НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри Інформаційних систем і технологій

_____ Швиденко Михайло Зіновійович
Підпис ініціали та прізвище

___ _____202_ р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

Веб-застосунок для управління особистими завданнями

Спеціальність 126 “Інформаційні системи та технології”

Гарант освітньої програми
Канд.екон.наук, доцент _____ Мокрієв Максим Володимирович
(науковий ступінь та вчене звання) (підпис) (ПіБ)

Керівник кваліфікаційної роботи
Професор, д-р.тех.наук _____ Смолій Вікторія Миколаївна
(науковий ступінь та вчене звання) (підпис) (ПіБ)

Виконав _____ Іскоростенський Олексій Олександрович
(підпис) (ПіБ)

Київ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Завідувач кафедри Інформаційних систем і технологій

_____ Швиденко Михайло Зіновійович
Підпис ініціали та прізвище
___ _____202_ р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студенту Іскоростенському Олексію Олександровичу

Спеціальності 126 “Інформаційні системи та технології”

1. Тема роботи: **“Веб-застосунок для управління особистими завданнями”** Затверджена наказом ректора від 16.12.2024 №2245С
2. Термін подання завершеної роботи на кафедру - – 10.06.2025р
3. Вихідні дані: вимоги клієнта до веб-застосунку управління особистими завданнями
4. Перелік питань, що розглядаються:

1. Які недоліки існуючих інструментів та потреби користувачів спонукали до розробки нового веб-застосунку для управління особистими завданнями?
2. Які архітектурні рішення (наприклад, клієнт-сервер) та ключові технології (для фронтенду, бекенду, бази даних) обрані для реалізації веб-застосунку?
3. Які основні функціональні можливості (створення, редагування завдань, встановлення дедлайнів, нагадування, робота з файлами) будуть реалізовані в застосунку?
4. Як буде забезпечено зручність використання, безпеку даних та проведено тестування розробленого веб-застосунку?

5. Календарний план

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Дослідження предметної області та постановка задачі	Вересень 2024 - Листопад 2024	Включає аналіз предметної області, існуючих ІТ, визначення вимог та розробку технічного завдання (розділ 1).
2	Проєктування інструментальних засобів створення та використання інформаційних систем і технологій	Грудень 2024 - Лютий 2025	Включає визначення проектних рішень, розробку логічної/фізичної структури, декомпозицію, розробку ЛМІ (розділ 2).
3	Реалізація та тестування розробленої інформаційної системи	Березень 2025 - Квітень 2025	Включає опис модулів, розробку ЛМІ, визначення технічних характеристик, тестування системи (розділ 3).
4	Оформлення результатів роботи, написання висновків та підготовка до захисту	Травень 2025	Написання вступу, висновків, списку джерел, оформлення додатків та підготовка презентації до захисту.

Керівник кваліфікаційної роботи
Професор, д-р.тех.наук _____ Смолій Вікторія Миколаївна
(науковий ступінь та вчене звання) (підпис) (ПіБ)

Завдання прийняв до виконання ____ Іскоростенський Олексій Олександрович
(підпис) (ПіБ)

Дата отримання завдання ___.___.____

Зміст

Вступ

У сучасному світі ефективне управління особистими завданнями стає все актуальнішим. Швидкий темп життя вимагає інструментів, які допомагають систематизувати, впорядковувати та контролювати виконання планів. Особливо важливими тут стають веб-застосунки, адже вони дозволяють керувати власними завданнями будь-де та з будь-якого пристрою.

Актуальність теми

Сучасний ритм життя вимагає ефективних інструментів для управління особистими справами. Хоча на ринку існує багато рішень, не всі вони однаково зручні, швидкі та зрозумілі. Саме тому ми вирішили розробити новий веб-застосунок, який би зосередився на зручній реєстрації, інтуїтивному управлінні нотатками та шаблонами, а також легкій роботі з файлами. Таке завдання видається нам перспективним в IT-сфері.
Мета цієї роботи – розробити веб-застосунок для управління особистими завданнями, який допоможе користувачам ефективно організовувати свої справи, краще контролювати їх виконання та підвищити власну продуктивність.

Щоб досягти поставленої мети, ми визначили такі основні завдання:

- Виконати аналіз предметної області та існуючих програмних рішень;
- Встановити функціональні вимоги до веб-застосунку;
- Розробити архітектуру системи;
- Реалізувати інтерфейс користувача;
- Реалізувати серверну частину;
- Забезпечити реєстрацію та авторизацію;
- Реалізувати збереження файлів;
- Здійснити програмну реалізацію серверної та клієнтської частини застосунку;
- Провести тестування працездатності та зручності використання системи.

Об’єктом дослідження є процес цифрової організації особистих завдань з використанням веб-технологій.

Предмет дослідження є архітектурні рішення, засоби для функціональної реалізації веб-застосунку для управління особистими нотатками та шаблонами з можливістю прикріпити файл.

Система реалізована за архітектурою «клієнт-сервер». Клієнтська частина створена за допомогою бібліотеки React і збирається інструментом Vite, що

забезпечує гнучкість в розробці і компонуванні. Інтерфейс розроблено відповідно до зручності, з адаптивною верстkою. Серверна частина побудована на Node.js з використанням Express та MongoDB для зберігання даних. Для керування даних застосовано бібліотеку Multer. Авторизація реалізована через JWT.

Наш веб-застосунок пропонує універсальний підхід до щоденного планування, поєднуючи необхідний функціонал зі зручністю використання.

1. Дослідження предметної області та постановка задачі

1.1 Аналіз предметної області, її структурна та функціональна особливість.

В сучасному інформаційному світі, де так багато відволікаючих факторів, ефективне управління власними справами стає надзвичайно важливим. Веб-застосунки, що допомагають систематизувати та виконувати щоденні завдання, стають тут у пригоді. Головне, щоб такі інструменти були доступними, функціональними та простими у використанні. Предметна область управління особистими завданнями охоплює сукупність процесів і дій, пов’язані із створенням, редагуванням, прикріпленням, плануванням справ, які виконує користувач. У загальному випадку це завдання повсякденного життя або робочого характеру, що вимагають фіксації, структурування, визначення термінів виконання та при потребі створення шаблону для повторного виконання.

Функціональна модель системи управління завданнями включає такі компоненти:

- Модуль автентифікації користувача – забезпечує індивідуальний доступ до системи та збереження персональних даних;
- Модуль створення та редагування завдань – дозволяє формувати текстові нотатки або за допомогою голосового введення , встановлювати теги та дедлайни;
- Модуль шаблонів – надає змогу зберігати заготовку для нотаток або завдань для повторного використання;
- Модуль прикріплення файлів – дозволяє додавати до завдань супровідні файли у PDF, txt, docx при не можливості відкриття в браузері файл завантажується на пристрій;
- Інтерфейс користувача – реалізує доступ до функцій системи через веб-браузер, забезпечуючи зручність використання.

З технічної сторони, предметна область має такі структурні особливості:

- Потреба у зберіганні структурованих даних
- Необхідність підтримки прикріплених файлів різних форматів
- Забезпечення безпечного доступу
- Адаптація під різні пристрої

Щоб бути справді корисними, сучасні системи для управління завданнями повинні бути не просто функціональними, але й легко масштабуватися, бути доступними з будь-якого пристрою та гнучкими у використанні. Тому, розробляючи цей веб-застосунок, ми зосередилися не тільки на базових операціях з нотатками, але й на таких важливих аспектах, як пошук, оновлення, можливість закріплення важливих записів та підтримка англійської мови.

Отже, розробка програмного забезпечення для управління особистими завданнями залишається актуальною, адже такі інструменти допомагають користувачам бути більш продуктивними завдяки кращій організації своєї діяльності.

1.2 Аналіз наявних інформаційних технологій предметної області.

Для організації особистих завдань в наш час існує чимала кількість веб-застосунків які забезпечують базову або розширену функціональність щодо планування справ. Ці застосунки використовуюся в повсякденному житті, так і в професійній діяльності. Аналіз наявних програмних рішень дозволяє виділити їх ключові функціональні можливості, переваги та обмеження, що в подальшому дає змогу сформувати вимоги до власного веб-застосунку.

Для аналізу обрано чотири популярні системи управління завданнями: Todoist [20], Microsoft To Do [8], Notion [14] і Google Keep [5]. Ці продукти в цій ніші є лідерами ринку завдяки широкому функціоналу, великій користувацькій базі та різноманітним підходам до організації завдань. Todoist і Microsoft To Do представляють спеціалізовані інструменти для планування, Notion вирізняється гнучкістю й можливостями кастомізації, а Google Keep орієнтований на простоту

та інтеграцію з екосистемою Google. Такий вибір дозволяє охопити різні аспекти предметної області та виявити прогалини.

Для кращого розуміння предметної області проведено порівняння її ключових функціональних і структурних особливостей із можливостями існуючих систем управління завданнями. Порівняння виконано за такими ознаками: підтримка голосового введення, робота з файлами, шаблони, локалізація, адаптивність інтерфейсу та безпека. Результати представлено в (табл 1.1).

Таблиця 1.1 – порівняння існуючих рішень

Ознака	ToDoist	Microsoft To Do	Notion	Google Keep	Пропонований веб-застосунок
Голосове введення	Не підтримує	Не підтримує	Не підтримує	Підтримує (обмежено, без укр. мови)	Підтримує (укр. та англ. мови)
Робота з файлами	Обмежена (лише преміум, до 100 МБ)	Не підтримує	Підтримує (PDF, зображення, обмежено безкоштовно)	Підтримує зображення	Підтримує (JPEG, PNG, PDF, DOC, DOCX, до 5 МБ)
Шаблони завдань	Підтримує (обмежено, преміум)	Не підтримує	Підтримує (гнучкі, кастомізовані)	Не підтримує	Підтримує (гнучкі, безкоштовно)
Локалізація	Англійська, часткова укр. підтримка	Повна укр. підтримка	Англійська, без укр. підтримки	Повна укр. підтримка	Повна укр. та англ. підтримка
Адаптивність інтерфейсу	Висока (веб, мобільні додатки)	Висока (веб, мобільні додатки)	Висока (але складний для новачків)	Висока (простий, але обмежений)	Висока (адаптивний, мінімалістичний)
Безпека	JWT, шифрування (преміум-функції)	Шифрування, інтеграція з MS	JWT, шифрування (преміум-функції)	Шифрування, Google-екосистема	JWT, шифрування паролів (bcrypt)

Аналіз порівняння:

Голосове введення: Жодна з розглянутих систем, крім Google Keep, не підтримує голосове введення, але навіть Google Keep має обмежену підтримку української мови. Пропонований веб-застосунок пропонує голосове введення з підтримкою української та англійської мов через Web Speech API, що відповідає вимогам локального ринку.

Робота з файлами: Todoist і Notion дозволяють прикріплювати файли, але лише в преміум-версіях або з обмеженнями. Google Keep підтримує лише зображення, а Microsoft To Do не має цієї функції. Пропонований застосунок підтримує популярні формати файлів (JPEG, PNG, PDF, DOC, DOCX) із чіткими обмеженнями (5 МБ, до 5 файлів на нотатку), що забезпечує зручність і доступність.

Шаблони завдань: Notion пропонує гнучкі шаблони, але вони складні для новачків. Todoist підтримує шаблони лише в преміум-версії. Microsoft To Do і Google Keep не мають цієї функції. Пропонований застосунок включає безкоштовні шаблони, що спрощує повторне використання завдань.

Локалізація: Microsoft To Do і Google Keep мають повну підтримку української мови, тоді як Todoist і Notion обмежені в цьому аспекті. Пропонований застосунок забезпечує повну підтримку української та англійської мов, що відповідає вимогам локального ринку.

Адаптивність інтерфейсу: Усі системи мають адаптивні інтерфейси, але Notion складний для новачків, а Google Keep обмежений у функціоналі. Пропонований застосунок пропонує мінімалістичний і адаптивний інтерфейс.

Безпека: Усі системи використовують шифрування, але лише Todoist і Notion застосовують JWT у преміум-версіях. Пропонований застосунок використовує JWT і bcrypt для захисту даних, що забезпечує високий рівень безпеки без додаткових витрат.

Проведений аналіз показує, що предметна область управління особистими завданнями вимагає поєднання гнучкості, простоти, безпеки та локалізації. Жодна з розглянутих систем не забезпечує повного набору необхідних функцій, особливо в контексті голосового введення, роботи з файлами та безкоштовних шаблонів із підтримкою української мови. Пропонований веб-застосунок спрямований на заповнення цих прогалин, пропонуючи адаптивний інтерфейс, підтримку локалізації, безпечне зберігання даних і розширені можливості для

роботи з файлами та шаблонами. Це робить його актуальним рішенням для підвищення продуктивності користувачів у повсякденному та професійному житті.

Реалізація серверної частини

Серверна частина веб-застосунку побудована на платформі Node.js із використанням фреймворку Express.js для створення REST API. Такий вибір зумовлений асинхронною природою Node.js, яка дозволяє ефективно обробляти велику кількість одночасних запитів, що є важливим для масштабованості системи [13]. Express.js забезпечує простоту створення маршрутів для обробки HTTP-запитів, таких як створення, редагування, видалення нотаток і шаблонів, а також управління автентифікацією [4]. Для оцінки вибору серверного фреймворку розглянемо порівняння Express.js з альтернативними рішеннями, такими як Django Python [3] і Spring Java [18], у (табл 1.2).

Таблиця 1.2 – Порівняння серверних фреймворків

Характеристика	Express.js	Django	Spring
Мова програмування	JavaScript (Node.js)	Python	Java
Продуктивність	Висока завдяки асинхронній обробці запитів	Середня, синхронна модель	Висока, але залежить від JVM
Простота реалізації	Проста, мінімалістична структура, швидке створення API	Складніша через вбудовані функції (ORM, адмін-панель)	Складна, потребує більше конфігурації
Екосистема	Велика, інтеграція з MongoDB, Multer, JWT	Велика, вбудована ORM (PostgreSQL, MySQL), підтримка шаблонів	Велика, але складніша інтеграція з не-Java технологіями
Гнучкість	Висока, дозволяє кастомізацію	Середня, орієнтована на швидку розробку з шаблонами	Висока, але потребує більше коду для простих задач
Розмір спільноти	Велика, активна підтримка	Велика, популярна в Python-спільноті	Велика, популярна в корпоративному секторі
Час розробки	Швидкий завдяки простоті та JavaScript-екосистемі	Середній через необхідність конфігурації ORM	Довший через складність налаштування

Обґрунтування вибору Express.js: Ми зупинили свій вибір на Express.js з кількох причин [4]. По-перше, він швидкий, простий у використанні та добре інтегрується з JavaScript-середовищем, яке ми вже використовували для фронтенду на React. По-друге, асинхронність Node.js дозволяє ефективно обробляти багато одночасних запитів, що важливо для продуктивності нашого застосунку. На відміну від Django, Express.js не накладає обмежень на структуру проєкту, що забезпечує гнучкість у розробці. Spring, хоча й потужний, є складнішим у налаштуванні та менш сумісним із MongoDB, що використовується в проєкті. Велика спільнота та екосистема бібліотек (наприклад, Multer, JWT) роблять Express.js оптимальним вибором для швидкої розробки REST API.

Вибір бази даних

Для зберігання даних використано MongoDB– нереляційну базу даних, яка підтримує гнучке зберігання структурованих і неструктурованих даних [9]. MongoDB ідеально підходить для зберігання нотаток, шаблонів і метаданих файлів завдяки своїй здатності працювати з документами у форматі JSON. Для спрощення взаємодії з базою даних використано ORM-бібліотеку Mongoose, яка забезпечує зручне створення схем і валідацію даних [10]. Наприклад, схема для колекції Notes включає поля title, content, tags, isPinned, userId, deadline, reminder та attachments, що дозволяє гнучко зберігати всю необхідну інформацію про нотатки.

Для обґрунтування вибору MongoDB розглянемо порівняння з реляційними базами даних PostgreSQL [15] і MySQL [12] у (табл 1.3).

Таблиця 1.3 – Порівняння баз даних

Характеристика	MongoDB	PostgreSQL	MySQL
Тип бази даних	Нереляційна (документоорієнтована)	Реляційна	Реляційна
Структура даних	Гнучка, JSON-подібні документи	Таблична, строга схема	Таблична, строга схема
Масштабованість	Горизонтальна (шардинг), ефективна для великих даних	Вертикальна, складніша горизонтальна масштабованість	Вертикальна, обмежена горизонтальна масштабованість
Швидкість обробки	Висока для неструктурованих даних	Висока для складних запитів і транзакцій	Висока для простих запитів
Підтримка файлів	Зберігання метаданих файлів, інтеграція з Multer	Складніше через BLOB або зовнішнє сховище	Складніше через BLOB або зовнішнє сховище
Простота інтеграції	Проста з JavaScript (Node.js, Mongoose)	Потребує ORM (наприклад, Sequelize)	Потребує ORM (наприклад, Sequelize)
Гнучкість	Висока, легко адаптується до змін у структурі даних	Обмежена, потребує міграцій для зміни схеми	Обмежена, потребує міграцій для зміни схеми

Обґрунтування вибору MongoDB: Наш вибір на користь MongoDB зумовлений її гнучкістю у роботі з неструктурованими даними, що ідеально підходить для зберігання нотаток і шаблонів із різноманітними полями, такими як теги, дедлайни та прикріплені файли. Документоорієнтована модель дозволяє швидко адаптувати структуру даних до нових вимог без складних міграцій, на відміну від реляційних баз, таких як PostgreSQL і MySQL, які потребують чітко визначених схем. До того ж, MongoDB легко масштабується горизонтально, що дозволяє ефективно працювати з великими обсягами даних, а це важливо для подальшого розвитку нашої системи. Крім того, проста інтеграція з Node.js через Mongoose і підтримка JSON-подібного формату спрощують розробку та зменшують час реалізації порівняно з реляційними базами, які потребують додаткових ORM або складних конфігурацій для роботи з файлами.

