НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

Комп'ютерних наук

_____Голуб Б.Л.

" " 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

«Програмне забезпечення інформаційної системи ведення нотаток для навчання»

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми

к.т.н, доцент	_____	Вайганг Г.О.
(Науковий ступень та вчене звання)	(підпис)	(ПІБ)

Керівник бакалаврської кваліфікаційної роботи

к.т.н., доцент	_____	Бородкіна І.Л.
(Науковий ступень та вчене звання)	(підпис)	(ПІБ)

Виконав _____ Бражник Д. В.
(підпис) (ПІБ)

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
_____Голуб Б.Л.
_“ ”_____20 р._

З А В Д А Н Н Я

на виконання бакалаврської кваліфікаційної роботи студенту

Бражнику Денису Валентиновичу

Спеціальність 121 – «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи: «Програмне забезпечення інформаційної системи ведення нотаток для навчання»

Затверджена наказом ректора НУБіП України від 16.12.2024 № 2248 «С».

Термін подання завершеної роботи на кафедру _____

Вихідні дані до бакалаврської кваліфікаційної роботи: документація бібліотек, нормативні та технічні документи

Перелік питань , які потрібно розробити: проаналізувати предметну область, аналоги, розробити вимоги до системи, спроектувати та розробити систему

Дата видачі завдання “_____”_____20___ р.

Керівник бакалаврської кваліфікаційної роботи

_____к.т.н., доцент_____ _____ ___Бородкіна І.Л. _

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Бражник Д.В.

(підпис) (ПІБ студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API — Інтерфейс програмування застосунків (API, Application Programming Interface)

БД — база даних

Clerk — сервіс для організації авторизації та аутентифікації користувачів у веб-додатках

Convex — хмарна платформа для зберігання та обробки даних у реальному часі

HTML — HTML (HyperText Markup Language) — мова розмітки, що використовується для структурування веб-сторінок

ІС — інформаційна система

JS — JavaScript, мова програмування для додавання динамічної поведінки веб-сторінкам

JSON — JavaScript Object Notation, текстова структура даних для взаємодії клієнта з сервером

Liveblocks — платформа для реалізації спільної роботи в режимі реального часу у веб-застосунках

MVC — Model-View-Controller, архітектурний шаблон побудови програмного забезпечення

Next.js — фреймворк для розробки серверних і клієнтських React-додатків

ПЗ — програмне забезпечення

React — бібліотека JavaScript для створення інтерфейсів користувача

SQL — Structured Query Language, мова структурованих запитів до баз даних

СУБД — система управління базами даних

Tailwind CSS — утилітарний CSS-фреймворк для швидкої розробки стилізованих інтерфейсів

UI — User Interface, інтерфейс користувача

ВСТУП

В епоху сучасних інформаційних технологій суспільстві постійне навчання стало необхідністю. Великий обсяг матеріалів, які потрібно засвоїти, систематизувати й зберігати, зумовлює потребу в ефективних цифрових інструментах для організації навчального процесу. Традиційні способи ведення конспектів, як-от записування в зошитах чи текстових редакторах без синхронізації, дедалі частіше поступаються місцем сучасним вебсервісам, що дають змогу не лише створювати нотатки, а й структурувати, зберігати та ділитися ними з іншими користувачами в реальному часі.

Потреба у зручному, швидкому та доступному інструменті для створення й ведення навчальних нотаток особливо актуальна для студентів, викладачів та всіх, хто займається самоосвітою. Сучасні вебтехнології дозволяють реалізувати такі системи у вигляді багатофункціональних, інтуїтивно зрозумілих і масштабованих вебзастосунків.

У зв’язку з цим виникає потреба у створенні інформаційної системи, яка дозволить:

- створювати, редагувати та структурувати нотатки;
- зберігати їх у хмарному сховищі;
- організовувати спільну роботу над матеріалами в реальному часі;
- забезпечувати авторизацію користувачів і захист даних.

Метою дослідження кваліфікаційної роботи є розробка вебзастосунку — інформаційної системи для ведення навчальних нотаток, яка забезпечить зручність у використанні, функціональність, швидкодію та підтримку спільної роботи.

Об’єктом дослідження дипломної роботи є створення програмного забезпечення інформаційної системи для створення та ведення навчальних нотаток.

Предметом дослідження є методи та засоби створення, зберігання, синхронізації та спільного редагування навчальних матеріалів у вебсередовищі.

Завдання дослідження:

1. Проаналізувати існуючі рішення в сфері створення та зберігання навчальних нотаток.
2. Сформулювати вимоги до майбутньої системи.
3. Обґрунтувати вибір технологій та інструментів для реалізації.
4. Розробити архітектуру програмного забезпечення.
5. Реалізувати вебзастосунок відповідно до технічного завдання.
6. Провести тестування функціоналу та оцінити зручність використання системи.

Для реалізації інформаційної системи використано сучасний стек вебтехнологій. Основним фреймворком обрано **Next.js 15**, що забезпечує швидку серверну та клієнтську обробку. Фреймворк працює на базі **Node.js**, що дозволяє реалізовувати серверну логіку, обробку API-запитів та серверний рендеринг. Для побудови інтерфейсу застосовано бібліотеку **React**, а для стилізації — утилітарний CSS-фреймворк **Tailwind CSS**. Як хмарну базу даних у реальному часі використано **Convex**, що дозволяє синхронізувати дані між користувачами без затримок. Для реалізації авторизації та аутентифікації застосовується **Clerk**. Можливість спільного редагування в реальному часі забезпечується завдяки **Liveblocks**.

Розроблена система має практичне значення для широкого кола користувачів, пов’язаних з навчальним процесом. Вона дозволяє:

- створювати та редагувати нотатки швидко і зручно;
- зберігати матеріали в хмарі з доступом з будь-якого пристрою;
- працювати над матеріалами в команді;
- мати сучасний інтерфейс і високу продуктивність.

У рамках даної бакалаврської роботи реалізовано повний цикл створення інформаційної системи — від аналізу предметної області до створення готового програмного продукту. Розроблена система відповідає вимогам сучасного програмного забезпечення та готова до практичного використання.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Опис предметної області

В умовах сучасної освіти обсяг інформації, яку необхідно опрацювати студентам та учням, постійно зростає. Традиційні способи фіксації навчального матеріалу — записи у зошитах або текстових документах — стають дедалі менш ефективними через свою обмежену функціональність, відсутність централізованого зберігання, а також неможливість швидкої організації, пошуку або спільної роботи над матеріалами.

Особливо актуальною є проблема втрати інформації: паперові нотатки можуть бути загублені або пошкоджені, а локальні файли — випадково видалені чи недоступні з інших пристроїв. Крім того, сучасне навчання часто передбачає колективну роботу над проєктами або обмін конспектами, що ускладнюється при використанні несинхронізованих або офлайн-методів зберігання даних.

Ці виклики зумовлюють потребу в інформаційній системі, що дозволяє зберігати, структурувати, редагувати та ділитися навчальними нотатками в зручний і безпечний спосіб. Така система має бути доступною з будь-якого пристрою, підтримувати спільну роботу в реальному часі, мати сучасний користувацький інтерфейс та забезпечувати збереження даних у хмарі.

Крім цього, актуальним завданням є впровадження інструментів, які сприяють організації навчального процесу: категоризація нотаток, можливість пошуку, прикріплення файлів, гнучке форматування тексту тощо. Це дозволяє користувачам адаптувати робочий простір до власного стилю навчання й ефективно керувати великою кількістю матеріалів.

Таким чином, інформаційна система для ведення навчальних нотаток повинна відповідати вимогам цифрової освіти: бути швидкою, доступною, зручною та безпечною. Її впровадження дозволить не лише підвищити ефективність індивідуального навчання, але й забезпечить можливість

співпраці, обміну знаннями та централізованого зберігання навчового контенту.

1.2. Аналіз вимог до програмної системи

Проаналізувавши предметну область розроблюваного програмного забезпечення інформаційної системи ведення нотаток для навчання, необхідно визначити вимоги до неї. Аналіз вимог дозволяє сформулювати очікування користувачів, оцінити технічні обмеження, спрогнозувати потенційні ризики та обрати відповідні інструменти реалізації.

Функціональні вимоги:

1. Реєстрація та авторизація користувачів — за допомогою сервісу Clerk користувачі можуть безпечно входити до системи.
2. Створення нових нотаток — користувач має можливість створювати нотатки з довільним текстом.
3. Редагування нотаток — зміна змісту нотатки в зручному редакторі.
4. Збереження змін у реальному часі — інтеграція з Convex забезпечує синхронізацію даних без перезавантаження сторінки.
5. Спільне редагування — завдяки платформі Liveblocks кілька користувачів можуть одночасно працювати над однією нотаткою.
6. Видалення нотаток — користувач може видаляти непотрібні або застарілі нотатки.
7. Організація нотаток за категоріями/папками — для зручного структурування інформації.
8. Інтуїтивно зрозумілий інтерфейс — мінімалістичний дизайн із використанням Tailwind CSS.
9. Підтримка темної та світлої теми — адаптація під індивідуальні вподобання користувача.
10. Пошук нотаток — система дозволяє швидко знаходити необхідні нотатки за ключовими словами.

Нефункціональні вимоги:

- Нефункціональні вимоги стосуються якості роботи системи, її продуктивності, надійності, масштабованості тощо:
- Швидкодія — інтерфейс має миттєво реагувати на дії користувача.
- Масштабованість — система повинна легко розширюватися для підтримки нових функцій.
- Надійність — виключення втрати даних завдяки збереженню в реальному часі.
- Безпека — авторизація через Clerk забезпечує захист особистих даних.
- Кросплатформеність — доступ до системи з будь-якого сучасного браузера (на ПК, планшеті чи смартфоні).
- Мінімалізм у дизайні — відсутність перевантаженого інтерфейсу, фокус на зручності.

Технічні вимоги:

- Середовище та технології, що використовуються для реалізації системи:
- Next.js 15 — фреймворк для серверного та клієнтського рендерингу.
- React — бібліотека для побудови інтерфейсу.
- Tailwind CSS — утилітарний CSS-фреймворк.
- Convex — база даних для роботи в реальному часі.
- Clerk — сервіс для аутентифікації.
- Liveblocks — спільна робота над документами.
- Node.js — середовище виконання JavaScript на сервері.
- Vercel — хостинг і платформа для розгортання застосунку.

Таким чином, системі NoteMinds поставлені чіткі функціональні та технічні вимоги, які визначають її логіку, архітектуру та взаємодію з користувачем. Їх виконання гарантує, що програмне рішення буде сучасним, надійним і корисним у навчальному процесі.

1.3. Моделювання предметної області

Моделювання предметної області має ключове значення у процесі розробки програмного забезпечення, адже дає змогу формалізувати знання про систему, що розробляється. Це покращує розуміння вимог, допоможе визначити потенційні проблеми та закладає основу для подальшого проектування та впровадження системи.

Уніфікована мова моделювання (Unified Modeling Language, UML) – це графічна мова для специфікації, візуалізації, конструю-вання і документування програмних систем. За допомогою UML можна розробити детальний план такої системи, який відображатиме і концептуальні елементи системи (системні функції та бізнес-процеси), і особливості її реалізації (класи, схеми баз даних, програмні компоненти багаторазового використання тощо) [1].

UML пропонує різноманітні діаграми для моделювання різних компонентів і аспектів системи:

- Діаграми варіантів використання (Use Case Diagrams) ілюструють функціонал системи з перспективи користувача.
- Діаграми класів (Class Diagrams) — описують структуру системи, включаючи класи, їх атрибути, методи та взаємозв'язки.
- Діаграми послідовностей (Sequence Diagrams) ілюструють послідовність взаємодій між об’єктами.
- Діаграми діяльності (Activity Diagrams) відображають потоки управління або даних у процесах.
- Діаграми станів (State Diagrams) — відображають зміни станів об'єкта протягом його життєвого циклу.

Моделювання предметної області доцільно почати з виділення основних абстракцій, які присутні в системі NoteMinds. На рисунку 1.1 представлено ключові абстракції системи.

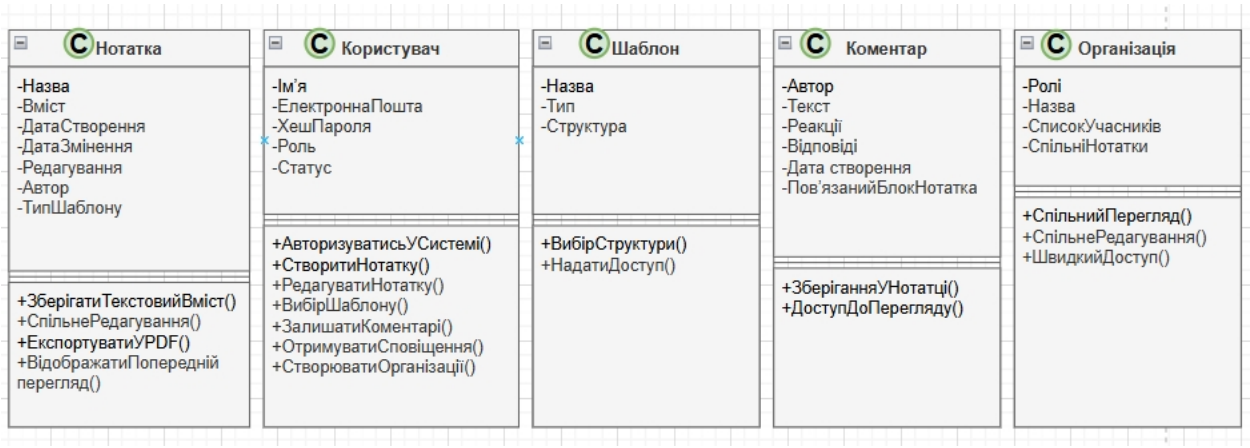

Рис. 1.1. Основні абстракції системи

Абстракція «Нотатка» містить всі властивості нотатки, такі як id, назва, вміст, дата створення, дата останнього редагування, автор, тип шаблону тощо. Ця абстракція відповідає за збереження та надання доступу до текстового вмісту, експорт у PDF та забезпечення спільного редагування.

Абстракція «Користувач» має властивості id, ім'я, email, роль (Student / Lecturer), аватар. Користувач може реєструватися й авторизуватися через Clerk, створювати, редагувати та видаляти власні нотатки, брати участь у спільних сесіях редагування та керувати папками й шаблонами в межах своєї організації.

Абстракція «Шаблон» зберігає id, назву, тип шаблону (лабораторна, курсовий проект, резюме тощо) і перелік блоків (заголовки, текстові поля, списки). Відповідальність цієї абстракції — надавати користувачеві структуру нотатки у момент створення та передавати метадані до клієнтського редактора.

Абстракція «Коментар» включає такі властивості, як id, автор, текст, дата створення і посилання на конкретний блок нотатки. Вона відповідає за

відображення коментарів поруч із відповідними фрагментами, підтримку ланцюгового обговорення й інтеграцію зі сповіщеннями.

Абстракція «Організація» містить id, назву, список учасників (з ролями Student, Lecturer) та перелік спільних структур (папок, шаблонів, нотаток). Ця сутність відповідає за керування правами доступу, централізоване зберігання матеріалів і координацію спільної роботи в групі.

Діаграма варіантів використання (прецедентів) (Use case diagram) – це одна з графічних діаграм UML, яка використовується для моделювання ПС, зокрема, об’єктно-орієнтованих систем. Вона задає концептуальну модель системи. Діаграма прецедентів є базою для подальшої деталізації ПС у формі фізичних і логічних моделей [2].

На рисунку 1.2 зображено діаграму прецедентів системи NoteMinds, яка демонструє взаємодію трьох основних акторів: студента, викладача та адміністратора. Студент після входу в систему може використовувати шаблони, створювати нотатки, експортувати їх у PDF, шукати серед них за ключовими словами, отримувати сповіщення, переходити до зовнішньої платформи E-Learn, працювати над нотаткою спільно з іншими користувачами, а також переглядати історію змін. Деякі дії включають інші: наприклад, спільний перегляд включає додавання коментарів, а перегляд логів — перевірку користувача.

Викладач додатково має змогу переглядати шаблони, перевіряти готовність нотаток студентів, залишати зауваження та управляти організаціями.

Адміністратор відповідає за управління обліковими записами, перегляд усіх студентських нотаток та перевірку активності користувачів. Усі користувачі починають роботу з авторизації, після чого мають доступ до функцій відповідно до своєї ролі.

Рис. 1.2. Діаграма прецедентів системи

На рисунку 1.3 представлено діаграму послідовності для користувача системи NoteMinds. Сценарій починається з того, що користувач відкриває сторінку. Інтерфейс (UI) надсилає запит на реєстрацію до сервера, а сервер — запит до бази даних на створення нового користувача. Після підтвердження, користувач обирає шаблон для нотаток, і сервер створює відповідну нотатку.

Далі відображається створена нотатка, і користувач натискає «поділитися» — система надсилає доступ викладачу. Після цього студент відкриває нотатку, відбувається запит до джерела даних для отримання її вмісту, а також збереження коментарів.

Як тільки користувач натискає «прийняти запрошення», сервер надсилає відповідне сповіщення, оновлює права доступу в сховищі даних, після чого зміни застосовуються до інтерфейсу.

Таким чином, діаграма відображає ключову логіку взаємодії між користувачем, інтерфейсом, сервером і базою даних в процесі опрацювання нотатками в системі.

