НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»
Завідувач кафедри
_____Наталія АУШЕВА
“___”_____2026 р.

Дипломна робота
на здобуття ступеня бакалавр

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “ Веб-система генерації конспектів на основі відеоконтенту за допомогою штучного інтелекту ”

Виконав: студент 4 курсу, групи ТР-21
Наумкін Дмитро Сергійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник асистент Костянтин ЗДОР _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент проф. каф. ТАЕ НН ІАТЕ,
д.т.н., проф Ольга ЧЕРНОУСЕНКО _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ (підпис)

Н.контроль ст.викладач Ольга БЕСПАЛА _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ - 2026

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____Наталія АУШЕВА
«___»_____2026 р.

ЗАВДАННЯ

на дипломну роботу студенту

Наумкіну Дмитру Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи “ Веб-система генерації конспектів на основі відеоконтенту за допомогою штучного інтелекту ”

Науковий керівник Здор Костянтин Андрійович, асистент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “ 28 ” травня 2026 року № 1992

2. Термін подання студентом роботи 04.06.2026

3. Вихідні дані до роботи FastAPI, PostgreSQL, OpenAI API, Deepgram, Cloudflare R2, Docker Compose, результати тестування програмних модулів

4. Перелік питань, які потрібно розробити _____

1) провести аналіз предметної області автоматизованого створення навчальних матеріалів на основі відеоконтенту;
2) проаналізувати наявні EdTech-рішення та обґрунтувати вибір технологій для розробки веб-системи;
3) розробити методи й алгоритми імпорту, транскрибації та перетворення навчального контенту в структурований конспект;

4) спроєктувати архітектуру веб-системи, структуру бази даних і REST API для роботи з навчальними матеріалами;
5) реалізувати backend- і frontend-частини веб-системи, а також провести контейнеризацію й тестування розробленої програмної системи.
5. Орієнтований перелік ілюстративного матеріалу схема загальної архітектури веб-системи; схема pipeline обробки навчального контенту; ER-діаграма бази даних; діаграма прецедентів; скриншоти основних користувацьких сценаріїв.
6. Дата видачі завдання 16.06.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	16. 06. 2025	Виконано
2.	Аналіз методів та засобів розв’язання задачі	20.04-24.04.26	Виконано
3.	Розробка архітектури та загальної структури системи	24.04-27.04.26	Виконано
4.	Розробка окремих підсистем	28.04-01.05.26	Виконано
5.	Програмна реалізація системи	02.05-07.05.26	Виконано
6.	Оформлення пояснювальної записки	08.05-29.05.26	Виконано
7.	Захист програмного забезпечення	14.05.26	Виконано
8.	Передзахист	04.06.26	Виконано
9.	Захист	15.06-19.06.2026	Виконано

Студент _____ Наумкін Д. С.
(підпис) (прізвище та ініціали)

Керівник _____ Здор К. А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська робота виконана на 55 сторінках, містить 10 рисунків, 19 таблиць, 1 додаток та 21 джерело у переліку джерел посилання.

Бакалаврська робота присвячена розробці веб-системи Lectura для автоматизованої генерації навчальних матеріалів на основі відеоконтенту з використанням технологій штучного інтелекту. Система дає змогу користувачеві імпортувати відео-, аудіо- та текстові джерела, отримувати структурований PDF-конспект, створювати картки для повторення, екзаменаційні питання та працювати з контекстним AI-чатом.

У роботі виконано аналіз предметної області та існуючих EdTech-рішень, опис методів транскрибації і генерації навчального звіту, проєктування клієнт-серверної архітектури, структури бази даних і REST API, а також тестування основних функціональних сценаріїв. Backend-частина реалізується мовою Python із використанням FastAPI, PostgreSQL, SQLAlchemy та Alembic. Для генерації навчальних матеріалів застосовуються моделі OpenAI API, для транскрибації - Deepgram, для зберігання PDF-файлів - Cloudflare R2, а локальний запуск забезпечується Docker Compose.

Практичне значення роботи полягає у скороченні часу підготовки до навчання та створенні єдиного середовища для роботи з лекційними матеріалами, конспектами, картками, тестами і чатом. Система може використовуватися студентами, викладачами та користувачами, які займаються самостійним навчанням.

Ключові слова: інженерія програмного забезпечення, програмні засоби, інформаційні технології, аналіз даних,обробка інформації, хмарне середовище, комп’ютерна система, нейронна мережа.

ABSTRACT

The bachelor thesis is presented on 55 pages and contains 10 figures, 19 tables, 1 appendix and 21 references.

The bachelor thesis is devoted to the development of the Lectura web system for automated generation of learning materials based on video content using artificial intelligence technologies. The system allows a user to import video, audio and text-based learning sources, generate a structured PDF summary, create flashcards, exam questions and interact with a context-aware AI assistant.

The thesis includes an analysis of the subject area and existing EdTech solutions, a description of transcription and educational report generation methods, design of the client-server architecture, database model and REST API, as well as testing of the main functional scenarios. The backend is implemented in Python using FastAPI, PostgreSQL, SQLAlchemy and Alembic. OpenAI API models are used for generating learning materials, Deepgram is used for transcription, Cloudflare R2 is used for PDF file storage, and Docker Compose is used for local execution.

The practical value of the work is to reduce the time required for study preparation and to provide a unified environment for lecture materials, summaries, flashcards, tests and document-based chat. The system can be used by students, teachers and independent learners.

Keywords: software engineering, software tools, information technologies, artificial intelligence, data analysis, information processing, cloud environment, computer system, neural network.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ

AI – Artificial Intelligence, штучний інтелект

API – Application Programming Interface, програмний інтерфейс взаємодії

JWT – JSON Web Token, токен авторизації

PDF – Portable Document Format

REST – Representational State Transfer

ORM – Object-Relational Mapping

R2 – Cloudflare R2 object storage

Saas – Software as a Service

RAG – Retrieval-Augmented Generation

JSON – JavaScript Object Notation

UI – User Interface, користувацький інтерфейс

UX – User Experience, користувацький досвід

ВСТУП

Сучасний освітній процес дедалі більше спирається на цифрові матеріали: відеолекції, записи онлайн-занять, PDF-документи, презентації, електронні конспекти, текстові нотатки та навчальні платформи. Такий формат надає студентам і викладачам значно більше можливостей для гнучкого навчання, однак одночасно створює проблему надлишку інформації. Користувач отримує доступ до великої кількості джерел, але значна частина роботи з їх опрацювання залишається ручною: перегляд відео, виписування ключових тез, структурування матеріалу, підготовка питань для самоперевірки та створення карток для повторення.

Особливо складною є робота з лекційним відеоконтентом. Відеозапис лекції є корисним джерелом, оскільки містить пояснення викладача, приклади, інтонаційні акценти та послідовність викладу теми. Водночас відео є лінійним форматом: для пошуку конкретного визначення або повторення окремого фрагмента користувачеві часто потрібно переглядати значну частину запису. Якщо лекція триває понад годину, ручне створення конспекту може зайняти більше часу, ніж сам перегляд матеріалу. Через це виникає потреба у веб-системі, яка автоматизує перетворення відеоконтенту на структуровані навчальні матеріали.

Актуальність роботи зумовлена необхідністю підвищення ефективності самостійної підготовки. Навчання не завершується отриманням джерела: користувач повинен зрозуміти його зміст, виділити головні поняття, повторити матеріал і перевірити себе. Універсальні AI-асистенти та окремі EdTech-сервіси частково розв’язують цю задачу, але не завжди надають єдиний сценарій роботи: імпорт джерела, транскрибація, формування PDF-конспекту, створення карток, генерація екзаменаційних питань і чат із контекстом документа. Саме такий повний цикл покладено в основу веб-системи Lectura.

Тема бакалаврської роботи: «Веб-система генерації конспектів на основі відеоконтенту за допомогою штучного інтелекту». Розроблювана система орієнтована на студентів, школярів старших класів, викладачів і користувачів, які займаються самостійним навчанням. Вона має забезпечувати роботу з різними

типами навчальних джерел, зокрема YouTube-посиланнями, аудіо- та відеофайлами, PDF-, Markdown- і TXT-документами.

Метою роботи є розробка веб-системи Lectura для автоматизованого перетворення лекційного відеоконтенту в PDF-конспекти, картки для повторення, екзаменаційні питання та контекстний чат із використанням технологій штучного інтелекту.

Об’єктом дослідження є процес організації самостійного навчання на основі лекційних матеріалів.

Предметом дослідження є методи та засоби розробки веб-системи для автоматизованого створення навчальних матеріалів на основі відео-, аудіо- та текстового контенту.

Для досягнення поставленої мети необхідно виконати такі завдання: проаналізувати предметну область автоматизованого створення навчальних матеріалів; дослідити існуючі EdTech-рішення; сформувати функціональні та нефункціональні вимоги до веб-системи; розробити методи й алгоритми обробки навчального контенту; спроєктувати архітектуру, базу даних і REST API; реалізувати основні модулі системи Lectura; провести тестування функціональних сценаріїв; визначити обмеження поточної версії та напрями подальшого розвитку.

Практичне значення роботи полягає у створенні програмного продукту, який скорочує час підготовки до навчання, допомагає структурувати лекційний матеріал, підтримує активне повторення та самоперевірку. Веб-система може бути використана для індивідуального навчання, підготовки до контрольних заходів, опрацювання відеолекцій і роботи з PDF-документами.

Оформлення пояснювальної записки виконано відповідно до встановлених вимог до структури та оформлення бакалаврських робіт. Бакалаврська робота складається зі вступу, чотирьох розділів, висновків, переліку джерел посилання і додатку. Загальний обсяг роботи становить 55 сторінки, робота містить 10 рисунків, 19 таблиць, 1 додаток та 21 джерело у переліку джерел посилання. Фрагменти ключового програмного коду розробленої веб-системи наведено в додатку А.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

У першому розділі розглянуто предметну область автоматизованого створення навчальних матеріалів на основі лекційного контенту, визначено проблеми самостійної роботи з відеолекціями та PDF-документами, виконано огляд існуючих EdTech-рішень і сформульовано вимоги до веб-системи Lectura. Результати цього розділу є основою для подальшого опису методів, алгоритмів і програмної реалізації системи.

1.1 Актуальність автоматизованого створення навчальних матеріалів

Цифровізація освіти призвела до того, що навчальний матеріал дедалі частіше подається у вигляді відеозаписів, онлайн-курсів, електронних презентацій, PDF-документів і текстових нотаток. Такий підхід є зручним для поширення знань, однак не завжди зручний для повторення та самостійної підготовки. Користувач може мати доступ до всіх необхідних джерел, але не мати готового структурованого конспекту, переліку ключових понять, карток або питань для перевірки знань.

Традиційний процес підготовки до навчального контролю передбачає кілька послідовних дій: перегляд лекції, виписування основних тез, формування короткого конспекту, повторення термінів, створення питань і перевірка відповідей. Кожен із цих етапів потребує часу. Якщо студент працює з кількома лекціями або великим набором PDF-документів, навантаження зростає, а частина підготовки перетворюється на механічну роботу.

Автоматизація цього процесу є актуальною з кількох причин. По-перше, вона зменшує час первинної обробки навчального матеріалу. По-друге, дає можливість швидко отримати структурований конспект, який можна переглядати незалежно

від початкового формату джерела. По-третє, дозволяє перейти від пасивного читання до активного навчання через картки, екзаменаційні питання та контекстний чат. По-четверте, така система може бути корисною як студентам, так і викладачам, які готують допоміжні матеріали.

Використання технологій штучного інтелекту дає можливість не лише скорочувати текст, а й формувати структуровані навчальні матеріали. У межах системи Lectura AI-моделі використовуються для створення навчального звіту, карток, екзаменаційних питань, перевірки відповідей і підтримки контекстного діалогу з документом. Завдяки цьому система розглядається не як простий генератор резюме, а як комплексне середовище для роботи з навчальним контентом.

1.2 Аналіз процесу самостійного навчання за лекційним контентом

Самостійне навчання за лекційним контентом можна розглядати як процес перетворення первинного джерела знань у зручну для повторення та перевірки форму. У найпростішому випадку користувач переглядає відео або читає документ, після чого вручну формує нотатки. Однак такий підхід має суттєві недоліки: він потребує часу, залежить від уважності користувача та не гарантує, що всі важливі поняття буде виділено правильно.

Лекційний відеоконтент має перевагу повноти пояснення, але є менш придатним для швидкого повторення. На відміну від текстового документа, відео складно переглядати вибірково: користувачеві потрібно пам’ятати часові мітки або повторно шукати потрібний фрагмент. Навіть за наявності автоматичних субтитрів транскрипт зазвичай не є готовим конспектом, оскільки він містить повтори, неструктуровані речення, службові фрази та відступи від теми.

Для ефективного навчання важливо не лише отримати стислий виклад матеріалу, а й організувати повторення. Картки допомагають перевіряти знання

окремих понять, екзаменаційні питання дозволяють оцінити розуміння теми, а чат із документом дає змогу уточнювати незрозумілі фрагменти. Отже, веб-система має підтримувати не один окремий інструмент, а повний навчальний сценарій.

Основні етапи самостійної роботи з лекційним матеріалом і можливості їх автоматизації наведено в таблиці 1.1.

Таблиця 1.1 — Етапи самостійної роботи з лекційним матеріалом

Етап	Дії користувача	Проблеми ручного виконання	Автоматизація в Lectura
Отримання джерела	Відео, аудіо, YouTube-посилання або документ	Матеріали зберігаються в різних форматах	Імпорт різних типів джерел
Перетворення у текст	Перегляд лекції або читання документа	Відео складно швидко опрацювати	Транскрибація та витяг тексту
Структурування	Виділення тем, тез і понять	Потребує часу та уваги	AI-звіт і PDF-конспект
Повторення і перевірка	Картки, питання, самоперевірка	Потрібно створювати вручну	Картки, екзамен, AI-чат

Як видно з таблиці 1.1, більшість етапів самостійної роботи має потенціал для автоматизації. Саме тому система Lectura орієнтована не лише на створення PDF-конспекту, а й на подальшу роботу з матеріалом: повторення, перевірку знань і уточнення змісту через чат.

1.3 Огляд існуючих EdTech-рішень

На ринку існує низка EdTech-сервісів, які частково розв’язують задачу автоматизованої підготовки навчалЩьних матеріалів. Для аналізу було розглянуто Google NotebookLM, Quizlet, Notion AI, StudyFetch/Knowt, а також універсальні AI-

асистенти на зразок ChatGPT або Claude. Кожне з цих рішень має власні переваги, однак не всі вони орієнтовані на повний цикл роботи з лекційним відеоконтентом.

Google NotebookLM є інструментом для роботи з джерелами, нотатками та питаннями до матеріалів. Його перевагою є можливість працювати з різними документами та отримувати відповіді з урахуванням завантажених джерел. Такий підхід відповідає загальним принципам цифрового навчання та мультимедійного опрацювання навчального контенту [1], [2]. Однак NotebookLM є загальним інструментом для роботи з джерелами, а не спеціалізованою системою формування навчального набору у вигляді PDF-конспекту, карток, екзамену та окремої бібліотеки матеріалів.

Quizlet орієнтований насамперед на картки, практичні тести та інструменти повторення. Це робить його корисним для активного запам’ятовування, оскільки самоперевірка й повторення належать до ефективних стратегій навчання [3], [4]. Водночас такий підхід не повністю закриває задачу перетворення відеолекції на структурований PDF-конспект. Notion AI є сильним інструментом для роботи з нотатками та документами в межах робочого простору, проте він не є окремою спеціалізованою веб-системою для генерації навчальних наборів із відеоконтенту.

StudyFetch і Knowt ближчі до задачі автоматизованого навчання, оскільки підтримують створення навчальних матеріалів, карток і тестів. Такі функції загалом відповідають підходам активного навчання, практичного тестування та повторення матеріалу [3], [4]. Водночас вони є зовнішніми комерційними платформами, у яких користувач або розробник не контролює архітектуру, базу даних, формат зберігання матеріалів, логіку фонової обробки та інтеграції з конкретними AI- або storage-сервісами.

Універсальні AI-асистенти можуть допомогти користувачеві узагальнити текст, пояснити фрагмент документа або створити питання. Проте вони не забезпечують власної предметної структури навчальної системи: таблиць learning packs, карток, екзаменаційних наборів, навчальних спроб, колекцій PDF-документів і статусів фонової обробки. Саме тому доцільною є розробка спеціалізованої веб-системи Lectura.

