнес-логіку додатку. Це дозволяє легше внесення змін в інтерфейс користувача або вибір різних технологій для відображення без значних змін у решті додатку.

Дизайн інтерфейсу користувача (UI/UX): дизайн інтерфейсу розроблявся з урахуванням зручності та привабливості для користувачів, відповідно до вимог проекту (рис. 2.3).

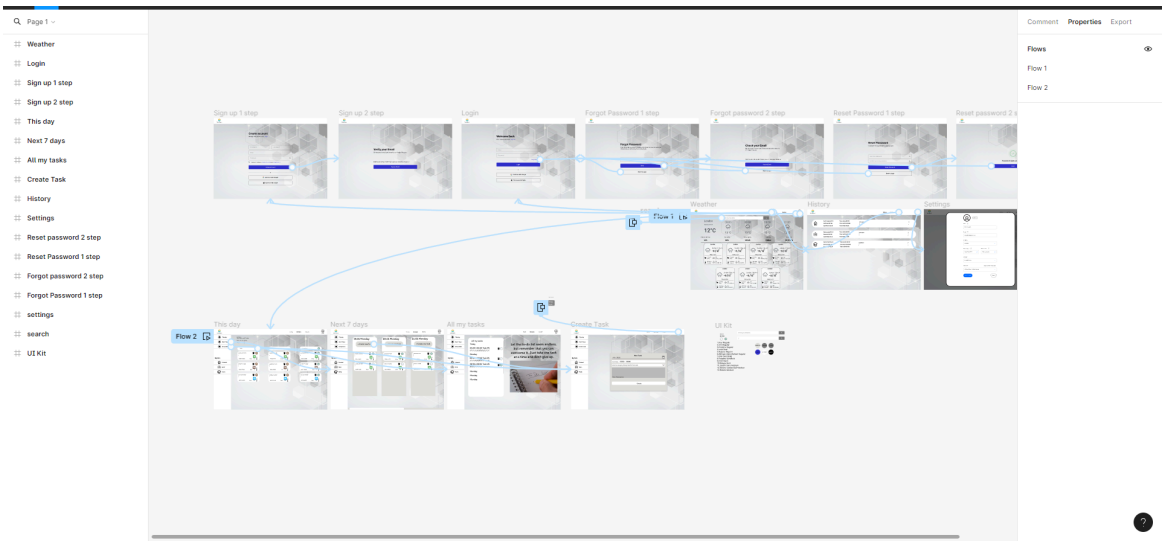


Рисунок. 2.3 – Макет в Figma

Користувальницький інтерфейс включає в себе компоненти, такі як кнопки, форми, та анімаційні ефекти, які надають зручності та естетичного задоволення.

Безпека: для захисту від несанкціонованого доступу та збереження конфіденційної інформації, використовується система аутентифікації та авторизації, яка імплементована через NextAuth.js.

У проекті використовується передбачений файл middleware, який допомагає обмежувати неавторизованим користувачам доступ до певних сторінок веб-сайту. Дані користувачів зберігаються у безпечній базі даних MongoDB з використанням методів шифрування.

Швидкість та оптимізація: для підвищення швидкості завантаження сторінок та оптимізації продуктивності, використовуються методи кешування даних та оптимізація запитів до бази даних.

Масштабованість та розширюваність: система розроблялася з урахуванням можливості масштабування для роботи з великим обсягом даних та користувачів, забезпечуючи гнучку архітектуру для майбутнього розширення(рис. 2.4).

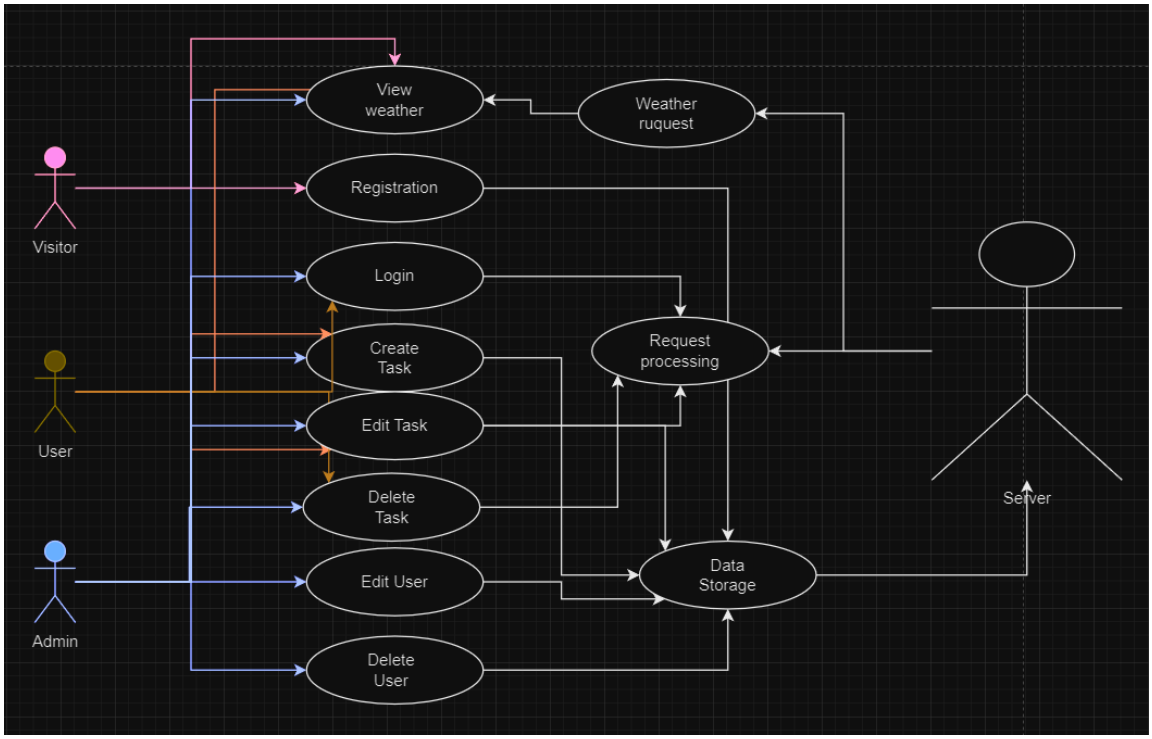


Рисунок. 2.4 – Діаграма прототипу додатку

Тестування та QA: для забезпечення високої якості та надійності програмного продукту, використовуються стратегії тестування, такі як функціональне, інтеграційне та модульне тестування. Застосовуються методи QA для перевірки якості та відповідності програмного продукту вимогам.

Проектування програмного продукту є важливим етапом розробки, оскільки воно визначає загальну архітектуру, дизайн та функціональність продукту. Від якості проектування залежить подальший успіх розробки та ефективне виконання поставлених завдань

2.5 Серверний рендеринг

Виходячи із обраного архітектурного шаблону та фреймворку, на якому розробляється додаток, він будується із невеликих компонентів. Певна кількість цих компонентів використовується з технологію серверного рендерингу (процес генерації HTML на сервері для кожного запиту користувача перед тим, як відправити відповідь клієнту). Це значно впливає на певні показники швидкодії додатку та взаємодії з даними (рис. 2.5).

