

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Платформа для надання юридичної допомоги з
штучного інтелекту»**

Виконав студент групи ІПЗ-21-3
БІЛЕЦЬКИЙ Євгеній Володимирович

Керівник роботи:
САВІЦЬКИЙ Роман Святославович

Рецензент:
КУШНІР Надія Олександрівна

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ЗАТВЕРДЖУЮ
Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«14» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: Білецький Євгеній Володимирович

Керівник роботи: Савіцький Роман Святославович

Тема роботи: «Платформа для надання юридичної допомоги з штучного інтелекту», затверджена Наказом закладу вищої освіти від 14.02.2025р., №61/с
Об'єктом дослідження є використання інформаційних технологій для автоматизації процесів надання юридичної допомоги та консультування.

Предметом дослідження є застосування сучасних технологій розробки вебзастосунків для створення платформи юридичної допомоги з інтеграцією функцій штучного інтелекту.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Савіцький Р. С.	20.02.2025	20.02.2025
2	Савіцький Р. С.	20.02.2025	20.02.2025
3	Савіцький Р. С.	20.02.2025	20.02.2025

РЕФЕРАТ

Кваліфікаційна робота освітнього ступеня «бакалавр» складається з веборієнтованої платформи для надання юридичної допомоги із застосуванням штучного інтелекту та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи містить 88 сторінок, 42 ілюстрації та 44 таблиці. Загальний обсяг – 121 сторінок.

Метою роботи є розробка платформи, яка забезпечує автоматизацію процесів надання юридичних послуг, інтеграцію чат-бота на базі ШІ, організацію відеоконсультацій із юристами, а також ефективне адміністрування користувачів, підписок і юридичних шаблонів. Система включає модулі для генерації та аналізу документів, управління профілями користувачів і юристів, адміністрування ботів та шаблонів документів, організації консультацій, ведення статистики та відгуків.

У роботі поставлені основні завдання на розробку платформи, проведено порівняльний аналіз аналогічних рішень, визначено конкурентні переваги системи. Обрано клієнт-серверну архітектуру, де клієнтська частина реалізована за допомогою фреймворку Next.js, а серверна частина — на базі NestJS із використанням MongoDB і ChromaDB. Описано основні сценарії використання системи, структуру бази даних, діаграми взаємодії та ключові алгоритми роботи модулів.

КЛЮЧОВІ СЛОВА: ЮРИДИЧНА ДОПОМОГА, АВТОМАТИЗАЦІЯ, ШТУЧНИЙ ІНТЕЛЕКТ, КОНСУЛЬТАЦІЯ, ПЛАТФОРМА, ДОКУМЕНТИ, ВЕБЗАСТОСУНОК, АДМІНІСТРУВАННЯ.

					ІПЗ.КР.Б – 121 – 25 – ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Білецький С.В.			Платформа для надання юридичної допомоги з штучного інтелекту		
Керівник		Савіцький Р.С.					
Рецензент		Кушнір Н.О.					
Зав. кафедри		Вакалюк Т.А.					
					Літ.	Арк.	Аркушів
						3	121
					Житомирська політехніка, група ІПЗ-21-3		

ABSTRACT

The bachelor's qualification thesis consists of a web-based platform for providing legal assistance using artificial intelligence and an explanatory note. The explanatory note to the qualification thesis contains 88 pages, 42 illustrations, and 44 tables. The total volume is 121 pages.

The aim of the thesis is to develop a modern platform that automates the processes of providing legal services, integrates an AI-based chatbot, organizes video consultations with lawyers, and enables efficient administration of users, subscriptions, and legal templates. The system includes modules for document generation and analysis, management of user and lawyer profiles, administration of bots and document templates, organization of consultations, and collection of statistics and feedback.

The main objectives of the thesis are to design and develop the platform, conduct a comparative analysis of similar solutions, and determine the competitive advantages of the system. A client-server architecture was chosen, with the client side implemented using the Next.js framework and the server side based on NestJS with MongoDB and ChromaDB. The main usage scenarios of the system, the database structure, interaction diagrams, and key algorithms of the modules are described.

KEYWORDS: LEGAL ASSISTANCE, AUTOMATION, ARTIFICIAL INTELLIGENCE, CONSULTATION, PLATFORM, DOCUMENTS, WEB APPLICATION, ADMINISTRATION.

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				4
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

РЕФЕРАТ	0
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ.....	10
1.1 Постановка задачі	10
1.2 Аналіз аналогів програмного продукту	12
1.3 Вибір архітектури платформи для надання юридичної допомоги з ІІІ ...	19
1.4 Обґрунтування вибору інструментальних засобів	22
Висновки до першого розділу	30
РОЗДІЛ 2. ПРОЄКТУВАННЯ ПЛАТФОРМИ ДЛЯ НАДАННЯ ЮРИДИЧНОЇ ДОПОМОГИ З ШТУЧНИМ ІНТЕЛЕКТОМ	31
2.1 Визначення варіантів використання та структури платформи	31
2.2 Розробка бази даних	42
2.3 Проєктування та реалізація алгоритмів роботи платформи	49
2.4 Реалізація платформи для надання юридичної допомоги з штучним інтелектом.....	53
Висновки до розділу 2	63
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПЛАТФОРМОЮ ДЛЯ НАДАННЯ ЮРИДИЧНОЇ ДОПОМОГИ З ШТУЧНИМ ІНТЕЛЕКТОМ.....	64
3.1 Порядок встановлення та налаштування параметрів додатку	64
3.2 Структура інтерфейсу та порядок взаємодії з платформою для надання юридичної допомоги з штучним інтелектом	68
3.3 Тестування роботи програмного продукту	83
Висновки до третього розділу	87
ВИСНОВКИ	88
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	89
ДОДАТКИ	94

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				5
Змн.	Арк.	№ докум.	Підпис	Дата		

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

СУБД — система управління бази даних.

TS — TypeScript.

SPA — Single-page application.

RAG — Retrieval-augmented generation.

БД — база даних.

API — Application Programming Interface.

AI — Artificial Intelligence.

ІІІ — штучний інтелект.

CSS — Cascading Style Sheets.

JSON — JavaScript Object Notation.

BSON — Binary JSON.

gRPC — gRPC Remote Procedure Call.

SDK — Software Development Kit.

REST — Representational State Transfer.

ML — Machine Learning.

HTTP — Hypertext Transfer Protocol.

URL — Uniform Resource Locator.

CORS — Cross-Origin Resource Sharing.

SSE — Server-Sent Events.

SSG — Static Site Generation.

Blob — Binary Large Object.

CI/CD — Continuous Integration / Continuous Delivery.

БММ — велика мовна модель.

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Темпи цифровізації та впровадження інформаційних технологій стрімко зростають, охоплюючи все більше сфер людської діяльності. Однією з ключових галузей, яка потребує модернізації та адаптації до викликів часу, є юридична допомога. Традиційний підхід до надання юридичних послуг часто супроводжується значними витратами часу, ресурсів і обмеженою доступністю для населення. Водночас стрімкий розвиток технологій штучного інтелекту відкриває нові можливості для автоматизації правових процесів, що дозволяє значно підвищити ефективність сфери.

Актуальність теми полягає в необхідності створення інноваційних рішень, які забезпечують швидкий доступ до юридичних послуг широкому колу громадян. Використання ШІ дозволяє вирішити низку проблем, пов'язаних із тривалими консультаціями, недостатньою обізнаністю громадян у правових питаннях, а також високими витратами на послуги юристів. Таким чином, розробка платформи для надання юридичної допомоги з використанням ШІ є не лише технологічним, а й соціально значущим завданням, оскільки вона сприяє демократизації доступу до правосуддя та підвищенню юридичної грамотності населення.

Метою кваліфікаційної роботи є створити сучасну веборієнтовану платформу, що інтегрує технології штучного інтелекту для автоматизації процесів юридичної допомоги. Платформа має забезпечити зручний інтерфейс для користувачів, надавати можливість отримання відповідей на прості правові питання, а також організовувати інтеграцію з реальними юристами для складніших завдань. Основною метою є оптимізація надання юридичних послуг і зменшення бар'єрів для доступу до них.

Практичними завданнями кваліфікаційної роботи є:

— проведення аналізу предметної області юридичних послуг з використанням цифрових технологій та штучного інтелекту. Вивчення існуючих програмних рішень і сервісів для онлайн-консультування, формулювання ключових вимог користувачів до майбутньої платформи;

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

- визначення та обґрунтування добірки інструментальних засобів і технологій для реалізації клієнської та серверної частини, сховища даних та компонентів штучного інтелекту для RAG-моделі;
- розроблення повного переліку функціональних і нефункціональних вимог, а також сценаріїв використання платформи залежно від ролі користувача — клієнт, юрист, адміністратор;
- проектування структури бази даних для зберігання інформації про облікові записи користувачів, профілі клієнтів і юристів, історію консультацій, шаблони документів, підписки, дані о ботах;
- розробка серверної частини сайту з використанням NestJS із реалізацією аутентифікації й авторизації, REST API для обміну повідомленнями, модулів роботи з RAG-движком для створення запитів до OpenAI та пошуку у векторній БД, а також інтеграція з платіжною системою Stripe для оплати підписок.
- розробка модуля відеочату для консультацій із реальним юристом у режимі реального часу з використанням WebRTC, забезпечення механізму встановлення з'єднання, передачі відео та аудіо;
- реалізація та інтеграція модуля генерації документів за шаблонами з організацією системи зберігання шаблонів договорів, побудовою логіки динамічної підстановки даних із використанням RAG-підходу, а також забезпечення можливості завантаження готового документа в PDF чи DOCX;
- створення інтерфейсу, що пропонує користувачеві інтуїтивно зрозумілу навігацію по платформі;
- проведення юніт-тестування серверної логіки та ручного тестування функціоналу клієнської частини з фіксацією й усуненням виявлених дефектів, підтвердження відповідності платформи заявленим вимогам;

Об'єктом дослідження є процеси надання юридичних послуг із використанням цифрових технологій та штучного інтелекту.

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				8
Змн.	Арк.	№ докум.	Підпис	Дата		

Предметом дослідження є веборієнтована платформа для надання юридичної допомоги, що автоматизує процеси правового консультування та забезпечує інтерактивну взаємодію між клієнтами та юристами.