1.4 Порівняння аналогів із веб-системою Lectura

Для визначення місця Lectura серед існуючих рішень було виконано порівняння за ключовими критеріями: підтримка джерел, генерація карток, тестів, чат із матеріалом, наявність повного pipeline і можливість контролю архітектури. Результати порівняння наведено в таблиці 1.2.

Таблиця 1.2 — Порівняння існуючих EdTech-рішень із веб-системою Lectura

Рішення	Документи	Картки	Тести	Чат	Pipeline
NotebookLM	+	±	±	+	±
Quizlet	±	+	+	−	±
Notion AI	+	−	−	+	−
StudyFetch/Knowt	+	+	+	±	±
AI-асистенти	+	±	±	+	−
Lectura	+	+	+	+	+

Порівняння показує, що більшість рішень спеціалізується на окремому сегменті задачі. Quizlet і Knowt зручні для роботи з картками, NotebookLM і Notion AI корисні для аналізу джерел, а універсальні AI-асистенти забезпечують гнучкий діалог. Проте Lectura поєднує кілька сценаріїв у межах однієї веб-системи: імпорт джерела, формування PDF-конспекту, створення карток, екзаменаційних питань, роботу з бібліотекою та контекстний чат.

Важливою відмінністю Lectura є контроль над архітектурою та даними. У межах бакалаврської роботи розробляється власна клієнт-серверна система, у якій можна описати структуру бази даних, API, алгоритми обробки, механізми авторизації, схему фонової обробки та особливості генерації навчальних матеріалів. Це дає змогу не лише використати існуючі AI-сервіси, а й інтегрувати їх у власну предметну модель навчального застосунку.

1.5 Постановка задачі розробки веб-системи

Задача бакалаврської роботи полягає у розробці веб-системи Lectura, яка забезпечує автоматизоване створення навчальних матеріалів на основі лекційного відеоконтенту з використанням технологій штучного інтелекту. Система має працювати як єдине середовище, у якому користувач може імпортувати джерело, отримати структурований PDF-конспект і надалі використовувати цей матеріал для повторення та самоперевірки.

Вхідними даними для системи є YouTube-посилання, аудіо- та відеофайли, PDF-документи, Markdown- і TXT-файли. Для аудіо- та відеоконтенту необхідно отримати текстову основу шляхом транскрибації. Для PDF-документів потрібно витягнути текстовий вміст, якщо документ має текстовий шар. Після цього текст має бути переданий до модуля генерації навчального звіту.

Вихідними даними системи є PDF-конспект, набір карток, набір екзаменаційних питань, відповіді AI-асистента, записи навчальних спроб і метадані навчальних матеріалів у бібліотеці користувача. Користувач повинен мати змогу повернутися до створених матеріалів, об’єднати кілька PDF у колекцію, поставити запитання до документа або пройти самоперевірку.

З погляду архітектури система має бути реалізована як клієнт-серверний веб-застосунок. Frontend відповідає за інтерфейс користувача, backend - за бізнес-логіку, авторизацію, роботу з базою даних, інтеграцію з AI-сервісами та формування PDF-файлів. Зберігання структурованих даних має виконуватися у реляційній базі даних, а PDF-файли мають розміщуватися в об’єктному сховищі.

1.6 Функціональні та нефункціональні вимоги

На основі аналізу предметної області та постановки задачі сформовано функціональні й нефункціональні вимоги до веб-системи Lectura. Функціональні вимоги описують можливості, які система повинна надавати користувачеві.

Нефункціональні вимоги визначають якісні характеристики системи: безпеку, зручність, розширюваність, стабільність роботи та обмеження ресурсів.

Таблиця 1.3 — Основні вимоги до веб-системи Lectura

Тип вимоги	Вимога	Пояснення
Функціональна	Реєстрація та авторизація	Користувач працює з власними матеріалами після входу в систему
Функціональна	Імпорт джерел	Підтримка відео, аудіо, YouTube-посилань, PDF, MD і TXT
Функціональна	Генерація PDF-конспекту	Створення структурованого навчального звіту у форматі PDF
Функціональна	Картки та екзамен	Формування інструментів повторення та самоперевірки
Функціональна	Контекстний AI-чат	Відповіді на запитання з урахуванням змісту PDF або колекції
Нефункціональна	Безпека доступу	JWT-авторизація, хешування паролів, перевірка власника ресурсу
Нефункціональна	Надійність обробки	Статуси задач queued, processing, ready, failed
Нефункціональна	Масштабованість	Модульна архітектура та можливість подальшого розширення
Нефункціональна	Зручність використання	Веб-інтерфейс для основних навчальних сценаріїв
Нефункціональна	Конфігурованість	Налаштування через змінні середовища та Docker Compose

До функціональних вимог належать реєстрація та авторизація користувача, імпорт навчальних джерел, транскрибація аудіо- та відеоконтенту, генерація PDF-конспекту, створення карток, генерація екзаменаційних питань, перевірка відповідей, робота з бібліотекою матеріалів, створення колекцій PDF-документів і контекстний чат.

Нефункціональні вимоги включають захист доступу до користувацьких ресурсів, хешування паролів, використання JWT-токенів, обмеження розміру

файлів, обробку помилок і можливість локального запуску через контейнеризоване середовище.

Узагальнений перелік вимог до веб-системи наведено в таблиці 1.3

Сформовані вимоги визначають межі поточної версії системи та водночас задають основу для її подальшого розвитку. У наступному розділі на основі цих вимог буде розглянуто методи й алгоритми обробки навчального контенту.

Вимоги було сформовано з урахуванням повного циклу роботи користувача з навчальним матеріалом. Користувач повинен мати змогу не лише завантажити джерело, а й отримати результат обробки, повернутися до нього пізніше та використати його для повторення. Тому функціональні вимоги охоплюють як початкові дії з імпорту матеріалу, так і подальші навчальні сценарії. Нефункціональні вимоги, у свою чергу, визначають умови, за яких ці сценарії мають виконуватися стабільно та безпечно. Особливе значення має захист користувацьких даних, оскільки кожен документ, набір карток або історія взаємодії з чатом належать конкретному користувачеві. Також важливо, щоб система коректно реагувала на помилки під час обробки великих файлів або взаємодії із зовнішніми AI-сервісами. Це дозволяє уникнути втрати проміжних результатів і зробити роботу застосунку більш передбачуваною. Модульність вимог також спрощує подальше розширення системи новими функціями. Наприклад, у майбутньому до переліку можливостей можна додати OCR, підтримку нових форматів або розширену аналітику навчального прогресу. Таким чином, вимоги описують не лише поточну реалізацію, а й напрям розвитку веб-системи.

2 МЕТОДИ ТА АЛГОРИТМИ ОБРОБКИ ВІДЕОКОНТЕНТУ І НАВЧАЛЬНИХ МАТЕРІАЛІВ

У другому розділі розглянуто методи й алгоритми, що забезпечують роботу веб-системи Lectura. На відміну від опису окремих програмних модулів, цей розділ зосереджений на логіці перетворення навчального джерела у структурований набір матеріалів. Основою системи є послідовний pipeline: отримання джерела, нормалізація контенту в текстове представлення, генерація структурованого навчального звіту, формування PDF-конспекту, створення карток, екзаменаційних питань і контексту для AI-чату.

Алгоритми системи мають враховувати різнорідність джерел, асинхронність довготривалих операцій, обмеження розміру файлів, необхідність контролю доступу до користувацьких матеріалів і залежність якості результату від вхідного контенту. Тому подальший опис подано як набір взаємопов’язаних методів, які в програмній реалізації об’єднуються в єдину веб-систему.

2.1 Загальна модель перетворення лекційного контенту в навчальний набір

У межах роботи лекційний контент розглядається як первинне джерело знань, яке ще не є достатньо зручним для швидкого повторення. Користувач може мати відеозапис лекції, аудіофайл, YouTube-посилання, PDF-документ, Markdown-файл або TXT-файл. Незалежно від типу джерела система повинна привести його до уніфікованої текстової форми, після чого сформувати навчальний набір.

Загальну модель перетворення лекційного контенту в навчальний набір наведено на рисунку 2.1.

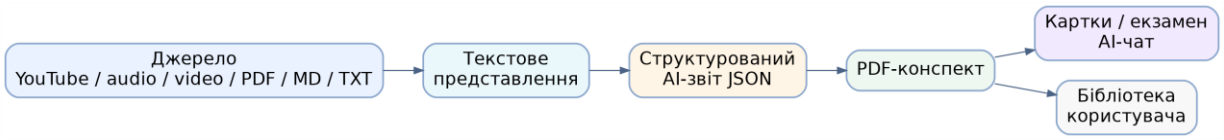

Рисунок 2.1 — Узагальнений pipeline обробки навчального контенту

Першим етапом є отримання джерела. Якщо користувач завантажує PDF, Markdown або TXT-документ, система працює з текстовим або документним представленням. Якщо джерелом є відео чи аудіо, необхідним проміжним етапом стає транскрибація. Після отримання тексту система передає його до AI-моделі, яка формує структурований звіт у JSON-форматі. Далі звіт перетворюється у PDF-конспект, а пов’язані з ним дані використовуються для генерації карток, екзаменаційних питань і контекстного чату.

Важливою властивістю такої моделі є розділення етапів. Система не формує одразу кінцевий PDF як суцільний текст. Спочатку створюється проміжна структурована модель звіту, яка містить окремі смислові блоки. Це дає змогу контролювати формат результату, повторно використовувати фрагменти матеріалу та розширювати систему без повної зміни pipeline.

2.2 Алгоритм імпорту та нормалізації джерел

Алгоритм імпорту відповідає за приймання джерела від користувача, визначення його типу та підготовку до подальшої обробки. Система підтримує кілька типів вхідних матеріалів, тому універсальний алгоритм повинен виконувати маршрутизацію залежно від формату джерела.

Основні типи джерел і способи їх нормалізації наведено в таблиці 2.1.

Таблиця 2.1 — Типи джерел і спосіб нормалізації

Тип джерела	Спосіб обробки	Результат нормалізації
YouTube-посилання	отримання медіаджерела, виділення аудіо, транскрибація	текстовий транскрипт лекції
аудіофайл	передавання аудіо до сервісу транскрибації	текстовий транскрипт
відеофайл	виділення аудіодоріжки та транскрибація	текстовий транскрипт
PDF-документ	витяг тексту зі сторінок документа через pypdf	текст документа або конспекту
Markdown / TXT	читання текстового вмісту та очищення службових символів	уніфікований текст

Алгоритм імпорту складається з перевірки типу джерела, перевірки розміру файлу, створення запису про обробку, вибору відповідного обробника та формування уніфікованого текстового представлення.

Схему алгоритму імпорту та нормалізації джерел наведено на рисунку 2.2.

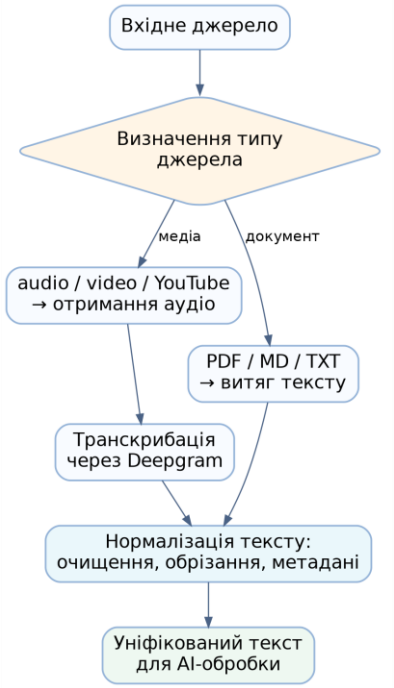

Рисунок 2.2 — Алгоритм імпорту та нормалізації джерел

У системі важливо не змішувати логіку різних типів джерел. Наприклад, PDF-документ не потребує транскрибації, але може потребувати витягування текстового шару. Відеофайл, навпаки, не має безпосередньої текстової структури, тому його потрібно спочатку перетворити на аудіо, а потім отримати транскрипт. Такий поділ робить алгоритм розширюваним: у майбутньому можна додати підтримку DOCX, презентацій або OCR для сканованих документів без зміни всієї системи.

2.3 Алгоритм транскрибації аудіо- та відеоматеріалів

Для аудіо- та відеоматеріалів ключовим етапом є транскрибація, тобто перетворення мовлення в текст. У веб-системі Lectura для цього використовується зовнішній сервіс Deepgram, а сама задача розглядається як приклад автоматичного розпізнавання мовлення [5]. Такий підхід дає змогу винести складну задачу розпізнавання мовлення за межі власного backend-коду та використовувати готовий API для роботи з попередньо записаними матеріалами.

Алгоритм транскрибації включає такі кроки:

1. Перевірка типу й розміру вхідного файлу.
2. Підготовка аудіопотоку.
3. Передавання даних до сервісу транскрибації.
4. Отримання текстового результату.
5. Збереження транскрипту.
6. Передавання тексту на етап генерації навчального звіту.

Для YouTube-посилання додатково виконується отримання або підготовка аудіоданих перед транскрибацією.

Отриманий транскрипт не є кінцевим навчальним матеріалом. Він може містити повтори, неповні фрази, службові вставки або помилки розпізнавання. Тому наступним етапом є нормалізація та структуризація тексту за допомогою AI-

моделі. У базі даних зберігаються метадані матеріалу, а великий текст транскрипту може зберігатися у файловій структурі поруч із результатами генерації.

2.4 Метод формування структурованого навчального звіту

Ключовою особливістю Lectura є те, що AI-модель використовується не лише для стислого переказу, а для формування структурованого навчального звіту.

Основні поля такого представлення наведено в таблиці 2.2.

Таблиця 2.2 — Структура проміжного JSON-звіту

Поле	Призначення
title	назва навчального матеріалу, сформована за змістом джерела
summary	коротке узагальнення теми та основних ідей
key_points	список ключових тез для швидкого повторення
sections	тематичні розділи з абзацами пояснень
tables	табличні узагальнення понять, порівнянь або класифікацій
charts	опис простих графіків або діаграм, якщо вони доречні для теми
glossary	перелік термінів і визначень
labels	локалізовані назви блоків у PDF-звіті

Формування структурованого звіту виконується за допомогою сучасних мовних моделей, що спираються на розвиток нейромережевих і transformer-підходів до обробки природної мови [6], [7], [8]. На вхід моделі передається текст джерела та інструкція щодо бажаної структури відповіді. На виході система очікує валідну JSON-структуру. Якщо відповідь не відповідає очікуваній схемі, backend може застосувати повторну спробу генерації, спрощення структури або fallback-логіку формування текстового звіту.

Перевага проміжного JSON-представлення полягає в тому, що воно відокремлює зміст від оформлення. AI-модель відповідає за смислове структурування матеріалу, а backend - за перевірку, форматування, перетворення у HTML і генерацію PDF. Завдяки цому один і той самий звіт можна відобразити в різних форматах або використати для додаткових навчальних модулів.

2.5 Алгоритм генерації PDF-конспекту

PDF-конспект є основним видимим результатом роботи системи. Він має бути придатним для читання, повторення та збереження в бібліотеці користувача. Алгоритм генерації PDF-конспекту починається після того, як сформовано структурований JSON-звіт.

Схему формування PDF-конспекту наведено на рисунку 2.3.

Рисунок 2.3 — Формування структурованого звіту та PDF-конспекту

Після цього HTML перетворюється у PDF за допомогою WeasyPrint. Якщо основний механізм недоступний, може використовуватися резервна генерація через ReportLab.

У базі даних не зберігається сам PDF-файл. Зберігаються метадані документа, власник, назва та ключ об’єкта у файловому сховищі. Такий підхід зменшує навантаження на PostgreSQL і дає змогу використовувати об’єктне сховище для великих файлів. Локальна файлова система використовується як проміжний етап генерації та кешування.

2.6 Алгоритм генерації карток для повторення

Картки призначені для активного повторення матеріалу. На відміну від PDF-конспекту, який підтримує читання та перегляд, картки змушують користувача відтворювати знання. У системі Lectura картки можуть створюватися автоматично на основі PDF-документа або learning pack, а також вручну користувачем.

У таблиці 2.3 наведено основні дані, що використовуються під час формування карток.

Розділення card set і окремих cards дає змогу створювати кілька наборів карток для одного PDF-документа. Наприклад, користувач може створити короткий набір для швидкого повторення та окремий набір підвищеної складності для підготовки до екзамену.