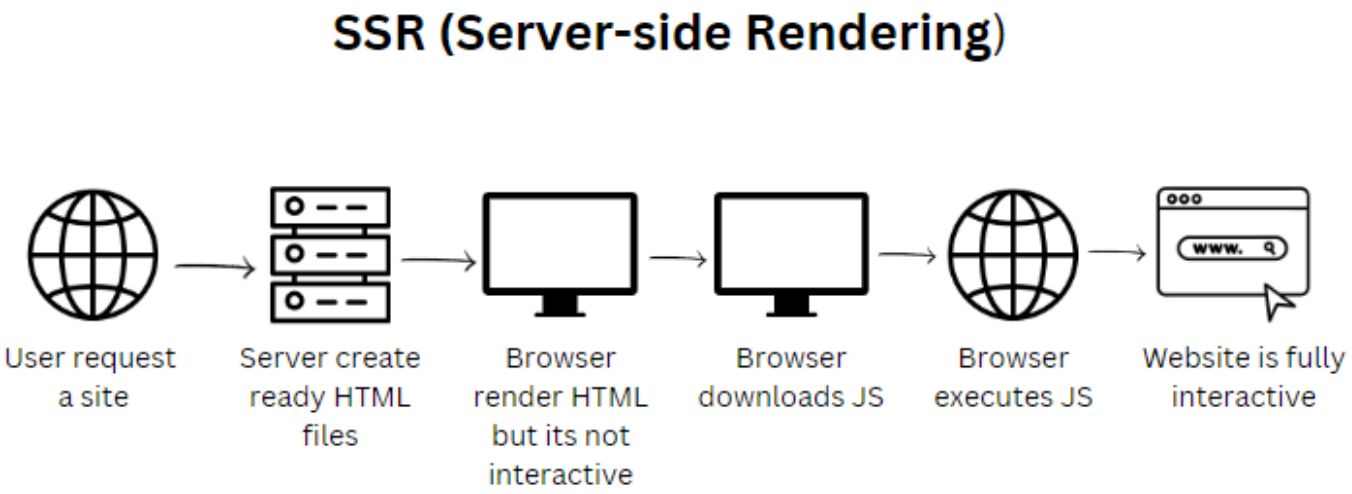


Рисунок. 2.5 – Процеси серверного рендерингу

Переваги:

Кращий SEO: статичний HTML, згенерований на сервері, краще індексується пошуковими системами, оскільки вони можуть легше аналізувати його зміст.

Швидка перша подача контенту (FMP): завдяки SSR, користувачі бачать контент швидше, оскільки він генерується на сервері і відправляється разом з першим відгуком на запит.

Покращення SEO та SMO: завдяки SSR, у вас є повний контроль над тим, як виглядає ваш веб-сайт на стороні сервера. Це може поліпшити якість шарингу на соціальних мережах.

Краща підтримка для браузерів та платформ: ви можете забезпечити, що ваш веб-сайт працює на всіх браузерах та платформах, оскільки весь контент генерується на сервері.

Недоліки:

1. Більша навантаженість на сервер: серверу потрібно буде генерувати HTML для кожного запиту, що може призвести до більшого навантаження на сервер, особливо при великому обсязі трафіку.
2. Помітне збільшення часу відгуку сервера: час відгуку сервера може збільшитися, особливо якщо ваш веб-сайт має складну логіку генерації сторінок або обробки великої кількості даних.
3. Складність кешування: кешування може бути складніше через динамічний характер змісту, що генерується на сервері.
4. Специфічні проблеми з безпекою: SSR може відкривати додаткові вектори атак, такі як XSS (Cross-Site Scripting), якщо не використовувати належні заходи безпеки.

2.6 OpenWeather API

OpenWeather API - це веб-API, яке було використане у розроблюваному додатку для створення функціоналу, пов’язаного з погодними даними.

Рисунок. 2.6 – Логотип OpenWeather API

Він надає доступ до різноманітної метеорологічної інформації для будь-якого місця на Землі. Ключові аспекти API:

1. Погодні дані: OpenWeather API дозволяє отримувати різноманітні погодні дані, такі як поточна температура, вологість, швидкість вітру, тиск та інші параметри. Усі ці параметри застосовано у додатку на вкладці “Weather”, де усі користувачі, включаючи не авторизованих, можуть проглянути інформацію про погоду та її показники у різних містах планети.

2. Прогноз погоди: можна отримати прогноз погоди на певний період часу для конкретного місця. Це може включати щоденний або годинний прогноз на кілька днів вперед. Запит на отримання такого прогнозу виконує додаток для відображення інформації в задачах.

3. Геолокація: API надає можливість отримання погодних даних за географічними координатами (широта та довгота) або за назвою місця.

4. Мапи погоди: OpenWeather також надає доступ до карт погоди, які відображають погоду на карті за допомогою різних шарів і маркерів.

5. Погодні шаблони: є також можливість використовувати шаблони для відображення погоди на веб-сайті або додатку, використовуючи стилізовані компоненти OpenWeather, які він пропонує.

6. Безкоштовний та платний доступ: OpenWeather API пропонує як безкоштовний, так і платний плани, які надають різні обсяги та типи доступної інформації. Безкоштовний план може мати обмеження за обсягом запитів або доступними функціями. У додатку використовується безкоштовний план з обмеженням у 1000 запитів на добу.

OpenWeather API надає різні методи запитів для отримання погодної інформації. Основні параметри запиту та формат відповіді можуть залежати від конкретного методу запиту та використовуваного плану доступу. Проте основні параметри та формати відповідей можуть бути наступними:

Параметри запиту:

- q (або city): назва міста, для якого потрібно отримати погоду;
- lat та lon: широта та довгота місця, для якого потрібно отримати погоду;
- zip (або zip code): поштовий індекс місця, для якого потрібно отримати погоду;
- id: ідентифікатор міста у базі даних OpenWeather;
- units: одиниці вимірювання для погодних параметрів (наприклад, "metric" для метричної системи або "imperial" для системи імперських одиниць).

Даний API має методи для отримання поточної погоди, погодних прогнозів на кілька днів вперед, історичних даних погоди та іншої інформації, такої як погодні мапи.

Формат відповіді може бути у формі JSON або XML, зазвичай JSON використовується як основний формат.

Повернуті дані містять інформацію про погоду (температура, вологість, швидкість вітру тощо), яка представлена в структурованому вигляді (рис. 2.7).

Деякі запити можуть також повертати іншу інформацію, таку як геолокація, часовий пояс, графіки погоди та інше.

Загальна структура запиту до OpenWeather API виглядає наступним чином:

https://api.openweathermap.org/data/2.5/{METHOD}?{PARAMS}&appid={YOUR_API_KEY}

де {METHOD} - метод запиту, {PARAMS} - параметри запиту, а {YOUR_API_KEY} - ваш ключ доступу до API.