1.3 Визначення вимог до інструментальних засобів створення і використання інформаційних систем та технологій, розробка технічного завдання.

Функціональні вимоги

Функціональні вимоги визначають основні можливості, які має забезпечувати веб-застосунок для управління особистими завданнями:

1. Автентифікація та авторизація користувача:
    - Автентифікація нового користувача починається з введення імені, електронної пошти та пароля.
    - Авторизація за допомогою електронної пошти та пароля.
    - Можливість зміни пароля авторизованим користувачем.
    - Вихід із системи з виходом активної сесії.
2. Управління нотатками:
    - Створення, редагування, видалення та перегляд нотаток.
    - Додавання до нотаток заголовка, вмісту, тегів, дедлайнів.
    - Закріплення пріоритетних нотаток.
    - Пошук нотаток за ключовими словами.
3. Голосове введення:
    - Можливість введення тексту в нотатках за допомогою голосового розпізнання з підтримкою української та англійської мов.
    - Підтримка проміжного тексту під час розпізнавання.
4. Управління шаблонами:
    - Створення, редагування, видалення та перегляд шаблонів для нотаток.
    - Застосування шаблонів для швидкого створення нотаток із уже попередньо визначеними полями.

5. Прикріплення файлів:
    - Додавання до нотаток та шаблонів файлів форматів PEG, PNG, PDF, DOC, DOCX.
    - Перегляд, завантаження та перейменування прикріплених файлів.
    - Видалення файлів з нотаток або шаблонів.
6. Календар:
    - Відображення нотаток з дедлайном у форматі календаря місяць, тиждень, день, година.
    - Можливість створення нотатки вибором дати в календарі.
    - Редагування нотатки кліком на подію в календарі.
7. Локалізація:
    - Підтримка англійської та української мов для інтерфейсу, повідомлень і форматування дат.
    - Визначення мови з можливістю ручного вибору.
8. Тема інтерфейсу:
    - Перемикання між світлою та темною темами та збереження обраної теми в локальному сховищі браузера

Для наочності взаємодії користувача з системою наведено діаграму прецедентів (Рис. 1.1), яка відображає основні функціональні можливості веб-застосунку.

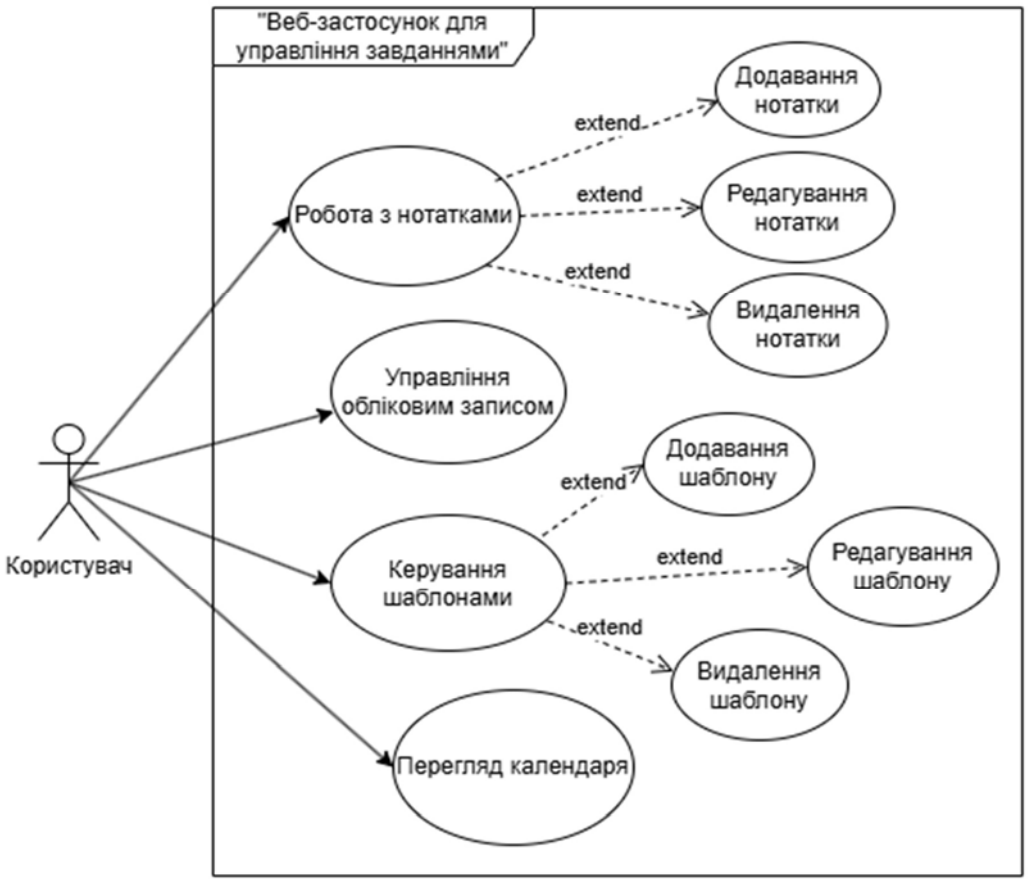

Рисунок 1.1 – Діаграма прецедентів

Діаграма прецедентів ілюструє взаємодію користувача із веб-застосунком для управління особистими завданнями. Основний актор "Користувач" виконує низку дій, які представлені як прецеденти. Кожен прецедент відображає ключову функціональність системи:

- "Робота з нотатками" включає створення, редагування та видалення нотаток, а також додавання тегів, дедлайнів та прикріплення файлів.
- "Управління обліковим записом" охоплює реєстрацію, авторизацію, зміну пароля та вихід із системи.
- "Керування шаблонами" дозволяє створювати, редагувати та застосовувати шаблони для повторного використання нотаток.
- "Перегляд календаря" забезпечує доступ до нотаток із дедлайнами в різних форматах (день, тиждень, місяць).

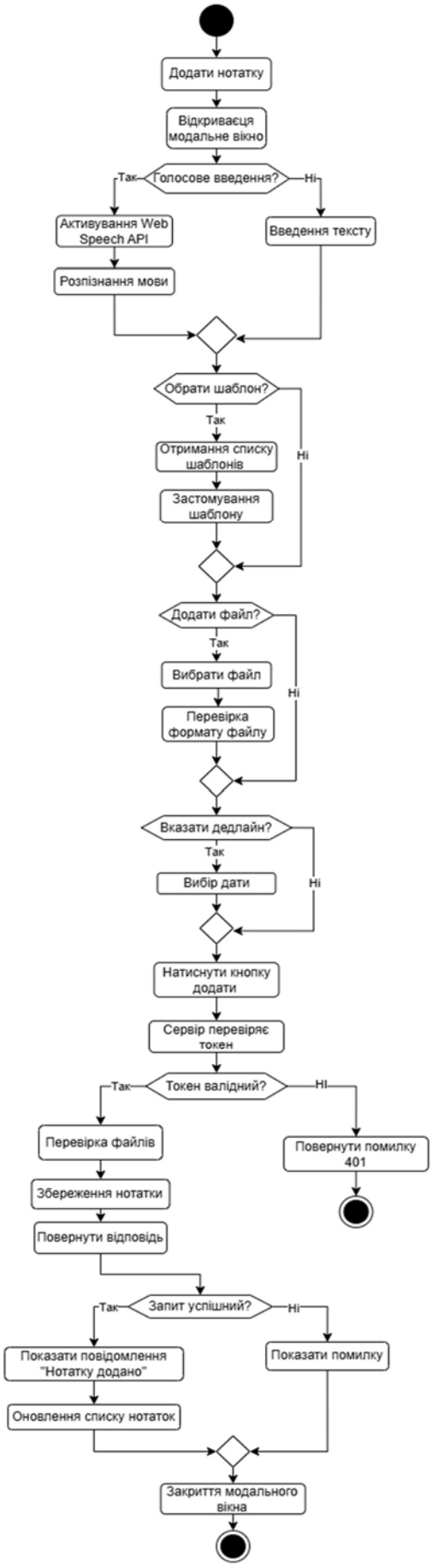

Рисунок 1.2 – Діаграма активності процес створення нотатки

На (Рис. 1.2) показано діаграму активності, яка демонструє повний цикл створення нотатки в нашому веб-застосунку. Все починається, коли користувач натискає кнопку «Додати нотатку», після чого відкривається модальне вікно. Користувач має можливість обрати спосіб введення даних, зокрема текстове введення або голосове введення через Web Speech API, додати файли з урахуванням обмежень щодо формату), а також вибрати шаблон для автоматичного заповнення полів. Крім того, користувач установлює дедлайн і параметри нагадувань. Після введення даних та натискання кнопки «Додати» система формує запит до сервера, де виконується перевірка авторизації, збереження файлів у директорії та запис нотатки до бази даних MongoDB. У разі успішного виконання операції користувач отримує сповіщення через тост-повідомлення, а список нотаток оновлюється. Діаграма також показує, як система реагує на можливі помилки – наприклад, на невалідний токен чи перевищення розміру файлу – та повертає користувачеві відповідне повідомлення.

Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики системи, які впливають на її продуктивність, зручність і безпеку:

1. Продуктивність:
- Час відгуку сервера на запит користувача не більше 2 секунд за нормальних умов.
- Обробка до 100 одночасних користувачів без значного зниження продуктивності.
2. Безпека:
- Використання JSON WEB Tokena для автентифікації та захисту маршрутів [7].
- Шифрування паролів у базі даних.
- Захист від атаки на авторизованого користувача і від шкідливого коду для системи через надійну обробку вхідних даних.

3. Масштабованість:

- Використання MongoDB для гнучкого зберігання структурованих даних нотаток, шаблонів і файлів [9].
- Можливість розширення функціоналу по типу інтеграцій або асистента з штучним інтелектом.

4. Доступність і адаптивність:

- Адаптивний інтерфейс, котрий сумісний з настільними та мобільними пристроями.
- Дотримання стандартів доступності WCAG 2.1 наприклад, використання Modal.setAppElement для коректної роботи модальних вікон.

5. Зручність використання:

- Інтуїтивно зрозумілий інтерфейс із мінімалістичним дизайном.
- Використання бібліотеки Tailwand CSS для швидкої розробки стилів [19].
- Відображення чітких повідомлень про помилки та успіх операції.

Технічне завдання

Назва проекту: Веб-застосунок для управління особистими завданнями

Мета проекту: Розробка веб-застосунку, який забезпечує зручне створення, редагування, організацію та планування особистих завдань із підтримкою шаблонів, прикріплення файлів, голосового введення та локалізації.

Основні компоненти системи:

1. Backend:

Технології: Node.js, Express.js, MongoDB, Mongoose, WebSocket, Multer для обробки файлів, JWT для автентифікації, node-schedule для планування.

Функціонал:

- REST API для управління користувачами, нотатками, шаблонами та файлами.
- Зберігання файлів у локальній папці uploads/.

Структура бази даних:

- Колекція Users: поля fullName, email, password, createdOn.
- Колекція Notes: поля title, content, tags, isPinned, userId, createdOn, deadline, reminder, attachments (масив із filename, path, mimetype, size).
- Колекція Templates: поля title, content, tags, userId, createdOn, deadline, attachments.

2. Frontend:

Технології: React, Vite, Tailwind CSS, react-router-dom, react-big-calendar, i18next, moment.js, lucide-react (іконки), react-modal.

Функціонал:

- Адаптивний інтерфейс із підтримкою світлої і темної тем, локалізація англійської та української мов.
- Сторінки: Home (список нотаток), Calendar календар із дедлайнами, Templates управління шаблонами, Login, SignUp.
- Компоненти: NoteCard, TemplateCard, AddEditNotes, AddEditTemplate, Navbar, SearchBar, PasswordInput, TagInput, ChangePasswordModal, Toast.
- Голосове введення через Web Speech API.
- Обробка файлів (додавання, видалення, перейменування, завантаження).

3. Інтеграція:

- Frontend взаємодіє з backend через Axios із перехопленням запитів для додавання токена автентифікації.
- Локалізація через i18next із синхронізацією форматів дати/часу через moment.js.

Обґрунтування вибору технологій:

React обрано через його компонентну архітектуру, яка забезпечує швидкість розробки інтерфейсів, повторне використання компонентів і підтримку

адаптивного дизайну. Це дозволяє створювати інтуїтивно зрозумілий і гнучкий інтерфейс для веб-застосунку.

Node.js і Express.js використовуються для створення швидкого та масштабованого серверного API завдяки асинхронній обробці запитів і широкій екосистемі бібліотек [13] [4].

MongoDB забезпечує гнучке зберігання неструктурованих даних, що ідеально підходить для нотаток, шаблонів і файлів із різними форматами та розмірами [9].

Vite обрано як інструмент для збирання клієнтської частини через його швидкість роботи в режимі розробки та оптимізацію [21].

Tailwind CSS дозволяє швидко створювати стилізовані інтерфейси з мінімалістичним дизайном, що відповідає вимогам зручності використання [19].

JWT і Multer використовуються для безпечної автентифікації та обробки файлів, відповідно, забезпечуючи захист даних [7] [11].

Інструментальні засоби:

- Серверна частина: Node.js, Express.js, MongoDB, Mongoose, WebSocket, Multer, JWT, node-schedule.
- Клієнтська частина: React, Vite, Tailwind CSS, react-router-dom, react-big-calendar, i18next, moment.js, Axios.
- Середовище розробки: VS Code.
- Тестування: Postman для API, браузерні інструменти розробника.

Вимоги до розгортання:

- Сервер: локальний сервер, localhost:8000 для API.
- База даних: MongoDB хмарна.
- Клієнт: Vite dev server (localhost:5173) для розробки, статичний хостинг для продакшену.

Обмеження:

- Максимальний розмір файлу 5 МБ.
- Максимальна кількість файлів на нотатку та шаблон 5.
- Підтримувані формати файлів: JPEG, PNG, PDF, DOC, DOCX.
- Підтримувані браузери: Chrome, Opera, Safari, Edge.

На основі аналізу предметної області сформовано функціональні та нефункціональні вимоги до веб-застосунку для управління особистими завданнями. Розроблено технічне завдання, яке включає використання сучасних технологій (React, Node.js, MongoDB), обґрунтованих їхньою ефективністю, гнучкістю та відповідністю вимогам продуктивності й адаптивності. Визначено структуру системи, інструментальні засоби та обмеження, що забезпечують створення зручного й безпечного рішення.

1.4. Аналіз методів реалізації

Щоб наш веб-застосунок для управління завданнями відповідав усім функціональним та нефункціональним вимогам, прописаним у технічному завданні, нам потрібно було ретельно підійти до вибору методів його реалізації. У цьому розділі здійснено аналіз підходів до реалізації основних компонентів системи, включаючи клієнтську та серверну частини, вибір бази даних, обробку даних, автентифікацію, взаємодію з файлами та локалізацію. Додатково проаналізовано вибір бази даних і серверного фреймворку шляхом порівняння альтернативних рішень. Аналіз проведено з урахуванням ефективності, масштабованості, безпеки та зручності використання, що є ключовими аспектами для створення сучасного програмного забезпечення.

Вибір технології для клієнтської частини

Для реалізації клієнтської частини веб-застосунку було обрано бібліотеку React у поєднанні з інструментом Vite. React забезпечує компонентну архітектуру, що дозволяє створювати модульні, повторно використовувані елементи інтерфейсу, такі як карти нотаток, форми введення та навігаційні панелі. Компонентний

підхід спрощує розробку, тестування та підтримку системи, оскільки кожен елемент інтерфейсу є ізольованим і може бути модифікований незалежно від інших. Використання Vite як інструменту збірки забезпечує швидке оновлення коду під час розробки завдяки технології Hot Module Replacement, що значно підвищує продуктивність розробників [21]. Крім того, Vite оптимізує кінцевий код для продакшену, зменшуючи розмір файлів і час завантаження сторінок.

Стилізація інтерфейсу реалізована за допомогою Tailwind CSS, що дозволяє створювати адаптивний і мінімалістичний дизайн без необхідності написання великої кількості власних стилів. Tailwind CSS забезпечує гнучкість у створенні адаптивних макетів, що відповідають вимогам до сумісності з різними пристроями (настільними ПК, планшетами, смартфонами) [19]. Наприклад, використання утиліт Tailwind CSS дозволило швидко реалізувати перемикання між світлою та темною темами, що зберігається в локальному сховищі браузера (localStorage), забезпечуючи персоналізацію користувацького досвіду.

Для маршрутизації між сторінками використано бібліотеку react-router-dom, яка забезпечує ефективне управління переходами між розділами застосунку, такими як "Головна", "Календар", "Шаблони" та сторінки автентифікації [17]. Інтеграція бібліотеки react-big-calendar дозволила реалізувати візуалізацію дедлайнів нотаток у форматі календаря, що полегшує планування для користувачів [16]. Локалізація інтерфейсу, яка підтримує українську та англійську мови, реалізована через бібліотеку i18next у поєднанні з moment.js для форматування дат і часу, що забезпечує відповідність [6].

Інші аспекти реалізації серверної частини

Автентифікація користувачів реалізована за допомогою JSON Web Tokens, які генеруються під час реєстрації або входу в систему [7]. Токени передаються в заголовках HTTP-запитів для захисту маршрутів, що забезпечує безпечний доступ до даних. Паролі користувачів хешуються за допомогою бібліотеки bcrypt, що гарантує їх захист від зловмисного доступу [2].