Таблиця 2.3 — Дані модуля карток

Сутність	Основні поля	Призначення
card_sets	id, user_id, pdf_id, pack_id, title, difficulty, mode, status	зберігання набору карток
cards	id, set_id, user_id, pdf_id, question, answer, context	зберігання окремих питань і відповідей
cards repetition fields	next_review_at, ease, streak, last_result, last_reviewed_at	планування повторення картки
study_attempts	user_answer, expected_answer, is_correct, feedback, duration_ms	фіксація відповіді користувача

2.7 Механізм spaced repetition

Для підвищення ефективності повторення в системі реалізовано базовий механізм spaced repetition. Його задача полягає в тому, щоб частіше показувати користувачеві складні картки та рідше повертатися до тих, на які він відповідає

правильно. У поточній версії використовується бінарна оцінка відповіді: correct або wrong. Такий підхід простіший за повноцінні алгоритми на зразок Anki, але узгоджується з дослідженнями ефективності самоперевірки та розподіленого повторення [3], [4].

Після кожної навчальної спроби система оновлює параметри картки та зберігає результат у базі даних. Це дозволяє реалізувати персоналізоване повторення навіть без складної аналітики

Логіку оновлення параметрів картки наведено в таблиці 2.4.

Таблиця 2.4 — Логіка оновлення параметрів картки

Результат відповіді	Оновлення параметрів	Дата наступного повторення
correct	streak = streak + 1; ease = min(ease + 0.15, 3.5); last_result = "correct"	now + max(1, streak) days
wrong	streak = 0; ease = max(ease - 0.2, 1.3); last_result = "wrong"	now + 1 day

У майбутньому алгоритм можна розширити шкалою again, hard, good, easy, а також більш точним розрахунком інтервалу повторення.

2.8 Алгоритм генерації та перевірки екзаменаційних питань

Екзаменаційний модуль призначений для самоперевірки після опрацювання PDF-конспекту. На відміну від карток, які частіше мають коротку форму питання-відповідь, екзаменаційні питання можуть вимагати вибору варіанта, пояснення причинно-наслідкового зв’язку або короткої розгорнутої відповіді.

Основні вхідні та вихідні дані екзаменаційного алгоритму наведено в таблиці 2.5.

Таблиця 2.5 — Вхідні та вихідні дані екзаменаційного модуля

Етап	Вхідні дані	Вихідні дані
генерація quiz set	PDF або learning pack, кількість питань, складність	набір quiz_items
проходження питання	question, options_json, user_answer	відповідь користувача
перевірка відповіді	user_answer, expected_answer, context	is_correct, feedback, confidence
збереження спроби	результат перевірки та метадані	запис у study_attempts

Після проходження екзамену відповідь користувача передається на перевірку, а результат зберігається у таблиці study_attempts.

Такий підхід дозволяє відокремити створення питань від їх проходження. Один згенерований quiz set може бути використаний повторно, а історія спроб користувача може застосовуватися для подальшої аналітики прогресу.

У поточній версії основний акцент зроблено на перевірці окремої відповіді та збереженні навчальної спроби.

2.9 Метод контекстного AI-чату за PDF-документом і колекцією

Контекстний AI-чат дає змогу користувачеві ставити запитання не до абстрактної моделі, а до конкретного PDF-документа або колекції документів. Це важливо для навчальної системи, оскільки користувач очікує відповідь саме в межах завантаженого матеріалу, а не загальне пояснення теми.

У системі текст PDF-документа витягується з документа та використовується як контекст для відповіді. Отриманий текст обмежується параметром PDF_CONTEXT_MAX_CHARS, після чого разом із запитанням користувача передається до AI-моделі. Для колекції документів тексти кількох PDF

об’єднуються в один контекст із позначеннями DOCUMENT 1, DOCUMENT 2 тощо.

Схему формування відповіді контекстного AI-чату наведено на рисунку 2.4.

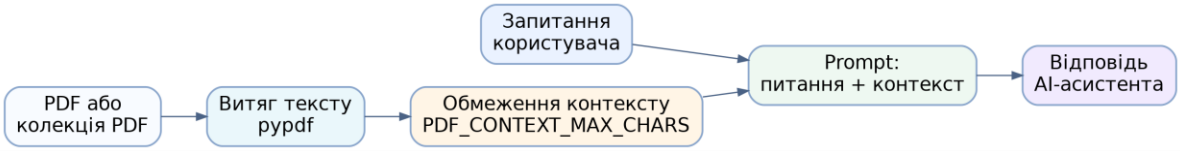

Рисунок 2.4 — Метод контекстного AI-чату за PDF-документом і колекцією

Метод контекстного чату можна описати такими кроками:

1. Користувач відкриває PDF або колекцію.
2. Система визначає джерело контексту.
3. Текст витягується або читається з кешу.
4. Обсяг тексту обмежується.
5. Формується prompt із питанням і контекстом.
6. AI-модель генерує відповідь.
7. Відповідь повертається у frontend-інтерфейс.

У поточній версії системи не використовується повноцінний vector search або окреме RAG-сховище. Для великих колекцій у майбутньому доцільно реалізувати поділ документів на фрагменти, побудову embeddings і пошук найбільш релевантних частин перед формуванням відповіді, що відповідає підходу retrieval-augmented generation [9].

Такий підхід дозволяє користувачеві ставити питання не до абстрактної AI-моделі, а до конкретного навчального матеріалу. Контекст документа обмежує відповідь змістом завантаженого PDF або колекції, що зменшує ймовірність нерелевантних пояснень. Водночас система повинна враховувати обмеження обсягу тексту, який може бути переданий до моделі за один запит. Тому важливо попередньо скорочувати або відбирати найбільш значущі фрагменти матеріалу.

3 ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-СИСТЕМИ LECTURA

У третьому розділі наведено проєктування та програмну реалізацію веб-системи Lectura. Якщо у другому розділі було описано методи та алгоритми перетворення навчального контенту в структурований набір матеріалів, то в цьому розділі розглянуто практичне втілення цих алгоритмів у вигляді клієнт-серверної системи. Особливу увагу приділено архітектурі застосунку, структурі програмного проєкту, базі даних, REST API, авторизації, імпорту джерел, генерації PDF-конспектів, роботі з картками, екзаменаційним модулем і контейнеризованому запуску.

Розроблена система має модульну будову. Backend-частина відповідає за бізнес-логіку, захищені API-запити, взаємодію з базою даних, зовнішніми AI-сервісами та файловим сховищем. Frontend-частина відповідає за відображення інтерфейсу користувача та надсилання запитів до API. Такий поділ дозволяє спростити супровід системи, ізолювати критичні ключі доступу на сервері та в майбутньому розширювати функціональність без повної перебудови застосунку.

3.1 Загальна архітектура веб-системи

Lectura реалізована як клієнт-серверна веб-система. Користувач взаємодіє з нею через браузер, а основні операції виконуються на серверному рівні. Frontend надсилає HTTP-запити до FastAPI backend, backend перевіряє авторизацію, виконує бізнес-логіку, звертається до PostgreSQL, файлового сховища Cloudflare R2 та зовнішніх сервісів OpenAI API і Deepgram.

Загальну архітектуру веб-системи Lectura наведено на рисунку 3.1.

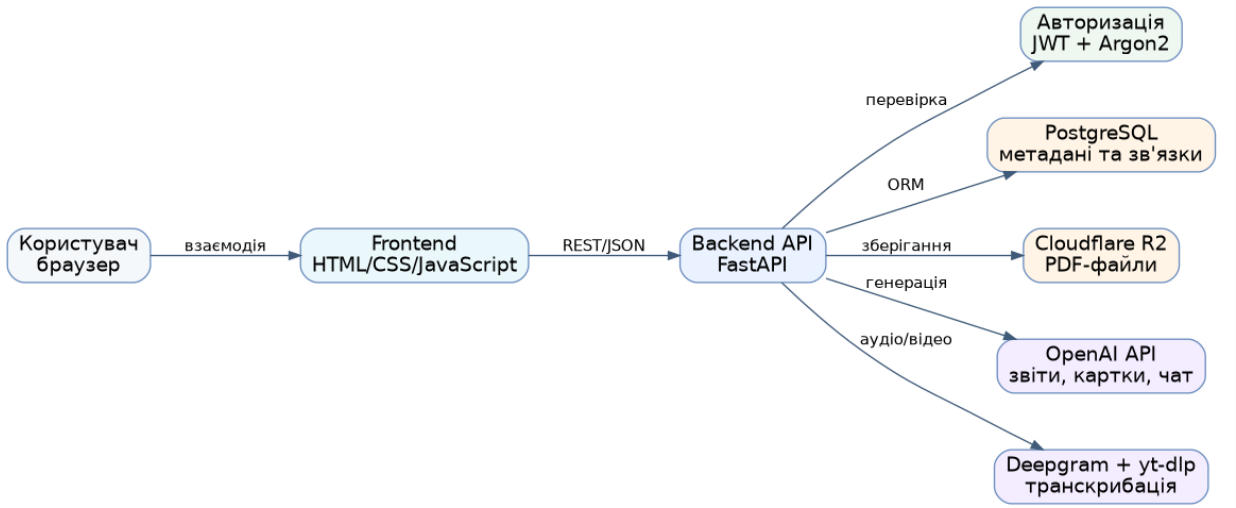

Рисунок 3.1 — Загальна архітектура веб-системи Lectura

У наведеній архітектурі frontend не звертається напряму до зовнішніх AI-сервісів або об’єктного сховища. Це принципове рішення, оскільки API-ключі OpenAI, Deepgram і Cloudflare R2 не повинні потрапляти до браузера користувача. Усі операції, пов’язані з транскрибацією, генерацією навчальних матеріалів, збереженням PDF-файлів і перевіркою прав доступу, виконуються backend-частиною. Такий поділ відповідальності узгоджується з принципами чистої архітектури програмних систем [10].

Серверна частина також виконує роль контролера стану довготривалих операцій. Наприклад, обробка відео або YouTube-посилання не може виконуватися миттєво в межах одного короткого запиту.

3.2 Структура програмного проєкту

Програмний проєкт поділено на backend- і frontend-частини. Backend розміщено в директорії Back, а клієнтські HTML-, CSS- і JavaScript-файли - у директорії Front. Окремо винесено тести, міграції бази даних, Docker-конфігурацію та приклад файлу змінних середовища.

Структуру основних директорій і файлів програмного проєкту наведено на рисунку 3.2.

Рисунок 3.2 — Структура програмного проєкту Lectura

У backend-частині директорія app/api містить API-маршрути, app/core - базову конфігурацію, засоби авторизації та безпеки, app/infra - інфраструктурну логіку взаємодії з базою даних і зовнішніми сервісами, app/reporting - модулі формування структурованого звіту й PDF-документа, а app/storage - логіку роботи з файловим сховищем. Директорія alembic використовується для міграцій структури бази даних.

Frontend-частина реалізована без фреймворків на основі HTML, CSS і JavaScript. Такий підхід було обрано для зменшення складності клієнтського рівня та швидкої реалізації основних сценаріїв. Основними сторінками є index.html, app.html, dashboard.html, viewer.html, flashcards.html, cards-study.html, quizzes.html і study.html.

Призначення основних частин програмного проєкту наведено в таблиці 3.1.

Таблиця 3.1 — Основні частини програмного проєкту

Елемент	Призначення
Back/app/api	Маршрути REST API для авторизації, PDF, learning packs, jobs, карток, екзаменів і AI-чату
Back/app/core	Конфігурація системи, JWT, параметри безпеки, робота зі змінними середовища
Back/app/infra	Інфраструктурні сервіси: база даних, інтеграції, зовнішні API
Back/app/reporting	Формування структурованого AI-звіту, HTML-представлення та PDF-конспекту
Back/app/storage	Завантаження та отримання PDF-файлів з Cloudflare R2
Back/alembic	Міграції PostgreSQL, що відстежують зміну структури таблиць
Front/*.html	Сторінки авторизації, створення learning pack, бібліотеки, перегляду PDF, карток та екзамену
Front/*.js	Логіка авторизації, роботи зі станом застосунку, конфігурації API та відображення UI
tests	Backend- і frontend-тести для перевірки ключових сценаріїв

Такий розподіл директорій робить структуру проєкту зрозумілою і дозволяє локалізувати зміни. Наприклад, модифікація генератора PDF-конспекту не потребує змін у сторінках frontend, а зміна логіки відображення карток не впливає на схему бази даних. Підтримуваність такої структури пов’язана із загальними практиками поступового покращення та рефакторингу програмного коду [11].

3.3 Проєктування бази даних

Для зберігання даних використовується PostgreSQL. Вибір реляційної бази даних зумовлений тим, що система містить багато взаємопов'язаних сутностей: користувачів, PDF-документи, навчальні набори, колекції, картки, екзаменаційні питання, навчальні спроби. Реляційна модель забезпечує транзакційність, цілісність зв'язків і зручну роботу з табличними даними [12], [13].

Для опису моделей і виконання запитів використовується SQLAlchemy, а для керування змінами структури бази даних - Alembic.

Узагальнену ER-діаграму бази даних Lectura наведено на рисунку 3.3.

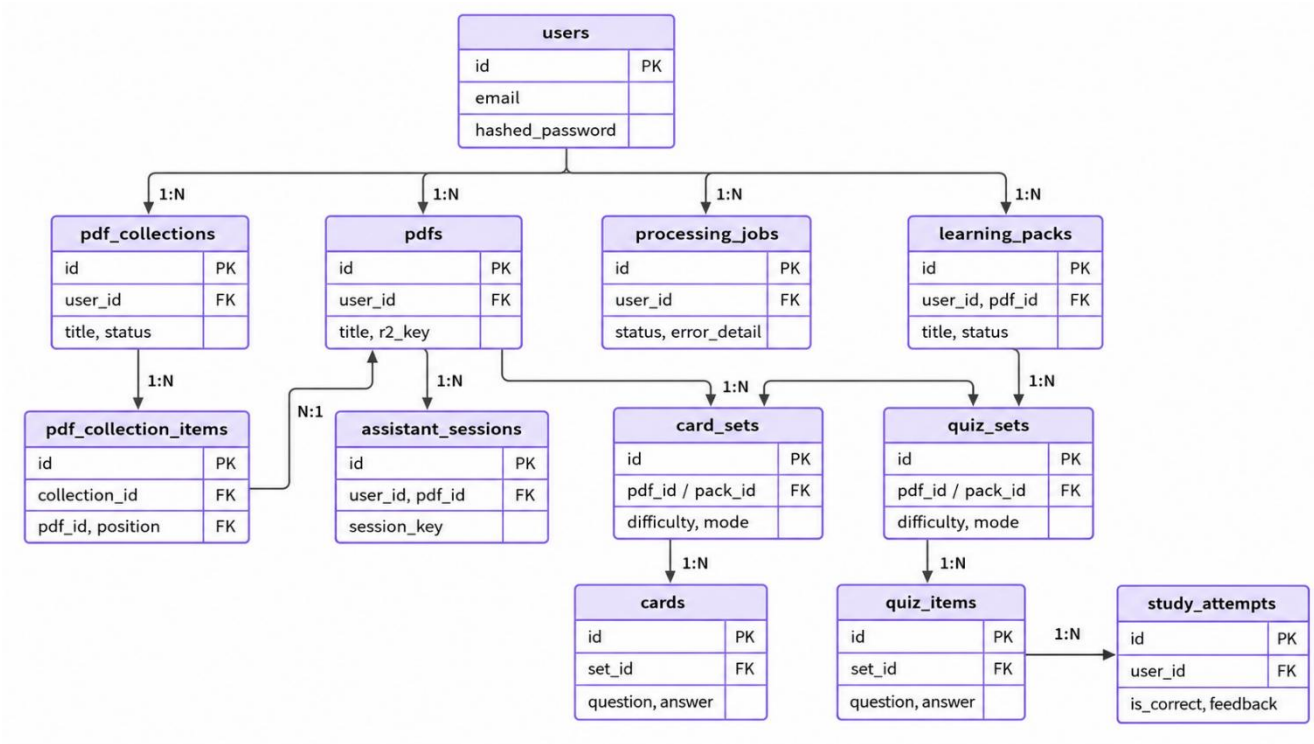

Рисунок 3.3 — Узагальнена ER-діаграма бази даних Lectura

База даних не зберігає великі PDF-файли безпосередньо в таблицях. У таблиці pdfs зберігаються метадані та ключ r2_key.

Опис основних таблиць бази даних наведено в таблиці 3.2.