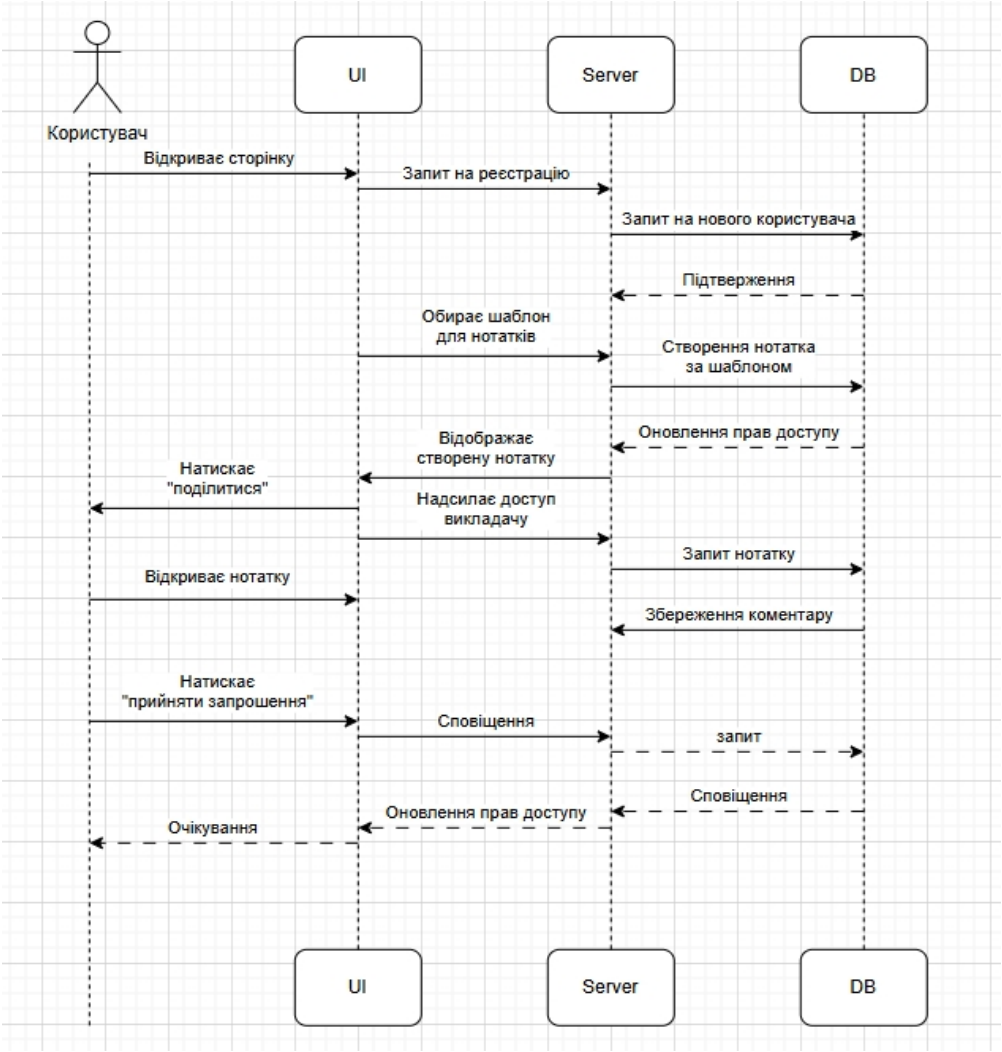

Рис. 1.3. Діаграма послідовності до проектованої системи

1.4. Огляд інформаційних джерел та існуючих рішень

Для ефективного проєктування інформаційної системи ведення нотаток проведено огляд літератури й наукових публікацій публікацій, методичних рекомендацій, а також існуючих рішень у сфері цифрових платформ для створення та збереження нотаток.

До основних інформаційних джерел належать:

Наукова література: підручники з проєктування інформаційних систем, керівництва з використання UML-діаграм, в яких описано методики структурного та об'єктно-орієнтованого аналізу ПЗ;

Методичні вказівки університетів, що містять приклади побудови діаграм прецедентів, послідовностей, класів тощо;

Документація та офіційні ресурси бібліотек та сервісів, що використовуються у проєкті (Next.js, Clerk, Convex, Liveblocks, Tailwind, Vercel), зокрема: nextjs.org, clerk.dev, convex.dev, liveblocks.io,

Також був проведений аналіз подібних рішень, що вже реалізовані на ринку.

1.4.1. Аналоги та існуючі рішення

На ринку представлено декілька популярних сервісів, які частково або повністю реалізують функціональність ведення нотаток для навчання:

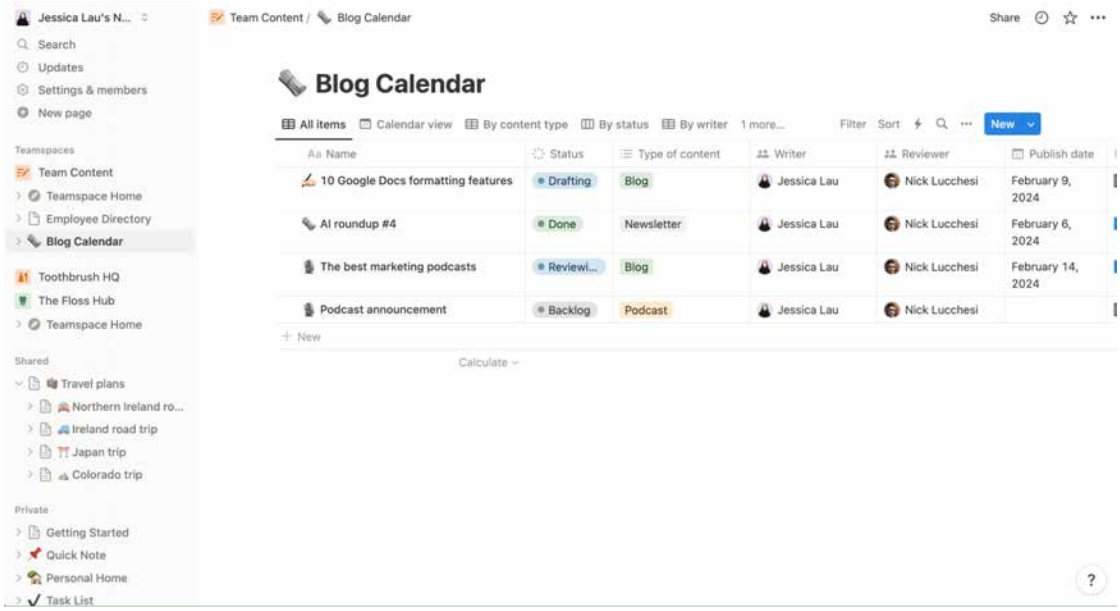

Рис. 1.4. Скріншот роботи сервісу «Notion»

Notion – універсальний сервіс для створення нотаток, баз даних, списків справ, співпраці. Має багатий функціонал, але не спеціалізований саме на навчальному процесі.

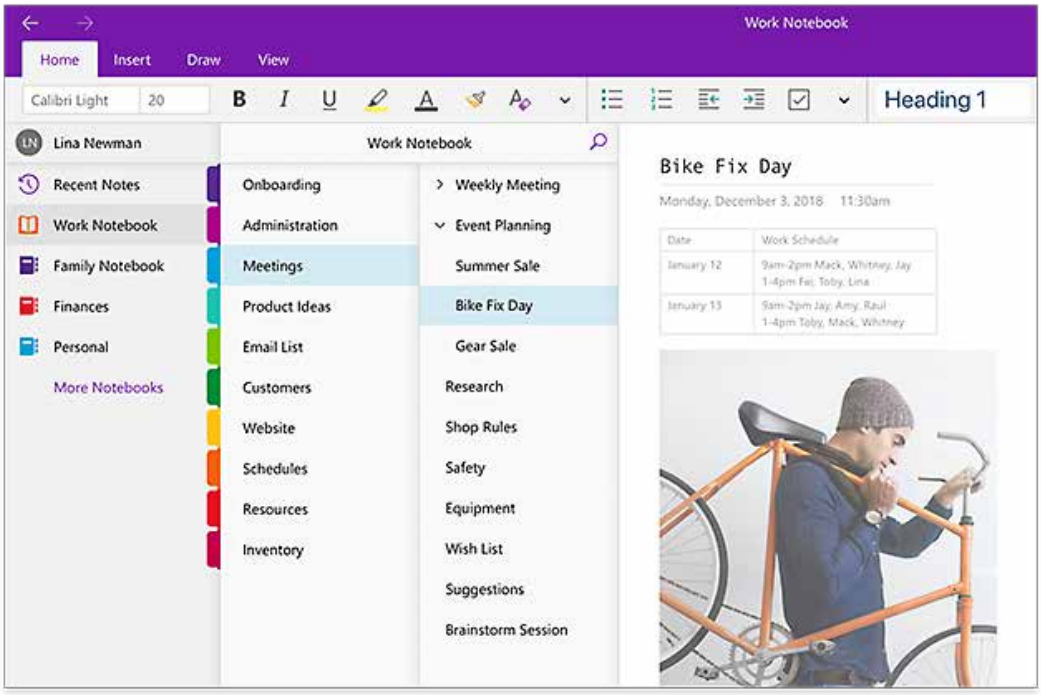

Рис. 1.5. Скріншот роботи сервісу «Microsoft OneNote»

Microsoft OneNote – орієнтований на створення рукописних або текстових нотаток, дозволяє інтеграцію з Microsoft Teams. Підходить для навчання, але має складний інтерфейс для новачків.

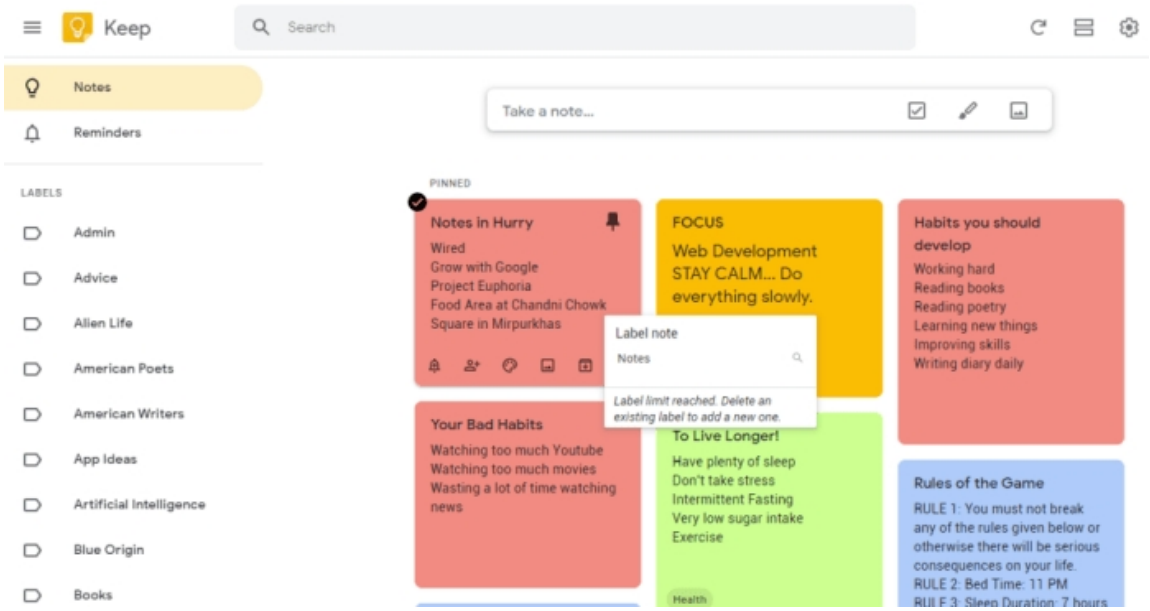

Рис. 1.6. Скріншот роботи сервісу «Google Keep»

Google Keep – проста система для нотаток, але не має підтримки форматування, шаблонів або співпраці в режимі реального часу.

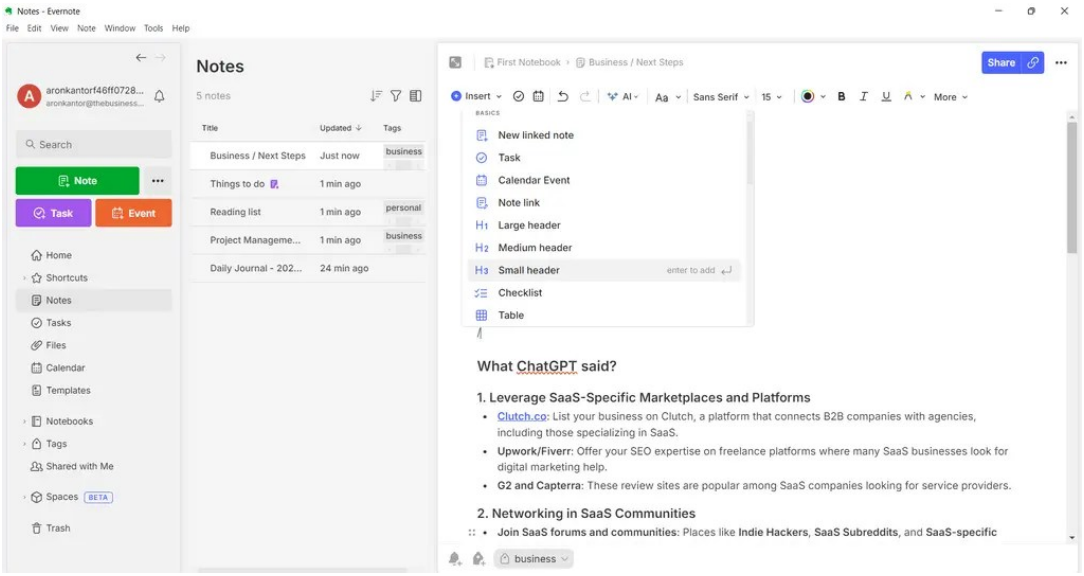

Рис. 1.7. Скріншот роботи сервісу «Evernote»

Evernote – підтримує теги, шаблони, форматування. Проте більшість функцій доступні у платній версії.

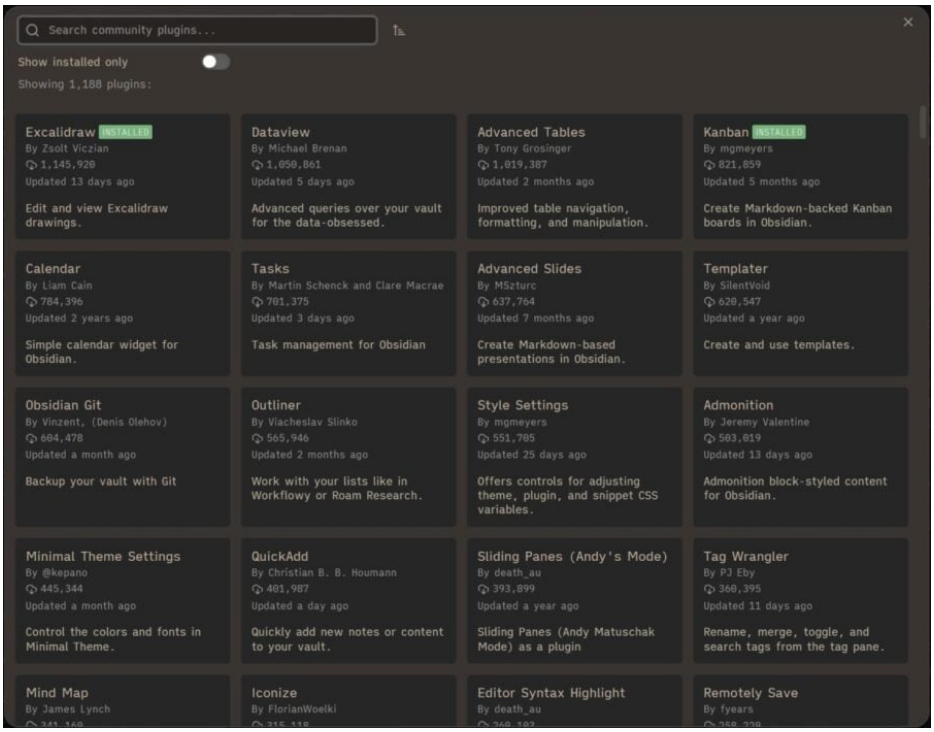

Рис. 1.8. Скріншот роботи сервісу «Obsidian»

Obsidian – локальний менеджер нотаток з підтримкою markdown, плагінів, візуалізації зв'язків між нотатками. Орієнтований більше на дослідницьку та індивідуальну роботу.

Таблиця 1.1 – Порівняння аналогів інформаційної системи NoteMinds

Критерій/ Система	**NoteMinds**	**Notion**	**OneNote**	**Google Keep**	**Evernote**	**Obsidian**
Співпраця в реальному часі	**Так**	Так	Обмежено	Ні	Ні	Ні
Підтримка шаблонів	**Так**	Так	Ні	Ні	Так	Через плагіни
Форматування тексту	**Так**	Так	Так	Обмежено	Так	Так (Markdown)
Експорт у PDF	**Так**	Так	Так	Ні	Платно	Так
Коментарі / фідбек	**Так**	Так	Частково	Ні	Так	Ні
Інтеграція з навчальними платформами	**Так (E-Learn)**	Обмежено	MS Teams	Ні	Ні	Ні
Безкоштовна версія	**Так**	Так	Так	Так	Обмежено	Так
Підтримка ролей користувачів	**Так (2 ролей)**	Ні	Ні	Ні	Ні	Ні

1.4.2. Відмінності запропонованого рішення

Інформаційна система NoteMinds, яка розробляється у межах даної дипломної роботи, має низку відмінностей:

- спеціалізація на освітньому процесі, з шаблонами для лабораторних, курсових тощо;
- спільна робота над нотатками з використанням Liveblocks (реальний час);
- авторизація через Clerk, з підтримкою ролей (студент, викладач);
- інтеграція з зовнішніми платформами (наприклад, E-Learn);
- перегляд логів, коментарі, експорт у PDF;
- зберігання даних у Convex, що забезпечує оптимальну швидкість та масштабованість;
- розгортання на Vercel, що забезпечує стабільну доступність системи.

