НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ **Ігор БОЛБОТ**	_____ **Белла ГОЛУБ**
(підпис)	(підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення створення презентації з використання технологій штучного інтелекту»

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми к.т.н., доцент	_____ (підпис)	**Ганна ВАЙГАНГ**
Керівник бакалаврської кваліфікаційної роботи	_____ (підпис)	**Максим НЕДЬОШЕВ**
Консультант бакалаврської кваліфікаційної роботи д.е.н., професор	_____ (підпис)	**Роман РУДЕНСЬКИЙ**
Виконав	_____ (підпис)	**Денис КОТЕЛЬНИЦЬКИЙ**

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2025 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ

_____Котельницькому Денису Сергійовичу_____
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи

«**Програмне забезпечення створення презентації з використання технологій штучного інтелекту**»

затверджена наказом від «19» грудня 2025 р. № 3196 «С»
Термін подання завершеної роботи на кафедру «22» травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту):
документації LLM-провайдерів, вимоги до створення веб-застосунків.
Перелік питань, що підлягають дослідженню:
аналіз предметної області, аналіз існуючих рішень, проектування архітектури та бази даних системи, розробка клієнтської та серверної частини веб-застосунку, реалізація AI-генерації слайдів і PDF-експорту, тестування та розгортання системи
Перелік графічних документів (за необхідності).

Дата видачі завдання «19» грудня 2026 р.

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **Максим НЕДЬОШЕВ**

Консультант бакалаврської кваліфікаційної роботи _____ (підпис) **Роман РУДЕНСЬКИЙ**
д.е.н., професор

Завдання взяв до виконання _____ (підпис) **Денис КОТЕЛЬНИЦЬКИЙ**

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

- AI (Artificial Intelligence) - штучний інтелект
- API (Application Programming Interface) - інтерфейс програмування застосунків
- CRUD (Create, Read, Update, Delete) - базові операції з даними: створення, читання, оновлення, видалення
- CSS (Cascading Style Sheets) - каскадні таблиці стилів
- HMR (Hot Module Replacement) - гаряча заміна модулів без перезавантаження сторінки
- HTTPS (HyperText Transfer Protocol Secure) - захищений протокол передачі гіпертексту
- IaaS (Infrastructure as a Service) - інфраструктура як послуга
- JSON (JavaScript Object Notation) - текстовий формат обміну даними
- JWT (JSON Web Token) - стандарт токенів для автентифікації
- LLM (Large Language Model) - велика мовна модель
- PDF (Portable Document Format) - переносний формат документів
- PCI DSS (Payment Card Industry Data Security Standard) - стандарт безпеки даних платіжних карток
- REST (Representational State Transfer) - архітектурний стиль взаємодії між компонентами розподіленої системи
- SaaS (Software as a Service) - програмне забезпечення як послуга
- SPA (Single Page Application) - односторінковий застосунок
- SQL (Structured Query Language) - мова структурованих запитів до баз даних
- SQLite - вбудована реляційна система управління базами даних
- SSE (Server-Sent Events) - технологія передачі подій від сервера до клієнта
- СУБД - система управління базами даних
- TLS (Transport Layer Security) - протокол захисту транспортного рівня
- UML (Unified Modeling Language) - уніфікована мова моделювання

ВСТУП

Підготовка якісних презентацій залишається трудомістким та часозатратним процесом. Більшість існуючих інструментів або вимагають значних ручних зусиль для формування вмісту та дизайну, або залежать від хмарних сервісів із передачею потенційно конфіденційних даних на сторонні сервери. Стрімкий розвиток великих мовних моделей (Large Language Models, LLM) та їх інтеграція у прикладне програмне забезпечення відкриває принципово нові можливості для автоматизації інтелектуальних задач, зокрема автоматичної генерації структурованого мультимедійного контенту. Побудова веборієнтованої системи, здатної автоматично створювати повноцінні презентації на основі природномовного опису теми, є актуальним завданням сучасної прикладної інформатики та інженерії програмного забезпечення.

Об'єкт дослідження - процес автоматизованої генерації структурованих мультимедійних презентацій на основі природномовного опису теми.

Предмет дослідження - методи та архітектурні рішення для побудови веборієнтованої системи генерації слайдів з використанням великих мовних моделей та клієнт-серверної архітектури.

Мета роботи - проектування та реалізація системи Slide AI, веб-застосунку для автоматичної генерації презентацій за введеним користувачем промптом, з повноцінним редактором слайдів, системою збереження та завантаження презентацій, підтримкою кількох LLM-провайдерів та експортом у формат PDF.

- Для досягнення поставленої мети визначено такі завдання дослідження:
- проаналізувати існуючі засоби автоматизованого створення презентацій та визначити їх переваги й недоліки;
- спроектувати архітектуру клієнт-серверної системи з підтримкою кількох LLM-провайдерів (OpenRouter, Gemini, LM Studio);
- розробити базу даних та серверну частину системи;
- реалізувати веб-інтерфейс із редактором слайдів та системою автентифікації користувачів;

- реалізувати функціонал збереження, завантаження та вкладень (PDF і зображення) для генерації;
- реалізувати експорт готових презентацій у формат PDF;
- провести тестування функціональних можливостей системи.

У процесі розробки використано такі методи та технології: об'єктно-орієнтоване проектування із застосуванням мови моделювання UML; клієнт-серверна архітектура на основі React та Express.js; управління станом застосунку за допомогою Zustand; зберігання даних у реляційній базі даних SQLite; автентифікація користувачів на основі JWT-токенів; отримання зображень через Pexels API; рендеринг та експорт PDF за допомогою Puppeteer.

Апробація результатів роботи. Основні положення та результати дослідження опубліковано у тезах: Котельницький Д.С. «Програмне забезпечення створення презентації з використання технологій штучного інтелекту». Тези подано до збірника наукових праць факультету інформаційних технологій Національного університету біоресурсів і природокористування України.

Структура та обсяг роботи. Пояснювальна записка складається зі вступу, чотирьох розділів основної частини, висновків, списку використаних джерел та додатків. Загальний обсяг - __ сторінок основного тексту. Список використаних джерел містить 32 найменувань.

У першому розділі виконано аналіз предметної області: розглянуто існуючі аналоги системи генерації презентацій, визначено їх переваги та недоліки, сформульовано постановку задачі та побудовано моделі предметної області у нотації UML.

У другому розділі описано проектування інформаційного забезпечення: побудовано логічну модель даних, обґрунтовано вибір СУБД, наведено фізичну структуру бази даних.

У третьому розділі описано розробку програмного забезпечення: архітектуру системи, компонентну структуру, вибір інструментарію та реалізацію ключових модулів.

У четвертому розділі наведено відомості щодо тестування системи, вимоги до апаратного та програмного забезпечення, а також порядок розгортання та інсталяції.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Презентація є одним із найпоширеніших форматів подання інформації у сучасному світі. Вона використовується в освіті, бізнесі, науці та повсякденній комунікації для структурованого донесення ідей, даних та концепцій до аудиторії. Типова презентація складається з послідовності слайдів, кожен з яких поєднує текстовий, візуальний та мультимедійний контент для забезпечення наочності та переконливості [1].

Процес створення якісної презентації є трудомістким та потребує значних витрат часу. Автор має самостійно визначити структуру матеріалу, дібрати та систематизувати інформацію, сформулювати ключові тези для кожного слайду, підібрати відповідні ілюстрації та забезпечити візуальну узгодженість оформлення. За різними оцінками, підготовка однієї презентації з 10-15 слайдів займає від 2 до 8 годин залежно від складності теми та досвіду автора [2].

Предметна область системи охоплює процес автоматизованої генерації структурованих мультимедійних презентацій. Об'єктами предметної області є: тема презентації (формулюється користувачем у вільній текстовій формі), план - структура розбивки теми на логічно пов'язані слайди, слайд - базова одиниця презентації з заголовком та мультимедійним вмістом, а також презентація як впорядкована колекція слайдів із єдиним оформленням.

Автоматизація процесу створення презентацій стала можливою завдяки стрімкому розвитку технологій обробки природної мови та великих мовних моделей (Large Language Models, LLM). Сучасні LLM здатні аналізувати текстовий запит користувача, генерувати структурований план матеріалу, формулювати змістовні тези для кожного розділу та визначати оптимальний тип візуалізації даних. Це відкриває можливість побудови систем, що скорочують час підготовки презентації з годин до хвилин, зберігаючи при цьому змістовну якість матеріалу [3].

Важливою складовою автоматизованої генерації є підбір релевантних зображень. Візуальний контент суттєво підвищує сприйняття інформації аудиторією та робить презентацію більш переконливою. Сучасні API фотостоків дозволяють автоматично добирати тематичні зображення на основі ключових слів, що позбавляє автора необхідності самостійно шукати ілюстративний матеріал [4].

Окремим аспектом предметної області є забезпечення конфіденційності даних. Багато існуючих хмарних сервісів генерації презентацій передають вміст запитів на зовнішні сервери, що може бути неприйнятним для корпоративних або чутливих матеріалів. Альтернативою є використання локальних або захищених LLM-провайдерів, що дозволяє зберегти дані в межах контрольованого середовища [5].

Таким чином, предметна область системи включає такі ключові процеси: отримання та аналіз теми від користувача, веб-пошук актуальної інформації за темою (research), автоматичне планування структури презентації, генерацію вмісту слайдів через LLM, підбір зображень, надання інструментів редагування та перегляду, а також експорт готового результату у поширені формати.

1.2 Моделювання предметної області

Для формалізованого опису предметної області системи створення презентацій розроблено комплекс UML-діаграм, що відображає функціональні вимоги, поведінку системи та взаємодію між її складовими [6].

Діаграма прецедентів (рис. 1.1) визначає дійових осіб системи та перелік функцій, доступних кожному з них. В системі виділено двох акторів: користувач та AI-сервіс. Користувач взаємодіє із системою через такі прецеденти: введення теми презентації, генерація слайдів, редагування слайдів та експорт презентації. AI-сервіс виконує обробку тексту та безпосередню генерацію вмісту слайдів у відповідь на запити системи.

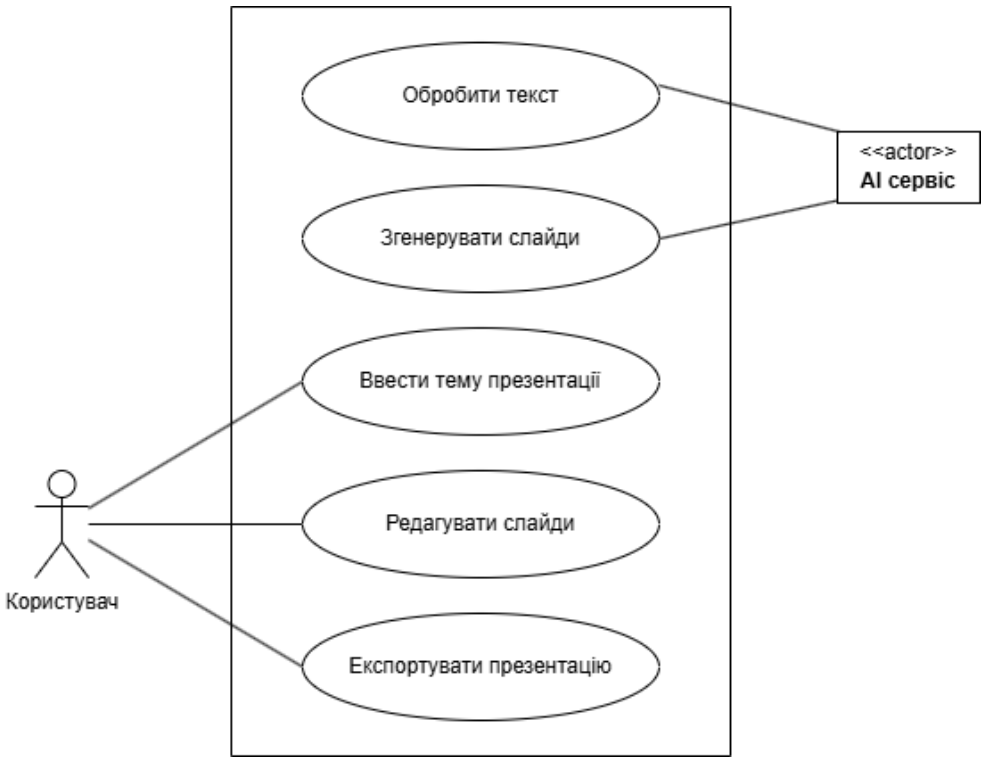


Рис. 1.1 - Діаграма прецедентів предметної області «створення презентацій»

Діаграма послідовності (рис. 1.2) ілюструє хронологічний порядок взаємодії між актором «Автор презентації», системою та зовнішнім актором «Пошукова система». Послідовність охоплює повний цикл роботи: вибір теми, пошук інформації у зовнішніх джерелах, додавання та наповнення слайдів, редагування оформлення, публікацію та експорт у зовнішній формат. Діаграма наочно показує, що система відповідає на кожну дію користувача підтвердженням виконання операції.

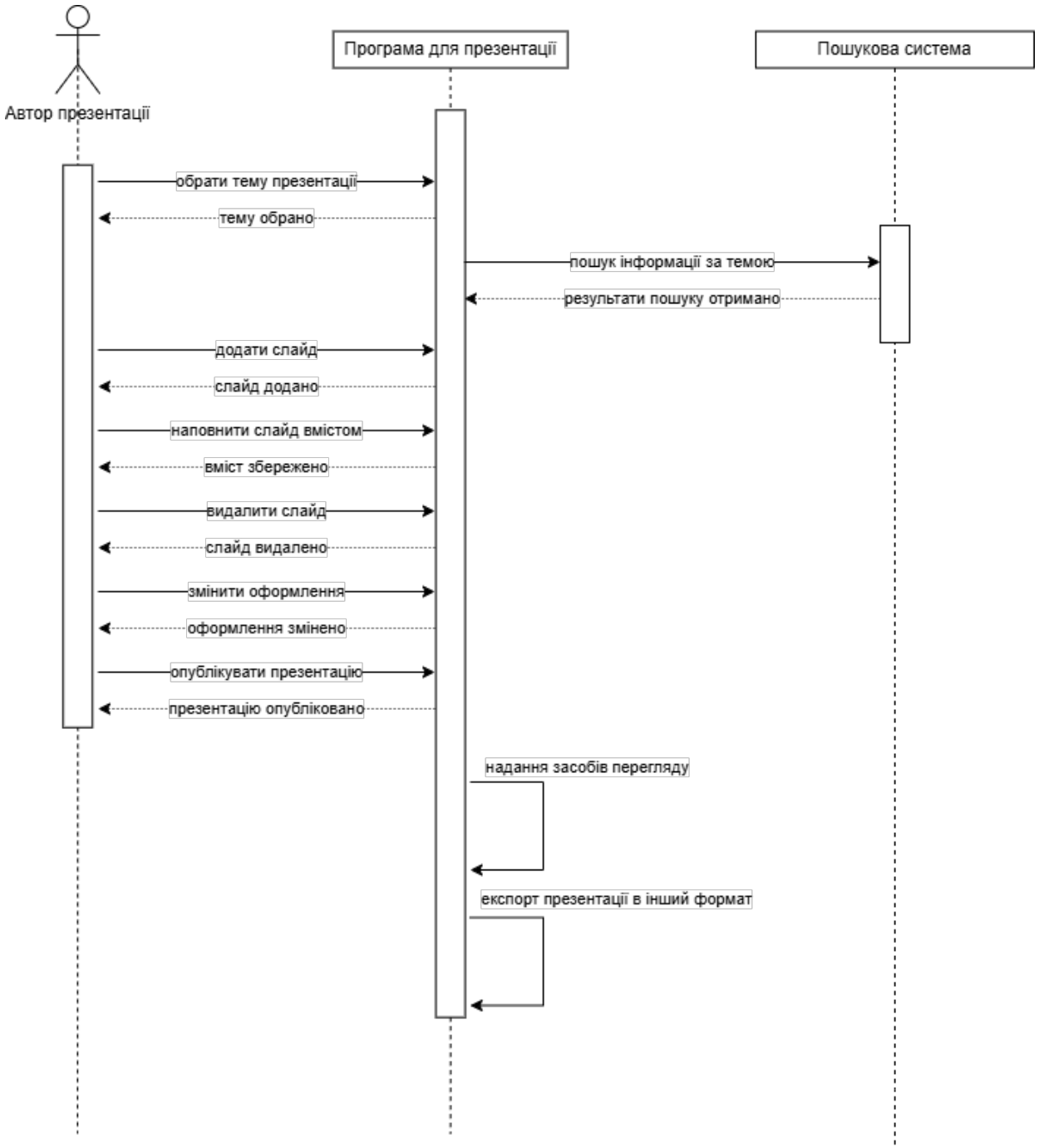


Рис. 1.2 - Діаграма послідовності взаємодії користувача із системою

Діаграма активності (рис. 1.3) описує алгоритм основного процесу роботи із системою. Процес починається з вибору теми користувачем, після чого виконується пошук інформації за темою - якщо результатів немає, запит уточнюється. Далі система здійснює пошук релевантних зображень та створює план презентації. На основі плану генеруються слайди, які користувач може

редагувати. Завершальним кроком є розгалуження: користувач може або експортувати презентацію у формат PDF, або запустити режим демонстрації.

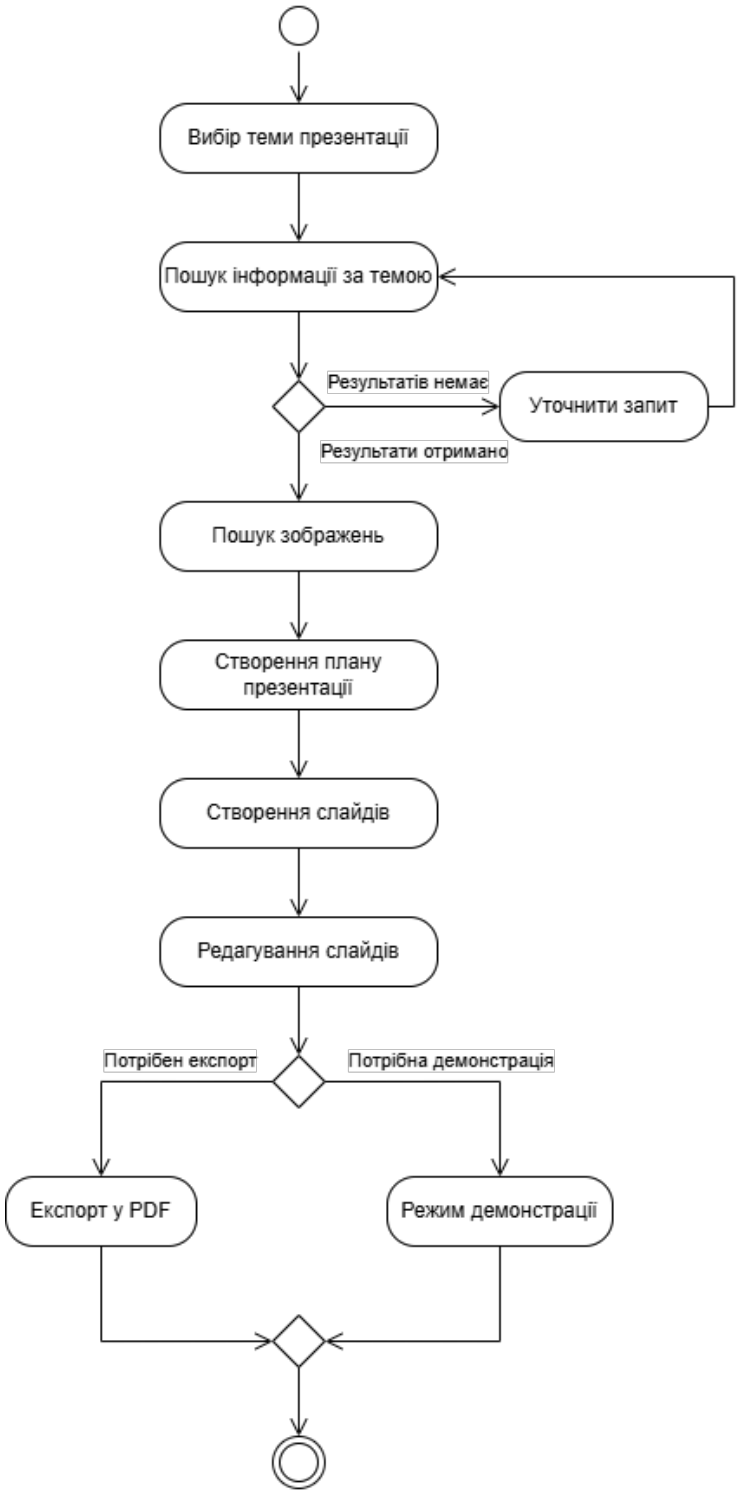


Рис. 1.3 - Діаграма активності процесу роботи із системою

Абстракції предметної області, виявлені в процесі аналізу, відображено на рис. 1.4. Основними сутностями системи є: User - зареєстрований користувач із обліковими даними; Presentation - об'єкт презентації, що належить користувачу та містить масив слайдів у JSON-форматі; Prompt - текстовий запит із параметрами генерації; SlideGeneratorAI - абстракція LLM-провайдера, що генерує слайди; Slide - окремий слайд у складі презентації [6].