Таблиця 3.2 — Основні таблиці бази даних Lectura

Таблиця	Призначення	Основні поля
users	Зберігання облікових записів користувачів	id, email, hashed_password, created_at
pdfs	Метадані PDF-документів і ключі файлів у R2	id, user_id, title, r2_key, created_at
learning_packs	Навчальні набори, пов’язані з джерелами	id, user_id, pdf_id, collection_id, title, status
processing_jobs	Контроль довготривалих задач обробки	id, user_id, job_type, status, result_json, error_detail
card_sets / cards	Набори карток і окремі картки для повторення	set_id, question, answer, next_review_at, ease, streak
quiz_sets / quiz_items	Екзаменаційні набори та питання	set_id, quiz_type, question, options_json, answer
assistant_sessions	Сесії контекстного AI-чату	id, user_id, pdf_id, session_key, last_activity_at
study_attempts	Навчальні спроби користувача	user_answer, expected_answer, is_correct, feedback, duration_ms
usage_events	Події використання та контроль лімітів	event_type, payload_json, created_at

Особливістю реалізації є те, що не всі зв’язки дублюються в дочірніх таблицях.

3.4 Проєктування REST API

Взаємодія frontend- і backend-частин реалізована через REST API. Кожна група endpoints відповідає окремому

Основні групи API-інтерфейсів наведено в таблиці 3.3.

Таблиця 3.3 — Основні групи REST API

Група	Endpoints	Призначення
Auth	POST /api/auth/register; POST /api/auth/login; GET /api/auth/me	Реєстрація, авторизація та отримання інформації про поточного користувача
Packs	GET /api/packs; GET /api/packs/{pack_id}; POST /api/packs/{pack_id}/cards; POST /api/packs/{pack_id}/exam	Робота з навчальними наборами, картками та екзаменами
PDF / Library	GET /api/pdfs; GET /api/pdfs/{pdf_id}; POST /api/import/document	Імпорт документів і отримання PDF-матеріалів користувача
Collections	GET /api/collections; POST /api/collections; GET /api/collections/{collection_id}	Створення та перегляд колекцій PDF-документів
Jobs	GET /api/jobs; GET /api/jobs/{job_id}; POST /api/jobs/transcribe/audio; POST /api/jobs/transcribe/youtube	Створення та перевірка стану довготривалих задач обробки
Assistant	POST /api/assistant/ask; POST /api/assistant/cards; POST /api/assistant/quizzes; POST /api/assistant/check	AI-чат, генерація карток, екзаменів і перевірка відповідей

FastAPI обрано як основний backend-фреймворк, оскільки він підтримує типізовані схеми запитів і відповідей, зручну побудову REST API та автоматичну генерацію OpenAPI-опису. Проєктування API у роботі спирається на принципи мережевої взаємодії та REST-архітектури [14].

Для захищених endpoints backend очікує JWT-токен. Після перевірки токена система визначає поточного користувача та виконує додаткову перевірку належності ресурсу. Це важливо для таких об’єктів, як pdfs, learning_packs,

card_sets, quiz_sets і collections, оскільки кожен користувач має працювати лише зі своїми навчальними матеріалами.

API також враховує довготривалі операції. Замість очікування завершення транскрибації або генерації PDF у межах одного HTTP-запиту frontend створює задачу та періодично перевіряє її статус. Це робить інтерфейс стійкішим до затримок зовнішніх сервісів.

Основні групи API-інтерфейсів наведено в таблиці 3.3.

3.5 Реалізація авторизації та контролю доступу

Авторизація користувачів реалізована за допомогою email, пароля та JWT-токенів. Після успішної реєстрації або входу користувач отримує access token, який використовується у подальших запитах до захищених API-методів. Паролі користувачів не зберігаються у відкритому вигляді: для них використовується хешування. Такий підхід відповідає базовим принципам захисту веб-застосунків і керування доступом [15], [16].

Механізм доступу складається з двох рівнів. Перший рівень - перевірка самого токена, тобто встановлення факту, що запит виконує авторизований користувач. Другий рівень - перевірка належності конкретного ресурсу. Наприклад, якщо користувач відкриває PDF за ідентифікатором, backend перевіряє, чи належить цей документ саме цьому користувачеві.

Послідовність авторизації можна описати так:

1. Користувач надсилає email і пароль.
2. Backend перевіряє пароль за хешем.
3. У разі успіху формується JWT.
4. Frontend зберігає токен і додає його до запитів.
5. Backend перевіряє токен і user ownership перед виконанням дії.

Такий підхід запобігає ситуаціям, коли користувач може змінити ідентифікатор ресурсу в URL і отримати доступ до чужого PDF-документа, набору

карток або екзаменаційного набору. Перевірка user ownership є обов’язковою для всіх сутностей, пов’язаних із навчальними матеріалами.

3.6 Реалізація імпорту навчального контенту та фонової обробки

Модуль імпорту навчального контенту підтримує кілька типів джерел: YouTube-посилання, аудіофайли, відеофайли, PDF-документи, Markdown і TXT. Незалежно від типу джерела система прагне привести матеріал до текстового представлення, яке далі може бути передане до AI-моделі.

Основні стани фонової задачі наведено в таблиці 3.4.

Таблиця 3.4 — Стани фонових задач обробки

Стан	Значення для користувача	Дія системи
queued	Задача створена й очікує виконання	Збереження запису processing_jobs і вхідних параметрів
processing	Матеріал обробляється	Транскрибація, AI-генерація, формування PDF або імпорт документа
ready	Результат сформовано успішно	Збереження result_json, оновлення пов’язаних pdf_id або pack_id
failed	Під час обробки сталася помилка	Збереження error_code та error_detail для відображення помилки

Для обробки довготривалих задач використовується таблиця processing_jobs. Після створення задачі frontend отримує її ідентифікатор і періодично перевіряє стан. Задача може перебувати у статусах queued, processing, ready або failed. У разі

помилки зберігаються error_code та error_detail, що дозволяє відобразити повідомлення користувачеві та полегшити налагодження.

Реалізація фонової обробки має кілька переваг. По-перше, вона не блокує браузер під час транскрибації або генерації PDF. По-друге, вона дозволяє зберігати проміжні результати та стан задачі. По-третє, така модель у майбутньому може бути розширена до окремого worker-рівня без зміни логіки frontend.

3.7 Реалізація генерації PDF-конспекту

Генерація PDF-конспекту реалізована як послідовність перетворень. Спочатку з джерела отримується текст, далі AI-модель формує структуроване JSON-представлення навчального звіту, після чого це представлення перетворюється у HTML і PDF. Для перетворення HTML у PDF використовується WeasyPrint, а як резервний механізм передбачено ReportLab.

Використання проміжного JSON-представлення є важливим рішенням. Воно дозволяє відокремити зміст звіту від його оформлення. AI-модель формує логічну структуру: назву, summary, key_points, sections, tables, charts і glossary. Після цього backend відповідає за стабільну верстку, формування HTML-шаблону, підключення стилів і створення PDF-файлу.

Згенерований PDF зберігається локально на етапі обробки, після чого завантажується в Cloudflare R2. У базі даних зберігається r2_key, за яким система може знайти відповідний файл. Такий підхід відокремлює структуровані дані від великих файлових об’єктів і спрощує масштабування файлового зберігання [13].

Окремо реалізовано роботу з кирилицею у PDF. Це важливо, оскільки навчальні матеріали можуть бути українською мовою і містити спеціальні символи, терміни або назви. Перевірка коректного відображення кирилиці винесена в окремі тести генерації звітів.

3.8 Реалізація карток, екзаменаційного модуля та AI-чату

Картки, екзаменаційні питання та AI-чат реалізовано як окремі функціональні модулі, але всі вони працюють із тими самими навчальними джерелами. Центральними об’єктами є PDF-документ і learning pack. Це дозволяє користувачеві не дублювати матеріали, а використовувати один сформований конспект для кількох сценаріїв навчання.

Модуль карток використовує таблиці card_sets і cards. Набір карток може бути створений автоматично за допомогою AI або доповнений вручну користувачем. Кожна картка містить питання, відповідь, контекст і параметри повторення. Після відповіді користувача оновлюються streak, ease, last_result, last_reviewed_at і next_review_at.

Зіставлення функціональних модулів і відповідних сутностей бази даних наведено в таблиці 3.5.

Таблиця 3.5 — Зв’язок функціональних модулів із таблицями бази даних

Модуль	Основні таблиці	Результат для користувача
Бібліотека матеріалів	pdfs, learning_packs, usage_events	Список створених PDF-конспектів і навчальних наборів
Колекції PDF	pdf_collections, pdf_collection_items	Групування кількох PDF-документів у спільний контекст
Картки	card_sets, cards, study_attempts	Питання-відповіді для повторення та оновлення next_review_at
Екзамен	quiz_sets, quiz_items, study_attempts	Набір питань для самоперевірки та feedback
AI-чат	assistant_sessions, pdfs	Відповіді на запитання з урахуванням контексту документа

Екзаменаційний модуль використовує таблиці quiz_sets, quiz_items і study_attempts. AI-модель формує питання, варіанти відповідей або очікувану відповідь, а після проходження система зберігає спробу користувача. Це дозволяє аналізувати, які питання були пройдені та чи була відповідь правильною.

AI-чат працює з текстом конкретного PDF-документа або колекції. Для PDF текст витягується з документа, після чого обмежується параметром PDF_CONTEXT_MAX_CHARS. Для колекції тексти кількох документів об’єднуються у спільний контекст із позначеннями DOCUMENT 1, DOCUMENT 2 тощо. У поточній реалізації чат не використовує окреме векторне сховище, але така можливість може бути додана в майбутньому на основі RAG-підходу [9].

3.9 Контейнеризація та налаштування середовища

Для спрощення запуску системи використовується Docker Compose. Контейнеризація дозволяє описати необхідні сервіси, залежності та параметри запуску в одному конфігураційному файлі. Такий підхід зменшує кількість ручних дій під час підготовки середовища та робить запуск системи відтворюваним, що відповідає практикам побудови сучасних сервісних і мікросервісних систем [17], [18].

Конфігураційні параметри винесено в env-змінні. У репозиторії використовується безпечний шаблон .env.example без реальних секретів. Реальні ключі доступу до OpenAI API, Deepgram і Cloudflare R2 не повинні потрапляти в текст роботи або відкриті файли. У дипломній роботі доцільно описувати лише назви змінних і їхнє призначення.

Основні групи конфігураційних параметрів наведено в таблиці 3.6.

Таблиця 3.6 — Основні групи змінних середовища

Група параметрів	Приклади змінних	Призначення
База даних	DATABASE_URL	Підключення backend-частини до PostgreSQL
Авторизація	JWT_SECRET	Підпис і перевірка JWT-токенів
AI-сервіси	OPENAI_API_KEY, ASSISTANT_MODEL, SUMMARIZER_MODEL	Генерація конспектів, карток, екзаменів і відповідей чату
Транскрибація	DEEPGRAM_API_KEY, DEEPGRAM_MODEL, FFMPEG_PATH	Обробка аудіо- та відеоматеріалів
Файлове сховище	R2_ENDPOINT, R2_BUCKET, R2_ACCESS_KEY_ID, R2_SECRET_ACCESS_KEY	Завантаження та отримання PDF-файлів
Обмеження	MAX_UPLOAD_BYTES, SAAS_MAX_GENERATIONS_PER_MONTH, PDF_CONTEXT_MAX_CHARS	Контроль розміру файлів, кількості генерацій і обсягу контексту

Локальний запуск системи виконується командою Docker Compose. Після запуску користувач відкриває frontend-сторінки в браузері, а backend забезпечує роботу API, бази даних, файлового сховища та інтеграцій.

4 ТЕСТУВАННЯ, ОЦІНЮВАННЯ ТА ДЕМОНСТРАЦІЯ РОБОТИ СИСТЕМИ

У четвертому розділі наведено методику перевірки веб-системи Lectura, описано backend- і frontend-тестування, перевірку основних сценаріїв користувача, демонстрацію роботи інтерфейсу, а також обмеження поточної версії та напрями подальшого розвитку. Тестування розглядається не лише як перевірка окремих функцій, а як підтвердження працездатності повного циклу: від імпорту навчального джерела до створення PDF-конспекту, карток, екзаменаційних питань і контекстного чату.

Особливістю системи є поєднання декількох різних типів операцій: робота з файлами, транскрибація, AI-генерація, збереження даних у PostgreSQL, передавання PDF-файлів у Cloudflare R2, взаємодія з користувачем через браузер і контроль доступу через JWT. Через це тестування охоплювало як програмні модулі backend, так і сценарії frontend-рівня, безпеку відображення даних та коректність результатів генерації.

4.1 Методика тестування програмної системи

Метою тестування було перевірити, що система коректно виконує основні функції, обробляє помилки, не допускає доступу до чужих ресурсів і формує придатні для використання навчальні матеріали. Для цього застосовано поєднання автоматизованих backend- і frontend-тестів та ручної перевірки користувацьких сценаріїв у браузері.

Загальну модель тестування веб-системи наведено на рисунку 4.1.

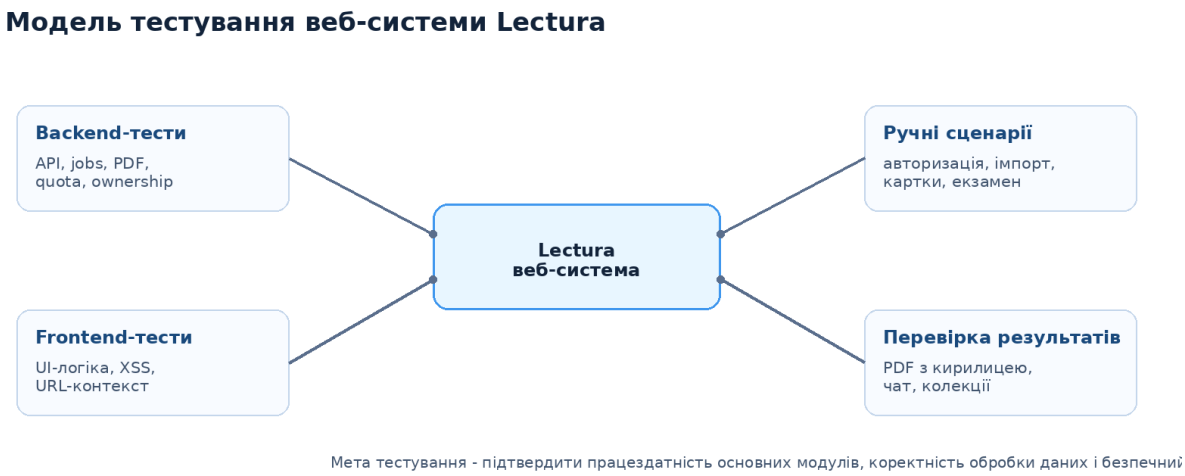

Рисунок 4.1 — Модель тестування веб-системи Lectura

Ручна перевірка дозволяє оцінити цілісний сценарій роботи користувача, який не завжди повністю відображається окремими unit-тестами. Основні напрями тестування наведено в таблиці 4.1.

Таблиця 4.1 — Основні напрями тестування системи

Напрям	Що перевіряється	Спосіб перевірки
Авторизація та доступ	Реєстрація, вхід, видача access token, належність ресурсів користувачу	Backend-тести, ручна перевірка API і браузера
Імпорт і обробка джерел	Імпорт Markdown/PDF, audio upload pipeline, YouTube pipeline, статуси processing_jobs	Backend-тести, ручні сценарії
Генерація навчальних матеріалів	PDF-конспект, картки, екзаменаційні питання, коректне збереження результатів	Backend-тести, перегляд результатів у UI
AI-чат і колекції	Робота асистента з PDF та колекцією кількох документів	Backend-тести, ручне тестування viewer
Frontend-безпека	Заборона небезпечного використання innerHTML, зниження XSS-ризиків	node:test

Тестування поділено на кілька рівнів. Backend-тести перевіряють бізнес-логіку, API, обробку даних і статуси задач. Frontend-тести спрямовані на клієнтську логіку та безпечне відображення даних. Ручна перевірка дозволяє оцінити цілісний сценарій роботи користувача, який не завжди повністю відображається окремими unit-тестами. Основні напрями тестування наведено в таблиці 4.1.

Такий підхід дозволяє перевірити не лише успішні сценарії, а й граничні ситуації: помилки обробки, перевищення лімітів, некоректні дані, спроби доступу до ресурсів іншого користувача та ризики небезпечного відображення тексту на frontend-рівні.