Таким чином, запропоноване рішення заповнює нішу між загальними інструментами для нотаток та повноцінними освітніми платформами, забезпечуючи гнучкість, простоту використання та фокус на навчанні.

1.5. Постановка завдання

Основною метою цієї роботи є створення інформаційної системи NoteMinds, яка забезпечує створення, редагування, збереження та спільну роботу з навчальними нотатками у зручному та функціональному середовищі. Система повинна бути орієнтована на студентів, викладачів та адміністраторів освітніх організацій.

Щоб досягти поставленої мети, потрібно виконати такі ключові завдання:

Розробити вебзастосунок із сучасним і зрозумілим інтерфейсом користувача, орієнтованим на зручність використання у навчальному процесі.

Реалізувати механізм реєстрації та автентифікації користувачів з підтримкою розподілу за ролями: Студент, Викладач, Адміністратор.

Забезпечити створення та редагування нотаток з підтримкою форматування тексту, збереженням історії змін, та можливістю коментування.

Передбачити функцію використання шаблонів, зокрема: лабораторні, курсові, лекційні тощо.

Реалізувати спільну роботу в реальному часі (на основі Liveblocks), включаючи синхронізацію редагувань, коментарів і змін.

Вбудувати систему пошуку, яка дає можливість шукати за вмісту власних нотаток за ключовими словами.

Забезпечити експорт нотаток у формат PDF, з урахуванням збереження стилів і структури документу.

Реалізувати систему сповіщень, яка інформує користувачів про події, пов’язані з нотатками (коментарі, запрошення, зміни).

Забезпечити інтеграцію з навчальними платформами (наприклад, перехід на курс у E-Learn за посиланням).

Розробити модуль адміністрування, що дає можливість керувати користувачами, організаціями та переглядати всі нотатки.

Система повинна бути розроблена з використанням сучасного вебстека, зокрема: Next.js 15, React, Tailwind CSS, Convex, Clerk, Liveblocks, а також Node.js для серверної логіки, де це необхідно.

У підсумку, очікується створення гнучкої та масштабованої інформаційної системи, що відповідає актуальним потребам сучасної освіти.

1.6. Висновки до розділу 1

Предметна область вказала на те, що зростаючу потребу в цифрових інструментах для створення, зберігання та спільного редагування навчальних нотаток, що зумовлено зростанням обсягу освітніх матеріалів та необхідністю їх організації.

Аналіз вимог сформулював чіткі функціональні (реєстрація, шаблони, редактор, спільна робота, пошук, експорт, сповіщення) та нефункціональні (швидкодія, безпека, кросплатформеність, масштабованість) критерії, що ляжуть в основу архітектури системи.

Формування моделі предметної області з використанням виділенням основних абстракцій (Користувач, Нотатка, Шаблон, Коментар, Організація, Пошуковий запит) надало структуроване уявлення про компоненти системи та їх взаємодію.

Аналіз інформаційних джерел і наявних рішень показав змогу порівняти NoteMinds з популярними аналогами (Notion, OneNote, Google Keep, Evernote, Obsidian) та обґрунтувати унікальні переваги нашої системи: фокус на освіті, ролі користувачів, реальний час співпраці, інтеграція з E-Learn.

Постановка завдання чітко окреслила комплекс робіт: від вибору технологічного стеку (Next.js 15, React, Tailwind CSS, Convex, Clerk, Liveblocks, Node.js) до реалізації інтерфейсу, серверної логіки, спільної роботи, пошуку та адміністрування.

Отже, на основі проведеного аналізу та моделювання сформовано базу для проектування архітектури та подальшої практичної реалізації NoteMinds у наступних розділах роботи.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Логічна модель даних у вигляді ER-діаграми

ER-моделювання являє собою низхідний підхід до проєктування БД, що починається з визначення найбільш важливих даних, так званих сутностей, і зв'язків між даними, що мають бути подані в моделі [3].

Ключові компоненти ER-діаграми: Сутніст (Entity) — об’єкт або поняття з реального світу, важливе для системи, зображується прямокутником (наприклад, «Студент», «Курс», «Замовлення»). Атрибут (Attribute) — характеристика сутності, зображується овалом, що з’єднується з сутністю; бувають прості (наприклад, «Ім’я»), складені (наприклад, «Адреса», що складається з «Місто», «Вулиця»), багатозначні (наприклад, кілька телефонних номерів). Ключовий атрибут (Key Attribute) — атрибут, який унікально ідентифікує кожен екземпляр сутності, позначається підкресленням.

Зв’язок (Relationship) — асоціація між сутностями, зображується ромбом (наприклад, «Записаний на», «Працює в»). Кардинальність (Cardinality) — визначає кількість екземплярів однієї сутності, що пов’язані з іншою (наприклад, один студент може бути записаний на кілька курсів, а курс — мати багато студентів). Типи зв’язків: відповідні кардинальності: 1:1, 1:N, M:N.

На рисунку 2.1 показано ER-модель бази даних, яка охоплює сутності Користувач, Шаблон, Колаборація, Організація, Нотатка, Відгук та Курс.

Кожна сутність має первинний ключ (наприклад, Ідентифікатор_користувача для Користувача) та атрибути (назва, пошта, пароль тощо).

Зв’язки між сутностями реалізовані через зовнішні ключі: наприклад, Шаблон (Ідентифікатор_шаблону) пов’язаний з Користувачем через Ідентифікатор_користувача (зв’язок 1:N, де один користувач створює багато шаблонів).

Колаборація є проміжною таблицею для зв’язку M:N між Нотатками, Шаблонами та Користувачами.

Організація керується користувачем (1:N), а Нотатка прив’язана до користувача та організації (N:1).

Відгуки залежать від Нотаток (1:N), а Курс може бути пов’язаний з Нотаткою (1:1).

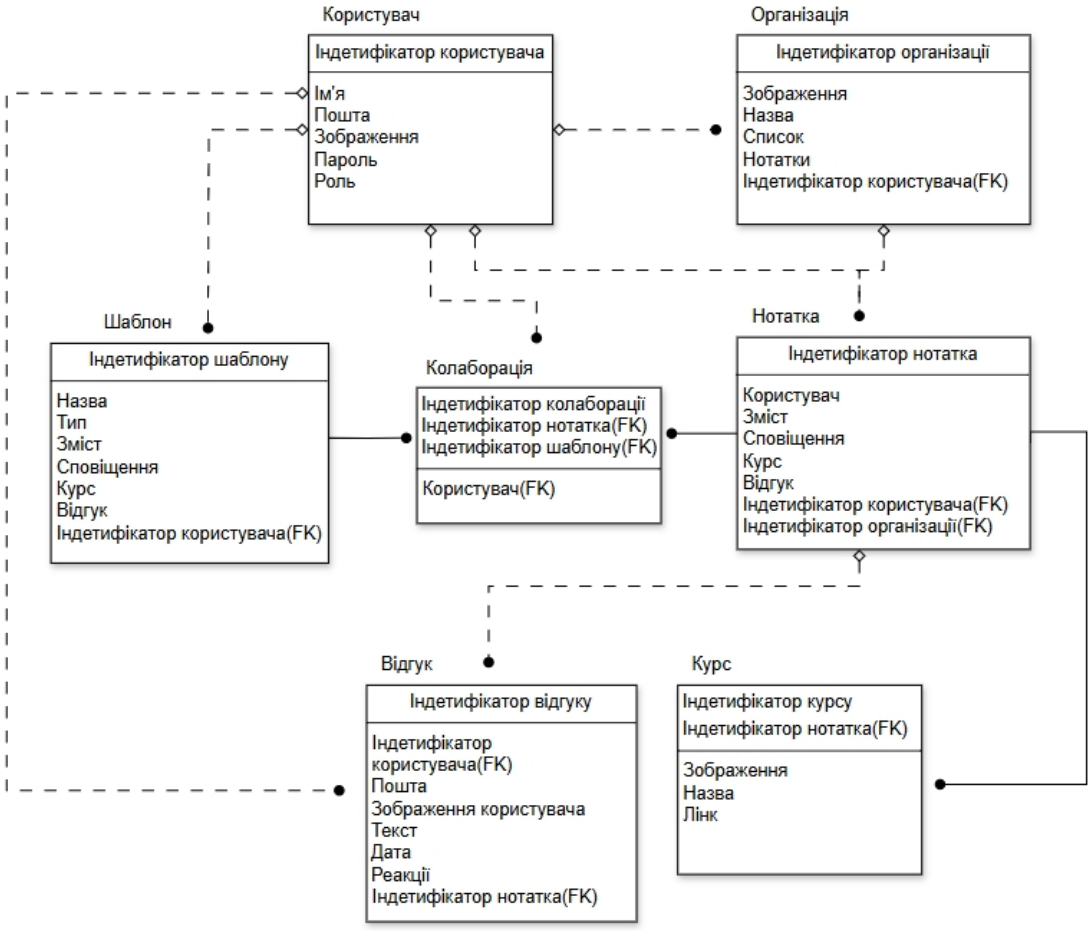

Рис. 2.1. Логічна модель даних у вигляді ER-діаграми

2.2. Діаграма класів та кооперацій

На відміну від діаграми послідовності, на діаграмі кооперації зображуються лише відношення між об’єктами, що відіграють певні ролі у взаємодії. З іншого боку, на цій діаграмі не вказується час у вигляді окремого виміру. Таким чином, послідовність взаємодій і паралельних потоків може бути визначена на діаграмі кооперації за допомогою порядкових номерів для відповідних асоціацій [4].

Зв’язок «Користувач–Нотатка» є неідентифікуючим (1:N), бо кожен користувач може створити багато нотаток, але нотатки мають власні id; зв’язок «Нотатка–Шаблон» теж неідентифікуючий (N:1), оскільки кожна нотатка використовує один шаблон, а один шаблон може застосовуватись у багатьох нотатках; «Користувач–Організація» має тип M:N через, адже користувач може належати до кількох організацій і навпаки; «Нотатка–Папка» — неідентифікуючий (N:1), бо нотатка може міститися в одній папці, а папка може включати багато нотаток; «Користувач–Сесія спільного редагування» (1:N) відображає участь одного користувача в багатьох сесіях, тоді як «Сесія спільного редагування–Нотатка» є ідентифікуючим (1:1), оскільки сесія існує лише разом із конкретною нотаткою.

Діаграма класів (class diagram) служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. Діаграма класів може відбивати різні взаємозв'язки між окремими сутностями предметної області, такими як об'єкти і підсистеми, а також описує їх внутрішню структуру і типи відношень [5].

Першим кроком під час моделювання класів було створено діаграму класів із застосуванням виключно асоціацій, що дозволяє чітко виокремити всі сутності системи та одразу побачити їхні взаємозв’язки. На рисунку 2.2 наведено діаграму класів з асоціативними зв’язками.

Взаємозв’язок «User–Note» показує, що користувач може створювати багато нотаток, а кожна нотатка належить одному користувачу. Ассоціація

«Note–Template» демонструє, що для створення нотатки зазвичай використовується один шаблон, тоді як один шаблон може застосовуватись у багатьох нотатках. Зв’язок «Note–Folder» вказує, що кожна нотатка може бути віднесена до однієї папки для зручного групування, тоді як папка містить багато нотаток.

Далі видно асоціацію «User–Collaboration», що відображає участь одного користувача у багатьох сесіях спільного редагування, а через зв’язок «Collaboration–Note» кожна сесія жорстко прив’язана до однієї конкретної нотатки. Ассоціативний зв’язок «Note–Comment» показує, що до однієї нотатки можна залишати багато коментарів, а кожен коментар належить конкретній нотатці й автору.

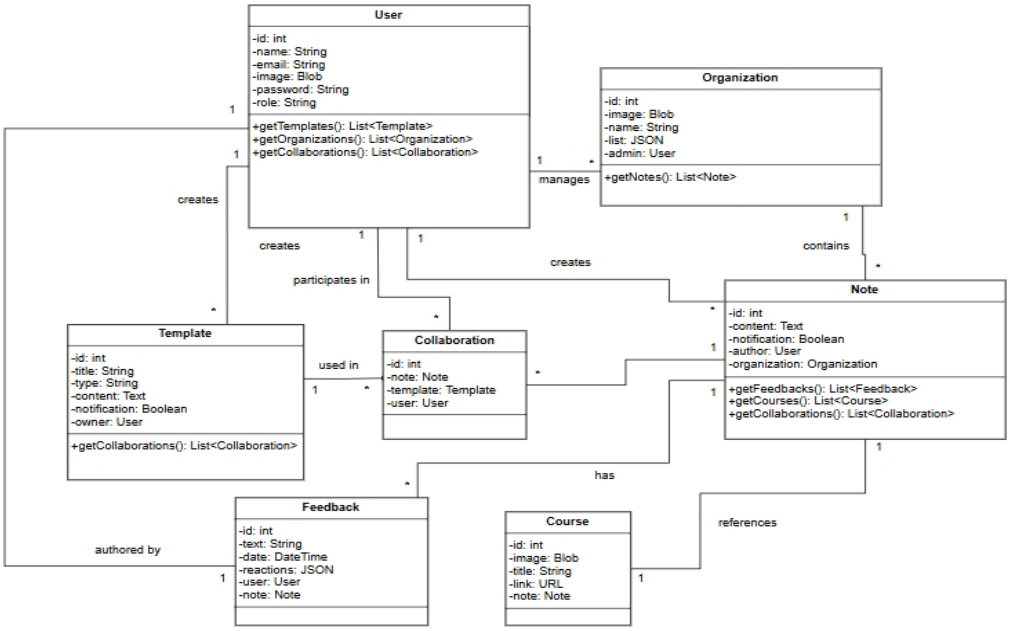

Рис. 2.2. Діаграма класів

2.3. Діаграма пакетів

Діаграма пакетів служить, в першу чергу, для організації елементів в групи за якою-небудь ознакою з метою спрощення структури і організації роботи з моделлю системи [6].

На рис. 2.3 представлено діаграму пакетів, яка відображає архітектуру інформаційної системи NoteMinds, поділену на логічні рівні та функціональні пакети. Така структура дозволяє забезпечити чітку модульність системи, що значно спрощує процеси створення, перевірки та підтримки в майбутньому.

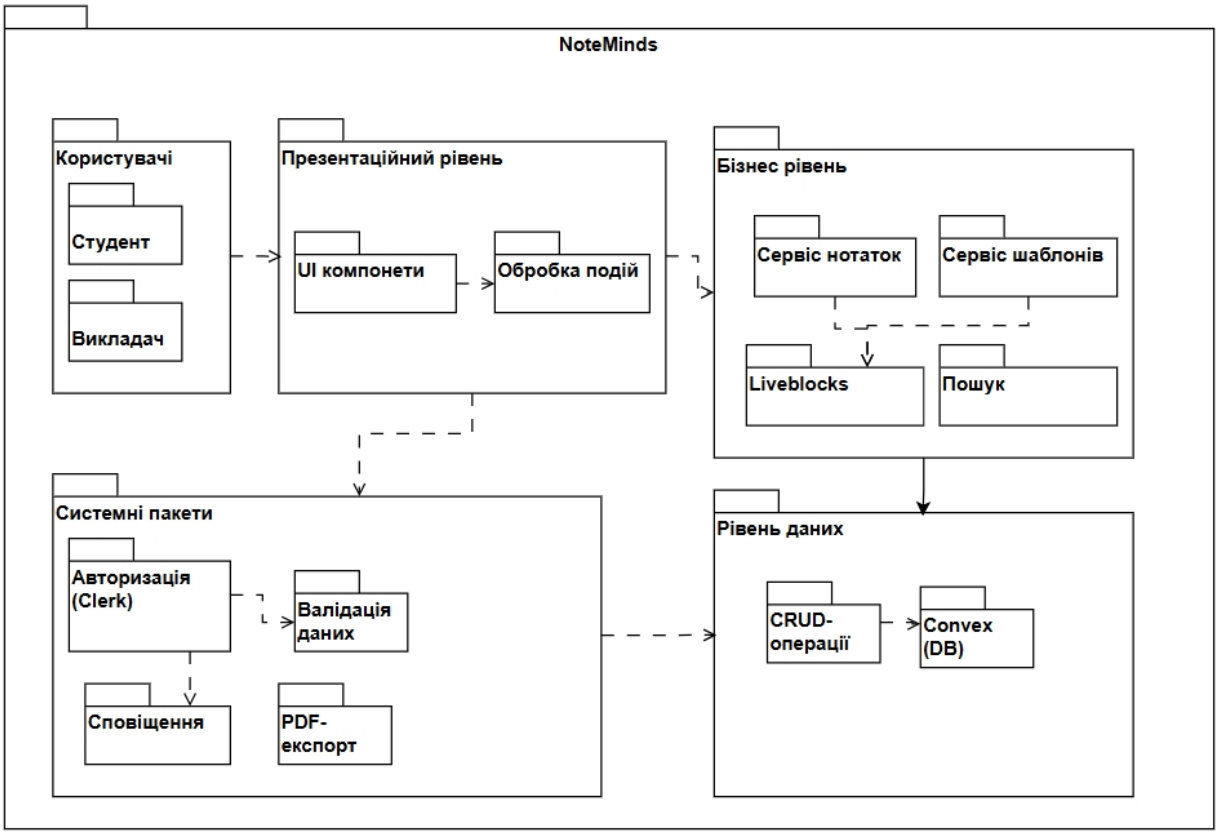

Рис. 2.3. Діаграма пакетів

Рівень презентації забезпечує користувацький інтерфейс і взаємодію з користувачем. Саме через цей рівень студенти, викладачі або адміністратори отримують доступ до основних можливостей: створення та редагування нотаток, перегляд шаблонів, спільна робота та керування особистими даними. Тут реалізована логіка відображення контенту, обробка подій (натискання кнопок, введення тексту тощо) і формування запитів до бізнес-рівня.