Обробка файлів здійснюється через бібліотеку Multer, яка дозволяє завантажувати файли у форматах JPEG, PNG, PDF, DOC, DOCX із обмеженням розміру до 5 МБ і кількості файлів до 5 на одну нотатку чи шаблон [11]. Файли зберігаються в директорії uploads/ на сервері, а їх метадані записуються в базу даних MongoDB [9]. Такий підхід забезпечує централізоване управління файлами та їх інтеграцію з нотатками.

Інтеграція клієнтської та серверної частин

Взаємодія між клієнтською та серверною частинами здійснюється через REST API з використанням бібліотеки Axios на фронтенді. Axios дозволяє надсилати асинхронні HTTP-запити і обробляти відповіді з урахуванням можливих помилок [1]. Наприклад, створення нової нотатки реалізовано через POST-запит до маршруту /add-note, який включає дані нотатки та токен авторизації в заголовку. Сервер перевіряє токен, валідує дані та зберігає нотатку в базі даних, повертаючи клієнту статус операції.

Для обробки помилок на сервері передбачено повернення відповідних кодів стану HTTP, що дозволяє клієнтській частині інформувати користувача через тост-повідомлення.

Аналіз ефективності методів реалізації

Ми переконалися, що обрані нами методи реалізації задовольняють усі вимоги до продуктивності, безпеки та зручності. Наприклад, використання React та Vite дозволяє сторінкам завантажуватися швидко, а інтерфейс реагує на дії користувача локально в межах 100 мс, що робить розробку швидкою та комфортною. MongoDB забезпечує швидкий доступ до даних завдяки індексації та нереляційній структурі, а Multer ефективно обробляє файли, уникаючи перевантаження сервера.

Безпека системи підтверджується використанням JWT для автентифікації, bcrypt для захисту паролів і валідацією вхідних даних на сервері, що мінімізує ризик атак, таких як SQL-ін’єкції чи XSS. Локалізація через i18next і адаптивність через

Tailwind CSS забезпечують доступність застосунку для широкої аудиторії, включаючи користувачів із різними пристроями та мовними вподобаннями.

Проаналізувавши обрані технології та підходи ми бачимо, що вони дозволяють ефективно реалізувати всі поставлені функціональні та нефункціональні вимоги. Завдяки компонентній архітектурі клієнта, асинхронній обробці серверних запитів, гнучкості MongoDB та безпечній автентифікації, ми отримали надійний фундамент для нашої системи управління завданнями. Порівняння баз даних і серверних фреймворків підтверджує доцільність вибору MongoDB і Express.js завдяки їхній гнучкості, продуктивності та простоті інтеграції.

2. Проєктування інструментальних засобів створення та використання інформаційних систем і технологій

2.1. Визначення проектних рішень

Щоб успішно реалізувати веб-застосунок для управління особистими завданнями та забезпечити виконання всіх вимог, ми прийняли низку ключових проєктних рішень.

Архітектура системи

Обрано клієнт-серверну архітектуру з наступними характеристиками:

Фронтенд:

- Реалізовано за допомогою бібліотеки React, що забезпечує компонентну архітектуру та високу швидкість розробки
- Інструмент збірки – Vite, що дозволяє швидко перезапускати застосунок під час розробки.
- Адаптивний інтерфейс з використанням Tailwind CSS для стилізації.

Бекенд:

- Реалізовано на платформі Node.js з використанням фреймворку Express.js для створення REST API.
- База даних – MongoDB, яка забезпечує зберігання структурованих даних про користувачів, нотатки та шаблони.
- Використання Multer для завантаження файлів і бібліотеки JWT для автентифікації.

Технології

Для реалізації веб-застосунку управління особистими завданнями було обрано низку сучасних технологій, які забезпечують ефективність розробки, масштабованість системи, а також її продуктивність і зручність для користувачів представлено в (табл. 2.1). Застосовані технології покривають усі аспекти розробки: від створення адаптивного інтерфейсу до обробки даних і реальних час сповіщень.

Вибір кожної технології ґрунтувався на її відповідності функціональним і нефункціональним вимогам проєкту. Фронтенд рішення забезпечують зручність використання і високу швидкість роботи застосунку, тоді як бекенд побудований для забезпечення безпечної, надійної обробки даних та їх зберігання. Інтеграція цих компонентів дозволяє створити потужний і гнучкий інструмент для планування завдань.

Таблиця 2.1 – технології із зазначенням їх переваг

Компонент системи	**Використана технологія**	**Переваги**
Фронтенд	React	Компонентна архітектура, швидкість розробки, адаптивність.
Фронтенд	Vite	Швидка збірка, оптимізація для продакшну.
Стилізація	Tailwind CSS	Проста адаптивна верстка, мінімалістичний дизайн.
Маршрутизація	React Router	Просте управління сторінками.
Бекенд	Node.js + Express.js	Асинхронна обробка запитів, масштабованість.
Зберігання даних	MongoDB	Гнучкість у роботі зі структурованими даними.
Збереження файлів	Multer	Зручна обробка файлів різних форматів.
Авторизація	JWT	Захищена автентифікація, простота інтеграції.
Локалізація	i18next	Підтримка кількох мов для інтерфейсу.
Планування задач	Node-schedule	Гнучке планування та нагадування.

Фронтенд:

- React Router для маршрутизації сторінок.
- React Big Calendar для відображення подій у форматі календаря.
- i18next для локалізації підтримка української та англійської мов.

Бекенд:

- Mongoose для взаємодії з MongoDB.
- Node-schedule для планування завдань.

Визначення функціональних компонентів

Діаграма компонентів (Рис. 2.1) ілюструє основні модулі веб-застосунку для управління особистими завданнями та їх взаємодію. Вона поділена на дві основні частини: клієнтську (Frontend) та серверну (Backend), а також зовнішні сервіси та базу даних. Нижче наведено детальний опис кожного компонента:

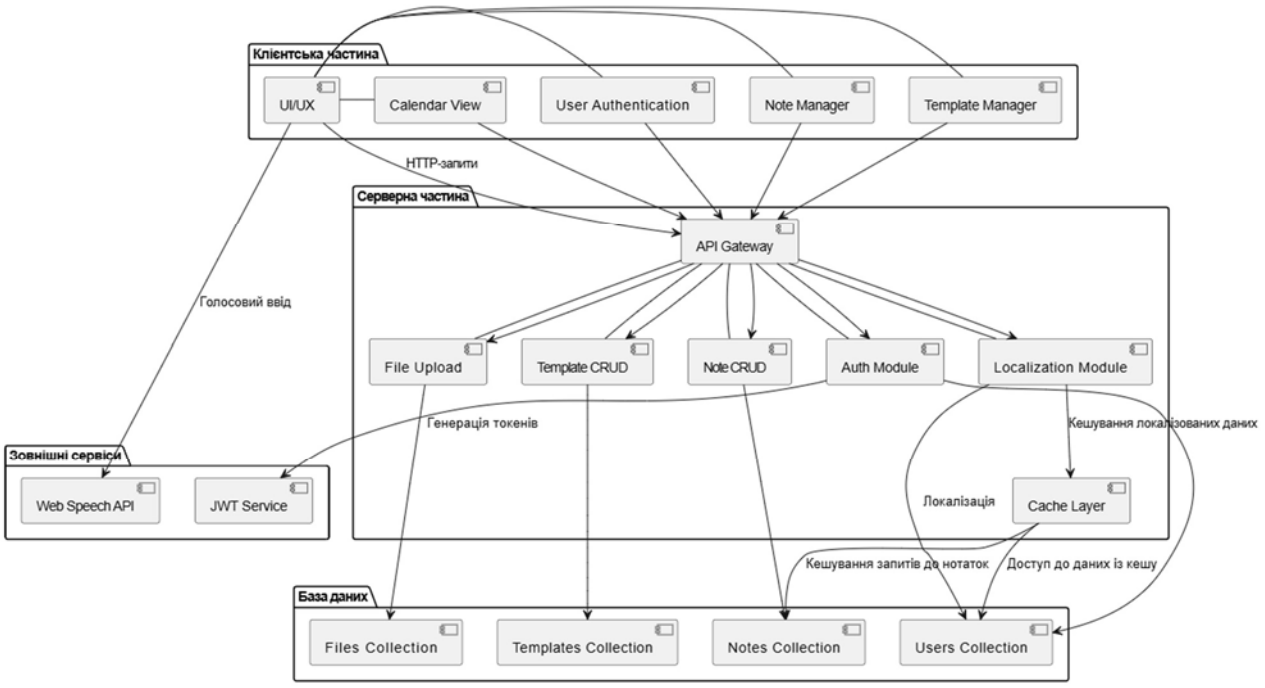

Рисунок 2.1 – Діаграма компонентів

Клієнтська частина (Frontend)

Клієнтська частина реалізована за допомогою бібліотеки React і містить такі модулі:

- User Authentication: Відповідає за реєстрацію, авторизацію та управління обліковими записами користувачів.
- Note Manager: Забезпечує створення, редагування, видалення та перегляд нотаток, включаючи додавання тегів, дедлайнів та файлів.
- Template Manager: Дозволяє працювати з шаблонами для швидкого створення нотаток.
- Calendar View: Відображає нотатки у вигляді календаря з можливістю перегляду за датами.

- UMUX (User Experience Module): Відповідає за адаптивний інтерфейс, теми (світлу/темну) та локалізацію (українська/англійська мови).

Серверна частина (Backend)

Серверна частина побудована на Node.js з використанням Express.js і включає такі модулі:

- API Gateway: Обробляє всі вхідні запити від клієнта та перенаправляє їх до відповідних модулів.
- Auth Module: Реалізує автентифікацію через JSON Web Token, шифрування паролів та захист маршрутів.
- Note CRUD: Забезпечує операції з нотатками (створення, читання, оновлення, видалення).
- Template CRUD: Аналогічний модуль для роботи з шаблонами.
- File Upload: Обробляє завантаження, зберігання та видалення файлів за допомогою Multer.
- Localization Module: Надає підтримку локалізації для серверних повідомлень.

Зовнішні сервіси

- Web Speech API: Використовується для голосового введення тексту в нотатки [22].
- JWT Service: Забезпечує генерацію та валідацію токенів для безпечної автентифікації [7].

База даних

Система використовує MongoDB для зберігання даних, які організовані у такі колекції:

- Users Collection: Зберігає дані користувачів.
- Notes Collection: Містить нотатки, включаючи заголовок, вміст, теги, дедлайни та прикріплені файли.

- Templates Collection: Зберігає шаблони для нотаток.
- Files Collection: Відповідає за метадані завантажених файлів (шлях, розмір, тип).

Взаємодія компонентів

- Клієнтська частина відправляє запити до серверної через REST API.
- Серверна частина обробляє запити, взаємодіє з базою даних та зовнішніми сервісами.
- Дані зберігаються та оновлюються в MongoDB, після чого сервер повертає відповідь клієнту.
- Клієнт відображає оновлені дані.

Діаграма компонентів наочно демонструє архітектуру веб-застосунку, розподіл функціональності між клієнтською та серверною частинами, а також інтеграцію з зовнішніми сервісами. Вона підкреслює модульність системи, що спрощує розробку, тестування та масштабування. Окремі компоненти (наприклад, Auth Module або File Upload) можуть бути легко модифіковані або замінені без впливу на інші частини системи. Використання сучасних технологій (React, Node.js, MongoDB) забезпечує високу продуктивність, безпеку та зручність для користувачів.

Модуль автентифікації:

- Реєстрація, авторизація та захист маршрутів за допомогою JWT.
- Захищене зберігання паролів із використанням bcrypt.

Модуль управління нотатками:

- Створення, редагування та видалення завдань.
- Додавання дедлайнів, тегів та прикріплення файлів.

Модуль шаблонів:

- Створення та використання заготовок для нотаток.

Інтерфейс користувача:

- Адаптивний дизайн для зручного доступу з мобільних і настільних пристроїв.
- Світла та темна теми.

Взаємодія компонентів

- Фронтенд звертається до бекенду через REST API для отримання та обробки даних.
- Бекенд забезпечує перевірку даних, взаємодію з базою даних та обробку запитів.
- Нотатки з дедлайнами інтегруються з календарем для візуалізації.

Особливості розробки

- Забезпечено обмеження на розмір файлів до 5 МБ і кількість прикріплень до 5 файлів на завдання.
- Використано захищений обмін даними між клієнтом і сервером через HTTPS.

2.2.Розробка логічної та фізичної структури

Розробка логічної та фізичної структури веб-застосунку для управління особистими завданнями є ключовим етапом проєктування, що забезпечує реалізацію вимог технічного завдання, сумісність компонентів і ефективну взаємодію між підсистемами. Логічна структура визначає абстрактне представлення системи, її основні модулі, інформаційні ресурси та їх взаємозв’язки, тоді як фізична структура описує апаратне та програмне забезпечення, необхідне для реалізації системи.

Логічна структура системи

Логічна структура системи відображає функціональні компоненти, їх взаємозв’язки та інформаційні потоки. Вона побудована на основі клієнт-серверної архітектури, де клієнтська частина відповідає за взаємодію з користувачем, а серверна — за обробку даних, автентифікацію та зберігання

інформації. Основні модулі системи розділені на клієнтську та серверну частини, а також базу даних і зовнішні сервіси.

Основні модулі логічної структури:

1. Клієнтська частина:

- Модуль автентифікації користувача: Забезпечує реєстрацію, авторизацію, зміну пароля та вихід із системи.
- Модуль управління нотатками: Дозволяє створювати, редагувати, видаляти та переглядати нотатки, а також додавати теги, дедлайни та прикріплені файли.
- Модуль управління шаблонами: Надає можливість створювати, редагувати та застосовувати шаблони для нотаток.
- Модуль календаря: Відображає нотатки з дедлайнами у форматі календаря.
- Модуль інтерфейсу користувача (UI/UX): Забезпечує адаптивний інтерфейс, підтримку світлої/темної тем і локалізацію (українська/англійська мови).
- Модуль голосового введення: Використовує Web Speech API для введення тексту голосом.

2. Серверна частина:

- API Gateway: Обробляє вхідні запити від клієнта та перенаправляє їх до відповідних модулів.
- Модуль автентифікації: Реалізує перевірку JWT-токенів, шифрування паролів і захист маршрутів.
- Модуль CRUD для нотаток і шаблонів: Забезпечує операції створення, читання, оновлення та видалення даних.
- Модуль обробки файлів: Використовує Multer для завантаження, зберігання та видалення файлів.
- Модуль локалізації: Надає серверні повідомлення з підтримкою кількох мов.

3. База даних:

Нормалізація даних

База даних розроблена згідно з принципами нормалізації до третьої нормальної форми, що забезпечує мінімізацію дублювання даних і збереження їхньої цілісності.

Структура бази даних

- Таблиця Users: Зберігає дані користувачів. Первинний ключ: id. Унікальні поля: email.
- Таблиця Notes: Містить нотатки. Первинний ключ: id. Зв’язок "один до багатьох" із таблицею Users через userId.
- Таблиця Templates: Зберігає шаблони нотаток. Первинний ключ: id. Зв’язок "один до багатьох" із таблицею Users через userId.
- Таблиця Tags: Зберігає теги для категоризації нотаток і шаблонів. Первинний ключ: id. Унікальне поле: name.
- Таблиця Files: Зберігає метадані файлів. Первинний ключ: id.
- Таблиця Note_Tags: Зв’язкова таблиця для нотаток і тегів. Первинний ключ: комбінація noteId і tagId.
- Таблиця Note_Files: Зв’язкова таблиця для нотаток і файлів. Первинний ключ: комбінація noteId і fileId.
- Таблиця Templates_Tags: Зв’язкова таблиця для шаблонів і тегів. Первинний ключ: комбінація templateId і tagId.
- Таблиця Templates_Files: Зв’язкова таблиця для шаблонів і файлів. Первинний ключ: комбінація templateId і fileId.