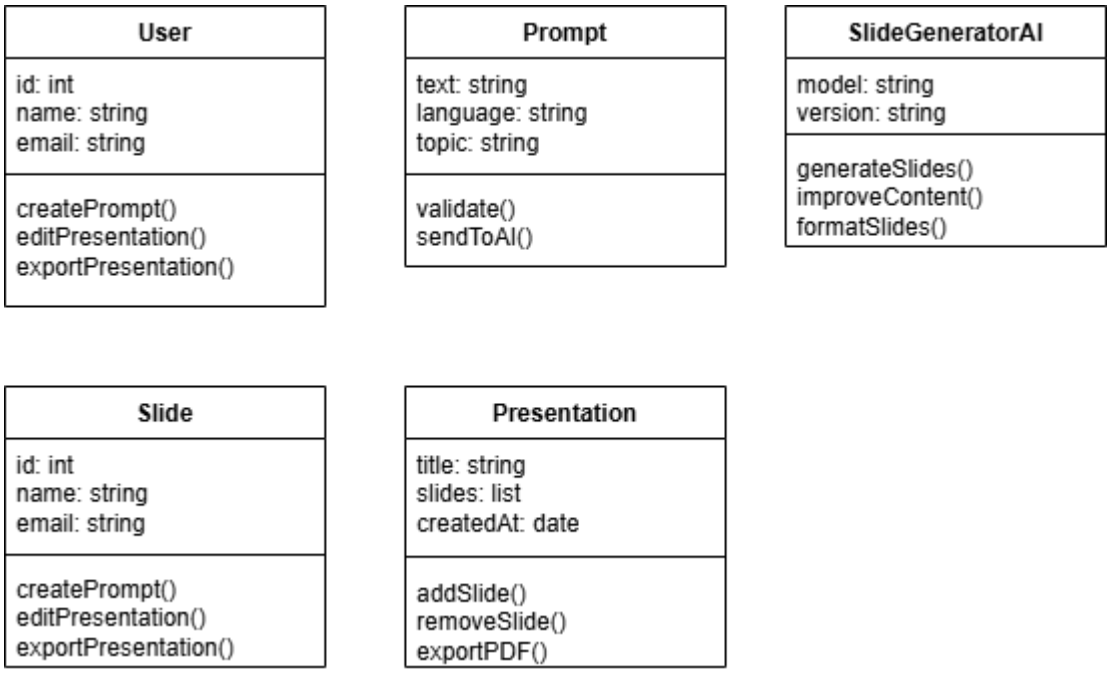


Рис. 1.4 - Сукупність абстракцій предметної області системи Slide AI

1.3 Огляд існуючих рішень

Ринок засобів для створення презентацій поділяється на два основні класи: традиційні редактори з ручним формуванням вмісту та сучасні AI-орієнтовані системи, що автоматизують частину або весь процес генерації. Для визначення вимог до розроблюваної системи було проведено огляд і порівняння найбільш поширених представників обох класів.

Microsoft PowerPoint є галузевим стандартом серед традиційних редакторів. Програма надає широкі можливості ручного форматування, анімації та шаблонів, проте формування змістовного наповнення слайдів повністю

покладається на користувача. Починаючи з версії Microsoft 365, до продукту інтегровано функцію Copilot, яка може генерувати чернетки слайдів на основі текстового опису, однак ця функція доступна лише в платній підписці та передбачає передачу даних на сервери Microsoft [7].

Google Slides - хмарний редактор, що дозволяє спільну роботу в реальному часі. Сервіс безкоштовний у базовому варіанті, проте не підтримує автоматичну генерацію вмісту без сторонніх розширень. Усі дані зберігаються на серверах Google, що може бути критичним для конфіденційних матеріалів [8].

Gamma.app - один із перших спеціалізованих AI-генераторів презентацій. Система дозволяє отримати структуровану презентацію за текстовим промптом протягом кількох секунд. Серед переваг - якісний дизайн та зручний інтерфейс. Суттєвими недоліками є обов'язкова реєстрація, хмарна обробка запитів, обмежена безкоштовна квота та відсутність можливості використання власних LLM [9].

Beautiful.ai - хмарний AI-сервіс для генерації слайдів, орієнтований на корпоративних користувачів. Платформа пропонує «розумні» шаблони, що автоматично адаптують макет під контент. Продукт є повністю платним і не пропонує локальної обробки даних [10].

Tome - AI-застосунок для сторітелінгу та презентацій, що генерує контент на основі промптів та інтегрується із сервісами генерації зображень. Сервіс доступний за підпискою, обробка відбувається виключно у хмарі [11].

Порівняльний аналіз розглянутих рішень за ключовими критеріями наведено у табл. 1.1.

Таблиця 1.1

Порівняльний аналіз існуючих рішень

Критерій	PowerPoint	Google Slides	Gamma.app	Beautiful.ai
AI-генерація слайдів	Частково	Ні	Так	Так
Безкоштовне використання	Частково	Так	Частково	Ні
Локальна обробка даних	Ні	Ні	Ні	Ні
Вибір LLM-провайдера	Ні	Ні	Ні	Ні
Редактор компонентів	Так	Так	Частково	Частково
Експорт у PDF	Так	Так	Так	Так
Збереження презентацій	Так	Так	Так	Так
Open source	Ні	Ні	Ні	Ні

Проведений аналіз (табл. 1.1) свідчить про те, що жодне з розглянутих рішень не забезпечує поєднання повноцінної AI-генерації, можливості вибору LLM-провайдера та локальної обробки даних. Виявлені недоліки існуючих рішень визначають вимоги до системи, що розробляється, та формулюються у постановці задачі.

1.4 Постановка задачі

На підставі проведеного аналізу предметної області та виявлених недоліків існуючих рішень сформульовано таку постановку задачі.

Необхідно розробити веб-застосунок для автоматичної генерації структурованих мультимедійних презентацій на основі природномовного опису теми, що вводиться користувачем. Система повинна забезпечувати:

- автентифікацію та реєстрацію користувачів із збереженням персональних даних та налаштувань у реляційній базі даних;
- генерацію повноцінних презентацій у форматі JSON-дерева слайдів на основі текстового промпту із підтримкою стрімінгової передачі результату;
- підтримку кількох LLM-провайдерів (OpenRouter, Google Gemini, LM Studio) з можливістю перемикання між ними в налаштуваннях;
- веб-пошук (research-агент) для збагачення контенту актуальними даними перед генерацією;
- підтримку файлових вкладень (PDF-документи та зображення) як контексту для генерації;
- повноцінний візуальний редактор компонентів слайдів із підтримкою drag-and-drop переміщення елементів та AI-редагування окремих слайдів;
- систему тем оформлення (dark, light, ocean, forest, sunset) із динамічною зміною кольорової схеми всіх компонентів;
- збереження та завантаження презентацій із автоматичним збереженням змін;
- отримання релевантних зображень через Pexels API та їх інтеграцію у слайди;
- експорт готових презентацій у формат PDF з відтворенням реального рендеру браузера через Puppeteer;
- розгортання системи на хмарній платформі Railway.

Система реалізується у вигляді Single Page Application (SPA) на основі React та Vite з Express.js-сервером на бекенді. Зберігання даних здійснюється у реляційній базі даних. Взаємодія між клієнтом і сервером відбувається через REST API із JWT-автентифікацією. Усі компоненти системи функціонують в єдиному Node.js-процесі, що спрощує розгортання та супровід [12].

Висновки до розділу 1

У першому розділі проведено аналіз предметної області системи автоматизованої генерації презентацій. Розглянуто п'ять представників ринку - Microsoft PowerPoint, Google Slides, Gamma.app, Beautiful.ai та Tome - та виявлено спільний недолік: відсутність поєднання AI-генерації, вільного вибору LLM-провайдера та локальної обробки даних.

Побудовано комплекс UML-діаграм предметної області: діаграму прецедентів, діаграму послідовності та діаграму активності, які формалізують вимоги до системи та поведінку її основних сценаріїв використання.

Сформульовано детальну постановку задачі, що охоплює функціональні вимоги до системи: автентифікацію користувачів, AI-генерацію слайдів із стрімінгом, підтримку кількох LLM-провайдерів, веб-пошук, файлові вкладення, візуальний редактор, систему тем, хмарне збереження та PDF-експорт.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних

Проектування інформаційного забезпечення системи Slide AI розпочато з побудови логічної моделі даних, яка визначає склад сутностей предметної області, їх атрибути та зв'язки між ними незалежно від конкретної реалізації бази даних [13].

Аналіз предметної області дозволив виділити дві основні сутності системи.

Сутність User описує зареєстрованого користувача системи. Вона є незалежною сутністю, оскільки має власний первинний ключ і не залежить від інших сутностей. Атрибутами сутності є: унікальний ідентифікатор користувача, електронна адреса (унікальна, без урахування регістру), хешований пароль, ім'я користувача та часова мітка реєстрації.

Сутність Presentation описує презентацію, що належить конкретному користувачу. Вона є залежною сутністю, оскільки містить зовнішній ключ, що посилається на сутність User. Атрибутами сутності є: унікальний ідентифікатор презентації, посилання на користувача-власника, назва презентації, вміст у форматі JSON (масив слайдів із компонентним деревом), ідентифікатор теми оформлення, а також часові мітки створення та останнього оновлення.

Між сутностями User та Presentation встановлено зв'язок типу «один до багатьох»: один користувач може мати довільну кількість презентацій, тоді як кожна презентація належить рівно одному користувачу. При видаленні користувача всі його презентації видаляються каскадно.

Логічну модель даних системи Slide AI наведено на рис. 2.1.

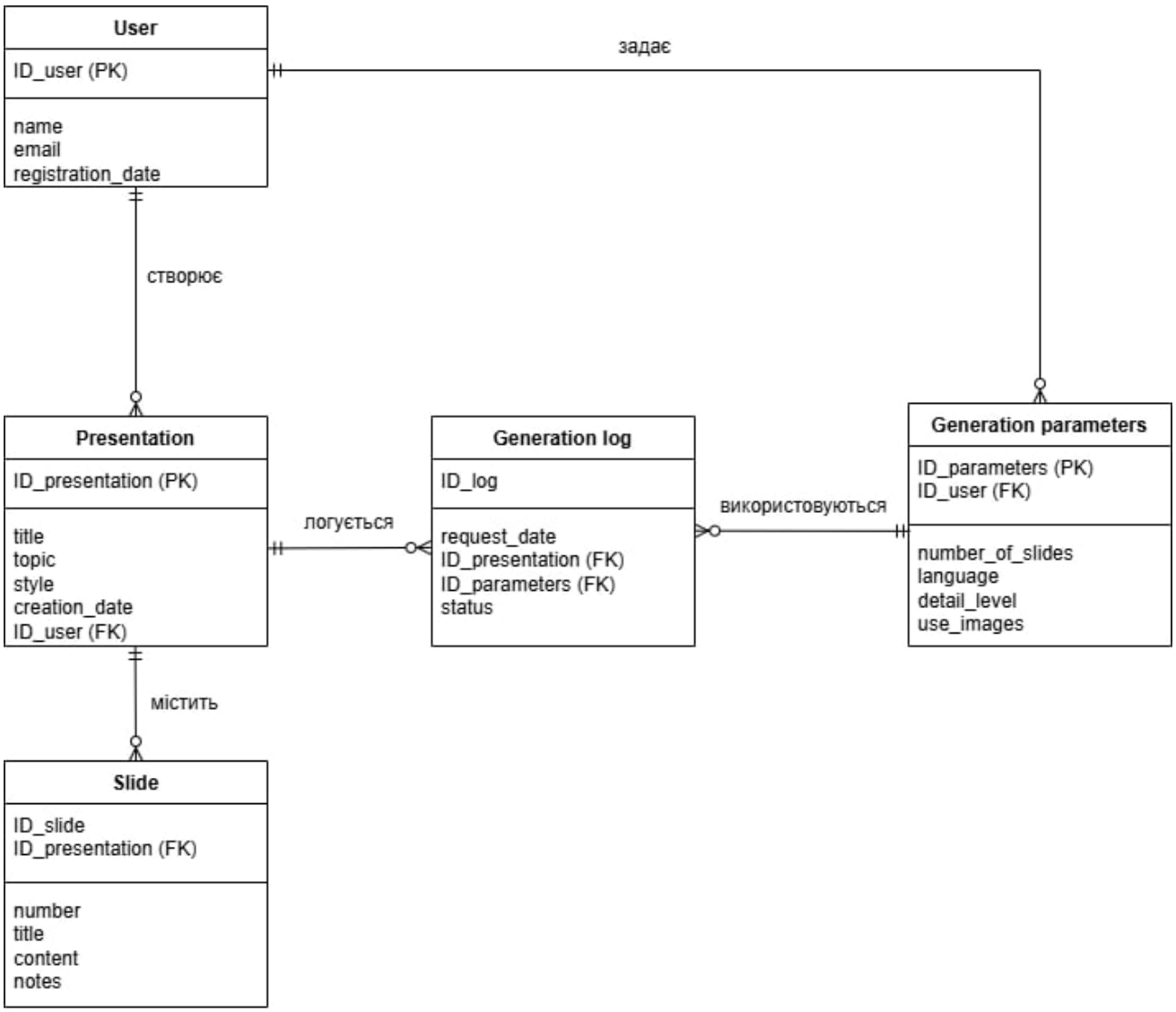


Рис. 2.1 - Логічна модель даних системи Slide AI

2.2 Вибір системи управління базами даних

Вибір СУБД є одним із ключових архітектурних рішень, що визначає надійність, продуктивність та складність розгортання системи. При виборі розглядалися дві реляційні СУБД: SQLite та PostgreSQL .

SQLite - вбудована реляційна СУБД, що зберігає всю базу даних в одному файлі на диску. Не потребує окремого серверного процесу, налаштування підключень або управління користувачами. Бібліотека better-sqlite3 надає синхронний API для роботи з SQLite у середовищі Node.js, що спрощує код

серверної частини та усуває необхідність управління асинхронними з'єднаннями з базою даних [14].

PostgreSQL - повнофункціональна клієнт-серверна реляційна СУБД з відкритим вихідним кодом. Підтримує складні запити, транзакції, реплікацію та горизонтальне масштабування. PostgreSQL є галузевим стандартом для виробничих веб-застосунків, що розгортаються у хмарних середовищах [15].

Порівняльний аналіз обох СУБД за ключовими критеріями наведено у табл. 2.1.

Таблиця 2.1

Порівняльний аналіз СУБД SQLite та PostgreSQL

Критерій	SQLite	PostgreSQL
Архітектура	Вбудована (serverless)	Клієнт-серверна
Розгортання	Не потребує налаштування	Окремий сервер/контейнер
Продуктивність (читання)	Висока	Висока
Конкурентний запис	Обмежений	Повна підтримка
Масштабованість	Низька	Висока
Складність інтеграції	Мінімальна	Середня
Підтримка хмарних платформ	Обмежена	Повна
Відкритий вихідний код	Так	Так

На етапі розробки та локального тестування для зберігання даних обрано SQLite. Основні причини такого вибору:

- відсутність потреби у налаштуванні окремого серверного процесу бази даних суттєво спрощує локальне розгортання і знижує поріг входу для запуску проекту;
- синхронний API бібліотеки better-sqlite3 дозволяє писати лаконічний серверний код без управління пулом з'єднань та обробки промісів;

- навантаження системи на етапі розробки та початкового запуску не передбачає інтенсивного конкурентного запису, тому обмеження SQLite щодо паралельного запису не є критичним;
- вся база даних зберігається в одному файлі, що спрощує резервне копіювання та перенесення даних [16].

Перед розгортанням системи на хмарній платформі Railway планується міграція з SQLite на PostgreSQL. Хмарна платформа Railway надає вбудовану підтримку PostgreSQL як керованого сервісу, що усуває необхідність самостійного адміністрування бази даних у виробничому середовищі. Перехід між СУБД потребуватиме мінімальних змін у коді завдяки використанню схожого SQL-синтаксису в обох системах [17].

2.3 Фізична структура бази даних

На основі логічної моделі даних розроблено фізичну структуру бази даних, що визначає конкретні типи даних, обмеження цілісності та індекси для кожної таблиці. Фізичну структуру реалізовано засобами мови SQL [18].

Слід зазначити, що на етапі первісного проєктування (до початку реалізації) логічна модель містила п'ять сутностей: User, Presentation, Slide, Generation log та Generation parameters. Однак у процесі розробки та оптимізації архітектури було прийнято рішення зберігати масив слайдів безпосередньо у полі data таблиці presentations у форматі JSON, що усунуло потребу в окремих таблицях для слайдів і журналу генерації. Така денормалізація є виправданою з огляду на характер даних: слайди однієї презентації завжди зчитуються та записуються як єдиний об'єкт, що спрощує логіку застосунку та підвищує швидкодію читання [19].

Таблиця users призначена для зберігання облікових даних зареєстрованих користувачів системи. Структуру таблиці наведено у табл. 2.2.

Таблиця 2.2

Структура таблиці users

Поле	Тип	Обмеження	Опис
id	INTEGER	PK, AUTOINCREMENT	Унікальний ідентифікатор
email	TEXT	NOT NULL, UNIQUE, NOCASE	Електронна адреса (без урахування регістру)
password	TEXT	NOT NULL	Хеш пароля (bcrypt)
name	TEXT	NOT NULL DEFAULT ''	Ім'я користувача
created_at	INTEGER	NOT NULL DEFAULT (unixepoch())	Часова мітка реєстрації (Unix timestamp)

Таблиця presentations призначена для зберігання презентацій користувачів. Поле data містить повну JSON-структуру всіх слайдів презентації у вигляді серіалізованого рядка. Структуру таблиці наведено у табл. 2.3.

Таблиця 2.3

Структура таблиці presentations

Поле	Тип	Обмеження	Опис
id	INTEGER	PK, AUTOINCREMENT	Унікальний ідентифікатор
user_id	INTEGER	NOT NULL, FK → users(id)	Посилання на власника (каскадне видалення)
title	TEXT	NOT NULL DEFAULT 'Untitled'	Назва презентації
data	JSON	NOT NULL	JSON-массив слайдів (серіалізований рядок)
theme_id	TEXT	NOT NULL DEFAULT 'dark'	Ідентифікатор теми оформлення
created_at	INTEGER	NOT NULL DEFAULT (unixepoch())	Часова мітка створення (Unix timestamp)
updated_at	INTEGER	NOT NULL DEFAULT (unixepoch())	Часова мітка останнього оновлення

Для підвищення швидкодії вибірки всіх презентацій конкретного користувача на поле user_id таблиці presentations створено індекс idx_presentations_user. Індекс дозволяє уникнути повного сканування таблиці при виконанні запитів типу «отримати всі презентації користувача з певним ідентифікатором», що є найбільш частою операцією читання у системі [18].

Цілісність даних між таблицями забезпечується механізмом зовнішніх ключів. Зовнішній ключ user_id у таблиці presentations посилається на первинний ключ таблиці users із параметром ON DELETE CASCADE, що гарантує автоматичне видалення всіх презентацій користувача при видаленні його

облікового запису. Схему бази даних із зображенням зв'язків між таблицями наведено на рис. 2.2.