За тематикою дипломної роботи були опубліковані тези [1-2].

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1 Постановка задачі

Основна мета роботи — розробка платформи для надання юридичної допомоги, яка використовує штучний інтелект для автоматизації процесів консультування, спрощення доступу до юридичних послуг і підвищення їх ефективності. Це дозволить не тільки розширити доступ до правової допомоги для широкого кола громадян, але й створити зручний інструмент для юристів і компаній, які надають юридичні послуги.

Платформа має складатися з єдиного інтегрованого середовища, яке поєднує різноманітні функціональні компоненти для забезпечення повного циклу надання юридичної допомоги. Першим елементом є модуль реєстрації та авторизації, що відповідає за створення облікових записів із різними рівнями доступу – клієнта, юриста чи адміністратора. Цей компонент не лише зберігає та захищає персональні дані користувачів, а й забезпечує двофакторну аутентифікацію, контроль сесій та можливість швидкого відновлення доступу, що гарантує високий рівень безпеки та зручності.

Наступним ключовим блоком є консультаційний сервіс на основі техніки *retrieval-augmented generation*, який спочатку здійснює пошук релевантних фрагментів із векторного сховища, а потім генерує відповіді з урахуванням вимог чинного українського законодавства [3]. Завдяки такому підходу користувачі отримують не тільки загальні рекомендації, а й деталізовані роз'яснення статей законів та нормативних актів.

Також у межах платформи також має бути реалізовано функціонал відео-консультацій, як окремий модуль, який відповідає за організацію захищених відеодзвінків між клієнтом і юристом. Сервіс передбачає планування відео-консультацій.

База знань, яка зберігається у векторному сховищі, виступає фундаментом усіх автоматизованих процесів: окремий модуль здійснює структурування,

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

класифікацію та регулярне оновлення даних з офіційних джерел, таких як державних реєстрів, сайтів органів влади, нормативно-правових актів. Завдяки вбудованому механізму парсингу, інформація завжди актуальна та достовірна, що дає змогу моделям штучного інтелекту оперувати свіжими даними, а юристам, спиратися на перевірені дані при підготовці складних консультацій.

Компонент управління підписками й оплати відповідає за оформлення тарифів, обробку транзакцій та контроль доступу до преміального контенту. Інтеграція з платіжними засобами дозволяє підтримувати різні методи оплати, автоматично виставляти рахунки та оновлювати статус підписки без втручання адміністратора.

Адміністративна панель забезпечує моніторинг активності користувачів, управління ролями та даними фахівців, оцінку ефективності надання відповідей ІІІ та додавання шаблонів документів, також при необхідності розширення бази знань завантаженням файлу до векторного сховища.

Крім перерахованих компонентів, до платформи буде додано окремий інструментарій для автоматизованої генерації юридичних документів за шаблонами. Цей модуль поєднує механізм вибору готових бланків із системою збору необхідних даних через інтерактивні форми, де користувач вводить базову інформацію про сторони, предмет договору, строки та інші параметри. ІІІ-движок на основі заданих полів динамічно підставляє ці дані у відповідні місця шаблону, а технологія RAG забезпечує використання актуальних нормативних норм і погоджених формулювань з бази знань для формування тексту документа, який адаптується під конкретну ситуацію користувача.

У сукупності всі ці компоненти створюють гнучку, масштабовану й безпечну платформу, здатну задовольнити потреби як індивідуальних користувачів, так і корпоративних клієнтів у сфері юридичного консалтингу.

Процес розробки платформи включає кілька ключових етапів, що реалізуються послідовно. Спочатку здійснюється проєктування та побудова архітектури, яка охоплює як серверну, так і клієнтську частину. На цьому етапі визначаються основні технічні компоненти, їх взаємодія та принципи

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

масштабованості системи. Після цього починається робота над інтерфейсом користувача, яка включає створення дизайну вебсторінок платформи.

Наступним етапом є розробка основного функціоналу платформи. Зокрема, реалізується система авторизації та аутентифікації користувачів з урахуванням ролей (клієнт, юрист, адміністратор). Впроваджується інтерактивна чат-система, яка використовує штучний інтелект для надання консультацій. Крім того, забезпечується можливість проведення відео-консультацій між користувачем та юристом, доступ до яких може залежати від передплати. Також реалізовується система оплати й підписки, функціонал для автоматичного парсингу юридичних документів та механізм їх генерації відповідно до потреб користувача.

Завершальним етапом є тестування розробленої платформи, яке передбачає перевірку її стабільності, безпеки, коректності роботи всіх функцій та відповідності поставленим вимогам.

Результатом реалізації поставленого завдання є веборієнтована платформа для надання юридичної допомоги, яка інтегрує штучний інтелект, відео-консультації та автоматизовану базу знань на основі законодавства України. Платформа забезпечує зручний доступ до правових послуг для громадян і ефективний інструмент для роботи юристів.

1.2 Аналіз аналогів програмного продукту

Основна мета платформи для надання юридичної допомоги полягає у забезпеченні доступності правових послуг шляхом використання технологій штучного інтелекту для автоматизації консультаційних процесів, інтеграції відео-чату для індивідуальних консультацій із юристами та підтримки актуальності бази знань на основі чинного законодавства України. Платформа сприяє розширенню доступу до правової допомоги, спрощує взаємодію між користувачами та юристами, а також забезпечує швидке й точне надання юридичних послуг, адаптованих до локальних умов. Вона підвищує ефективність юридичних процесів, оптимізуючи час і ресурси клієнтів та

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

спеціалістів. Відносно цих критеріїв розглянемо аналогічні платформи для надання юридичної допомоги.

AI Lawyer

AI Lawyer (рис 1.1) — це інноваційна платформа, яка змінює ринок юридичних послуг за допомогою технологій штучного інтелекту [4]. Вона автоматизує юридичні дослідження, спрощує складні терміни та забезпечує ефективну роботу з документами. Платформа підходить як для користувачів, що шукають доступні юридичні консультації, так і для професіоналів та студентів, які бажають оптимізувати свою роботу.

Серед переваг платформи варто відзначити:

1. База промтів для полегшення взаємодії з ШІ, дозволяє користувачам вибирати готові підказки для формулювання запитів, що значно спрощує використання платформи для новачків. Це зручно, оскільки не вимагає глибоких знань у правовій термінології чи навичок роботи з штучним інтелектом.

2. Історія чатів, яка дає змогу користувачам відстежувати попередні консультації, повертатися до важливих відповідей і аналізувати отриману інформацію, що є особливо корисним у тривалих правових процесах.

3. Можливість завантаження власних документів, що дозволяє користувачам передавати свої юридичні файли для автоматичної обробки, аналізу або створення нових документів на їхній основі, що значно економить час і знижує ризик помилок, пов'язаних із людським фактором.

4. Пошук інформації в інтернеті, який використовує ШІ для отримання релевантних даних і відповідей на правові запити, робить консультації більш точними та оперативними.

Водночас платформа має недоліки, які можуть обмежувати її використання в певних випадках:

1. Обмежений функціонал для взаємодії з юристами, оскільки відсутні інструменти, такий як відео-консультації, які необхідні для розв'язання складних

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				13
Змн.	Арк.	№ докум.	Підпис	Дата		

правових питань або переговорів, і це може відштовхнути користувачів, які потребують безпосередньої комунікації зі спеціалістами.

2. База промтів не адаптована до національного законодавства, тому універсальні підказки не враховують специфіку законодавства різних країн, зокрема України, що може призводити до помилкових рекомендацій або недостатньо точних відповідей.

3. Висока вартість підписки, оскільки навіть із урахуванням знижки річний тариф на рівні \$99,99 може бути фінансово недоступним для користувачів із країн у яких середнім чи низький рівень доходу, створюючи бар'єр для ширшого використання платформи.

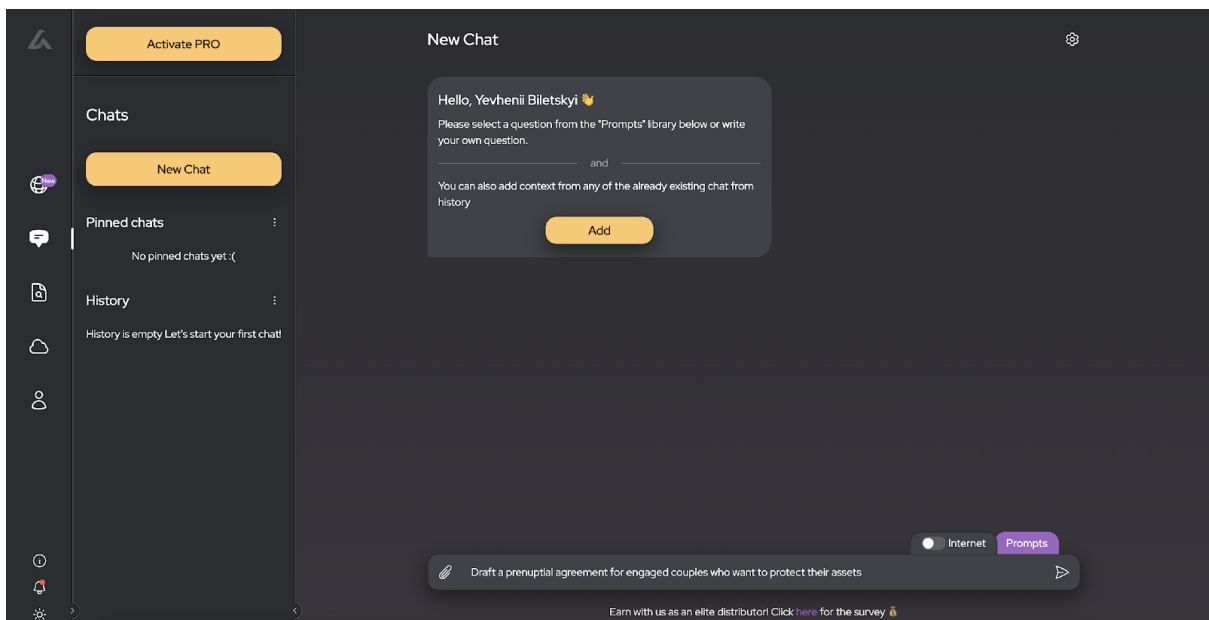


Рисунок 1.1 AI Lawyer.