Діаграма на (Рис. 2.2) демонструє зв’язки між основними таблицями системи:

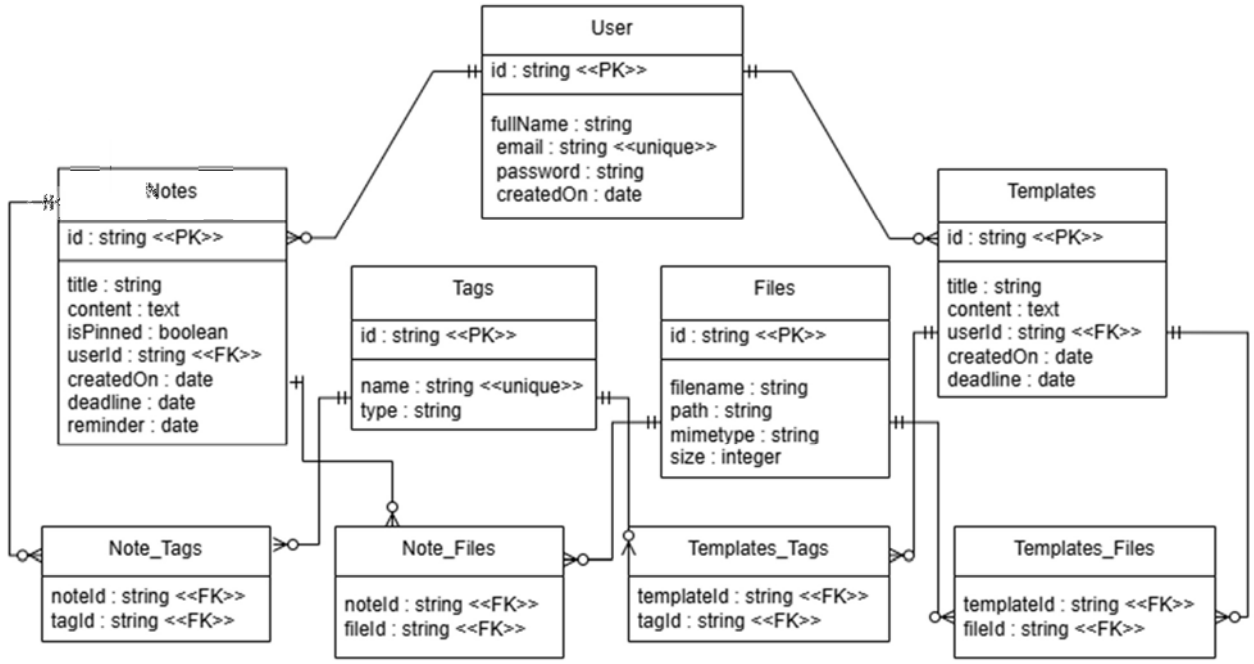

Рисунок 2.2 – Діаграма сутність-зв’язок

Пояснення зав’язків і компонентів

- Користувачі (Users): Кожен користувач може створювати багато нотаток і шаблонів. Зв’язок реалізований через поле userId у таблицях Notes і Templates.
- Нотатки (Notes): Нотатка може бути позначена кількома тегами та мати прикріплені файли. Ці зв’язки реалізовані через таблиці Note_Tags і Note_Files.
- Шаблони (Templates): Шаблон може бути пов’язаний із тегами та файлами аналогічно до нотаток.
- Теги (Tags): Використовуються для категоризації нотаток і шаблонів. Один тег може бути пов’язаний із багатьма нотатками та шаблонами.
- Файли (Files): Містять метадані прикріплень до нотаток і шаблонів. Один файл може бути прикріплений до багатьох нотаток або шаблонів.

4. Зовнішні сервіси:

- Web Speech API: Використовується для голосового введення.
- JWT Service: Генерація та перевірка токенів для автентифікації.

Інформаційні потоки:

- Клієнтська частина відправляє HTTP-запити (GET, POST, PUT, DELETE) до API Gateway через Axios.
- Сервер обробляє запити, взаємодіє з MongoDB для зберігання/отримання даних і повертає відповідь клієнту.
- Дані про нотатки, шаблони та файли синхронізуються між клієнтом і сервером через REST API.
- Локалізація інтерфейсу та повідомлень забезпечується через i18next і moment.js.

Фізична структура системи

Фізична структура показує яке апаратне та програмне забезпечення, необхідне для реалізації веб-застосунку, а також розгортання системи на сервері та клієнтських пристроях.

Апаратне забезпечення:

Сервер:

- Процесор: 2 ядра, 2.4 ГГц або вище.
- Оперативна пам’ять: 4 ГБ для локального сервера, 8 ГБ для продакшену.
- Дисковий простір: 20 ГБ для зберігання бази даних і файлів (з урахуванням папки uploads/).
- Операційна система: Windows Server.

Клієнтські пристрої:

- Сумісність з настільними ПК, ноутбуками, планшетами та смартфонами.
- Підтримувані браузери: Chrome, Opera, Safari, Edge.
- Мінімальна роздільна здатність екрана 320*480 пікселів для адаптивного дизайну.

Програмне забезпечення

- Серверна частина:
- Node.js для виконання JavaScript.
- Express.js для створення REST API.
- MongoDB для зберігання даних.
- Multer для обробки файлів.
- JWT для автентифікації.

Клієнтська частина:

- React для створення компонентного інтерфейсу.
- Vite для швидкої збірки та розробки.
- Tailwind CSS для стилізації.
- Бібліотеки: react-router-dom, react-big-calendar, i18next, moment.js, Axios, lucide-react, react-modal.

Середовище розробки:

- Редактор коду Visual Studio Coode
- Інструменти тестування: Postman для API, браузерні інструменти розробника.

Розгортання:

- Локальний сервер: API на localhost:8000, клієнт на localhost:5173.
- Хмарна база даних: MongoDB Atlas для продакшену.
- Статичний хостинг для клієнтської частини.

Розроблена логічна структура системи чітко окреслює функціональні модулі, такі як автентифікація, управління нотатками, шаблонами та календарем, а також їх інформаційні потоки, що забезпечує відповідність функціональним вимогам. Фізична структура, побудована на сучасних технологіях, таких як React, Node.js і MongoDB, гарантує продуктивність, безпеку та адаптивність системи.

Використання діаграми сутність-звязок, дозволяє наочно продемонструвати зв’язки між компонентами. Ці рішення створюють надійну основу для подальшої декомпозиції системи на модулі та реалізації людино-машинного інтерфейсу.

2.3. Способи взаємодії між підсистемами

Для надійної, безпечної та ефективної взаємодії між фронтендом та бекендом у нашому веб-застосунку ми використовуємо кілька ключових механізмів.

HTTP-запити

Фронтенд взаємодіє з бекендом через REST API, використовуючи звичайні HTTP-методи:

- GET — для отримання даних.
- POST — для реєстрація користувача, додавання нотаток та шаблонів.
- PUT — для оновлення існуючих нотаток та шаблонів.
- DELETE — для видалення нотаток та шаблонів.

HTTP-запит для створення нові нотатки наведено в (Додаток А лістинг А 1).

Цей код демонструє використання бібліотеки axios для створення POST-запиту. У ньому передаються дані нової нотатки, а також токен авторизації через заголовок запиту. Такий підхід забезпечує безпеку, обмежуючи доступ лише авторизованим користувачам.

Маршрут для обробки цього запиту наведено в (Додаток А лістинг А 2).

У цьому маршруті обробляється запит, перевіряються обов’язкові поля, а також додається нова нотатка до бази даних MongoDB. Використання проміжного програмного забезпечення authenticateToken гарантує, що доступ до маршруту мають лише авторизовані користувачі.

Реєстрація та авторизація

Бекенд використовує токени аутентифікації (JWT), які генеруються під час реєстрації або входу користувача. Ці токени передаються у заголовках запитів (Authorization) для перевірки доступу до захищених ресурсів.

Лістинг генерації токена під час входу користувача показано в (Додаток А лістинг А 3).

Цей маршрут перевіряє облікові дані користувача і генерує токен за допомогою бібліотеки jsonwebtoken. Токен використовується для підтвердження особи користувача під час доступу до захищених ресурсів.

Фронтенд зберігає токени в локальному сховищі (localStorage) чи cookies для повторного використання

WebSocket

Для інтерактивних функцій, таких як надсилання нагадувань у реальному часі, використовується WebSocket-з'єднання. Наприклад:

Користувач підключається до WebSocket-сервера через унікальний токен.

Сервер надсилає нагадування чи інші повідомлення клієнт, лістинг підключення WebSocket в (Додаток А лістинг А 4 і А 5).

Цей код ілюструє підключення до WebSocket сервера і обробку повідомлень типу "reminder". Використання токена забезпечує ідентифікацію користувача під час встановлення з’єднання.

Сервер перевіряє токен під час встановлення з’єднання. Це гарантує, що тільки авторизовані користувачі можуть отримувати нагадування або взаємодіяти через WebSocket.

Завантаження файлів

Використовується бібліотека Multer для обробки завантажених файлів на бекенді. Фронтенд передає файли у формі багаточастинного запиту (формат multipart/form-data).

Оброблені файли зберігаються у визначеній директорії на сервері, а інформація про них записується в базу даних, лістинг обробки завантаження файлу в (Додаток А лістинг А 6 і А 7).

Форма даних використовується для передачі файлу, а токен авторизації забезпечує безпечне завантаження. Сервер обробляє запит і зберігає файл у визначеній директорії.

На бекенді маршрут для обробки завантаження

Бекенд використовує бібліотеку Multer для обробки файлів, які зберігаються з унікальними іменами у визначеній директорії. Це дозволяє уникнути конфліктів і забезпечує ефективну роботу із завантаженнями.

База даних

Бекенд працює з базою даних MongoDB, де зберігаються всі необхідні дані: користувачі, нотатки, налаштування.

Для кожної взаємодії з базою використовується модель доступу, що реалізована через ORM бібліотеку Mongoose.

Обробка помилок

На рівні бекенду передбачена обробка помилок з коректними відповідями, які інформують фронтенд про проблему (наприклад, 400 — некоректний запит, 401 — неавторизований доступ).

2.4. Декомпозиція системи на модулі

Ми розділили веб-застосунок на модулі, щоб забезпечити його гнучкість, можливість повторного використання коду та спростити подальший супровід. Було обрали архітектуру, що складається з бекенд-компонента, побудованого на основі технологій Node.js, Express та MongoDB, та фронтенд-частини, реалізованої за допомогою React із Vite. У цьому підрозділі наводиться детальна декомпозиція системи на основні модулі з описом їхньої функціональності,

обґрунтуванням вибору, а також включається ER-діаграма, яка відображає структуру бази даних. Такий підхід дозволяє реалізувати вимоги технічного завдання, забезпечити сумісність та ефективну взаємодію між компонентами.

Обґрунтування декомпозиції

Ми розділили систему на модулі, виходячи з ключових вимог: безпечна автентифікація, зручне управління нотатками й шаблонами, робота з файлами та система сповіщень. Такий модульний підхід ми обрали тому, що він дає гнучкість, дозволяє легко масштабувати систему та вдосконалювати окремі компоненти незалежно один від одного. Кожен модуль розроблено з урахуванням його ролі в загальній архітектурі, що полегшує тестування.

Основні модулі системи

Бекенд (Серверна частина)

Ця підсистема відповідає за обробку вхідних запитів, управління даними, автентифікацію користувачів, зберігання файлів та відправлення сповіщень. Її реалізація базується на сучасних технологіях, що забезпечують високу продуктивність і надійність.

Модуль автентифікації та авторизації

- Функціональне призначення: Здійснює реєстрацію нових користувачів, авторизацію, вихід із системи, зміну пароля та перевірку JWT-токенів для забезпечення безпечного доступу.
- Файли: backend/index.js (включає маршрути /create-account, /login, /logout, /change-password, /get-user), backend/models/user.model.js (опис схеми користувача), backend/utilities.js (логіка перевірки токенів).
- Залежності: jsonwebtoken (для роботи з токенами), bcrypt (для хешування паролів).

- Обґрунтування: Виділення цього модуля зумовлено необхідністю захисту даних та персоналізації доступу для кожного користувача, що є критичним для конфіденційності.

Модуль управління нотатками

Функціональне призначення: Забезпечує створення, редагування, видалення, пошук, закріплення нотаток, а також управління дедлайнами та нагадуваннями.

Підмодулі:

- Створення нотатки (обробка маршруту /add-note).
- Управління файлами (інтеграція з завантаженими документами).
- Нагадування (координація з дедлайнами).

Файли: backend/index.js (маршрути /add-note, /edit-note, /delete-note, /get-all-notes, /search-notes, /update-note-pinned), backend/models/note.model.js (схема нотаток).

Залежності: mongoose (для роботи з MongoDB), multer (для обробки завантажень).

Обґрунтування: Поділ на підмодулі підвищує масштабованість і дозволяє оптимізувати кожну операцію окремо.

Модуль управління шаблонами

- Функціональне призначення: Дозволяє створювати, редагувати, видаляти та отримувати шаблони для швидкого створення нотаток.
- Файли: backend/index.js (маршрути /create-template, /edit-template, /delete-template, /get-all-templates), backend/models/template.model.js (схема шаблонів).
- Залежності: mongoose, multer.
- Обґрунтування: Цей модуль спрощує повторне використання даних, що є важливим для підвищення продуктивності користувачів.

Модуль управління файлами

- Функціонале призначення: Здійснює завантаження, видалення, перейменування та доступ до файлів (зображення, PDF, Word-документи).
- Файли: backend/index.js (маршрути /get-file, логіка multer для завантаження), директорія Uploads для зберігання.
- Залежності: multer (обробка мультимедіа), fs (файлова система).
- Обґрунтування: Виділення цього модуля забезпечує централізоване управління медіафайлами, що полегшує їх інтеграцію з нотатками.

Модуль бази даних

- Функціонале призначення: Забезпечує підключення до MongoDB, управління схемами та виконання операцій із даними.
- Файли: backend/index.js (підключення через mongoose), backend/models/*.js (схеми користувачів, нотаток, шаблонів).
- Залежності: mongoose.
- Обґрунтування: Централізує доступ до даних, що спрощує синхронізацію між модулями.

Фронтенд (Клієнтська частина)

Ця підсистема відповідає за створення інтуїтивного інтерфейсу користувача, взаємодію з бекендом, відображення даних та підтримку локалізації. Її реалізація базується на сучасних веб-технологіях для забезпечення високої продуктивності та адаптивності.

Модуль навігації та маршрутизації

- Функціонале призначення: Забезпечує навігацію між сторінками (Головна, Календар, Шаблони, Вхід, Реєстрація).
- Файли: frontend/notes-app/src/App.jsx (налаштування маршрутів), frontend/notes-app/src/components/Navbar/Navbar.jsx (компонент навігаційної панелі).

- Залежності: react-router-dom.
- Обґрунтування: Цей модуль є основою для зручного доступу до всіх функцій системи.

Модуль автентифікації

- Функціонале призначення: Надає інтерфейс для входу, реєстрації, зміни пароля та перегляду профілю користувача.
- Файли: frontend/notes-app/src/pages/Login/Login.jsx, frontend/notes-app/src/pages/SignUp/SignUp.jsx, frontend/notes-app/src/components/Modals/ChangePasswordModal.jsx, frontend/notes-app/src/components/Cards/ProfileInfo.jsx.
- Залежності: axios (для HTTP-запитів), react-i18next (для локалізації).
- Обґрунтування: Забезпечує безперервну інтеграцію з бекенд-системою автентифікації.

Модуль управління нотатками

- Функціонале призначення: Дозволяє створювати, редагувати, видаляти, переглядати, шукати нотатки, а також додавати теги та файли.
- Файли: frontend/notes-app/src/pages/Home/Home.jsx, frontend/notes-app/src/pages/Home/AddEditNotes.jsx, frontend/notes-app/src/components/Cards/NoteCard.jsx, frontend/notes-app/src/components/Modals/ViewNoteModal.jsx, frontend/notes-app/src/components/Input/TagInput.jsx.
- Залежності: axios, react-modal, lucide-react (іконки).
- Обґрунтування: Цей модуль є центральним для користувацького досвіду, тому його деталізація підвищує зручність.

Модуль управління шаблонами

- Функціонале призначення: Забезпечує створення, редагування, видалення та перегляд шаблонів нотаток.

- Файли: frontend/notes-app/src/pages/Template/Template.jsx, frontend/notes-app/src/pages/Template/AddEditTemplate.jsx, frontend/notes-app/src/components/Cards/TemplateCard.jsx.
- Залежності: axios, react-modal.
- Обґрунтування: Підтримує ефективність роботи, дозволяючи користувачам повторно використовувати шаблони.

Модуль календаря

- Функціонале призначення: Відображає нотатки в календарі та дозволяє обирати дату для створення нових записів.
- Файли: frontend/notes-app/src/pages/Calendar/Calendar.jsx.
- Залежності: react-big-calendar, moment.
- Обґрунтування: Додає візуальну організацію даних у часі.

Модуль локалізації

- Функціонале призначення: Підтримує інтерфейс англійською та українською мовами.
- Файли: frontend/notes-app/src/i18n.js, ресурси перекладів.
- Залежності: i18next, react-i18next.
- Обґрунтування: Розширює аудиторію завдяки мультимовності.

Модуль теми

- Функціонале призначення: Дозволяє перемикатися між темами (світлою/темною) та зберігає вибір у localStorage.
- Файли: frontend/notes-app/src/components/Navbar/Navbar.jsx, frontend/notes-app/src/components/ThemeToggle/ThemeToggle.jsx, frontend/notes-app/src/index.css.
- Залежності: tailwindcss.
- Обґрунтування: Покращує доступність і комфорт використання.

Модуль стилів та UI

- Функціонале призначення: Забезпечує стилізацію компонентів та адаптивний дизайн.
- Файли: frontend/notes-app/src/index.css, frontend/notes-app/tailwind.config.js.
- Залежності: tailwindcss, autoprefixer.
- Обґрунтування: Гнучкість Tailwind CSS дозволяє швидко адаптувати дизайн.

Взаємодія модулів

Модулі системи взаємодіють через стандартизовані протоколи. Зокрема, фронтенд звертається до бекенду через REST API, використовуючи HTTP-запити, такі як POST /add-note із JSON-структурою { "title": "Нова нотатка", "content": "Текст" }, для створення нотаток. Модуль сповіщень реалізує реальний час через WebSocket, отримуючи дані про дедлайни з модуля нотаток і надсилаючи нагадування. Модуль автентифікації передає JWT-токен у заголовках запитів, забезпечуючи захист маршрутів. Локалізація та теми реалізуються виключно на фронтенді, що зменшує навантаження на серверну частину.

ER-діаграма структури бази даних

Для ілюстрації логічної структури бази даних, що підтримує функціонування системи, наведено ER-діаграму (Рис. 2.3). Вона відображає сутності USER, NOTE та TEMPLATE, їхні атрибути та зв’язки.