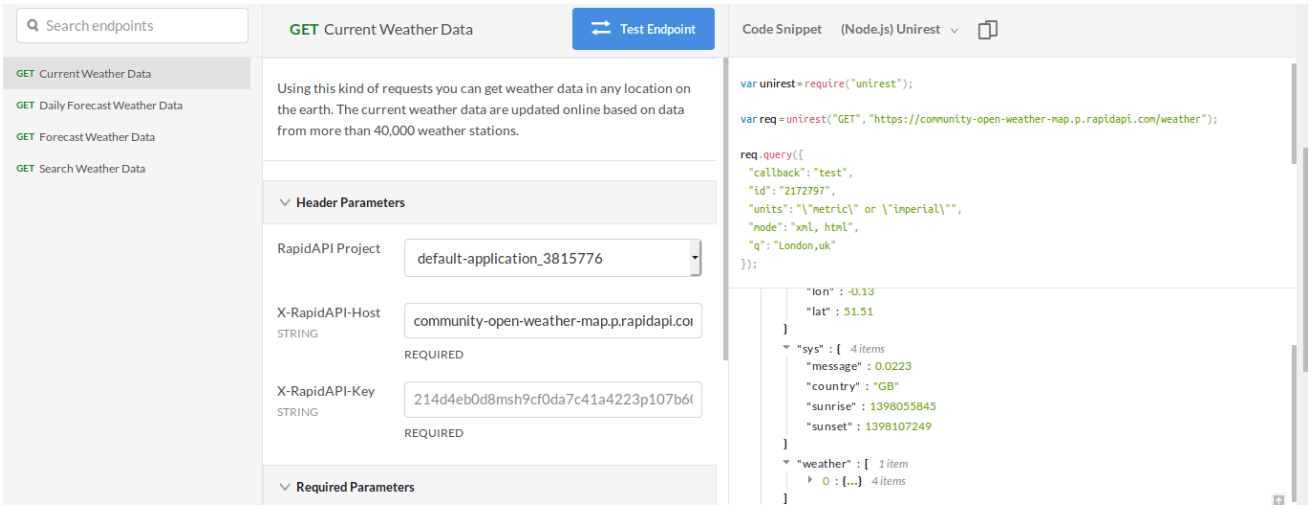


Рисунок. 2.7 – Приклад запиту до OpenWeather API та його відповіді

Розуміння доступних параметрів та методів дозволяє точно налаштувати запити до API та ефективно використовувати отриману інформацію.Переваги розуміння параметрів та методів API: точність запитів, ефективність використання, глибоке розуміння API.

— Точність запитів: правильне використання параметрів дозволяє отримувати саме ті дані, які вам потрібні, уникаючи зайвої інформації.
— Ефективність використання: знання методів API допомагає швидко та легко виконувати необхідні дії, економлячи час та ресурси.
— Глибоке розуміння API: чим краще ви розумієте параметри та методи, тим краще ви зможете використовувати можливості API та інтегрувати його у свою систему.

Висновки до розділу 2

У другому розділі пояснювальної записки детально розглянуто процес проектування додатку управління задачами з інтеграцією погоди. Основні завдання додатку включають забезпечення можливості користувачам створювати, редагувати та видаляти задачі, переглядати історію виконаних задач, а також отримувати актуальну інформацію про погодні умови. Ці функції спрямовані на поліпшення організації робочого процесу та планування, враховуючи зовнішні фактори, які можуть впливати на виконання задач.

Для створення візуального дизайну додатку було використано інструмент Figma. Це дозволило створити інтерактивні макети та прототипи, які допомогли визначити структуру інтерфейсу, розміщення елементів та загальний вигляд додатку. Такий підхід забезпечив можливість врахування зручності використання додатку ще на етапі проектування.

Вибір технологій для розробки додатку ґрунтувався на вимогах до функціональності, продуктивності та зручності підтримки. Було обрано NextJS як основний фреймворк для створення як клієнтської, так і серверної частин додатку. Використання NextJS дозволяє ефективно реалізувати серверний рендеринг, що покращує продуктивність та SEO. React був обраний для побудови компонентів

інтерфейсу, завдяки його популярності та широкій підтримці спільноти. MongoDB була обрана як база даних для зберігання інформації про користувачів та їх задачі через її гнучкість та можливість масштабування. Архітектурний шаблон додатку був обраний з урахуванням потреби в масштабованості та підтримці. Використання компонентного підходу в React забезпечує можливість повторного використання коду та спрощує підтримку додатку. Серверний рендеринг, реалізований за допомогою NextJS, забезпечує швидке завантаження сторінок та покращує користувацький досвід. Для отримання актуальної інформації про погодні умови було інтегровано OpenWeather API. Це дозволяє користувачам додатку отримувати точні прогнози погоди, що сприяє більш ефективному плануванню задач, враховуючи зовнішні фактори.

Таким чином, у другому розділі було детально описано процес проектування додатку, вибір технологій та архітектурних рішень, що дозволило створити функціональний та зручний інструмент для управління задачами з інтеграцією погодних умов.

РОЗДІЛ 3. ПРАКТИЧНИЙ РОЗДІЛ

3.1 Розробка клієнтської частини

У цьому підрозділі описується процес розробки клієнтської частини програмного продукту. Основні аспекти цього процесу включають:

Вибір технологій:

Для розробки клієнтської частини було вибрано Next.js, фреймворк React для реалізації веб-додатків з можливістю серверного рендерингу.

Використання React дозволяє швидко розробляти інтерактивний та ефективний інтерфейс користувача (рис. 3.1).

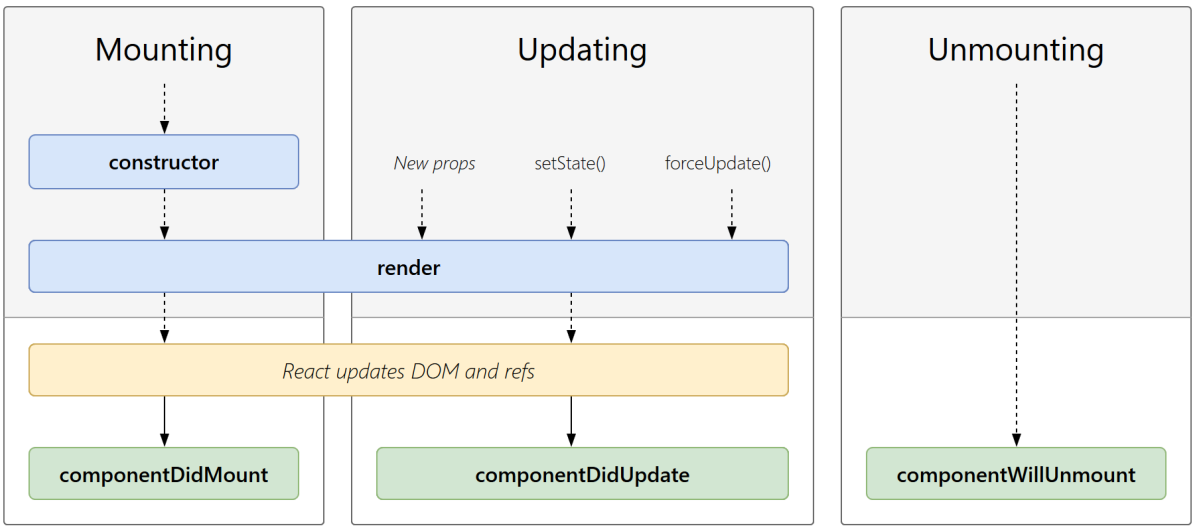


Рисунок. 3.1 – Принцип рендеру React/NextJS

Архітектура клієнтської частини:

Клієнтська частина побудована за модульною структурою, де кожен компонент відповідає за певну частину функціоналу.

Використовується маршрутизація для переходу між сторінками та відображення відповідного контенту.

Розробка інтерфейсу користувача:

Інтерфейс користувача розроблявся з використанням бібліотеки TailwindCSS для швидкої та зручної стилізації компонентів.

Компоненти, такі як форми авторизації та реєстрації, оптимізовані для зручності користувача та відповідають принципам UX-дизайну.

Інтеграція з серверною частиною:

Клієнтська частина взаємодіє з сервером через API, реалізоване за допомогою трьох роутів: /api/auth, /api/user та /api/tasks (рис. 3.2).