Law.co

Law.co (рис 1.2) позиціонує себе як передову платформу штучного інтелекту для юристів та юридичних фірм, яка спрямована на діджиталізацію та оптимізацію юридичних операцій за допомогою сучасних технологій [5]. Платформа надає інструменти для автоматизації рутинних завдань, аналізу юридичних документів і створення контрактів, що допомагає знизити витрати часу та підвищити ефективність роботи юристів.

Серед переваг цієї платформи варто відзначити:

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

1. Можливість вибору тарифів, оскільки користувачам доступні різні рівні підписки — від безкоштовного плану для ознайомлення до розширених платних опцій, як-от AI Counsel і GPT Contract Drafter, що дозволяє обрати оптимальний варіант відповідно до індивідуальних потреб.

2. Автоматизація процесів, завдяки якій платформа дозволяє швидко створювати та аналізувати контракти, шаблони й інші юридичні документи із використанням генеративного штучного інтелекту, що значно спрощує виконання рутинних завдань для юридичних фахівців і компаній.

3. Широкий функціонал для роботи з контрактами, адже розширені можливості — такі як Generative AI Contract Drafting, автоматичний аналіз, перевірка на відповідність вимогам та доступ до понад 20 000 шаблонів контрактів, забезпечують зручність і оперативність підготовки юридичних документів.

4. База даних із пошуком, що забезпечує доступ до великого масиву інформації, включаючи пошук у кейсах і завантаження даних з можливістю гнучкої фільтрації для більш ефективної роботи з матеріалами.

До недоліків можна віднести наступне:

1. Висока вартість передплати, адже платні тарифи, зокрема GPT Contract Drafter за \$99 на місяць для одного користувача, можуть бути фінансово непосильними для невеликих юридичних фірм або індивідуальних спеціалістів.

2. Обмежена локалізація, оскільки платформа не враховує особливості українського законодавства та працює лише з документами, що стосуються правової системи США, що значно знижує її корисність для міжнародних користувачів.

3. Відсутність інтерактивного консультування, адже попри широкий функціонал, сервіс не підтримує відеочати чи прямі консультації з юристами, що може бути критично важливим для окремих користувачів.

4. Незручний інтерфейс, оскільки перевантаження функціями ускладнює навігацію для новачків, а також недостатня увага до зручності

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

користування та інтуїтивності може створювати труднощі навіть для досвідчених користувачів.

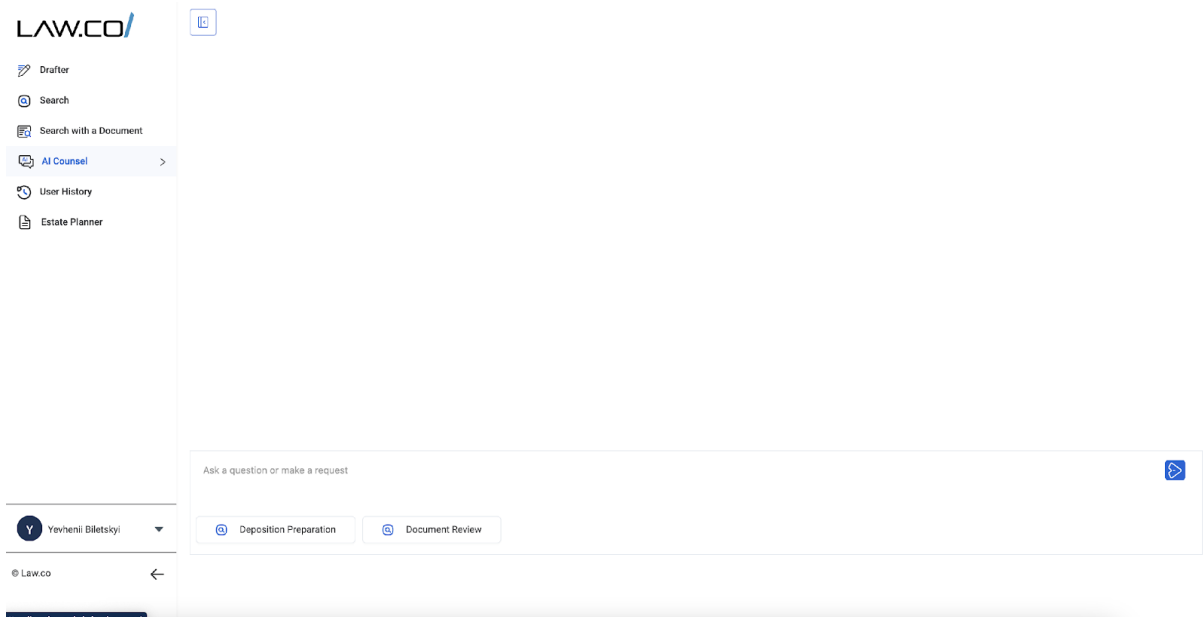


Рисунок 1.2 law.co

DocDraft

DocDraft (рис 1.3) — це сучасна платформа, яка поєднує технології штучного інтелекту з експертним юридичним досвідом для автоматизації процесу створення документів та отримання юридичної допомоги [6]. Вона розрахована як на приватних осіб, так і на малий бізнес, пропонуючи швидкі та ефективні рішення для підготовки юридичних документів, а також доступ до послуг юристів.

Серед переваг цієї платформи варто відзначити:

1. Автоматизоване створення документів, оскільки платформа дозволяє генерувати юридичні документи різних типів за допомогою штучного інтелекту.
2. Послуги підбору юристів, що дає змогу користувачам звертатися до перевірених фахівців для перегляду документів, отримання консультацій або юридичного супроводу.
3. Доступність та прозорість, оскільки DocDraft пропонує конкурентну ціну підписки вартість, якої складає 19,99 доларів на місяць і включає

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

необмежене створення документів, їх збереження у зашифрованій базі даних та доступ до юридичних послуг.

4. Простий процес створення документів, де система використовує покрокові питання для адаптації документів під індивідуальні потреби користувача, зберігаючи їхню унікальність та юридичну значимість.

5. Безпечне зберігання даних, адже вся інформація зберігається у зашифрованій базі даних, що забезпечує конфіденційність та захист від несанкціонованого доступу.

До недоліків можна віднести наступне:

1. Обмежена локалізація, оскільки платформа орієнтована на англomовний ринок і враховує лише законодавство США, що значно зменшує її ефективність для користувачів з інших країн, у тому числі з України.

2. Відсутність інтеграції відео-консультацій, адже сервіс не дає змоги спілкуватися з юристами у форматі відеочату, що може бути важливим для вирішення складних правових питань.

3. Інтерфейс, який може бути незручним для новачків, попри сучасний дизайн, платформа може виявитися складною у використанні для користувачів без юридичного досвіду, особливо під час налаштування шаблонів документів.

4. Відсутність функції аналізу юридичних кейсів, адже платформа зосереджена на створенні документів і не має інструментів для аналізу правових справ чи пошуку відповідей на юридичні питання.

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				17
Змн.	Арк.	№ докум.	Підпис	Дата		

What document will you draft?

Enter the legal document you need or describe your legal goal...



Residential Lease Agreement

Mutual Non-Disclosure Agreement

NY State LLC Operating Agreement

Promissory Note

Рисунок 1.3 Docdraft

Згідно з результатами аналізу аналогів, були виявлені їхні переваги та недоліки, що дозволяє врахувати всі необхідні аспекти при розробці власної платформи. Отримані дані сприятимуть створенню продукту, який буде більш адаптованим до потреб користувачів та локальних умов.

Порівняння функціональних можливостей продуктів наведено нижче (табл. 1.1).

Таблиця 1.1

Функціональні можливості сучасних юридичних платформ

Функціональні можливості	AILawyer	Law.co	DocDraft	Розроблена платформа
Зручний вебінтерфейс	Так	Ні	Так	Так
Доступність для українського законодавства	Ні	Ні	Ні	Так
Відео-консультації з юристом	Ні	Ні	Ні	Так
Генерація документів	Так	Так	Так	Так
Інтеграція ШІ-ботів	Так	Так	Ні	Так

Продовження таблиці 1.1.

Адміністрування ролей користувачів	Ні	Ні	Ні	Так
Керування підписками	Так	Так	Так	Так
Можливість завантаження власних документів	Так	Так	Так	Так
Доступність для малого бізнесу	Ні	Ні	Так	Так
Проста та прозора система тарифів	Ні	Ні	Так	Так
Можливість залишати фідбек на відповіді ШІ	Ні	Ні	Ні	Так
Безпечне зберігання даних	Так	Так	Так	Так

У кожній з представлених платформ є свої унікальні функції та обмеження, що можуть задовольнити різні категорії користувачів. Наприклад, AILawyer і Law.co мають розвинені ШІ-модулі для автоматизації створення документів, однак обмежені адаптацією до національного законодавства та інтерактивними сервісами. DocDraft пропонує конкурентну ціну й простий процес створення документів, але також не враховує специфіку українського права та не має інструментів для відеоконсультацій.

Розроблена юридична платформа об'єднує ключові переваги сучасних рішень і долає їхні недоліки. Забезпечує підтримку українського законодавства, можливість відеоконсультацій, гнучке адміністрування, а також безпечне зберігання даних і прозору систему підписок. Саме це дозволяє зробити продукт максимально адаптованим до локальних потреб і зручним для широкого кола користувачів.

1.3 Вибір архітектури платформи для надання юридичної допомоги з ШІ

Для розробки платформи для надання юридичної допомоги було обрано клієнт-серверну архітектуру [7]. Такий підхід передбачає взаємодію клієнтських

програм із сервером для отримання, обробки та збереження необхідних даних. Клієнт-серверна архітектура ґрунтується на принципі розподілу ролей між двома основними компонентами: клієнтом і сервером.

Клієнт відповідає за вебінтерфейс, з яким взаємодіють користувачі, а також за збирання даних, які передаються на сервер для подальшої обробки. Сервер, у свою чергу, виконує бізнес-логіку платформи, обробляє запити клієнтів, забезпечує валідацію даних і зберігає інформацію у базі даних (рис. 1.4).