4.2 Backend-тестування

Backend-тестування було спрямоване на перевірку серверної логіки, API, роботи з моделями даних, механізму авторизації, фонової обробки, генерації PDF і модулів навчання. У поточній реалізації backend-тести розміщено у файлах test_product_hardening.py та test_report_fonts.py. Вони перевіряють як звичайні функціональні сценарії, так і окремі ситуації, пов’язані з надійністю продукту. Серверний код реалізовано мовою Python, що є основною мовою backend-частини системи [19].

Окрему увагу приділено перевірці коректності PDF-документів, оскільки PDF-конспект є головним результатом роботи системи. У тестах перевіряється, що документ не втрачає українські символи, а текст залишається читабельним. Це важливо, оскільки більшість навчальних матеріалів у роботі орієнтована на українськомовного користувача.

Перелік основних backend-перевірок наведено в таблиці 4.2.

Таблиця 4.2 — Backend-тести системи Lectura

Група перевірок	Приклади сценаріїв	Очікуваний результат
Авторизація	Реєстрація користувача, видача access_token, отримання поточного користувача	Користувач створюється, токен повертається, захищені endpoints доступні лише авторизованому користувачу
Learning pack та імпорт	Створення legacy learning pack, імпорт Markdown-документа, audio upload pipeline	Матеріал додається до системи, створюється відповідний запис і запускається обробка
Processing jobs	Формування payload для polling, перехід задачі у стан ready, обробка failed-сценаріїв	Задача має коректний статус, frontend може відстежувати її виконання
Картки та повторення	Генерація card set, ручне створення картки, перевірка відповіді, оновлення spaced repetition	Картки зберігаються, study_attempt фіксується, next_review_at оновлюється
Екзаменаційний модуль	Створення quiz set, генерація quiz_items, перевірка відповіді користувача	Питання доступні користувачу, відповідь оцінюється, feedback зберігається
Чат і колекції	Розділення assistant session за PDF, чат по колекції з кількома PDF paths	Асистент використовує правильний контекст і не змішує сесії різних документів
Обмеження та безпека	Quota exceeded, облік ліміту cards, перевірка user ownership	Backend блокує перевищення лімітів і доступ до чужих ресурсів
PDF і кирилиця	Формування PDF з українським текстом	Кирилиця відображається коректно, PDF придатний для перегляду

Також перевіряється логіка статусів processing_jobs. Для користувача важливо бачити, що система не зависла, а виконує задачу в одному зі станів queued, processing, ready або failed. Якщо задача завершується помилкою, backend повинен

зберегти error_code та error_detail, щоб frontend міг показати зрозуміле повідомлення.

4.3 Frontend-тестування

Frontend-тестування спрямоване на перевірку клієнтської логіки та безпечного відображення даних. У поточній реалізації frontend-тест розміщено у файлі review_fixes.test.js і виконується засобами node:test. Основний акцент зроблено на захисті від небезпечного вставлення HTML і коректній роботі контексту, що передається через URL. Клієнтська частина реалізована мовою JavaScript, яка використовується для побудови інтерактивної логіки веб-інтерфейсу [20].

Оскільки система відображає текст, що може надходити від користувача або генеруватися AI-моделлю, важливо уникати прямого неконтрольованого використання innerHTML. Тому frontend-тести перевіряють, що текстові дані обробляються без виконання потенційно небезпечного коду [16].

Основні перевірки frontend-рівня наведено в таблиці 4.3.

Таблиця 4.3 — Frontend-перевірки

Перевірка	Призначення	Очікуваний результат
Заборона небезпечного innerHTML	Зниження XSS-ризиків під час відображення AI-відповідей і введення користувача	Текст відображається як дані, а не виконується як код
URL-driven context	Визначення активного PDF, pack або collection за параметрами сторінки	Frontend надсилає запити до правильного ресурсу
Відображення станів	Показ ready, processing, failed або порожніх станів інтерфейсу	Користувач розуміє поточний стан обробки
Робота з формами	Передавання параметрів імпорту, генерації карток і екзамену	Форми формують коректний запит до backend

Крім того, перевіряється URL-driven context. Це потрібно для того, щоб сторінки viewer, cards і exam коректно визначали, з яким PDF, learning pack або collection працює користувач. Помилка в такому механізмі могла б призвести до відкриття неправильного контексту або некоректного запиту до API.

Frontend-тестування не замінює повноцінних end-to-end тестів у браузері, однак воно дозволяє зафіксувати важливі помилки клієнтської логіки. У майбутньому доцільно додати Playwright або аналогічний інструмент для автоматизованого проходження повного сценарію користувача.

4.4 Перевірка основних сценаріїв користувача

Окрім автоматизованих тестів, було перевірено основні сценарії роботи користувача.

Така перевірка потрібна, оскільки система складається з кількох взаємопов’язаних модулів, і помилка може проявитися не в окремій функції, а в переході між етапами роботи.

Наведені сценарії підтверджують, що система виконує основний цикл роботи з навчальним матеріалом. Особливо важливими є сценарії, які поєднують кілька підсистем: імпорт джерела, фонову обробку, генерацію PDF, відображення в бібліотеці та подальшу роботу з чатом або картками

Також під час перевірки враховувалася коректність реакції системи на типові дії користувача: завантаження матеріалу, очікування завершення обробки, відкриття створеного документа та перехід до додаткових навчальних інструментів. Окрему увагу було приділено збереженню зв’язку між створеним PDF-конспектом, картками, екзаменаційними питаннями та AI-чатом. Це дозволило переконатися, що користувач може працювати з матеріалом не як з окремим файлом, а як з повноцінним навчальним набором. У разі виникнення помилок система повинна повідомляти користувача про проблему та не

порушувати доступ до вже створених матеріалів. Такий підхід дав змогу оцінити не лише працездатність окремих модулів, а й цілісність користувацького сценарію загалом.

Основні сценарії користувача та очікувані результати наведено в таблиці 4.4.

Таблиця 4.4 — Перевірка основних сценаріїв користувача

№	Сценарій	Очікуваний результат	Стан
1	Реєстрація нового користувача	Створюється обліковий запис, пароль зберігається у хешованому вигляді	Виконано
2	Авторизація користувача	Користувач отримує JWT-токен і переходить до основного інтерфейсу	Виконано
3	Імпорт YouTube-посилання або файлу	Створюється processing job, користувач бачить стан обробки	Виконано
4	Формування PDF-конспекту	PDF створюється, зберігається і доступний у бібліотеці	Виконано
5	Робота з PDF viewer	Користувач відкриває конспект і бачить сформований матеріал	Виконано
6	AI-чат за PDF	Асистент відповідає з урахуванням тексту конкретного документа	Виконано
7	Генерація карток	Створюється набір карток, доступний для перегляду й повторення	Виконано
8	Генерація карток, екзаменів і перевірка обмежень	Створюються навчальні завдання, оновлюються спроби, backend блокує quota exceeded і доступ до чужих ресурсів	Виконано

4.5 Демонстрація роботи веб-системи

Демонстрація роботи системи показує повний шлях користувача в браузері. Спочатку користувач проходить авторизацію, потім імпортує навчальне джерело, очікує завершення обробки, відкриває PDF-конспект, працює з AI-чатом, картками або екзаменаційним модулем.

Основний сценарій роботи користувача наведено на рисунку 4.2.

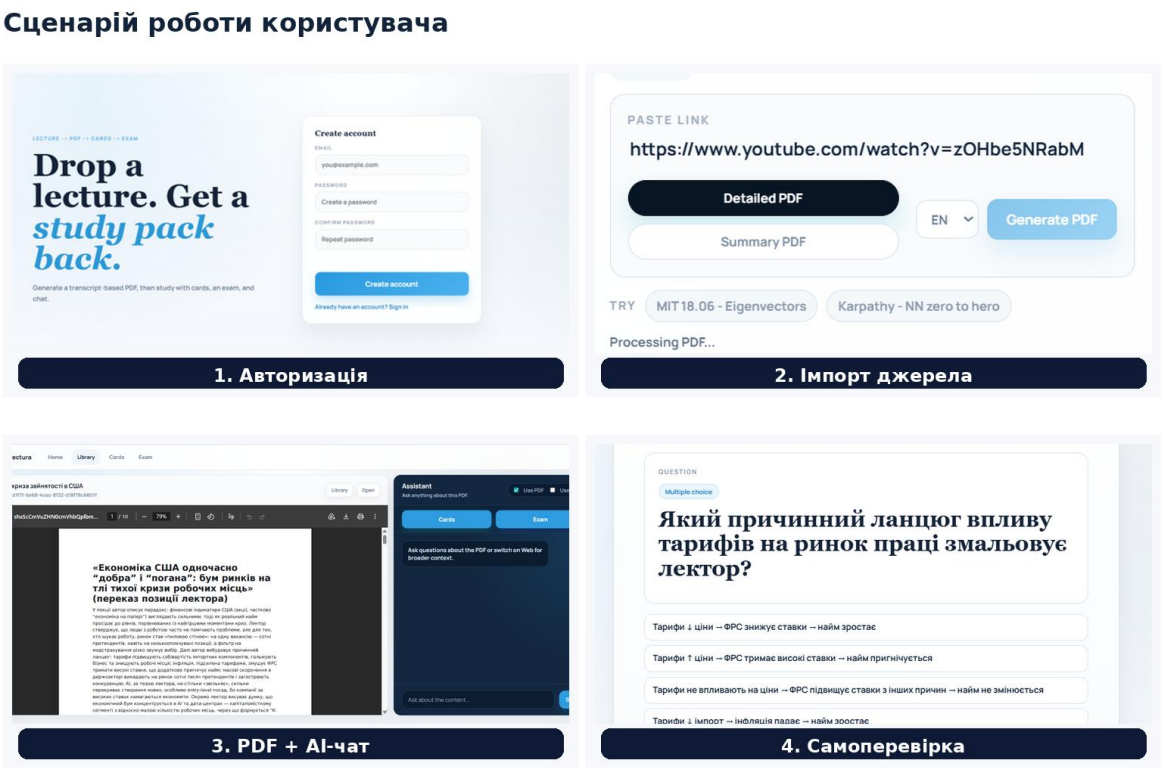

Рисунок 4.2 — Основний сценарій роботи користувача у веб-системі Lectura

Перший етап - авторизація або створення облікового запису. Після входу користувач отримує доступ до власних навчальних матеріалів. Другий етап - імпорт джерела, наприклад YouTube-посилання, аудіо-, відео- або PDF-документа. Третій етап - перегляд сформованого PDF-конспекту та робота з AI-чатом, який використовує текст документа як контекст. Четвертий етап - самоперевірка за допомогою екзаменаційного модуля або карток. Додаткові інтерфейсні модулі системи наведено на рисунку 4.3.

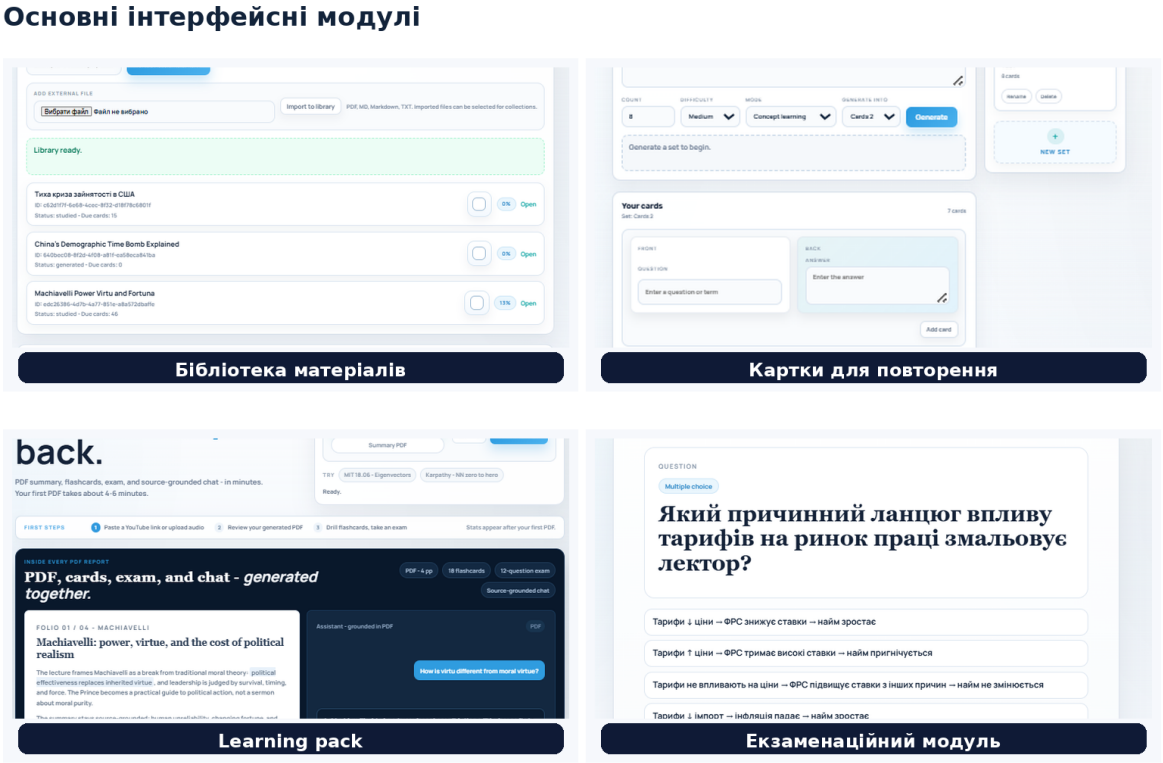

Рисунок 4.3 — Основні інтерфейсні модулі веб-системи Lectura

Бібліотека матеріалів дозволяє користувачеві повертатися до вже створених навчальних наборів, відкривати PDF-документи, переходити до карток або екзаменаційних завдань. Сторінка карток використовується для створення й повторення питань, а екзаменаційний модуль - для активної самоперевірки. Завдяки цьому система підтримує не лише пасивне читання конспекту, а й активне засвоєння матеріалу.

4.6 Оцінювання обмежень поточної версії

Поточна версія системи реалізує основні функції, необхідні для демонстрації працездатності веб-системи, однак має низку обмежень. Їх важливо враховувати, оскільки вони визначають напрями подальшого розвитку та межі застосування системи. Для AI-систем особливо важливо враховувати ризики помилкової

генерації, нестабільної поведінки та залежності результату від якості вхідних даних [21].

Основні обмеження поточної версії наведено в таблиці 4.5.

Таблиця 4.5 — Обмеження поточної версії системи

Обмеження	Вплив на систему	Можливе покращення
Відсутність повноцінного RAG-сховища	Для великих PDF у контекст AI-чату потрапляє лише обмежений фрагмент тексту	Додати chunking, embeddings і векторний пошук
Базова фонова обробка	Поточна модель processing_jobs придатна для демонстрації, але не є промисловою чергою задач	Використати Celery, RQ або інший worker-based підхід
Відсутність OCR	Скановані PDF без текстового шару можуть оброблятися некоректно	Додати OCR для зображень і сканів
Проста модель spaced repetition	Оцінка відповіді має лише два стани: correct або wrong	Розширити шкалу оцінювання та алгоритм інтервалів
Обмежена аналітика прогресу	Користувач не отримує повної статистики навчання	Додати дашборд успішності, складні теми, історію прогресу
Відсутність адміністративної панелі	Немає окремого інтерфейсу для перегляду користувачів, помилок і usage events	Реалізувати admin dashboard

Наведені обмеження не заважають демонстрації основної ідеї системи, але вони показують, які зміни потрібні для переходу від навчального прототипу до більш масштабованого продукту.

4.7 Перспективи розвитку системи

Подальший розвиток Lectura може відбуватися у кількох напрямах. Перший напрям - удосконалення алгоритмів роботи з документами. Для цього доцільно

реалізувати RAG-підхід: розбивати текст документа на фрагменти, формувати embeddings, зберігати їх у векторному сховищі та перед відповіддю AI-асистента вибирати найбільш релевантні фрагменти.

Другий напрям - покращення механізму фонової обробки. У поточній версії статуси queued, processing, ready і failed вже створюють основу для довготривалих задач, але для більшого навантаження варто використовувати окремі worker-процеси та чергу задач. Це дозволить паралельно обробляти кілька великих матеріалів і краще контролювати повторні спроби після помилок.

Третій напрям - розвиток навчальної аналітики. Система може показувати кількість пройдених карток, відсоток правильних відповідей, складні теми, історію екзаменаційних спроб і рекомендації щодо повторення. Такий функціонал зробить Lectura не лише генератором матеріалів, а й інструментом супроводу навчального процесу.