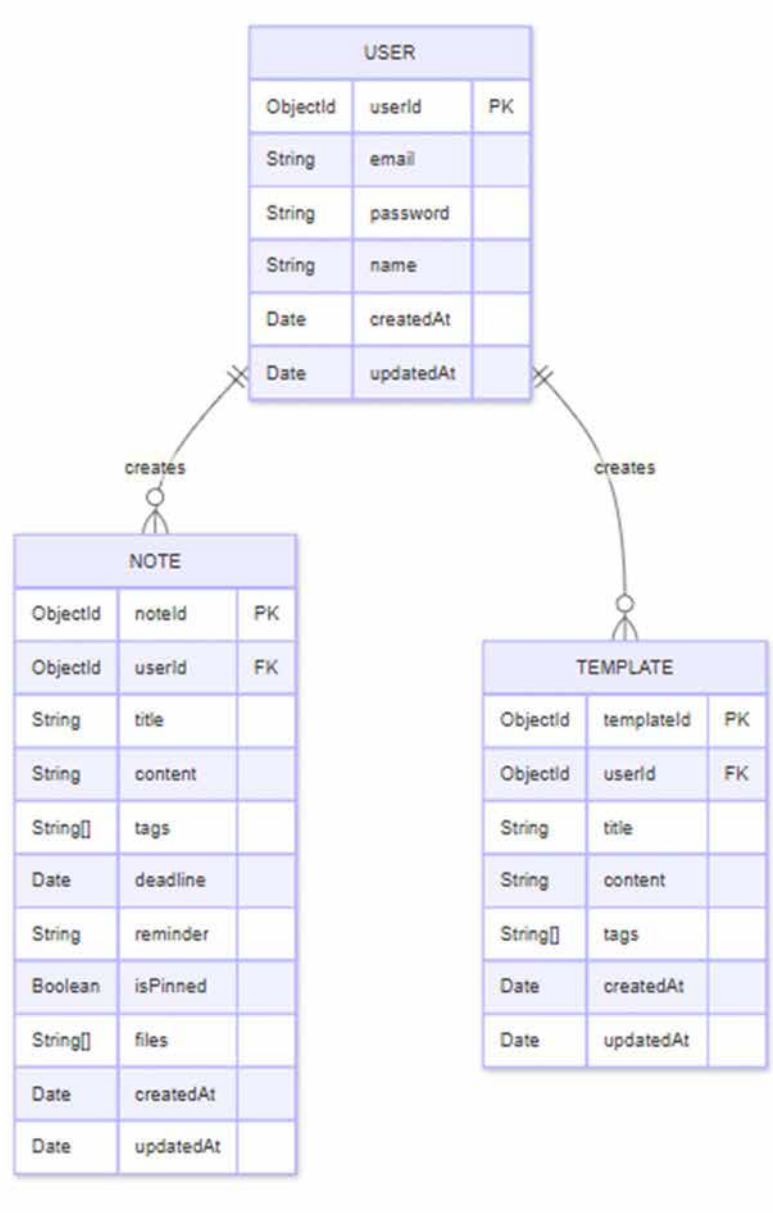

Рисунок 2.3 – ER-діаграма бази даних

Опис: Сутність USER (з атрибутами userId (PK), email, password, name, createdAt, updatedAt) пов’язана з NOTE (з noteId (PK), userId (FK), title, content, tags (String[]), deadline, reminder, isPinned, files (String[]), createdAt, updatedAt) та TEMPLATE (з templateId (PK), userId (FK), title, content, tags (String[]), createdAt, updatedAt) через зв’язок "один до багатьох". Ця модель інтегрується з модулями, наприклад, поле userId пов’язує модуль автентифікації з модулем нотаток, а files підтримує функціонал модуля управління файлами.

Декомпозиція системи на модулі забезпечує високу модульність, повторне використання коду та масштабованість. Обґрунтування вибору модулів базується на функціональних вимогах, а ER-діаграма відображає логічну структуру бази даних, що підтримує всі компоненти.

2.5. Розробка людино машинного інтерфейсу

Створюючи веб-застосунок для управління особистими завданнями ми приділили особливу увагу розробці людино-машинного інтерфейсу. Нашою метою було зробити взаємодію користувачів із системою максимально ефективною та зручною. Ми розробили інтерфейс, керуючись принципами ергономіки, інтуїтивності та адаптивності, щоб він був зручним для широкого кола користувачів, незалежно від їхнього рівня технічної підготовки. Далі ми детальніше розглянемо ключові елементи інтерфейсу, як вони взаємодіють, та на яких принципах побудована їх реалізація. Візуалізація основних екранних форм допоможе краще зрозуміти структуру ЛМІ. В основі нашого підходу лежать сучасні методології UX/UI дизайну.

Елементи людино-машинного інтерфейсу

Визначено ключові елементи інтерфейсу, які реалізують основні функції системи. До них належать:

- Навігаційна панель: Забезпечує доступ до основних розділів застосунку, таких як "Головна", "Календар", "Шаблони", "Вихід" та "Зміна паролю". Реалізовано через компонент Navbar.jsx, інтегрований у файл App.jsx для маршрутизації, що дозволяє користувачам легко переключатися між функціональними модулями.
- Карти нотаток: Відображають інформацію про нотатки у вигляді інтерактивних карток (компонент NoteCard.jsx), які підтримують перегляд, редагування та закріплення записів, сприяючи візуальній організації даних.
- Модальні вікна: Використовуються для детального перегляду нотаток (компонент ViewNoteModal.jsx) та створення нових записів (компонент,

пов’язаний із AddEditNotes.jsx), забезпечуючи зручний доступ до додаткових функцій без перезавантаження сторінки.

- Форми введення: Включають поля для створення та редагування нотаток (компонент AddEditNotes.jsx) та шаблонів (компонент AddEditTemplate.jsx), а також інструмент для додавання тегів (TagInput.jsx), що полегшує введення даних.
- Тост-повідомлення: Надають зворотний зв’язок користувачу про успішні або невдалі операції (компонент Toast.jsx), підвищуючи інтерактивність і зручність використання.
- Календар: Інтегровано через компонент Calendar.jsx, який відображає нотатки в часовому контексті та дозволяє обирати дати для планування, сприяючи ефективному управлінню часом.
- Кнопки дій: Інтерактивні елементи, такі як кнопка "Додати нотатку" на головній сторінці та кнопки "Зберегти" та "Очистити" у модальному вікні, забезпечують швидкий доступ до основних операцій.

Взаємозв’язок елементів інтерфейсу

Елементи інтерфейсу пов’язані між собою для забезпечення логічної послідовності дій користувача. Навігаційна панель слугує центральним елементом, який спрямовує користувача до відповідних сторінок, де компоненти, такі як карти нотаток і календар, відображають дані, отримані з бекенду через REST API. Модальні вікна активуються з карток нотаток або кнопок дій (наприклад, "Додати нотатку"), забезпечуючи доступ до форм введення без зміни основного екрану. Тост-повідомлення інтегровано з усіма інтерактивними елементами для миттєвого інформування про результати дій, що підвищує зручність і зрозумілість інтерфейсу.

Принципи реалізації

Було використано сучасні методи дизайну UI/UX, зокрема:

- Адаптивність: Інтерфейс адаптовано до різних розмірів екранів завдяки використанню Tailwind CSS, конфігурація якого визначена у файлі tailwind.config.js, що забезпечує коректне відображення на мобільних і настільних пристроях.
- Локалізація: Підтримка англійської та української мов реалізовано через бібліотеку i18next із конфігурацією у файлі i18n.js, дозволяючи перемикатися між мовами без втрати функціональності.
- Темна/світла тема: Реалізовано через компонент ThemeToggle.jsx, який зберігає вибір користувача у localStorage, забезпечуючи персоналізацію та комфорт.
- Інтерактивність: Використання WebSocket для реальних сповіщень і бібліотеки react-modal для динамічних вікон підвищує зручність і швидкість реагування системи.

Візуалізація інтерфейсу

Для ілюстрації структури людино-машинного інтерфейсу наведено зображення ключових екранних форм. На (Рис. 2.4) представлено вигляд головної сторінки застосунку, яка включає навігаційну панель із розділами "Головна", "Календар", "Шаблони", поле пошуку, кнопку "Додати нотатку", іконку перемикання теми та виходу чи зміни паролю при натисканні на аватар. Центральна частина сторінки відображає графічний елемент із двома нотатками та інструкцію для нового користувача, що підкреслює інтуїтивність інтерфейсу. На (Рис. 2.5) показано модальне вікно створення нотатки, яке містить поля для введення заголовка, вмісту, дедлайну, нагадування, тегів та файлів, а також кнопки "Зберегти" та "Очистити". Ці зображення демонструють логічну організацію елементів і їхню інтеграцію з функціоналом системи.

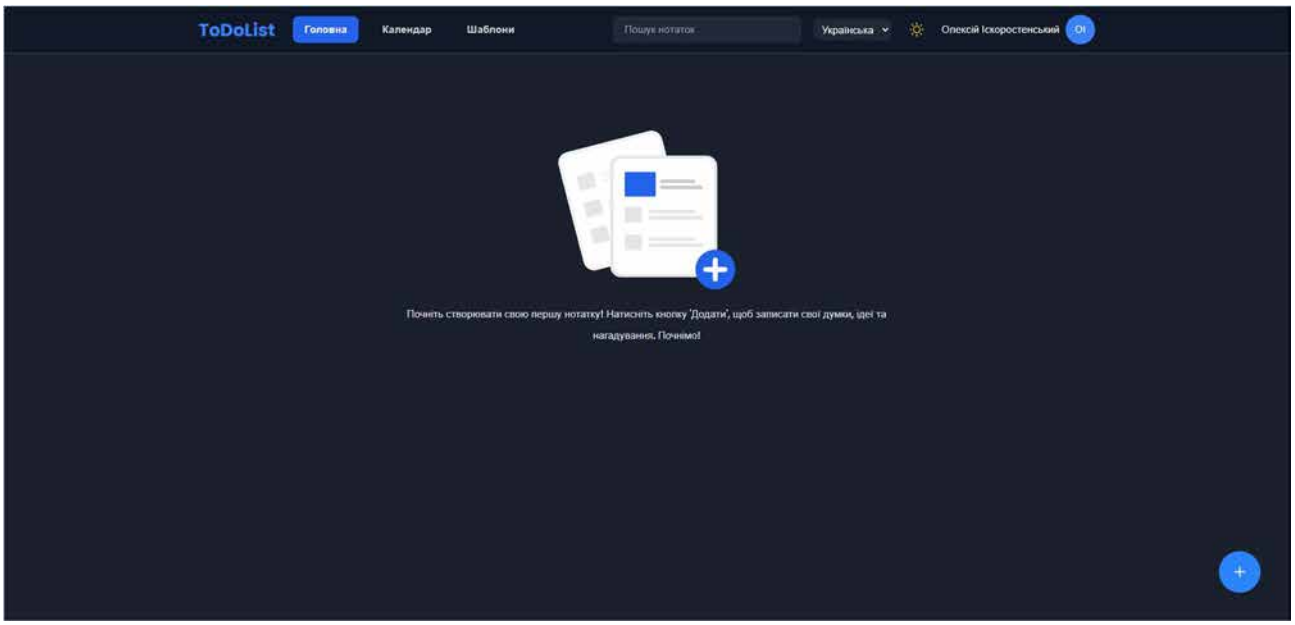

Рисунок 2.4 - Вигляд головної сторінки

Опис: Головна сторінка відображає навігаційну панель, поле пошуку, кнопку "Додати нотатку" та графічний елемент із нотатками, що спрощує перше знайомство з системою.

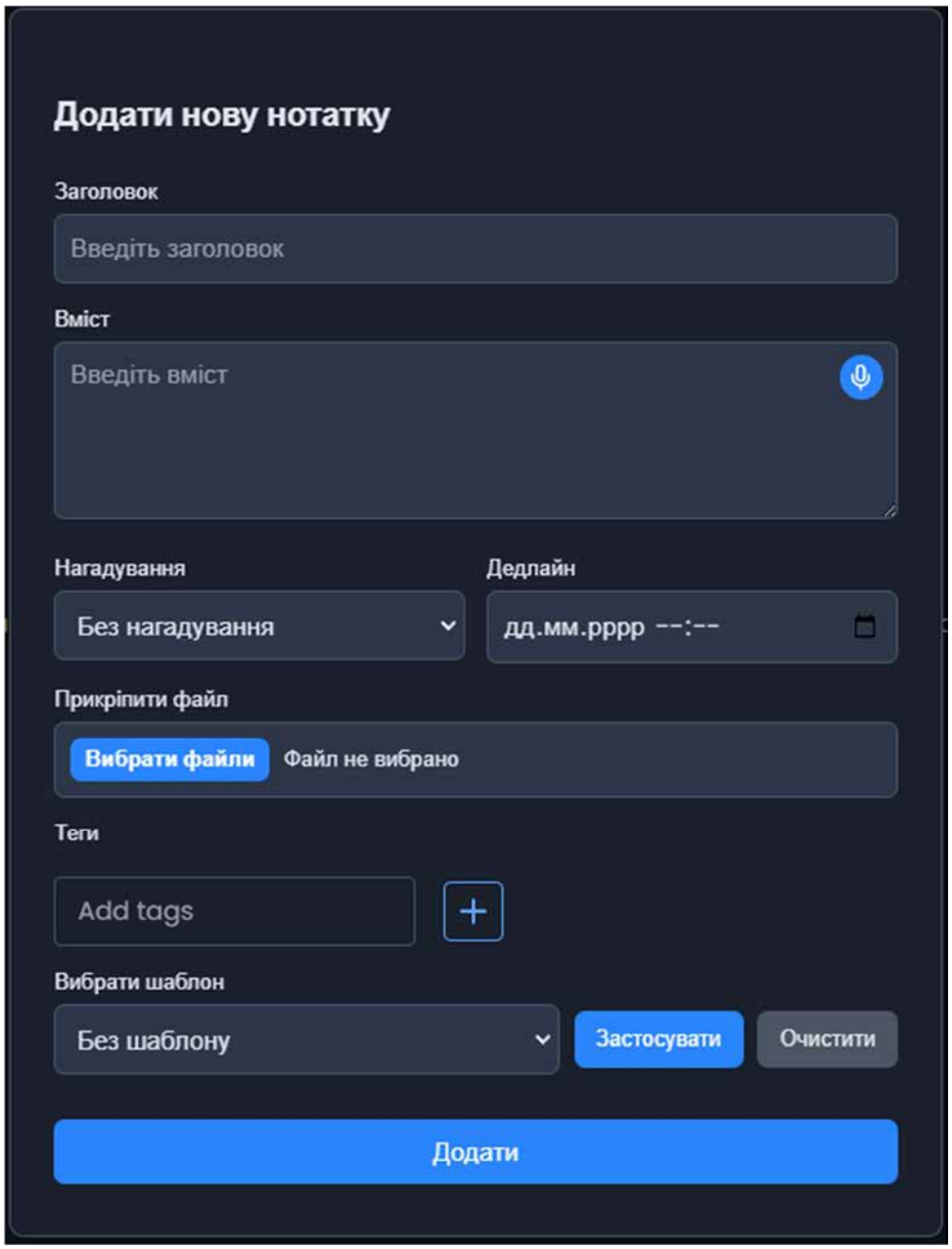

Рисунок 2.5 - Модальне вікно створення нотатки

Опис: Модальне вікно включає поля для введення даних (заголовок, вміст, дедлайн, нагадування, теги, файли) та кнопки дій, забезпечуючи зручність створення нових записів.

Розробка людино-машинного інтерфейсу для веб-застосунку базується на інтеграції інтуїтивних і взаємопов’язаних елементів, реалізованих із застосуванням сучасних принципів дизайну UI/UX. Адаптивність, локалізація, тематична персоналізація та інтерактивність забезпечують високу зручність використання.

3. Реалізація та тестування розробленої інформаційної системи

3.1. Опис модулів системи

Інформаційна система ToDoList складається з двох основних частин: серверної і клієнтської. Кожен компонент системи розбито на модулі, які відповідають за окремі функції, такі як автентифікація, управління нотатками, шаблонами та відображення календаря. Нижче наведено детальний опис модулів системи, їх функціонального призначення та взаємодії.

Серверна частина

Серверна частина розроблена з використанням Node.js та фреймворку Express, із MongoDB як базою даних. Основні модулі бекенду включають:

Модуль автентифікації:

- Функціонал: Реєстрація, вхід, зміна пароля, вихід користувача.
- Реалізація: Ендпоінти /create-account, /login, /change-password, /logout у файлі backend/index.js. Для захисту даних використовується хешування паролів та JSON Web Tokens.
- Модель: User (backend/models/user.model.js), яка містить поля fullName, email, password.

Модуль управління нотатками:

- Функціонал: Створення, редагування, видалення, закріплення, пошук нотаток, підтримка вкладень.
- Реалізація: Ендпоінти /add-note, /edit-note/:noteId, /delete-note/:noteId, /update-note-pinned/:noteId, /search-notes, /get-all-notes у backend/index.js. Завантаження файлів реалізовано через multer з обмеженням до 5 файлів розміром до 5 МБ.

- Модель: Note (backend/models/note.model.js), яка включає поля title, content, tags, userId, deadline, reminder, attachments, isPinned.

Приклад реалізації ендпоінта для створення нотатки (Додаток Б Лістинг Б 1)

Модуль управління шаблонами:

- Функціонал: Створення, редагування, видалення шаблонів для нотаток.
- Реалізація: Ендпоінти /create-template, /edit-template/:templateId, /delete-template/:templateId, /get-all-templates у backend/index.js.
- Модель: Template (backend/models/template.model.js), аналогічна до Note, але без полів reminder та isPinned.

Клієнтська частина

Клієнтська частина побудована на React із використанням Tailwind CSS для стилізації та i18next для української та англійської мов. Основні модулі клієнтської частини включають:

Модуль автентифікації:

- Функціонал: Інтерфейси для реєстрації, входу, зміни пароля.
- Реалізація: Компоненти Login.jsx, SignUp.jsx, ChangePasswordModal.jsx у frontend/notes-app/src/pages. Запити до API виконуються через axiosInstance.js.