Рисунок 1.4 Клієнт-серверна архітектура

Сервер відповідає за реалізацію основної бізнес-логіки платформи, обробляючи запити користувачів і взаємодіючи з базою даних. Він забезпечує інтеграцію з OpenAI API [8] для автоматизації юридичних консультацій і зберігає дані, зокрема історію консультацій, профілі користувачів і інформацію про підписки. Крім того, сервер виконує перевірку прав доступу до різних функцій платформи. Це включає контроль доступу до преміум-функцій, таких як відео-консультації.

Для підвищення точності та достовірності відповідей платформа використовує метод RAG. Спочатку система здійснює пошук і відбір релевантних та достовірних даних із векторного сховища, після чого на основі відібраної інформації генерує відповіді користувачу за допомогою мовної моделі. Такий підхід дозволяє отримувати більш обґрунтовані й актуальні відповіді клієнта, оскільки генеративна модель спирається на перевірені джерела

замість суто статистичних закономірностей. Схематичне зображення роботи методу RAG наведено на рисунку 1.5.



Рисунок 1.5 – Схема роботи методу Retrieval-Augmented Generation.

Клієнтська частина платформи буде реалізована у вигляді односторінкового додатку (SPA), що забезпечить швидкий і зручний інтерфейс для користувачів. Завдяки SPA користувачі зможуть взаємодіяти з платформою без необхідності перезавантаження сторінок, що покращить продуктивність і забезпечить більш плавний досвід роботи [9]. Основною задачею клієнта є збір даних від користувачів, відображення результатів консультацій, доступ до функцій, таких як історія чатів, управління підписками, і надсилання запитів до серверної частини. Клієнтська частина також забезпечить адаптивний дизайн, що дозволить платформі коректно працювати на різних пристроях, включаючи мобільні телефони, планшети та комп'ютери.

Також серверна частина платформи побудована за модульно-фічевим принципом, де кожна бізнес-область (авторизація, повідомлення, відеоконсультації, підписки тощо) розміщена в окремому модулі зі своїми контролерами, сервісами й моделями. Ін'єкція залежностей зводить прямі

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				21
Змн.	Арк.	№ докум.	Підпис	Дата		

залежності до мінімуму, що полегшує тестування та заміну реалізацій. Патерн Фасад приховує складні модулі за зрозумілими інтерфейсами. Завдяки такому підходу зберігається чіткий розподіл обов'язків між шаром обробки запитів, бізнес-логікою та моделями даних, а вся функціональність залишається ізольованою в доменних модулях, що сприяє масштабуванню й простоті підтримки коду.

1.4 Обґрунтування вибору інструментальних засобів

Для розробки проєкту були обрані сучасні та ефективні інструментальні засоби, що забезпечують високу продуктивність, гнучкість і масштабованість майбутньої платформи. Основною мовою програмування став TypeScript [10], який завдяки статичній типізації зменшує кількість помилок під час розробки, спрощує рефакторинг та забезпечує зручну інтеграцію з такими технологіями, як Next.js, NestJS і Mongoose [11-13]. Це дозволяє досягти більшої безпеки та зрозумілості у взаємодії між фронтендом і бекендом.

У клієнтській частині платформи використовується стек, що включає Next.js, який забезпечує оптимізацію рендерингу завдяки використанню серверних компонентів, вбудованої маршрутизації та підтримки SSR/SSG [14-15], що сприяє швидкому завантаженню сторінок і покращенню користувацького досвіду. Для реалізації механізмів автентифікації обрано next-auth [16], який дозволяє інтегрувати різні способи входу в систему наприклад через OAuth або за допомогою email та паролю, та також ефективно працює із сесіями, забезпечуючи належний рівень безпеки.

Обробка API-запитів на клієнті здійснюється за допомогою react-query [17], що забезпечує ефективне керування кешем, дозволяє легко організовувати автоматичне оновлення даних і спрощує роботу з асинхронними операціями. Для створення адаптивного та сучасного інтерфейсу використовується Tailwind CSS [18], що дозволяє розробляти компоненти без написання великої кількості кастомного CSS [19] і значно прискорює робочий процес. У побудові інтерфейсу також беруть участь компоненти Radix UI [20], які відзначаються високою

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				22
Змн.	Арк.	№ докум.	Підпис	Дата		

доступністю та зручною кастомізацією через Tailwind CSS, завдяки чому легко створювати навіть складні елементи дизайну без втрати контролю над їх виглядом.

Управління станом форм і їх валідація реалізується з використанням react-hook-form, що мінімізує кількість непотрібних перерендерень, спрощує обробку подань і перевірку даних, особливо у великих і складних формах. Для забезпечення надійної типобезпечної валідації як на клієнті, так і на сервері, використовується бібліотека zod, яка інтегрується з react-hook-form [21] і дозволяє гарантувати, що структура введених користувачем даних відповідає очікуваним схемам. Усі ці інструменти у поєднанні створюють цілісну, ефективну і легко підтримувану архітектуру клієнтської частини платформи.

Для серверної частини вибрано стек технологій, який забезпечує надійність, масштабованість та простоту у підтримці складної бізнес-логіки. Основу серверної архітектури становить NestJS, що підтримує модульну архітектуру та інверсію керування, завдяки чому стає можливим ефективно масштабування системи й організація коду у вигляді зрозумілих, ізольованих модулів. Для роботи з базою даних обрано Mongoose, який надає потужний інструментарій для моделювання документів у MongoDB, гнучку валідацію даних та ефективно управління операціями зберігання та отримання інформації.

Для впровадження штучного інтелекту у платформі використовується фреймворк LangChain [22], який дозволяє інтегрувати сучасні великі мовні моделі, автоматизувати обробку даних, створювати складні ланцюги завдань для аналізу текстів, генерації відповідей та взаємодії із зовнішніми ШІ-сервісами. Завдяки LangChain забезпечується доступ до провідних мовних моделей, зокрема OpenAI, що дає змогу реалізувати автоматичне формування юридичних документів і підвищити рівень автоматизації юридичної допомоги.

Для роботи з векторними представленнями тексту та організації швидкого пошуку за змістом використовується бібліотека ChromaDB [23]. Це спеціалізований інструмент, який оптимізований для зберігання й пошуку ембеддінгів, що дозволяє ефективно реалізувати систему смислового пошуку

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				23
Змн.	Арк.	№ докум.	Підпис	Дата		

документів, а також підтримувати функціонал інтелектуальних чатботів та асистентів у межах платформи.

Щоб забезпечити надійну аутентифікацію та захист даних користувачів, у платформі використовується бібліотека Passport.js [24], яка підтримує різноманітні стратегії авторизації, зокрема JWT і OAuth [25-26]. Це дозволяє ефективно контролювати доступ до ресурсів платформи, забезпечуючи як простоту інтеграції, так і високий рівень безпеки для різних категорій користувачів.

Для автоматизованого отримання текстової інформації з PDF-документів у серверній частині використовується бібліотека pdf-parse [27], що дає змогу ефективно обробляти юридичні та супровідні документи, здійснювати аналітику й забезпечувати зручний пошук у великому обсязі текстових даних.

Окрім цього, у межах задач парсингу даних та автоматизації взаємодії з вебресурсами застосовується бібліотека puppeteer [28]. Її можливості дозволяють організувати збір актуальної інформації з відкритих джерел у режимі реального часу, але й здійснювати автоматичне оновлення інформації, що підвищують практичну цінність і функціональність платформи.

Для відправки сервісних повідомлень, таких як підтвердження реєстрації або відновлення пароля, інтегрується nodemailer [29], що дозволяє надійно організувати роботу з електронною поштою на сервері. Захист чутливої інформації, зокрема паролів користувачів, здійснюється за допомогою бібліотеки bcrypt, яка забезпечує безпечне хешування та зберігання паролів у базі даних.

Для попередньої обробки великих текстових масивів використовується бібліотека Langchain Textsplitters [30]. Вона дає змогу розбивати значні обсяги тексту на логічно завершені та структуровані фрагменти, що особливо важливо перед проведенням векторизації, аналізу або генерації відповідей штучним інтелектом. Завдяки такому підходу до підготовки даних забезпечується їхня якість, релевантність і точність подальшої обробки у межах платформи.

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				24
Змн.	Арк.	№ докум.	Підпис	Дата		

Для створення ефективної, масштабованої та надійної бази даних у межах проєкту було проведено аналіз чотирьох систем управління базами даних, серед яких MongoDB, PostgreSQL, MySQL та Cassandra [31-34]. Оцінка здійснювалась за ключовими параметрами, такими як максимальний обсяг даних, алгоритми безпеки, продуктивність, підтримка транзакцій, масштабованість і зручність налаштування.

Для вибору бази даних важливо врахувати максимальний обсяг збережених даних, який підтримує кожна СУБД. Це забезпечує можливість роботи системи з великими масивами інформації без збоїв (табл. 1.2).

Таблиця 1.2

	Максимальний обсяг збережених даних	Особливості обмеження
MongoDB	Теоретично необмежений (до 1 ТБ на колекцію у 32-бітній системі).	Залежить від файлової системи. При використанні шардінгу можливо зберігати петабайти даних.
PostgreSQL	Обмежено файловою системою сервера (наприклад, до 64 ТБ на таблицю у x64).	Підтримує розподілене зберігання даних для роботи з великими обсягами.
MySQL	Теоретично необмежений (практично — кілька терабайтів на таблицю).	Обмеження можуть бути через файлову систему (наприклад, 64 ТБ у InnoDB).
Cassandra	Необмежений (масштабування за рахунок додавання нод у кластер).	Горизонтальне масштабування дозволяє працювати з петабайтами даних.

Також для платформи безпека даних є важливим аспектом для будь-якої СУБД. Було проаналізовано алгоритми шифрування, механізми контролю доступу та можливості відновлення даних після збоїв (табл. 1.3).

Таблиця 1.3

	Алгоритми шифрування даних	Контроль доступу	Відновлення після збою
--	----------------------------	------------------	------------------------

Продовження таблиці 1.3.

