Полтавський університет економіки і торгівлі
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)
«_____»_____2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
«РОЗРОБКА ВЕБ-ДОДАТКУ ДЛЯ СПІЛЬНОГО ВЕДЕННЯ НОТАТОК»

зі спеціальності 122 «Комп’ютерні науки»
освітня програма «Комп’ютерні науки»
ступеня бакалавра

Виконавець роботи Мяло Роман Миколайович
_____ «___»_____2026 р.
(підпис)
Науковий керівник к., ф.-м. н., доц., Черненко Оксана Олексіївна
_____ «___»_____2026 р.
(підпис)
Рецензент

ПОЛТАВА 2026

ЗАТВЕРДЖУЮ
Завідувач кафедри _____Олена ОЛЬХОВСЬКА
«____»_____ 202_ р.

ЗАВДАННЯ ТА КАЛЕНДАРНИЙ ГРАФІК
ВИКОНАННЯ КВАЛІФАКАЦІЙНОЇ РОБОТИ

на тему «розробка веб-додатку для спільного ведення нотаток»
зі спеціальності 122 «Комп'ютерні науки»
освітня програма «Комп'ютерні науки»
ступеня бакалавр
Прізвище, ім'я, по батькові Мяло Роман Миколайович
Затверджена наказом ректора № 213-Н від «01» жовтня 2025 р.
Термін подання студентом роботи «___» _____ 20__ р.
Вихідні дані до кваліфікаційної роботи: публікації та лекційний матеріал з теми, навчальні тренажери в дистанційних курсах з комп'ютерних наук.
Зміст пояснювальної записки (перелік питань, які потрібно розробити)

ВСТУП
1. ПОСТАНОВКА ЗАДАЧІ
2. ІНФОРМАЦІЙНИЙ ОГЛЯД
   2.1. Огляд існуючих систем управління знаннями та платформ для спільної роботи
   2.2. Аналіз архітектурних підходів до синхронізації даних: OT проти CRDT
   2.3. Дослідження специфіки User Experience (UX) у додатках реального часу
   2.4. Обґрунтування необхідності та актуальності розробки
3. ТЕОРЕТИЧНА ЧАСТИНА
   3.1. Теоретичні засади побудови розподілених систем спільного редагування
   3.2. Алгоритмічна база та структури даних для безконфліктної реплікації (YATA)
   3.3. Проєктування архітектури системи (UML-діаграми)
   3.4. Обґрунтування вибору технологічного стеку та мов програмування
4. ПРАКТИЧНА ЧАСТИНА
   4.1. Розробка серверної частини та проєктування бази даних
   4.2. Реалізація клієнтського інтерфейсу та механізмів Local-first
   4.3. Розробка користувацького інтерфейсу (UI/UX)
   4.4. Тестування системи та аналіз результатів

ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТКИ

Розділ	ПІП, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі	Черненко О. О.		
Інформаційний огляд	Черненко О. О.		
Теоретична частина	Черненко О. О.		
Практична реалізація	Черненко О. О.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ		
2. Вивчення методичних рекомендацій та стандартів та звіт керівнику		
3. Постановка задачі		
4. Інформаційний огляд джерел бібліотек та інтернету		
5. Теоретична частина		
6. Практична частина		
7. Закінчення оформлення		
8. Доповідь студента на кафедрі		
9. Доробка (за необхідністю), рецензування		

Дата видачі завдання «____»_____ 202_ р.

Здобувач вищої освіти Мяло Роман

Науковий керівник к. ф.-м. н., доц. Оксана ЧЕРНЕНКО

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на _____
(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 202_ р.

Секретар ЕК_____ _____
(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«____» _____ 202_ р.

Погоджено
Науковий керівник _____
к. ф.-м. н., доц. Оксана ЧЕРНЕНКО
«____» _____ 202_ р.

План

кваліфікаційної роботи ступеня бакалавра
зі спеціальності 122 Комп'ютерні науки
освітня програма 122 Комп'ютерні науки
Мяло Роман Миколайович
Прізвище, ім'я, по батькові

на тему «Розробка веб-додатку для спільного ведення нотаток»

ВСТУП
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ
РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД
- 2.1 Огляд існуючих систем управління знаннями та платформ для спільної роботи
- 2.2 Аналіз архітектурних підходів до синхронізації даних: OT проти CRDT
- 2.3 Дослідження специфіки User Experience (UX) у додатках реального часу
- 2.4 Обґрунтування необхідності та актуальності розробки

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА
- 3.1 Теоретичні засади побудови розподілених систем спільного редагування
- 3.2 Алгоритмічна база та структури даних для безконфліктної реплікації (YATA)
- 3.3 Проєктування архітектури системи (UML-діаграми)
- 3.4 Обґрунтування вибору технологічного стеку та мов програмування

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА
- 4.1 Розробка серверної частини та проєктування бази даних
- 4.2 Реалізація клієнтського інтерфейсу та механізмів Local-first
- 4.3 Розробка користувацького інтерфейсу (UI/UX)
- 4.4 Тестування системи та аналіз результатів

ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТКИ

Здобувач вищої освіти _____ Мяло Роман
«____» _____ 202_ р.

РЕФЕРАТ

Записка: 49 сторінки, в тому числі з яких: основна частина – 47 сторінок т, літературних джерел – 22 назв, рисунків – 5, UML-діаграм – 3.

LOCAL-FIRST, CRDT, YJS, NEXT.JS, NESTJS, СПІЛЬНЕ РЕДАГУВАННЯ, ОФЛАЙН-РЕЖИМ, WEBSOCKET.

Мета роботи – проєктування та програмна реалізація веб-додатка для спільного ведення нотаток у реальному часі, побудованого за архітектурою Local-first для забезпечення безперебійної роботи в офлайн-режимі.

Об’єкт розробки – програмне забезпечення у вигляді веб-застосунку для організації та спільного редагування текстової інформації.

Методи дослідження та інформаційне забезпечення – методи системного та порівняльного аналізу (для вибору алгоритмів синхронізації), теорія графів та структур даних (для реалізації CRDT), принципи Чистої архітектури (Clean Architecture) та патерни проєктування. Використано сучасні інструменти розробки: Next.js, NestJS, Prisma ORM, PostgreSQL, Yjs.

Результати дослідження. Розроблено веб-додаток, який вирішує проблему залежності від інтернет-з'єднання при роботі з документами. Завдяки впровадженню архітектури Local-first та використанню локальної бази даних IndexedDB, застосунок працює автономно. Реалізовано механізм спільного редагування на базі алгоритмів CRDT (Conflict-free Replicated Data Types), що забезпечує математично детерміноване злиття змін без конфліктів. Спроєктовано надійну серверну інфраструктуру для збереження бінарних зліпків документів та управління правами доступу.

Рекомендації щодо використання результатів дослідження. Розроблений веб-додаток рекомендовано використовувати як інструмент для особистого управління знаннями та командної роботи над текстами. Архітектурні рішення, застосовані в роботі, можуть слугувати основою для розробки інших відмовостійких розподілених систем реального часу.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	**Пояснення умовних позначень, скорочень, символів**
API	Application Programming Interface (Інтерфейс програмування застосунків) – набір визначень взаємодії між різними програмами.
CRDT	Conflict-free Replicated Data Type (Безконфліктні репліковані типи даних) – структури даних, що дозволяють розподіленим системам автоматично вирішувати конфлікти злиття.
OT	Operational Transformation (Операційна трансформація) – алгоритм синхронізації даних, що базується на центральному сервері.
DOM	Document Object Model (Об'єктна модель документа) – програмний інтерфейс для HTML та XML документів.
ORM	Object-Relational Mapping (Об'єктно-реляційне відображення) – технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов.
JWT	JSON Web Token – відкритий

	стандарт для створення токенів доступу, що базується на форматі JSON.
UX/UI	User Experience / User Interface (Користувацький досвід / Користувацький інтерфейс) – підходи до проєктування взаємодії користувача з програмою.
WebSocket	Протокол зв'язку, що забезпечує повнодуплексний зв'язок поверх одного TCP-з'єднання, ідеальний для додатків реального часу.

ВСТУП

Актуальність теми. У сучасному цифровому світі інструменти для спільної роботи над документами (SaaS-платформи) стали невід'ємною частиною як професійної, так і особистої діяльності. Більшість популярних систем управління знаннями (Notion, Google Docs) побудовані за класичною Cloud-first (хмарною) архітектурою. Їхнім критичним недоліком є повна залежність від стабільного інтернет-з'єднання. При втраті зв'язку такі додатки часто блокують можливість редагування або призводять до втрати даних при спробі подальшої синхронізації.

Вирішенням цієї проблеми є перехід до архітектурної парадигми «Local-first» [1], де первинним джерелом істини виступає локальна база даних на пристрої користувача. Це дозволяє працювати з документами автономно, а синхронізацію виконувати у фоновому режимі при відновленні з'єднання. Проте розробка таких систем вимагає вирішення складних алгоритмічних задач, пов'язаних із розв'язанням конфліктів редагування без залучення центрального сервера-арбітра [2]. Саме тому дослідження та імплементація безконфліктних алгоритмів синхронізації у веб-середовищі є актуальним завданням сучасної інженерії програмного забезпечення.

Метою роботи є проєктування та програмна реалізація веб-додатка для спільного ведення нотаток, що забезпечує автоматичне узгодження даних (data consistency) у розподіленому середовищі та підтримує повноцінний офлайн-режим завдяки архітектурі Local-first.

Об’єктом розробки є програмне забезпечення у вигляді веб-застосунку для організації та спільного редагування текстової інформації.

Предметом розробки є методи, алгоритми та архітектурні патерни безконфліктної синхронізації даних у веб-додатках реального часу.

Методи дослідження. У роботі використано: методи системного аналізу - для дослідження предметної області та існуючих аналогів; порівняльний аналіз

- для вибору алгоритму синхронізації (порівняння Operational Transformation та CRDT); методи об’єктно-орієнтованого проєктування та графічного моделювання (UML) - для побудови архітектури системи; експериментальний метод - для перевірки коректності злиття даних при одночасному редагуванні.

Новизна даної роботи полягає у комплексній інтеграції парадигми Local-first із сучасним стеком веб-технологій (Next.js, NestJS) та алгоритмами CRDT (Conflict-free Replicated Data Types). На відміну від традиційних систем, де вирішення конфліктів покладається на сервер, у розробленій системі імплементовано математично детерміноване злиття змін безпосередньо на клієнті (алгоритм YATA). Крім того, запропоновано гібридну модель збереження даних, де метадані та права доступу керуються реляційною базою даних через Prisma ORM, а історія редагувань зберігається у вигляді бінарних зліпків (Snapshots), що суттєво оптимізує навантаження на серверну інфраструктуру.

Практична значимість роботи полягає у створенні працездатного та відмовостійкого програмного продукту, який може бути використаний широким колом користувачів для управління знаннями та командної роботи. Розроблена архітектура та програмний код можуть слугувати базою для створення інших розподілених систем, що вимагають високої доступності та роботи в умовах нестабільного зв'язку.

Пояснювальна записка до кваліфікаційної роботи складається з чотирьох розділів:

- перший розділ містить постановку задачі розробки;
- другий розділ присвячено інформаційному аналізу: досліджено існуючі системи управління знаннями, виявлено їхні архітектурні недоліки та обґрунтовано потребу у створенні Local-first рішення;
- третій розділ містить теоретичне обґрунтування: описано математичну базу алгоритмів CRDT, наведено UML-діаграми архітектури системи та обґрунтовано вибір технологічного стеку;

– четвертий розділ охоплює опис практичної реалізації програмного продукту, включаючи розробку серверної та клієнтської частин, а також результати тестування системи.