Модуль управління нотатками:

- Функціонал: Відображення, створення, редагування, видалення нотаток, пошук, закріплення.
- Реалізація: Компоненти Home.jsx, AddEditNotes.jsx, NoteCard.jsx, ViewNoteModal.jsx. Використовується react-big-calendar для календаря.

Код компонента для відображення нотатки (Додаток Б Лістинг Б 2).

Модуль управління шаблонами:

- Функціонал: Створення, редагування, видалення шаблонів, їх застосування до нотаток.
- Реалізація: Компоненти Template.jsx, AddEditTemplate.jsx, TemplateCard.jsx.

Модуль календаря:

- Функціонал: Відображення дедлайнів нотаток у режимах місяця, тижня, дня, списку.
- Реалізація: Компонент Calendar.jsx із використанням react-big-calendar.

Модуль інтерфейсу:

- Функціонал: Підтримка багатомовності (українська, англійська), темного/світлого режимів, голосового вводу.
- Реалізація: Navbar.jsx для навігації, i18n.js для локалізації, Tailwind CSS для адаптивного дизайну.

3.2. Людино-машинний інтерфейс та інструкції для користувачів

Людино-машинний інтерфейс системи ToDoList розроблено з акцентом на інтуїтивність, адаптивність і підтримку різних категорій користувачів. Інтерфейс побудовано з використанням React і Tailwind CSS, що забезпечує адаптивний дизайн для десктопних і мобільних пристроїв [19]. Система підтримує українську та англійську локалізації, а також темний і світлий режими для комфортної роботи в різних умовах.

Основні елементи інтерфейсу

Навігаційна панель

Містить логотип, меню для переходу до сторінок (Головна, Календар, Шаблони, Профіль), перемикач мови (en/uk) та теми (темна/світла).

Навігаційна панель у світлому та темному режимах (Рис. 3.1).

Показано панель із випадаючим списком мов, виходу і зміни паролю на іконці профіля і кнопкою перемикання теми.

Рисунок 3.1 – Навігаційна панель

Сторінка календаря

Відображає дедлайни нотаток у режимах місяця, тижня, дня, списку. Календар у режимі місяця (Рис. 3.2). Показані події (нотатки).

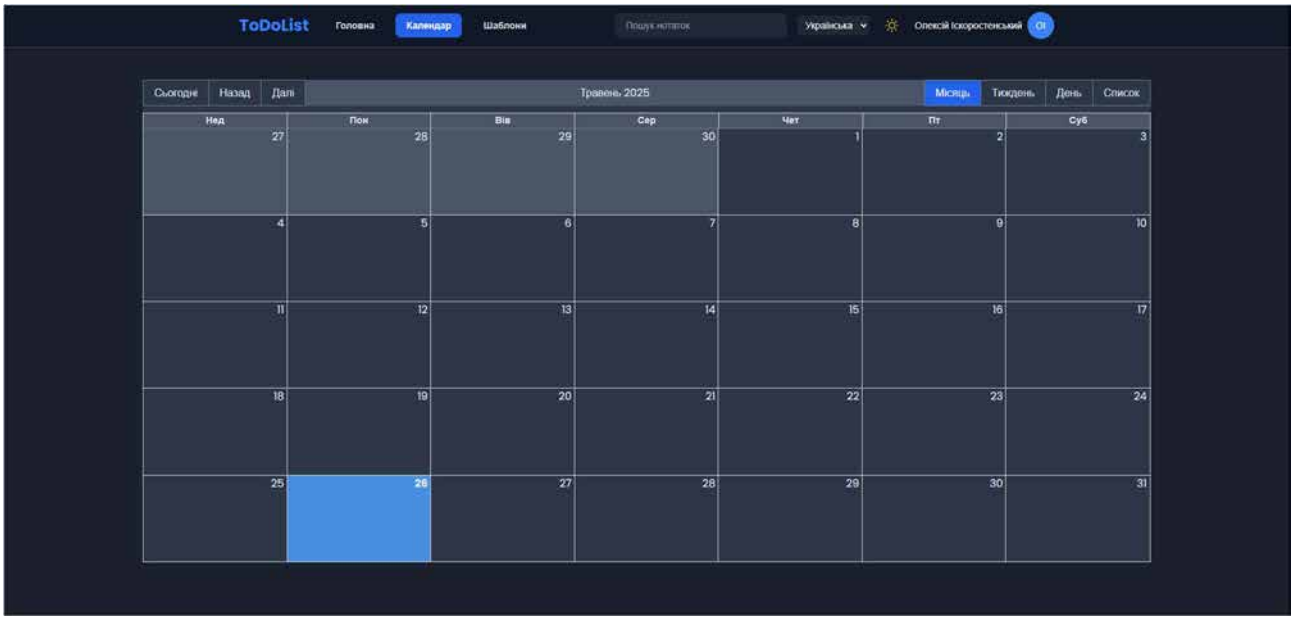

Рисунок 3.2 – Сторінка календаря

Сторінка шаблонів

Відображає список шаблонів, дозволяє створювати, редагувати, видаляти їх.

Сторінка шаблонів (Рис. 3.3). Показані картки шаблонів.

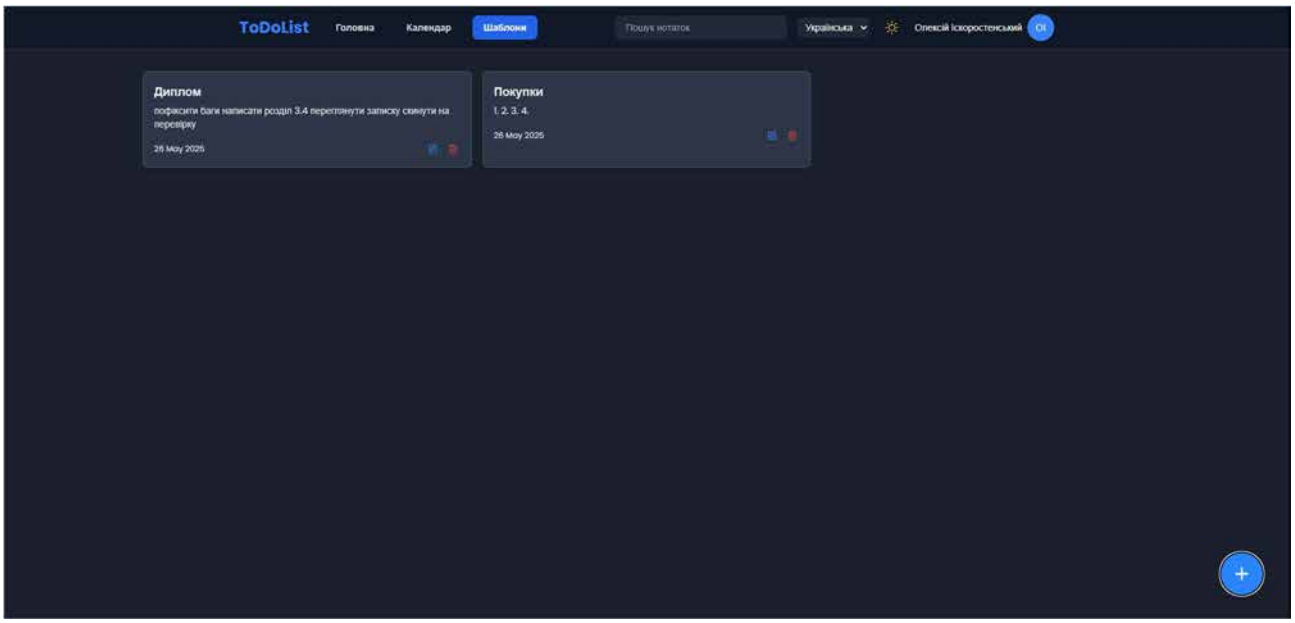

Рисунок 3.3 – Сторінка шаблонів

Сторінки автентифікації

Форми для входу та реєстрації з полями для email, пароля, імені (для реєстрації).

Спливаючі кнопки для зміни мови та теми. Форма входу та реєстрації (Рис.3.4 та Рис. 3.5).

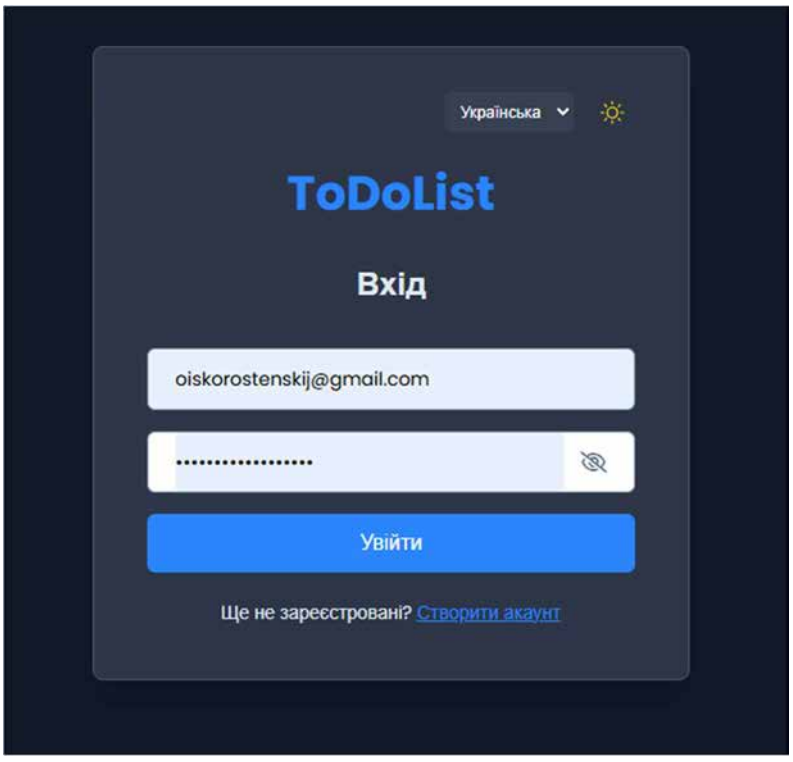

Рисунок 3.4 – Форма входу

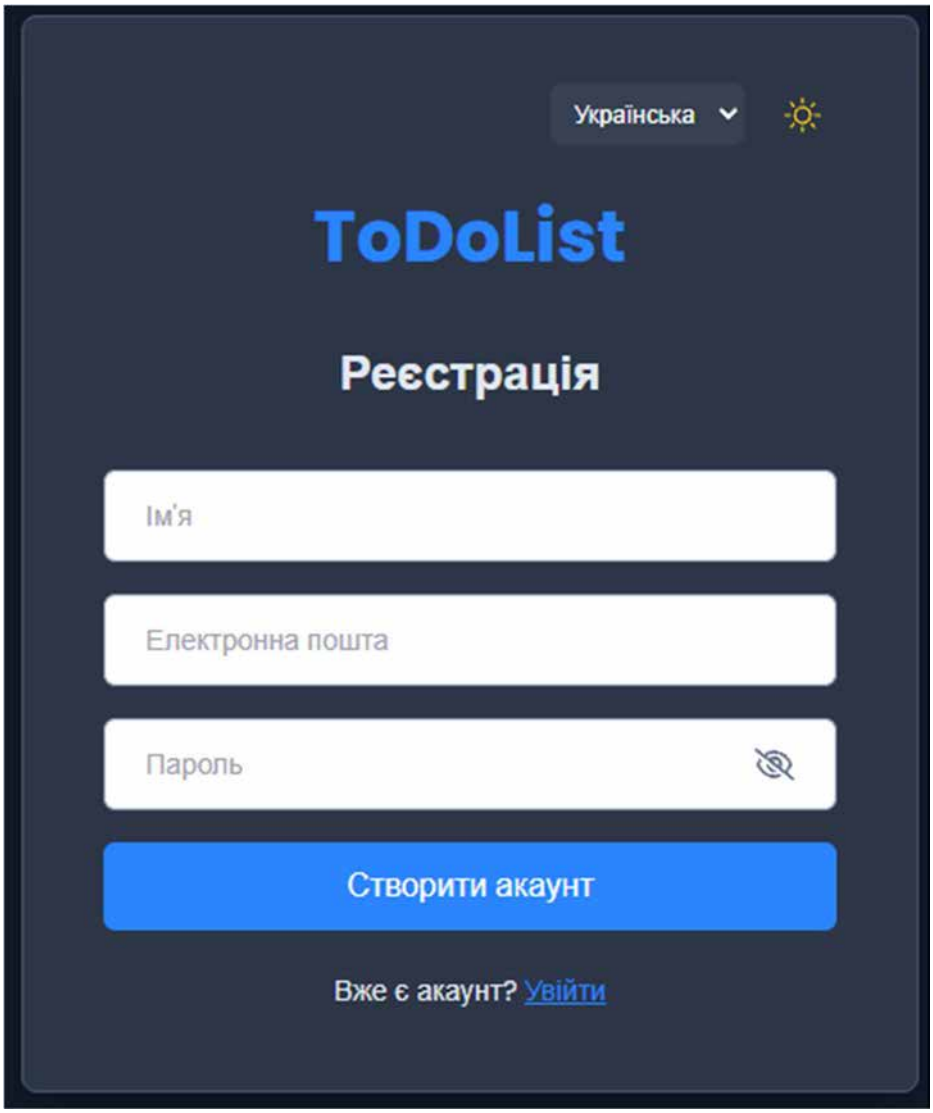

Рисунок 3.5 – Форма Реєстрації

Інструкції для користувачів

Для забезпечення зручності використання системи розроблено інструкції, які охоплюють основні сценарії роботи:

Реєстрація та вхід:

- Перейдіть на сторінку реєстрації, введіть ім’я, email і пароль.
- Після успішної реєстрації увійдіть, використовуючи email і пароль.

Створення нотатки:

- На головній сторінці натисніть кнопку «Додати нотатку».
- Заповніть поля: заголовок, вміст, теги, дедлайн, нагадування.
- Завантажте до 5 файлів (зображення, PDF, Word), якщо потрібно.
- Натисніть «Зберегти» або оберіть шаблон для автоматичного заповнення.

Робота з календарем:

- Перейдіть на сторінку календаря.
- Виберіть режим (місяць, тиждень, день, список).
- Клацніть на дату, щоб створити нотатку, або на подію, щоб редагувати.

Створення шаблону:

- На сторінці шаблонів натисніть «Створити шаблон».
- Заповніть поля аналогічно до нотатки та збережіть.

Налаштування інтерфейсу:

- У навігаційній панелі виберіть мову (українська/англійська).
- Переключіть тему (темна/світла) для комфортної роботи.

3.3. Технічні характеристики системи

Система ToDoList розроблена з урахуванням сучасних вимог до веб-додатків, що забезпечує її продуктивність, масштабованість і безпеку. Нижче наведено технічні характеристики системи, включаючи апаратні, програмні та архітектурні аспекти.

Апаратні вимоги

Сервер

- Процесор: 2-ядерний, 2 ГГц або вище.
- Оперативна пам’ять: 4 ГБ.
- Дисковий простір: 10 ГБ для бази даних і файлів вкладень.

- Операційна система: Windows Server.

Клієнт

- Пристрій: Персональний комп’ютер, планшет або смартфон.
- Браузер: Chrome, Safari, Edge.
- Інтернет-з’єднання: 1 Мбіт/с або вище.

Програмне забезпечення

Сервер:

- Node.js: v16.x або вище.
- MongoDB: v4.4 або вище.
- Залежності: Express, mongoose, bcrypt, jsonwebtoken, multer, ws, node-schedule.

Клієнт:

- React: v18.x.
- Залежності: axios, i18next, react-big-calendar, tailwindcss, react-speech-recognition.
- Браузер із підтримкою WebSocket і Notification API.

Архітектура системи

Система базується на клієнт-серверній архітектурі:

- Бекенд: REST API та WebSocket для асинхронних нагадувань. MongoDB використовується для зберігання даних користувачів, нотаток і шаблонів.
- Фронтенд: Односторінковий додаток на React, який взаємодіє з бекендом через HTTP-запити та WebSocket.
- Безпека: JWT для автентифікації, HTTPS для шифрування, хешування паролів.

3.4. Стратегія та методика тестування

Якість програмного забезпечення – це ключовий аспект розробки, який прямо визначає, наскільки продукт буде функціональним, надійним, продуктивним та зручним у використанні. Щоб наш веб-застосунок відповідав усім вимогам та мав мінімум дефектів, ми розробили та застосували комплексну стратегію тестування, яка охоплює різні рівні та типи перевірок.

Визначення стратегій тестування

В основі нашої стратегії тестування лежать принципи ітеративного та інкрементального підходів, які ми інтегрували безпосередньо в процес розробки. Це означає, що ми тестуємо компоненти системи поетапно: від найпростіших до складніших, а також перевіряємо нефункціональні вимоги. Головне для нас – переконатися, що розроблений продукт працює правильно (верифікація) і відповідає очікуванням користувачів (валідація), а також виявити та виправити можливі дефекти якомога раніше.

Рівні тестування:

Модульне тестування – це тестування окремих, ізольованих компонентів або функцій застосунку.

- Для серверної частини це перевірка коректності роботи окремих функцій контролерів, моделей, допоміжних функцій.
- Для клієнтської частини це тестування окремих React-компонентів в ізоляції. Перевірялось їх візуальне відображення при різних запитах, коректна обробка введення користувача та ініціалізація стану.

Інтеграційне тестування – це перевірка взаємодії між модулями та компонентами системи.

- Для серверної частини: Тестування взаємодії між контролерами, сервісами та базою даних . Перевірка роботи API-ендпоінтів, що вимагають взаємодії кількох модулів.