Четвертий напрям - підтримка нових джерел. Доцільно додати роботу з презентаціями, DOCX-документами, субтитрами, вебсторінками та сканованими матеріалами через OCR. Це розширить застосування системи для різних форматів навчального контенту.

П’ятий напрям - покращення користувацького досвіду та персоналізації навчання. У майбутньому система може надавати користувачеві гнучкі налаштування формату конспекту, складності карток, типів екзаменаційних питань і режимів повторення. Це дозволить адаптувати навчальні матеріали до різних стилів навчання та рівня підготовки користувача. Також доцільно додати збереження індивідуальних налаштувань, щоб система могла автоматично застосовувати їх під час наступних генерацій. Такий підхід зробить Lectura більш зручною для регулярного використання в навчальному процесі.

ВИСНОВКИ

У бакалаврській роботі було досліджено задачу автоматизованого створення навчальних матеріалів на основі лекційного відеоконтенту та розроблено веб-систему Lectura. Система орієнтована на студентів, викладачів і користувачів, які працюють з відеолекціями, PDF-документами, текстовими матеріалами та потребують швидкого формування конспектів, карток, екзаменаційних питань і контекстного чату.

У першому розділі було проаналізовано предметну область цифрового навчання, проблему ручної підготовки конспектів і самоперевірки, а також існуючі EdTech-рішення. Було встановлено, що окремі сервіси добре реалізують роботу з документами, картками або AI-чатом, однак не завжди забезпечують повний цикл перетворення лекційного джерела у структурований навчальний набір. На основі цього сформовано функціональні та нефункціональні вимоги до веб-системи Lectura.

У другому розділі було розроблено методи та алгоритми обробки навчального контенту. Описано загальну модель перетворення джерела в навчальний набір, алгоритм імпорту та нормалізації матеріалів, транскрибацію аудіо- та відеоконтенту, формування структурованого JSON-звіту, генерацію PDF-конспекту, створення карток і екзаменаційних питань, механізм spaced repetition та метод контекстного AI-чату за PDF-документом або колекцією.

У третьому розділі було спроєктовано та реалізовано клієнт-серверну архітектуру веб-системи. Backend-частину реалізовано на основі Python і FastAPI, для зберігання структурованих даних використано PostgreSQL, для роботи з файлами - Cloudflare R2, для транскрибації - Deepgram, а для генерації навчальних матеріалів - моделі OpenAI API. Frontend-частину реалізовано за допомогою HTML, CSS і JavaScript. Також описано структуру програмного проєкту, базу даних, REST API, авторизацію, фонову обробку, генерацію PDF-конспектів і контейнеризоване середовище запуску.

У четвертому розділі було проведено тестування та оцінювання роботи системи. Перевірено backend- і frontend-частини, основні сценарії користувача, генерацію PDF-документів, коректне відображення кирилиці, роботу карток, екзаменаційного модуля, AI-чату, колекцій документів, обмежень використання та контролю доступу до ресурсів. Результати перевірки показали, що система виконує поставлені функціональні вимоги та може використовуватися як основа для подальшого розвитку.

Практичне значення роботи полягає у створенні програмного продукту, який скорочує час підготовки до навчання, допомагає структурувати лекційний матеріал і підтримує активне засвоєння знань через картки, екзаменаційні питання та контекстний AI-чат. Подальший розвиток системи доцільно спрямувати на впровадження RAG/vector search, OCR для сканованих PDF, розширену аналітику прогресу, покращений алгоритм spaced repetition, адміністративну панель і підтримку нових типів навчальних джерел.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bates T. Teaching in a Digital Age: Guidelines for Designing Teaching and Learning. Vancouver: Tony Bates Associates Ltd., 2015. 517 p.

2. Mayer R. E. Multimedia Learning. 3rd ed. Cambridge: Cambridge University Press, 2020. 350 p.

3. Dunlosky J., Rawson K. A., Marsh E. J., Nathan M. J., Willingham D. T. Improving students' learning with effective learning techniques: promising directions from cognitive and educational psychology. Psychological Science in the Public Interest. 2013. Vol. 14, no. 1. P. 4–58. DOI: 10.1177/1529100612453266.

4. Brown P. C., Roediger H. L., McDaniel M. A. Make It Stick: The Science of Successful Learning. Cambridge, MA: Harvard University Press, 2014. 313 p.

5. Jurafsky D., Martin J. H. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. 3rd ed. Stanford: Stanford University, 2026.

6. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser L., Polosukhin I. Attention is all you need. Advances in Neural Information Processing Systems. 2017. Vol. 30.

7. Brown T. B., Mann B., Ryder N., Subbiah M., Kaplan J., Dhariwal P. et al. Language models are few-shot learners. Advances in Neural Information Processing Systems. 2020. Vol. 33. P. 1877–1901.

8. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge, MA: MIT Press, 2016. 800 p.

9. Lewis P., Perez E., Piktus A., Petroni F., Karpukhin V., Goyal N. et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems. 2020. Vol. 33. P. 9459–9474.

10. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston: Prentice Hall, 2017. 432 p.

11. Fowler M. Refactoring: Improving the Design of Existing Code. 2nd ed. Boston: Addison-Wesley Professional, 2018. 448 p.

12. Silberschatz A., Korth H. F., Sudarshan S. Database System Concepts. 7th ed. New York: McGraw-Hill, 2020. 1376 p.

13. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol: O'Reilly Media, 2017. 590 p.

14. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures: doctoral dissertation. Irvine: University of California, 2000.

15. Stallings W., Brown L. Computer Security: Principles and Practice. 4th ed. Boston: Pearson, 2018.

16. Sullivan B., Liu V. Web Application Security: A Beginner's Guide. New York: McGraw-Hill, 2012. 331 p.

17. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol: O'Reilly Media, 2021. 616 p.

18. Richardson C. Microservices Patterns: With Examples in Java. Shelter Island: Manning Publications, 2018. 520 p.

19. Matthes E. Python Crash Course: A Hands-On, Project-Based Introduction to Programming. 3rd ed. San Francisco: No Starch Press, 2023. 552 p.

20. Flanagan D. JavaScript: The Definitive Guide. 7th ed. Sebastopol: O'Reilly Media, 2020. 704 p.

21. Amodei D., Olah C., Steinhardt J., Christiano P., Schulman J., Mané D. Concrete Problems in AI Safety. arXiv preprint arXiv:1606.06565. 2016.

ДОДАТОК А

Фрагменти ключового програмного коду веб-системи

«Lectura — веб-система генерації конспектів і навчальних наборів на основі відеоконтенту»

УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ІАТЕ_ЦТЕ_ТР_21

Мови програмування — Python, JavaScript, HTML, CSS

Аркушів 19

Київ — 2026

Лістинг ключових фрагментів програмного коду веб-системи Lectura

Програмні засоби:

Backend — Python, FastAPI, SQLAlchemy, Alembic;

AI-модулі — OpenAI API, Deepgram;

Сховище файлів — Cloudflare R2;

Frontend — HTML, CSS, JavaScript;

Тестування — pytest, unittest та JavaScript-тести frontend-логіки.

==========

ЧАСТИНА 1. ДОМЕННА МОДЕЛЬ, АВТОРИЗАЦІЯ ТА БАЗОВІ РЕСУРСИ

==========

Файл: Back/app/core/models.py

```
class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    email = Column(String(255), unique=True, index=True, nullable=False)
    hashed_password = Column(String(255), nullable=False)
    created_at = Column(DateTime(timezone=True), server_default=func.now(), nullable=False)

class PDF(Base):
    __tablename__ = "pdfs"

    id = Column(String(36), primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"), index=True, nullable=False)
    title = Column(Text, nullable=True)
    r2_key = Column(String(512), nullable=False)
    created_at = Column(DateTime(timezone=True), server_default=func.now(), nullable=False)

class PDFCollection(Base):
    __tablename__ = "pdf_collections"

    id = Column(String(36), primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"), index=True, nullable=False)
    title = Column(Text, nullable=False)
    status = Column(String(32), nullable=False, default="ready")
    created_at = Column(DateTime(timezone=True), server_default=func.now(), nullable=False)
    last_activity_at = Column(DateTime(timezone=True), server_default=func.now(), nullable=False)

class PDFCollectionItem(Base):
```

```
    __tablename__ = "pdf_collection_items"

    id = Column(String(36), primary_key=True, index=True)
    collection_id = Column(String(36), ForeignKey("pdf_collections.id"), index=True, nullable=False)
    user_id = Column(Integer, ForeignKey("users.id"), index=True, nullable=False)
    pdf_id = Column(String(36), ForeignKey("pdfs.id"), index=True, nullable=False)
    position = Column(Integer, nullable=False, default=0)
    created_at = Column(DateTime(timezone=True), server_default=func.now(), nullable=False)


class LearningPack(Base):
    __tablename__ = "learning_packs"

    id = Column(String(36), primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"), index=True, nullable=False)
    pdf_id = Column(String(36), ForeignKey("pdfs.id"), index=True, nullable=True)
    collection_id = Column(String(36), ForeignKey("pdf_collections.id"), index=True, nullable=True)
    title = Column(Text, nullable=False)
    source_type = Column(String(32), nullable=False)
    status = Column(String(32), nullable=False, default="ready")
    created_at = Column(DateTime(timezone=True), server_default=func.now(), nullable=False)
    last_activity_at = Column(DateTime(timezone=True), server_default=func.now(), nullable=False)


class ProcessingJob(Base):
    __tablename__ = "processing_jobs"

    id = Column(String(36), primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"), index=True, nullable=False)
    job_type = Column(String(64), index=True, nullable=False)
    status = Column(String(32), index=True, nullable=False, default="queued")
    pdf_id = Column(String(36), ForeignKey("pdfs.id"), index=True, nullable=True)
    pack_id = Column(String(36), ForeignKey("learning_packs.id"), index=True, nullable=True)
    resource_type = Column(String(32), nullable=True)
    resource_id = Column(String(36), nullable=True)
    payload_json = Column(Text, nullable=True)
    result_json = Column(Text, nullable=True)
    error_code = Column(String(64), nullable=True)
    error_detail = Column(Text, nullable=True)
    created_at = Column(DateTime(timezone=True), server_default=func.now(), nullable=False)
    updated_at = Column(DateTime(timezone=True), server_default=func.now(), nullable=False)
    started_at = Column(DateTime(timezone=True), nullable=True)
    finished_at = Column(DateTime(timezone=True), nullable=True)


class Card(Base):
    __tablename__ = "cards"

    id = Column(String(36), primary_key=True, index=True)
    set_id = Column(String(36), index=True, nullable=False)
    user_id = Column(Integer, ForeignKey("users.id"), index=True, nullable=False)
    pdf_id = Column(String(36), ForeignKey("pdfs.id"), index=True, nullable=False)
```

```
    question = Column(Text, nullable=False)
    answer = Column(Text, nullable=False)
    context = Column(Text, nullable=False)
    next_review_at = Column(DateTime(timezone=True), nullable=True)
    ease = Column(Float, nullable=False, default=2.5)
    streak = Column(Integer, nullable=False, default=0)
    last_result = Column(String(16), nullable=True)
    last_reviewed_at = Column(DateTime(timezone=True), nullable=True)
    created_at = Column(DateTime(timezone=True), server_default=func.now(), nullable=False)


class QuizItem(Base):
    __tablename__ = "quiz_items"


    id = Column(String(36), primary_key=True, index=True)
    set_id = Column(String(36), index=True, nullable=False)
    user_id = Column(Integer, ForeignKey("users.id"), index=True, nullable=False)
    pdf_id = Column(String(36), ForeignKey("pdfs.id"), index=True, nullable=False)
    position = Column(Integer, nullable=False)
    quiz_type = Column(String(16), nullable=False)  # "choice" | "fill"
    question = Column(Text, nullable=False)
    options_json = Column(Text, nullable=True)  # JSON string with 4 options for choice type
    answer = Column(Text, nullable=False)
    context = Column(Text, nullable=False)
    created_at = Column(DateTime(timezone=True), server_default=func.now(), nullable=False)


class AssistantSession(Base):
    __tablename__ = "assistant_sessions"


    id = Column(String(36), primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"), index=True, nullable=False)
    pdf_id = Column(String(36), ForeignKey("pdfs.id"), index=True, nullable=True)
    session_key = Column(String(255), unique=True, index=True, nullable=False)
    created_at = Column(DateTime(timezone=True), server_default=func.now(), nullable=False)
# ... фрагмент скорочено для компактного додатку
```

Файл: Back/app/api/auth.py

```
import os
from datetime import datetime, timedelta
from typing import Optional


from fastapi import Depends, HTTPException
from fastapi.security import OAuth2PasswordBearer
from jose import JWTError, jwt
from passlib.context import CryptContext
from sqlalchemy.orm import Session


from app.storage.db import get_db
from app.core.models import User
```

```
pwd_context = CryptContext(schemes=["argon2"], deprecated="auto")
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="/api/auth/login")
MAX_PASSWORD_BYTES = 4096



def _secret_key() -> str:
    return os.getenv("JWT_SECRET", "change_me")



def _algorithm() -> str:
    return os.getenv("JWT_ALGORITHM", "HS256")



def _token_expire_minutes() -> int:
    return int(os.getenv("ACCESS_TOKEN_EXPIRE_MINUTES", "120"))



def verify_password(plain_password: str, hashed_password: str) -> bool:
    return pwd_context.verify(plain_password, hashed_password)



def get_password_hash(password: str) -> str:
    return pwd_context.hash(password)



def validate_password_length(password: str) -> None:
    if len(password.encode("utf-8")) > MAX_PASSWORD_BYTES:
# ... фрагмент скорочено для компактного додатку
```

Файл: Back/app/infra/server.py

```
def _api_error(status_code: int, detail: str, code: Optional[str] = None, hint: Optional[str] = None) ->
HTTPException:
    payload = {
        "detail": detail,
        "code": code or _error_code(status_code),
        "hint": hint or _error_hint(status_code),
    }
    return HTTPException(status_code=status_code, detail=payload)


def _create_job(db, user_id: int, job_type: str, payload: Optional[dict] = None) -> ProcessingJob:
    now = datetime.utcnow()
    row = ProcessingJob(
        id=str(uuid.uuid4()),
        user_id=user_id,
        job_type=job_type,
```

```
        status="queued",
        payload_json=json.dumps(payload or {}, ensure_ascii=False),
        updated_at=now,
    )
    db.add(row)
    db.commit()
    db.refresh(row)
    return row


def _mark_job_ready(db, job: ProcessingJob, result: dict, pdf_id: Optional[str] = None, pack_id:
Optional[str] = None) -> None:
    now = datetime.utcnow()
    job.status = "ready"
    job.pdf_id = pdf_id or result.get("pdf_id")
    job.pack_id = pack_id or result.get("pack_id")
    job.resource_type = result.get("resource_type", "learning_pack")
    job.resource_id = result.get("resource_id") or result.get("pack_id") or result.get("pdf_id")
    job.result_json = json.dumps(result, ensure_ascii=False)
    job.error_code = None
    job.error_detail = None
    job.finished_at = now
    job.updated_at = now
    db.commit()


def _mark_job_failed(db, job: ProcessingJob, exc: Exception) -> None:
    now = datetime.utcnow()
    job.status = "failed"
    job.error_code = exc.__class__.__name__
    job.error_detail = str(exc)
    job.finished_at = now
    job.updated_at = now
    db.commit()


def _ensure_pack_for_pdf(db, user_id: int, pdf_id: str, title: Optional[str], source_type: str =
"generated_pdf") -> LearningPack:
    row = (
        db.query(LearningPack)
        .filter(LearningPack.user_id == user_id, LearningPack.pdf_id == pdf_id,
LearningPack.collection_id.is_(None))
        .first()
    )
    now = datetime.utcnow()
    if row:
        if title and row.title != title:
            row.title = title
        row.status = "ready"
        row.last_activity_at = now
        db.flush()
        return row
```

```
    row = LearningPack(
        id=str(uuid.uuid4()),
        user_id=user_id,
        pdf_id=pdf_id,
# ... фрагмент скорочено для компактного додатку
```