Обсяг пояснювальної записки до кваліфікаційної роботи складає: 49 сторінки, в тому числі з яких: основна частина – 47 сторінок, літературних джерел – 22 назв, рисунків – 5, UML-діаграми – 3.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

Головною задачею кваліфікаційної роботи є проєктування та розробка веб-додатка для спільного ведення нотаток, що функціонує за архітектурною парадигмою «Local-first» [1]. Створювана система повинна забезпечувати безперебійну роботу користувача з текстовими документами в умовах відсутності або нестабільності інтернет-з'єднання. При цьому критично важливою вимогою є гарантування консистентності даних під час подальшої фонової синхронізації з іншими учасниками мережі [3].

Для досягнення поставленої мети необхідно вирішити комплекс взаємопов'язаних завдань. Насамперед передбачається проведення глибокого аналізу предметної області та існуючих рішень, таких як Notion, Google Docs та Obsidian. Це дозволить виявити вразливості класичної клієнт-серверної архітектури при роботі в офлайн-режимі та обґрунтувати потребу у створенні децентралізованого рішення.

Наступним кроком є дослідження алгоритмів синхронізації даних. Необхідно здійснити порівняльний аналіз методів операційної трансформації (Operational Transformation) [4] та безконфліктних реплікованих типів даних (Conflict-free Replicated Data Types) [5]. На основі цього аналізу планується обґрунтувати вибір алгоритму YATA (Yet Another Transformation Approach) [6], який забезпечує злиття змін без обов'язкової участі центрального сервера-арбітра.

Значна увага в роботі має бути приділена проєктуванню архітектури системи. Завдання полягає у розробці логічної та фізичної структури веб-додатка з використанням методології об'єктно-орієнтованого проєктування. Для візуалізації процесів авторизації, локального збереження та фонової синхронізації даних необхідно побудувати відповідні UML-діаграми, зокрема діаграми прецедентів, послідовності та діяльності [7].

Практична реалізація вимагає розробки надійної серверної інфраструктури. Завданням цього етапу є проєктування бекенду на базі фреймворку NestJS із суворим дотриманням принципів Чистої архітектури (Clean Architecture) [8]. Крім того, необхідно розробити схему реляційної бази даних PostgreSQL за допомогою Prisma ORM. База даних повинна забезпечувати збереження метаданих, управління правами доступу на основі ролей (RBAC) та зберігання історії редагувань у вигляді бінарних зліпків. Для забезпечення обміну даними між клієнтами передбачається налаштування WebSocket-сервера.

Паралельно з розробкою серверної частини ставиться завдання реалізації клієнтського застосунку. Інтерфейс користувача має бути побудований на базі фреймворку Next.js [9] та інтегрований із текстовим редактором Tiptap. Ключовим завданням на цьому етапі є впровадження механізму локального збереження даних у браузері за допомогою IndexedDB [10], що виступатиме первинним джерелом істини для клієнта. Для забезпечення повноцінного досвіду спільної роботи необхідно також реалізувати протокол Awareness [11], який відповідатиме за візуалізацію присутності інших користувачів та відображення їхніх віддалених курсорів у реальному часі.

Завершальним етапом роботи є тестування створеної системи. Завдання полягає у перевірці працездатності додатка в умовах імітації розриву мережевого з'єднання та підтвердженні коректності вирішення конфліктів при одночасному редагуванні одного документа кількома користувачами в офлайн-режимі з подальшим відновленням зв'язку.

Виконання зазначених завдань дозволить створити повноцінний продукт, який вирішує проблему залежності від постійного підключення до мережі, забезпечуючи при цьому високу швидкість відгуку інтерфейсу та надійність збереження користувацьких даних.

РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1 Огляд існуючих систем управління знаннями та платформ для спільної роботи

Процес проєктування сучасної веб-системи для спільного ведення нотаток вимагає ретельного аналізу ринкових аналогів. Це дозволяє не лише виявити стандарти індустрії, до яких звикли користувачі, але й знайти архітектурні слабкі місця існуючих рішень. Для детального аналізу було обрано три платформи, що втілюють кардинально різні філософії роботи з інформацією: Notion (блокова структура та централізовані бази даних), Obsidian (мережева структура та локальні файли) та Google Keep (потокова структура та швидка синхронізація).

Платформа Notion позиціонується як універсальний робочий простір [14]. Її архітектура базується на концепції блоків, де кожен елемент контенту (абзац, заголовок, зображення або елемент списку) є окремим об'єктом у базі даних (див. рисунок 2.1).

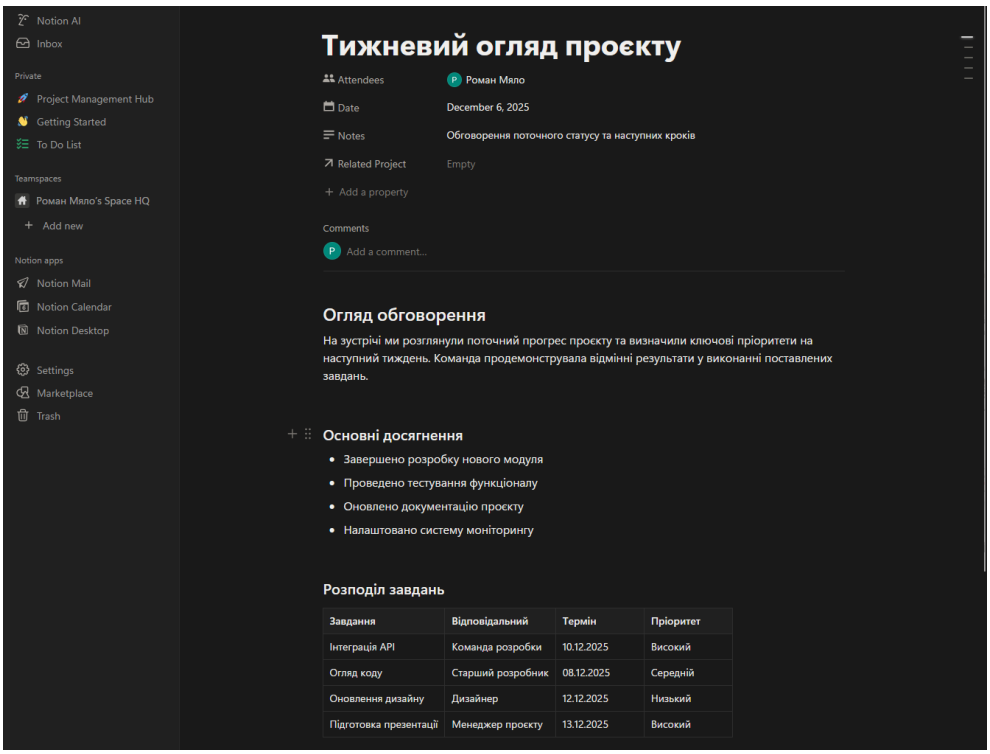

Рисунок 2.1 – Інтерфейс робочого простору Notion з блоковою структурою

Такий підхід забезпечує високу гнучкість: користувач може перетворити будь-який текст на сторінку, завдання або рядок бази даних. Крім того, Notion має потужний інструмент для створення зв’язків між сторінками, що дозволяє будувати складні системи управління знаннями. Платформа також підтримує одночасне редагування та коментування конкретних блоків.

Проте, в контексті проєктування відмовостійких систем, Notion має суттєві архітектурні недоліки. Головним з них є «важкість» клієнтського застосунку. Оскільки кожна сторінка складається з тисяч окремих об'єктів-блоків, додаток потребує значних ресурсів браузера та стабільного інтернет-з'єднання. Офлайн-режим у Notion реалізований лише частково (через механізм кешування), і при відсутності мережі користувач часто втрачає доступ до вкладених сторінок. Крім того, існує проблема прив'язки до екосистеми: експорт складної структури Notion у звичайні формати (наприклад, Markdown) часто призводить до втрати метаданих або форматування.

Протилежний підхід пропонує система Obsidian, яка є яскравим представником філософії «Local-first» [15]. Ця програма орієнтована на індивідуальних дослідників та розробників, які цінують швидкість, приватність та контроль над власними даними (див. рисунок 2.2).

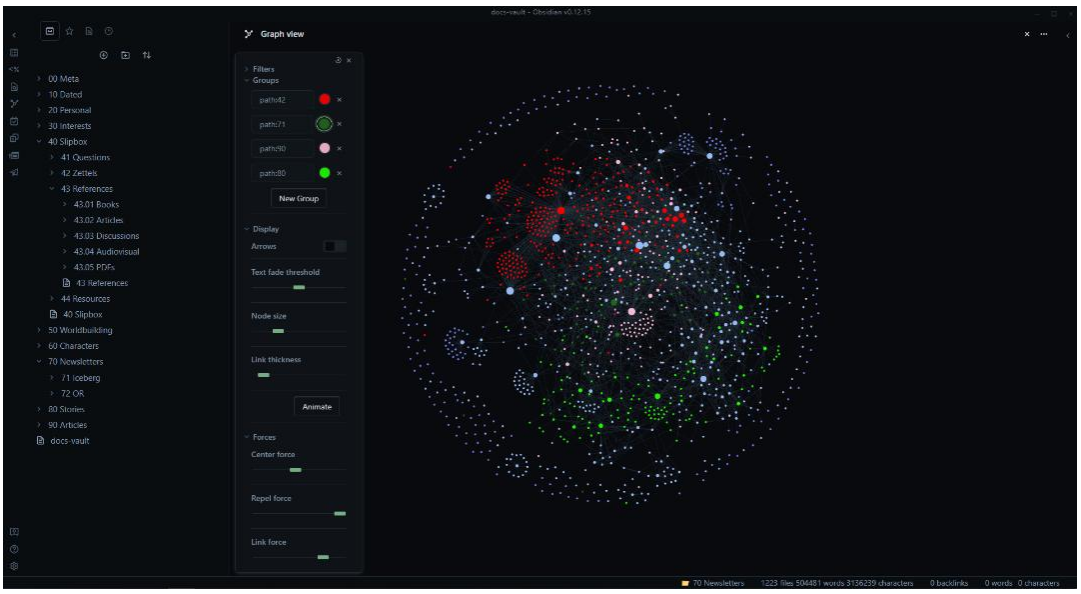

Рисунок 2.2 – Інтерфейс Obsidian та візуалізація графу зв'язків

Основою Obsidian є зберігання інформації у вигляді звичайних текстових файлів формату Markdown безпосередньо на пристрої користувача. Це забезпечує абсолютну швидкість роботи (відкриття та пошук нотаток відбувається миттєво) та незалежність від серверів компанії-розробника. Унікальною особливістю системи є візуалізація зв'язків між нотатками у вигляді графу знань, що дозволяє знаходити неочевидні асоціації.

Слабкою стороною Obsidian є організація спільної роботи. Оскільки файли зберігаються локально, для синхронізації між користувачами доводиться використовувати сторонні хмарні сервіси або Git-репозиторії. Такий підхід унеможливлює справжнє спільне редагування в реальному часі, оскільки виникають конфлікти версій файлів на рівні файлової системи, що створює високий поріг входження для нетехнічних користувачів.

Google Keep є прикладом максимально спрощеного підходу до ведення нотаток, який імітує стікери на дошці (див. рисунок 2.3). Цей сервіс оптимізований для миттєвого запису короткої інформації.