Бізнес-рівень містить основну логіку роботи NoteMinds. Тут зосереджені всі сервіси, що забезпечують обробку дій користувачів — створення нотаток, збереження змін, спільна робота у режимі реального часу (за допомогою Liveblocks), керування шаблонами, пошук нотаток, перегляд логів тощо. Саме цей рівень реалізує зв’язок між даними та інтерфейсом, контролюючи правила взаємодії та перевірки доступів.

Рівень даних відповідає за зберігання та роботу з базою даних, яка реалізована через хмарний сервіс Convex. Саме тут зберігаються всі дані нотаток, користувачів, шаблонів, коментарів і логів. Цей рівень надає API для читання, запису, оновлення та фільтрації інформації, забезпечуючи цілісність і послідовність даних.

Системні пакети охоплюють функції авторизації та автентифікації (через Clerk), обробку PDF-експорту, валідацію введених даних, відправку сповіщень і забезпечення безпеки доступу. Ці компоненти інтегруються на всіх рівнях і виконують ключову роль у підтримці стабільності та безпеки системи.

Завдяки такій архітектурі система NoteMinds є легкою для супроводу, придатною до розширення функціоналу та забезпечує надійну платформу з метою використання в освітній сфері матеріалами в інтерактивному форматі.

2.4. Діаграма компонентів

Діаграма компонентів призначена для вивчення складу компонентів майбутнього ПЗ та вказівки послідовності компіляції та збірки окремих модулів [7].

Компонент являє собою фізичний модуль програмного коду. Компонент часто вважають синонімом пакету, але ці поняття можуть відрізнятися, оскільки компоненти являють собою фізичне об'єднання програмного коду [8].

Діаграма компонентів дозволяє розробнику перевірити, чи може відповідний набір артефактів правильно реалізувати функціонал системи, забезпечуючи тим самим її введення в експлуатацію [9].

У процесі проєктування NoteMinds було побудовано діаграму компонентів (рис. 2.4), яка ілюструє основні структурні елементи системи та шляхи їхньої взаємодії. Ця модель допомагає закласти гнучку, масштабовану й легко підтримувану архітектуру, розділивши додаток на чіткі функціональні блоки.

На рівні представлення (Presentation) зосереджені React-компоненти й сторінки Next.js, через які кінцевий користувач взаємодіє із системою. Сюди входять компоненти на зразок Note (відображення переліку нотаток із прев'ю), Templates (вибір готових шаблонів), SearchBar.jsx (пошукова панель за ключовими словами), NotificationPanel.jsx (список сповіщень), Organization (керування організацією й ролями). Всі вони реалізовані з використанням Tailwind CSS і забезпечують обробку подій користувача, побудову запитів до бізнес-рівня та відображення результатів.

Рівень бізнес-логіки (Application) представлений Next.js API-роутами й сервісними модулями, які централізовано обробляють запити клієнта та координують взаємозв'язок між елементами даних і презентації. Наприклад, /api/notes відповідає за створення, оновлення й видалення нотаток; /api/templates — за надання списку доступних шаблонів; /api/search — за пошук по вмісту нотаток; /api/notifications — за генерацію та доставку сповіщень. Усі ці контролери інтегруються з Convex SDK для збереження й читання даних, викликають служби Liveblocks для ініціалізації сесій спільного редагування та використовують Clerk для перевірки прав і автентифікації.

Рівень інфраструктури, що забезпечує доступ до даних містить компоненти, що безпосередньо працюють із зовнішніми сервісами. Сюди належать ConvexClient (обгортує виклики до Convex-функцій), ClerkAuthService (забезпечує реєстрацію й вхід через Clerk) та LiveblocksClient

(ініціалізує й синхронізує сесії спільного редагування). Ці пакети гарантують цілісність і послідовність даних, обробку помилок на рівні з’єднань та зберігають єдину точку доступу до сторонніх API.

Така структура компонентів дозволяє чітко відокремити UI-частину від бізнес-логіки й інфраструктурних залежностей, що полегшує створення нових функцій, полегшує автоматизоване тестування та забезпечує можливість масштабування системи у майбутньому.

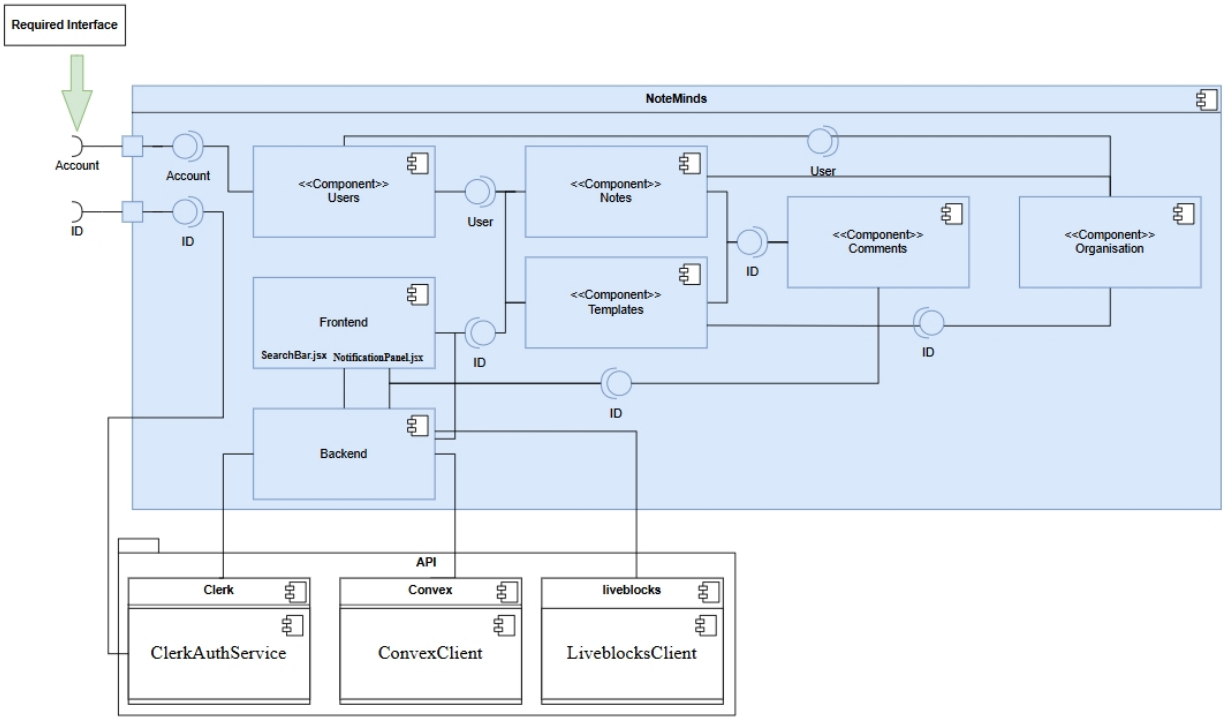

Рис. 2.4. Діаграма компонентів

2.5. Висновок до розділу 2

Проєктування інформаційного та програмного забезпечення для інформаційної системи ведення нотаток для навчання заклало міцний фундамент для створення стабільного та функціонального продукту. Під час розробки ER-діаграми були чітко зафіксовані основні сутності — User, Note, Template, Folder, Comment, LiveSession — разом із їхніми атрибутами та типами зв’язків. Це дозволило спроектувати логічну модель даних з урахуванням цілісності та відсутності дублювання інформації.

Діаграми класів доповнили ER-модель, описавши статичні аспекти системи: класи, їх властивості (id, title, content, authorId тощо) і методи (створення, редагування, пошук, експорт у PDF). Такий підхід допоміг узгодити структуру коду з моделлю даних і спростив подальшу реалізацію бізнес-логіки.

Діаграми кооперацій (collaboration diagrams) показали, як об’єкти-екземпляри цих класів взаємодіють під час типових сценаріїв: створення нотатки, ініціація спільного редагування через Liveblocks, додавання коментаря та надсилання сповіщень. Вони продемонстрували зв’язки між NoteEditor, ConvexClient, LiveblocksClient і NotificationService, що забезпечує прозору комунікацію в системі.

Модульний поділ компонентів на Presentation (React/Next.js), Application (API-роути, сервісні модулі для бізнес-логіки) та Infrastructure (ConvexClient, ClerkAuthService, LiveblocksClient) гарантує чітке розмежування відповідальностей. Додаткові глобальні модулі — для валідації даних, обробки помилок та безпеки — інтегруються на всіх рівнях. Така архітектура підтримує масштабованість, полегшує тестування та майбутнє розширення системи NoteMinds.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Система управління інформаційною базою

У сучасних вебдодатках вибір системи керування базами даних (СУБД) належить до основних аспектів архітектурних рішень, оскільки від цього залежить швидкість обробки, цілісність та масштабованість даних. Існують два основні підходи до зберігання інформації: реляційні СУБД, які оперують жорстко структурованими таблицями з чітко визначеними зв’язками «один-до-багатьох» та «один-до-одного», і документо-орієнтовані бази, що дозволяють працювати з гнучкими, напівструктурованими записами (зокрема у форматі JSON). Реляційні системи, як-от PostgreSQL чи MySQL, традиційно забезпечують високу узгодженість даних, підтримку складних SQL-запитів і нормалізацію до третьої форми, але можуть потребувати більше ресурсів для горизонтального масштабування та забезпечення синхронізації в реальному часі. Документо-орієнтовані БД, наприклад MongoDB, надають гнучкість схеми й ефективно обробляють великі об’єми динамічної інформації, проте можуть поступатися реляційним у гарантіях транзакційної цілісності.

Для програмного забезпечення інформаційної системи ведення нотаток для навчання, де критичною є миттєва синхронізація нотаток між користувачами та відсутність конфліктів при спільному редагуванні, обрано хмарну платформу Convex. Convex поєднує переваги документно-орієнтованої моделі (зберігання JSON-подібних об’єктів із гнучкою схемою) і вбудовані механізми реального часу для автоматичної синхронізації даних, що значно спрощує розробку серверної логіки на Node.js та забезпечує високу продуктивність і масштабованість без складного налаштування кластерів. Такий вибір дозволяє уникнути дублювання даних, зберегти цілісність інформації та надати користувачам можливість працювати з нотатками одночасно та без затримок.

Серед сучасних реляційних СУБД виділяють низку провідних рішень, що відрізняються функціональністю та особливостями реалізації. Таблиця 3.1 містить порівняльний огляд основних представників.

Таблиця 3.1 – Порівняльний аналіз реляційних СУБД

СУБД	**Переваги**	**Недоліки**
PostgreSQL	Повна підтримка ACID-транзакцій Розширені аналітичні та JSON-функції Велика спільнота й відкрите ПЗ PL/pgSQL	Складніше адміністрування Вища ресурсомісткість Повільніше, ніж прості СУБД, на операціях лише читання
MySQL	Швидкість читання й проста інсталяція Підтримка InnoDB, MyISAM Широка екосистема	Обмежені аналітичні можливості Відмінності в реалізації SQL-стандарту Деякі функції вимагають платних версій
Microsoft SQL Server	Інтеграція з Azure/.NET Потужні інструменти адміністрування BI-функції	Висока вартість ліцензій Enterprise Традиційно орієнтований на Windows
SQLite	Zero-configuration, вбудована Висока швидкість для невеликих обсягів Ідеальна для	Обмежена паралельність запису (блокування всієї БД) Відсутність клієнт-серверної архітектури

	прототипів	
Convex	Документно-орієнтована модель з гнучкою схемою Вбудована синхронізація в реальному часі Простота налаштування кластерів	Менше можливостей для складних SQL-запитів Залежність від зовнішнього сервісу та його SLA

У контексті нашої теми — програмного забезпечення інформаційної системи ведення нотаток для навчання — у якості СУБД обрана Convex, хмарна платформа з документно-орієнтованою моделлю даних і вбудованими механізмами реального часу.

Convex дозволяє зберігати нотатки як JSON-подібні об’єкти, автоматично синхронізувати зміни між клієнтами без додаткових налаштувань кластера та гарантує низьку затримку при одночасному редагуванні. Це робить її ідеальною для нашої задачі, де критично важлива миттєва кооперація й консистентність даних.

3.2. Розробка інформаційної бази

Розробка інформаційної бази для програмного забезпечення інформаційної системи ведення нотаток для навчання починається з опису схеми даних у Convex (файл schema.ts). Що визначає колекцію documents (аналог таблиці в реляційній БД):

Рис. 3.1. Таблиці в реляційній БД NoteMinds

Першим кроком було визначення базової структури даних у файлі schema.ts, де описано колекції documents, comments, notifications, memberships тощо. Кожна колекція отримує власний первинний ключ (id) і набір атрибутів із явно зазначеними типами:

- documents: поля title, initialContent, ownerId, roomID, organizationId
- comments: поля documentId, authorId, text, createdAt
- notifications: поля userId, type, payload, createdAt
- memberships: поля userId, organizationId, role

Для забезпечення швидкого та ефективного користування даними у схему додано індекси:

- .index("by_owner_id", ["ownerId"]) для миттєвого вибору нотаток за власником;
- .index("by_organization_id", ["organizationId"]) для групового доступу в межах організації;
- .searchIndex("search_title", { searchField: "title", filterFields: ["ownerId", "organizationId"] }) для повнотекстового пошуку за заголовками з фільтрацією.

З метою гарантування цілісності та безпеки даних реалізовано декларативні правила доступу (security rules). Наприклад, лише власник нотатки або член відповідної організації може читати документ, а змінювати його може тільки автор:

```
allow("documents", {
  read: ({ auth, doc }) =>
    auth.userId === doc.ownerId || auth.userOrgIds.includes(doc.organizationId),
  write: ({ auth, doc }) =>
    auth.userId === doc.ownerId
});
```

Рис. 3.2. Фрагмент правила доступу до документів у базі даних Convex — дозволи на читання і запис залежно від користувача та організації.

Додавання коментарів та створення сповіщень відбувається через серверні «watchers»: при вставці нового запису в comments спрацьовує функція, яка автоматично формує відповідний запис у notifications та надсилає WebSocket-подію через Liveblocks. Це еквівалентно використанню тригерів у реляційних СУБД для каскадної обробки змін.

Для реалізації бізнес-логіки синхронізації створені мутації і запити:

- **createDocument** — вставка нового документа в documents;
- **updateDocument** — оновлення полів title, initialContent та updatedAt;
- **deleteDocument** — видалення документа з автоматичним прибиранням зв’язаних коментарів;

- **searchDocuments** — індексований пошук за ключовими словами у заголовку й вмісті.

Нарешті, для керування ролями в організаціях використовується колекція memberships, що реалізує зв’язок “багато-до-багатьох” між користувачами та групами. Перевірка прав доступу до нотаток та функцій відбувається через спеціальну серверну утиліту checkAccess, яка звертається до memberships і повертає результат:

```
export function checkAccess(db, userId, document) {
  return db.table("memberships")
    .filter(m => m.userId === userId && m.organizationId === document.organizationId)
    .first() !== null;
}
```

Рис. 3.3. Перевірка доступу користувача до документа враховуючи його статус учасника в організації.

Таким чином, розробка інформаційної бази охопила всі етапи: від визначення структури колекцій і індексів до налаштування безпеки, автоматичних сповіщень і серверних механізмів обробки даних. Це забезпечує високу продуктивність, цілісність та масштабованість системи ведення нотаток для навчання.

3.3. Вибір інструментарію для створення прикладного програмного забезпечення

Розробка програмного забезпечення інформаційної системи ведення нотаток для навчання передбачає ретельний аналіз технологічного стеку, який гарантує високу продуктивність, масштабованість, безпеку та зручність як для кінцевих користувачів, так і для команди розробників. Оскільки система повинна підтримувати інтенсивну асинхронну синхронізацію даних, багатий клієнтський інтерфейс і гнучку серверну логіку, було розглянуто кілька підходів.

Першим кандидатом був Django на Python: він добре підходить для швидкого створення MVP за рахунок «із коробки» наявних компонентів

(ORM, адмінка, аутентифікація), але під високими навантаженнями та при необхідності реального часу складно досягти низьких латенцій без значної кастомізації. Spring Boot на Java забезпечує корпоративну надійність і масштабованість, але вимагає залучення досвідчених розробників і значного часу на налаштування проєкту. ASP.NET Core із C# і Razor Pages також розглядався як потужне рішення з чітким розділенням рівнів MVC, проте впровадження та підтримка .NET-інфраструктури для невеликої освітньої платформи могли виявитися надмірними.

Врахувавши всі фактори — величину очікуваних одночасних сесій редагування, необхідність безшовної інтеграції з хмарними сервісами, мінімальну затримку при синхронізації та швидкий темп розробки — було обрано JavaScript-стек на базі Next.js 15 та React.

- Next.js 15 забезпечує універсальне рендерення (SSR/SSG), автоматичний розподіл коду й надшвидкий час відгуку, працюючи поверх Node.js. Це дозволяє мати єдину мову на фронтенді і бекенді, спрощуючи розгортання та підтримку.
- React із Tailwind CSS надає швидку розробку інтуїтивного та адаптивного UI, де компоненти легко повторно використовувати та стилізувати за допомогою утилітарних класів.
- Node.js як середовище виконання гарантує неблокуючу обробку запитів і чудово підходить для асинхронних операцій з Convex і Liveblocks.