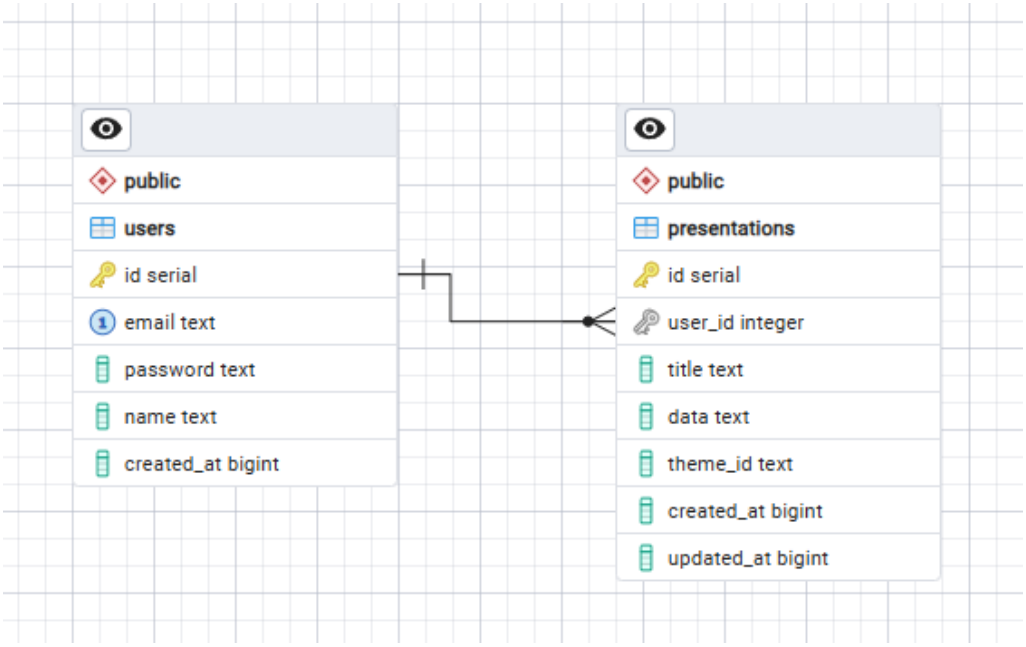


Рис. 2.2 - Фізична структура бази даних системи Slide AI

Висновки до розділу 2

У другому розділі спроектовано інформаційне забезпечення системи Slide AI. Побудовано логічну модель даних, що містить дві основні сутності - User та Presentation - зі зв'язком «один до багатьох». Прийняте рішення щодо зберігання масиву слайдів у форматі JSON безпосередньо у таблиці presentations є обґрунтованою денормалізацією, що спрощує архітектуру та підвищує швидкодію читання даних.

Проведено порівняльний аналіз СУБД SQLite та PostgreSQL. На етапі розробки обрано SQLite завдяки простоті розгортання та відсутності необхідності у налаштуванні окремого серверного процесу. Перед розгортанням у виробничому середовищі планується міграція на PostgreSQL, яку підтримує хмарна платформа Railway.

Розроблено фізичну структуру бази даних: визначено типи даних, обмеження цілісності та індекси для таблиць users і presentations.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура системи

Система Slide AI побудована за клієнт-серверною архітектурою та складається з двох основних частин: клієнтської (frontend) та серверної (backend), що функціонують як єдиний Node.js-застосунок. Така організація дозволяє спростити розгортання, оскільки Express-сервер одночасно обслуговує REST API та роздає статичні файли зібраного React-застосунку [20].

Клієнтська частина являє собою Single Page Application (SPA), реалізовану на основі бібліотеки React з використанням збирача Vite. Маршрутизація між сторінками здійснюється на стороні клієнта засобами React Router. Управління глобальним станом застосунку реалізовано за допомогою бібліотеки Zustand, яка забезпечує реактивне оновлення інтерфейсу при зміні даних. Стилізацію компонентів виконано засобами Tailwind CSS [21].

Серверна частина побудована на основі фреймворку Express.js та виконується у середовищі Node.js. Сервер реалізує REST API для обробки запитів від клієнта, забезпечує автентифікацію користувачів на основі JWT-токенів, взаємодіє з базою даних через бібліотеку better-sqlite3, а також виконує функції проксі-сервера для запитів до зовнішніх LLM-провайдерів та Pexels API. Окрім цього, сервер запускає екземпляр браузера Chromium через Puppeteer для рендерингу та експорту презентацій у формат PDF [22].

Взаємодія між клієнтською та серверною частинами відбувається через HTTP/HTTPS-протокол. Клієнт надсилає запити до REST API сервера, передаючи JWT-токен у заголовку Authorization для підтвердження автентичності. Генерація слайдів відбувається у режимі стрімінгу: сервер отримує відповідь від LLM-провайдера у вигляді потоку токенів та передає її клієнту через Server-Sent Events (SSE), що дозволяє відображати слайди по мірі їх генерації без очікування повної відповіді [23].

Зовнішні сервіси, з якими взаємодіє система:

- LLM-провайдери - OpenRouter, Google Gemini або LM Studio (локальна модель). Провайдер обирається у налаштуваннях системи; запити до API провайдера виконуються на стороні сервера для захисту API-ключів;
- Pexels API - сервіс для пошуку та отримання стокових фотографій. Зображення підбираються автоматично на основі ключових слів теми слайду та вбудовуються у відповідні компоненти;
- Tavily API - використовується research-агентом для виконання веб-пошуку та збагачення контексту перед генерацією презентації. Запити проксіюються через власний сервер системи.

3.2 Діаграма класів та кооперацій

Діаграма класів (рис. 3.2) відображає статичну структуру основних абстракцій системи та відносини між ними. У системі Slide AI виділено п'ять ключових класів: User, Prompt, SlideGeneratorAI, Slide та Presentation [6].

Клас User представляє зареєстрованого користувача системи. Атрибутами є ідентифікатор, ім'я та електронна адреса. Клас надає методи createPrompt() для ініціювання генерації, editPresentation() для редагування існуючої презентації та exportPresentation() для її експорту у зовнішній формат.

Клас Prompt інкапсулює параметри запиту на генерацію: текст промпту, мову презентації та тему. Методи validate() перевіряють коректність вхідних даних перед надсиланням запиту, а sendToAI() здійснює безпосереднє звернення до LLM-провайдера.

Клас SlideGeneratorAI є абстракцією LLM-провайдера. Атрибути model та version визначають конкретну модель, що використовується. Методи generateSlides() запускає процес генерації, improveContent() дозволяє покращити вміст окремого слайду, formatSlides() приводить отриманий JSON до внутрішнього формату системи.

Клас Slide описує окремий слайд у складі презентації. Клас Presentation є агрегатом, що містить впорядкований список слайдів. Методи addSlide() та

removeSlide() забезпечують управління складом презентації, exportPDF() запускає процес рендерингу та збереження у PDF-файл.

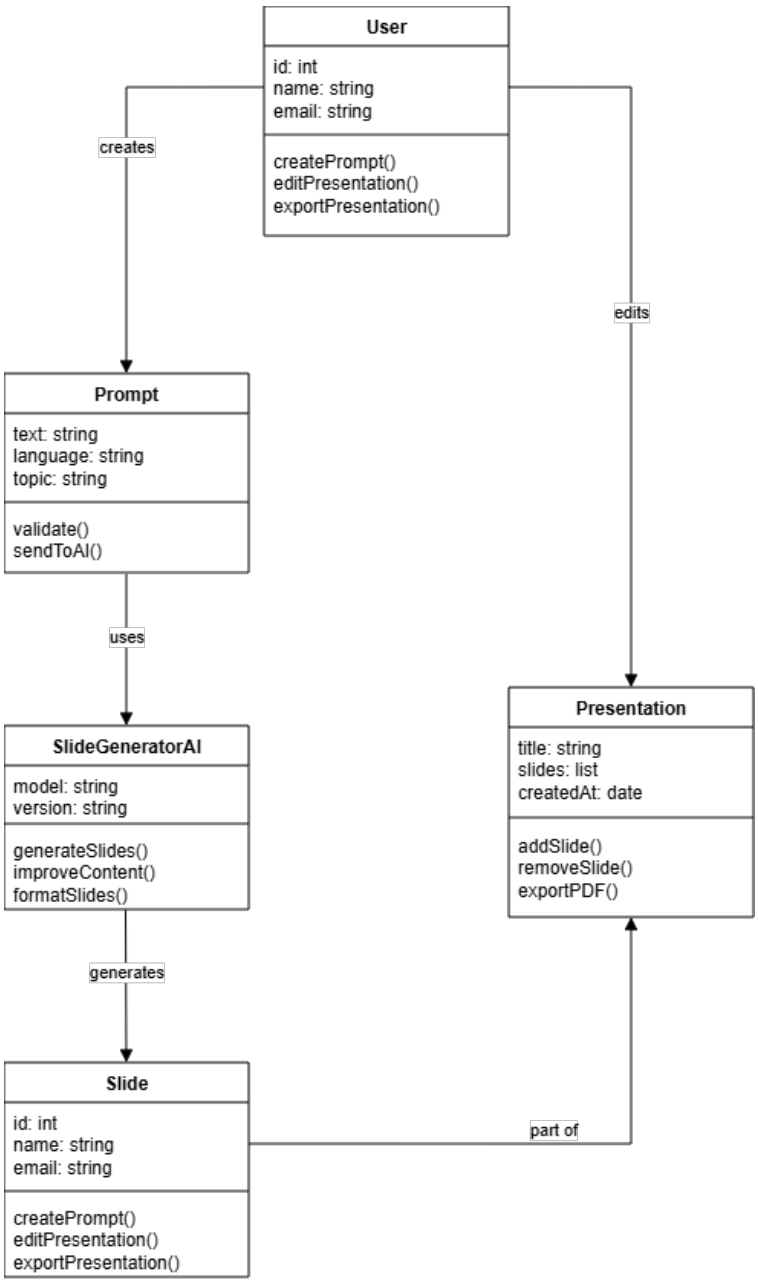


Рис. 3.2 - Діаграма класів системи Slide AI

Для деталізації взаємодії між класами побудовано дві діаграми кооперацій. Перша кооперація «Створення промпту» (рис. 3.3) показує послідовність взаємодії між класами User, Prompt та SlideGeneratorAI у процесі формування та надсилання запиту на генерацію. Друга кооперація «Редагування презентації»

(рис. 3.4) відображає взаємодію між SlideGeneratorAI, User, Slide та Presentation при редагуванні вмісту слайдів [6].

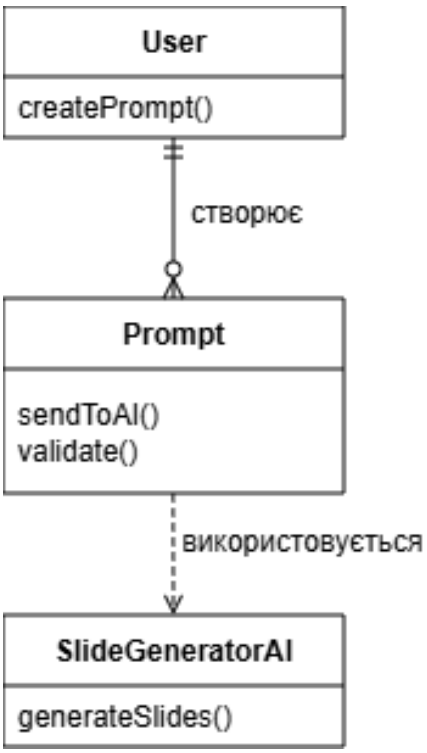


Рис. 3.3 - Кооперація «Створення промпту»

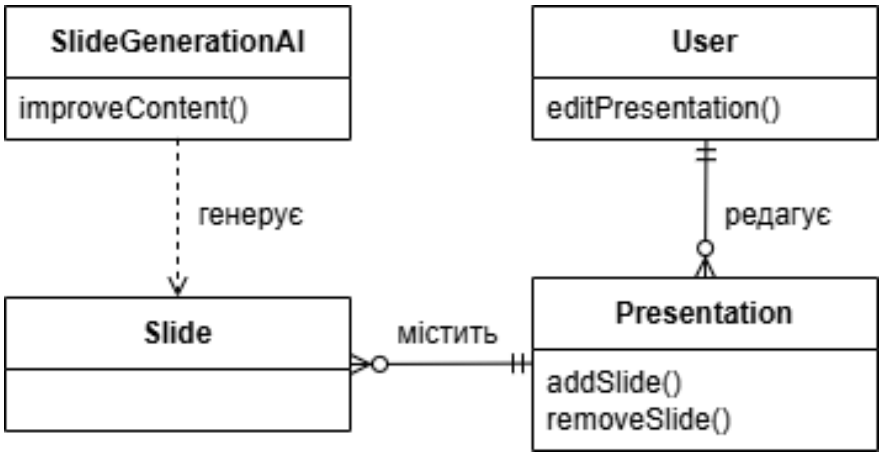


Рис. 3.4 - Кооперація «Редагування презентації»

3.3 Діаграма пакетів та компонентів

Діаграма пакетів (рис. 3.5) відображає організаційну структуру програмного забезпечення та залежності між його складовими частинами. Система розподілена між двома основними пакетами верхнього рівня: Клієнт та Сервер [22].

Пакет Клієнт містить два підпакети. Підпакет «Інтерфейс користувача» об'єднує компоненти веб-сторінок та навігації між ними. Підпакет «Презентація» містить компоненти редактора слайдів, контролер презентації та бібліотеку UI-компонентів слайдів. Клієнтська частина імпортує дані з серверної через REST API.

Пакет Сервер містить три підпакети. Підпакет «AI-ядро» включає агенти генерації (presentation-agent, slide-generator, research-agent, slide-fixer) та адаптери LLM-провайдерів. Підпакет «Сховище даних» відповідає за роботу з базою даних: зберігання облікових даних користувачів та презентацій.

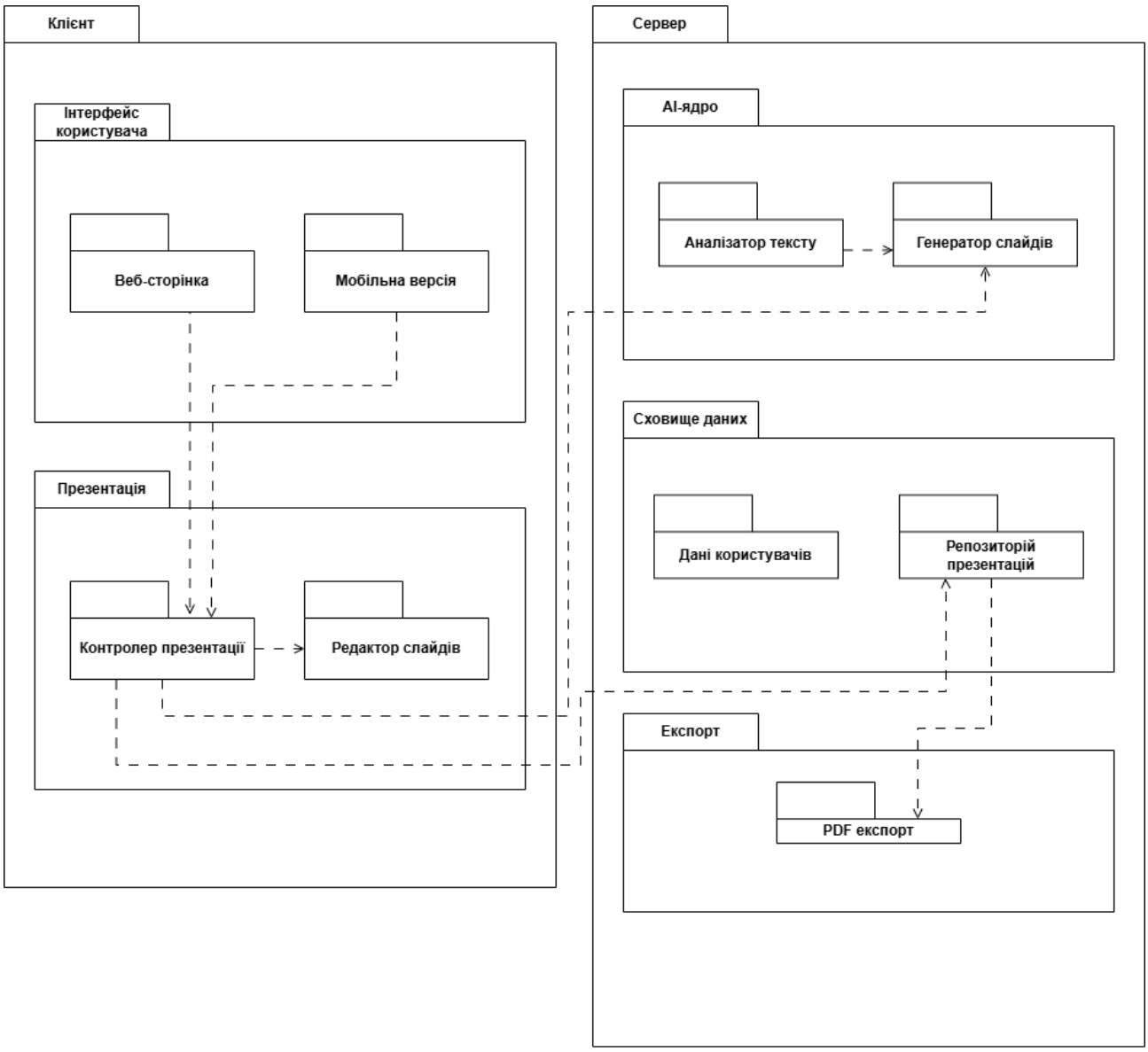


Рис. 3.5 - Діаграма пакетів системи Slide AI

Діаграма компонентів (рис. 3.6) відображає архітектуру системи Slide AI на рівні основних замінюваних компонентів та інтерфейсів взаємодії між ними.

На клієнтській стороні виділено два компоненти: Browser (середовище виконання) та Frontend (React + Vite SPA). Взаємодія між ними відбувається через протокол HTTPS. Frontend з'єднується з Backend через інтерфейс REST API - останній забезпечує стрімінгову передачу слайдів у процесі генерації.

Компонент Backend (Express.js) є центральним вузлом системи та взаємодіє з усіма іншими компонентами через чітко визначені інтерфейси: з Database (PostgreSQL) - через інтерфейс SQL, з AI Agents - через інтерфейс Agent

API, із зовнішнім Payment Service (Stripe / LiqPay) - через Payment API, та із зовнішнім Image Provider (Pexels) - через Image API.

Компонент AI Agents (LLM Integration) відповідає за генерацію вмісту слайдів та веб-пошук. Він взаємодіє з двома зовнішніми провайдерами: LLM Providers (Gemini або LM Studio) через інтерфейси Gemini API і OpenAI compatible API та Research Provider (Tavily) через інтерфейс Research API.

Така архітектура забезпечує слабку зв'язаність компонентів: кожен із них може бути замінений незалежно - наприклад, LLM-провайдер або платіжний сервіс - без зміни інших частин системи.