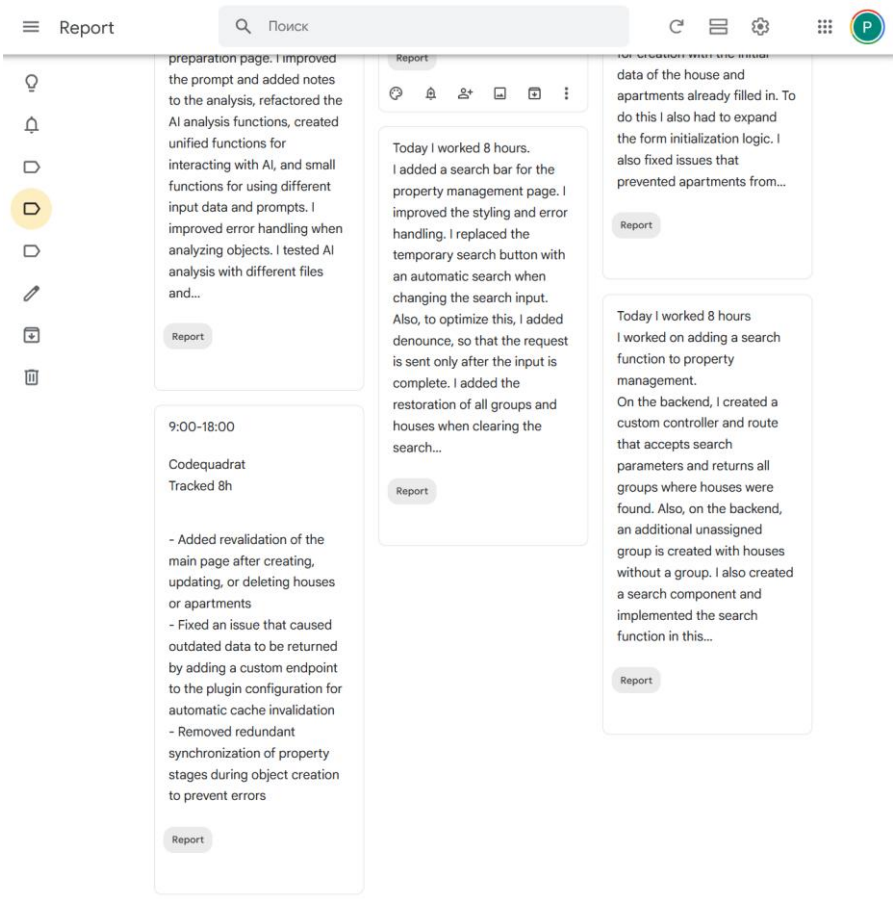

Рисунок 2.3 – Інтерфейс Google Keep у вигляді дошки стікерів

З технічної точки зору, Google Keep відзначається ефективним алгоритмом синхронізації, який забезпечує майже миттєве оновлення даних на всіх пристроях, навіть при слабкому мобільному інтернеті. Перевагою системи є швидкість синхронізації: використання алгоритмів передачі дельт (змін) дозволяє бачити зміни на іншому пристрої менш ніж за секунду.

Однак, Google Keep має суттєві обмеження щодо структурування інформації. Відсутність складної ієрархії папок, підтримки форматування тексту (Markdown або Rich Text) та неможливість створення вкладених документів обмежує сферу використання системи лише простими списками та нагадуваннями.

Проведений аналіз виявив суттєву проблему на ринку систем управління знаннями. Користувачі часто змушені обирати між зручністю синхронізації та спільної роботи (Notion, Google Keep) та надійністю, швидкістю і якістю структурування інформації (Obsidian). Розроблений у даній роботі веб-додаток вирішує цю дилему, поєднавши швидкість локальних додатків, Markdown-

розмітку та можливості безконфліктного одночасного редагування без блокування інтерфейсу.

2.2 Аналіз архітектурних підходів до синхронізації даних: переваги та недоліки

Забезпечення спільної роботи над документами в режимі реального часу є однією з найскладніших задач у сучасній веб-розробці. На відміну від звичайних веб-сайтів, де користувач просто отримує статичні дані з сервера, системи типу Knowledge Management Systems вимагають миттєвого узгодження змін, зроблених різними користувачами на різних пристроях, часто в умовах нестабільного інтернет-з'єднання. Існує два різних архітектурних підходи до вирішення цієї проблеми: централізований, на базі операційної трансформації (Operational Transformation), та децентралізований, на базі безконфліктних реплікованих типів даних (Conflict-free Replicated Data Types). Вибір конкретного підходу визначає не лише технічний стек проєкту, але й користувацький досвід (UX), зокрема можливість повноцінної роботи офлайн.

Операційна трансформація (Operational Transformation - OT). Цей підхід історично є стандартом для систем спільного редагування і використовується в таких продуктах, як Google Docs та ранніх версіях Etherpad [4]. Суть методу полягає в тому, що кожна дія користувача (вставка символу, видалення рядка) перетворюється на математичну операцію, яка відправляється на центральний сервер. Сервер виступає «єдиним джерелом істини»: він отримує операції від усіх клієнтів, вибудовує їх у чергу, трансформує (змінює індекси вставки/видалення) для вирішення конфліктів і розсилає затверджений результат назад усім учасникам.

Перевагами підходу OT є:

- Передбачуваність: Наявність центрального сервера гарантує, що всі користувачі зрештою побачать документ однаковим.

- Економія ресурсів клієнта: Основні обчислення та вирішення конфліктів відбуваються на сервері, що дозволяє клієнтському додатку бути відносно «легким».

Проте, для побудови систем за парадигмою Local-first, підхід OT має критичні недоліки:

- Залежність від мережі: Оскільки логіка вирішення конфліктів знаходиться виключно на сервері, робота в режимі офлайн є вкрай обмеженою. Без з'єднання з сервером клієнт не може гарантувати правильність злиття змін, що призводить до блокування інтерфейсу або втрати даних при тривалому розриві зв'язку.
- Складність масштабування: Сервер стає «вузьким місцем», оскільки повинен обробляти та трансформувати кожну літеру від кожного користувача в реальному часі. Обчислювальна складність трансформації сильнозростає при збільшенні кількості учасників.

Безконфліктні репліковані типи даних (Conflict-free Replicated Data Types - CRDT). Це сучасний математичний підхід, який набув популярності з розвитком концепції децентралізованих систем [5]. Він використовується в таких передових системах, як Figma (для синхронізації певних елементів), Apple Notes та сучасних колаборативних плагінах. В архітектурі CRDT усі репліки даних (копії документа на пристроях користувачів) є рівноправними. Злиття змін відбувається математично передбачуваним шляхом без необхідності центрального узгодження. Це означає, що конфлікти неможливі за дизайном - система завжди може автоматично об'єднати дві версії документа, спираючись на властивість комутативності операцій.

Перевагами підходу CRDT є:

- Повноцінний Offline-режим: Користувач може редагувати документ без інтернету як завгодно довго. При появі мережі локальні зміни автоматично та безконфліктно синхронізуються з іншими клієнтами.

- Децентралізація: Архітектура не вимагає потужного центрального сервера для складної логіки; сервер може виконувати роль простого ретранслятора повідомлень.
- Низька затримка: Зміни застосовуються локально миттєво, не чекаючи підтвердження від сервера, що дозволяє реалізувати «оптимістичний інтерфейс».

Основним недоліком CRDT історично вважався оверхед пам'яті. Для забезпечення можливості автоматичного злиття, історія змін (метадані) зберігається разом із документом, що може збільшувати обсяг даних. Однак сучасні реалізації алгоритмів, зокрема YATA (Yet Another Transformation Approach) у бібліотеці Yjs [6], ефективно вирішують цю проблему через механізми компресії та збирання сміття.

Аналіз показує, що для розробки сучасного веб-додатка нотаток, який позиціонується як інструмент для персональної продуктивності з можливістю командної роботи, архітектура на базі CRDT є значно перспективнішою. Вона дозволяє реалізувати ключову вимогу - надійну роботу в офлайн-режимі, що є слабким місцем OT-систем. Використання гібридної моделі (CRDT на клієнті + легкий WebSocket сервер для ретрансляції) дозволить поєднати швидкість локальних додатків із зручністю хмарної синхронізації.

2.3 Дослідження специфіки User Experience (UX) у додатках реального часу

Проєктування інтерфейсів для систем спільного редагування в реальному часі вимагає вирішення специфічних проблем користувацького досвіду (UX), які не зустрічаються у традиційних веб-додатках. Фундаментальною технічною проблемою розподілених систем є мережева затримка. Якщо інтерфейс текстового редактора чекатиме підтвердження від сервера на кожне натискання клавіші, процес набору тексту стане переривчастим і непридатним для комфортної роботи.

Патерн «Optimistic UI» та компенсація латентності. Для вирішення проблеми мережевих затримок у сучасних веб-додатках використовується архітектурний патерн Optimistic UI. Суть підходу полягає в тому, що система миттєво відображає дію користувача (наприклад, введення символу або створення нового блоку) на екрані, оптимістично припускаючи, що сервер успішно її обробить. Сама ж синхронізація даних відбувається асинхронно у фоновому режимі.

Такий підхід створює у користувача відчуття абсолютної швидкодії, що є критично важливим для текстових редакторів. Проте реалізація Optimistic UI накладає на систему обов'язок коректно обробляти «відкат» змін у рідкісних випадках помилки сервера або конфлікту версій, повідомляючи про це користувача ненав'язливо, без руйнування поточного контексту роботи.

Концепція Workspace Awareness (Обізнаність про робочий простір). Згідно з дослідженнями у сфері комп'ютерно-підтримуваної спільної роботи (CSCW), ефективна взаємодія неможлива без розуміння контексту дій інших учасників [11]. У фізичному світі люди бачать рухи та погляди колег. У цифровому середовищі ці невербальні сигнали необхідно емулювати візуальними засобами. Ключовими елементами Workspace Awareness для текстових редакторів є відповіді на три запитання: «Хто?», «Де?» та «Що робить?».

- Ідентифікація учасників (Хто?): Візуалізація списку активних користувачів, які наразі переглядають або редагують документ. Зазвичай це реалізується через динамічний стек аватарів у верхній панелі інтерфейсу.
- Локалізація уваги (Де?): Відображення позиції фокусу уваги колег. Стандартом індустрії стали «віддалені курсори» - кольорові маркери з іменами користувачів, що переміщуються документом у реальному часі.

- Візуалізація дій (Що робить?): Підсвічування виділеного тексту або блоків, над якими наразі працює інший учасник, для запобігання дублюванню роботи.

При проєктуванні механізмів Awareness виникає проблема управління когнітивним навантаженням. Інтенсивна візуалізація дій інших користувачів несе ризик створення інформаційного шуму. Наприклад, якщо десять користувачів одночасно редагують документ, хаотичний рух курсорів може відволікати від власної роботи. Тому при розробці UX важливо дотримуватись балансу:

- рух курсорів має бути плавним (з використанням математичної інтерполяції), щоб уникнути ефекту візуальних «стрибків»;
- інформація про присутність має бути контекстною (наприклад, відображення повного імені користувача лише при наведенні на його курсор або під час активного друку);
- використання кольорового кодування дозволяє підсвідомо асоціювати зміни в тексті з конкретним учасником без необхідності вчитуватись у текстові підписи.

Візуалізація стану синхронізації та довіра. У системах, що підтримують офлайн-режим, критично важливим аспектом UX є прозоре інформування користувача про статус збереження даних. Користувач повинен чітко розуміти, чи його дані знаходяться лише на локальному пристрої, чи вже успішно збережені в хмарі. Відсутність зрозумілого зворотного зв'язку щодо статусу синхронізації призводить до зниження довіри до системи та провокує користувача на зайві дії, такі як ручне оновлення сторінки або створення резервних копій.