==========

ЧАСТИНА 2. AI-ПАЙПЛАЙН, ЧАТ, ЧАНКУВАННЯ ТА НАВЧАЛЬНІ НАБОРИ

==========

Файл: Back/app/api/assistant.py

```
def _split_text_chunks(text: str, chunk_chars: int, overlap: int) -> list[dict]:
    cleaned = re.sub(r"\s+", " ", text or "").strip()
    if not cleaned:
        return []
    chunk_size = max(200, int(chunk_chars or 2500))
    overlap_size = max(0, min(int(overlap or 0), chunk_size // 2))
    chunks = []
    start = 0
    index = 1
    while start < len(cleaned):
        end = min(len(cleaned), start + chunk_size)
        if end < len(cleaned):
            boundary = cleaned.rfind(" ", start + chunk_size // 2, end)
            if boundary > start:
                end = boundary
        fragment = cleaned[start:end].strip()
        if fragment:
            chunks.append({"index": index, "text": fragment})
            index += 1
        if end >= len(cleaned):
            break
        start = max(0, end - overlap_size)
    return chunks


def _rank_chunks(chunks: list[dict], query: str) -> list[dict]:
    terms = _tokenize_query(query)
    if not terms:
        return chunks


    ranked = []
    for chunk in chunks:
        body = str(chunk.get("text") or "")
        lower = body.lower()
        score = 0
        for term in terms:
            score += lower.count(term)
        if score:
            ranked.append({**chunk, "score": score})
```

```
    if not ranked:
        return chunks
    ranked.sort(key=lambda item: (-int(item.get("score") or 0), int(item.get("index") or 0)))
    return ranked


def _build_chunked_context(
    text: str,
    query: str = "",
    source_label: str = "PDF",
    chunk_chars: Optional[int] = None,
    overlap: Optional[int] = None,
    max_chunks: Optional[int] = None,
    max_chars: Optional[int] = None,
) -> str:
    limit_chars = max_chars if max_chars is not None else
_env_int("PDF_CONTEXT_MAX_CHARS", 20000)
    if limit_chars <= 0:
        limit_chars = len(text or "")
    chunk_size = chunk_chars if chunk_chars is not None else
_env_int("PDF_CONTEXT_CHUNK_CHARS", 2500)
    overlap_size = overlap if overlap is not None else
_env_int("PDF_CONTEXT_CHUNK_OVERLAP", 350)
    chunk_limit = max_chunks if max_chunks is not None else
_env_int("PDF_CONTEXT_MAX_CHUNKS", 8)
    chunk_limit = max(1, int(chunk_limit or 1))


    chunks = _split_text_chunks(text, chunk_size, overlap_size)
    selected = _rank_chunks(chunks, query)[:chunk_limit]
    selected.sort(key=lambda item: int(item.get("index") or 0))


    parts = []
    used = 0
    for chunk in selected:
        header = f"[{source_label} | FRAGMENT {chunk['index']}]\n"
        body = str(chunk.get("text") or "").strip()
        remaining = limit_chars - used - len(header)
        if remaining <= 0:
            break
        if len(body) > remaining:
            body = body[:remaining].rstrip()
        part = f"{header}{body}"
        parts.append(part)
        used += len(part) + 2
        if used >= limit_chars:
            break
    return "\n\n".join(parts)[:limit_chars].strip()


def _extract_pdf_text(pdf_path: Path, query: str = "", source_label: Optional[str] = None) -> str:
    sidecar_text = _read_pdf_sidecar_text(pdf_path)
    if sidecar_text:
```

```
        return _build_chunked_context(sidecar_text, query=query, source_label=source_label or
pdf_path.name)


    scan_max_chars = _env_int("PDF_CONTEXT_SCAN_MAX_CHARS", 200000)
    reader = PdfReader(str(pdf_path))
    parts = []
    for page in reader.pages:
        text = page.extract_text() or ""
        if text:
            parts.append(text)
        if scan_max_chars > 0 and sum(len(p) for p in parts) >= scan_max_chars:
            break
    text = "\n".join(parts)
    if scan_max_chars > 0 and len(text) > scan_max_chars:
        text = text[:scan_max_chars]
    return _build_chunked_context(text, query=query, source_label=source_label or pdf_path.name)
# ... фрагмент скорочено для компактного додатку
```

Файл: Back/app/core/agents.py

```
def _assistant_agent(use_web: bool = False) -> Agent:
    tools = [WebSearchTool()] if use_web else []
    return Agent(
        name="Assistant",
        tools=tools,
        instructions=(
            "You are an AI learning assistant. Respond in Ukrainian by default, "
            "even when the document or user message is in another language. "
            "Use another language only when the user explicitly asks for it. "
            "Optimize for insight, not repetition. "
            "Assume the user has access to the PDF and does not need it read aloud. "
            "When asked about a section, table, or figure, explain what it means, "
            "why it matters, and how it connects to the main idea. "
            "Paraphrase rather than quote; avoid long citations. "
            "Make the chat feel light and conversational, not like a report. "
            "Default to very concise answers: 1-3 short sentences or up to 3 bullets. "
            "Only provide detailed breakdowns if explicitly requested. "
            "Use the provided PDF as the primary source of truth."
        ),
    )


def _report_agent(mode: str = "full") -> Agent:
    mode = (mode or "full").lower()
    summary_instructions=(
        "This is SUMMARY mode. The goal is NOT to explain the lecture in detail, "
        "but to capture its intellectual skeleton. "
        "Do NOT reconstruct the lecturer's full reasoning or narrative flow. "
        "Reduce the content to conclusions, claims, and high-level mechanisms only. "
        "Each section should answer: 'What is asserted?' not 'Why in detail?'. "
        "Limit sections to 3–6 total. "
        "Each section must contain at most 1 short paragraph (2–4 sentences). "
        "Prefer bullet-like paragraphs over flowing prose. "
```

```
        "Omit illustrative examples unless they are the ONLY way the claim is made. "
        "Avoid rhetorical emphasis, storytelling, and extended causal chains. "
        "Tables should replace text whenever possible. "
        "Glossary is optional and should include at most 3 terms."
    )
    return Agent(
        name="ReportBuilder",
        instructions=(
            "You generate a structured analytical report based on a lecture transcript. "
            "Write every user-facing JSON string in Ukrainian. "
            "Translate the lecture content into natural Ukrainian while preserving proper names, terms, "
            "tone, and framing of the transcript. "
            "Do NOT neutralize, sanitize, or soften ideological, moral, or critical statements. "
            "Remove filler, verbal tics, sponsor segments, and purely organizational remarks, "
            "but preserve rhetorical emphasis and repetition when it reinforces an argument. "
            "Add honesty markers throughout: regularly attribute claims to the lecturer/author, "
            "e.g., \"лектор стверджує\", \"у лекції зазначено\", \"автор припускає\", "
            "\"приклад лектора\", \"пряме формулювання ідеї\". "
            "Make it clear this is a retelling of the lecturer's views, not objective scientific fact. "
            "The report must be detailed, explanatory, and causal, not just descriptive. "
            "Explicitly explain why events, ideas, or structures lead to subsequent outcomes "
            "when this logic is present in the transcript. "
            "Use many sections rather than few: each major idea, mechanism, or phase of the lecture "
            "should have its own section. "
            "Avoid merging ideology, economics, technology, and consequences into a single section "
            "unless the transcript explicitly does so. "
            "Use concrete examples, quotations (paraphrased), and implications when present in the
transcript. "
            "Do not invent facts, numbers, or interpretations not grounded in the transcript. "
            "Tables should be included only when the transcript contains clear comparative or structural
information. "
            "Charts should be included only when the transcript implies quantitative relationships; "
            "do not fabricate data. Leave tables or charts empty if not applicable. "
            "Add a short glossary at the end with the most complex or non-obvious terms (2-6 items) "
            "if such terms appear; otherwise leave glossary empty. "
            + ((summary_instructions + " ") if mode == "summary" else "")
            + "Output ONLY valid JSON, no markdown or commentary. "
            + "Use this schema exactly: "
            "{"
            '"title": str, '
# ... фрагмент скорочено для компактного додатку
```

Файл: Back/app/infra/server.py

```
@app.post("/api/assistant/ask")
def assistant_ask_endpoint(
    payload: AssistantRequest,
    current_user: User = Depends(get_current_user),
    db=Depends(get_db),
) -> dict:
    try:
        collection_id = (getattr(payload, "collection_id", None) or "").strip()
        collection_pdf_paths = None
```

```
        pdf_path = None
        if payload.use_pdf:
            if collection_id:
                _get_collection_or_404(collection_id, current_user, db)
                collection_pdf_paths = [row.r2_key for row in _collection_pdf_rows(collection_id,
current_user, db)]
            else:
                pdf_id = payload.pdf_id
                if not pdf_id:
                    latest = (
                        db.query(PDF)
                        .filter(PDF.user_id == current_user.id)
                        .order_by(PDF.created_at.desc())
                        .first()
                    )
                    if not latest:
                        raise _api_error(404, "No PDFs for this user", code="pdf_missing")
                    pdf_id = latest.id


                pdf_row = db.query(PDF).filter(PDF.id == pdf_id, PDF.user_id == current_user.id).first()
                if not pdf_row:
                    raise _api_error(404, "PDF not found", code="pdf_not_found")
                pdf_path = pdf_row.r2_key


        session_id = _touch_assistant_session(
            db,
            current_user.id,
            payload.pdf_id if payload.use_pdf and not collection_id else None,
            collection_id if payload.use_pdf else None,
        )
        answer = ask_assistant(
            message=payload.message,
            session_id=session_id,
            pdf_path=pdf_path,
            pdf_paths=collection_pdf_paths,
            use_web=payload.use_web,
            db=db,
            user_id=current_user.id,
            pdf_id=payload.pdf_id if payload.use_pdf and not collection_id else None,
        )
        _record_usage_event(
            db,
            current_user.id,
            "assistant_ask",
            pdf_id=payload.pdf_id if payload.use_pdf and not collection_id else None,
            payload={"use_pdf": payload.use_pdf, "use_web": payload.use_web, "collection_id":
collection_id or None},
        )
        return {"answer": answer, "session_id": session_id}
    except ValueError as exc:
        raise _api_error(400, str(exc)) from exc
```

```
    except Exception as exc:  # pragma: no cover - defensive for runtime errors
        logger.exception("Assistant failed")
        raise _api_error(500, f"{exc.__class__.__name__}: {exc}") from exc


def _normalize_card_item(raw_item: dict, index: int) -> dict:
    if not isinstance(raw_item, dict):
        raise HTTPException(status_code=502, detail=f"Invalid card at index {index}")


    card_type = str(raw_item.get("type", "")).strip().lower()
    if card_type not in {"term_definition", "date_event"}:
        raise HTTPException(
            status_code=502,
            detail=f"Card {index} must include type term_definition or date_event",
        )


    left = str(raw_item.get("left", "")).strip()
    right = str(raw_item.get("right", "")).strip()
    context = str(raw_item.get("context", "")).strip()
    if not left or not right or not context:
        raise HTTPException(
            status_code=502,
            detail=f"Card {index} must include non-empty left, right, and context",
# ... фрагмент скорочено для компактного додатку
```

==========

ЧАСТИНА 3. ІМПОРТ ФАЙЛІВ, ТРАНСКРИБАЦІЯ ТА ФОРМУВАННЯ PDF

==========

Файл: Back/app/infra/server.py

```
def _validate_document_import(file: UploadFile) -> str:
    suffix = Path(file.filename or "").suffix.lower()
    if suffix not in _document_import_suffixes():
        raise _api_error(
            400,
            f"Unsupported document type: {suffix or 'unknown'}.",
            code="unsupported_document_type",
            hint="Upload PDF, MD, Markdown, or TXT.",
        )
    return suffix


@app.post("/api/import/document")
def import_document_endpoint(
    file: UploadFile = File(...),
    current_user: User = Depends(get_current_user),
    db=Depends(get_db),
):
    suffix = _validate_document_import(file)
    max_bytes = int(os.getenv("MAX_DOCUMENT_IMPORT_BYTES", str(25 * 1024 * 1024)))
    content = file.file.read(max_bytes + 1)
```

```
    if len(content) > max_bytes:
        raise _api_error(
            413,
            f"Document exceeds the {max_bytes} byte limit.",
            code="document_too_large",
            hint="Upload a smaller PDF, MD, Markdown, or TXT file.",
        )


    if not content:
        raise _api_error(400, "Document is empty.", code="empty_document")
    if not r2_configured():
        raise ValueError("R2 is not configured. Set R2_ENDPOINT, R2_BUCKET,
R2_ACCESS_KEY_ID, R2_SECRET_ACCESS_KEY.")
    _enforce_pack_quota(db, current_user.id)


    pdf_id = str(uuid.uuid4())
    title = _plain_title_from_filename(file.filename)
    output_dir = Path(os.getenv("REPORT_OUTPUT_DIR", "outputs"))
    output_dir.mkdir(parents=True, exist_ok=True)
    output_path = output_dir / f"{pdf_id}.pdf"


    sidecar_text = ""
    if suffix == ".pdf":
        output_path.write_bytes(content)
        sidecar_text = _extract_pdf_text_best_effort(output_path)
    else:
        try:
            text = content.decode("utf-8")
        except UnicodeDecodeError:
            text = content.decode("utf-8", errors="replace")
        sidecar_text = _strip_markdown_for_pdf(text)
        _text_document_to_pdf(sidecar_text, title, output_path)
    _write_pdf_sidecar_text(output_dir, pdf_id, sidecar_text)


    r2_key = f"{pdf_id}.pdf"
    upload_pdf(output_path, r2_key)
    row = PDF(id=pdf_id, user_id=current_user.id, r2_key=r2_key, title=title)
    db.add(row)
    db.flush()
    pack = _ensure_pack_for_pdf(db, current_user.id, pdf_id, title, source_type="imported_document")
    db.commit()
# ... фрагмент скорочено для компактного додатку
```

Файл: Back/app/core/transcribe.py

```
def _download_video_with_ytdlp(url: str, workdir: Path) -> Path:
    if YoutubeDL is None:
        raise ValueError("YOUT_API_KEY is not set and yt-dlp is not installed.")


    start = time.monotonic()
```

```
    output_template = str(workdir / "source.%(ext)s")
    ydl_opts = {
        "format": os.getenv("YTDLP_FORMAT", "bestaudio/best"),
        "outtmpl": output_template,
        "noplaylist": True,
        "quiet": True,
        "no_warnings": True,
    }
    with YoutubeDL(ydl_opts) as ydl:
        info = ydl.extract_info(url, download=True)
        filename = ydl.prepare_filename(info)


    output_path = Path(filename)
    if not output_path.exists() or output_path.stat().st_size == 0:
        raise ValueError("yt-dlp returned an empty media file.")


    logger.info(
        "download_video_ytdlp_ms=%d size_bytes=%d",
        int((time.monotonic() - start) * 1000),
        output_path.stat().st_size,
    )
    return output_path


def _convert_to_wav(input_path: Path, output_path: Path) -> float:
    start = time.monotonic()
    ffmpeg_path = _resolve_ffmpeg()
    custom_filter = os.getenv("AUDIO_FILTER", "").strip()
    atempo_filter = ""
    default_filter = ""
    speed_factor = 1.0


    if not custom_filter:
        highpass_raw = os.getenv("AUDIO_HIGHPASS", "200").strip()
        lowpass_raw = os.getenv("AUDIO_LOWPASS", "3000").strip()
        speed_raw = os.getenv("AUDIO_SPEED", "1.0").strip()
        try:
            speed = float(speed_raw)
        except ValueError as exc:
            raise RuntimeError(f"AUDIO_SPEED must be a number, got: {speed_raw!r}") from exc


        speed_factor = speed
        atempo_filter = _build_atempo_filter(speed)
        parts = [f"highpass=f={highpass_raw}", f"lowpass=f={lowpass_raw}"]
        if atempo_filter:
            parts.append(atempo_filter)
        parts.append("aresample=16000")
        default_filter = ",".join(parts)
    else:
        speed_factor = _parse_atempo_speed(custom_filter)
```

```
    command = [
        ffmpeg_path,
        "-y",
        "-i",
        str(input_path),
        "-ac",
        "1",
        "-vn",
    ]

# ... фрагмент скорочено для компактного додатку
```