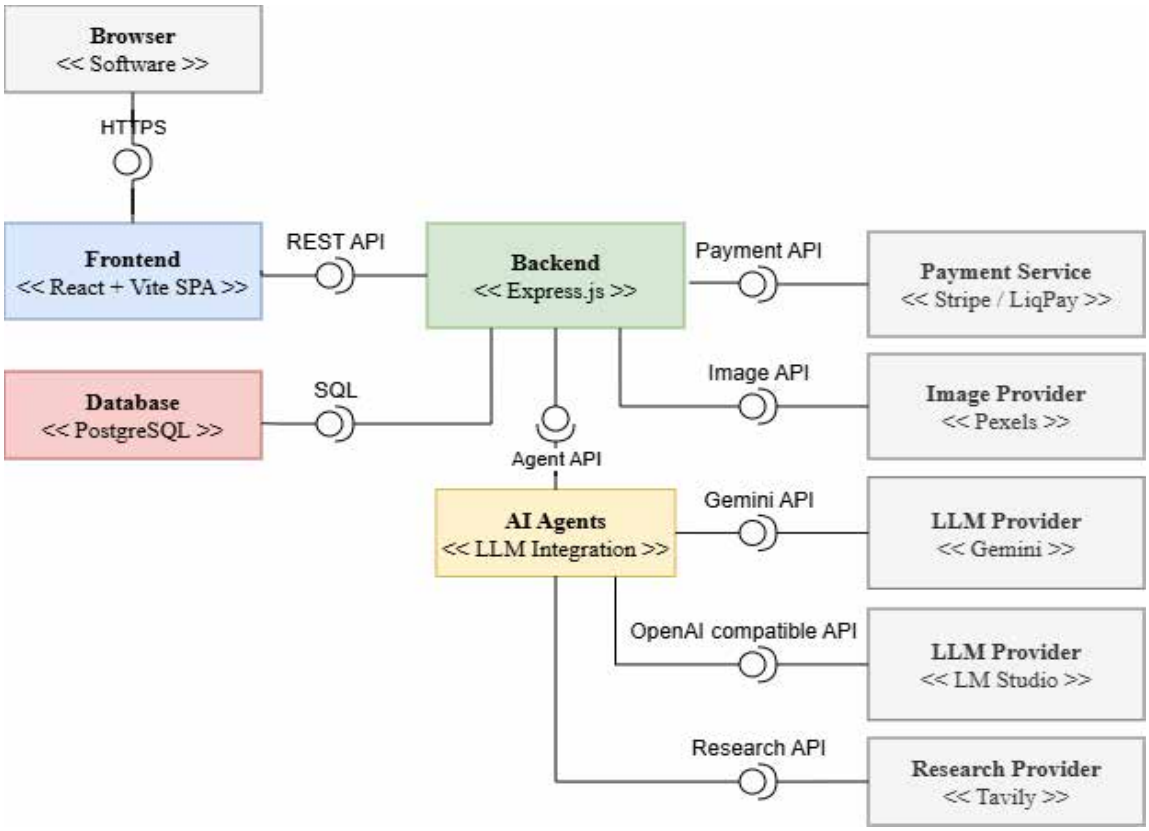


Рис. 3.6 - Діаграма компонентів системи Slide AI

3.4 Вибір інструментарію розробки

Вибір засобів розробки визначається архітектурними рішеннями, прийнятими у попередніх розділах, а також вимогами до продуктивності, зручності супроводу та можливості розгортання у хмарному середовищі. Перелік основних технологій та бібліотек наведено у табл. 3.1.

Таблиця 3.1

Інструментарій розробки системи Slide AI

Технологія / бібліотека	Версія	Призначення
React	19	Бібліотека побудови інтерфейсу користувача
TypeScript	5.x	Статична типізація JavaScript
Vite	5.x	Інструмент збірки та сервер розробки
React Router	6.x	Клієнтська маршрутизація між сторінками
Zustand	4.x	Управління глобальним станом застосунку
Tailwind CSS	3.x	Утилітарний CSS-фреймворк для стилізації
Recharts	2.x	Бібліотека побудови графіків та діаграм у слайдах
Express.js	4.x	Серверний фреймворк для побудови REST API
Node.js	20+	Серверне середовище виконання JavaScript
better-sqlite3	9.x	Синхронний драйвер SQLite для Node.js
jsonwebtoken	9.x	Генерація та верифікація JWT-токенів
bcryptjs	2.x	Хешування паролів користувачів
Puppeteer	22.x	Автоматизація Chromium для рендерингу PDF
jsPDF	2.x	Збірка сторінок у PDF-документ
pdfjs-dist	4.x	Розбір PDF-файлів, завантажених як вкладення
Railway	-	Хмарна платформа для розгортання застосунку

React обрано як основу клієнтської частини завдяки компонентній моделі, широкій екосистемі та підтримці TypeScript. Використання TypeScript забезпечує статичну типізацію на всьому клієнтському коді, що суттєво знижує кількість помилок при роботі зі складними JSON-структурами слайдів [24].

Vite обрано замість Create React App завдяки значно швидшому холодному старту сервера розробки та миттєвому оновленню модулів (HMR). Zustand є більш легкою альтернативою Redux: для управління станом не потребує бойлерплейту та добре інтегрується з TypeScript [25].

Express.js обрано як серверний фреймворк завдяки мінімалістичному API, широкій підтримці middleware та простоті налаштування маршрутів. Puppeteer використовується для рендерингу слайдів у Chromium з подальшим збереженням результату у PDF, що гарантує точне відтворення CSS-стилів та шрифтів порівняно з бібліотеками, що генерують PDF без участі браузера [26].

3.5 Алгоритм генерації презентацій та ключові модулі

Центральним процесом системи є генерація презентації на основі текстового промпту користувача. Алгоритм реалізовано у модулі presentation-agent.ts та складається з кількох послідовних етапів.

На першому етапі, якщо увімкнено research-агент, система формує серію пошукових запитів на основі теми презентації та звертається до Tavily API для виконання веб-пошуку. Отримані результати обробляються та включаються до системного промпту як додатковий контекст із актуальними даними.

Повний код модуля research-агента з реалізацією пошуку через Tavily API наведено у додатку А.

На другому етапі формується системний промпт, що містить: опис формату JSON-дерева слайдів, каталог доступних компонентів із їх пропсами, правила оформлення, контекст з результатів пошуку (за наявності) та вміст завантажених користувачем вкладень (PDF-документів або зображень). Вміст

PDF-вкладень попередньо витягується за допомогою pdfjs-dist, а зображення передаються у форматі base64.

На третьому етапі сформований запит надсилається до обраного LLM-провайдера у режимі стрімінгу. Модель генерує JSON-масив об'єктів слайдів, де кожен слайд описується деревом компонентів з типами, пропсами та дочірніми елементами. По мірі надходження токенів функція extractSlides() виконує інкрементний парсинг JSON та передає готові слайди до сховища Zustand, що забезпечує миттєве відображення результатів в інтерфейсі [23].

Лістинг функції extractSlides та основного алгоритму генерації слайдів наведено у додатку Б.

На четвертому етапі кожен слайд з JSON-дерева рекурсивно рендериться у React-компоненти модулем renderer.tsx. Функція renderNode() обходить дерево вузлів та відображає їх у відповідні компоненти з бібліотеки slide-components, застосовуючи поточну тему оформлення через React Context (useTheme).

Реалізацію функції рекурсивного рендерингу JSON-дерева слайдів наведено у додатку В.

Модуль slide-fixer.ts реалізує алгоритм AI-редагування окремого слайду. Користувач може ввести текстову інструкцію щодо бажаних змін, після чого агент отримує поточне JSON-дерево слайду та інструкцію, надсилає запит до LLM і замінює вміст слайду оновленою структурою. Це дозволяє точково виправляти або доповнювати окремі слайди без повторної генерації всієї презентації.

Модуль exportPDF.tsx реалізує процес конвертації презентації у PDF-файл. Для кожного слайду серверна частина запускає Puppeteer, відкриває спеціальний маршрут /pdf-slide із HTML-представленням слайду та виконує скріншот сторінки з роздільною здатністю 1920x1080 пікселів. Отримані зображення збираються у єдиний PDF-документ за допомогою jsPDF та повертаються клієнту для завантаження [26].

Блок-схему алгоритму генерації презентації наведено в додатку Д.

3.6 Демонстрація інтерфейсу користувача

Для розробки інтерфейсу користувача обрано React у поєднанні з Tailwind CSS. Застосунок має адаптивний дизайн та підтримує п'ять тем оформлення: dark, light, ocean, forest та sunset. Інтерфейс складається з навігаційної панелі (Navbar) та шести основних сторінок, доступ до яких здійснюється через клієнтський маршрутизатор React Router [27].

Сторінка авторизації є першою точкою входу для незареєстрованого користувача. Вона містить форму з полями для введення електронної адреси та пароля, а також перемикач між режимами входу та реєстрації. Після успішної аутентифікації сервер повертає JWT-токен, який зберігається у сховищі Zustand із подальшою персистентністю через localStorage. Інтерфейс сторінки авторизації наведено на рис. 3.7.

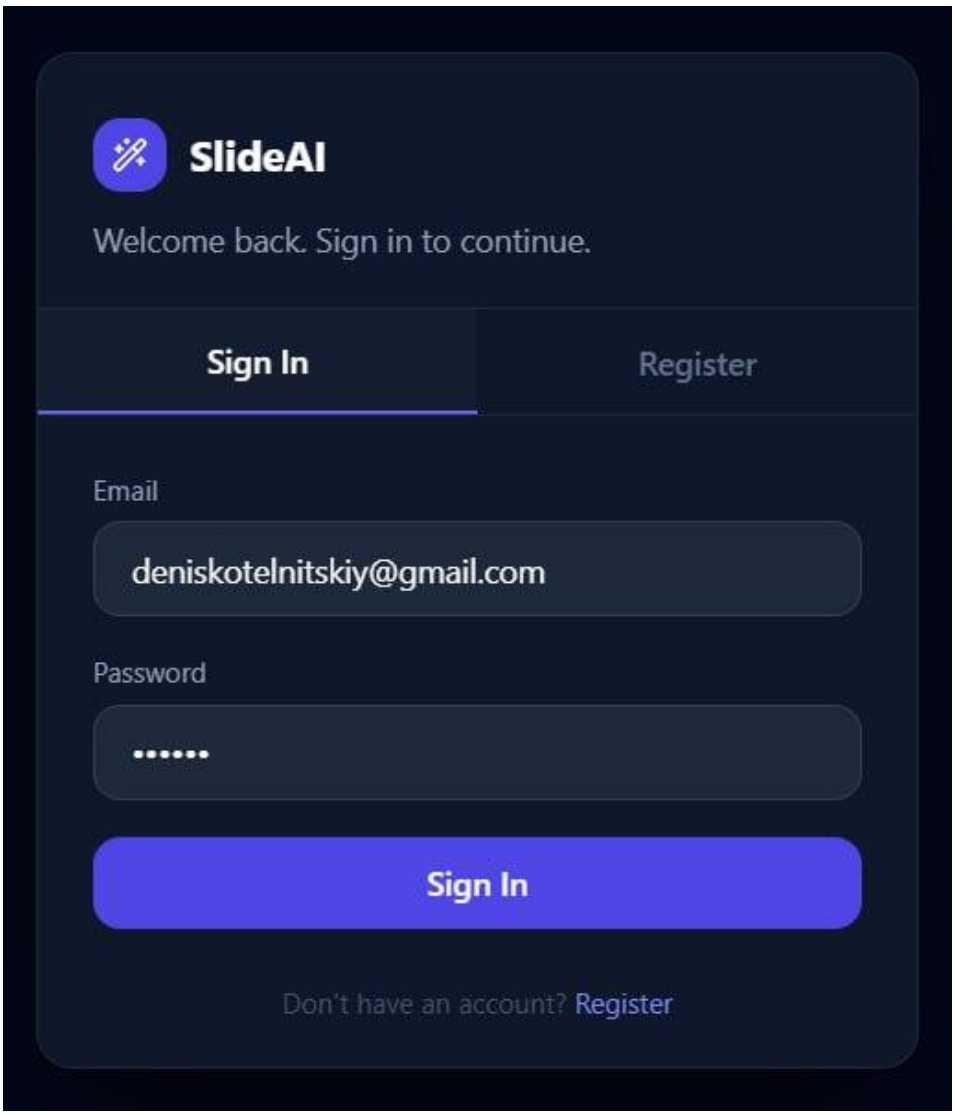


Рис. 3.7 - Сторінка авторизації системи Slide AI

Головна сторінка (HomePage) є стартовою точкою для авторизованого користувача. Вона містить поле введення промпту для генерації нової презентації, вибір кількості слайдів, мови та інших параметрів, а також перемикач увімкнення research-агента. Нижня частина сторінки демонструє приклади тематик для натхнення. Інтерфейс головної сторінки наведено на рис. 3.8.

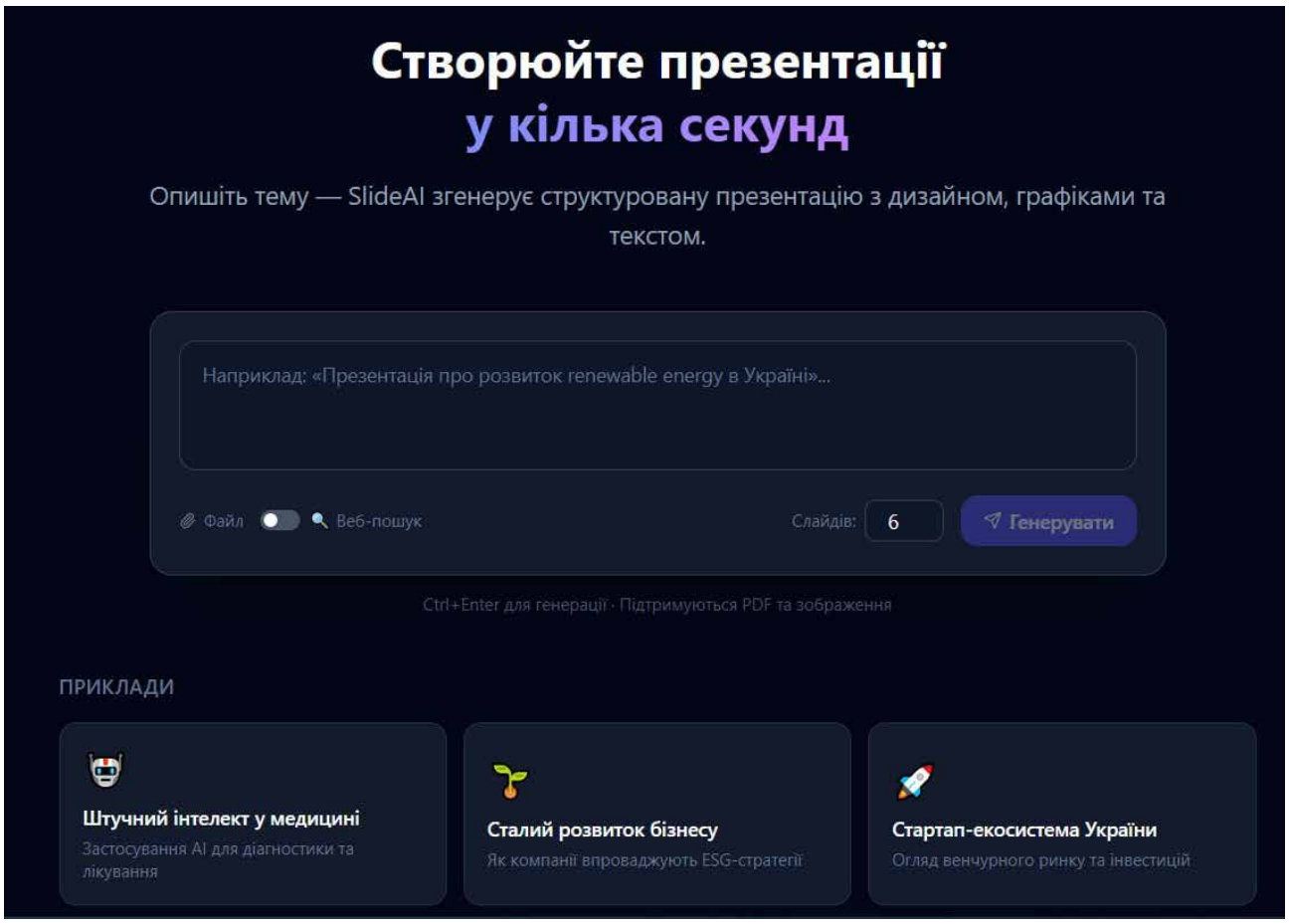


Рис. 3.8 - Головна сторінка системи Slide AI

Сторінка редактора (EditorPage) є центральним робочим середовищем системи. Вона має триколонкову структуру: ліва панель містить список слайдів із мініатюрами та елементи управління генерацією, центральна область відображає поточний слайд у форматі 16:9 із кнопками навігації, а права панель містить дві вкладки - редактор компонентів та вибір теми оформлення. Інтерфейс сторінки редактора наведено на рис. 3.9.

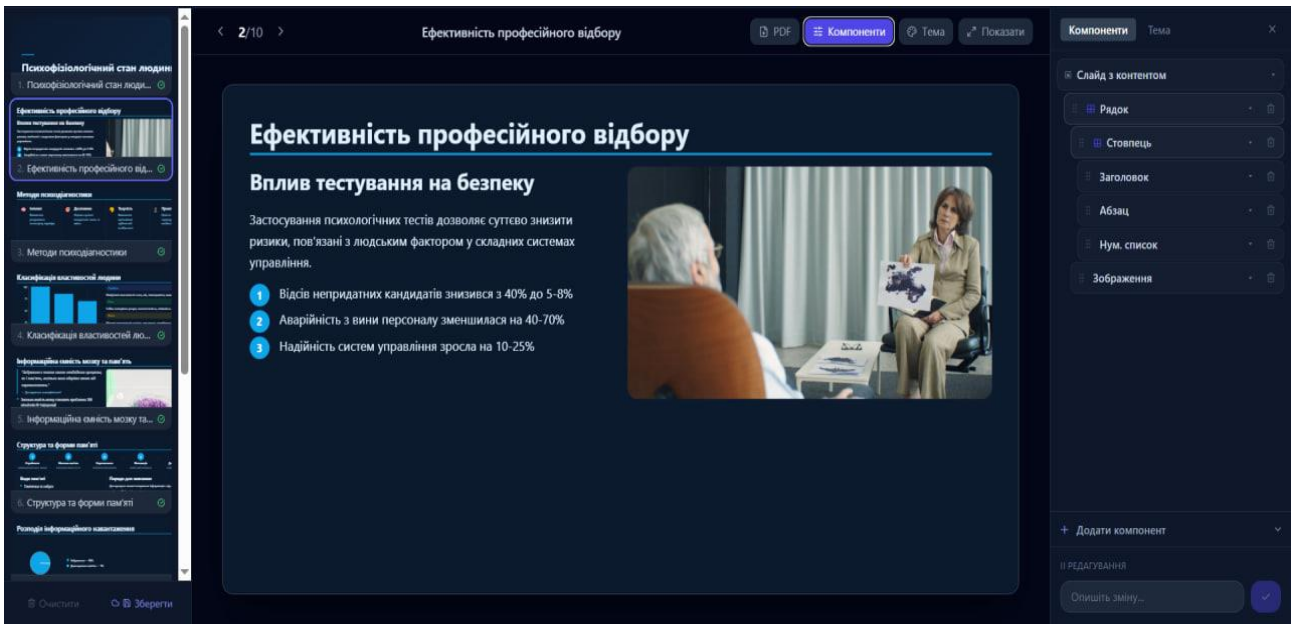


Рис. 3.9 - Сторінка редактора презентації

Приклад згенерованого слайду демонструє можливості бібліотеки компонентів системи. Слайди можуть містити текстові блоки, списки, цитати, графіки (стовпчасті, лінійні, кругові), діаграми потоку, зображення з Pexels, картки з метриками та інші компоненти. Приклад згенерованого слайду з різними компонентами наведено на рис. 3.10.