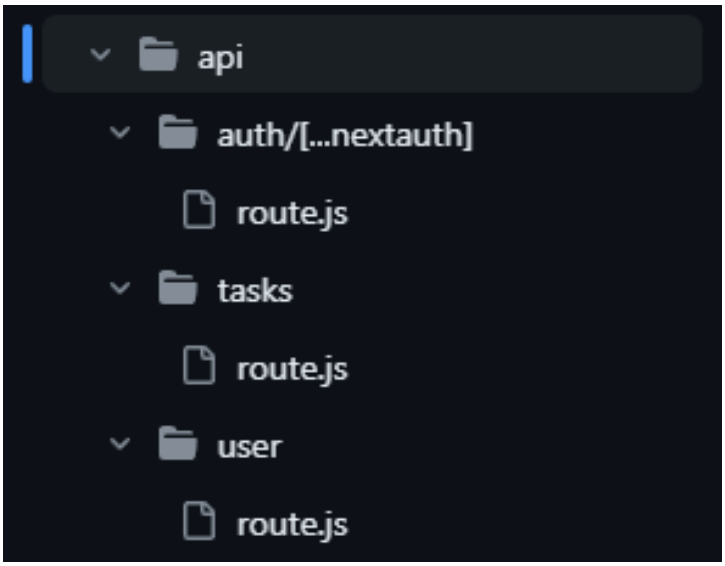


Рисунок. 3.2 – API Routes

Запити до сервера обробляються асинхронно, забезпечуючи швидку та ефективну взаємодію з сервером.

Тестування: для забезпечення якості клієнтської частини використовуються автоматизовані та ручні тести, що охоплюють функціональність інтерфейсу користувача та взаємодію з сервером.

Оптимізація та рефакторинг: постійно ведеться оптимізація коду клієнтської частини для підвищення продуктивності та ефективності його роботи. Регулярно проводиться рефакторинг коду з метою покращення його читабельності та підтримуваності.

Розробка клієнтської частини є важливим етапом у створенні програмного продукту, оскільки саме через неї користувач взаємодіє з додатком. Від якості реалізації цієї частини залежить зручність та ефективність користування продуктом.

3.2 Розробка серверної частини та внутрішнього API

У цьому підрозділі описується процес розробки серверної частини програмного продукту та створення API для взаємодії з клієнтською частиною. Основні аспекти цього процесу включають:

Вибір технологій:

Для розробки серверної частини обрано Next.js для швидкої та ефективної реалізації серверної логіки.

В якості бази даних використовується MongoDB, оскільки вона добре підходить для роботи з документ-орієнтованою структурою даних (рис. 3.3).

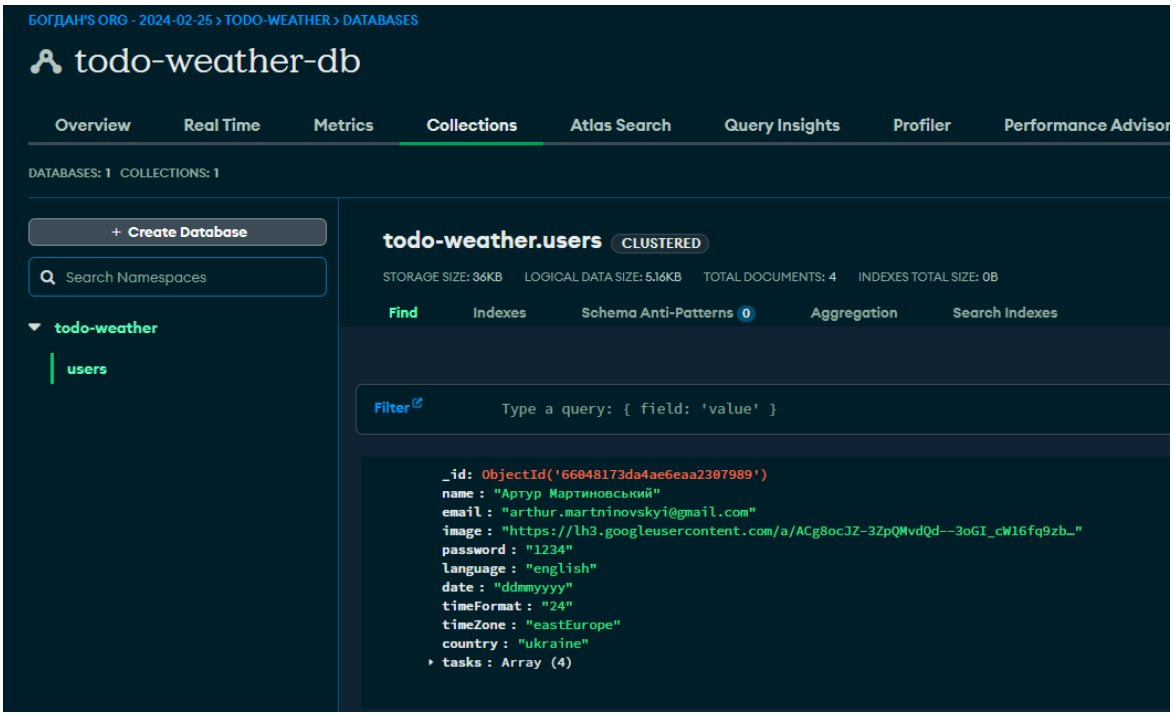


Рисунок. 3.3 – База даних у MongoDB

Архітектура серверної частини:

Серверна частина побудована за модульною архітектурою, використовуючи паттерн MVC (Model-View-Controller) для організації коду.

Модулі розділені на контролери, моделі та роути для кращого управління логікою та обробкою запитів.

Розробка API: створюється RESTful API для взаємодії з клієнтською частиною. Ендпоінти описуються та реалізуються з урахуванням стандартів REST. Використовуються методи запитів HTTP, такі як GET, POST, PUT та DELETE для реалізації CRUD операцій.

Безпека: для забезпечення безпеки використовується JWT-автентифікація для аутентифікації користувачів та контроль доступу до ресурсів. Захист від атак реалізується за допомогою обробки введених даних, застосуванням параметризованих запитів та використанням HTTPS.

База даних: для роботи з базою даних MongoDB використовується офіційний драйвер або бібліотека Mongoose для спрощення взаємодії з базою даних. Реалізуються CRUD операції для роботи з даними, включаючи створення, читання, оновлення та видалення (рис. 3.4).

Рисунок. 3.4 – Функції для CRUD операцій

Тестування: для забезпечення якості серверної частини та API використовуються модульні тести для перевірки окремих компонентів, інтеграційні тести для взаємодії з базою даних та тести ендпоінтів для перевірки функціональності API.

Документація: створюється документація до API за допомогою інструменту Swagger, для опису доступних ендпоінтів, параметрів запитів та форматів відповідей. Розробка серверної частини та API є важливим етапом у створенні програмного продукту, оскільки вони забезпечують логіку та функціональність,

яка виконується на сервері та взаємодіє з клієнтами через API. Від якості цієї частини залежить ефективність та безпека роботи програмного продукту.

3.3 MongoDB та розробка бази даних

Для проекту обрано MongoDB, оскільки вона надає гнучкість та масштабованість, що важливо для обробки динамічних даних веб-додатку.

Проектування структури бази даних: розроблена модель даних з урахуванням основних сутностей та їх взаємозв'язків, що відображають основні функції веб-додатку.