З метою забезпечення збереження та синхронізації даних у реальному часі обрана хмарна платформа Convex, яка поєднує документно-орієнтовану модель даних із вбудованими механізмами реплікації й повідомлень. Це знімає потребу у власній інфраструктурі СУБД та кластерному адмініструванні.

Clerk виконує роль сервісу авторизації та аутентифікації, надаючи готові UI-компоненти та SDK для управління сесіями користувачів із ролями Student, Lecturer, Administrator.

Liveblocks інтегрується на клієнті та сервері для організації спільного редагування нотаток у реальному часі: його SDK дозволяє бачити курсори інших учасників і миттєво відображати зміни в документі.

Остаточний стек для розробки:

- Frontend: Next.js 15 + React + Tailwind CSS
- Backend/Runtime: Node.js (Next.js API Routes)
- Database & Sync: Convex
- Auth & User Management: Clerk
- Real-Time Collaboration: Liveblocks
- Deployment: Vercel

Отже, такий набір інструментів забезпечить швидку розробку, плавну інтеграцію всіх компонентів та можливість подальшого масштабування й розвитку інформаційної системи ведення нотаток для навчання.

3.4. Алгоритмізація та програмування програмних модулів

Програмування – це складний, творчий інженерний процес, неможливий без застосування відповідних технологій та теоретичних знань [10].

Авторизація користувачів у програмному забезпеченні інформаційної системи ведення нотаток для навчання забезпечується через сервіс Clerk, що спрощує реалізацію безпечного входу та керування сесіями.

Процес авторизації починається на сторінці входу, де користувач зазначає адресу електронної пошти й пароль. Після кліку кнопки «Увійти» клієнтська частина виконує перевірку на заповненість полів: якщо будь-яке з полів порожнє, виводиться повідомлення про необхідність заповнити обидва поля.

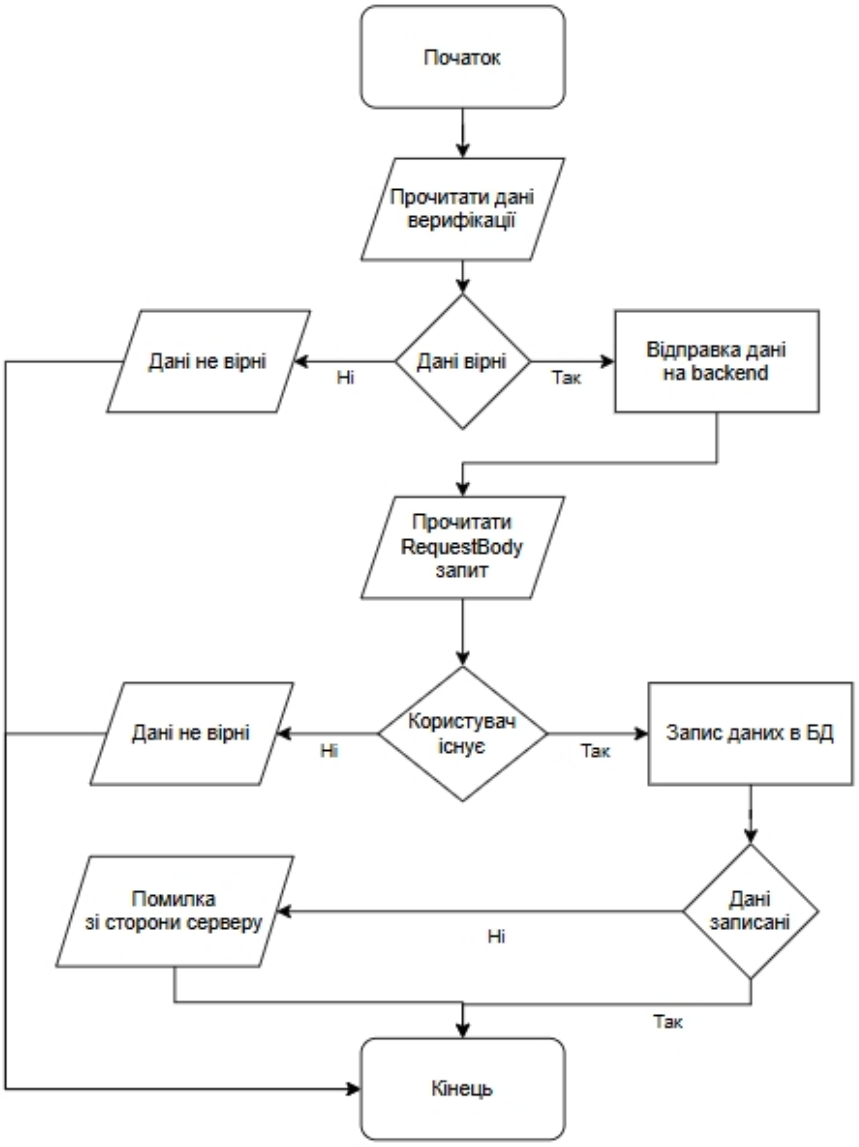

Рис. 3.4. Блок-схема алгоритму авторизації

Якщо обидва поля заповнені, браузер надсилає запит до API Clerk для перевірки достовірності облікових даних. Clerk перевіряє, чи існує користувач із такою поштою, і порівнює переданий пароль із хешованим значенням у своїй базі.

У разі невідповідності або відсутності облікового запису Clerk повертає помилку «Invalid credentials», і інтерфейс відображає повідомлення «Неправильна електронна адреса або пароль».

Якщо ж дані коректні, Clerk генерує сесію, встановлює відповідні HTTP-кукі із токенами доступу та оновлення, і повертає клієнту об’єкт користувача з його userId та ролями. Після цього фронтенд автоматично перенаправляє авторизованого користувача на головну панель (Dashboard), де доступні функції створення й редагування нотаток, перегляд сповіщень та пошук.

У разі успішної авторизації додатково виконується виклик локальної утиліти checkAccess, яка підтверджує, що користувач має необхідні ролі (Student, Lecturer) для доступу до відповідних розділів системи. Якщо роль не дозволяє доступ, система перенаправляє користувача на сторінку з повідомленням «Доступ заборонено».

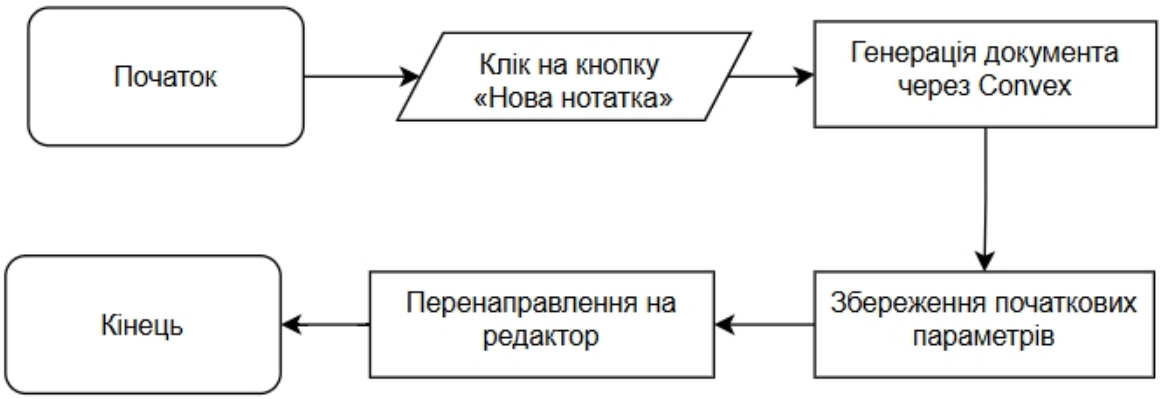

Рис. 3.5. Блок-схема алгоритму створення нової нотатки

Опис кроків до рис.3.5:

Початок: Алгоритм запускається, коли користувач клікає на кнопку «Нова нотатка».

Генерація документа через Convex: Система створює новий документ у базі даних за допомогою Convex (сервісу для роботи з даними). На цьому етапі ініціалізується унікальний ідентифікатор документа.

Збереження початкових параметрів: Документу присвоюються стартові параметри (наприклад: дата створення, автор, заголовок за замовчуванням, права доступу).

Перенаправлення на редактор: Користувач автоматично переходить на сторінку редактора, де може редагувати нову нотатку.

Кінець: Алгоритм завершується після успішного відкриття редактора.

Мета: Забезпечити швидке створення документа зі стандартними налаштуваннями та перехід до його редагування.

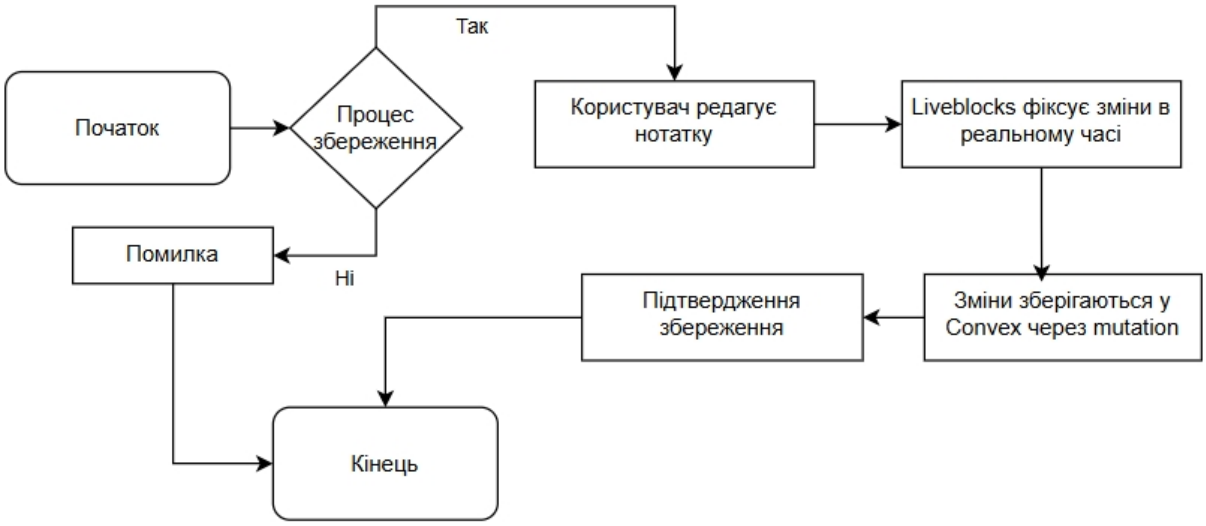

Рис. 3.6. Блок-схема алгоритму збереження змін у нотатці

Опис кроків до рис.3.6:

Початок: Алгоритм активується, коли користувач вносить зміни в текст або структуру нотатки.

Liveblocks фіксує зміни в реальному часі: Бібліотека Liveblocks відстежує та фіксує кожну дію користувача (наприклад, введення тексту, форматування) без необхідності ручного збереження.

Зміни зберігаються у Convex через mutation: Зафіксовані зміни передаються до серверної частини через Convex-мутацію (функцію для оновлення даних). Convex забезпечує атомарність операції, щоб уникнути часткового збереження або конфліктів.

Підтвердження збереження: Система отримує підтвердження від Convex про успішне збереження змін. Користувачеві може в

Кінець: Алгоритм завершується після успішного оновлення даних.

Мета: Гарантувати надійне збереження змін у режимі реального часу без ризику втрати даних.

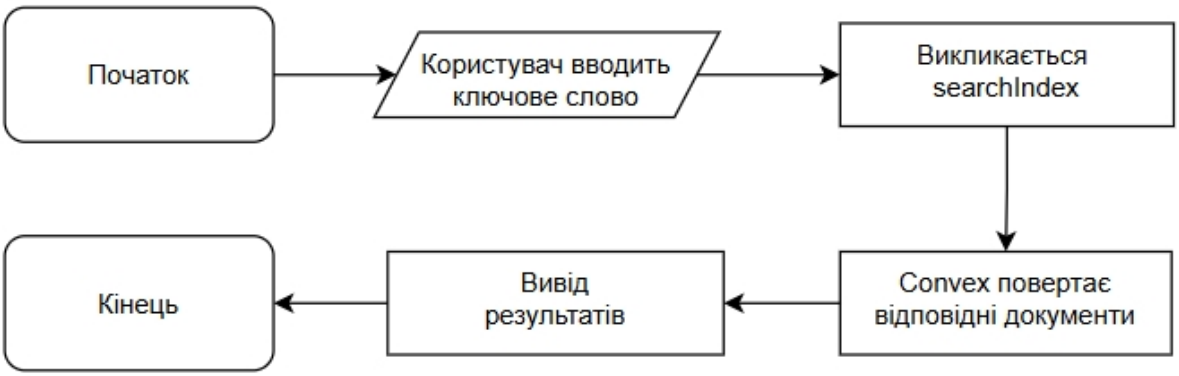

Рис. 3.7. Блок-схема алгоритму пошуку нотаток

Опис кроків до рис.3.7:

Початок: Алгоритм активується, коли користувач вводить ключове слово в поле пошуку.

Викликається searchIndex: Система використовує пошуковий індекс (наприклад, оптимізовану структуру даних або зовнішній сервіс), щоб швидко знайти документи, що відповідають запиту.

Convex повертає відповідні документи: Запит передається до бази даних через Convex, який фільтрує документи за ключовим словом і повертає результат (наприклад, список нотаток зі збігами у заголовку або тексті).

Вивід результатів: Знайдені нотатки відображаються користувачеві у вигляді списку або карток.

Кінець: Алгоритм завершується після відображення результатів.

Мета: Забезпечити миттєвий пошук інформації з мінімальними затримками.

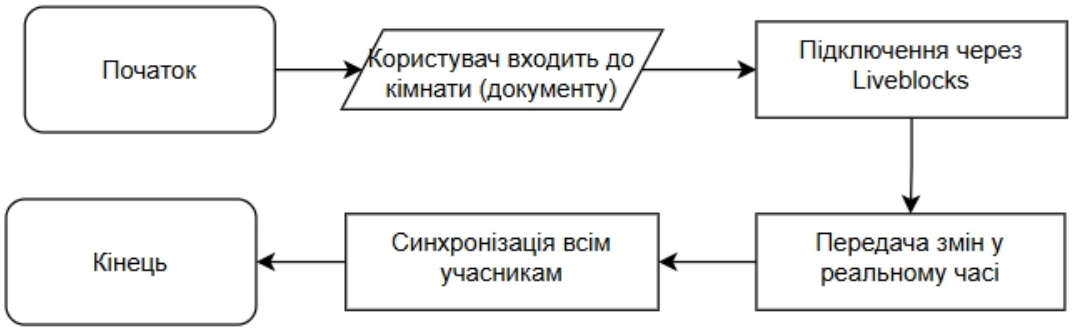

Рис. 3.8. Блок-схема алгоритму спільної роботи над нотаткою

Опис кроків до рис.3.8:

Початок: Алгоритм запускається, коли користувач відкриває спільний документ або вводить ідентифікатор кімнати.

Підключення через Liveblocks: Система встановлює з’єднання з сервісом Liveblocks, який створює або приєднується до "кімнати" (віртуального простору для спільної роботи). Користувач отримує доступ до документу.

Передача змін у реальному часі:

Кожна дія користувача (наприклад, редагування тексту, додавання коментарів) фіксується Liveblocks.

Зміни упаковуються у формат, сумісний із системою (наприклад, операційні трансформації для уникнення конфліктів).

Синхронізація всім учасникам:

Liveblocks надсилає зміни до серверу Convex для збереження.

Оновлення транслюються усім підключеним користувачам через WebSocket або інший протокол реального часу.

Кінець: Процес триває, доки користувач не залишить кімнату.

Мета: Забезпечити синхронну роботу декількох користувачів над документом без конфліктів версій.

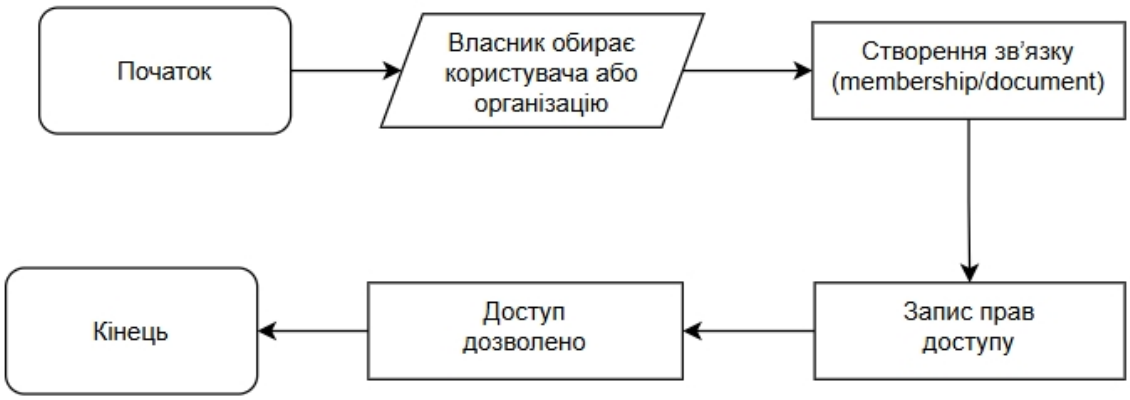

Рис. 3.9. Блок-схема алгоритму надання доступу до нотатки

Опис кроків до рис.3.9:

Початок: Алгоритм запускається, коли власник нотатки вирішує надати доступ іншому користувачу або організації.