MongoDB	AES256 для шифрування даних на диску; TLS для передачі даних.	Ролі користувачів із гранульованими дозволами.	Підтримка реплікаційних наборів.
PostgreSQL	Шифрування SSL/TLS, підтримка клієнтського шифрування даних.	Ролі та політики доступу (Row-Level Security).	Архівування WAL, асинхронна реплікація.
MySQL	Шифрування SSL/TLS, Transparent Data Encryption (TDE).	Ролі користувачів, обмеження на рівні таблиць.	Резервне копіювання та реплікація.
Cassandra	Підтримка SSL/TLS для передачі; інтеграція із зовнішніми шифрувальниками.	Контроль доступу через плейстор.	Резервні копії на рівні кластерів.

Продуктивність бази даних визначається швидкістю запису, читання та підтримкою транзакцій (табл. 1.4). Це критично важливо для систем, які обробляють великий обсяг даних у реальному часі.

Таблиця 1.4

	Швидкість запису	Швидкість читання	Продуктивність транзакцій
MongoDB	Висока, особливо для великих обсягів даних.	Висока, завдяки індексам.	Базова підтримка транзакцій (до 2-3 нод).
PostgreSQL	Помірна (залежить від конфігурації).	Висока для складних запитів із JOIN.	Висока підтримка транзакцій ACID.
MySQL	Помірна, залежить від використаного рушія (InnoDB/Memory).	Висока для простих запитів.	Висока, з повною підтримкою ACID.

Продовження таблиці 1.4.

Cassandra	Дуже висока для розподілених кластерів.	Висока, якщо дані денормалізовані.	Обмежена підтримка транзакцій (lightweight).
-----------	---	------------------------------------	--

Масштабованість бази даних визначає її здатність ефективно працювати за умов збільшення навантаження. Аналіз проводився за типом масштабування, перевагами та недоліками (табл. 1.5).

Таблиця 1.5

	Тип масштабування	Переваги	Недоліки
MongoDB	Горизонтальна	Простий шардінг для масштабування.	Може вимагати додаткової оптимізації індексів.
PostgreSQL	Вертикальна	Стабільність і продуктивність для транзакцій.	Обмежена можливість горизонтального масштабування.
MySQL	Вертикальна	Просте налаштування для середніх навантажень.	Відсутність гнучкого горизонтального масштабування.
Cassandra	Горизонтальна	Розподілена архітектура, автоматичний розподіл даних.	Складність у налаштуванні.

Після аналізу було обрано MongoDB як основну СУБД для проєкту. Вона забезпечує гнучкість у роботі з даними, підтримує горизонтальне масштабування та швидко інтегрується з такими технологіями, як NestJS і Mongoose. Це дозволяє створити продуктивну, масштабовану та сучасну систему, яка відповідає вимогам проєкту.

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				27
Змн.	Арк.	№ докум.	Підпис	Дата		

Для вибору оптимальної векторної бази даних у межах проєкту було проаналізовано чотири популярні рішення, а саме ChromaDB, Pinecone, Weaviate та FAISS [34-37]. Оцінювання відбувалося за основними критеріями, серед яких максимальний обсяг даних, швидкість пошуку векторів, підтримка масштабування, інтеграція з іншими сервісами та простота розгортання.

Важливим критерієм при виборі векторної бази є здатність ефективно обробляти великі обсяги ембеддінгів і швидко здійснювати пошук найближчих сусідів (табл. 1.6).

Таблиця 1.6

База даних	Максимальний обсяг даних	Особливості зберігання
ChromaDB	Необмежений (залежить від об'єму сховища)	Підтримка роботи на локальних і хмарних серверах, адаптація до великих обсягів
Pinecone	Підтримка мільярдів векторів	Автоматичне масштабування в хмарі, оптимізоване для production-навантажень
Weaviate	Теоретично необмежений	Вбудоване масштабування, розподілене зберігання даних
FAISS	Залежить від доступної оперативної пам'яті та диску	Орієнтований на локальне розгортання без вбудованого масштабування

Критичною для платформи є також швидкість пошуку та масштабованість рішення в умовах зростання кількості даних (табл. 1.7).

Таблиця 1.7

База даних	Швидкість пошуку	Масштабованість
ChromaD B	Висока для мільйонів об'єктів	Горизонтальне масштабування через шардінг
Pinecone	Дуже висока, оптимізація для хмарних сервісів	Повністю кероване горизонтальне масштабування
Weaviate	Висока, із підтримкою hybrid search (вектори + фільтри)	Вбудоване кластеризаційне масштабування
FAISS	Дуже висока для локальних пошуків	Потрібна ручна організація масштабування

Інтеграція з іншими системами також є важливим фактором, особливо для використання в AI/ML проєктах (табл. 1.8).

Таблиця 1.8

База даних	Інтеграція та підтримка
ChromaD B	Інтеграція з LangChain, підтримка REST API, Python SDK
Pinecone	Інтеграція з LangChain, OpenAI, Hugging Face, підтримка gRPC
Weaviate	Підтримка GraphQL API, інтеграція з Hugging Face, LangChain
FAISS	Інтеграція через Python-бібліотеки, немає прямої серверної API

Після аналізу було обрано ChromaDB як основну векторну базу даних для проєкту. Вона забезпечує високу продуктивність пошуку векторів, просту інтеграцію з інструментами обробки природної мови, такими як LangChain, і дозволяє швидко масштабувати систему відповідно до зростання кількості

даних. Використання ChromaDB забезпечить стабільність роботи AI-модулів і швидкий доступ до релевантної інформації у великих обсягах даних.

Висновки до першого розділу

У першому розділі було сформульовано мету та практичні завдання кваліфікаційної роботи, які передбачають розробку сучасної платформи для надання юридичної допомоги із застосуванням штучного інтелекту. Проведено аналіз ринку та існуючих аналогів, що дозволило виокремити основні функціональні можливості майбутньої системи: інтеграцію консультативного сервісу з ШІ, модуль відео-консультацій, персоналізований кабінет користувача, автоматизовану генерацію юридичних документів, а також управління підписками і оплатою.

Дослідження наявних платформ дозволило визначити конкурентні переваги та недоліки існуючих рішень, що стало основою для розробки власної архітектури. Особливий акцент зроблено на забезпеченні підтримки українського законодавства, впровадженні гнучкого адміністрування ролей користувачів, прозорій тарифної політики та безпечного зберігання даних. Запропоновано інтеграцію інструментів автоматизації та персоналізації для підвищення якості консультування й ефективності роботи всієї платформи.

У розділі також обґрунтовано вибір архітектурних і технологічних рішень. Для розробки платформи обрано клієнт-серверну архітектуру, яка чітко розмежовує інтерфейс користувача та бізнес-логіку. Для збереження й обробки даних використовується MongoDB із Mongoose, а для семантичного пошуку – ChromaDB. Клієнтська частина побудована із застосуванням сучасних технологій таких як Next.js, TypeScript, React Query, Tailwind CSS, Radix UI, а серверна із впровадженням NestJS, LangChain, OpenAI API, Puppeteer, pdf-parse, Passport.js та інших засобів. Комплексний вибір інструментів забезпечує масштабованість, ефективність та зручність використання платформи для різних категорій користувачів.

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				30
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЄКТУВАННЯ ПЛАТФОРМИ ДЛЯ НАДАННЯ ЮРИДИЧНОЇ ДОПОМОГИ З ШТУЧНИМ ІНТЕЛЕКТОМ

2.1 Визначення варіантів використання та структури платформи

Платформа для надання юридичної допомоги з використанням штучного інтелекту надає вебінтерфейс, через який користувачі можуть отримувати консультації від ШІ, бронювати відео-дзвінки з юристами, генерувати юридичні документи та завантажувати власні документи для аналізу. Функціональність платформи залежить від активності користувачів. Користувачів поділено на три основні групи, а саме звичайні користувачі, юристи та адміністратори. Звичайні користувачі можуть ставити запитання до ШІ, створювати запити на консультації з юристами, а також генерувати й надсилати документи для аналізу за допомогою ШІ. Юристи відповідають за перевірку відповідей, проведення консультацій та оновлення шаблонів документів для генерації. Адміністратори відповідають за управління ботами, модерування юристів та наповнення баз знань. Для наглядної взаємодії між елементами додатку було створено діаграму варіантів використання (рис. 2.1).

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				31
Змн.	Арк.	№ докум.	Підпис	Дата		