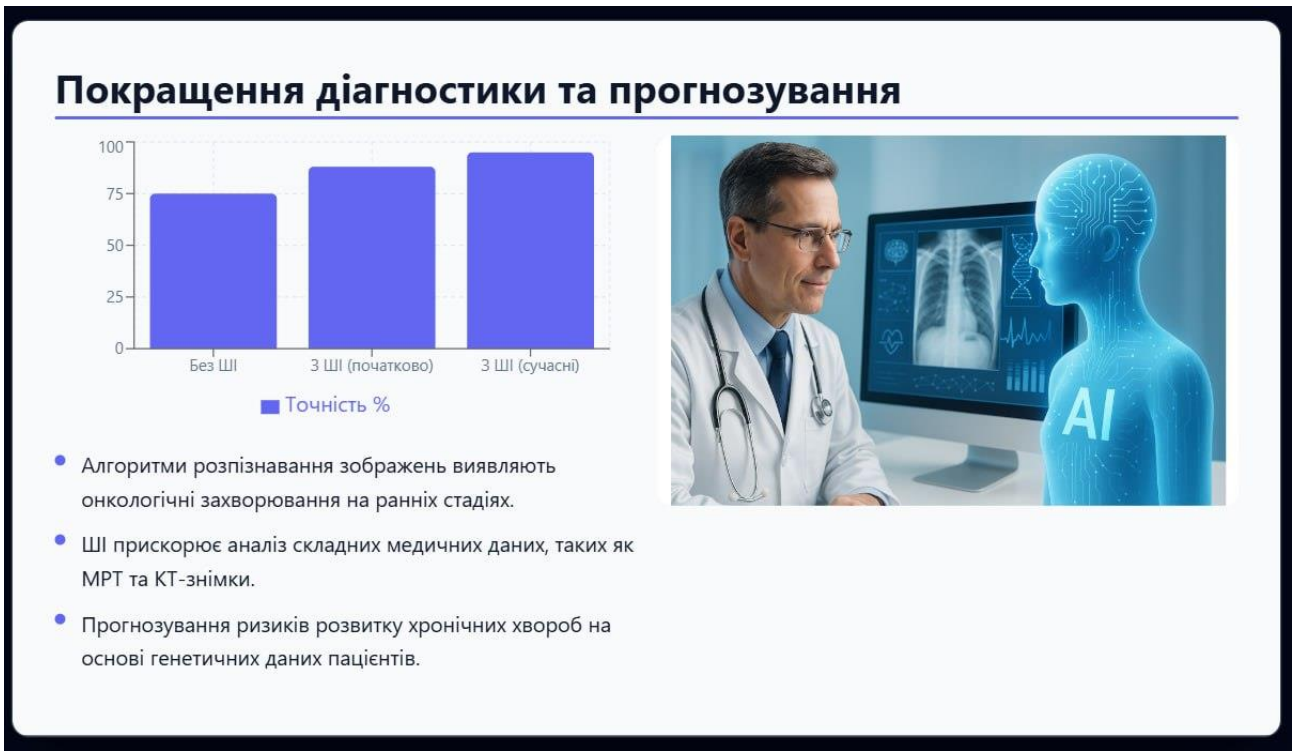


Рис. 3.10 - Приклад згенерованого слайду

Права панель редактора у режимі «Компоненти» відображає ієрархічне дерево компонентів поточного слайду. Користувач може обрати будь-який вузол дерева для редагування його пропсів, переміщувати вузли методом drag-and-drop, а також додавати нові компоненти з палітри. Інтерфейс панелі компонентів наведено на рис. 3.11.

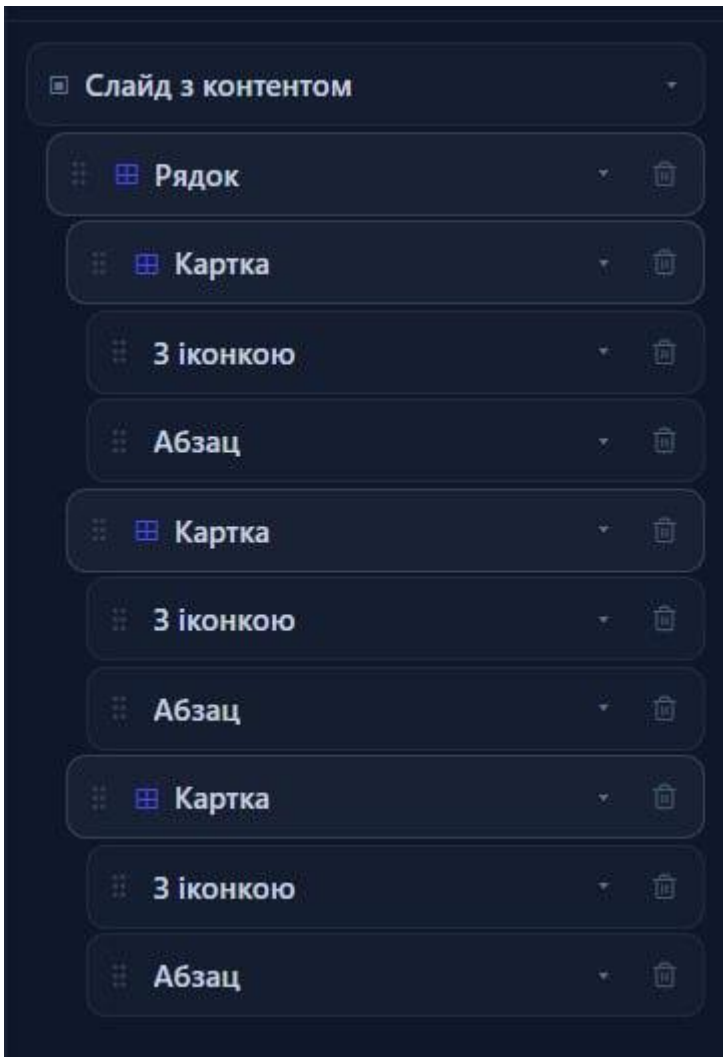


Рис. 3.11 - Панель редагування компонентів слайду

Права панель редактора у режимі «Тема» дозволяє обрати одну з п'яти тем оформлення: dark (темна), light (світла), ocean (морська), forest (лісова) та sunset (захід сонця). Зміна теми миттєво застосовується до всіх компонентів усіх слайдів без повторної генерації завдяки використанню React Context. Інтерфейс панелі вибору теми наведено на рис. 3.12.

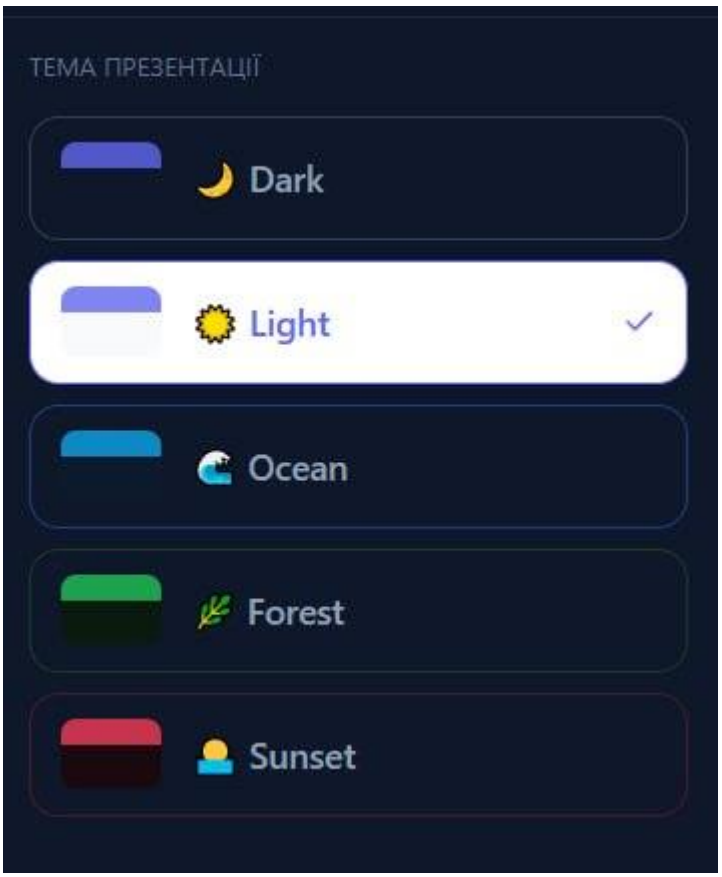


Рис. 3.12 - Панель вибору теми оформлення

Сторінка збережених презентацій (PresentationsPage) відображає галерею всіх презентацій поточного користувача. Кожна картка містить мініатюру першого слайду, назву презентації та дату останнього оновлення. Натискання на картку відкриває презентацію у редакторі. Автоматичне збереження змін відбувається з затримкою 1,5 секунди після кожного редагування. Інтерфейс сторінки збережених презентацій наведено на рис. 3.13.

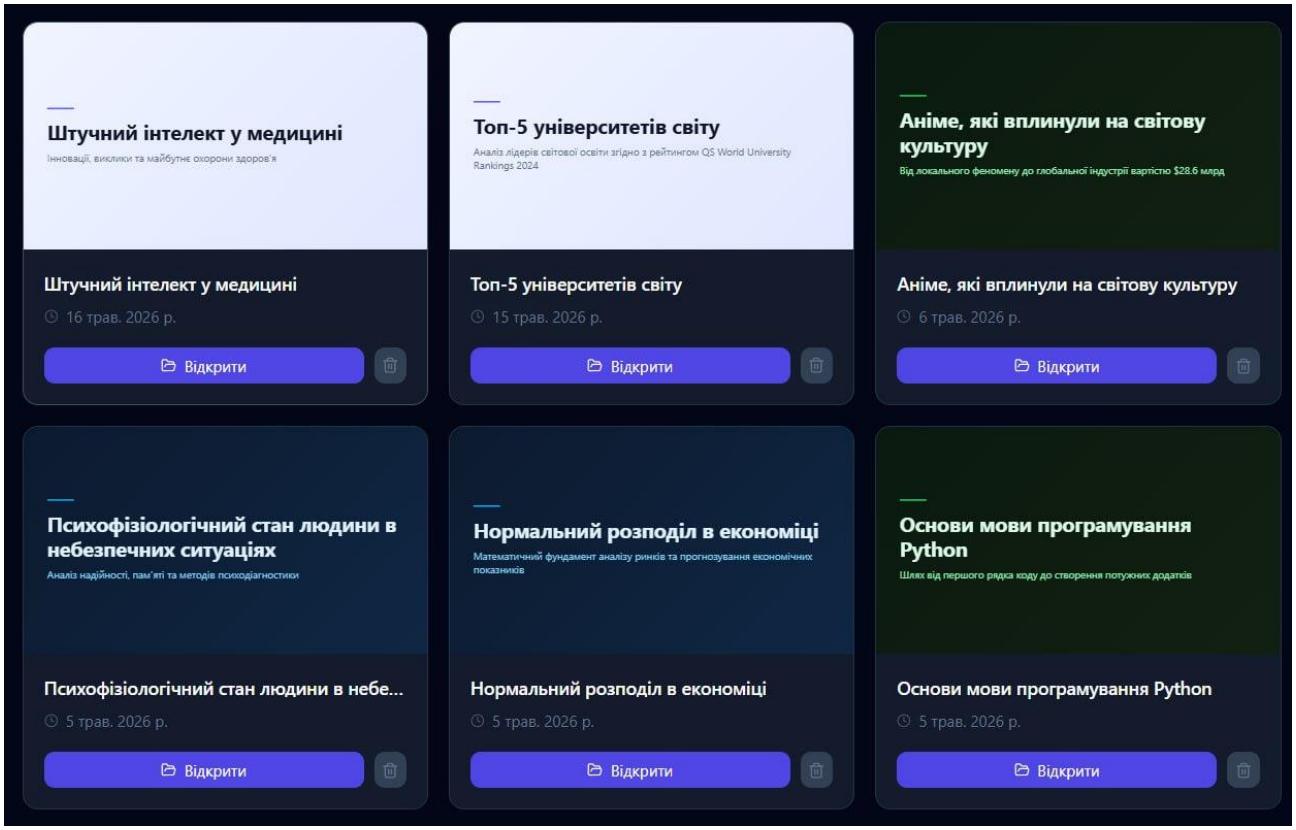


Рис. 3.13 - Сторінка збережених презентацій

Діалогове вікно конвертації у PDF відображається при натисканні кнопки експорту. Вікно показує прогрес рендерингу кожного слайду та повідомляє про завершення процесу. Після успішної конвертації користувач отримує PDF-файл для завантаження. Інтерфейс діалогового вікна конвертації наведено на рис. 3.14.

Рис. 3.14 - Діалогове вікно конвертації у PDF

Сторінка налаштувань (SettingsPage) надає доступ до конфігурації системи. Тут користувач може обрати LLM-провайдера (OpenRouter, Google Gemini або LM Studio), ввести відповідні API-ключі, налаштувати ключ Pexels API для підбору зображень та ключ Tavily API для research-агента. Усі налаштування зберігаються локально через механізм персистентності Zustand. Інтерфейс сторінки налаштувань наведено на рис. 3.15.

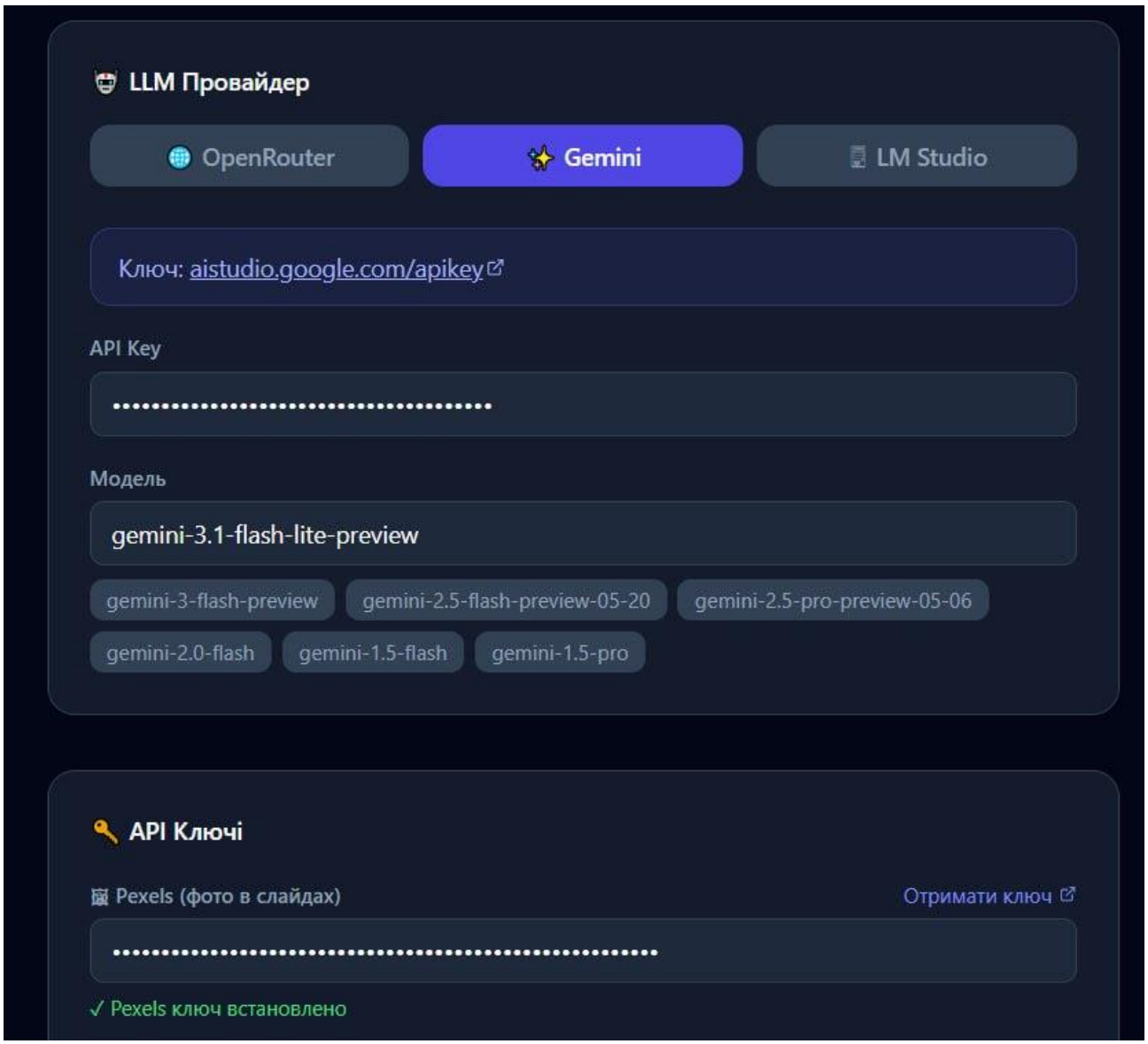


Рис. 3.15 - Сторінка налаштувань системи

Сторінка «Про проект» (AboutPage) містить інформацію про систему Slide AI: опис призначення, перелік ключових можливостей, використаний стек технологій та контактні дані розробника. Інтерфейс сторінки «Про проект» наведено на рис. 3.16.

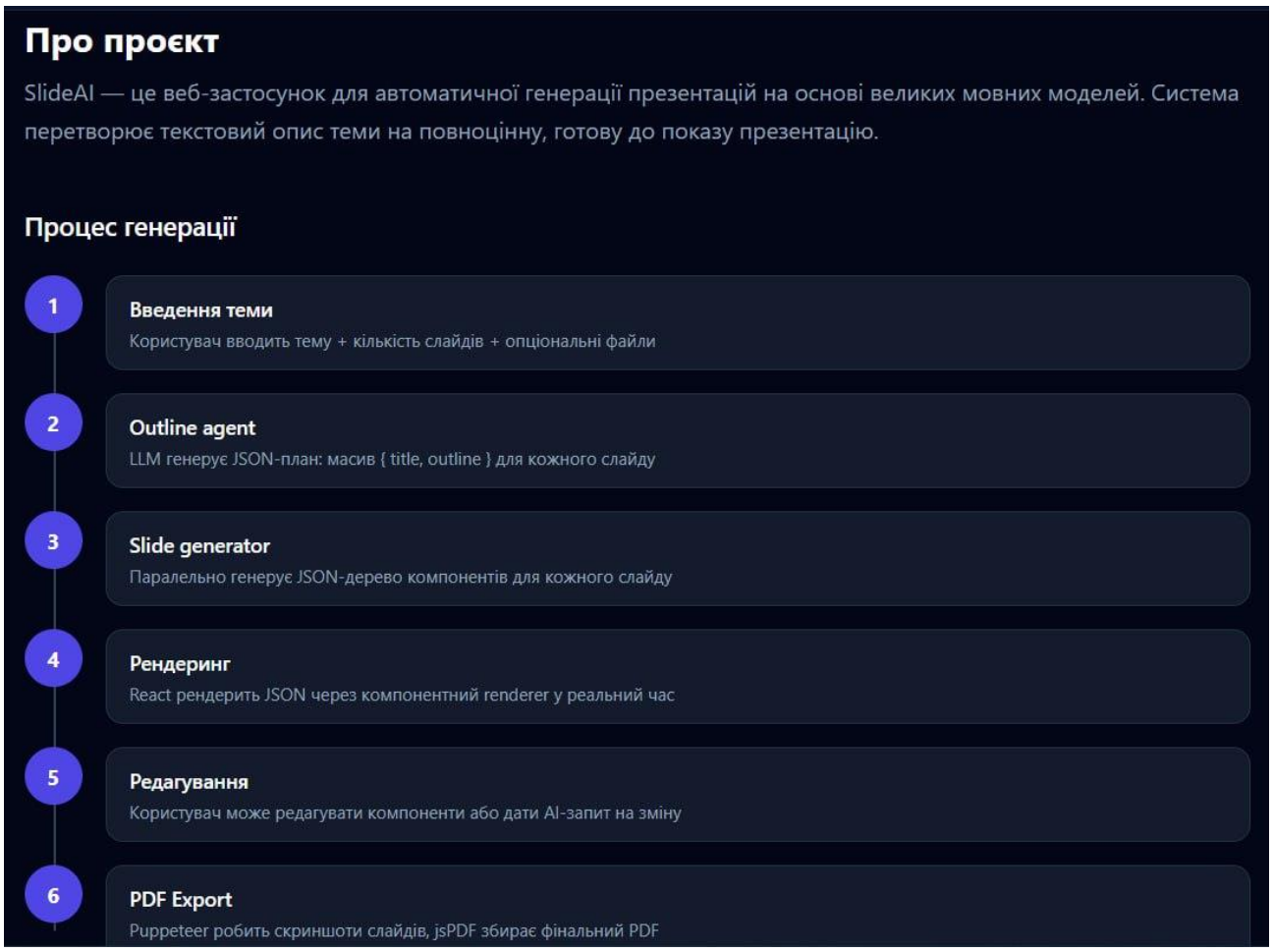


Рис. 3.16 - Сторінка «Про проект»

Висновки до розділу 3

У третьому розділі описано розробку програмного забезпечення системи Slide AI. Визначено клієнт-серверну архітектуру системи на основі React та Express.js з єдиним Node.js-процесом. Побудовано UML-діаграми класів, кооперацій, пакетів та компонентів, що відображають статичну та динамічну структуру системи.

Обґрунтовано вибір інструментарію розробки: React 19, TypeScript, Vite, Zustand, Tailwind CSS, Express.js, better-sqlite3, Puppeteer та інших бібліотек. Описано алгоритм генерації презентацій, що включає опціональний етап веб-пошуку через research-агента, стрімінгову генерацію JSON-дерева слайдів через LLM-провайдера та рекурсивний рендеринг компонентного дерева.