Вибір користувача/організації:

Власник вказує електронну пошту, ім’я або ID користувача/організації через інтерфейс (наприклад, випадаючий список або поле пошуку).

Створення зв’язку:

Система генерує зв’язок між документом і об’єктом доступу (наприклад, через таблицю membership у базі даних Convex, де зберігаються зв’язки document_id ↔ user_id).

Запис прав доступу:

Визначаються рівні прав (наприклад: read, write, admin).

Параметри зберігаються у відповідному полі зв’язку (наприклад, поле permissions у таблиці membership).

Доступ дозволено:

Система підтверджує операцію, і обраний користувач/організація отримує доступ до нотатки.

Користувачеві може надіслатися сповіщення або запрошення.

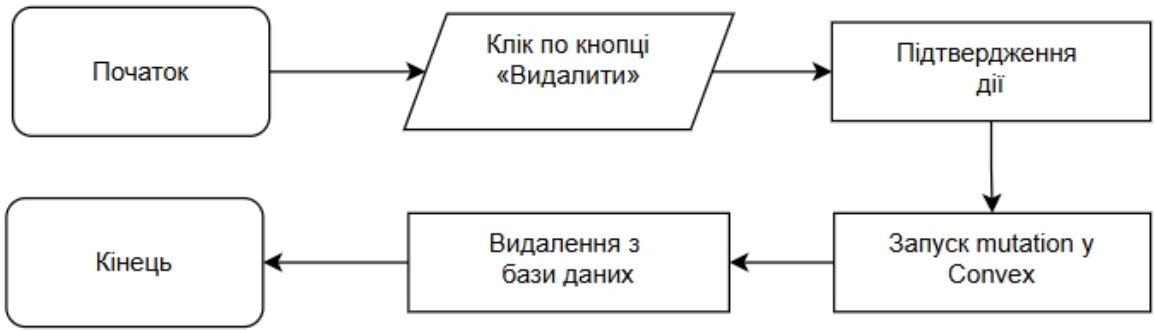

Рис. 3.10. Блок-схема алгоритму видалення нотатки

Опис кроків до рис.3.10:

Початок: Алгоритм активується, коли користувач натискає кнопку «Видалити» в інтерфейсі нотатки.

Підтвердження дії:

Система виводить модальне вікно з попередженням ("Ви впевнені?").

Користувач підтверджує або скасовує операцію.

Запуск mutation у Convex:

Після підтвердження викликається мутація Convex, яка відповідає за видалення даних.

Мутація перевіряє права користувача (чи є він власником або адміном).

Видалення з бази даних:

Запис нотатки видаляється з таблиці документів у Convex.

Додатково можуть очищуватись зв’язки (наприклад, у таблиці membership).

Оновлення інтерфейсу:

Сторінка нотатки закривається, а список документів оновлюється (видалена нотатка зникає).

Користувач отримує сповіщення про успішне видалення.

Мета:

Забезпечити безпечне та контрольоване видалення даних, уникнувши випадкових або несанкціонованих дій.

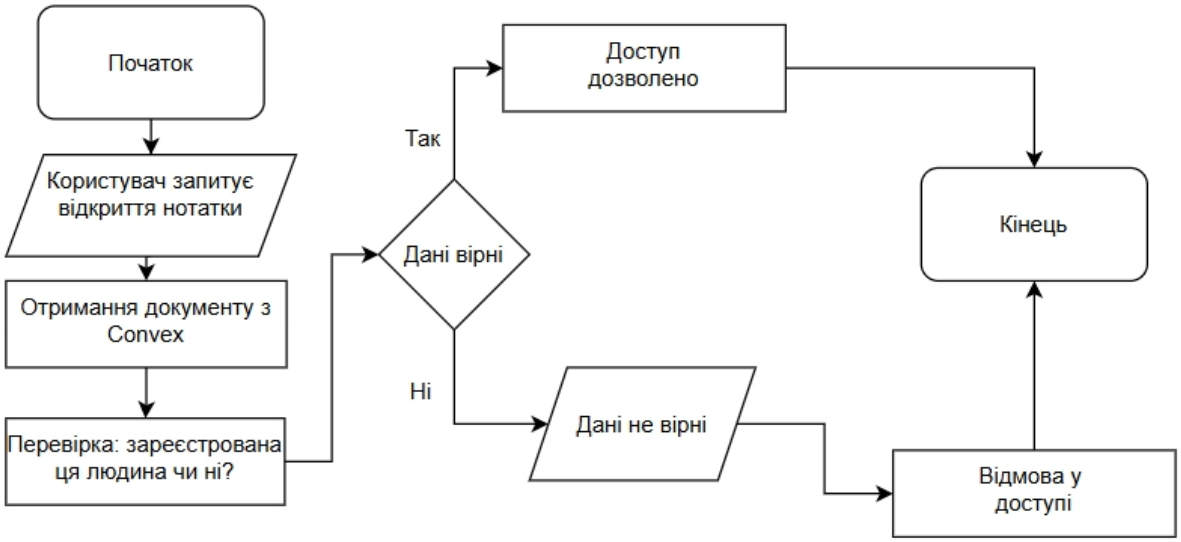

Рис. 3.11. Блок-схема алгоритму перевірки доступу до документу

Опис кроків до рис.3.11:

Початок: Алгоритм активується, коли користувач намагається відкрити нотатку (наприклад, через посилання або зі списку документів).

Отримання документу з Convex:

Система виконує запит до бази даних Convex, щоб отримати метадані нотатки (ідентифікатор документа, права доступу).

Перевірка:

Чи є користувач зареєстрованим у системі (наявність user_id у базі користувачів).

Дозвіл/відмова:

Якщо користувач зареєстрований і має відповідні права на нотатку, він отримує доступ.

Якщо користувач не зареєстрований або не має прав — система показує помилку "Доступ заборонено".

Мета:

Забезпечити безпеку даних, дозволяючи доступ лише авторизованим користувачам.

3.5. Висновки до розділу 3

Розробка інформаційної системи ведення нотаток для навчання складалася з таких основних етапів:

Проєктування та реалізація інформаційної бази з використанням Convex — визначення колекцій documents, comments, memberships, налаштування індексів і пошукових індексів, а також правил безпеки для даних.

Створення серверної частини на основі Next.js API Routes та Node.js — реалізація CRUD-операцій, логіки синхронізації (Liveblocks), аутентифікації й авторизації (Clerk), серверних «watchers» для сповіщень та функції checkAccess для контролю доступу.

Створення клієнтського інтерфейсу з React і Tailwind CSS — побудова компонентів для редактора нотаток, списку документів, пошукової панелі, панелі сповіщень, сторінок профілю й організацій, реалізація темної/світлої теми.

Реалізація основних функцій: реєстрація й вхід користувачів, створення, редагування та видалення нотаток із миттєвим збереженням, спільна робота в реальному часі, пошук за заголовками й вмістом, експорт у PDF, коментування та система сповіщень.

Використання документно-орієнтованої СУБД Convex забезпечило гнучкість схеми та автоматичну синхронізацію даних, а інтеграція з Clerk гарантувала надійну безпеку облікових записів і хешування паролів. Компонент Liveblocks дозволив реалізувати безшовне спільне редагування без додаткових налаштувань серверів. Завершення цих етапів створило стабільний, захищений та масштабований продукт, готовий до інтеграції в освітні процеси та подальшого розвитку.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУТАЦІЇ СИСТЕМИ

4.1. Тестування системи

Тестування інформаційної системи ведення нотаток для навчання складалося з кількох взаємопов’язаних етапів, кожен із яких покликаний гарантувати якість, надійність та відповідність розробленого рішення вимогам користувачів.

Насамперед було проведено модульне тестування окремих компонентів і сервісів. Для перевірки коректності роботи React-компонентів (наприклад, редактора нотаток, списку документів чи пошукової панелі) використані Jest та React Testing Library. Паралельно у бекенді на Next.js тестувалися серверні функції — мутації і запити Convex — за допомогою моків, що імітують базу даних. Це дозволило впевнитися, що кожна одиниця коду виконує свою задачу без помилок і витоку даних.

Далі провели інтеграційне тестування, яке охопило взаємодію між клієнтом, сервером і зовнішніми сервісами (Clerk для аутентифікації й Liveblocks для синхронізації). За допомогою Mock Service Worker (msw) ми відтворювали відповіді API та перевіряли, як в інтерфейсі обробляються успішні та помилкові сценарії: створення нотатки, пошук по заголовку, обмін коментарями й авторизація користувача.

End-to-End (E2E) тестування здійснювалося в Cypress: з імітацією реальної сесії користувача перевіряли весь життєвий цикл нотатки — від реєстрації й входу до спільного редагування в двох браузерах одночасно. Ці сценарії підтвердили, що дані коректно зберігаються в Convex, а Liveblocks синхронізує зміни без затримок.

Окремо провели тестування продуктивності сервісних маршрутів та інтерфейсу. За допомогою k6 ми навантажували API -— одночасно надсилали сотні запитів на створення й оновлення нотаток, вимірюючи час відповіді та стабільність роботи. На клієнті із Lighthouse перевіряли швидкість рендерингу сторінок і час першого відгуку, прагнучи забезпечити відчутний користувачем комфорт навіть при повільному інтернеті.

Не менш важливим було тестування безпеки. Використовуючи OWASP ZAP, ми сканували додаток на вразливості: перевіряли захист від XSS у текстових полях, CSRF-атаки на API й налаштування CORS. Для аналізу залежностей застосували Snyk — це допомогло виявити застарілі пакети й усунути потенційні ризики.

Насамкінець відбулося приймальне тестування за участі кількох студентів і викладачів. У їхніх руках інтерфейс NoteMinds виявив себе зрозумілим і зручним: вони створювали нотатки, працювали в реальному часі та залишали коментарі, підтверджуючи, що система відповідає заявленим цілям і справді полегшує навчальний процес.

Завдяки комплексному підходу й автоматизації більшості перевірок нам вдалося досягти високого рівня якості: усі критичні функції працюють стабільно, а нефункціональні вимоги (продуктивність, безпека, зручність) задоволені з запасом.

На рисунку 4.1 зображено інтерфейс створення нової нотатки в системі NoteMinds. Користувач має змогу ввести заголовок нотатки, додати текстовий вміст, а також скористатися панеллю інструментів для форматування. У верхній частині присутні кнопки для збереження нотатки, скасування редагування та додавання співавторів. Такий інтерфейс забезпечує зручність введення інформації та швидкий доступ до основних функцій редагування.

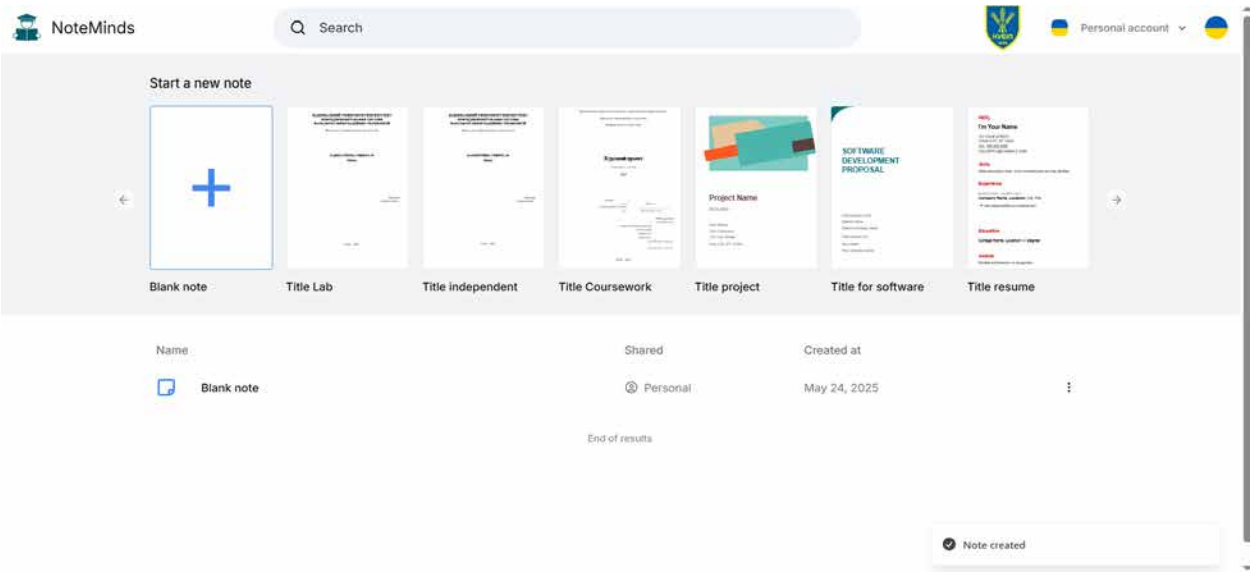

Рис. 4.1. Створення нотатки в системі

Рисунок 4.2 демонструє процес взаємодії користувача з уже створеною нотаткою. Тут відображено відкрите вікно нотатки, де видно її вміст, можливість редагування, перегляду змін у реальному часі (за допомогою Liveblocks), а також коментування або додавання тегів. У правій частині інтерфейсу розміщено панель учасників, яка відображає інших користувачів, що наразі працюють над цією ж нотаткою.

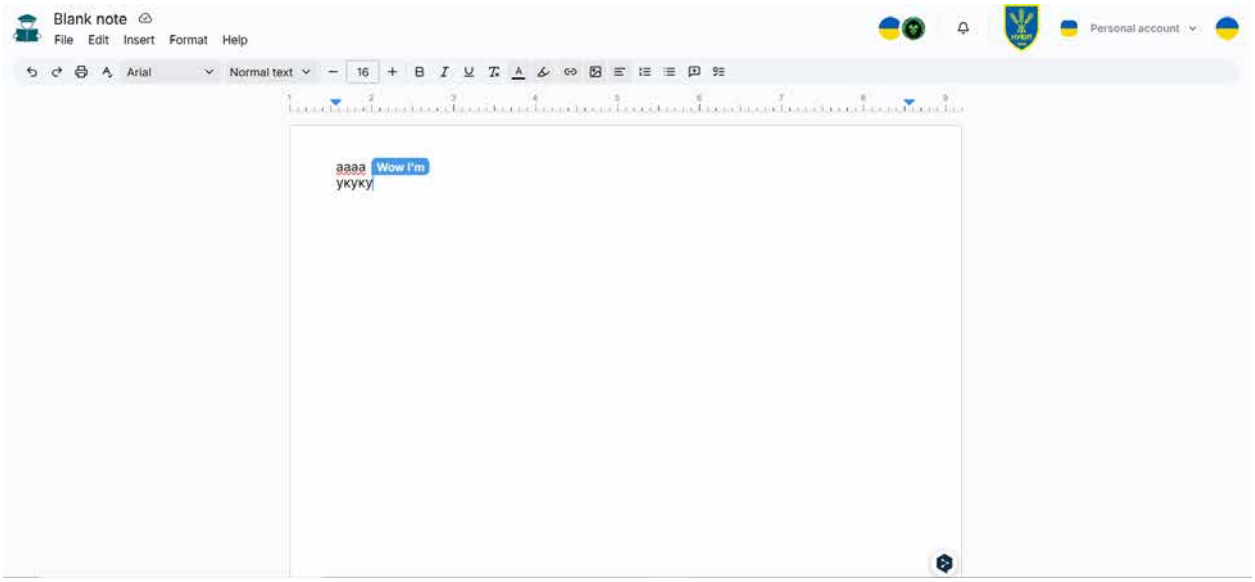

Рис. 4.2. Робота з нотаткою

4.2. Вимоги до апаратного та програмного забезпечення

Щоб забезпечити стабільну роботу інформаційної системи ведення нотаток для навчання, необхідно заздалегідь визначити мінімальні й рекомендовані вимоги до обладнання та програмного забезпечення. Це дозволить гарантувати плавне функціонування, швидкий відгук інтерфейсу та коректну обробку запитів, а також спростить підготовку середовища для впровадження і подальшого супроводу системи.

Огляд діаграми розміщення

На діаграмі розміщення (рис. 4.1) показані ключові вузли, на яких виконуються різні частини NoteMinds:

- Клієнтський вузол (Browser) — персональні комп’ютери, планшети або смартфони користувачів, що завантажують статичні файли через CDN і встановлюють WebSocket-з’єднання з Liveblocks.
- CDN / Static Hosting — Vercel Edge, яке роздає бандли React/Next.js та стилі Tailwind CSS.
- Serverless Functions — Next.js API Routes, розгорнені на безсерверних вузлах Vercel, виконують бізнес-логіку та звертаються до Convex, Clerk і Liveblocks.
- Convex Cloud — хмарне сховище колекцій документів із вбудованими індексами та реплікацією даних у реальному часі.
- Clerk Cloud — сервіс аутентифікації та управління сесіями.
- Liveblocks Cloud — сервіс спільного редагування через WebSocket.