- Для клієнтської частини: Тестування взаємодії React-компонентів з API-сервісами.

Системне тестування: Комплексна перевірка всієї системи як єдиного цілого.

- Тестування повних сценаріїв користувача: реєстрація, вхід, створення, редагування, видалення завдань, пошук, фільтрація за тегами, закріплення, відкріплення нотаток, використання шаблонів, управління профілем, перемикання мови та теми.
- Тестування взаємодії між фронтендом і бекендом у реальних умовах експлуатації, включаючи передачу файлів та обробку помилок.

Приймальне тестування: Проводиться для підтвердження відповідності системи до вимог та готовності розгортання. На цьому етапі може залучатись кінцевий користувач для верифікації згідно очікування використання.

Типи тестування:

1. Функціональне тестування: Перевірка відповідності всіх функціональних вимог, описаних у технічному завданні. Це включає тестування всіх CRUD-операцій для нотаток, користувачів, шаблонів, а також специфічних функцій, таких як прикріплення файлів, встановлення дедлайнів, закріплення нотаток та пошук.
2. Тестування безпеки: Перевірка вразливостей системи. Охоплює тестування:

- Коректності механізмів аутентифікації та авторизації перевірка токенів JWT.
- Хешування паролів безпечне зберігання паролів у базі даних.
- Обробки файлових завантажень.

3. Тестування зручності використання: Оцінка легкості та інтуїтивності використання інтерфейсу користувачами. Перевіряється зрозумілість навігації, розміщення елементів керування, зручність роботи з формами.

4. Кроссбраузерне тестування: Перевірка коректного відображення та функціональності інтерфейсу в різних сучасних веб-браузерах для забезпечення консистентного користувацького досвіду.
5. Локалізаційне тестування: Перевірка коректності відображення текстів та перекладу функціональних елементів при перемиканні мови інтерфейсу.
6. Тестування темної та світлої теми: Перевірка коректності застосування стилів та читабельності елементів інтерфейсу при перемиканні між світлою та темною темами.

Методика тестування

Для реалізації вищезгаданої стратегії тестування була обрана методика, що поєднує ручне тестування як основний метод верифікації функціоналу. Автоматизоване тестування, хоча і є перспективним напрямком, не було основним фокусом даного етапу розробки.

Ручне тестування:

Ручне тестування застосовувалось на всіх рівнях, особливо на етапах системного та приймального тестування, для перевірки загального користувацького досвіду, візуального оформлення та неочевидних сценаріїв взаємодії. Це включає:

- Позитивні сценарії: Перевірка очікуваної поведінки системи при коректних вхідних даних та стандартних діях користувача.
- Негативні сценарії: Перевірка коректної обробки помилок та відхилень при некоректних вхідних даних, невірних діях користувача.

Вимоги до проведення експериментів

Для забезпечення об'єктивності та відтворюваності результатів тестування були сформовані наступні вимоги:

- Ізольоване середовище: Тестування проводиться у контрольованому середовищі, ізольованому від реальних даних, щоб уникнути їх

пошкодження або некоректних змін. Це означає використання окремої тестової бази даних MongoDB.

- Тестові дані: Набір тестових даних, що охоплюють як типові, так і інші випадки використання. Це включає створення тестових користувачів, завдань з різними статусами, тегами, дедлайнами та вкладеннями різних типів.
- Фіксація результатів: Результати кожного виконаного тест-кейсу документуються, включаючи його ідентифікатор, короткий опис, вхідні та очікувані вихідні значення, фактичні вихідні значення, а також інформацію про збіг чи розбіжності цих значень та статус проходження.
- Відтворюваність: Тест-кейси спроектовані таким чином, щоб їх можна було багаторазово повторити з однаковими вхідними даними, що забезпечує стабільність результатів та можливість верифікації виправлень дефектів.

3.5. Результати тестування

Після реалізації програмно-технічних засобів та формування стратегії і методики тестування, наступним етапом є безпосереднє проведення експериментів та фіксація їх результатів. Цей підрозділ містить опис процесу виконання тест-кейсів, аналіз отриманих даних та їх зіставлення з очікуваними результатами, що дозволяє оцінити ступінь працездатності та відповідності розробленої системи вимогам технічного завдання.

Організація проведення тестування

Тестування розробленого веб-застосунку проводилось у спеціально підготовленому тестовому середовищі, що імітує умови реальної експлуатації. Це середовище включало:

- Серверна частина: Node.js з Express.js, запущений локально на порту 8000.
- База даних: MongoDB.

- Клієнтська частина: Веб-застосунок на React.js, доступний через веб-браузер за адресою http://localhost:5173.

Розроблені тест-кейси та їх виконання

Наведено перелік розроблених тест-кейсів.

ID Тест-кейсу: NOTE-001

Назва Тест-кейсу: Створення нової нотатки з усіма полями

Опис: Перевірка успішного створення нотатки з заповненням всіх можливих полів.

Передумови: Користувач авторизований.

Кроки: 1.Перейти на сторінку створення нотатки. 2.Ввести заголовок, зміст, додати теги, встановити дедлайн, завантажити файл. 3.Натиснути кнопку "Зберегти нотатку".

Очікуваний результат: Нотатка успішно створена та відображається в списку нотаток з усіма введеними даними.

Фактичний результат: Після створення з’являється повідомлення про успішне створення.

Статус: Успіх (див. рис. В 1 Додаток В).

ID Тест-кейсу: NOTE-002

Назва Тест-кейсу: Створення нотатки без заголовка (або обов'язкового поля).

Опис: Перевірка валідації при створенні нотатки без обов'язкових полів.

Передумови: Користувач авторизований.

Кроки: 1.Перейти на сторінку створення нотатки. 2.Спробувати створити нотатку, залишивши заголовок порожнім. 3.Натиснути кнопку "Зберегти нотатку".

Очікуваний результат: Система виводить повідомлення про помилку валідації для незаповнених полів.

Фактичний результат: Система вивела повідомлення "Будь ласка, введіть заголовок"

Статус: Успіх (див. рис.В 2 Додаток В).

ID Тест-кейсу: NOTE-003

Назва Тест-кейсу: Редагування існуючої нотатки

Опис: Перевірка успішного редагування існуючої нотатки.

Передумови: Користувач авторизований, існує хоча б одна нотатка.

Кроки: 1. Перейти до списку нотаток. 2. Вибрати існуючу нотатку для редагування. 3. Внести зміни в заголовок, зміст, теги, дедлайн або вкладення. 4.Натиснути кнопку "Зберегти зміни".

Очікуваний результат: Нотатка успішно оновлена, зміни відображаються в списку нотаток.

Фактичний результат: Виведення повідомлення про успішне оновлення, і зміни відображаються коректно.

Статус: Успіх (див. рис.В 3 Додаток В).

ID Тест-кейсу: NOTE-004

Назва Тест-кейсу: Перегляд деталей нотатки

Опис: Перевірка відображення всіх деталей нотатки.

Передумови: Користувач авторизований, існує хоча б одна нотатка.

Кроки: 1.Перейти до списку нотаток. 2.Натиснути на нотатку для перегляду її деталей.

Очікуваний результат: Відкривається вікно з усіма деталями нотатки, включаючи заголовок, зміст, теги, дедлайн та вкладення.

Фактичний результат: Всі деталі нотатки відображаються коректно.

Статус: Успіх (див. рис. В 4 Додаток В).

ID Тест-кейсу: NOTE-005

Назва Тест-кейсу: Пошук нотаток

Опис: Перевірка функціональності пошуку нотаток за ключовими словами.

Передумови: Користувач авторизований, є кілька нотаток.

Кроки: 1.Перейти до списку нотаток. 2.Ввести ключове слово в поле пошуку. 3. Натиснути кнопку пошуку.

Очікуваний результат: Відображаються лише нотатки, які містять введене ключове слово в заголовку або змісті.

Фактичний результат: Відображення нотатки з ключовим словом.

Статус: (див. рис. В 5 Додаток В).

ID Тест-кейсу: NOTE-007

Назва Тест-кейсу: Перегляд завантаженого вкладення

Опис: Перевірка можливості відкриття та перегляду завантажених вкладень.

Передумови: Користувач авторизований, нотатка має вкладення.

Кроки: 1.Перейти до деталей нотатки з вкладенням. 2.Натиснути на назву вкладення.

Очікуваний результат: Вкладення успішно відкривається у браузері або завантажується на пристрій.

Фактичний результат: Відкриття в браузері.

Статус: Успіх. (див. рис. В 6 Додаток В).

Висновок

В результаті виконання цієї бакалаврської кваліфікаційної роботи ми успішно розробили та реалізували веб-застосунок для управління особистими завданнями. Ми досягли поставленої мети, створивши зручний, безпечний та функціональний інструмент, який забезпечує ефективну цифрову організацію справ користувача та сприяє підвищенню його особистої продуктивності.

У процесі дослідження та розробки було вирішено такі ключові завдання:

Проведено ґрунтовний аналіз предметної області управління особистими завданнями та здійснено огляд існуючих програмних рішень. Це дозволило визначити конкурентні переваги та сформувати вимоги до власного застосунку, зокрема акцентуючи на повноцінній підтримці української мови, можливості голосового введення, гнучкій роботі з файлами та шаблонами без необґрунтованих платних обмежень.

Сформульовано детальні функціональні та нефункціональні вимоги, на основі яких розроблено технічне завдання. Воно охопило розширений набір можливостей: безпечну автентифікацію користувачів, комплексне управління нотатками та шаблонами, прикріплення файлів різних форматів, інтеграцію з календарем, багатомовну локалізацію та налаштування тем інтерфейсу.

Розроблено та реалізовано клієнт-серверну архітектуру системи. Клієнтська частина, створена з використанням React, Vite та Tailwind CSS, забезпечує адаптивний та інтуїтивно зрозумілий інтерфейс. Серверна частина, побудована на Node.js та Express.js із використанням бази даних MongoDB, відповідає за обробку логіки, зберігання даних та безпеку.

Здійснено декомпозицію системи на логічно завершені модулі, що підвищило її масштабованість, полегшило розробку, тестування та подальший супровід.

Проведено комплексне тестування розробленого веб-застосунку, яке підтвердило його функціональну придатність, зручність використання.

Забезпечено надійну та ефективну взаємодію між підсистемами через REST API та WebSocket, що дозволило реалізувати систему сповіщень у реальному часі та інтерактивну роботу з файлами.

Розроблений веб-застосунок надає користувачам такі переваги:

Скорочення часу на організацію та планування завдань.

Підвищення особистої ефективності завдяки систематизації справ.

Гнучкість та персоналізація користувацького досвіду через адаптивний дизайн, вибір тем, мови інтерфейсу та перегляд завдань у календарі.

Безпечне зберігання особистих даних із використанням сучасних методів автентифікації та захисту інформації.

Практичне значення роботи полягає у створенні готового до використання програмного продукту, який може бути адаптований не лише для індивідуального використання, але й для потреб корпоративного сектору, навчальних закладів чи інших організацій, де існує потреба в ефективному плануванні завдань. Функціонал системи є гнучким і може бути розширений, наприклад, шляхом інтеграції з популярними календарями (Google Calendar), хмарними сховищами або впровадженням елементів штучного інтелекту для інтелектуальної допомоги у плануванні.

Таким чином, усі поставлені у вступі завдання бакалаврської кваліфікаційної роботи виконані в повному обсязі, а мета дослідження досягнута. Отримані результати демонструють практичну цінність розробленого веб-застосунку та можуть слугувати основою для подальших наукових досліджень і комерційних розробок у галузі інформаційних технологій для управління особистою продуктивністю.

Список використаних джерел

1. Axios: Introduction. Axios-http.com. [Електронний ресурс]. — Режим доступу: https://axios-http.com/docs/intro. — Назва з екрана. — Дата звернення: 01.05.2025.
2. Bcrypt.js. Npmjs.com. [Електронний ресурс]. — Режим доступу: https://www.npmjs.com/package/bcryptjs. — Назва з екрана. — Дата звернення: 02.05.2025.
3. Django Project. Djangoproject.com. [Електронний ресурс]. — Режим доступу: https://www.djangoproject.com/. — Назва з екрана. — Дата звернення: 03.05.2025.
4. Express – Node.js web application framework. Nodejs.org. [Електронний ресурс]. — Режим доступу: https://nodejs.org/en/. — Назва з екрана. — Дата звернення: 04.05.2025.
5. Google Keep. Google.com. [Електронний ресурс]. — Режим доступу: https://keep.google.com/. — Назва з екрана. — Дата звернення: 05.05.2025.
6. i18next: Getting Started. i18next.com. [Електронний ресурс]. — Режим доступу: https://www.i18next.com/overview/getting-started. — Назва з екрана. — Дата звернення: 06.05.2025.
7. Introduction to JSON Web Tokens. JWT.IO. [Електронний ресурс]. — Режим доступу: https://jwt.io/introduction. — Назва з екрана. — Дата звернення: 07.05.2025.
8. Microsoft To Do. Office.com. [Електронний ресурс]. — Режим доступу: https://to-do.office.com/tasks/. — Назва з екрана. — Дата звернення: 08.05.2025.
9. MongoDB Manual. Mongodb.com. [Електронний ресурс]. — Режим доступу: https://www.mongodb.com/docs/. — Назва з екрана. — Дата звернення: 09.05.2025.
10. Mongoose ODM Guide. Mongoosejs.com. [Електронний ресурс]. — Режим доступу: https://mongoosejs.com/docs/guide.html. — Назва з екрана. — Дата звернення: 10.05.2025.
11. Multer. Npmjs.com. [Електронний ресурс]. — Режим доступу: https://www.npmjs.com/package/multer. — Назва з екрана. — Дата звернення: 11.05.2025.
12. MySQL. MySQL.com. [Електронний ресурс]. — Режим доступу: https://www.mysql.com/. — Назва з екрана. — Дата звернення: 12.05.2025.
13. Node.js API Documentation. Nodejs.org. [Електронний ресурс]. — Режим доступу: https://nodejs.org/docs/latest/api/. — Назва з екрана. — Дата звернення: 13.05.2025.

14. Notion. Notion.com. [Електронний ресурс]. — Режим доступу: https://www.notion.com/. — Назва з екрана. — Дата звернення: 14.05.2025.
15. PostgreSQL. Postgresql.org. [Електронний ресурс]. — Режим доступу: https://www.postgresql.org/. — Назва з екрана. — Дата звернення: 15.05.2025.
16. React Big Calendar: About. Jquense.github.io. [Електронний ресурс]. — Режим доступу: https://jquense.github.io/react-big-calendar/examples/index.html?path=/story/about-big-calendar--page. — Назва з екрана. — Дата звернення: 16.05.2025.
17. React Router: Home. Reactrouter.com. [Електронний ресурс]. — Режим доступу: https://reactrouter.com/home. — Назва з екрана. — Дата звернення: 17.05.2025.
18. Spring Framework. Spring.io. [Електронний ресурс]. — Режим доступу: https://spring.io/projects/spring-framework. — Назва з екрана. — Дата звернення: 18.05.2025.
19. Tailwind CSS Documentation: Installation using Vite. Tailwindcss.com. [Електронний ресурс]. — Режим доступу: https://tailwindcss.com/docs/installation/using-vite. — Назва з екрана. — Дата звернення: 19.05.2025.
20. Todoist. Todoist.com. [Електронний ресурс]. — Режим доступу: https://www.todoist.com/. — Назва з екрана. — Дата звернення: 20.05.2025.
21. Vite Guide. Vite.dev. [Електронний ресурс]. — Режим доступу: https://vite.dev/guide/. — Назва з екрана. — Дата звернення: 21.05.2025.
22. Web Speech API. Developer.mozilla.org. [Електронний ресурс]. — Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API. — Назва з екрана. — Дата звернення: 22.05.2025.