Проведено детальну демонстрацію інтерфейсу користувача, що охоплює всі основні сторінки та елементи системи: сторінку авторизації, головну сторінку, редактор презентацій із панелями компонентів та тем, галерею збережених презентацій, діалог конвертації у PDF, сторінку налаштувань та сторінку «Про проект».

4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ СИСТЕМИ

4.1 Тестування системи

Тестування є невід'ємним етапом розробки програмного забезпечення, що дозволяє виявити та усунути помилки до введення системи в експлуатацію. Для системи Slide AI застосовано метод ручного функціонального тестування, в рамках якого перевірялась відповідність реалізованих функцій системи сформульованим вимогам [28].

Функціональне тестування передбачає перевірку кожної функції системи шляхом введення тестових даних та порівняння отриманих результатів із очікуваними. Тестування охопило всі основні модулі системи: автентифікацію, генерацію презентацій, редактор слайдів, систему збереження, платіжний модуль та PDF-експорт.

Результати функціонального тестування основних сценаріїв використання системи наведено у таблиці в додатку Е.

За результатами функціонального тестування всі 15 тест-кейсів пройдено успішно. Система коректно обробляє як штатні сценарії використання, так і граничні випадки - введення некоректних даних, спроби несанкціонованого доступу та використання функцій без активної підписки.

Для наочної демонстрації результатів тестування генерації наведено приклад повноцінної презентації, згенерованої системою Slide AI за один запит. Презентацію створено на тему «Штучний інтелект у 2026 році» з використанням теми оформлення dark та увімкненим research-агентом. Нижче наведено зображення кожного слайду згенерованої презентації (рис. 4.1-4.9).

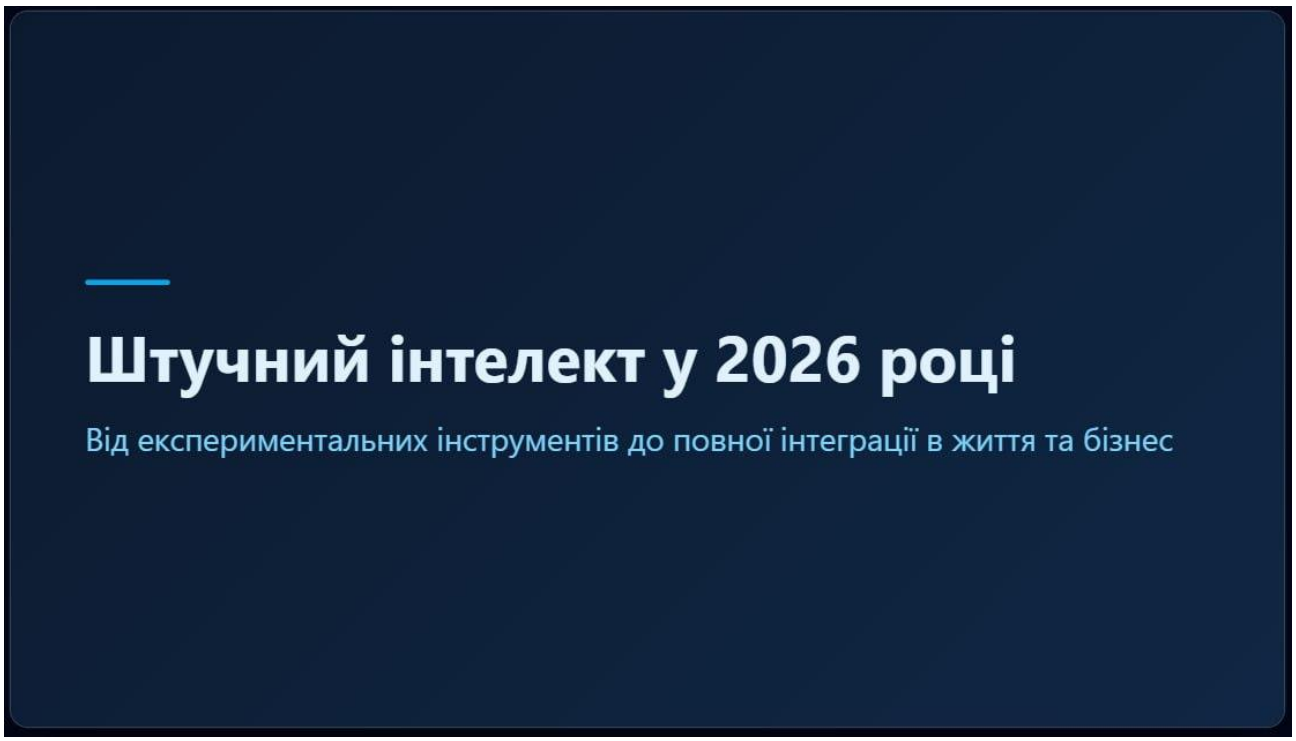


Рис. 4.1 - Слайд 1: титульний слайд презентації

Рис. 4.2 - Слайд 2: еволюція ШІ

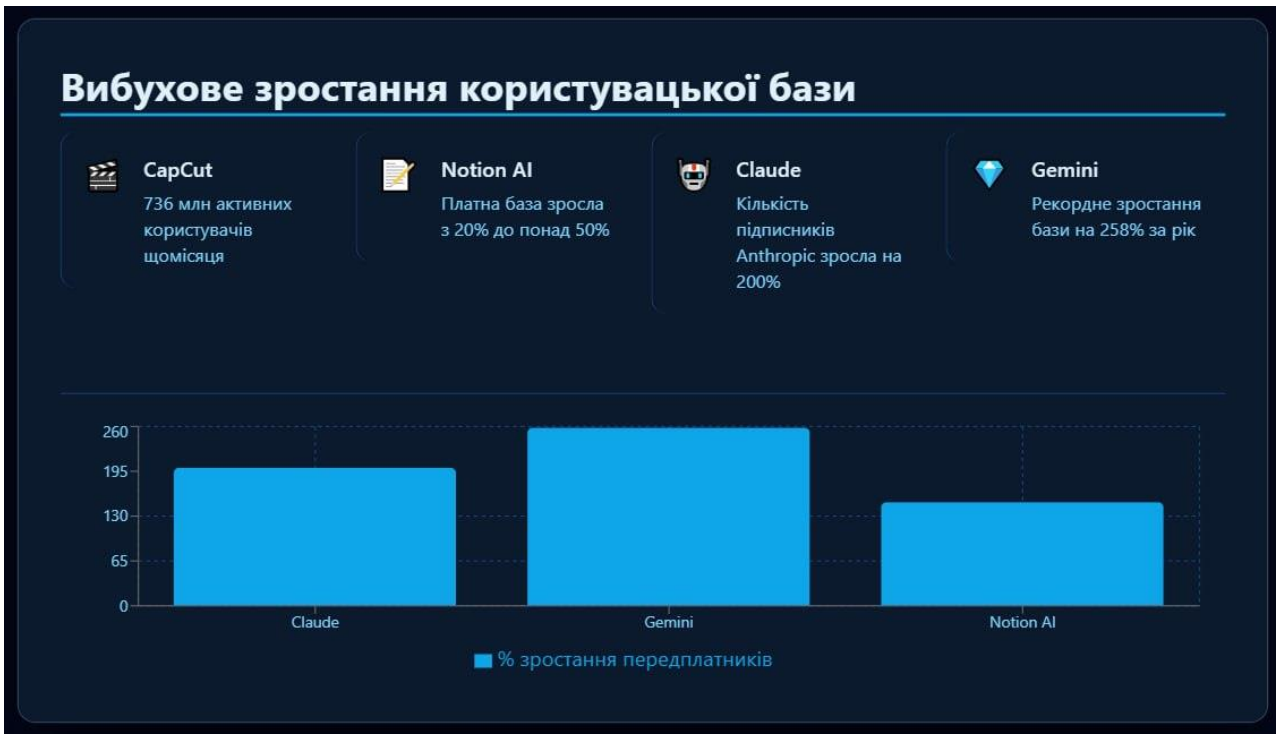


Рис. 4.3 - Слайд 3: зростання користувацької бази

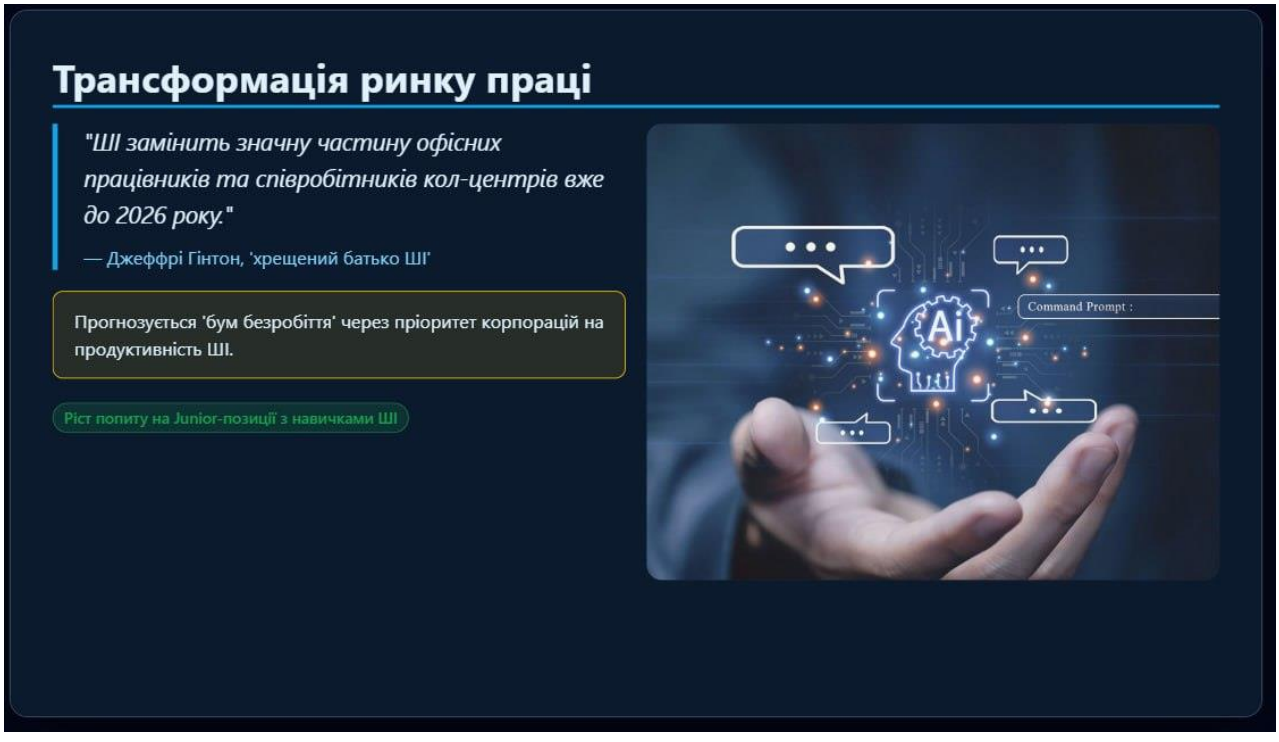


Рис. 4.4 - Слайд 4: трансформація ринку праці

Рис. 4.5 - Слайд 5: екосистеми та інтеграція сервісів

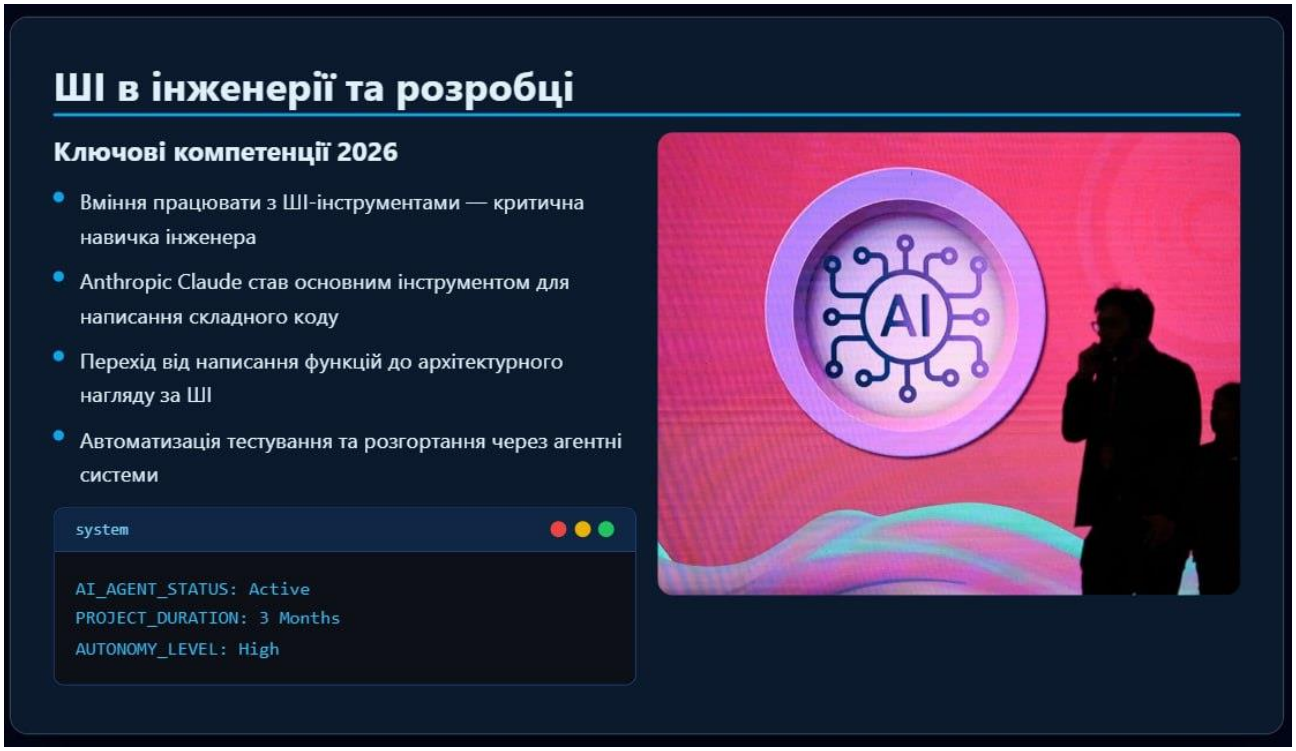


Рис. 4.6 - Слайд 6: ШІ в інженерії та розробці

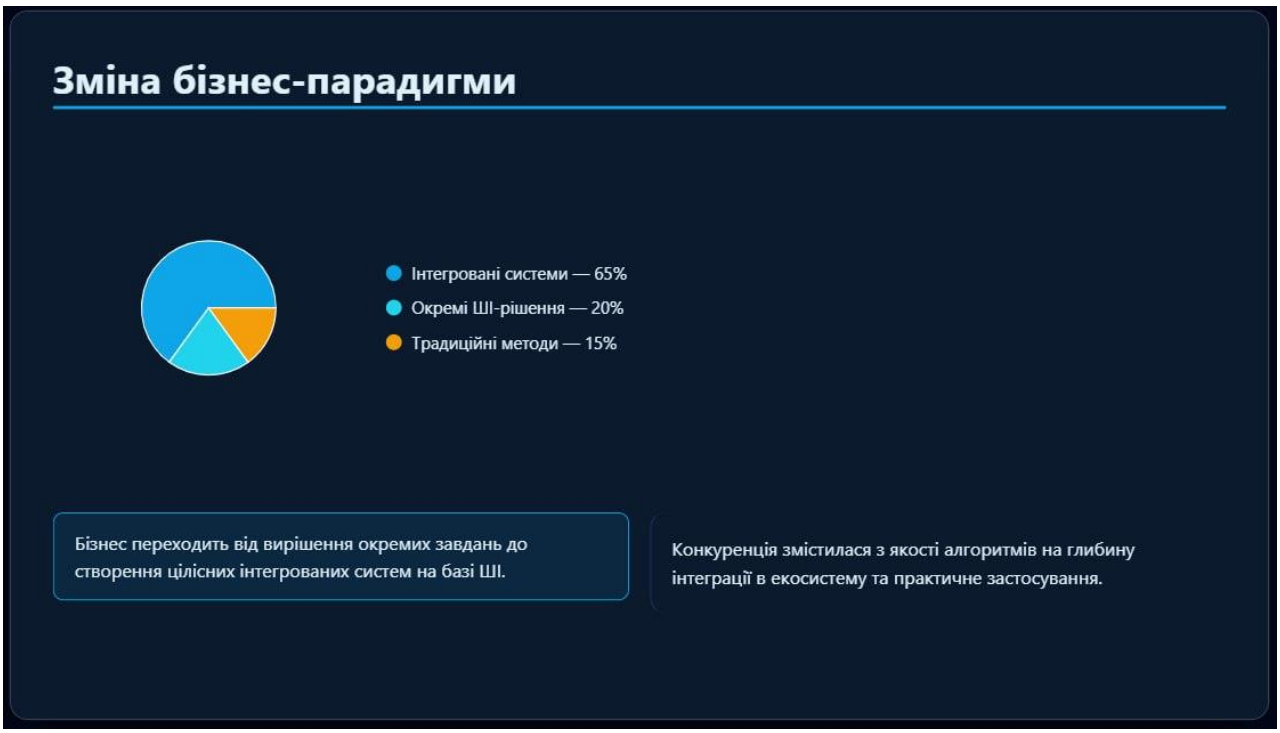


Рис. 4.7 - Слайд 7: зміна бізнес-парадигми

Рис. 4.8 - Слайд 8: щоденне життя з ШІ

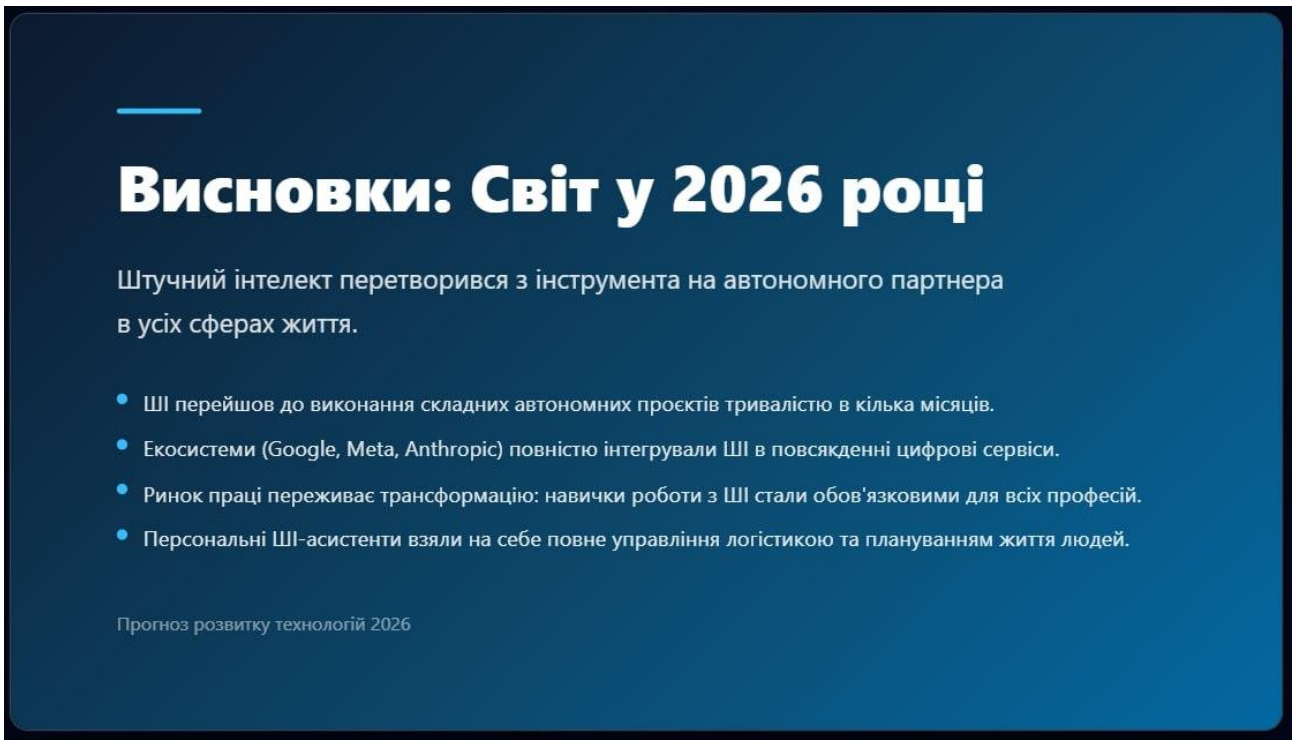


Рис. 4.9 - Слайд 9: завершальний слайд презентації

4.2 Вимоги до апаратного та програмного забезпечення

Оскільки система Slide AI розгорнута на хмарній платформі Railway та функціонує як веб-застосунок, вимоги до обладнання кінцевого користувача є мінімальними. Користувач взаємодіє із системою виключно через веб-браузер, а всі обчислення, звернення до LLM та збереження даних відбуваються на стороні сервера [29].