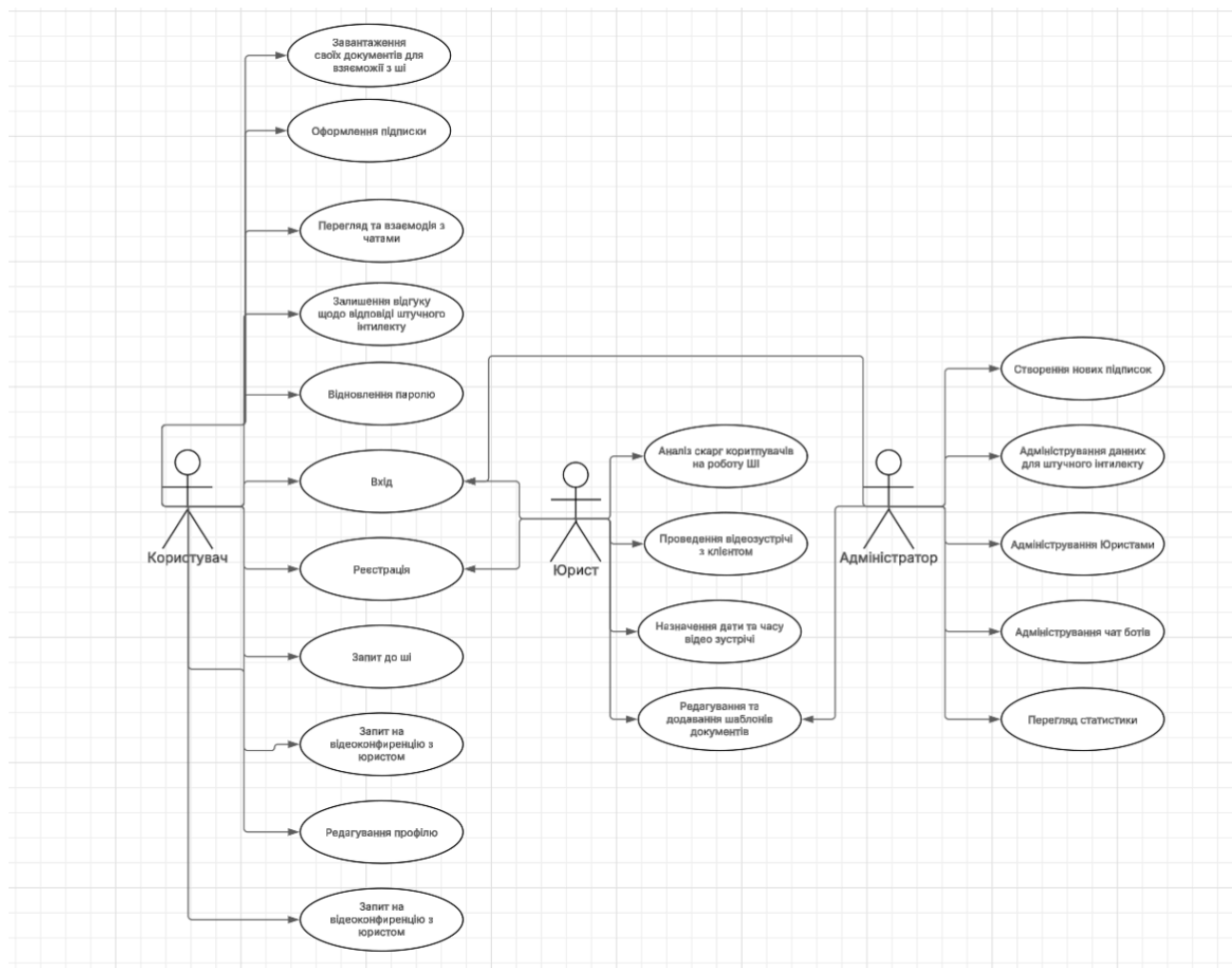


Рисунок. 2.1. Діаграма варіантів використання

Опис основних варіантів використання наведено нижче (табл 2.1. – 2.13).

Таблиця 2.1

Опис прецеденту «Вхід/Реєстрація»

Поле	Опис
Назва варіанту використання	Вхід/Реєстрація
Основні актори	Користувач, Юрист, Адміністратор
Короткий опис	Користувачі, юристи та адміністратори здійснюють реєстрацію або вхід для доступу до системи.

Таблиця 2.2

Опис прецеденту «Запит до ШІ»

Поле	Опис
Назва варіанту використання	Запит до ШІ
Основні актори	Користувач
Короткий опис	Користувач ставить запитання штучному інтелекту та отримує відповідь.

Таблиця 2.3

Опис прецеденту «Запит на відеоконференцію з юристом»

Поле	Опис
Назва варіанту використання	Запит на відеоконференцію з юристом
Основні актори	Користувач, Юрист
Короткий опис	Користувач надсилає запит на проведення відеозустрічі з юристом, юрист підтверджує запит.

Таблиця 2.4

Опис прецеденту «Завантаження документів»

Поле	Опис
Назва варіанту використання	Завантаження своїх документів для взаємодії з ШІ
Основні актори	Користувач
Короткий опис	Користувач завантажує документи для аналізу або використання у запитах до штучного інтелекту.

Таблиця 2.5

Опис прецеденту «Редагування профілю»

Поле	Опис
Назва варіанту використання	Редагування профілю
Основні актори	Користувач, Юрист
Короткий опис	Користувач або юрист можуть змінювати свої персональні дані та налаштування облікового запису.

Таблиця 2.6

Опис прецеденту «Оформлення підписки»

Поле	Опис
Назва варіанту використання	Оформлення підписки
Основні актори	Користувач
Короткий опис	Користувач оформляє або продовжує підписку для отримання розширеного функціоналу платформи.

Таблиця 2.7

Опис прецеденту «Залишення відгуку щодо відповіді ШІ»

Поле	Опис
Назва варіанту використання	Залишення відгуку щодо відповіді ШІ
Основні актори	Користувач
Короткий опис	Користувач залишає фідбек про якість відповіді, згенерованої штучним інтелектом.

Таблиця 2.8

Опис прецеденту «Проведення відеозустрічі з клієнтом»

Поле	Опис
Назва варіанту використання	Проведення відеозустрічі з клієнтом
Основні актори	Юрист
Короткий опис	Юрист проводить відеозустріч із користувачем у погоджений час.

Таблиця 2.9

Опис прецеденту «Аналіз скарг користувачів»

Поле	Опис
Назва варіанту використання	Аналіз скарг користувачів
Основні актори	Юрист
Короткий опис	Юрист аналізує скарги клієнтів щодо наданої допомоги або роботи ІІІ.

Таблиця 2.10

Опис прецеденту «Створення нових підписок»

Поле	Опис
Назва варіанту використання	Створення нових підписок
Основні актори	Адміністратор
Короткий опис	Адміністратор створює або редагує доступні тарифні плани підписки для користувачів.

Опис прецеденту «Адміністрування даних для ІІІ»

Поле	Опис
Назва варіанту використання	Адміністрування даних для ІІІ
Основні актори	Адміністратор
Короткий опис	Адміністратор додає нові документи та знання для навчання чат-ботів ІІІ.

Таблиця 2.12

Опис прецеденту «Адміністрування юристів»

Поле	Опис
Назва варіанту використання	Адміністрування юристів
Основні актори	Адміністратор
Короткий опис	Адміністратор керує обліковими записами юристів та їх доступом до функцій платформи.

Таблиця 2.13

Опис прецеденту «Перегляд статистики»

Поле	Опис
Назва варіанту використання	Перегляд статистики
Основні актори	Адміністратор
Короткий опис	Адміністратор переглядає загальну аналітику щодо використання платформи, активності чат-ботів і результативності консультацій.

Основною метою створення платформи для надання юридичної допомоги за допомогою штучного інтелекту є формування сучасного онлайн-простору, що забезпечує швидкий, зручний та доступний спосіб отримання правових консультацій. Платформа об'єднує автоматизовані сервіси ШІ та досвідчених юристів, створюючи умови для оперативної взаємодії користувачів з професійною юридичною спільнотою.

Її призначення полягає у підтримці громадян у вирішенні юридичних питань шляхом надання можливості отримувати відповіді через штучний інтелект, бронювати відеоконсультації з фахівцями, створювати правові документи, передавати свої документи для аналізу за допомогою ШІ. Платформа також дозволяє здійснювати контроль якості наданих консультацій шляхом збору зворотного зв'язку. Таким чином, система сприяє підвищенню рівня правової грамотності користувачів, розширює доступ до якісної правової допомоги та забезпечує зручність отримання послуг незалежно від місцезнаходження клієнта.

Вимоги користувачів

Внутрішній користувач – Звичайний користувач

1. Реєстрація та авторизація на платформі.
2. Отримання консультацій від штучного інтелекту у форматі чату.
3. Залишення відгуку на відповідь, отриману від ШІ.
4. Бронювання відеоконсультацій з юристами, вибір дати та часу зустрічі.
5. Завантаження власних документів для аналізу за допомоги ШІ.
6. Генерація стандартних юридичних документів на основі запропонованих шаблонів.
7. Перегляд історії чатів.
8. Редагування особистого профілю (зміна пароля, тощо).
9. Оформлення підписки для розширеного функціоналу.

Внутрішній користувач – Юрист

1. Перегляд запитів користувачів на відеоконсультації.

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				37
Змн.	Арк.	№ докум.	Підпис	Дата		

2. Призначення дати та часу для проведення відеозустрічей.
3. Проведення відеоконсультацій із користувачами через платформу.
4. Перевірка за потреби відповідей, наданих штучним інтелектом.
5. Аналіз скарг та відгуків користувачів щодо відповідей ІІІ.
6. Додавання та оновлення шаблонів документів.

Внутрішній користувач – Адміністратор

1. Управління обліковими записами юристів (додавання, редагування, деактивація).
2. Додавання нових ботів та оновлення баз знань для ІІІ.
3. Ручне завантаження документів для покращення відповідей ІІІ.
4. Модерація контенту, перевірка якості наданих відповідей ботами.
5. Створення і налаштування тарифних планів для підписок.
6. Перегляд аналітики та статистики роботи платформи (кількість запитів, консультацій, підписок тощо).

Функціональні вимоги

Платформа надання юридичних послуг має забезпечувати наступний перелік можливостей:

1. Реєстрація та авторизація користувачів усіх ролей (користувач, юрист, адміністратор).
2. Створення запитів до чат-бота на базі штучного інтелекту та отримання автоматизованих відповідей.
3. Бронювання відеозустрічей з юристами через календар доступності.
4. Завантаження власних документів для аналізу штучним.
5. Створення юридичних документів за шаблонами.
6. Можливість залишати відгуки на відповіді чат-бота.
7. Редагування особистого профілю.
8. Оформлення та продовження підписок для доступу до додаткового функціоналу.
9. Перегляд історії чатів та консультацій.

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				38
Змн.	Арк.	№ докум.	Підпис	Дата		

10. Для юристів — управління своїми консультаціями, перевірка запитів користувачів, додавання та оновлення шаблонів документів.

11. Для адміністраторів — управління чат-ботами, модерування бази знань, робота з юристами та формування звітності.

Нефункціональні вимоги

1. Інтерфейс користувача має бути максимально інтуїтивним, адаптованим для швидкого освоєння (до 5 хвилин для нових користувачів).
2. Середній час відповіді ШІ на запит користувача не повинен перевищувати 10 секунд при стандартному навантаженні.
3. Платформа має бути стійкою до помилок і забезпечувати доступність сервісу не менше ніж 99 % у режимі 24/7.
4. У разі збою система повинна автоматично відновлювати роботу без втрати даних користувача.
5. Продуктивність системи має забезпечувати одночасне обслуговування не менше 10000 активних користувачів без погіршення якості сервісу. Модулі обробки документів і генерації відповідей повинні бути масштабованими для роботи із великими обсягами даних.
6. Зберігання інформації про чат-сесії, завантажені файли та відеозустрічі повинно бути організовано із дотриманням принципів безпеки та резервного копіювання.
7. Вебверсія повинна адаптуватися до різних пристроїв і коректно працювати як на ПК, так і на мобільних пристроях.