Додаток А

Лістинг А 1

```
import axios from 'axios';

const addNote = async (noteData) => {
  try {
    const token = localStorage.getItem('token');
    const response = await axios.post(
      'http://localhost:5000/add-note',
      noteData,
      {
        headers: {
          Authorization: Bearer ${token},
          'Content-Type': 'application/json',
        },
      }
    );
    console.log('Note added successfully:', response.data);
  } catch (error) {
    console.error('Error adding note:', error.response?.data || error.message);
  }
};
addNote({
  title: 'Нова нотатка',
```

```
  content: 'Опис нотатки...',
  tags: ['приклад', 'тест'],
  deadline: '2025-05-30T10:00:00Z',
});
```

Лістинг А 2

```
app.post('/add-note', authenticateToken, async (req, res) => {
  const { title, content, tags, deadline } = req.body;
  const { user } = req.user;
  if (!title || !content) {
    return res.status(400).json({ error: true, message: 'Title and content are required' });
  }
  try {
    const newNote = new Note({
      title,
      content,
      tags,
      deadline: deadline ? new Date(deadline) : null,
      userId: user._id,
    });
    await newNote.save();
    res.status(201).json({ success: true, note: newNote });
  } catch (error) {
    console.error('Error adding note:', error);
```

```
    res.status(500).json({ error: true, message: 'Internal Server Error' });
  }
});
```

Лістинг А 3

```
const jwt = require('jsonwebtoken');
app.post('/login', async (req, res) => {
  const { email, password } = req.body;
  const user = await User.findOne({ email });
  if (!user) {
    return res.status(400).json({ error: true, message: 'User not found' });
  }
  const isMatch = await bcrypt.compare(password, user.password);
  if (!isMatch) {
    return res.status(400).json({ error: true, message: 'Invalid credentials' });
  }
  const token = jwt.sign({ user: { _id: user._id } },
process.env.ACCESS_TOKEN_SECRET, { expiresIn: '1d' });
  res.json({ token });
});
```

Лістинг А 4

```
const ws = new WebSocket('ws://localhost:8080?token=USER_JWT_TOKEN');
ws.onmessage = (event) => {
  const data = JSON.parse(event.data);
  if (data.type === 'reminder') {
```

```
    alert(Reminder: ${data.title});
  }
};
```

Лістинг А 5

```
const wss = new WebSocket.Server({ port: 8080 });
wss.on('connection', (ws, req) => {
  const token = req.url.split('token=')[1];
  if (!token) {
    ws.close();
    return;
  }
  jwt.verify(token, process.env.ACCESS_TOKEN_SECRET, (err, decoded) => {
    if (err) {
      ws.close();
      return;
    }
    const userId = decoded.user._id;
    console.log(`User ${userId} connected`);
    ws.on('message', (message) => console.log(`Received: ${message}`));
    ws.on('close', () => console.log(`User ${userId} disconnected`));
  });
});
```

Лістинг А 6

```
const uploadFile = async (file) => {
  const formData = new FormData();
  formData.append('file', file);
  try {
```

```
    const response = await axios.post('http://localhost:5000/upload', formData, {
      headers: {
        Authorization: Bearer ${token},
        'Content-Type': 'multipart/form-data',
      },
    });
    console.log('File uploaded successfully:', response.data);
  } catch (error) {
    console.error('Error uploading file:', error.response?.data || error.message);
  }
};
```

Лістинг А 7

```
const multer = require('multer');

const storage = multer.diskStorage({
  destination: './uploads',
  filename: (req, file, cb) => {
    cb(null, ${Date.now()}-${file.originalname});
  },
});
const upload = multer({ storage });
app.post('/upload', authenticateToken, upload.single('file'), (req, res) => {
  if (!req.file) {
    return res.status(400).json({ error: true, message: 'No file uploaded' });
  }
  res.status(200).json({ success: true, filePath: /uploads/${req.file.filename} });
```

Додаток Б

Лістинг Б 1

```
// backend/index.js

app.post("/add-note", authenticateToken, upload.array("files", 5), async (req, res) =>
{
  const { title, content, tags, deadline, reminder } = req.body;
  const { user } = req.user;
  if (!title  !content) {
    return res.status(400).json({ error: true, message: "Title and content are required"
});
  }
  try {
    const attachments = req.files ? req.files.map((file) => ({
      filename: file.originalname,
      path: path.join("Uploads", file.filename).replace(__dirname, "").replace(/^[\\\/]/,
""),
      mimetype: file.mimetype,
      size: file.size,
    })) : [];
    const note = new Note({
      title,
      content,
      tags: tags ? JSON.parse(tags) : [],
      userId: user._id,
      deadline: deadline ? new Date(deadline) : null,
      reminder: reminder  null,
      attachments,
    });
```

```
    await note.save();
    return res.json({
      error: false,
      note,
      message: "Note added successfully",
    });
  } catch (error) {
    console.error("Error adding note:", error);
    return res.status(500).json({ error: true, message: "Internal Server Error" });
  }
});
```

Лістинг Б 2

```
// frontend/notes-app/src/components/Cards/NoteCard.jsx
const NoteCard = ({ title, content, date, tags, deadline, attachments, onEdit, onDelete, onPinNote, isPinned, noteId, getAllNotes, onView }) => {
  return (
    <div className="col-span-1 p-4 border rounded-md shadow-sm dark:border-dark-border dark:bg-dark-card">
      <h4 className="text-lg font-medium">{title}</h4>
      <p className="text-sm text-gray-600 dark:text-dark-text">{content}</p>
      {tags.map((tag, index) => <span key={index} className="badge">{tag}</span>)}
      {deadline && <p className={new Date(deadline) < new Date() ? "badge-overdue" : "badge-active"}>{moment(deadline).format("DD MMM YYYY")}</p>}
      {attachments?.length > 0 && attachments.map((file) => (
        <div key={file._id} onClick={() => handleOpenFile(file._id, file.filename, file.mimetype)}>
          {file.mimetype.startsWith("image/") ? <FileImage /> : file.mimetype === "application/pdf" ? <FileText /> : <File />}
```

```
          {decodeURIComponent(file.filename)}
        </div>
      ))}
      <div className="flex space-x-2">
        <Eye onClick={() => onView({ _id: noteId })} title={t("view")} />
        <Edit onClick={() => onEdit({ _id: noteId })} title={t("edit")} />
        <Pin onClick={() => onPinNote(noteId, !isPinned)} title={isPinned ? t("unpin")
: t("pin")} />
        <Trash2 onClick={() => onDelete(noteId)} title={t("remove")} />
      </div>
    </div>
  );
};
```

Додаток В

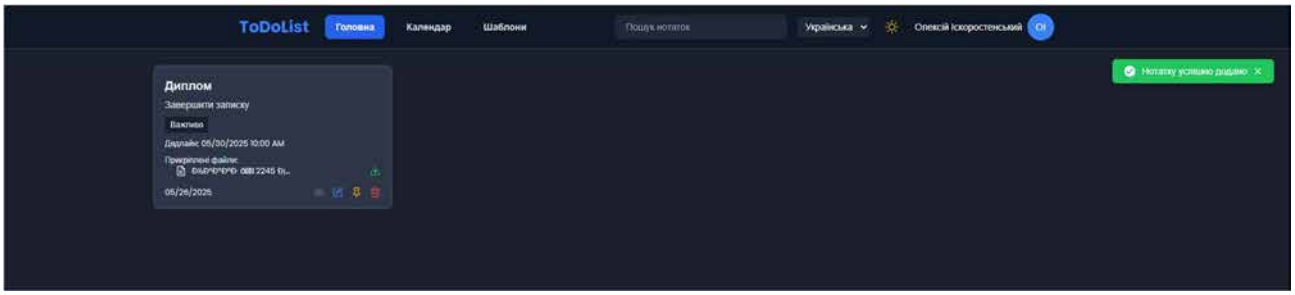

Рисунок В 1 – Успішне створення нотатки

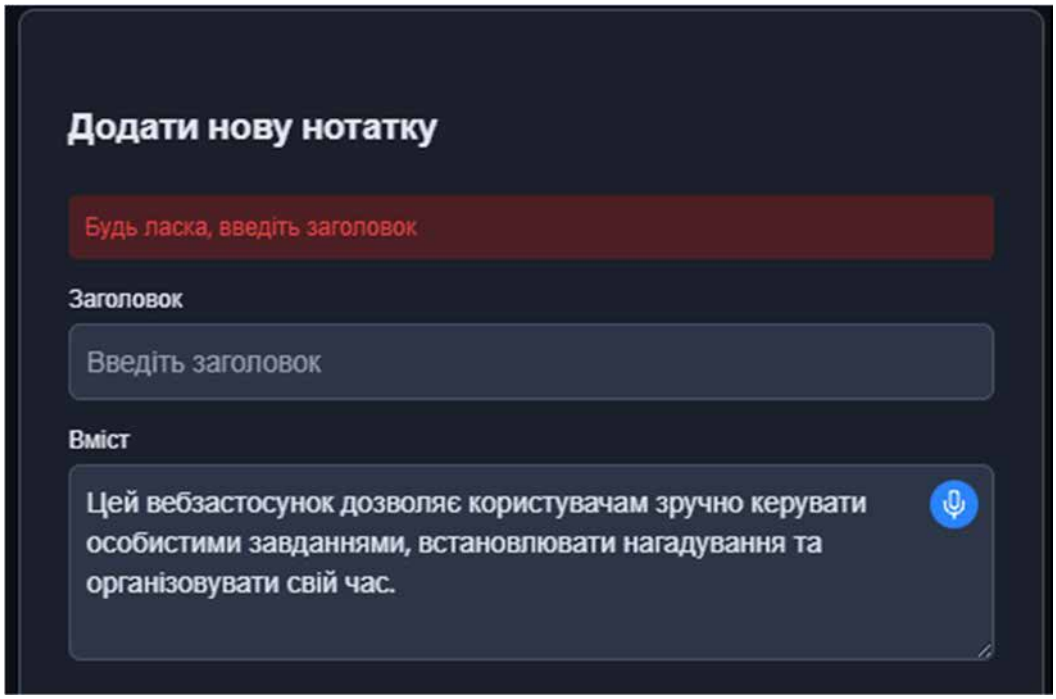

Рисунок В 2 - Помилка валідації для незаповнених полів

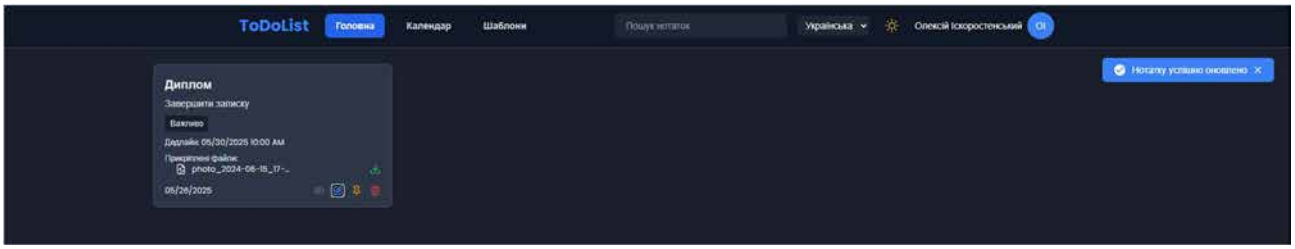

Рисунок В 3 – Успішне оновлення нотатки

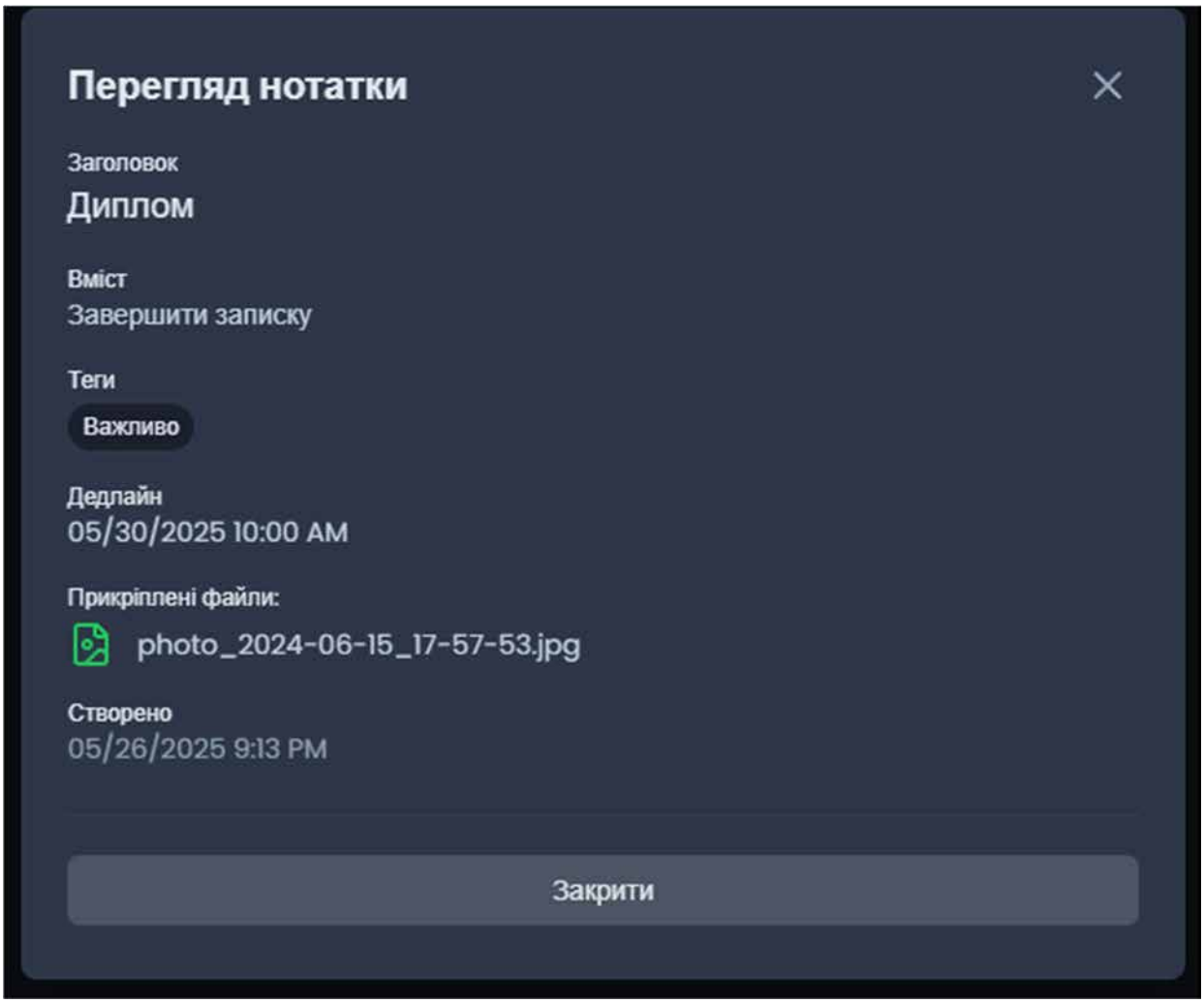

Рисунок В 4 – Вікно з усіма деталями нотатки

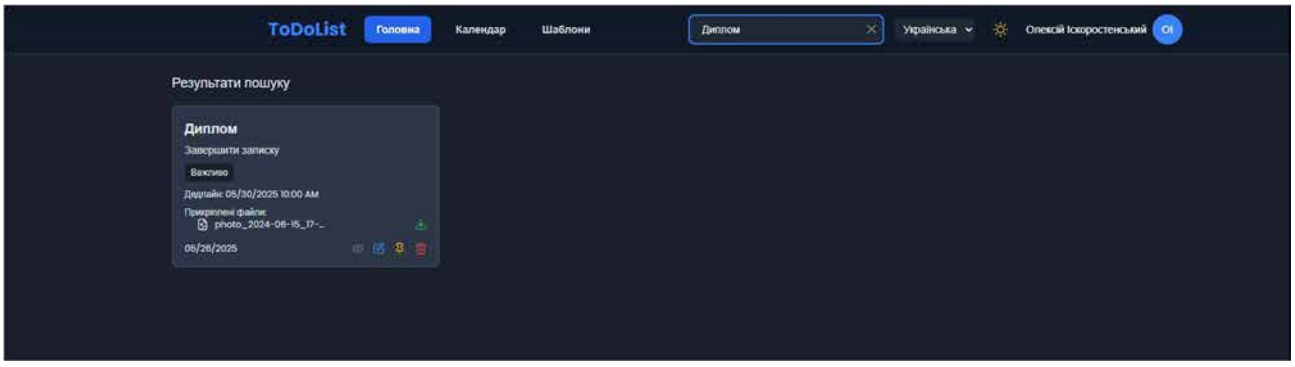

Рисунок В 5 - Відображення нотатки з ключовим словом

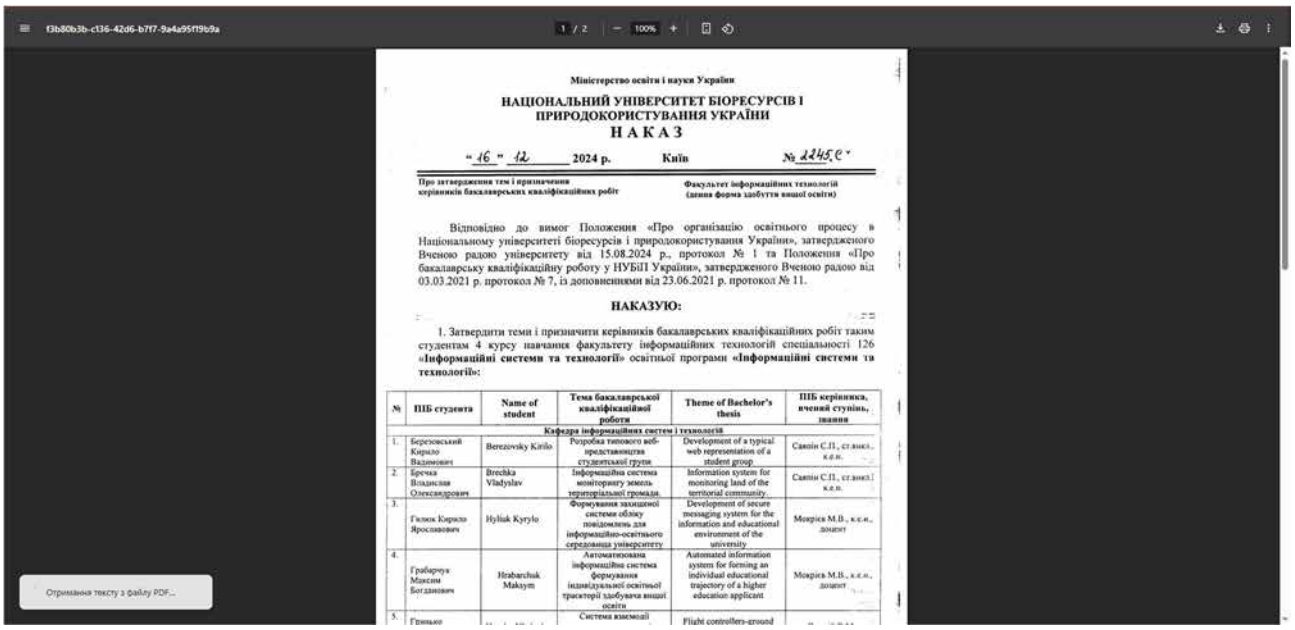

Рисунок В 6 – Відкриття в браузері