Таким чином, проєктування інтерфейсу для розроблюваного веб-додатка має базуватися на принципах Optimistic UI для забезпечення плавності роботи. Необхідно реалізувати механізм візуалізації присутності, оскільки це є ключовим фактором, що відрізняє інструмент для колаборації від звичайного локального редактора. Також слід передбачити чітку систему індикації станів

з'єднання (Online / Offline / Syncing), щоб користувач почувався впевнено, працюючи з даними в умовах нестабільної мережі.

2.4 Обґрунтування необхідності та актуальності розробки

У сучасних умовах стрімкого розвитку інформаційних технологій та цифровізації робочих процесів, постійне зростання обсягів інформації вимагає від користувачів використання ефективних інструментів для її структурування. Багатозадачність та необхідність швидкого доступу до робочих матеріалів зумовлюють високий попит на цифрові системи управління знаннями. Однак, як показав проведений інформаційний огляд, існуючі на ринку рішення не завжди здатні задовольнити комплексні потреби сучасного користувача, особливо в контексті надійності доступу до даних.

Хоча на ринку представлено безліч цифрових інструментів для ведення нотаток - від простих списків завдань до комплексних корпоративних платформ - вони мають суттєві архітектурні компроміси. Системи, орієнтовані на глибоке структурування та локальне зберігання, часто не мають вбудованих механізмів для спільної роботи в реальному часі. З іншого боку, популярні хмарні платформи для колаборації виявляються абсолютно безпорадними при втраті інтернет-з'єднання, що ставить під загрозу безперервність робочого процесу та збереження даних.

У результаті користувачі змушені йти на компроміс: або відмовлятися від зручності спільного редагування заради автономності, або повністю покладатися на стабільність серверів сторонніх компаній. Крім того, перенасиченість функціями у багатьох сучасних платформах призводить до ускладнення інтерфейсу, що знижує концентрацію уваги та ефективність роботи.

Саме тому виникає потреба у створенні нового веб-додатка для ведення нотаток, який би вирішував ці архітектурні та функціональні проблеми.

Основна ідея полягає у розробці інструменту, що поєднує в собі найкращі практики існуючих рішень, але позбавлений їхніх критичних недоліків.

Запропонований у цій кваліфікаційній роботі веб-додаток вирізняється такими ключовими особливостями:

використання архітектури, орієнтованої на локальне збереження даних, що гарантує миттєвий доступ до нотаток та можливість повноцінного редагування в офлайн-режимі;

впровадження математичних алгоритмів безконфліктної реплікації, що дозволяє організувати спільне редагування документів без ризику втрати даних при злитті версій;

мінімалістичний та інтуїтивно зрозумілий інтерфейс, який не перевантажує користувача зайвим функціоналом і дозволяє сфокусуватися безпосередньо на контенті;

використання популярних стандартів форматування тексту, що запобігає прив'язці користувача до закритої екосистеми та полегшує експорт даних.

Зростаюча потреба у надійних, швидких та незалежних від якості інтернет-з'єднання інструментах свідчить про високу актуальність створення такого веб-додатка. Розробка системи, яка здатна забезпечити безшовний перехід між автономною роботою та хмарною синхронізацією, відповідає сучасним тенденціям розвитку програмного забезпечення.

Отже, проєктування та реалізація веб-додатка для спільного ведення нотаток на базі децентралізованих алгоритмів синхронізації є не лише актуальним інженерним завданням, але й практично значущим рішенням для підвищення особистої та командної продуктивності.

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1 Теоретичні засади побудови розподілених систем спільного редагування та моделі узгодженості даних

Сучасна парадигма розробки інтерактивних веб-додатків поступово зміщується від класичної централізованої моделі до децентралізованих підходів. Одним із таких підходів є концепція програмного забезпечення, орієнтованого на локальне збереження даних. Основна ідея полягає в тому, що клієнтський пристрій володіє повною копією даних, а мережа використовується лише для фонової синхронізації. Це дозволяє забезпечити роботу додатка без очікування відповіді сервера, що важливо для текстових редакторів.

При проєктуванні будь-якої розподіленої системи розробник стикається з фундаментальними обмеженнями, які описуються теоремою Брюера, також відомою як CAP-теорема [3]. Згідно з цією теоремою, у розподіленій обчислювальній системі неможливо одночасно забезпечити три наступні властивості:

- Узгодженість - кожне читання отримує найостанніший запис або помилку.
- Доступність - кожен запит отримує успішну відповідь, без гарантії, що вона містить найостанніший запис.
- Стійкість до розділення - система продовжує працювати, незважаючи на втрату або затримку довільної кількості повідомлень між вузлами мережі.

Оскільки для мобільних та веб-додатків стійкість до розривів зв'язку є обов'язковою умовою (властивість P), архітектурний вибір звужується до дилеми між CP-системами (узгодженість та стійкість до розділення) та AP-системами (доступність та стійкість до розділення) [12].

Для текстового редактора, де користувач повинен мати змогу вводити текст у будь-який момент часу, блокування інтерфейсу заради очікування синхронізації є неприпустимим. Тому для розроблюваного додатка обрано AP-модель. Вона гарантує, що система завжди приймає зміни користувача, навіть у режимі офлайн, забезпечуючи високу доступність. Проте вибір AP-моделі вимагає вирішення проблеми узгодженості даних після відновлення зв'язку.

Еволюція моделей узгодженості в розподілених базах даних пройшла шлях від узгодженості у кінцевому рахунку до строгої узгодженості у кінцевому рахунку.

Класична модель узгодженості у кінцевому рахунку є базовою для багатьох сучасних NoSQL систем. Вона гарантує, що за відсутності нових оновлень усі вузли системи з часом прийдуть до однакового стану. Проте ця модель має суттєвий недолік при спільному редагуванні текстів: можливі конфлікти часто вирішуються за принципом «останній запис перемагає». У контексті текстового документа це може призвести до безповоротної втрати абзаців або речень, написаних одним із користувачів під час відсутності мережі.

Для вирішення цієї проблеми у проєкті використано математичну модель строгої узгодженості у кінцевому рахунку (SEC) [5]. Вона надає сильніші гарантії: будь-які два вузли, які отримали однаковий набір оновлень (незалежно від порядку їх надходження), гарантовано перебуватимуть в ідентичному стані. Ця властивість досягається завдяки використанню спеціальних структур даних, які забезпечують передбачуване злиття змін без виникнення конфліктів.

Порівняльний аналіз моделей узгодженості для розподілених систем наведено у таблиці 3.1.

Таблиця 3.1 – Порівняльний аналіз моделей узгодженості

Характеристика	Сувора узгодженість (Strong Consistency)	Узгодженість у кінцевому рахунку (Eventual Consistency)	Строга узгодженість у кінцевому рахунку (SEC)
Гарантія читання	Завжди актуальні дані	Можливі застарілі дані	Гарантія конвергенції (збіжності) станів
Затримка запису	Висока (блокування до підтвердження сервером)	Низька (асинхронна реплікація)	Низька (миттєве локальне застосування операцій)
Вирішення конфліктів	Запобігає конфліктам через механізми блокування	Часто використовує підхід Last-Write-Wins (можлива втрата даних)	Математично детерміноване злиття (без втрати даних)
Сфера застосування	Банківські та фінансові системи	Кешування, соціальні мережі, аналітика	Системи спільного редагування, децентралізовані додатки

Отже, теоретичне обґрунтування підтверджує, що для забезпечення повноцінного офлайн-режиму та спільного редагування без втрати даних, архітектура веб-додатка повинна базуватися на моделі Strong Eventual Consistency. Практична реалізація цієї моделі вимагає застосування специфічних алгоритмів, які будуть розглянуті у наступному підрозділі.

3.2 Алгоритмічна база та структури даних для безконфліктної реплікації

Реалізація моделі строгої узгодженості у кінцевому рахунку вимагає використання спеціалізованих структур даних, здатних поглинати конкурентні зміни без виникнення конфліктів. У розробленому проєкті використовується алгоритм YATA (Yet Another Transformation Approach), який лежить в основі бібліотеки Yjs [6]. Цей алгоритм належить до сімейства безконфліктних реплікованих типів даних (CRDT) і оптимізований спеціально для роботи з лінійними послідовностями, такими як текстові документи.

Внутрішня структура даних. Використання звичайного масиву символів є неефективним для розподілених систем. Якщо два користувачі одночасно вставляють символ на початку документа, індекси всіх наступних елементів зміщуються. Це робить некоректними операції інших користувачів, що базуються на старих абсолютних позиціях.

Алгоритм YATA вирішує цю проблему, переходячи від абсолютної позиції до відносної. Документ представляється як двозв'язний список, де кожен елемент (символ або об'єкт) є окремою структурою, що називається Item. Кожен Item унікально ідентифікується складеним ключем, який містить:

- clientID: унікальний ідентифікатор користувача (або сесії), який створив цей елемент.
- clock: логічний годинник (лічильник Лампорта), що монотонно зростає з кожною новою операцією даного клієнта.

Місце вставки нового елемента визначається не числовим індексом, а посиланням на сусідні елементи, які вже існують у документі: origin (елемент зліва) та originRight (елемент справа). Завдяки цьому «намір» вставки залишається незмінним навіть при зміні навколишнього контексту іншими користувачами.

Механізм обробки видалень. У системах CRDT видалення елемента не призводить до його фізичного знищення з пам'яті. Це пов'язано з тим, що інші

конкурентні операції (які ще знаходяться в процесі передачі мережею) можуть посилатися на цей елемент як на свій origin. Замість фізичного видалення використовується механізм «надгробків»: елемент позначається як видалений за допомогою булевого прапорця (deleted = true). Такий елемент ігнорується при рендерингу тексту в інтерфейсі, але залишається у структурі двозв'язного списку для забезпечення правильної навігації та вставки майбутніх елементів.

Для запобігання неконтрольованому зростанню історії змін та витокам пам'яті, алгоритм використовує оптимізацію Garbage Collection. Система періодично аналізує граф залежностей і фізично видаляє ті елементи-надгробки, які гарантовано більше не будуть потрібні для вирішення майбутніх конфліктів.

Алгоритм вирішення конфліктів. Найскладнішим сценарієм у розподіленому редагуванні є одночасна вставка різними користувачами тексту в одну й ту саму позицію (тобто нові елементи мають однаковий origin). Для вирішення таких колізій алгоритм YATA використовує строгі математичні правила, що гарантують повний порядок елементів на всіх вузлах мережі:

- Заборона перетину зв'язків: Лінії зв'язку нового елемента з його origin не повинні перетинатися з лініями конкурентних елементів. Це правило зберігає логічну цілісність вставлених блоків тексту.
- Транзитивність: Порядок елементів має бути математично узгодженим. Якщо елемент A знаходиться перед B, а B перед C, то A гарантовано знаходиться перед C.
- Детермінізм за ідентифікатором: Якщо попередні правила не дають однозначної відповіді щодо порядку вставки, пріоритет визначається лексикографічним порівнянням clientID користувачів. Це гарантує, що всі вузли приймуть однакове рішення без звернення до центрального сервера.

Оптимізація пам'яті. Зберігання кожного введеного символу як окремого об'єкта Item з метаданими (ID, посилання на сусідів) створює значний оверхед пам'яті. Для зменшення накладних витрат застосовується техніка Struct

Merging. Послідовні символи, введені одним користувачем без перерв, об'єднуються в єдиний текстовий блок. Якщо інший користувач вносить зміни всередину цього блоку, він автоматично розщеплюється на менші частини. Ця оптимізація дозволяє наблизити продуктивність CRDT-структур до нативних рядкових операцій та суттєво зменшити обсяг даних, що передаються мережею.

Таким чином, математичний апарат та структури даних алгоритму YATA забезпечують надійний фундамент для побудови децентралізованих систем спільного редагування.