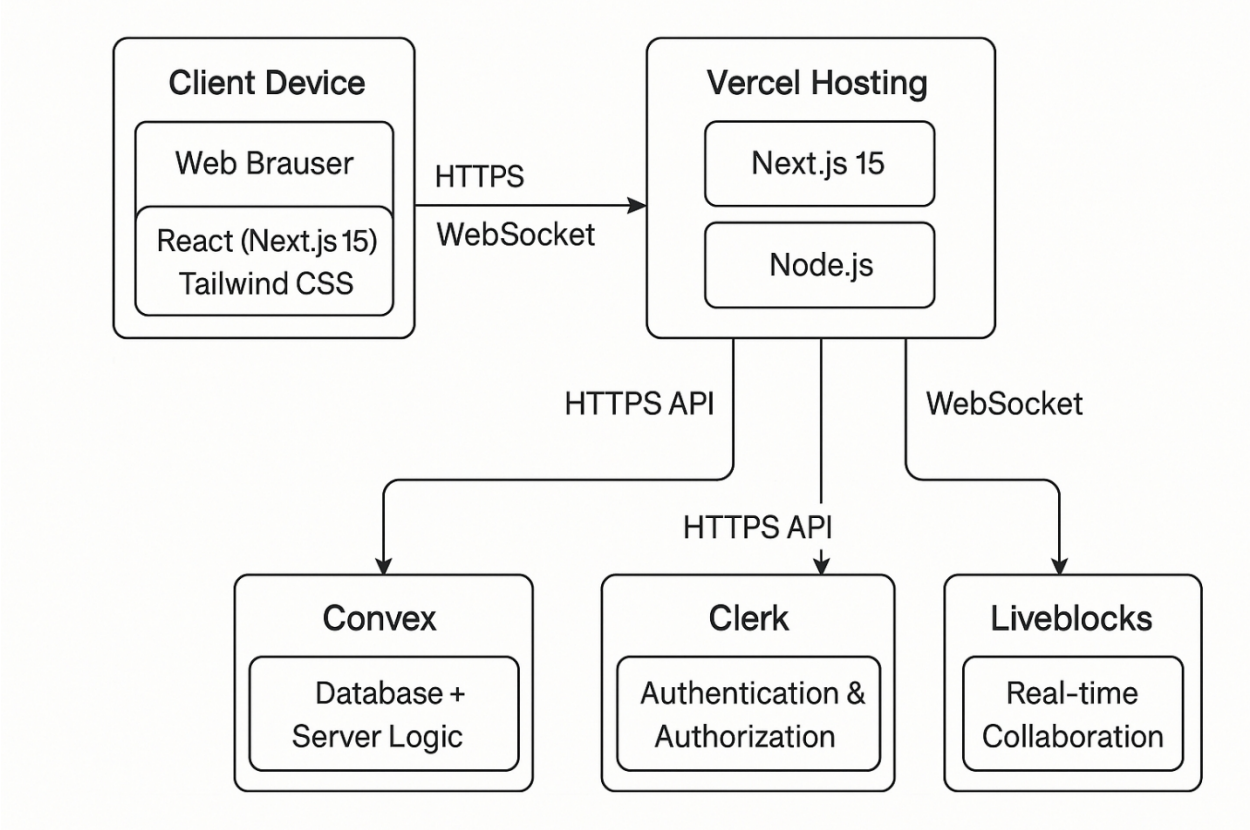

Рис. 4.3. Діаграма розгортання

Таблиця 4.1 – Апаратні та програмні вимоги

Компонент	**Мінімальні вимоги**	**Рекомендовані вимоги**
Користувацький пристрій	CPU:2-ядерний RAM:4ГБ Браузер: Chrome/Firefox останньої версії	CPU:4-ядерний RAM:8ГБ Браузер: Chrome/Firefox/Edge останніх версій
Сервер безсерверних функцій (Vercel)	256 МБ RAM на один виклик функції* ~50 МБ тимчасового сховища	Автоматичне масштабування без обмежень

Хмарна база Convex	Підтримка HTTPS, TLS 1.2+ Необмежена пам’ять у хмарі*	Сервісне SLA 99,9 % доступності
Clerk (Auth)	HTTPS, OAuth 2.0 підтримка	HTTPS, HTTP/2, JWT
Liveblocks	WebSocket-з’єднання	Автоматичне шардінг-з’єднань для масштабування
CI/CD (Vercel)	Git-проєкт на GitHub/GitLab	Автоматичні деплої на кожен мердж

* Примітка: обмеження Convex і Vercel регулюються тарифним планом; для безкоштовних акаунтів можуть застосовуватися ліміти.

Мережева інфраструктура:

- Канал з мінімальною пропускною здатністю 10 Мбіт/с для кожного розробника та 100 Мбіт/с для серверних точок (CDN, Convex).
- Низька затримка (< 100 мс) у процесі клієнт-серверної взаємодії для комфортного спільного редагування.

Операційна система та середовище:

- Node.js 18+
- NPM або Yarn останніх версій
- В ОС-середовищі: Linux (Ubuntu 20.04+), macOS 11+, Windows 10+.

Резервне копіювання та моніторинг:

- Використовуйте вбудовані механізми Convex для знімків (snapshots) даних.
- Налаштуйте моніторинг з боку Vercel і зовнішні сервіси (Datadog, Grafana) для відстеження затримок функцій і помилок.

Безпека:

- Відключіть HTTP в адміністративних консолях Vercel і Convex — всі з’єднання тільки через HTTPS.
- Забезпечте регулярне оновлення залежностей (npm audit, Snyk) і оновлення компонентів.

Отже, забезпечивши відповідне апаратне та програмне середовище, ми отримаємо стабільну, продуктивну та захищену платформу ведення нотаток для навчання, готову до розгортання і масштабування відповідно до потреб користувацької аудиторії.

4.3. Склад інсталяційного пакету

Оскільки інформаційна система NoteMinds є сучасним веб-додатком, який функціонує у хмарному середовищі, традиційна інсталяція на локальні пристрої кінцевих користувачів не потрібна. Усі основні компоненти системи розміщені на спеціалізованих платформах, які автоматично забезпечують її працездатність, масштабованість та доступність через інтернет.

Процес розгортання NoteMinds починається з підготовки хостингового середовища. У нашому випадку для розміщення клієнтської та серверної частин використано платформу Vercel, яка забезпечує швидке завантаження інтерфейсу завдяки CDN (Content Delivery Network), а також виконує безсерверні функції на основі запитів користувача.

У якості бази даних використовується Convex — хмарне сховище документів, яке не потребує окремого встановлення чи конфігурації. Замість цього, підключення до бази даних реалізується через REST або RPC-інтерфейси, а вся логіка роботи з даними визначається у вигляді функцій, які компілюються під час збірки та автоматично розгортаються на сервері Convex.

Для реалізації аутентифікації використовується сервіс Clerk, який також працює у хмарі та інтегрується в систему через SDK. Він не вимагає окремого серверного налаштування, оскільки всі компоненти реєстрації, входу,

відновлення пароля тощо вже доступні через API та інтерфейсні компоненти React.

Крім цього, у системі реалізовано спільну роботу користувачів над нотатками в реальному часі. За це відповідає хмарна платформа Liveblocks, яка забезпечує WebSocket-з'єднання для синхронізації дій декількох користувачів. Додаткове встановлення серверів або клієнтів не потрібне — достатньо підключити бібліотеку до клієнтської частини застосунку.

Компоненти інсталяційного пакету NoteMinds

Оскільки застосунок повністю орієнтований на хмарну архітектуру, інсталяційний пакет формується у вигляді структурованого репозиторію з вихідним кодом. Після компіляції система готова до розгортання на Vercel без необхідності в окремій установці.

Основні елементи інсталяційного пакета:

- package.json – файл, у якому визначені залежності проєкту, скрипти для запуску та конфігурація збірки.
- next.config.js – конфігурація для фреймворку Next.js, що описує шляхи, маршрутизацію та оптимізацію.
- convex/ – директорія з серверними функціями для взаємодії з базою даних.
- app/ або pages/ – сторінки інтерфейсу, розроблені з використанням React.
- styles/ – CSS або Tailwind класи для оформлення.
- public/ – статичні файли (іконки, логотипи, SEO-файли).
- .env.local – файл середовища з ключами доступу до API (Clerk, Convex, Liveblocks).
- Компоненти авторизації (<SignIn />, <UserButton />) — імпортуються з Clerk SDK.
- Модулі спільної роботи — реалізовані через Liveblocks Provider.

У разі використання CI/CD, наприклад на GitHub, система автоматично збирається та розгортається при кожному злитті змін у головну гілку (main).

Vercel самостійно створює інстанс, компілює проєкт і публікує його з урахуванням вказаних змінних середовища.

Розміщення та доступність

Після успішного деплою на Vercel система отримує автоматично згенеровану адресу (на кшталт noteminds.vercel.app), за якою вона доступна користувачам. У разі потреби підключається власне доменне ім’я — для цього потрібно в налаштуваннях домену створити A-запис або CNAME, а на платформі Vercel додати відповідне доменне ім’я до проєкту.

У підсумку, інсталяційний пакет NoteMinds не є класичним архівом або виконуваним файлом, а є повністю автоматизованою CI/CD-процедурою розгортання з вихідного коду у хмарну інфраструктуру. Це забезпечує гнучкість, високу доступність та легкість масштабування.

4.4. Висновки до розділу 4

Це розділ присвячено детальному аналізу етапів встановлення та налаштування системи NoteMinds, що слугує для ведення навчальних нотаток. Проведено тестування основних функціональних можливостей системи, що дозволило перевірити її стабільність, відповідність поставленим вимогам та працездатність усіх ключових модулів. Особливу увагу було приділено перевірці роботи реєстрації, авторизації, створення та редагування нотаток, функції спільного доступу, а також пошуку контенту. Усі протестовані компоненти працюють коректно та не викликають критичних помилок.

Було проаналізовано вимоги до апаратного та програмного забезпечення, на основі чого сформовано оптимальні конфігурації для розміщення системи у хмарному середовищі. Враховуючи використання сучасних хостингових платформ (зокрема Vercel, Convex, Clerk), система не потребує встановлення складного серверного середовища вручну — більшість процесів автоматизовано або делеговано зовнішнім сервісам. Це значно спрощує процес розгортання та подальшого обслуговування системи.

Також було детально описано структуру інсталяційного пакету, який формується з вихідного коду системи. Цей підхід дозволяє оперативно вносити зміни до логіки проєкту, підтримувати його в актуальному стані та швидко масштабувати при зростанні навантаження. Розгортання системи є швидким, безпечним і зручним завдяки інтеграції з CI/CD-процесами, що підтримуються на платформі Vercel.

Загалом, система NoteMinds демонструє високий рівень готовності до використання в навчальному процесі, є надійною, адаптованою до потреб сучасного користувача та технічно гнучкою для подальшого розвитку.

ВИСНОВКИ

У межах виконання дипломної роботи було реалізовано повноцінну інформаційну систему NoteMinds, призначену для створення, зберігання та колективного редагування навчальних нотаток. Головною метою проєкту було розробити сучасний, зручний та функціональний веб-додаток, що сприятиме підвищенню ефективності навчального процесу за рахунок цифровізації особистих та спільних записів. Поставлена мета досягнута повною мірою: створена система відповідає всім визначеним вимогам та демонструє стабільну і надійну роботу.

Процес дослідження включав вивчення наявних цифрових рішень, що підтримують навчання та взаємодію. Це дозволило окреслити переваги й недоліки аналогів, а також визначити напрямки, в яких варто розвивати власне програмне забезпечення. На основі аналізу було спроєктовано архітектуру системи, яка об’єднує зручний користувацький інтерфейс із потужною серверною логікою та гнучким зберіганням даних у хмарному середовищі.

Система NoteMinds побудована на сучасному стеку технологій: React з використанням Next.js 15 для фронтенду, Tailwind CSS для стилізації, Convex як база даних з одночасною взаємодією, та Clerk для керування автентифікацією. Також інтегровано сервіс Liveblocks, що забезпечує

можливість колективного редагування документів у реальному часі. Розгортання проєкту здійснено на платформі Vercel, що гарантує високу швидкість завантаження та автоматичне масштабування.

В ході розробки реалізовано весь базовий функціонал: створення нотаток, їх редагування, видалення, збереження змін у реальному часі, пошук по контенту, а також управління доступом до документів. Окрему увагу приділено безпеці та захисту персональних даних користувачів, зокрема — шифруванню, хешуванню паролів та обмеженню доступу до нотаток.

Здійснено комплексне тестування системи, яке показало її готовність до реального використання: перевірено працездатність основних сценаріїв, коректність обробки помилок та надійність обміну даними між клієнтом і сервером.

Слід зазначити, що навіть попри завершеність основного етапу розробки, проєкт має великий потенціал для подальшого розвитку. Зокрема, в майбутньому можна реалізувати інтеграцію з іншими освітніми сервісами, підтримку форматів імпорту/експорту нотаток, мобільну адаптацію або окремий мобільний застосунок, а також розширення можливостей редактора (наприклад, додавання зображень, таблиць або математичних формул).

Загалом, розроблене рішення повністю готове до впровадження у навчальний процес і може бути успішно використане як окремими студентами, так і цілими навчальними групами або викладачами. Воно поєднує сучасні технології, простоту у користуванні та масштабовану архітектуру, що робить його актуальним і перспективним інструментом у цифровій освіті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дудзяний І.М. Об'єктно-орієнтоване моделювання програмних систем: Навчальний посібник. Львів: Видавничий центр ЛНУ імені Івана Франка, 2007. 108 с.
2. Автоматизація та комп’ютерно-інтегровані технології у виробництві та освіті: стан, досягнення, перспективи розвитку: матеріали Всеукраїнської науково-практичної Internet-конференції. – Черкаси, 2021. - 330 с. – [Укр. мова.]
3. Гайна Г. А. Основи проектування баз даних: навч. посіб. Київ: КНУБА, 2005. 204 с.
4. Методичні вказівки до організації самостійної роботи студентів (СРС) з дисципліни «Сучасні технології автоматизованого проектування та верифікації програм». Мова моделювання UML / уклад. Я.Ю. Дорогий, О.О. Дорога-Іванюк, – Київ.: НТУУ “КПІ ім. І. Сікорського”, 2021. – 60 с.
5. Застосування uml для моделювання та проектування інформаційних систем Методичні вказівки до лабораторного практикуму з дисципліни „Об’єктноорієнтований аналіз та проектування‖ для студентів напряму підготовки 123 — "Комп’ютерна інженерія"./ Укл. А.М. Акименко, І.В. Богдан, А.С. Посадська — Чернігів: ЧНТУ, 2018. — 37 с. укр. Мовою.
6. Проектування інформаційних систем: навчальний посібник / В.С. Авраменко, А.С. Авраменко. – Черкаси: Черкаський національний університет ім. Б. Хмельницького, 2017. – 434 с.: іл.
7. С. С. ВЕЛИКОДНИЙ, Ж. В. БУРЛАЧЕНКО, С. С. ЗАЙЦЕВА-ВЕЛИКОДНА Сучасний стан наукових досліджень та технологій в промисловості. 2019. № 2 (8) УДК 004.4'22 + 658.5 DOI: https://doi.org/10.30837/2522-9818.2019.8.025
8. Гаркуша І.М. Конспект лекцій з дисципліни “Проектування інформаційних систем” для студентів галузі знань 12 “Інформаційні

технології” спеціальності 126 “Інформаційні системи та технології”. – Д.: НТУ «ДП», 2020. – 75 с.

9. Акименко, А. М., & Нестеренко, С. О. (2021). UML-МОДЕЛЬ СИСТЕМИ УПРАВЛІННЯ БЕЗПІЛОТНИМ АВІАЦІЙНИМ КОМПЛЕКСОМ. Технічні науки та технології, (1(7), 116–124. вилучено із http://tst.stu.cn.ua/article/view/104939
10. Рудий Т. В., Паранчук Я. С., Сеник В. В. Алгоритмізація та програмування. Частина 1. Структурне програмування : навчальний посібник. Львів : Львівський державний університет внутрішніх справ, 2023. 240 с.