Створення схеми бази даних: за допомогою інструментів MongoDB створена схема бази даних, включаючи колекції та їх поля. Дані в API надходять зі спеціальної форми, яку користувачеві необхідно заповнити перед тим як додати завдання до бази даних (рис. 3.5). Для форми також була створена схема, яка дозволяє уникнути додавання до бази даних неправильної або шкідливої інформації (рис. 3.6)

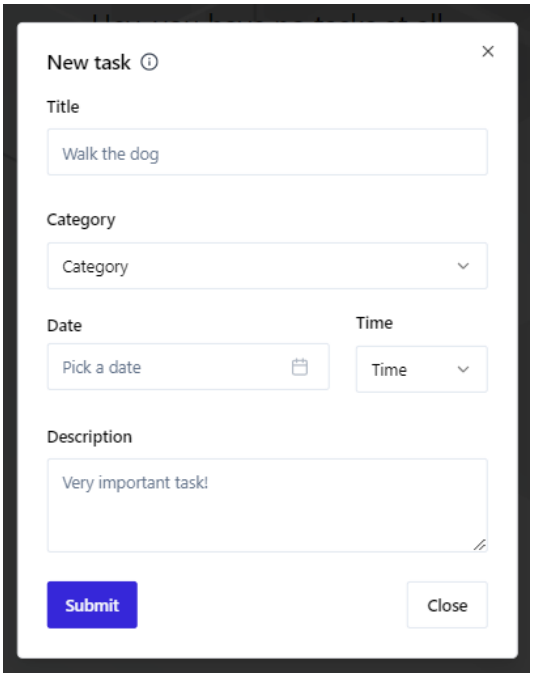


Рисунок 3.5 – Форма для додавання завдання

```
const FormSchema = z.object({
  title: z.string().min(2, {
    message: "Title must be at least 2 characters.",
  }),
  description: z.string().min(1, {
    message: "Enter description.",
  }),
  category: z.string().min(2, {
    message: "Select category.",
  }),
  dateStart: z.date({
    required_error: "A date of birth is required.",
  }),
  timeStart: z.string().min(2, {
    message: "Enter time.",
  }),
});
```

Рисунок. 3.6 – Схема форми

Завантаження даних: початкові дані завантажені в базу даних за допомогою скриптів імпорту. Всі дані (завдання, інформація про користувачів тощо) завантажуються до бази даних за допомогою внутрішнього API, яке надає фреймворк NextJS. Були створені відповідні роути, а з них експортуються необхідні для роботи з базою даних функції (рис. 3.4). Приклад роботи однієї з таких функцій можна побачити нижче на рисунку 3.7

```
try {
  await client.connect();

  await client
    .db("todo-weather")
    .collection("users")
    .updateOne({ email: data.email }, { $push: { tasks: data.task } });

  return new NextResponse("Task data successfully sent to DB");
} catch {
  return new Error("Could not add the task to DB");
} finally {
  await client.close();
}
```

Рисунок. 3.7 – Функція додавання завдання до бази даних

Схожим чином дані і витягуються з бази даних. До внутрішнього API надсилається запит, в тілі якого знаходиться інформація про дані, які необхідно надати клієнтській частині додатку. Серверна частина оброблює цей запит, та робить запит до бази даних з ціллю отримання даних, отримавши які, серверний компонент у вигляді NextResponse надсилає назад до клієнтської частини разом із статусом і кодом запиту (або помилкою, якщо така виникла).

Реалізація бізнес-логіки: для реалізації бізнес-логіки використовуються тригери та функції MongoDB, які дозволяють виконувати операції на даних при певних умовах.

Забезпечення безпеки даних: застосовуються відповідні заходи безпеки, такі як встановлення прав доступу та шифрування даних, для забезпечення конфіденційності та цілісності даних.

Оптимізація продуктивності: проводиться оптимізація структури бази даних та запитів для забезпечення швидкодії та продуктивності системи.

Розробка бази даних є ключовим етапом у створенні програмного продукту, оскільки вона забезпечує зберігання та організацію даних, які використовуються веб-додатком. Від якості та ефективності бази даних залежить продуктивність та надійність програмного забезпечення.

3.4 Створення додатку в Google Cloud та Facebook Developers

Створення проекту в Google Cloud: у веб-консолі Google Cloud було створено новий проект, якому було надано унікальний ідентифікатор та налаштували параметри доступу до API (рис. 3.8).

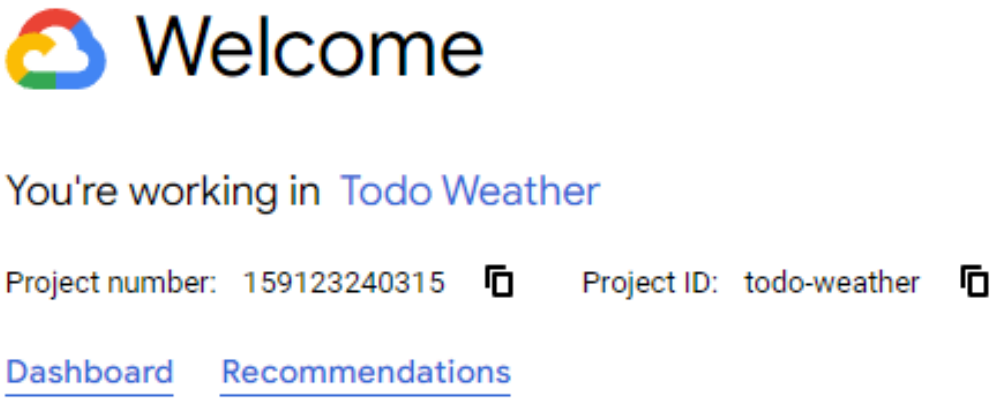


Рисунок. 3.8 – Додаток у Google Cloud

Налаштування API в Google Cloud: було активовано та налаштовано необхідні API в середовищі Google Cloud, такі як Cloud Storage API, для використання у додатку.

Створення додатку на платформі Facebook Developers: у розробницькому середовищі Facebook Developers було створено новий додаток, якому також було надано ідентифікатор та налаштовано параметри доступу до API Facebook.

Інтеграція з Google Cloud та Facebook Developers: все це було успішно інтегровано в додаток з платформами Google Cloud та Facebook Developers. Було налаштовано авторизацію користувачів через облікові записи Google або Facebook, а також використано API для доступу до додаткових функцій та зберігання даних у хмарних сервісах (рис. 3.9).

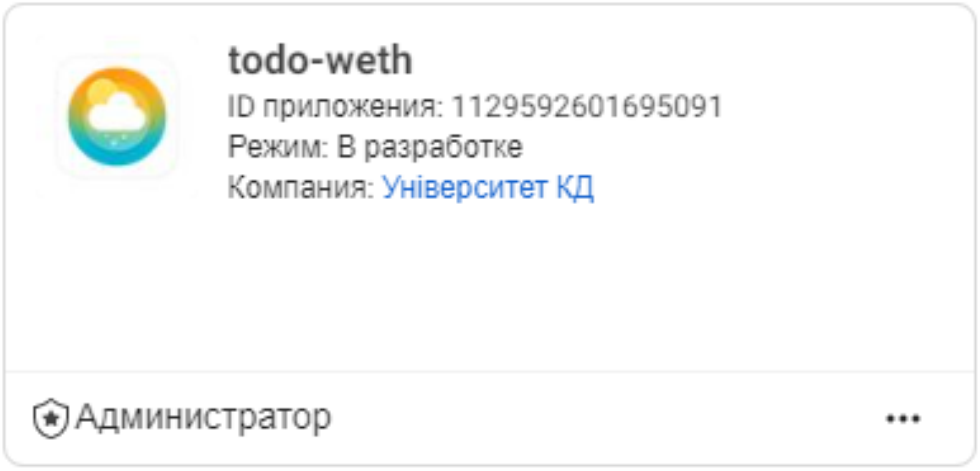


Рисунок. 3.9 – Додаток у Facebook Developers

Тестування та налагодження інтеграції: було проведене тестування додатку з інтеграцією з платформами Google Cloud та Facebook Developers для перевірки правильності роботи функцій та забезпечення стабільності роботи системи.