3.3 Проєктування архітектури системи (UML-діаграми)

Для забезпечення чіткого розуміння логіки роботи веб-додатка, взаємодії його компонентів та ролей користувачів, архітектура системи була спроєктована з використанням уніфікованої мови моделювання UML [7]. Моделювання проведено на трьох рівнях: функціональних вимог, часової взаємодії компонентів та алгоритмічної логіки процесів.

Діаграма прецедентів ілюструє функціональні можливості системи з точки зору кінцевого користувача. У розроблюваній системі виділено одну основну роль - «Користувач», який взаємодіє з системою через веб-інтерфейс.

Основні сценарії використання включають:

- управління обліковим записом (реєстрація, авторизація, відновлення сесії);
- управління документами (створення, редагування, видалення нотаток);
- спільна робота (генерація посилання для доступу, надання прав на читання або редагування);
- автономна робота (редагування документа в офлайн-режимі з подальшою автоматичною синхронізацією).

Діаграма прецедентів розробленого веб-додатка наведена на рисунку 3.1.

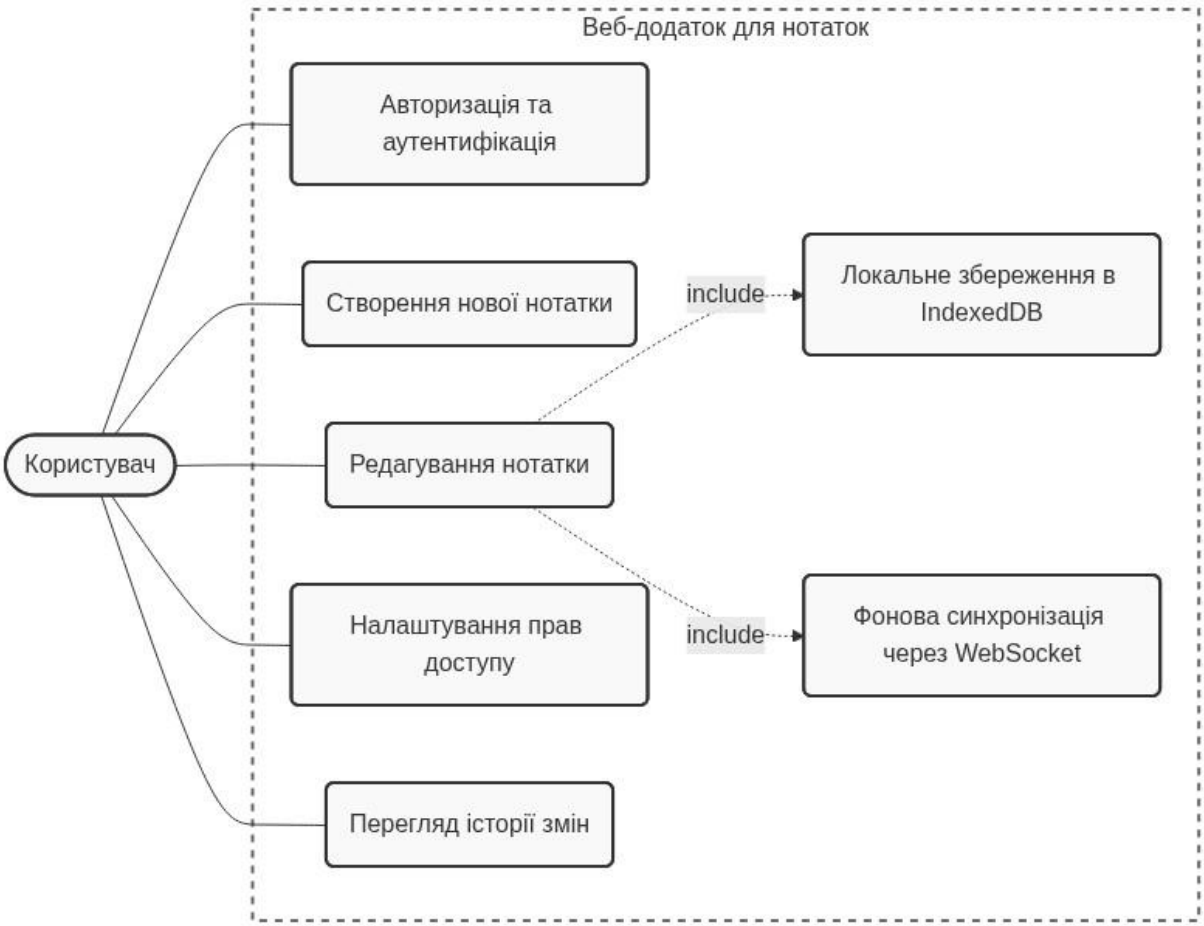

Рисунок 3.1 – Діаграма прецедентів веб-додатка

Діаграма послідовності потрібна для розуміння процесу синхронізації даних у реальному часі. Вона показує часові особливості передачі повідомлень між об'єктами системи під час спільного редагування документа.

У процесі беруть участь чотири основні компоненти: Клієнт 1, Клієнт 2, WebSocket-сервер та База даних.

Процес синхронізації відбувається за наступним сценарієм:

- Клієнт 1 вносить зміни до документа. Зміни миттєво застосовуються до локальної моделі та зберігаються в IndexedDB.
- Клієнт 1 генерує бінарне оновлення і відправляє його на WebSocket-сервер.
- Сервер отримує оновлення, застосовує його до своєї копії документа в оперативній пам'яті та ретранслює це оновлення Клієнту 2.
- Клієнт 2 отримує оновлення, алгоритм YATA безконфліктно зливає його з локальним станом, після чого інтерфейс Клієнта 2 оновлюється.

- Асинхронно, з використанням механізму дебаунсингу, сервер серіалізує поточний стан документа і зберігає бінарний зліпок у базу даних для забезпечення персистентності.

Діаграма послідовності процесу синхронізації наведена на рисунку 3.2.

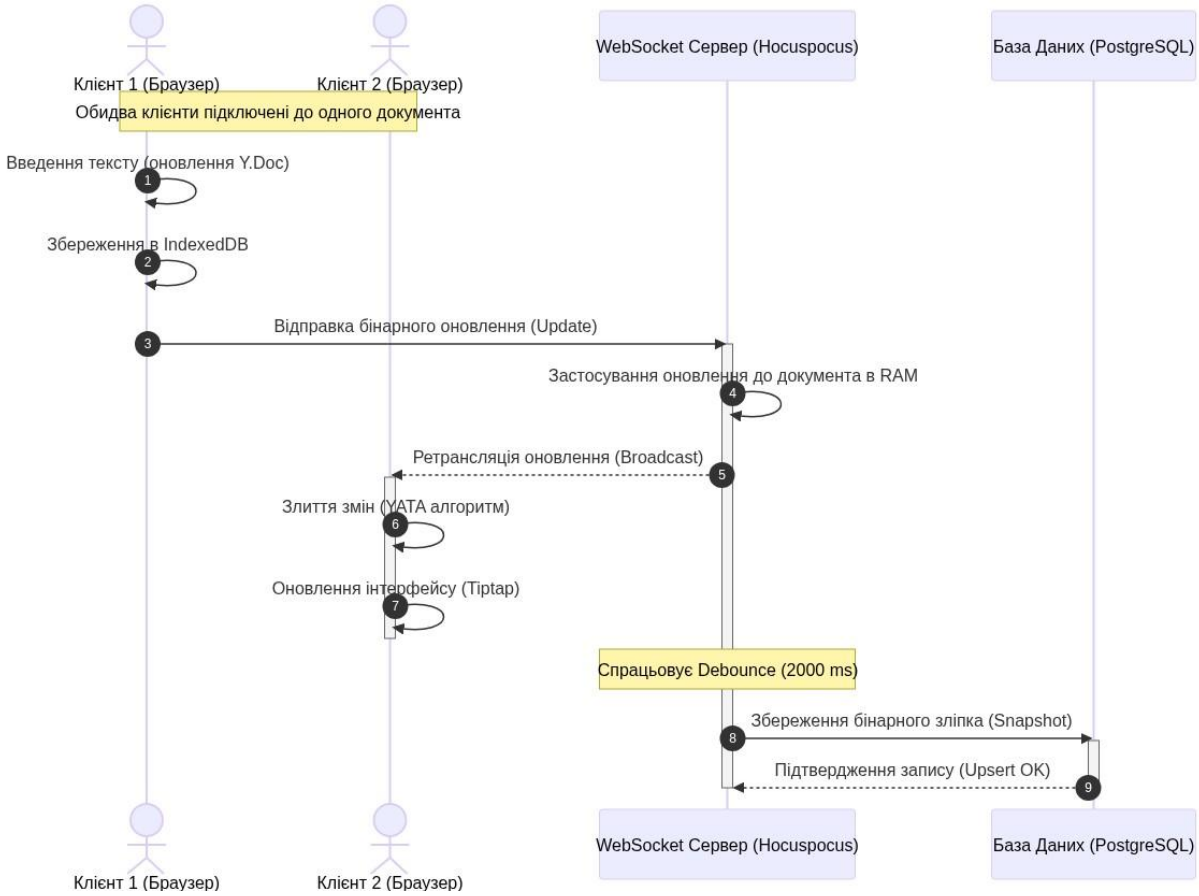

Рисунок 3.2 – Діаграма послідовності процесу синхронізації даних

Діаграма діяльності візуалізує алгоритмічну логіку роботи системи в умовах втрати та відновлення інтернет-з'єднання.

Алгоритм обробки офлайн-стану включає наступні кроки:

- Система фіксує втрату з'єднання з WebSocket-сервером.
- Інтерфейс переходить у режим «Offline», інформуючи користувача.
- Усі подальші зміни, внесені користувачем, серіалізуються та зберігаються виключно в локальній базі IndexedDB.
- Система періодично намагається відновити з'єднання.

- Після успішного відновлення зв'язку ініціюється процес «Handshake»: клієнт відправляє серверу свій вектор стану.
- Сервер обчислює різницю між станом клієнта та актуальним станом на сервері, після чого сторони обмінюються відсутніми пакетами даних.
- Локальна модель оновлюється, інтерфейс переходить у режим «Synced».

Діаграма діяльності процесу відновлення синхронізації наведена на рисунку 3.3.