ДОДАТОК А. Код северної частини

```
//noteminds\convex\documents.ts
import { mutation, query } from "./_generated/server";
import { ConvexError, v } from "convex/values";
import { paginationOptsValidator } from "convex/server";

export const getByIds = query({
  args: { ids: v.array(v.id("documents")) },
  handler: async (ctx, { ids }) => {
    const documents = [];

    for (const id of ids) {
      const document = await ctx.db.get(id);

      if (document) {
        documents.push({ id: document._id, name: document.title });
      } else {
        documents.push({ id, name:"[Removed]" })
      }
    }

    return documents;
  },
});

export const create = mutation({
  args:{ title: v.optional(v.string()), initialContent: v.optional(v.string()) },
  handler: async (ctx, args) => {
```

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

    if (!user) {
      throw new ConvexError("Unathorized");
    }

    const organizationId = (user.organization_id ?? undefined) as
    | string
    | undefined;

    return await ctx.db.insert("documents",{
      title: args.title ?? "Untitle coument",
      ownerId: user.subject,
      organizationId,
      initialContent: args.initialContent,
    });
  },
});

export const get = query({
    args: {paginationOpts: paginationOptsValidator, search: v.optional(v.string())},
    handler: async (ctx, { search, paginationOpts }) => {
      const user = await ctx.auth.getUserIdentity();

      if (!user) {
        throw new ConvexError("Unauthorized");
      }

      const organizationId = (user.organization_id ?? undefined) as
```

```
  | string
  | undefined;

  // пошук нотаток організації
if (search && organizationId) {
  return await ctx.db
  .query("documents")
  .withSearchIndex("search_title", (q) =>
    q.search("title", search).eq("organizationId", organizationId)
  )
  .paginate(paginationOpts)
}

console.log({ user });

// персональний пошук нотаток
if (search) {
  return await ctx.db
    .query("documents")
    .withSearchIndex("search_title", (q) =>
      q.search("title", search).eq("ownerId", user.subject)
  )
  .paginate(paginationOpts)
}

// всі організаційні нотатки
if (organizationId) {
  return await ctx.db
  .query("documents")
```

```
                .withIndex("by_organization_id", (q) => q.eq("organizationId",
organizationId))
      .paginate(paginationOpts);
    }

    //всі персональні нотатки
    return await ctx.db
      .query("documents")
      .withIndex("by_owner_id", (q) => q.eq("ownerId", user.subject))
      .paginate(paginationOpts);
    // do something with tasks
  },
 });

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

  if (!user){
   throw new ConvexError("Unauthorized");
  }

  const organizationId = (user.organization_id ?? undefined) as
  | string
  | undefined;

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

```
    if (!document) {
      throw new ConvexError("Note not found");
    }

    const isOwner = document.ownerId === user.subject;
    const isOrganizationMember = document.organizationId === organizationId;

    if (!isOwner && !isOrganizationMember) {
      throw new ConvexError("Unauthorized");
    }

    return await ctx.db.delete(args.id);
  },
});

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

    if (!user){
      throw new ConvexError("Unauthorized");
    }

    const organizationId = (user.organization_id ?? undefined) as
    | string
    | undefined;

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

```
    if (!document) {
      throw new ConvexError("Note not found");
    }

    const isOwner = document.ownerId === user.subject;
    const isOrganizationMember = document.organizationId === organizationId;

    if (!isOwner && !isOrganizationMember) {
      throw new ConvexError("Unauthorized");
    }

    return await ctx.db.patch(args.id, {title: args.title});
  },
});

export const getById = query({
  args: { id: v.id("documents")},
  handler: async (ctx, { id }) => {
    const document = await ctx.db.get(id);

    if (!document) {
      throw new ConvexError("Note not found");
    }

    return document;
  },
});
```

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

export default defineSchema({
   documents: defineTable({
      title: v.string(),
      initialContent: v.optional(v.string()),
      ownerId: v.string(),
      roomID: v.optional(v.string()),
      organizationId: v.optional(v.string()),
   })
      .index("by_owner_id", ["ownerId"])
      .index("by_organization_id", ["organizationId"])
      .searchIndex("search_title", {
         searchField: "title",
         filterFields: ["ownerId", "organizationId"],
      }),
 });
// src/app/api/liveblocks-auth/route.tsf
import { Liveblocks } from "@liveblocks/node";
import { ConvexHttpClient } from "convex/browser";
import { auth, currentUser } from "@clerk/nextjs/server";
import { api } from "../../../../convex/_generated/api";

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

```
const liveblocks = new Liveblocks({
  secret: process.env.LIVEBLOCKS_SECRET_KEY!,
});

export async function POST(req: Request) {
  const { sessionClaims } = await auth();
  if (!sessionClaims) {
    return new Response("Unauthorized", { status: 401 });
  }

  const user = await currentUser();
  if (!user) {
    return new Response("Unauthorized", { status: 401 });
  }

  const { room } = await req.json();
  const document = await convex.query(api.documents.getById, { id: room });
  if (!document) {
    return new Response("Unauthorized", { status: 401 });
  }

  const userOrgId = sessionClaims.org_id;
  const isOwner = document.ownerId === user.id;
  const isOrgDoc = Boolean(document.organizationId);
  const isUserNoOrg = !userOrgId;
  const isPersonalDoc = !document.organizationId;

  // Allow if owner, or organization docs, or two non-org users on personal docs
  if (!isOwner && !isOrgDoc && !(isPersonalDoc && isUserNoOrg)) {
```

```
    return new Response("Unauthorized", { status: 401 });
  }

   const name = user.fullName ?? user.primaryEmailAddress?.emailAddress ??
"Anonymous";
      const nameToNumber = name.split("").reduce((acc, char) =>
acc+char.charCodeAt(0), 0);
  const hue = Math.abs(nameToNumber) % 360;
  const color = `hsl(${hue}, 80%, 60%)`;

  const session = liveblocks.prepareSession(user.id, {
    userInfo: {
      name,
      avatar: user.imageUrl,
      color,
    },
  });

  session.allow(room, session.FULL_ACCESS);
  const { body, status } = await session.authorize();

  return new Response(body, { status });
}
// src/app/[...]/actions.tsf
"use server";

import { ConvexHttpClient } from "convex/browser";
import { auth, clerkClient } from "@clerk/nextjs/server";
import { Id } from "../../../../../../convex/_generated/dataModel";
```

```
import { api } from "../../../../../../convex/_generated/api";

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

// Отримуємо документи за списком id
export async function getDocuments(ids: Id<"documents">[]) {
  return await convex.query(api.documents.getByIds, { ids });
}

// Отримуємо користувачів із Clerk.
// Якщо є org_id у сесії — фільтруємо по ньому, інакше повертаємо всіх.
export async function getUsers() {
  const { sessionClaims } = await auth();
  // Викликаємо clerkClient() як async ф-цію
  const clerk = await clerkClient();

  let response;
  if (sessionClaims?.org_id) {
    response = await clerk.users.getUserList({
      organizationId: [sessionClaims.org_id],
    });
  } else {
    response = await clerk.users.getUserList({});
  }

  return response.data.map((user) => ({
    id: user.id,
    name:
```

```
      user.fullName ??
      user.primaryEmailAddress?.emailAddress ??
      "Anonymous",
    avatar: user.imageUrl,
    color: "",
  }));
}
```

ДОДАТОК Б. Код клієнтської частини

```
const clerkConfig = {
    providers: [
        {
            domain: "…",
            applicationID: "convex",
        },
    ],
};

export default clerkConfig;
"use client";


import { usePaginatedQuery } from "convex/react";

import { Navbar } from "./frontend/layout/navbar";

import { TemplatesGallery } from "./templates-qallery";


import { api } from "../../../convex/_generated/api";
```

```
import { DocumentsTable } from "./frontend/components/document/documents-table";

import { useSearchParam } from "@/hooks/use-search-param";


const Home = () => {
  const [search] = useSearchParam();
  const {
    results,
    status,
    loadMore
  } = usePaginatedQuery(api.documents.get, {search}, { initialNumItems: 5 });


  return (
    <div className="min-h-screen flex flex-col">
      <div className="fixed top-0 left-0 right-0 z-10 h-16 bg-white p-4">
        <Navbar />
      </div>
      <div className="mt-16">
        <TemplatesGallery />
        <DocumentsTable
          documents={results}
          loadMore={loadMore}
          status={status}
```

```
        />
      </div>
    </div>
  )
}

export default Home;
"use client";

import { Input } from "@/components/ui/input";
import { Button } from "@/components/ui/button";

import { useSearchParam } from "@/hooks/use-search-param";
import { SearchIcon, XIcon } from "lucide-react";
import { useState, useRef } from "react";

export const SearchInput = () => {
    const [search, setSearch] = useSearchParam();
    const [value, setValue] = useState(search);

    const inputRef = useRef<HTMLInputElement>(null);
```

```
const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    setValue(e.target.value);
};

const handleClear = () => {
    setValue("");
    setSearch("");
    inputRef.current?.blur();
};

const handleSubmit = (e: React.FormEvent<HTMLFormElement>) => {
    e.preventDefault();
    setSearch(value);
    inputRef.current?.blur();
};

return (
    <div className="flex-1 flex items-center justify-center">
        <form
            onSubmit={handleSubmit}
            className="relative max-w-[720px] w-full"
        >
```

```
<Input
  value={value}
  onChange={handleChange}
  ref={inputRef}
  placeholder="Search"
   className="md:text-base placeholder:text-neutral-800 px-14 w-full border-none focus-visible:shadow-[0_1px_1px_0_rgba(65,69,73,.3),0_1px_3px_1px_rgba(65,69,73,.15)] bg-[#F0F4F8] rounded-full h-[48px] focus-visible:right-0 focus:bg-white"
/>
<Button
  type="submit"
  variant="ghost"
  size="icon"
    className="absolute left-3 top-1/2 -translate-y-1/2 [&_svg]:size-5 rounded-full"
>
  <SearchIcon />
</Button>
{value && (
  <Button
    onClick={handleClear}
    type="button"
    variant="ghost"
```

```
                    size="icon"
                  className="absolute right-3 top-1/2 -translate-y-1/2 [&_svg]:size-5
rounded-full"
                >
                  <XIcon />
              </Button>
              )}
            </form>
        </div>
    );
};
"use client";

import {
    Carousel,
    CarouselContent,
    CarouselItem,
    CarouselNext,
    CarouselPrevious,
} from "@/components/ui/carousel";

import { cn } from "@/lib/utils";
import { templates } from "@/constants/templates";
```

```
import { useRouter } from "next/navigation";
import { useMutation } from "convex/react";
import { api } from "../../../convex/_generated/api";
import { useState } from "react";
import { toast } from "sonner";

export const TemplatesGallery = () => {
    const router = useRouter();
    const create = useMutation(api.documents.create);
    const [isCreating, setIsCreating] = useState(false);

    const onTemplateClick = (title: string, initialContent: string) => {
        setIsCreating (true);
        create ({ title, initialContent })
            .catch(() => toast.error("Something went wrong"))
            .then((documentId) => {
                toast.success("Note created")
                router.push(`/documents/${documentId}`);
            })
            .finally(() =>{
                setIsCreating(false);
            });
```

```
};


return(
    <div className="bg-[#F1F3F4]">
        <div className="max-w-screen-xl mx-auto px-16 py-6 flex flex-col gap-y-4">
            <h3 className="font-medium">Start a new note</h3>
            <Carousel>
                <CarouselContent className="-ml-4">
                    {templates.map((template) => (
                        <CarouselItem
                            key={template.id}
                            className="basis-1/2 sm:basis-1/3 md:basis-1/4 lg:basis-1/5 xl:basis-1/6 2xl:basis-[14.285714%] pl-4"
                        >
                            <div
                                className={cn(
                                    "aspect-[3/4] flex flex-col gap-y-2.5",
                                    isCreating && "pointer-events-none opacity-50"
                                )}
                            >
                                <button
                                    disabled={isCreating}
```

```
                                    onClick={() => onTemplateClick(template.label,
template.initialContent)}
                                    style={{
                                        backgroundImage: `url(${template.imageUrl})`,
                                        backgroundSize: "cover",
                                        backgroundPosition: "center",
                                        backgroundRepeat: "no-repeat",
                                    }}
                                    className="size-full hover:border-blue-500 rounded-sm
border hover:bg-blue-50 transition flex flex-col items-center justify-center gap-y-
4 bg-white"
                                />
                                <p className="text-sm font-medium truncate">
                                    {template.label}
                                </p>
                            </div>
                        </CarouselItem>
                    ))}
                </CarouselContent>
                <CarouselPrevious />
                <CarouselNext />
            </Carousel>
        </div>
    </div>
```

```
    );
};
// avatars.tsx
"use client";

import { ClientSideSuspense } from "@liveblocks/react";
import { useOthers, useSelf } from "@liveblocks/react/suspense";
import { Separator } from "@/components/ui/separator";

const AVATAR_SIZE = 36;

export const Avatars = () => {
    return (
        <ClientSideSuspense fallback={null}>
            <AvatarStack />
        </ClientSideSuspense>
    );
};

const AvatarStack = () => {
    const users = useOthers();
    const currentUser = useSelf();
```

```
    if (users.length === 0) return null;


    return (
        <>
            <div className="flex items-center">
                {currentUser && (
                    <div className="relative ml-2">
                        <Avatar src={currentUser.info.avatar} name="You"/>
                    </div>
                )}
                <div className="flex">
                    {users.map(({connectionId, info}) => {
                        return (
                            <Avatkey={connectionId} src={info.avatar} name={info.name}
/>
                        )
                    })}
                </div>
            </div>
            <Separator orientation="vertical" className="h-6"/>
        </>
    )
```

```
}


interface AvatarProps {
   src: string;
   name: string;
};


const Avatar = ({src, name}: AvatarProps) => {
   return (
      <div
         style = {{ width: AVATAR_SIZE, height: AVATAR_SIZE }}
            className="group -ml-2 flex shrink-0 plase-content-center relative border-4 border-white rounded-full bg-gray-400"
      >
         <div className="opacity-0 group-hover:opecity-100 absolute top-ful py-1 px-2 text-white text-xs rounded-lg mt-2.5 z-10 bg-black whitespace-nowrap translate-opacity">
            {name}
         </div>
         {/* eslint-disable-next-line @next/next/no-img-element */}
         <img
            alt={name}
            src={src}
            className="size-full rounded-full"
```

```
          />
        </div>
    );
};
import { ClientSideSuspense, useThreads } from "@liveblocks/react/suspense";
import {
  AnchoredThreads,
  FloatingComposer,
  FloatingThreads,
} from "@liveblocks/react-tiptap";
import { Editor } from "@tiptap/react";

export const Threads = ({ editor }: { editor: Editor | null }) => {
  return (
    <ClientSideSuspense fallback={null}>
      <ThreadsList editor={editor}/>
    </ClientSideSuspense>
  );
};

function ThreadsList({ editor }: { editor: Editor | null }) {
  const { threads } = useThreads({ query: { resolved: false } });
```

```
  return (
    <>
      <div className="anchored-threads">
        <AnchoredThreads editor={editor} threads={threads} />
      </div>
      <FloatingThreads
        editor={editor}
        threads={threads}
        className="floating-threads"
      />
      <FloatingComposer editor={editor} className="floating-composer" />
    </>
  );
}
// document.tsx
"use client"

import { Room } from "./room";
import { Editor } from "./frontend/components/editor/editor";
import { Navbar } from "./navbar";
import { Toolbar } from "./toolbar";
```

```
import { Preloaded, usePreloadedQuery } from "convex/react";
import { api } from "../../../../convex/_generated/api";

interface DocumentProps{
   preloadedDocument: Preloaded<typeof api.documents.getById>
};

export const Document = ({ preloadedDocument }: DocumentProps) => {
   const document = usePreloadedQuery(preloadedDocument);

 return (
   <Room>
     <div className="min-h-screen bg-[#FaFBFD]">
        <div  className="flex flex-col px-4 pt-2 gap-y-2 fixed top-0 left-0 right-0
z-10 bg-[#FAFBFD] print:hidden">
          <Navbar data={document}/>
          <Toolbar />
        </div>
        <div className="pt-[114px] print:pt-0">
           <Editor initialContent={document.initialContent}/>
        </div>
     </div>
   </Room>
```

```
  );
};

// src/app/[...]/room.tsxd
"use client";

import { ReactNode, useEffect, useState } from "react";
import {
  LiveblocksProvider,
  RoomProvider,
  ClientSideSuspense,
} from "@liveblocks/react/suspense";
import { useParams } from "next/navigation";
import { FullscreenLoader } from "@/components/fullscreen-loader";
import { getUsers, getDocuments } from "./backend/actions/actions";
import { toast } from "sonner";
import { Id } from "../../../../convex/_generated/dataModel";

import {
  RIGHT_MARGIN_DEFAULT,
  LEFT_MARGIN_DEFAULT,
} from "@/constants/margins";
```

```
type User = { id: string; name: string; avatar: string; color: string; };

export function Room({ children }: { children: ReactNode }) {
  const { documentId } = useParams() as { documentId: string };

  // users = null — поки не завантажили; потім або [] або список
  const [users, setUsers] = useState<User[] | null>(null);

  useEffect(() => {
    getUsers()
      .then(setUsers)
      .catch((err) => {
        console.error(err);
        toast.error("Не вдалося отримати список користувачів");
        // У випадку помилки припиняємо лоадер, щоб не блокувати весь інтерфейс
        setUsers([]);
      });
  }, []);

  // Чекаємо на завантаження
  if (users === null) {
```

```
  return <FullscreenLoader label="Room loading..." />;
}

return (
  <LiveblocksProvider
    throttle={16}
    authEndpoint={async () => {
      const res = await fetch("/api/liveblocks-auth", {
        method: "POST",
        body: JSON.stringify({ room: documentId }),
      });
      if (!res.ok) {
        throw new Error("Liveblocks auth failed");
      }
      return res.json();
    }}
    resolveUsers={({ userIds }) =>
      // Завжди повертаємо або User, або undefined
      userIds.map((id) => users.find((u) => u.id === id))
    }
    resolveMentionSuggestions={({ text }) => {
      const filtered = text
```

```
      ? users.filter((u) =>
          u.name.toLowerCase().includes(text.toLowerCase())
        )
      : users;
    return filtered.map((u) => u.id);
  }}
  resolveRoomsInfo={async ({ roomIds }) => {
    const docs = await getDocuments(roomIds as Id<"documents">[]);
    return docs.map((d) => ({ id: d.id, name: d.name }));
  }}
>
  <RoomProvider
    id={documentId}
    initialStorage={{
      leftMargin: LEFT_MARGIN_DEFAULT,
      rightMargin: RIGHT_MARGIN_DEFAULT,
    }}
  >
    <ClientSideSuspense
      fallback={<FullscreenLoader label="Room loading..." />}
    >
      {children}
```

```
        </ClientSideSuspense>
      </RoomProvider>
    </LiveblocksProvider>
  );
}
import type { Metadata } from "next";
import { Inter } from "next/font/google";
import "@liveblocks/react-ui/styles.css";
import "@liveblocks/react-tiptap/styles.css";
import "./globals.css";

import { NuqsAdapter } from "nuqs/adapters/next/app";
import { ConvexClientProvider } from "@/components/convex-client-provider";
import { Toaster } from "@/components/ui/sonner";

const inter = Inter ({
  subsets: ["latin"],
})

export const metadata: Metadata = {
  title: "NoteMinds",
  description: "Generated by create next app",
```

```
};


export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode;
}>) {
  return (
    <html lang="en">
      <body
        className={inter.className}
      >
        <NuqsAdapter>
          <ConvexClientProvider>
            <Toaster />
            {children}
          </ConvexClientProvider>
        </NuqsAdapter>
      </body>
    </html>
  );
}
```