Аналіз функціональних вимог дав змогу визначити основні сутності для реалізації програмної частини платформи, а саме User, LawyerProfile, Conversation, Message, Bot, Subscription, SubscriptionPlan, Feedback, LegalTemplate, VectorCollection, File та VideoConsultation. Для наочного представлення структури системи було розроблено діаграму класів (рис. 2.2). На основі виявлених сутностей та їх взаємозв'язків було сформовано основні поля та функціональні можливості для кожної моделі, що дозволило чітко окреслити структуру модулів платформи. Було виділено модулі для управління

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				39
Змн.	Арк.	№ докум.	Підпис	Дата		

користувачами, організації чатів, бронювання та проведення відеоконсультацій, обробки підписок і тарифних планів, зберігання шаблонів юридичних документів, роботи з колекціями векторних сховищ і файлами для забезпечення ІІІ-функціоналу, а також обробки користувацьких відгуків. Окрім цього, додано модулі для ведення профілів юристів і забезпечення багаторівневої взаємодії між учасниками платформи. Для зберігання даних було обрано документо-орієнтовану базу даних MongoDB, яка дозволяє гнучко працювати зі структурованими документами у форматі JSON, що забезпечує масштабованість, високу продуктивність при обробці великої кількості запитів та зручність у подальшому розвитку функціональності платформи.

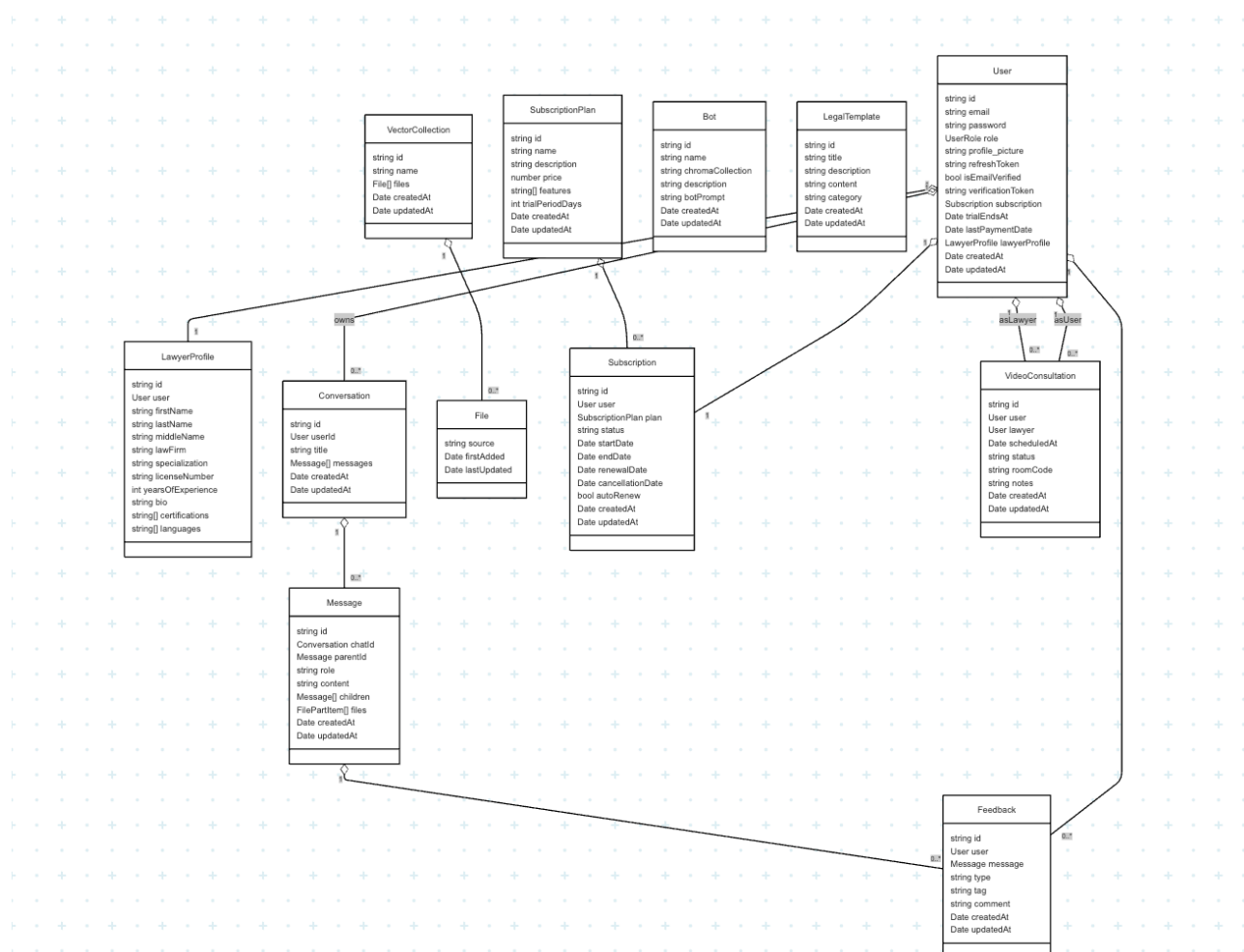


Рисунок 2.2. Діаграма класів

Клас User відповідає за сутність користувача платформи та містить основну інформацію про обліковий запис. Окрім стандартних полів, користувач

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				40
Змн.	Арк.	№ докум.	Підпис	Дата		

має роль (користувач, юрис адміністратор), що визначає його функціональні можливості в системі. Користувач може взаємодіяти з чат-ботами, створювати й переглядати розмови, залишати відгуки, оформлювати підписки, а також, у разі ролі юрист, мати пов'язаний профіль юриста. Крім цього, користувач може брати участь у відеоконсультаціях як клієнт або юрист.

Клас LawyerProfile містить розширену інформацію про профіль юриста, таку як ім'я, прізвище, досвід, спеціалізацію, ліцензії, сертифікати та володіння мовами. Він пов'язується з користувачем, який має роль юриста, і дозволяє системі відображати профілі спеціалістів для користувачів.

Клас Conversation містить інформацію про окрему сесію спілкування користувача з чат-ботом або юристом. Кожна розмова складається з послідовності повідомлень, що пов'язані з користувачем та зберігають структуру діалогу.

Клас Message відповідає за збереження окремого повідомлення в межах розмови. Повідомлення має роль (користувача або бот), текстове наповнення, а також може містити вкладені повідомлення. Окрім тексту, до повідомлення можуть бути прикріплені файли та зображення, а також воно може отримувати оцінки та фідбек.

Клас Bot представляє штучного інтелектуального помічника, який обробляє юридичні запити користувачів. Бот має власну колекцію знань, опис та налаштування промпу для взаємодії із системою.

Клас Subscription містить інформацію про підписку користувача, включаючи посилання на користувача, тарифний план, статус, а також дати початку, завершення та автоматичного поновлення. Таким чином організується управління доступом до платних функцій платформи.

Клас SubscriptionPlan визначає основні характеристики тарифних планів, такі як назва, опис, ціна, список доступних можливостей та тривалість пробного періоду. Він використовується при оформленні та управлінні підписками користувачів.

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				41
Змн.	Арк.	№ докум.	Підпис	Дата		

Клас Feedback забезпечує механізм збору відгуків користувачів щодо отриманих відповідей у чаті або результатів консультацій. Відгук містить тип (like чи dislike), теги (наприклад, «корисно», «недоречно», «шкідливо»), а також, за потреби, текстовий коментар.

Клас LegalTemplate є основою для автоматизованого створення юридичних документів на платформі. Він містить назву, опис, текст шаблону та категорію документа. Користувачі можуть генерувати документи, підставляючи власні дані у відповідні шаблони.

Клас VectorCollection дозволяє організовувати та зберігати колекції векторних представлень даних, що використовуються для роботи із семантичним пошуком. Кожна колекція містить масив файлів, для яких ведеться облік дати додавання та оновлення.

Клас File містить інформацію про джерело файлу, а також дати першого додавання та останнього оновлення. Він використовується для зберігання документів, які можуть бути проіндексовані й використані у векторних колекціях.

Клас VideoConsultation реалізує функціонал бронювання та проведення відеоконсультацій між користувачем і юристом. Він містить посилання на клієнта та юриста, дату та час зустрічі, статус консультації, код доступу до кімнати та додаткові нотатки.

2.2 Розробка бази даних

Для зберігання даних користувачів, чатів, консультацій, підписок та фінансових транзакцій у додатку використовується база даних MongoDB. Це одна з найпопулярніших систем управління базами даних, що працює за документо-орієнтованою моделлю. MongoDB забезпечує високу гнучкість у роботі з даними, дозволяючи зберігати інформацію у вигляді структурованих документів без необхідності жорсткої схемної залежності. Вибір MongoDB для даного проекту обумовлений особливостями предметної області, де необхідно працювати з динамічно змінними структурами даних (чат-сесії, повідомлення,

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				42
Змн.	Арк.	№ докум.	Підпис	Дата		

підписки, документи). Завдяки підтримці вкладених об'єктів та колекцій документів у форматі BSON, MongoDB дозволяє ефективно реалізувати складні взаємозв'язки між користувачами, повідомленнями, консультаціями та юридичними шаблонами без необхідності суворого нормалізованого розділення таблиць, характерного для реляційних систем.

База даних системи побудована за документо-орієнтованою моделлю та складається з 12 основних колекцій:

1. Users – колекція для зберігання даних користувачів платформи, їхніх ролей, підписок, статусу підтвердження електронної пошти, а також зв'язків із профілями юристів.

2. LawyerProfiles – окрема колекція для детального зберігання профілів юристів, що містять інформацію про спеціалізацію, ліцензії, сертифікати та досвід роботи.

3. Conversations – колекція для зберігання інформації про чат-сесії користувачів із ботом.

4. Messages – колекція для зберігання усіх повідомлень у рамках консультацій з ШІ, із підтримкою вкладених повідомлень, файлів і зображень.