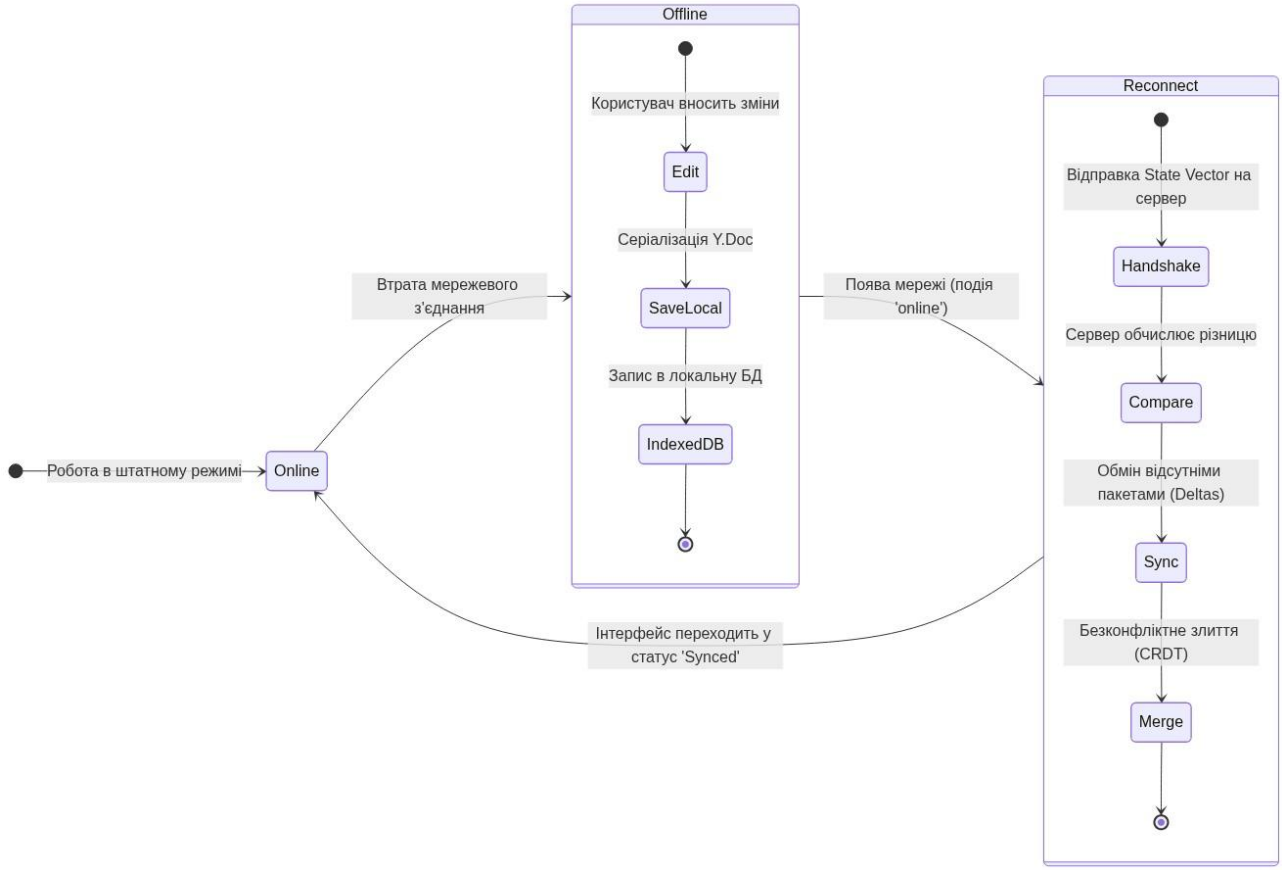

Рисунок 3.3 – Діаграма діяльності процесу офлайн-роботи та відновлення синхронізації

Проведене UML-моделювання дозволило чітко формалізувати вимоги до системи та визначити зони відповідальності кожного архітектурного компонента. Наступним кроком є вибір конкретних технологій та мов програмування для реалізації спроєктованої архітектури.

3.4 Обґрунтування вибору технологічного стеку

Для реалізації спроєктованої архітектури веб-додатка необхідно було обрати сучасний, надійний та масштабований технологічний стек. Вибір інструментарію базувався на вимогах до високої продуктивності клієнтської частини, стабільності з'єднання в реальному часі та необхідності суворого типізування даних на всіх етапах розробки.

Мова програмування: TypeScript. В якості основної мови розробки як для клієнтської, так і для серверної частини обрано TypeScript [19]. Це надмножина мови JavaScript, яка додає статичну типізацію. Використання TypeScript дозволяє виявляти архітектурні та логічні помилки на етапі компіляції, а не під час виконання програми. Крім того, використання єдиної мови на фронтенді та бекенді дозволяє перевикористовувати інтерфейси та типи даних (DTO), що значно пришвидшує процес розробки та знижує ризик розсинхронізації моделей даних між клієнтом і сервером.

Клієнтська частина (Frontend): Next.js та React. В якості основи клієнтської частини обрано фреймворк Next.js [9], побудований поверх бібліотеки React [20]. Використання архітектури App Router у Next.js дозволяє застосувати гібридний підхід до рендерингу. Статичні елементи інтерфейсу (навігація, панелі інструментів) формуються на сервері за допомогою React Server Components (RSC), що забезпечує мінімальний час до першого відмальовування.

Сам інтерактивний текстовий редактор завантажується як клієнтський компонент. Для реалізації текстового редактора обрано headless-фреймворк Tiptap [21]. На відміну від традиційних редакторів, Tiptap не має вбудованих стилів, що дозволяє повністю кастомізувати інтерфейс за допомогою Tailwind CSS. Важливою перевагою Tiptap є його нативна сумісність із бібліотекою Yjs, що значно спрощує інтеграцію механізмів спільного редагування.

Серверна частина: NestJS. Для розробки серверної інфраструктури обрано прогресивний Node.js фреймворк NestJS [8]. Вибір обґрунтований його

архітектурними особливостями. NestJS з коробки змушує розробника дотримуватися принципів Чистої архітектури (Clean Architecture) [13] та SOLID. Фреймворк має вбудований контейнер інверсії управління (IoC) та механізм впровадження залежностей. Це дозволяє чітко розділити бізнес-логіку від логіки маршрутизації та доступу до даних.

Для забезпечення синхронізації в реальному часі на базі NestJS розгорнуто WebSocket-сервер з використанням бібліотеки Hocuspocus. Вона виступає провайдером для Yjs-документів, керуючи кімнатами та ретранслюючи бінарні оновлення між підключеними клієнтами.

База даних та ORM: PostgreSQL та Prisma. Для забезпечення надійності зберігання даних обрано реляційну базу даних PostgreSQL [17]. На відміну від NoSQL рішень (наприклад, MongoDB), PostgreSQL забезпечує сувору цілісність даних (ACID) та має нативну підтримку зберігання бінарних масивів (тип bytea), що є критично важливим для збереження зліпків Yjs-документів.

Для взаємодії серверної частини з базою даних інтегровано об'єктно-реляційне відображення (ORM) Prisma [16]. Prisma генерує строгі TypeScript-типи на основі декларативної схеми бази даних. У парадигмі Чистої архітектури Prisma ізольована на інфраструктурному рівні за допомогою патерну «Репозиторій», що відокремлює бізнес-логіку застосунку від деталей реалізації SQL-запитів.

Локальне збереження: IndexedDB. Для реалізації повноцінного офлайн-досвіду дані зберігаються безпосередньо на пристрої клієнта. Використання стандартного localStorage є недоцільним через жорсткі обмеження обсягу (зазвичай до 5 МБ) та синхронну природу API, що блокує головний потік браузера. Тому для проєкту обрано IndexedDB [10] - вбудовану транзакційну базу даних браузера. Вона дозволяє асинхронно зберігати великі бінарні масиви даних, не впливаючи на плавність роботи інтерфейсу.

Обраний технологічний стек повністю відповідає поставленим завданням і забезпечує надійний фундамент для розробки масштабованого та відмовостійкого веб-додатка.

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

4.1 Розробка серверної частини та проєктування бази даних

Процес практичної реалізації веб-додатка розпочався з проєктування серверної інфраструктури. Основною вимогою до бекенду було забезпечення надійного збереження метаданих користувачів та бінарних станів документів, а також управління правами доступу.

Проєктування схеми бази даних (Prisma ORM). Для взаємодії з реляційною базою даних PostgreSQL було використано Prisma ORM. Схема бази даних (schema.prisma) була спроєктована з урахуванням специфіки роботи алгоритмів CRDT. Традиційні текстові редактори зберігають вміст документа у вигляді звичайного рядка (Plain Text) або HTML-розмітки. Однак, для забезпечення можливості безконфліктного злиття змін у майбутньому, система повинна зберігати повну історію операцій.

Тому було прийнято архітектурне рішення розділити сутність нотатки на дві таблиці:

- Note (Метадані): Зберігає текстову інформацію, таку як ідентифікатор документа, назва, дата створення та ідентифікатор власника.
- Document (Бінарний стан): Зберігає безпосередньо контент нотатки. Для цього використано тип даних Bytes (який мапиться на тип bytea у PostgreSQL). У цьому полі зберігається трансформований масив Uint8Array, згенерований бібліотекою Yjs.

Крім того, для реалізації спільного доступу було створено таблицю NoteAccess, яка реалізує зв'язок «багато-до-багатьох» між користувачами та нотатками, зберігаючи роль користувача (наприклад, VIEWER або EDITOR).

Реалізація бізнес-логіки (NestJS). Серверну частину побудовано на базі фреймворку NestJS із застосуванням модульної архітектури. Розподіл системи на окремі логічні домени (наприклад, AuthModule, NotesModule,

CollaborationModule) дозволяє досягти низької зв’язності між компонентами та високої згуртованості всередині кожного модуля.

Для дотримання принципів Чистої архітектури взаємодія з базою даних була інкапсульована за допомогою патерну «Репозиторій». Було створено інтерфейс INotesRepository, який визначає контракти для роботи з даними (наприклад, getBinaryData, saveBinaryData). Практична реалізація цього інтерфейсу (PrismaNotesRepository) інжектується в сервіси NestJS через механізм Dependency Injection. Такий підхід дозволяє легко тестувати бізнес-логіку за допомогою мок-об'єктів без необхідності підключення до реальної бази даних.

Інтеграція WebSocket-сервера. Для забезпечення синхронізації в реальному часі до екосистеми NestJS було інтегровано WebSocket-сервер на базі бібліотеки Hocuspocus. Інтеграція реалізована у вигляді окремого HocuspocusService, який ініціалізує сервер під час ініціалізації додатка.

Ключовим етапом налаштування сервера стала імплементація розширення @hocuspocus/extension-database. Це розширення використовує два основні хуки для зв'язку з базою даних:

Хук fetch: Викликається, коли перший клієнт підключається до кімнати (документа). Сервер звертається до PrismaNotesRepository, завантажує чи синхронізує бінарний стан документа з PostgreSQL і розгортає його в оперативній пам'яті (RAM).

Хук store: Викликається періодично, коли в документі відбуваються зміни. Для оптимізації навантаження на базу даних використовується механізм дебаунсингу (Debouncing) з інтервалом у 2000 мілісекунд. Сервер серіалізує поточний стан документа за допомогою методу Y.encodeStateAsUpdate та виконує операцію Upsert до бази даних.

Така гібридна модель (оперативна пам'ять для швидкої синхронізації + реляційна база даних для надійності зберігання) гарантує високу продуктивність системи та надійний захист даних від втрати у випадку критичного збою серверної інфраструктури.

4.2 Реалізація клієнтської частини та механізмів Local-first

Після розробки серверної інфраструктури наступним етапом стала реалізація клієнтського застосунку. Головним завданням було забезпечення миттєвого завантаження документів та повноцінної роботи користувача в умовах відсутності інтернет-з'єднання.

Організація архітектури фронтенду (Next.js). Клієнтська частина побудована на базі фреймворку Next.js з використанням архітектури App Router. Проєкт структуровано за принципами Feature-Sliced Design (FSD), що передбачає логічний поділ коду не за технічними аспектами (компоненти, хуки, стилі), а за функціональними модулями (наприклад, editor, auth, documents). Це забезпечує високу модульність та полегшує подальше масштабування кодової бази.

Для управління глобальним станом застосунку (наприклад, даними авторизованого користувача та налаштуваннями теми) використано бібліотеку Zustand, яка відрізняється мінімалістичним API та відсутністю зайвого бойлерплейту порівняно з Redux.

Інтеграція текстового редактора (Tiptap). В якості основного інструменту для взаємодії з текстом було інтегровано headless-редактор Tiptap. Вибір цього інструменту зумовлений його нативною підтримкою алгоритмів CRDT. Замість стандартної внутрішньої моделі даних (JSON або HTML), Tiptap було налаштовано на використання структури Y.XmlFragment з бібліотеки Yjs. Це означає, що кожне натискання клавіші користувачем автоматично транслюється у CRDT-операцію, готову до реплікації.