Вимоги до апаратного забезпечення з боку користувача наведено у табл. 4.1.

Таблиця 4.1

Вимоги до апаратного забезпечення користувача

Компонент	Мінімальні вимоги	Рекомендовані вимоги
Процесор	Двоядерний, 1.5 ГГц	Чотириядерний, 2.0 ГГц і вище
Оперативна пам'ять	4 ГБ	8 ГБ і більше
Місце на диску	500 МБ (для браузера та кешу)	1 ГБ і більше
Роздільна здатність екрана	1280x720	1920x1080 і вище
Інтернет-з'єднання	5 Мбіт/с	25 Мбіт/с і вище

Програмні вимоги з боку користувача є мінімальними: необхідна наявність одного з сучасних веб-браузерів із підтримкою JavaScript та Server-Sent Events. Підтримувані браузери наведено у табл. 4.2.

Таблиця 4.2

Підтримувані веб-браузери

Браузер	Мінімальна версія
Google Chrome	100 і вище
Mozilla Firefox	100 і вище
Microsoft Edge	100 і вище
Safari	15 і вище
Opera	85 і вище

Вимоги до серверної інфраструктури забезпечуються хмарною платформою Railway автоматично. Railway надає керовані обчислювальні ресурси та базу даних PostgreSQL як сервіс, що усуває необхідність самостійного адміністрування серверного обладнання. Серверна частина системи функціонує

в контейнеризованому середовищі з автоматичним масштабуванням ресурсів залежно від навантаження [30].

4.3 Розгортання системи на платформі Railway

Railway - хмарна платформа для розгортання веб-застосунків, що надає інфраструктуру як послугу (IaaS) з автоматичним виявленням типу застосунку, збіркою та розгортанням із репозиторію GitHub. Платформа підтримує Node.js-застосунки без додаткового налаштування контейнерів, що суттєво спрощує процес деплою [30].

Архітектура розгортання системи Slide AI на Railway складається з двох сервісів: основного веб-застосунку (Node.js / Express.js) та керованої бази даних PostgreSQL. Обидва сервіси функціонують в ізольованій приватній мережі Railway, що забезпечує безпечну взаємодію між ними без відкриття портів бази даних у зовнішню мережу.

Процес розгортання системи включає такі етапи:

- підключення репозиторію GitHub до проекту Railway - платформа автоматично відстежує зміни у головній гілці та запускає новий деплой при кожному push;
- налаштування змінних середовища (Environment Variables) - JWT_SECRET, DATABASE_URL (надається Railway автоматично при створенні PostgreSQL-сервісу), а також API-ключі для LLM-провайдерів, Pexels та платіжного сервісу;
- автоматична збірка застосунку - Railway виконує npm install та npm run build для компіляції клієнтської частини React+Vite у статичні файли;
- запуск серверного процесу командою node server/index.js - Express-сервер роздає зібраний SPA та обслуговує REST API;
- підключення до PostgreSQL здійснюється через змінну DATABASE_URL, яку Railway автоматично додає до середовища виконання; при першому старті виконується міграція схеми бази даних.

Railway надає автоматичне TLS-шифрування для всіх вхідних з'єднань через власний домен виду slideai.up.railway.app або через налаштований кастомний домен. Вся взаємодія між клієнтом та сервером відбувається виключно через HTTPS [30].

Схему розгортання системи наведено на рис. 4.10.

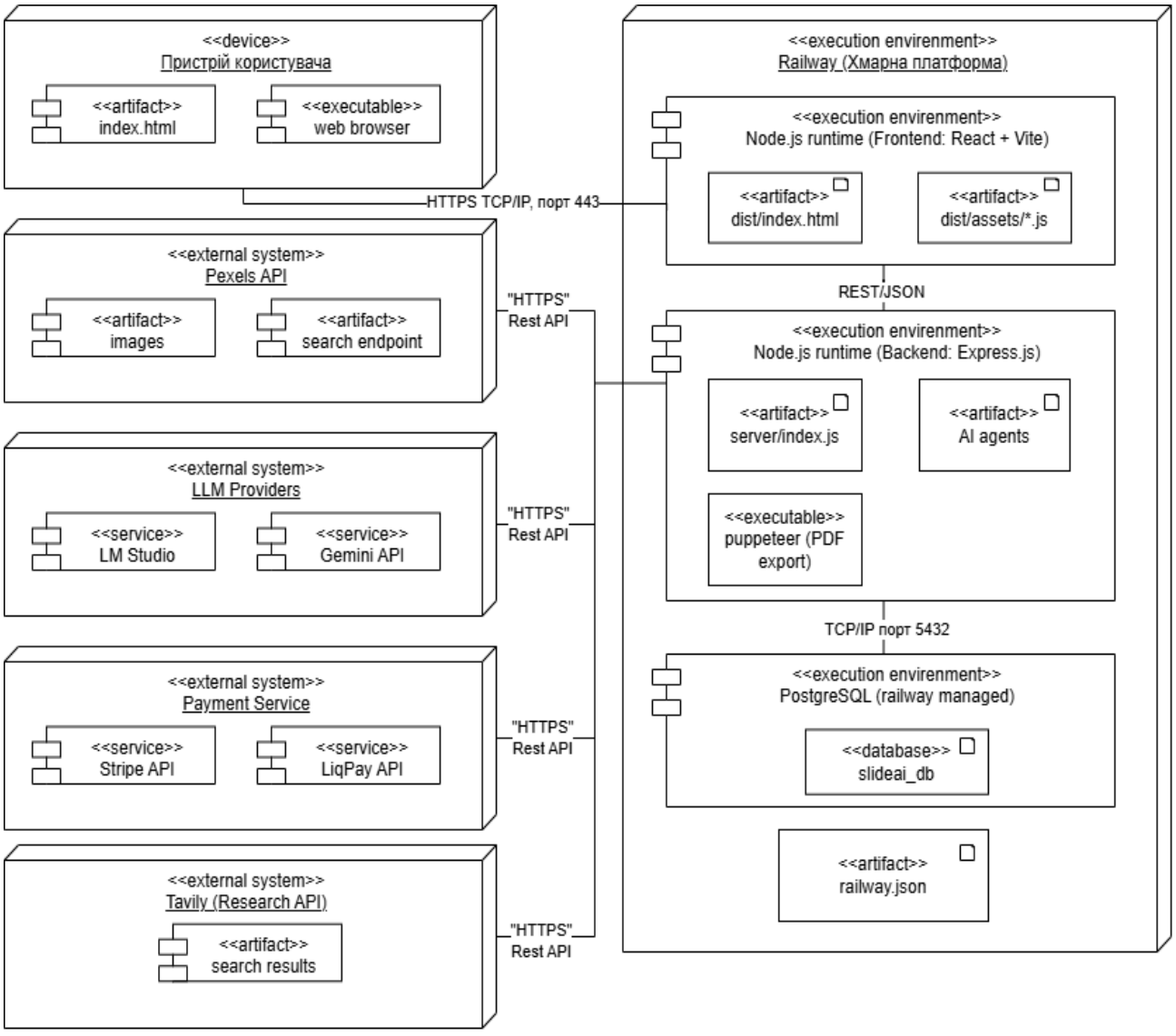


Рис. 4.10 - Діаграма розгортання системи Slide AI

4.4 Бізнес-модель та платіжна система

Система Slide AI реалізована за SaaS-моделлю (Software as a Service) з монетизацією за рахунок підписки. Суть бізнес-моделі полягає в тому, що

розробник системи придбаває корпоративний пакет доступу до LLM API за оптовою ціною та надає кінцевим користувачам доступ до цього API через власну платформу у рамках доступного тарифного плану. Така модель дозволяє користувачам генерувати презентації без необхідності самостійно реєструватися та оплачувати доступ до LLM [31].

Тарифна система передбачає два рівні доступу. Безкоштовний рівень надає обмежений доступ до базових функцій системи з можливістю тестування перед оформленням підписки. Платний рівень відкриває повний доступ до генерації презентацій через LLM API, research-агента, необмеженого збереження презентацій та пріоритетної обробки запитів.

Платіжний модуль системи інтегрується із зовнішнім платіжним провайдером (Stripe або LiqPay) через REST API. Процес оформлення підписки охоплює такі кроки:

- користувач обирає тарифний план на сторінці підписки та натискає кнопку оплати;
- клієнтська частина ініціює запит до PaymentModule серверної частини, який формує сесію оплати у платіжному провайдері;
- користувач вводить платіжні дані у захищеній формі платіжного провайдера; платіжні дані не передаються та не зберігаються на серверах системи Slide AI;
- після успішного платежу провайдер надсилає підтвердження на webhook-ендпоінт системи; статус підписки користувача оновлюється в базі даних;
- користувач отримує доступ до повного функціоналу системи без необхідності перезавантаження сторінки.

Зберігання платіжних даних та обробка транзакцій повністю делеговані платіжному провайдеру, що відповідає стандарту безпеки PCI DSS. Система Slide AI зберігає лише статус підписки користувача та дату її закінчення [32].

4.5 Склад інсталяційного пакету

Оскільки система Slide AI розгорнута на хмарній платформі Railway та доступна через веб-браузер, кінцевий користувач не здійснює інсталяцію жодного програмного забезпечення. Для початку роботи з системою достатньо відкрити адресу застосунку в підтримуваному веб-браузері та зареєструватися.

Склад серверного інсталяційного пакету (для розробників або адміністраторів, що здійснюють розгортання власного екземпляру системи) наведено у табл. 4.3.

Таблиця 4.3

Склад інсталяційного пакету системи Slide AI

Компонент	Тип	Опис
src/	Директорія	Вихідний код клієнтської частини (React + TypeScript)
server/	Директорія	Вихідний код серверної частини (Express.js)
package.json	Конфігураційний файл	Залежності та скрипти збірки проекту
vite.config.ts	Конфігураційний файл	Налаштування збирача Vite для клієнтської частини
.env	Файл середовища	Змінні середовища: JWT_SECRET, DATABASE_URL, API-ключі
public/fonts/	Директорія	Шрифти для коректного рендерингу PDF через Puppeteer
dist/ (після збірки)	Директорія	Зібрані статичні файли клієнтської частини

Для розгортання власного екземпляру системи необхідно виконати такі кроки: клонувати репозиторій, створити файл .env із необхідними змінними середовища, виконати команду npm install для встановлення залежностей, npm

run build для збірки клієнтської частини та node server/index.js для запуску серверного процесу. При розгортанні на Railway усі ці кроки виконуються автоматично [30].

Висновки до розділу 4

У четвертому розділі описано впровадження та експлуатацію системи Slide AI. Проведено функціональне тестування 15 тест-кейсів, що охоплюють усі ключові сценарії використання системи. Всі тест-кейси пройдено успішно, що підтверджує відповідність реалізованих функцій сформульованим вимогам.

Визначено вимоги до апаратного та програмного забезпечення користувача. Оскільки система розгорнута у хмарі, вимоги з боку клієнта є мінімальними та обмежуються наявністю сучасного веб-браузера та стабільного інтернет-з'єднання.

Описано процес розгортання системи на хмарній платформі Railway, що включає автоматичну збірку з репозиторію GitHub, налаштування змінних середовища, підключення керованої PostgreSQL-бази даних та автоматичне TLS-шифрування всіх з'єднань.

Описано SaaS-модель монетизації системи, засновану на підписковому доступі до LLM API. Платіжний модуль інтегровано із зовнішнім провайдером за стандартом PCI DSS, що гарантує безпечну обробку платежів без зберігання платіжних даних на серверах системи. Наведено склад інсталяційного пакету для розробників.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі вирішено актуальне науково-практичне завдання - розроблено веб-застосунок Slide AI для автоматичної генерації структурованих мультимедійних презентацій на основі природномовного опису теми з використанням технологій великих мовних моделей.

У першому розділі проведено аналіз предметної області та встановлено, що жодне з існуючих рішень - Microsoft PowerPoint, Google Slides, Gamma.app, Beautiful.ai та Tome - не забезпечує поєднання повноцінної AI-генерації, вільного вибору LLM-провайдера та локальної обробки даних. Побудовано комплекс UML-діаграм предметної області та сформульовано детальну постановку задачі з 11 функціональними вимогами до системи.

У другому розділі спроектовано інформаційне забезпечення системи. Логічна модель даних містить дві сутності - User та Presentation - зі зв'язком «один до багатьох». Прийнято рішення про зберігання масиву слайдів у форматі JSON безпосередньо у таблиці presentations, що є обґрунтованою денормалізацією і підвищує швидкодію читання. Обґрунтовано вибір PostgreSQL як СУБД для виробничого середовища та розроблено фізичну структуру бази даних з індексами та каскадним видаленням.

У третьому розділі описано розробку програмного забезпечення на основі клієнт-серверної архітектури React + Express.js. Реалізовано стрімінгову генерацію JSON-дерева слайдів через LLM-провайдера з рекурсивним рендерингом компонентів, систему п'яти тем оформлення з динамічною зміною кольорової схеми через React Context, візуальний drag-and-drop редактор компонентів слайдів, research-агента для веб-пошуку актуальних даних, підтримку файлових вкладень та PDF-експорт через Puppeteer. Обґрунтовано вибір технологічного стеку з 16 бібліотек та інструментів.

У четвертому розділі описано впровадження та експлуатацію системи. Функціональне тестування охопило 15 тест-кейсів - усі пройдено успішно.

Систему розгорнуто на хмарній платформі Railway з автоматичною збіркою з репозиторію GitHub, керованою базою даних PostgreSQL та TLS-шифруванням. Описано SaaS-модель монетизації на основі підписки з платіжним модулем, інтегрованим за стандартом PCI DSS.

Практична цінність роботи полягає в тому, що розроблена система дозволяє скоротити час підготовки структурованої мультимедійної презентації з годин до хвилин. Система підтримує кілька LLM-провайдерів (OpenRouter, Google Gemini, LM Studio), що дає користувачу можливість обирати між хмарними та локальними моделями залежно від вимог до конфіденційності та вартості.

Подальший розвиток системи передбачає впровадження payment-системи для реалізації SaaS-моделі монетизації, вдосконалення research-агента для підвищення якості пошуку актуальних даних, а також повноцінне розгортання на платформі Railway з міграцією бази даних на PostgreSQL.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Маєр Р. Є. Мультимедійне навчання. 2-ге вид. Кембридж : Cambridge University Press, 2009. 320 с.
2. Аткінсон К. Поза межами маркерів. Редмонд : Microsoft Press, 2011. 336 с.
3. Браун Т. Б. Мовні моделі як навчання з кількох прикладів. Advances in Neural Information Processing Systems, 2020. Т. 33. 1877–1901 с.
4. Pexels API. Документація [Електронний ресурс]. - URL: https://www.pexels.com/api/documentation/ (дата звернення: 27.04.2026).
5. Боммасані Р. Можливості та ризики фундаментальних моделей. Стенфорд : Stanford University, 2021. 214 с.
6. Буч Г. Уніфікована мова моделювання. Посібник користувача. Берлін : Suhrkamp Verlag, 2006. 432 с.
7. Microsoft PowerPoint - офіційний сайт продукту [Електронний ресурс]. - URL: https://www.microsoft.com/uk-ua/microsoft-365/powerpoint (дата звернення: 28.04.2026).
8. Google Slides - офіційна сторінка сервісу [Електронний ресурс]. - URL: https://workspace.google.com/products/slides/ (дата звернення: 28.04.2026).
9. Gamma.app - AI-генератор презентацій [Електронний ресурс]. - URL: https://gamma.app (дата звернення: 29.04.2026).
10. Beautiful.ai - платформа для корпоративних презентацій [Електронний ресурс]. - URL: https://www.beautiful.ai (дата звернення: 29.04.2026).
11. Tome - AI-застосунок для сторітелінгу [Електронний ресурс]. - URL: https://tomeapp.ai/ (дата звернення: 30.04.2026).
12. Node.js. Посібники [Електронний ресурс]. - URL: https://nodejs.org/docs/latest/api/ (дата звернення: 30.04.2026).
13. Коннолі Т. Бази даних. Проектування, реалізація та супровід. Берлін : Berlin Verla, 2003. 1436 с.

14. better-sqlite3. Високопродуктивний SQLite3 для Node.js [Електронний ресурс]. - URL: https://github.com/WiseLibs/better-sqlite3 (дата звернення: 01.05.2026).
15. PostgreSQL. Документація. Мова SQL [Електронний ресурс]. - URL: https://www.postgresql.org/docs/16/index.html (дата звернення: 01.05.2026).
16. SQLite. Документація. Коли використовувати SQLite [Електронний ресурс]. - URL: https://www.sqlite.org/whentouse.html (дата звернення: 02.05.2026).
17. Railway. Документація. Плагін PostgreSQL [Електронний ресурс]. - URL: https://docs.railway.app/databases/postgresql (дата звернення: 02.05.2026).
18. SQLite. Документація. CREATE TABLE [Електронний ресурс]. - URL: https://www.sqlite.org/lang_createtable.html (дата звернення: 03.05.2026).
19. Фаулер М. Шаблони архітектури корпоративних застосунків. Бостон : Addison-Wesley, 2002. 533 с.
20. Node.js. Документація. Модуль HTTP [Електронний ресурс]. - URL: https://nodejs.org/api/http.html (дата звернення: 03.05.2026).
21. Tailwind CSS. Документація. Основні концепції [Електронний ресурс]. - URL: https://tailwindcss.com/docs/utility-first (дата звернення: 05.05.2026).
22. Express.js. Документація. Маршрутизація [Електронний ресурс]. - URL: https://expressjs.com/en/guide/routing.html (дата звернення: 05.05.2026).
23. MDN Web Docs. Події, надіслані сервером [Електронний ресурс]. - URL: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events (дата звернення: 06.05.2026).
24. TypeScript. Документація. TypeScript для програмістів JavaScript [Електронний ресурс]. - URL: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html (дата звернення: 06.05.2026).
25. Vite. Документація. Чому Vite [Електронний ресурс]. - URL: https://vitejs.dev/guide/why.html (дата звернення: 08.05.2026).

26. Puppeteer. Документація. Початок роботи [Електронний ресурс]. - URL: https://pptr.dev/guides/getting-started (дата звернення: 08.05.2026).
27. React Router. Документація. Основні концепції [Електронний ресурс]. - URL: https://reactrouter.com/en/main/start/concepts (дата звернення: 10.05.2026).
28. Майерс Г. Д. Мистецтво тестування програмного забезпечення. 3-тє вид. Хобокен : Wiley, 2011. 240 с.
29. MDN Web Docs. Прогресивні веб-застосунки [Електронний ресурс]. - URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps (дата звернення: 10.05.2026).
30. Railway. Документація. Розгортання [Електронний ресурс]. - URL: https://docs.railway.app/guides/deployments (дата звернення: 13.05.2026).
31. Чоударі С. П. Масштабування платформи. Platform Thinking Labs, 2015. 270 с.
32. Stripe. Документація. Відповідність стандарту PCI [Електронний ресурс]. - URL: https://stripe.com/docs/security/guide (дата звернення: 20.05.2026).