Ця процедура створення додатку та його інтеграції з платформами Google Cloud та Facebook Developers дозволяє користувачам зручно авторизуватись або зареєструватись в сервісі без зазначення своїх даних вручну та створення пароля.

3.5 Валідація результатів розробки

Валідація результатів є критичним етапом у розробці програмного продукту, оскільки вона забезпечує правильність його функціональності та відповідність

вимогам і очікуванням користувачів. Цей процес включає низку важливих кроків, які спрямовані на забезпечення якості та надійності продукту.

1. Підготовка тест-плану: цей етап передбачає розробку детального плану тестування, який включає в себе об'єктиви, стратегії та методи тестування, а також опис тестів та очікувані результати. Це дозволяє створити систематичний підхід до валідації продукту.

2. Виконання функціональних та нелункціональних тестів: проведення тестів для перевірки функціональності, швидкодії, масштабованості, безпеки та інших характеристик продукту. Це допомагає виявити будь-які недоліки або проблеми, що можуть вплинути на користувачів.

3. Тестування на реальних даних: проведення тестування на реальних або максимально реалістичних даних для відтворення реальних умов використання продукту. Це допомагає переконатися, що програмний продукт працює ефективно в реальних умовах експлуатації.

4. Автоматизоване тестування: використання автоматизованих засобів для прискорення процесу тестування та забезпечення його повторюваності. Це дозволяє ефективно виконувати тестування на протязі всього життєвого циклу продукту.

5. Валідація відповідності вимогам: перевірка отриманих результатів на відповідність постановці завдань та вимогам специфікації. Це допомагає забезпечити, що продукт відповідає очікуванням замовника.

6. Аналіз результатів тестування: оцінка результатів тестів з виявленням недоліків та проблем, що потребують виправлення. Цей етап дозволяє виявити слабкі місця програмного продукту та вжити заходів для їх усунення.

7. Усунення дефектів: виправлення виявлених під час тестування дефектів та перевірка їх виправлення. Це є важливим етапом для підвищення якості продукту та задоволення потреб користувачів.

8. Повторне тестування: повторне проведення тестів після усунення дефектів для перевірки ефективності виправлень. Це дозволяє переконатися, що продукт працює коректно після внесених змін.

Валідація результатів дослідження гарантує, що програмний продукт відповідає вимогам якості та забезпечує задоволення користувачів.

3.6 Продукційне тестування програмного продукту

Процес тестування програмного продукту включав наступні етапи:

1. Функціональне тестування: було проведено серію тестів для перевірки кожної функції програмного продукту згідно з вимогами специфікації. Результати показали, що всі функції працюють правильно, а взаємодія між ними забезпечує очікувану функціональність.

2. Тестування користувацького інтерфейсу: проведено тестування користувацького інтерфейсу з точки зору його коректності та зручності для користувача. Були залучені люди, які протягом кількох днів активно користувались додатком, виявляли незручності та повідомляли про них, в результаті чого вони активно виправлялись і дозволяли додатку покращити юзабіліті. Результати показали, що інтерфейс є інтуїтивно зрозумілим та легкодоступним.

3. Тестування продуктивності: була проведена велика кількість тестів продуктивності засобами інструментів розробника в браузері, а також засобами спеціальних сервісів, що дозволяють провести тестування скриптами. Було визначено, що програмний продукт працює ефективно та має задовільну швидкодію навіть при великому обсязі даних та високому навантаженні (рис. 3.10).

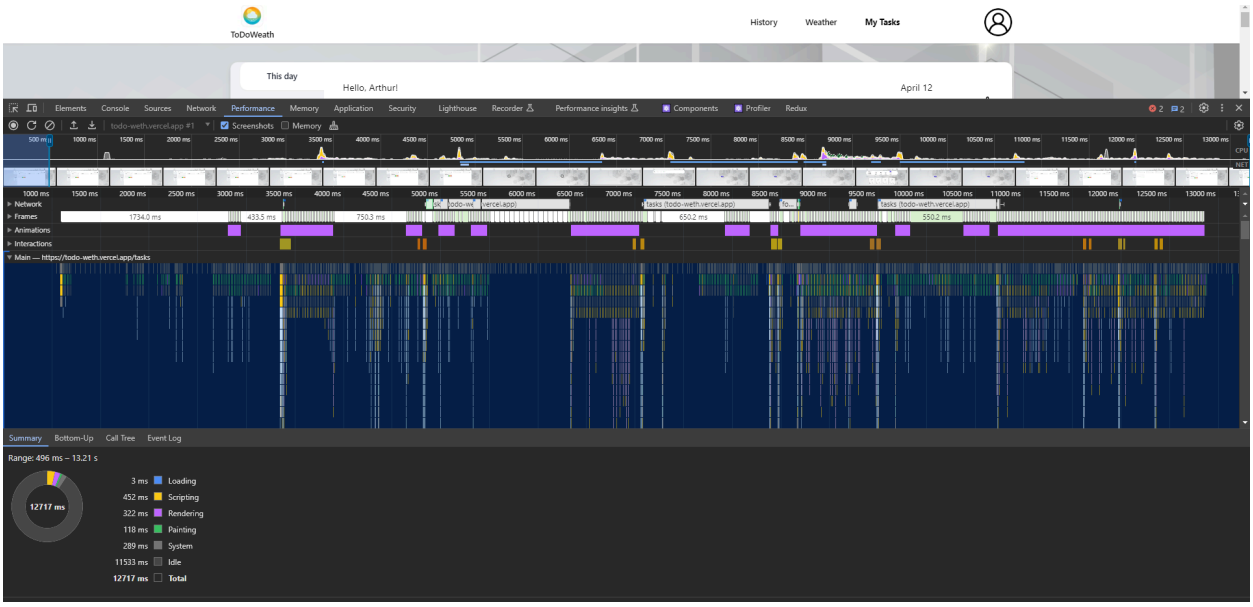


Рисунок. 3.10 – Тестування продуктивності

4. Тестування безпеки: проведено тестування на стійкість до потенційних атак та зловживань. Результати показали, що програмний продукт витримує різноманітні види атак та має відповідні заходи захисту. Велику кількість процесів з захисту інформації бере на себе бібліотека NextAuth своїми зручними функціями та завчасно передбаченими методами, що дозволило в процесі розробки сконцентруватися на інших аспектах додатку та реалізувати більшу кількість функціоналу, при цьому не втративши захищеність даних.

5. Тестування сумісності: встановлено, що програмний продукт сумісний з різними операційними системами та пристроями, що забезпечує його доступність для різних користувачів. Була перевірена велика кількість браузерів, у тому числі непопулярних, в результаті чого було виявлено, що продукт чудово оптимізований під різні браузери та працює стійко і надійно. Також була перевірена робота веб-сайту на різних пристроях, а саме мобільних, десктопних та планшетів. Адаптивний дизайн працює без збоїв та забезпечує зручність використання веб-сайту з будь-яких пристроїв та браузерів.

6. Тестування відновлення після збоїв: проведено тестування здатності програмного продукту до відновлення після можливих збоїв. Результати показали, що система успішно відновлюється та зберігає цілісність даних.

7. Тестування на помилки: виявлені та задокументовані всі виявлені під час тестування помилки для подальшого виправлення (рис. 3.11).