Файл: Back/app/reporting/report.py

```
def _build_pdf_reportlab(report: Dict[str, Any], output_path: Path) -> Path:
    from reportlab.lib import colors
    from reportlab.lib.pagesizes import A4
    from reportlab.lib.styles import getSampleStyleSheet
    from reportlab.pdfbase import pdfmetrics
    from reportlab.pdfbase.ttfonts import TTFont
    from reportlab.platypus import (
        Image,
        ListFlowable,
        ListItem,
        Paragraph,
        SimpleDocTemplate,
        Spacer,
        Table,
        TableStyle,
    )
    from reportlab.lib.utils import ImageReader


    def _register_unicode_fonts() -> Tuple[str, str]:
        normal_path = _first_existing_path(_font_candidates())
        bold_path = _first_existing_path(_bold_font_candidates()) or normal_path
        if not normal_path:
            logger.warning("unicode_report_font_not_found fallback=Helvetica")
            return "Helvetica", "Helvetica-Bold"


        try:
            pdfmetrics.getFont("ReportFont")
        except KeyError:
            pdfmetrics.registerFont(TTFont("ReportFont", str(normal_path)))
        try:
            pdfmetrics.getFont("ReportFont-Bold")
        except KeyError:
            pdfmetrics.registerFont(TTFont("ReportFont-Bold", str(bold_path)))
        pdfmetrics.registerFontFamily(
            "ReportFont",
            normal="ReportFont",
            bold="ReportFont-Bold",
```

```
            italic="ReportFont",
            boldItalic="ReportFont-Bold",
        )
        return "ReportFont", "ReportFont-Bold"


    body_font, bold_font = _register_unicode_fonts()
    styles = getSampleStyleSheet()
    for style_name in ("BodyText", "Bullet"):
        if style_name in styles:
            styles[style_name].fontName = body_font
    for style_name in ("Title", "Heading1", "Heading2", "Heading3"):
        if style_name in styles:
            styles[style_name].fontName = bold_font
    doc = SimpleDocTemplate(str(output_path), pagesize=A4)
    max_width = doc.width
    story: List[Any] = []


    title = report.get("title", "Report")
# ... фрагмент скорочено для компактного додатку
```

==========

ЧАСТИНА 4. FRONTEND: СТАН, БЕЗПЕКА РЕНДЕРИНГУ, КАРТКИ ТА ІСПИТ

==========

Файл: Front/auth.js

```
(function () {
  const TOKEN_LIFETIME_MS = 48 * 60 * 60 * 1000;
  const AUTH_ERROR_CODE = "AUTH_REDIRECT";
  const LOGIN_PAGE = "index.html";


  const decodeJwtPayload = (token) => {
    try {
      const parts = token.split(".");
      if (parts.length < 2) return null;
      const base64Url = parts[1].replace(/-/g, "+").replace(/_/g, "/");
      const padded = base64Url + "=".repeat((4 - (base64Url.length % 4)) % 4);
      return JSON.parse(atob(padded));
    } catch (error) {
      return null;
    }
  };


  const clearAuth = () => {
    localStorage.removeItem("access_token");
    localStorage.removeItem("token_type");
    localStorage.removeItem("token_saved_at");
  };


  const redirectToLogin = () => {
```

```
    clearAuth();
    if (!window.location.pathname.endsWith(`/${LOGIN_PAGE}`) &&
!window.location.pathname.endsWith(LOGIN_PAGE)) {
      window.location.href = LOGIN_PAGE;
    }
  };

  const isTokenExpired = (token) => {
    if (!token) return true;

    const payload = decodeJwtPayload(token);
    if (payload && typeof payload.exp === "number") {
      return Date.now() >= payload.exp * 1000;
    }

    const savedAt = Number(localStorage.getItem("token_saved_at") || 0);
    if (!Number.isFinite(savedAt) || savedAt <= 0) {
# ... фрагмент скорочено для компактного додатку
```

Файл: Front/app-state.js

```
(function (root, factory) {
  const api = factory();
  if (typeof module !== "undefined" && module.exports) {
    module.exports = api;
  }
  root.AppState = api;
})(typeof window !== "undefined" ? window : globalThis, function () {
  const resolveSearch = (search) => {
    if (typeof search === "string") return search;
    if (typeof window !== "undefined" && window.location) return window.location.search || "";
    return "";
  };

  const getParams = (search) => new URLSearchParams(resolveSearch(search));

  const buildPageUrl = (path, params = {}) => {
    const query = new URLSearchParams();
    Object.entries(params).forEach(([key, value]) => {
      if (value !== undefined && value !== null && value !== "") {
        query.set(key, String(value));
      }
    });
    const encoded = query.toString();
    return encoded ? `${path}?${encoded}` : path;
  };

  const resolvePdfContext = (search, fallback = {}) => {
    const params = getParams(search);
    const pdfId = params.get("pdf_id") || fallback.pdfId || "";
```

```
  const pdfTitle = params.get("pdf_title") || fallback.pdfTitle || "";
  const packId = params.get("pack_id") || fallback.packId || "";
  return { pdfId, pdfTitle, packId };
 };

 const readPdfContext = () => {
  const fallback = {
   pdfId: sessionStorage.getItem("pdfId") || "",
   pdfTitle: sessionStorage.getItem("pdfTitle") || "",
   packId: sessionStorage.getItem("packId") || "",
  };
  return resolvePdfContext(undefined, fallback);
 };

 const writePdfContext = ({ pdfId = "", pdfTitle = "", packId = "", dataUrl = "" }) => {
  if (pdfId) sessionStorage.setItem("pdfId", pdfId);
# ... фрагмент скорочено для компактного додатку
```

Файл: Front/ui.js

```
(function (root, factory) {
 const api = factory();
 if (typeof module !== "undefined" && module.exports) {
  module.exports = api;
 }
 root.UI = api;
})(typeof window !== "undefined" ? window : globalThis, function () {
 const clearNode = (node) => {
  if (!node) return;
  while (node.firstChild) node.removeChild(node.firstChild);
 };

 const el = (tag, options = {}) => {
  const node = document.createElement(tag);
  if (options.className) node.className = options.className;
  if (options.text !== undefined) node.textContent = String(options.text);
  if (options.type) node.type = options.type;
  if (options.attrs) {
   Object.entries(options.attrs).forEach(([key, value]) => {
    if (value !== undefined && value !== null) {
     node.setAttribute(key, String(value));
    }
   });
  }
  if (options.dataset) {
   Object.entries(options.dataset).forEach(([key, value]) => {
    if (value !== undefined && value !== null) {
     node.dataset[key] = String(value);
    }
   });
  }
```

```
    return node;
  };

  const setStatus = (node, message, tone = "info") => {
# ... фрагмент скорочено для компактного додатку
```

Файл: Front/viewer.html

```
    const withPdfViewOptions = (url) => {
      if (!url) return "";
      const separator = url.includes("#") ? "&" : "#";
      return `${url}${separator}zoom=72&navpanes=0`;
    };

    const appendBubble = (text, role) => {
      const bubble = document.createElement("div");
      bubble.className = `chat-bubble ${role || ""}`.trim();
      bubble.textContent = text;
      messages.appendChild(bubble);
      messages.scrollTop = messages.scrollHeight;
      return bubble;
    };

    const appendTyping = () => {
      const bubble = document.createElement("div");
      bubble.className = "chat-bubble assistant typing";
      bubble.appendChild(window.UI.el("span", { className: "typing-dot" }));
      bubble.appendChild(window.UI.el("span", { className: "typing-dot" }));
      bubble.appendChild(window.UI.el("span", { className: "typing-dot" }));
      messages.appendChild(bubble);
      messages.scrollTop = messages.scrollHeight;
      return bubble;
    };

    const openPdfFromCollection = (pdf) => {
      const targetId = String(pdf?.id || pdf?.pdf_id || "").trim();
      if (!targetId) return;
      const targetTitle = String(pdf?.title || targetId).trim();
      window.AppState.clearPdfDataUrl?.();
      window.AppState.clearCollectionContext?.();
      window.AppState.writePdfContext({ pdfId: targetId, pdfTitle: targetTitle });
      window.location.href = window.AppState.buildPageUrl("viewer.html", {
        pdf_id: targetId,
        pdf_title: targetTitle,
      });
    };

    const setSending = (sending) => {
# ... фрагмент скорочено для компактного додатку
```

Файл: Front/flashcards.html

```
    const buildCardNode = (card, index) => {
      const item = window.UI.el("div", { className: "flashcard" });
      const front = window.UI.el("div", { className: "flashcard-front" });
      front.appendChild(window.UI.el("span", { className: "flashcard-label", text: "Question" }));
      front.appendChild(window.UI.el("p", { text: card.question }));

      const back = window.UI.el("div", { className: "flashcard-back" });
      back.appendChild(window.UI.el("span", { className: "flashcard-label", text: "Answer" }));
      back.appendChild(window.UI.el("p", { text: card.answer }));

      const actions = window.UI.el("div", { className: "flashcard-actions" });
      actions.appendChild(window.UI.el("button", { className: "flashcard-toggle", text: "Show
answer", type: "button" }));
      actions.appendChild(
        window.UI.el("button", {
          className: "flashcard-remove",
          text: "Delete",
          type: "button",
          attrs: { "aria-label": "Remove card" },
          dataset: { index },
        }),
      );

      item.appendChild(front);
      item.appendChild(back);
      item.appendChild(actions);
      return item;
    };

    const renderCards = () => {
      if (!grid) return;
      const active = getActiveSet();
      const cards = active ? (cardsBySet[active.id] || []) : [];
      window.UI.clearNode(grid);

      if (!cards.length) {
        if (empty) empty.style.display = "block";
        updateCounts();
        return;
      }

      if (empty) empty.style.display = "none";
      cards.forEach((card, index) => {
        grid.appendChild(buildCardNode(card, index));
      });
      updateCounts();
# ... фрагмент скорочено для компактного додатку
```

Файл: Front/cards-study.html

```
    const shuffleItems = (items) => {
      const shuffled = items.slice();
      for (let i = shuffled.length - 1; i > 0; i -= 1) {
        const j = Math.floor(Math.random() * (i + 1));
        [shuffled[i], shuffled[j]] = [shuffled[j], shuffled[i]];
      }
      return shuffled;
    };


    const resetState = () => {
      answered = false;
      card.classList.remove("is-correct", "is-wrong", "is-revealed", "is-blurred");
      if (correctEl) correctEl.textContent = "";
      if (answerInput) answerInput.value = "";
      if (nextButton) nextButton.textContent = "Check";
      if (showButton) {
        showButton.textContent = "Show answer";
        showButton.setAttribute("aria-pressed", "false");
      }
      startedAt = performance.now();
    };


    const updateProgress = () => {
      const total = cards.length;
      const answeredCount = Math.min(answeredCardIds.size, total);
      const correctCount = Math.min(correctCardIds.size, total);
      if (progressEl) progressEl.setAttribute("aria-label", `${answeredCount} answered from ${total},
${correctCount} correct`);
      if (progressCountEl) progressCountEl.textContent = total ? `${answeredCount} / ${total}
answered` : "0 / 0 answered";
      if (progressScoreEl) progressScoreEl.textContent = `${correctCount} correct`;
      if (progressBarEl) progressBarEl.style.width = total ? `${Math.round((answeredCount / total) *
100)}%` : "0%";
      if (setNameEl) setNameEl.textContent = activeSetName ? `Set: ${activeSetName}` : "Set: -";
    };


    const markProgress = (isCorrect) => {
      const current = cards[index];
      if (!current) return;
      const key = current.id || `card-${index}`;
      answeredCardIds.add(key);
      if (isCorrect) correctCardIds.add(key);
      else correctCardIds.delete(key);
      updateProgress();
    };


    const recordAttempt = (isCorrect, userAnswer, feedback) => {
      const current = cards[index];
# ... фрагмент скорочено для компактного додатку
```

Файл: Front/study.html

```
    const shuffleItems = (items) => {
      const shuffled = items.slice();
      for (let i = shuffled.length - 1; i > 0; i -= 1) {
        const j = Math.floor(Math.random() * (i + 1));
        [shuffled[i], shuffled[j]] = [shuffled[j], shuffled[i]];
      }
      return shuffled;
    };


    const normalizeType = (value) => {
      const type = String(value || "").trim().toLowerCase();
      return type === "fill" || type === "fill_blank" ? "fill" : "choice";
    };


    const updateProgress = () => {
      const total = items.length;
      const answeredCount = Math.min(answeredQuestionIds.size, total);
      const correctCount = Math.min(correctQuestionIds.size, total);
      if (progressEl) progressEl.setAttribute("aria-label", `${answeredCount} answered from ${total},
${correctCount} correct`);
      if (progressCountEl) progressCountEl.textContent = total ? `${answeredCount} / ${total}
answered` : "0 / 0 answered";
      if (progressScoreEl) progressScoreEl.textContent = `${correctCount} correct`;
      if (progressBarEl) progressBarEl.style.width = total ? `${Math.round((answeredCount / total) *
100)}%` : "0%";
      if (setNameEl) setNameEl.textContent = activeSetName ? `Exam: ${activeSetName}` : "Exam: -
";
    };


    const markProgress = (isCorrect) => {
      const current = items[index];
      if (!current) return;
      const key = current.id || `question-${index}`;
      answeredQuestionIds.add(key);
      if (isCorrect) correctQuestionIds.add(key);
      else correctQuestionIds.delete(key);
      updateProgress();
    };


    const recordAttempt = (isCorrect, userAnswer, feedback) => {
      const current = items[index];
      if (!current) return;
      const key = current.id || `question-${index}`;
      attemptsById.set(key, {
        question: current.question,
        type: current.type,
        userAnswer: userAnswer || "(blank)",
        correctAnswer: current.answer || "not available",
```

```
        feedback: feedback || "",
# ... фрагмент скорочено для компактного додатку
```

==========

ЧАСТИНА 5. ТЕСТИ ТА КОНФІГУРАЦІЯ

==========

Файл: Back/docker-compose.yml

```
services:
  backend:
    build:
      context: ..
      dockerfile: Back/Dockerfile
    ports:
      - "8000:8000"
    env_file:
      - .env
    volumes:
      - ./outputs:/app/outputs
      - ./data:/app/data
      - ${FRONTEND_HOST_DIR:-../Front}:/app/frontend:ro
    environment:
      REPORT_OUTPUT_DIR: /app/outputs
      PDF_CACHE_DIR: /app/data/pdf_cache
      FRONTEND_DIR: /app/frontend
    extra_hosts:
      - "host.docker.internal:host-gateway"
    depends_on:
      postgres:
        condition: service_healthy


  postgres:
    image: postgres:18
# ... фрагмент скорочено для компактного додатку
```

Файл: Front/tests/review_fixes.test.js

```
const assert = require("node:assert/strict");
const fs = require("node:fs");
const path = require("node:path");
const test = require("node:test");


const root = path.resolve(__dirname, "..");


test("dashboard, flashcards, and quizzes avoid string-based innerHTML rendering", () => {
  const targets = ["dashboard.html", "flashcards.html", "quizzes.html"];


  for (const filename of targets) {
    const source = fs.readFileSync(path.join(root, filename), "utf8");
    assert.equal(
```

```
      source.includes("innerHTML ="),
      false,
      `${filename} still uses innerHTML assignment`,
    );
  }
});

test("shared app state helper exists and restores pdf context from URL", async () => {
  const stateModulePath = path.join(root, "app-state.js");
  assert.equal(fs.existsSync(stateModulePath), true, "app-state.js is missing");

  const state = require(stateModulePath);
  const current = state.resolvePdfContext(
    "?pdf_id=pdf-123&pdf_title=Linear%20Algebra",
    {
      pdfId: "",
      pdfTitle: "",
# ... фрагмент скорочено для компактного додатку
```

Файл: Back/tests/test_product_hardening.py

```
import io
import os
import sys
import tempfile
import unittest
from pathlib import Path
from types import SimpleNamespace
from urllib.parse import unquote
from unittest.mock import patch


from fastapi import UploadFile
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker


PROJECT_ROOT = Path(__file__).resolve().parents[1]
if str(PROJECT_ROOT) not in sys.path:
    sys.path.insert(0, str(PROJECT_ROOT))


from app.api.schemas import RegisterRequest
from app.core.models import AssistantSession, Base, Card, CardSet, ChatMessage, LearningPack,
PDF, PDFCollection, PDFCollectionItem, ProcessingJob, QuizItem, QuizSet, StudyAttempt, User
from app.core import transcribe
from app.infra import server



class ProductHardeningTests(unittest.TestCase):
    def setUp(self) -> None:
        self.tempdir = tempfile.TemporaryDirectory()
```

```
        self.db_path = Path(self.tempdir.name) / "test.sqlite3"
        self.engine = create_engine(f"sqlite:///{self.db_path}")
        self.SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=self.engine)
        Base.metadata.create_all(bind=self.engine)
        self.db = self.SessionLocal()


    def tearDown(self) -> None:
        self.db.close()
# ... фрагмент скорочено для компактного додатку
```