Додаток А

Реалізація research-агента (research-agent.ts)

```
// research-agent.ts
// Research-агент: веб-пошук через Tavily API
// з каскадним fallback на Anthropic та LLM

export interface ResearchResult {
  topic:   string
  facts:   string
  sources: string[]
  images:  string[]
}

// Промпт для структурування отриманих фактів
const EXTRACT_SYSTEM = `You are a fact extractor.
Given raw web search results, extract specific facts and statistics.
Return JSON only:
{"facts":[
  {"category":"Statistics","items":["fact with number and year"]},
  {"category":"Key Players","items":["name: achievement"]},
  {"category":"Timeline",   "items":["year: event"]},
  {"category":"Comparisons","items":["A vs B: numbers"]}
],"sources":["url"]}
Rules: real numbers only. 15-20 facts total.`

// ─── Головна функція з каскадним вибором провайдера ──────────

export async function researchTopic(
  topic: string,
  llm: LLM,
  onStatus?: (msg: string) => void,
  anthropicKey?: string,
  tavilyKey?: string,
): Promise<ResearchResult> {
  onStatus?.('Researching: ' + topic + '...')

  // Пріоритет 1: Tavily (безкоштовний реальний веб-пошук)
  if (tavilyKey) {
    try {
      const r = await researchWithTavily(topic, tavilyKey, llm, onStatus)
      if (r.facts) { onStatus?.('Web search complete'); return r }
    } catch (err) {
      console.warn('[Research] Tavily failed:', err)
    }
  }

  // Пріоритет 2: Anthropic web_search (платний)
  if (anthropicKey) {
    try {
      const r = await researchWithAnthropic(topic, anthropicKey)
      if (r.facts) return r
    } catch (err) {
      console.warn('[Research] Anthropic failed:', err)
    }
  }

  // Пріоритет 3: LLM fallback (дані з тренування моделі)
  return researchWithLLM(topic, llm)
}
```

```
// ─── Реалізація Tavily-пошуку ──────────────────────────────

async function researchWithTavily(
  topic: string, tavilyKey: string,
  llm: LLM, onStatus?: (msg: string) => void,
): Promise<ResearchResult> {
  onStatus?.('Шукаю в інтернеті (Tavily)...')

  // Запит через серверний проксі (обхід CORS)
  const response = await fetch('/tavily', {
    method: 'POST',
    headers: {
      'Content-Type':  'application/json',
      'x-tavily-key':  tavilyKey,
    },
    body: JSON.stringify({
      query:               topic,
      search_depth:        'basic',
      max_results:         5,
      include_answer:      true,
      include_images:      true,
    }),
  })

  const data = await response.json() as TavilyResponse
  const results = data.results ?? []
  const sources = results.map(r => r.url)

  // Фільтруємо лише прямі URL на зображення
  const images = (data.images ?? [])
    .slice(0, 8)
    .filter(u => /\.(jpg|jpeg|png|webp)(\?|$)/i.test(u))

  // Формуємо контекст для LLM з результатів Tavily
  const webContent = [
    data.answer ? `Summary: ${data.answer}` : '',
    ...results.map(r => `[${r.title}]\n${r.content}`),
  ].filter(Boolean).join('\n\n---\n\n')

  if (!webContent.trim())
    throw new Error('No content returned from Tavily')

  onStatus?.('Обробляю результати...')

  // LLM структурує веб-дані у категоризовані факти
  const structured = await extractFactsWithLLM(topic, webContent, llm)
  return { ...structured, sources, images }
}

// ─── Парсинг відповіді LLM у структурований об'єкт ─────────

function parseJSON(text: string, topic: string): Omit<ResearchResult,
'images'> {
  try {
    const m = text.match(/\{[\s\S]*\}/)
    if (!m) return { topic, facts: '', sources: [] }
    const p = JSON.parse(m[0])
    if (!p?.facts) return { topic, facts: text.slice(0, 800), sources: [] }
    const facts = p.facts
      .map(c => '**' + c.category + ':**\n'
        + c.items.map(i => '- ' + i).join('\n'))
```

```
      .join('\n\n')
    return { topic, facts, sources: p.sources ?? [] }
  } catch {
    return { topic, facts: '', sources: [] }
  }
}

// Форматує результати research для вставки у промпт
export function formatResearchForPrompt(r: ResearchResult): string {
  if (!r.facts) return ''
  return '\n\nRESEARCHED FACTS (use in slides):\n'
    + r.facts
    + '\n\nUse these real numbers and names.'
}
```

Додаток Б

Алгоритм генерації слайдів (presentation-agent.ts)

```
// presentation-agent.ts
// Головний агент генерації презентацій.
// Генерує ВСІ слайди за ОДИН LLM-запит у режимі стрімінгу.

import type { LLM } from '../llm/llm'
import { researchTopic, formatResearchForPrompt } from './research-agent'
import type { Attachment } from '../../utils/attachments'

// ─── Системний промпт для LLM ──────────────────────────────

const SYSTEM_PROMPT = `
You generate a complete presentation as a JSON array of slides.
Stream the entire array in ONE response.
Output ONLY valid JSON - no markdown fences, no explanation.

OUTPUT FORMAT:
[
  { slide JSON },
  { slide JSON },
  ...
]

Each slide JSON uses this node format:
  Text node:      {"t": "text"}
  Component node: {"c": "ComponentName", "p": {props}, "ch": [children]}

AVAILABLE COMPONENTS:
  TitleSlide      p: title, subtitle?    - root of slide 1 ONLY
  DefaultSlide    p: title               - root of all other slides
  Row      p: cols(int), gap(int)        - grid columns
  Stack    p: gap(int)                   - vertical stack
  Card     p: accent?                    - framed box
  BulletList    p: items(string[])
  NumberedList  p: items(string[])
  IconItem      p: icon(emoji), title, description?
  BarChart     p: data([{name,v}]), bars([{key,label?}]), height?
  PieChart     p: data([{name,value}]), height?
  FlowDiagram  p: steps([{label,description?}])
  SlideImage   p: query(English phrase), fit?, rounded?, height?
  ClosingSlide p: title, subtitle?, points(string[])?, author?

CONTENT RULES:
- Slide 1: ALWAYS TitleSlide
- LAST slide: ALWAYS ClosingSlide with real conclusions
- Match the user's language exactly
- No placeholder text - every number and fact must be real

MANDATORY LAYOUT ROTATION (use each pattern roughly once):
PATTERN A - STATS ROW: Row cols=3, each col: Card with Stat
PATTERN B - COMPARISON: Row cols=2 → Card A + Card B with BulletList
PATTERN C - ICON GRID: Row cols=3 → Card with IconItem
PATTERN D - PIE CHART: PieChart + Row cols=2 with Alert and Quote
PATTERN E - BAR CHART FOCUS: Chart 60% + BulletList 3 items below
PATTERN F - FLOW PROCESS: FlowDiagram 4-5 steps + Paragraphs
PATTERN G - IMAGE HERO: Row cols=2 → Stack(text) + SlideImage
`
```

```
// ── Інкрементний парсинг JSON-масиву під час стрімінгу ──────────────────

function extractSlides(text: string): SlideItem[] {
  const slides: SlideItem[] = []

  const startIdx = text.indexOf('[')
  if (startIdx === -1) return slides

  const content = text.slice(startIdx + 1)

  // Рахуємо фігурні дужки щоб знайти межі кожного об'єкта
  let depth = 0
  let start = -1

  for (let i = 0; i < content.length; i++) {
    const ch = content[i]
    if (ch === '{') {
      if (depth === 0) start = i
      depth++
    } else if (ch === '}') {
      depth--
      if (depth === 0 && start !== -1) {
        const candidate = content.slice(start, i + 1)
        try {
          const parsed = JSON.parse(candidate)
          if (parsed?.c) {
            slides.push({
              title: parsed.p?.title ?? `Slide ${slides.length + 1}`,
              outline: '',
              json: candidate,
            })
          }
        } catch { /* неповний JSON під час стрімінгу - пропускаємо */ }
        start = -1
      }
    }
  }
  return slides
}

// ── Головна функція генерації ──────────────────────────────────────────

export async function generatePresentation({
  llm, message, slideCount = 6,
  useResearch = false, attachments = [],
  onStatus, onSlideReady, onProgress,
}: PresentationAgentOptions): Promise<SlideItem[]> {

  const { anthropicKey, tavilyKey } = useSettingsStore.getState()

  // Крок 1: Research - збір актуальних даних з інтернету
  let research: ResearchResult | null = null
  if (useResearch) {
    research = await researchTopic(
      message, llm, onStatus, anthropicKey, tavilyKey
    )
  }

  // Крок 2: Обробка вкладень (PDF та зображення)
  let attachmentContext = ''
  for (const att of attachments) {
    if (att.type === 'pdf') {
```

```
      const MAX = 7000
      const text = att.content.length > MAX
        ? att.content.slice(0, MAX) + '...[скорочено]'
        : att.content
      attachmentContext += `\n\n=== ДОКУМЕНТ: ${att.name} ===\n${text}`
    } else if (att.type === 'image' && anthropicKey) {
      onStatus?.(`Аналіз зображення ${att.name}...`)
      const description = await analyzeImageWithClaude(
        att.content, att.mediaType!, anthropicKey, att.name
      )
      attachmentContext += `\n\n=== ЗОБРАЖЕННЯ: ${att.name}
===\n${description}`
    }
  }

  // Крок 3: Формування фінального промпту
  const researchNote = research ? formatResearchForPrompt(research) : ''
  const userPrompt = `Topic: ${message.trim()}
Generate EXACTLY ${slideCount} slides.
- Slide 1: TitleSlide
- Slides 2 to ${slideCount - 1}: content slides with varied layouts
- Slide ${slideCount}: ClosingSlide with real conclusions
${researchNote}${attachmentContext}`

  const messages = [
    { role: 'system' as const, content: SYSTEM_PROMPT },
    { role: 'user'   as const, content: userPrompt },
  ]

  let allSlides: SlideItem[] = []
  let lastSlideCount = 0

  const emitSlides = (slides: SlideItem[]) => {
    for (let i = 0; i < slides.length; i++) {
      if (slides[i]!.json !== allSlides[i]?.json) {
        onSlideReady(i, slides[i]!)   // відображаємо слайд одразу
      }
    }
    if (slides.length > lastSlideCount) {
      lastSlideCount = slides.length
      onProgress(slides.length)
    }
    allSlides = slides
  }

  // Крок 4: Стрімінг - слайди з'являються по мірі генерації
  await llm.stream(messages, {
    onChunk: (_delta, accumulated) => {
      const slides = extractSlides(accumulated)
      if (slides.length > 0) emitSlides(slides)
    },
    onComplete: (full) => {
      const slides = extractSlides(full)
      if (slides.length > 0) emitSlides(slides)
    },
    onError: (err) => { throw err },
  })

  if (allSlides.length === 0)
    throw new Error('No slides generated. Check LLM connection.')

  return allSlides
}
```

Додаток В

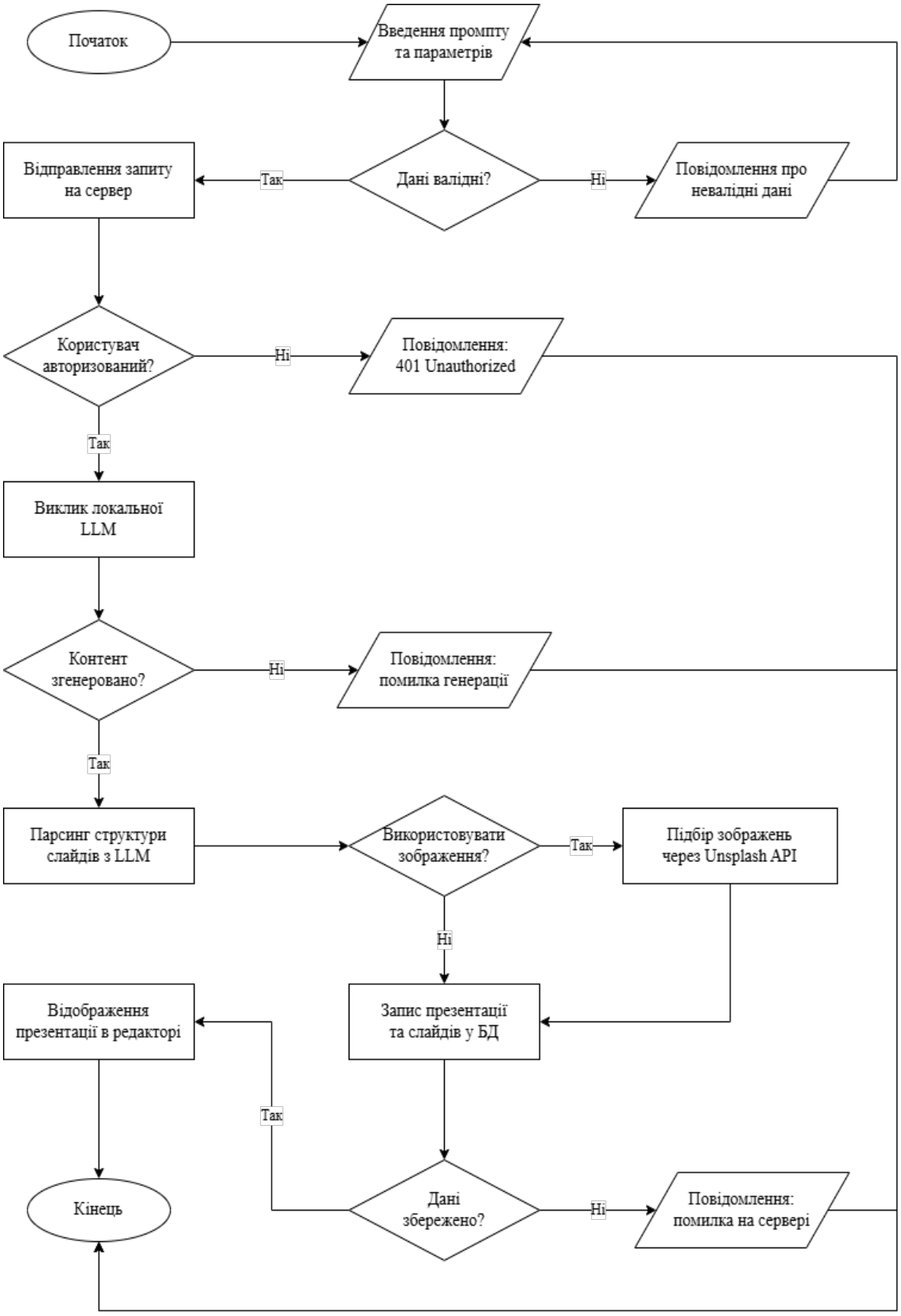
Рекурсивний рендер JSON-дерева слайдів (renderer.tsx)

```
// renderer.tsx
// Рекурсивний рендер JSON-дерева компонентів у React-елементи

import React from 'react'
import * as C from './index'

// Формат вузла JSON-дерева слайду:
//   Компонент: { "c": "ComponentName", "p": {props}, "ch": [children] }
//   Текст:     { "t": "plain text string" }
export interface SlideNode {
  c?:  string                    // назва компонента
  p?:  Record<string, unknown>   // пропси
  ch?: SlideNode[]               // дочірні вузли
  t?:  string                    // текстовий вузол
}

// Реєстр усіх доступних компонентів слайдів
const COMPONENTS: Record<string, React.ComponentType<any>> = {
  TitleSlide:   C.TitleSlide,
  DefaultSlide: C.DefaultSlide,
  SectionSlide: C.SectionSlide,
  Row:          C.Row,
  Stack:        C.Stack,
  Card:         C.Card,
  Header:       C.Header,
  Paragraph:    C.Paragraph,
  Divider:      C.Divider,
  Badge:        C.Badge,
  Quote:        C.Quote,
  Alert:        C.Alert,
  BulletList:   C.BulletList,
  NumberedList: C.NumberedList,
  IconItem:     C.IconItem,
  BarChart:     C.BarChart,
  LineChart:    C.LineChart,
  PieChart:     C.PieChart,
  FlowDiagram:  C.FlowDiagram,
  SlideImage:   C.SlideImage,
  ClosingSlide: C.ClosingSlide,
}

// Рекурсивна функція рендерингу вузла
export function renderNode(
  node: SlideNode,
  key?: number
): React.ReactNode {
  if (!node) return null

  // Текстовий вузол - повертаємо рядок напряму
  if (node.t !== undefined) return node.t

  if (!node.c) return null

  const Component = COMPONENTS[node.c]
  if (!Component) {
    console.warn('[renderer] Unknown component:', node.c)
    return null
  }
```

```
  // Рекурсивно рендеримо дочірні вузли
  const children = node.ch?.map(
    (child, i) => renderNode(child, i)
  )

  // Створюємо React-елемент з пропсами та дочірніми вузлами
  return React.createElement(
    Component,
    { key, ...node.p },
    children?.length ? children : undefined,
  )
}

// Парсинг JSON-рядка слайду з обробкою помилок
export function tryParseSlideJSON(raw: string): SlideNode | null {
  // Видаляємо markdown-огорожі якщо є
  const cleaned = raw
    .replace(/^```[\w]*\n?/m, '')
    .replace(/\n?```$/m, '')
    .trim()

  try {
    const parsed = JSON.parse(cleaned)
    if (parsed?.c) return parsed as SlideNode
  } catch { /* */ }

  // Спроба витягти JSON з тексту
  const match = cleaned.match(/\{[\s\S]*"c"[\s\S]*\}/)
  if (match) {
    try {
      const parsed = JSON.parse(match[0])
      if (parsed?.c) return parsed as SlideNode
    } catch { /* */ }
  }
  return null
}
```

Додаток Д

Блок-схема алгоритму генерації презентації



Додаток Е

Результати функціонального тестування системи Slide AI

№	Тест-кейс	Очікуваний результат	Фактичний результат	Статус
1	Реєстрація нового користувача з коректними даними	Обліковий запис створено, JWT-токен видано, перенаправлення на головну сторінку	Відповідає очікуваному	Пройдено
2	Реєстрація з вже існуючою електронною адресою	Повідомлення про помилку «Email вже використовується»	Відповідає очікуваному	Пройдено
3	Вхід до системи з коректними обліковими даними	Успішна автентифікація, токен збережено в localStorage	Відповідає очікуваному	Пройдено
4	Вхід з некоректним паролем	Повідомлення про помилку «Невірний email або пароль»	Відповідає очікуваному	Пройдено
5	Генерація презентації з 5 слайдів без research-агента	Презентація з 5 слайдів згенерована та відображена в редакторі	Відповідає очікуваному	Пройдено
6	Генерація презентації з увімкненим research-агентом	Виконано веб-пошук, контекст додано до промпту, слайди містять актуальні дані	Відповідає очікуваному	Пройдено
7	Завантаження PDF-файлу як вкладення перед генерацією	Вміст PDF витягнуто та включено до контексту запиту	Відповідає очікуваному	Пройдено
8	Редагування вмісту слайду через drag-and-drop у SlideEditor	Вузол переміщено, презентацію автоматично збережено	Відповідає очікуваному	Пройдено
9	AI-редагування окремого слайду через текстову інструкцію	Слайд оновлено відповідно до інструкції без зміни інших слайдів	Відповідає очікуваному	Пройдено
10	Зміна теми оформлення презентації	Кольорова схема всіх компонентів усіх слайдів змінилась миттєво	Відповідає очікуваному	Пройдено

№	Тест-кейс	Очікуваний результат	Фактичний результат	Статус
11	Збереження та завантаження презентації	Презентація збережена в БД та коректно відновлена при повторному відкритті	Відповідає очікуваному	Пройдено
12	Експорт презентації у PDF	PDF-файл сформовано з коректним відображенням усіх слайдів та стилів	Відповідає очікуваному	Пройдено
13	Оформлення підписки користувачем	Платіж оброблено, тариф активовано, доступ до LLM API надано	Відповідає очікуваному	Пройдено
14	Спроба генерації без активної підписки	Повідомлення про необхідність оформити підписку для доступу до генерації	Відповідає очікуваному	Пройдено
15	Доступ до захищених маршрутів без JWT-токена	Перенаправлення на сторінку авторизації	Відповідає очікуваному	Пройдено

Додаток Ж

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Котельницький Денис Сергійович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота (проєкт) на тему «Програмне забезпечення створення презентації з використання технологій штучного інтелекту» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що:
    - [ ] інструменти генеративного ШІ не використовувалися.
    - [✓] інструменти ШІ Claude та ChatGPT були використані для:
        - [✓] перекладу іншомовних джерел;
        - [✓] стилістичного редагування та перевірки граматики;
        - [✓] допомоги у написанні/відлагодженні програмного коду;
        - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___» _____ 2026 р. _____ **Денис КОТЕЛЬНИЦЬКИЙ**
(підпис)