Для забезпечення сумісності з екосистемою популярних систем управління знаннями (зокрема Obsidian), редактор було розширено плагінами для підтримки Markdown-розмітки. Користувач має змогу використовувати стандартні шорткати (наприклад, ## для заголовків або > для цитат), які автоматично перетворюються на відповідні візуальні блоки.

Реалізація механізмів Local-first (IndexedDB). Найважливішим архітектурним рішенням клієнтської частини стала імплементація локального збереження даних. Для цього було розроблено кастомний React-хук useYjs, який інкапсулює логіку ініціалізації документа та управління життєвим циклом провайдерів синхронізації.

Процес ініціалізації документа відбувається у два етапи:

- Локальне завантаження: Спочатку ініціалізується провайдер IndexeddbPersistence. Він звертається до локальної бази даних браузера (IndexedDB) за унікальним ідентифікатором нотатки і десеріалізує збережений бінарний стан у пам'ять об'єкта Y.Doc.
- Мережева синхронізація: Паралельно ініціалізується HocuspocusProvider, який встановлює WebSocket-з'єднання з сервером для фонової синхронізації.

Критично важливим аспектом реалізації став захист від Race Condition. Графічний інтерфейс редактора (компонент Tiptap) монтується в DOM-дерево виключно після отримання події synced від провайдера IndexedDB. До цього моменту користувачу відображається стан завантаження. Це архітектурне рішення унеможливлює ситуацію, за якої порожній екземпляр редактора міг би ініціювати подію видалення існуючого контенту та затерти збережені дані користувача.

Завдяки такій реалізації, при повторному відкритті нотатки (навіть за повної відсутності інтернету), документ завантажується миттєво з локального сховища. Усі зміни, внесені користувачем в офлайн-режимі, надійно зберігаються в IndexedDB. При відновленні мережевого з'єднання HocuspocusProvider автоматично виконує процес з’єднання, обчислює вектори станів та безконфліктно відправляє накопичені локальні зміни на сервер.

4.3 Розробка користувацького інтерфейсу (UI/UX)

Інтерфейс користувача (UI) розроблено з акцентом на мінімалізм та усунення когнітивного перевантаження. Враховуючи, що основним завданням додатка є створення та редагування текстового контенту, було прийнято рішення відмовитися від класичних багатопанельних інтерфейсів (на кшталт Microsoft Word) на користь концепції «Zen Mode».

Концепція Zen Mode та стилізація. Інтерфейс редактора побудований таким чином, щоб забезпечити максимальне фокусування користувача на тексті. Усі допоміжні елементи керування приховані за замовчуванням і з'являються лише за потреби (наприклад, при виділенні тексту або наведенні на блок).

Для стилізації інтерфейсу використано фреймворк Tailwind CSS. Завдяки плагіну @tailwindcss/typography, текстовий контент автоматично отримує оптимізовані типографічні відступи, міжрядкові інтервали та розміри шрифтів, що забезпечує високу читабельність.

Головна сторінка редактора нотаток представлена на рисунку 4.1.

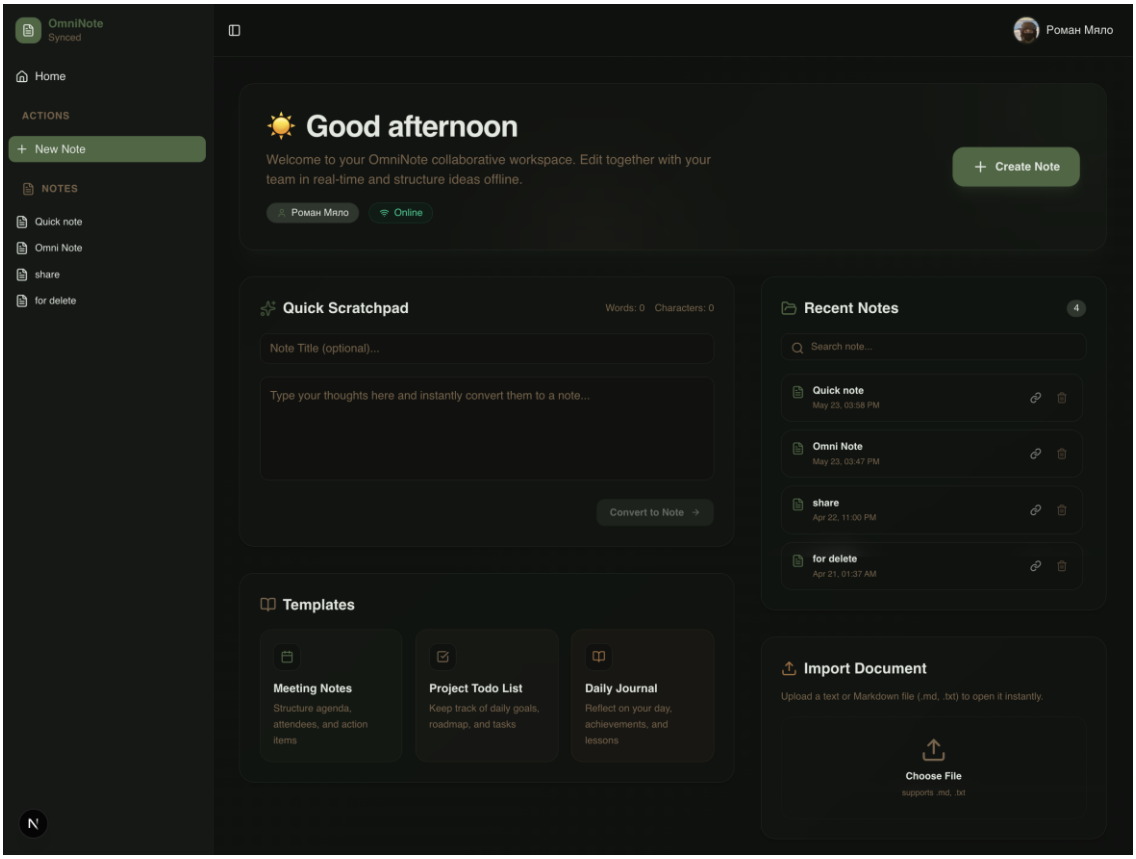

Рисунок 4.1 – Інтерфейс текстового редактора

Візуалізація спільної роботи. Оскільки додаток проєктувався для командної роботи, критично важливим елементом UX стала візуалізація присутності інших учасників. Для створення ефекту спільного простору реалізовано механізм відображення віддалених курсорів.

Завдяки інтеграції розширення CollaborationCursor для Tiptap та використанню протоколу Awareness від Yjs, система в реальному часі відстежує та транслює координати курсорів усіх активних користувачів у кімнаті. Кожен учасник отримує унікальний колір маркера та іменну мітку.

Для уникнення візуального шуму при одночасній роботі великої кількості користувачів, імена над курсорами відображаються лише в момент активного друку або при наведенні миші. Візуалізація процесу спільного редагування наведена на рисунку 4.2.

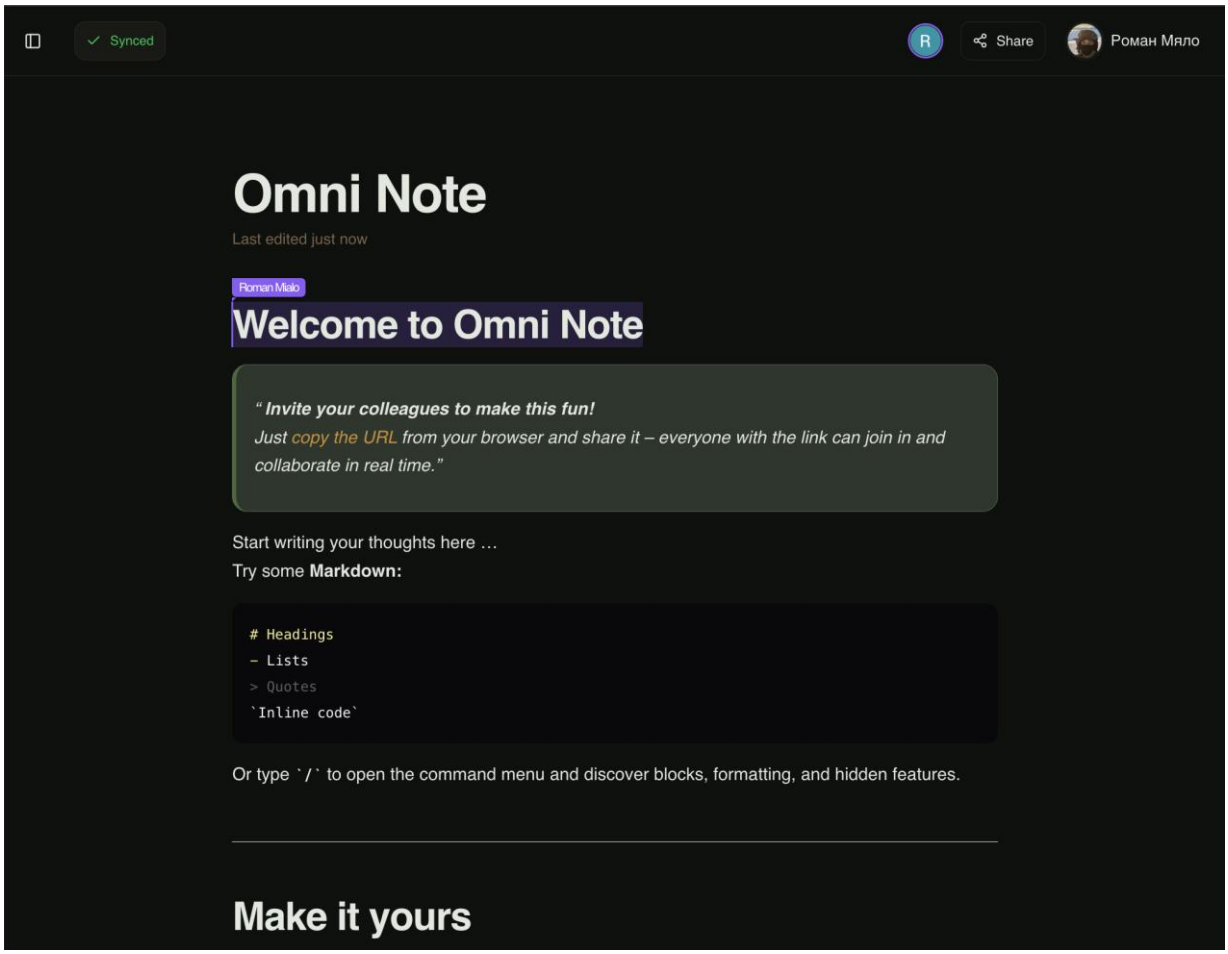

Рисунок 4.2 – Візуалізація віддалених курсорів при спільному редагуванні

Індикація стану синхронізації. Враховуючи архітектуру Local-first, користувач повинен чітко розуміти стан збереження своїх даних. Для цього в інтерфейсі реалізовано динамічний індикатор статусу підключення. Він має три стани:

- «Збережено локально» (Офлайн): Інформує користувача, що мережеве з'єднання відсутнє, але дані надійно збережені в IndexedDB.
- «Синхронізація» (Анімація): Відображається в момент відновлення зв'язку та передачі пакетів даних на сервер.
- «Збережено в хмарі» (Онлайн): Підтверджує, що всі локальні зміни успішно репліковані на сервері та доступні іншим учасникам.

Такий підхід до проєктування UX підвищує довіру користувача до системи та усуває страх втрати даних при роботі в умовах нестабільного інтернету.

4.4 Тестування системи та аналіз результатів

Для перевірки коректності роботи спроєктованої архітектури та підтвердження виконання поставлених завдань було проведено комплексне тестування веб-додатка. Основний акцент під час тестування робився на перевірці механізмів Local-first, стійкості системи до розривів мережевого з'єднання та коректності роботи алгоритмів CRDT при вирішенні конфліктів спільного редагування.