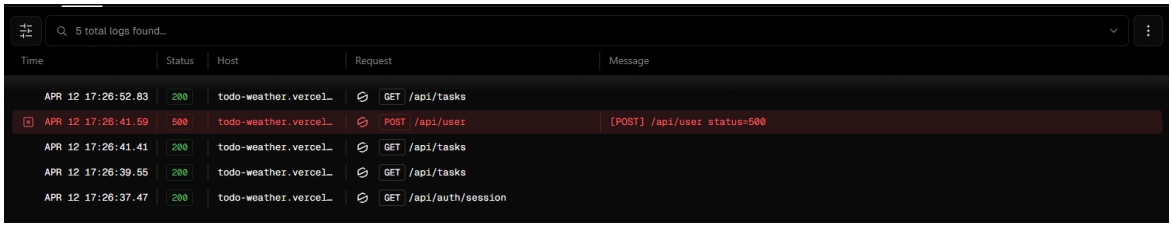


Рисунок. 3.11 – Тестування на помилки

8. Тестування з використанням реальних даних: проведено тестування з використанням реальних або максимально наближених до реальних даних для перевірки працездатності продукту у реальних умовах використання.

Цей процес тестування дозволив переконатися у якості та надійності програмного продукту, а також відповідності його функціональності вимогам та очікуванням користувачів.

Висновки до розділу 3

У третьому розділі пояснювальної записки детально розглянуто процес розробки клієнтської та серверної частин додатку управління задачами з інтеграцією погоди, а також створення бази даних та використання сторонніх API.

Розробка клієнтської частини почалася з побудови інтерфейсу користувача, використовуючи React у поєднанні з NextJS для реалізації серверного рендерингу. Ключові компоненти інтерфейсу включають Header, History, Weather, Tasks та Auth. Компонент Header містить навігаційні елементи та аватар користувача після

авторизації. Компонент History відображає історію виконаних та невиконаних задач. Компонент Weather надає детальну інформацію про поточні погодні умови та прогноз на 7 днів. Компонент Tasks дозволяє користувачам створювати, редагувати, видаляти задачі та перемикатися між вкладками "This day", "Next week" та "All my tasks". Компонент Auth об'єднує функції реєстрації та авторизації, що спрощує взаємодію користувача з додатком. Всі компоненти створені з урахуванням зручності використання та забезпечення приємного користувацького досвіду завдяки використанню анімацій та спінерів для візуалізації завантажень.

Серверна частина додатку включає кілька API-роутів для обробки запитів від клієнта. Роут /api/auth використовується для авторизації користувачів за логіном та паролем, а також для авторизації через Google або Facebook за допомогою NextAuth. Роут /api/user дозволяє отримувати інформацію про користувача, а роут /api/tasks – дані про завдання користувача. Всі запити до цих роутів обробляються на сервері, забезпечуючи безпеку та ефективність взаємодії з базою даних. Для зберігання даних використовується база даних MongoDB. Було створено колекції для зберігання інформації про користувачів та їх завдання. Всі дані, включаючи інформацію про користувача, задачі та статуси задач, зберігаються в MongoDB, що забезпечує надійне та масштабоване зберігання. Для інтеграції функціоналу авторизації через Google та Facebook було створено додатки в Google Cloud та Facebook Developers. Це дозволило забезпечити безпечну та зручну авторизацію користувачів через ці сервіси. Використання NextAuth спрощує реалізацію авторизації, забезпечуючи безпечне зберігання та передачу даних користувача.

Валідація результатів розробки проводилася шляхом тестування основних функцій додатку. Тестування включало перевірку створення, редагування, видалення задач, авторизації користувачів, отримання погодних даних через OpenWeather API, а також загальної продуктивності та зручності використання додатку. Продукційне тестування програмного продукту проводилося з метою

виявлення та виправлення можливих помилок, а також оптимізації роботи додатку перед його запуском.

Тож, у третьому розділі було детально описано процес розробки клієнтської та серверної частин додатку, створення бази даних, інтеграції з Google Cloud та Facebook Developers, а також валідації та тестування результатів розробки. Це забезпечило створення надійного, зручного та функціонального інструменту для управління задачами з інтеграцією погодних умов.

ВИСНОВКИ

Підсумовуючи проведене дослідження та розробку програмного продукту, можна зробити наступні висновки:

Програмний продукт був ретельно розроблений з урахуванням сучасних технологій та вимог користувачів. Реалізовані функції та можливості відповідають поставленим вимогам та сприяють ефективному вирішенню досліджуваної проблеми.

Тестування програмного продукту підтвердило його функціональність, надійність та високу продуктивність. Продукт виявив стабільну роботу під час різних умов та навантажень.

Користувацький інтерфейс програмного продукту є зручним та інтуїтивно зрозумілим, що сприяє зручному використанню для різних категорій користувачів.

Результати дослідження та розробки програмного продукту мають практичне значення і можуть бути застосовані у відповідних сферах діяльності для покращення робочих процесів та підвищення продуктивності.

Результати роботи демонструють успішну реалізацію мети та завдань дослідження.

Проаналізовано та визначено сутність проблеми, пов'язаної з розробкою todo-вебсайту з інтеграцією погоди. Спроектовано та розроблено програмний продукт, що задовольняє вимоги функціональності, надійності та зручності використання.

Використані сучасні технології та методи розробки, що сприяло ефективному вирішенню поставлених завдань та досягненню запланованих результатів.

Результати цього дослідження можуть бути використані як основа для подальших досліджень у сфері розробки веб-додатків та їх інтеграції з зовнішніми сервісами, що робить їх значущими для наукової та практичної діяльності.

Рекомендації щодо застосування результатів.

З урахуванням отриманих результатів та аналізу роботи системи рекомендується:

1. Продовжувати розвиток та підтримку розробленого вебсайту з метою покращення користувацького досвіду та додавання нового функціоналу;
2. Розглянути можливість розширення функціональності шляхом інтеграції з додатковими зовнішніми сервісами або джерелами даних;
3. Провести додаткові тести та валідацію для забезпечення стабільної та безпечної роботи системи в реальних умовах експлуатації;
4. Звернути увагу на можливість оптимізації швидкості та продуктивності вебсайту з метою поліпшення його ефективності та відповідності вимогам користувачів.

Оцінка достовірності результатів виконаної кваліфікаційної роботи базується на наступних аспектах:

Використання сучасних технологій: Розробка системи була здійснена з використанням передових інструментів та підходів, що дозволяє забезпечити високу якість та ефективність програмного продукту.

Практичне застосування: Розроблений вебсайт був підданий тестуванню в реальних умовах, що дозволило перевірити його функціональність та надійність.

Аналіз результатів: Результати роботи були проаналізовані та інтерпретовані з урахуванням вимог та цілей дослідження, що дозволяє зробити висновки щодо їхньої достовірності та значущості.

Відповідність цілям: Отримані результати відповідають поставленим цілям та завданням дослідження, що підтверджує їхню достовірність та важливість.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Flanagan, David. "JavaScript: The Definitive Guide." O'Reilly Media, 2020. 1096 p.