Тестування автономної роботи Offline-first. Метою цього етапу було підтвердити, що додаток здатний повноцінно функціонувати без доступу до центрального сервера, а локальна база даних (IndexedDB) надійно зберігає внесені зміни.

Сценарій тестування включав наступні кроки:

1. Користувач відкриває існуючий документ за наявності інтернет-з'єднання.
2. За допомогою інструментів розробника у браузері (Chrome DevTools -> Network) імітується повна втрата мережі.
3. Користувач вносить значні зміни до документа: додає нові абзаци, форматує текст, видаляє існуючі блоки.
4. Сторінка браузера примусово оновлюється.

Результат: Після оновлення сторінки в офлайн-режимі документ завантажився миттєво. Усі внесені зміни були повністю збережені та коректно відображені в інтерфейсі. Це підтверджує, що механізм y-indexeddb успішно серіалізує стан документа та виступає надійним первинним джерелом істини для клієнта.

Тестування безконфліктного злиття даних (CRDT). Метою цього етапу було перевірити здатність алгоритму YATA коректно об'єднувати конкурентні зміни, зроблені різними користувачами одночасно в умовах відсутності зв'язку.

Сценарій тестування (імітація стану Split-Brain):

1. Два користувачі (Клієнт А та Клієнт Б) відкривають один і той самий документ.
2. Обидва клієнти переходять в офлайн-режим, втрачаючи зв'язок із WebSocket-сервером та один з одним.
3. Клієнт А вставляє нове речення на початку документа.
4. Клієнт Б одночасно вставляє інше речення в ту саму позицію на початку документа, а також видаляє один з абзаців у кінці тексту.
5. Обидва клієнти відновлюють підключення до мережі (режим «Online»).

Результат: Після відновлення зв'язку клієнти автоматично виконали обмін векторами станів із сервером. Система успішно об'єднала зміни без втручання користувачів та без виведення повідомлень про помилки конфлікту версій. Речення, додані обома клієнтами на початку документа, були збережені (порядок їх розміщення був визначений алгоритмом YATA на основі

ідентифікаторів клієнтів). Абзац, видалений Клієнтом Б, був коректно видалений і на екрані Клієнта А. Обидва користувачі отримали ідентичний стан документа.

Аналіз продуктивності та затримок. Для оцінки швидкодії системи було проведено вимірювання затримок при спільному редагуванні в режимі реального часу. Завдяки використанню протоколу WebSocket та оптимізації передачі даних (відправка лише бінарних дельт, а не всього документа), затримка відображення символу, введеного одним користувачем, на екрані іншого користувача в середньому не перевищувала 40-100 мілісекунд (за умови стабільного широкосмугового з'єднання). Така затримка є непомітною для людського ока, що створює відчуття абсолютно синхронної роботи.

Крім того, механізм дебаунсингу на стороні сервера продемонстрував свою ефективність. Сервер виконував операцію запису бінарного зліпка до бази даних PostgreSQL лише після 2 секунд бездіяльності користувачів, що дозволило уникнути перевантаження бази даних надлишковими транзакціями під час активного друку.

Для підтвердження практичної значимості та стабільності розробки було використано даний проєкт в реальних умовах. Веб-додаток використовувався командою протягом місяця як основний інструмент для управління процесом створення іншого програмного проєкту.

У рамках тестування система застосовувалася для вирішення наступних завдань:

- Спільне проєктування архітектури: одночасне створення та редагування технічної документації, описів API та структур баз даних.
- Управління завданнями: фіксація вимог, створення списків завдань та відстеження прогресу їх виконання.
- Асинхронна робота: внесення ідей та правок до документації в умовах відсутності інтернет-з'єднання (наприклад, під час поїздок) з подальшою автоматичною синхронізацією.

За результатами реального використання було зроблено наступні висновки:

- Надійність офлайн-режиму: механізм Local-first повністю виправдав себе. Усі зміни, внесені в офлайн-режимі, безконфліктно зливалися з основною версією документа після відновлення мережі. Жодного випадку втрати даних або перезапису чужих змін зафіксовано не було.
- Ефективність спільної роботи: візуалізація присутності та відображення віддалених курсорів дозволили уникнути дублювання роботи під час одночасного редагування технічних специфікацій.
- Зручність інтерфейсу: мінімалістичний дизайн та підтримка Markdown-розмітки дозволили швидко структурувати технічні тексти, не відволікаючись на складні панелі форматування.

Підсумок тестування. Проведені експерименти та тестування в реальних умовах розробки підтвердили, що спроєктована архітектура успішно вирішує проблему залежності від інтернет-з'єднання. Алгоритми CRDT гарантують математично точне злиття даних, гібридна модель збереження (IndexedDB на клієнті + PostgreSQL на сервері) забезпечує високу надійність, а продуманий UX/UI-дизайн робить систему ефективним інструментом для командної роботи та управління знаннями.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було успішно вирішено актуальне науково-практичне завдання - спроєктовано та програмно реалізовано відмовостійкий веб-додаток для спільного ведення нотаток, побудований за архітектурною парадигмою «Local-first».

Проведений на початку роботи аналіз предметної області та існуючих рішень (Notion, Google Docs, Obsidian) дозволив виявити фундаментальний недолік класичної хмарної архітектури - повну залежність від стабільного інтернет-з'єднання. Це обґрунтувало необхідність розробки системи, яка здатна поєднати автономність локальних додатків із можливостями хмарної колаборації. Для досягнення цієї мети було здійснено порівняльний аналіз алгоритмів синхронізації, який довів, що використання безконфліктних реплікованих типів даних (CRDT) та алгоритму YATA є значно ефективнішим рішенням для забезпечення повноцінного офлайн-режиму, ніж традиційна операційна трансформація (OT). Теоретично обґрунтовано вибір AP-моделі згідно з CAP-теоремою та моделі Strong Eventual Consistency, що гарантує доступність системи та математично детерміноване злиття даних без їх втрати.

На основі теоретичних досліджень було спроєктовано архітектуру системи. За допомогою UML-моделювання розроблено логічну структуру додатка, яка чітко розмежовує процеси локального збереження, фонової синхронізації через WebSocket та персистентного зберігання даних на сервері. Практична реалізація цієї архітектури розпочалася зі створення серверної інфраструктури. Було розроблено бекенд на базі фреймворку NestJS із дотриманням принципів Чистої архітектури. Для взаємодії з даними спроєктовано схему реляційної бази PostgreSQL з використанням Prisma ORM. Важливим досягненням стала реалізація гібридної моделі зберігання: метадані та права доступу керуються реляційно, тоді як історія редагувань зберігається у

вигляді бінарних зліпків (Snapshots), що суттєво оптимізує навантаження на сервер.

Паралельно було розроблено клієнтський застосунок на базі фреймворку Next.js, інтегрований із текстовим редактором Tiptap, що підтримує Markdown-розмітку. Успішно імплементовано механізм локального збереження даних за допомогою IndexedDB, який виступає первинним джерелом істини для клієнта. Значну увагу приділено сучасним UX-рішенням: розроблено мінімалістичний користувацький інтерфейс, орієнтований на максимальну концентрацію. Для забезпечення ефективної командної роботи реалізовано протокол Workspace Awareness, який візуалізує присутність інших учасників та їхні віддалені курсори в реальному часі.

Завершальним етапом стало комплексне тестування системи. Експериментальна перевірка підтвердила працездатність додатка в умовах імітації втрати мережевого з'єднання. Алгоритми CRDT продемонстрували коректне та безконфліктне злиття змін при одночасному редагуванні документа кількома користувачами в офлайн-режимі з подальшим відновленням зв'язку.

Результати розробки пройшли успішне практичне тестування в реальних умовах. Веб-додаток використовувався командою як основний інструмент для спільного проєктування архітектури та управління завданнями під час створення іншого програмного продукту, що підтвердило його надійність та ефективність. За матеріалами дослідження були підготовлені та опубліковані тези доповіді у збірнику матеріалів XLVIX Міжнародної наукової конференції студентів та аспірантів «Актуальні питання розвитку науки та забезпечення якості освіти у XXI столітті».

Отже, розроблений веб-додаток повністю відповідає поставленим вимогам. Використання парадигми Local-first та алгоритмів CRDT дозволило створити надійний, швидкий та незалежний від якості інтернет-з'єднання інструмент для управління знаннями та спільної роботи. Результати роботи мають високу практичну цінність і можуть бути використані як основа для розробки інших відмовостійких розподілених систем реального часу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kleppmann M., Wiggins A., van Hardenberg P., et al. Local-first software: You own your data, in spite of the cloud // Ink & Switch. 2019. URL: https://www.inkandswitch.com/local-first/ (дата звернення: 15.05.2024).
2. Tanenbaum A. S., van Steen M. Distributed Systems: Principles and Paradigms. 3rd ed. CreateSpace Independent Publishing Platform, 2017. 696 p.
3. Brewer E. A. CAP twelve years later: How the "rules" have changed. Computer. 2012. Vol. 45, № 2. P. 23–29.
4. Ellis C. A., Gibbs S. J. Concurrency control in groupware systems. ACM SIGMOD Record. 1989. Vol. 18, № 2. P. 399–407.
5. Shapiro M., Preguiça N., Baquero C., Zawirski M. Conflict-free Replicated Data Types. Stabilization, Safety, and Security of Distributed Systems. Grenoble, France : Springer, 2011. P. 386–400.
6. Y.js Documentation. Shared Types & YATA Algorithm. URL: https://docs.yjs.dev/ (дата звернення: 15.05.2024).
7. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Addison-Wesley Professional, 2003. 208 p.
8. NestJS Documentation. A progressive Node.js framework. URL: https://docs.nestjs.com/ (дата звернення: 15.05.2024).
9. Next.js Documentation. The React Framework for the Web. URL: https://nextjs.org/docs (дата звернення: 15.05.2024).
10. MDN Web Docs. IndexedDB API // Mozilla Developer Network. URL: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API (дата звернення: 15.05.2024).
11. Gutwin C., Greenberg S. A Descriptive Framework of Workspace Awareness for Real-Time Groupware. Computer Supported Cooperative Work (CSCW). 2002. Vol. 11. P. 411–446.

12. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol : O’Reilly Media, 2017. 616 p.
13. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.
14. Notion – The all-in-one workspace for your notes, tasks, wikis, and databases. URL: https://www.notion.so/ (дата звернення: 15.05.2024).
15. Obsidian – Sharpen your thinking. URL: https://obsidian.md/ (дата звернення: 15.05.2024).
16. Prisma Documentation. Next-generation Node.js and TypeScript ORM. URL: https://www.prisma.io/docs (дата звернення: 15.05.2024).
17. PostgreSQL: The World's Most Advanced Open Source Relational Database. URL: https://www.postgresql.org/docs/ (дата звернення: 15.05.2024).
18. Fette I., Melnikov A. The WebSocket Protocol. RFC 6455 // IETF. 2011. URL: https://datatracker.ietf.org/doc/html/rfc6455 (дата звернення: 15.05.2024).
19. TypeScript Documentation. Typed JavaScript at Any Scale. URL: https://www.typescriptlang.org/docs/ (дата звернення: 15.05.2024).
20. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. Sebastopol : O’Reilly Media, 2020. 298 p.
21. Tiptap – The headless rich text editor framework for web artisans. URL: https://tiptap.dev/ (дата звернення: 15.05.2024).
22. Material Design 3 Guidelines. Layout and Responsiveness // Google Design. URL: https://m3.material.io/ (дата звернення: 15.05.2024).