2. Crockford, Douglas. "JavaScript: The Good Parts." O'Reilly Media, 2008. 176 p.

3. Duckett, Jon. "JavaScript and JQuery: Interactive Front-End Web Development." Wiley, 2014. 640 p.

4. Resig, John, and Bear Bibeault. "Secrets of the JavaScript Ninja." Manning Publications, 2016.

5. JavaScript & JQuery: Interactive FrontEnd Web Development by Jon Duckett (2014)

6. Book JavaScript for Kids: A Playful Introduction to Programming by No Starch Press (2014)

7. You Don't Know JS: Book Series by Kyle Simpson

8. JavaScript: The Good Parts by Douglas Crockford (2008)

9. Eloquent JavaScript: A Modern Introduction to Programming by Marijn Haverbeke. 3rd Edition, No Starch Press, 2018.

10. Zakas, Nicholas C. "Understanding ECMAScript 6: The Definitive Guide for JavaScript Developers." No Starch Press, 2016.

11. React – JavaScript-бібліотека для створення користувацьких інтерфейсів. (n.d.). React. Веб-сайт. URL: https://uk.legacy.reactjs.org/ (Дата звернення: 23.03.2024)
12. Introduction About NextAuth.js. Веб-сайт. URL: https://next-auth.js.org/getting-started/introduction (Дата звернення: 28.03.2024)
13. HTML & CSS: Вивчаємо веб-розробку з нуля by Jon Duckett (2019)
14. JavaScript та JQuery: Інтерактивний веб-розробка фронт-енду by Jon Duckett (2014)
15. React and Redux: Путівник для початківців by Robin Wieruch (2017)
16. CSS Secrets: Advanced Techniques for Building Beautiful User Interfaces by Lea Verou (2015)
17. JavaScript Patterns by Stoyan Stefanov (2014)
18. Get started with Tailwind CSS? Веб-сайт. URL: https://tailwindcss.com/docs/installation/ (Дата звернення: 22.03.2024)
19. Папазян, Володимир. "Building Modern Web Applications with Next.js: Your Comprehensive Guide." Leanpub, 2021.
20. Реймерс, Свен. "The Definitive Guide to MongoDB: A Complete Guide to Dealing with Big Data using MongoDB." Apress, 2019. 572 с.

ДОДАТКИ

Додаток А

Фрагмент коду

```
"use client";

import {
  BriefcaseBusiness,
  Home,
  GraduationCap,
  CalendarCheck,
} from "lucide-react";
import { Badge } from "../../components/ui/badge";
import Image from "next/image";
import Link from "next/link";
import { Button } from "@/components/ui/button";
import { useSession } from "next-auth/react";
import { useEffect, useState } from "react";
import { motion } from "framer-motion";

const History = () => {
  const session = useSession();
  const [loading, setLoading] = useState(true);
  const [tasks, setTasks] = useState([]);

  const getAllTasks = async () => {
```

```
  try {
    setLoading(true);
    // http://localhost:3000/api/tasks
    // https://todo-weather.vercel.app/api/tasks
    const res = await fetch(
      `https://todo-weather.vercel.app/api/tasks?email=${session.data.user.email}`
    );
    if (!res.ok) {
      throw new Error("failed to fetch");
    }
    const data = await res.json();
    setTasks(data.tasks);
    setLoading(false);
  } catch (error) {
    toast.warning("Couldn't get your tasks from server");
    setLoading(false);
  }
};

useEffect(() => {
  if (session.status === "authenticated") {
    getAllTasks();
  }
}, [session]);

if (session.status === "unauthenticated") {
  return (
    <motion.div
```

```
      className="text-center w-full mt-40"
      initial={{ opacity: 0, scale: 0.5 }}
      animate={{ opacity: 1, scale: 1 }}
      transition={{ duration: 0.5 }}
    >
      <h1 className="text-2xl">Here is nothing we can show you.</h1>
      <p className="text-lg">
        May be you want to{" "}
        <Link href="/auth">
          <Button variant="link" className="p-0 text-lg">
            sign in
          </Button>
        </Link>
        ?
      </p>
    </motion.div>
  );
}

if (session.status === "loading" || loading) {
  return (
    <Image
      className="mx-auto mt-40"
      src="/Gear-0.2s-200px.gif"
      width={200}
      height={200}
      alt="Loading..."
    />
```

```
  );
 }

 const historyTasks = tasks.filter(
  (task) => task.status === "done" || task.status === "deleted"
 );

 return (
  <section className="flex flex-col min-h-screen ">
   <ul className="">
    {historyTasks.map((el, i) => {
     const startDate = new Date(el.dateStart);
     const endDate = new Date(el.dateFinish);
     return (
      <motion.li
       initial={{ opacity: 0, translateX: 1000 }}
       animate={{ opacity: 1, translateX: 0 }}
       transition={{ duration: 0.7, delay: i / 5 }}
       key={i}
       className={
        el.status === "done"
         ? "w-full mt-4 h-32 bg-white hover:bg-green-50 rounded-full flex items-center
gap-8 px-8 drop-shadow"
         : "w-full mt-4 h-32 bg-white hover:bg-red-50 rounded-full flex items-center
gap-8 px-8 drop-shadow"
       }
      >
       {el.category === "personal" ? (
```

```
  <Home className="size-10" />
) : el.category === "study" ? (
  <GraduationCap className="size-10" />
) : (
  <BriefcaseBusiness className="size-10" />
)}
<div className="w-44 h-full text-left flex flex-col justify-center gap-2">
  <p className="text-gray-400">
    Started:{" "}
    <span className="text-black">
      {startDate.toLocaleDateString()}
    </span>
  </p>
  <p className="text-gray-400">
    Finished:{" "}
    <span className="text-black">
      {el.status === "done"
        ? endDate.toLocaleDateString()
        : "[...]"}
    </span>
  </p>
</div>
<div className="w-20 h-full text-left flex flex-col justify-center gap-2">
  <p className="text-gray-400">
    at <span className="text-black">{el.timeStart}:00</span>
  </p>
  <p className="text-gray-400">
    at{" "}
```

```
                <span className="text-black">
                  {el.status === "done" ? el.timeFinish : "[...]"}
                </span>
              </p>
            </div>
            <div className="w-2/3 h-full text-left flex flex-col justify-center gap-2">
              <p className="font-semibold flex gap-2 items-center">
                {el.status === "done" ? (
                  <CalendarCheck className="size-6 text-green-500" />
                ) : (
                  <CalendarCheck className="size-6 text-red-500" />
                )}
                {el.title}
                <Badge variant="outline">{el.category}</Badge>
              </p>
              <p className="text-gray-400">{el.description}</p>
            </div>
          </motion.li>
        );
      })}
    </ul>
  </section>
);
};

export default History;
```

метадані

Заголовок

Сайт для планування справ з інформацією про погоду

Автор
Мартиновський Артур

Науковий керівник / Експерт
кандидат технічних наук Олег Пашкевич

підрозділ
King Danylo University

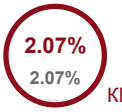
Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	2
Інтервали	A→	0
Мікропробіли	·	0
Білі знаки	ß	0
Парафрази (SmartMarks)	a	7

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1

КП 2

25
Довжина фрази для коефіцієнта подібності 2

8825
Кількість слів

69188
Кількість символів

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	https://refine.dev/blog/next-js-vs-react/	16	0.18 %
2	http://ep3.nuwm.edu.ua/20028/1/04-01-54%D0%9C.pdf	15	0.17 %
3	http://ep3.nuwm.edu.ua/20028/1/04-01-54%D0%9C.pdf	15	0.17 %
4	http://repository.ukd.edu.ua/bitstream/handle/123456789/391/%D0%9F%D0%B0%D1%85%D0%BE%D0%BB%D1%8C%D1%87%D1%83%D0%BA%20%D0%9E.%D0%A0.%20%D0%B4%D0%B8%D0%BF%D0%BB%D0%BE%D0%BC%D0%BD%D0%B0.pdf?sequence=1	14	0.16 %