5. Bots – колекція для управління даними про інтелектуальних ботів, що мають власну колекцію знань та промпт для роботи з користувачами.

6. Subscriptions – колекція для обліку активних і минулих підписок користувачів, їхніх статусів та термінів дії.

7. SubscriptionPlans – колекція для опису тарифних планів, вартості, функціоналу та тривалості пробних періодів.

8. Feedbacks – колекція для зберігання відгуків користувачів на отримані відповіді від ШІ, з фіксацією типу оцінки, тегів і коментарів.

9. LegalTemplates – колекція для управління шаблонами юридичних документів, які можуть автоматично наповнюватися даними користувача.

10. VideoConsultations – колекція для зберігання заявок на відеоконсультації, із інформацією про клієнта, юриста, дату й час зустрічі та статус проведення.

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				43
Змн.	Арк.	№ докум.	Підпис	Дата		

11. VectorCollections – колекція для структурованого зберігання векторних представлень даних, що використовуються для III-пошуку чи обробки документів.

12. Files – допоміжна колекція для зберігання інформації про завантажені документи до векторного сховища, дати їх додавання та оновлення.

Опис кожної з колекцій бази даних наведений у таблицях 2.14–2.22.

Таблиця 2.14

Опис колекції Users

Поле	Тип даних	Опис
email	String	Адреса електронної пошти користувача
password	String	Хешований пароль користувача
role	Enum (user, admin, lawyer)	Роль користувача у системі
profile_picture	String	URL фото профілю
refreshToken	String	Токен оновлення сесії
isEmailVerified	Boolean	Статус підтвердження електронної пошти
verificationToken	String	Токен для підтвердження реєстрації
subscription	ObjectId	Посилання на підписку
trialEndsAt	Date	Дата завершення безкоштовного періоду
lastPaymentDate	Date	Дата останнього платежу
lawyerProfile	ObjectId	Посилання на профіль юриста
createdAt	Date	Дата створення користувача
updatedAt	Date	Дата останнього оновлення

Таблиця 2.15

Опис колекції Conversations

Поле	Тип даних	Опис
userId	ObjectId	Посилання на користувача
title	String	Назва розмови
messages	Array [ObjectId]	Масив повідомлень у розмові
createdAt	Date	Дата створення розмови
updatedAt	Date	Дата останнього оновлення

Таблиця 2.16

Опис колекції Messages

Поле	Тип даних	Опис
chatId	ObjectId	Посилання на розмову
parentId	ObjectId	Посилання на батьківське повідомлення
role	Enum (user, assistant)	Роль відправника
content	String	Текст повідомлення
children	Array[ObjectId]	Посилання на дочірні повідомлення
files	Array [FilePartItem]	Масив вкладень (файли, зображення)
createdAt	Date	Дата створення повідомлення
updatedAt	Date	Дата останнього оновлення

Таблиця 2.17

Опис колекції Bots

Поле	Тип даних	Опис
name	String	Назва бота
chromaCollection	String	Назва колекції бази знань, що використовується ботом
description	String	Опис призначення бота
botPrompt	String	Спеціальний промпт для бота
createdAt	Date	Дата створення бота

Продовження таблиця 2.17

updatedAt	Date	Дата останнього оновлення
-----------	------	---------------------------

Таблиця 2.18

Опис колекції Subscriptions

Поле	Тип даних	Опис
user	ObjectId	Посилання на користувача
plan	ObjectId	Посилання на тарифний план
status	Enum (active, canceled, paused, expired)	Статус підписки

Продовження таблиці 2.18.

startDate	Date	Дата початку підписки
endDate	Date	Дата завершення підписки
renewalDate	Date	Дата поновлення підписки
cancellationDate	Date	Дата скасування підписки
autoRenew	Boolean	Автоматичне продовження підписки
createdAt	Date	Дата створення
updatedAt	Date	Дата останнього оновлення

Таблиця 2.19

Опис колекції SubscriptionPlans

Поле	Тип даних	Опис
name	String	Назва тарифного плану
price	Number	Вартість тарифу
features	String[]	Перелік доступних функцій
trialPeriodDays	Number	Тривалість пробного періоду
description	String	Опис тарифного плану
createdAt	Date	Дата створення
updatedAt	Date	Дата останнього оновлення

Таблиця 2.20

Опис колекції VideoConsultations

Поле	Тип даних	Опис
user	ObjectId	Посилання на користувача
lawyer	ObjectId	Посилання на юриста
scheduledAt	Date	Запланована дата та час зустрічі
status	Enum (pending, approved, canceled, completed)	Статус консультації
roomCode	String	Код доступу до кімнати
notes	String	Додаткові примітки користувача
createdAt	Date	Дата створення
updatedAt	Date	Дата останнього оновлення

Таблиця 2.21

Опис колекції VectorCollections

Поле	Тип даних	Опис
name	String	Назва колекції
files	Array[File]	Масив файлів у колекції
createdAt	Date	Дата створення
updatedAt	Date	Дата останнього оновлення

Таблиця 2.22

Опис колекції Feedbacks

Поле	Тип даних	Опис
user	ObjectId	Посилання на користувача
message	ObjectId	Посилання на повідомлення
type	Enum (like, dislike)	Тип відгуку
tags	String[]	Перелік тегів відгуку
comment	String	Текстовий коментар
createdAt	Date	Дата створення

updatedAt	Date	Дата останнього оновлення
-----------	------	---------------------------

Таблиця 2.23

Опис колекції LegalTemplates

Поле	Тип даних	Опис
title	String	Назва шаблону
description	String	Короткий опис шаблону
content	String	Основний текст шаблону
category	String	Категорія документа (договір, скарга тощо)
createdAt	Date	Дата створення
updatedAt	Date	Дата останнього оновлення

Таблиця 2.24

Опис колекції Files

Поле	Тип даних	Опис
source	String	Джерело файлу
firstAdded	Date	Дата першого додавання
lastUpdated	Date	Дата останнього оновлення

Таблиця 2.25

Опис колекції LawyerProfiles

Поле	Тип даних	Опис
user	ObjectId	Посилання на користувача
firstName	String	Ім'я юриста
lastName	String	Прізвище юриста
middleName	String	По батькові
lawFirm	String	Назва юридичної компанії
specialization	String	Спеціалізація
licenseNumber	String	Номер ліцензії
yearsOfExperience	Number	Стаж роботи

bio	String	Біографія
certifications	Array[String]	Сертифікати
languages	Array[String]	Мови
createdAt	Date	Дата створення профілю
updatedAt	Date	Дата останнього оновлення

Таким чином, структура бази даних побудована на основі дванадцятти колекцій, що забезпечують повний цикл функціонування платформи: від реєстрації користувачів і взаємодії з чат-ботами до організації консультацій із юристами, управління підписками та обробки фінансових транзакцій. Структуру бази даних системи зображено на рисунку 2.3.

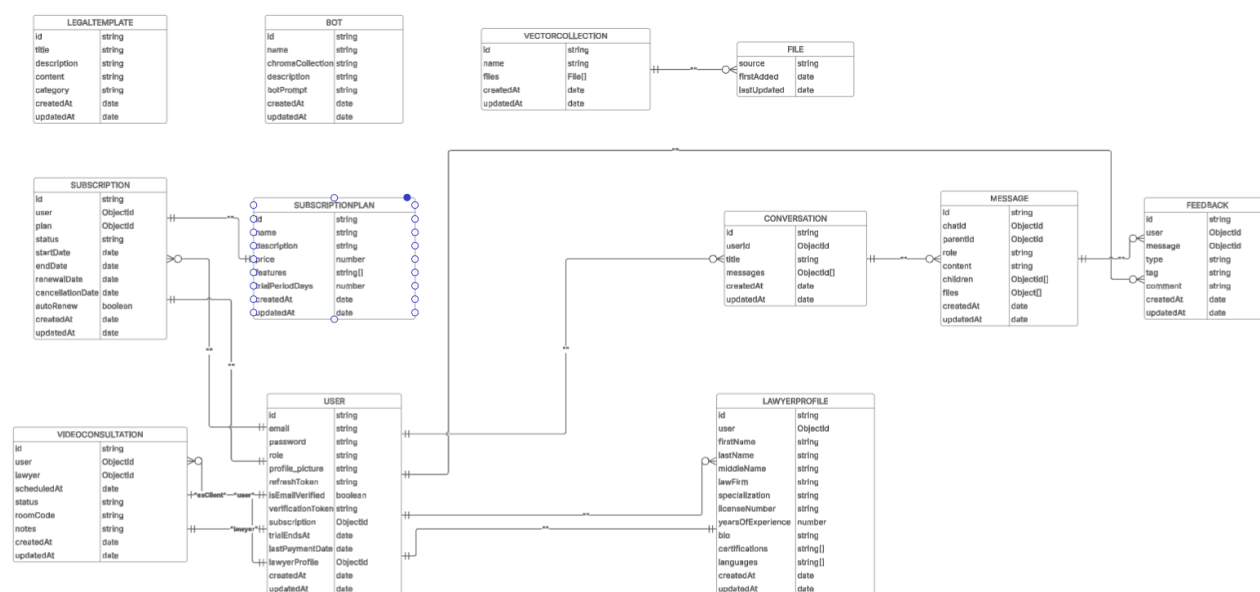


Рисунок 2.3 Структура бази даних.

2.3 Проектування та реалізація алгоритмів роботи платформи

Оскільки архітектурною основою платформи обрано клієнт-серверну модель, застосунок має чіткий поділ на серверну та клієнтську частини. Для організації обміну даними між ними використовується архітектурний стиль REST, який базується на концепції ресурсів та стандартних HTTP-запитах.

У межах REST-архітектури об'єкти системи представлені у вигляді ресурсів з унікальними URL-адресами, а взаємодія з ними здійснюється через такі HTTP-методи:

- GET – отримання даних із сервера;
- POST – створення нового запису в системі;
- PUT – повне оновлення наявного ресурсу;
- PATCH – часткове оновлення ресурсу;
- DELETE – видалення ресурсу.

Серверна частина платформи реалізована на базі фреймворку NestJS, який забезпечує обробку HTTP-запитів, взаємодію з базою даних MongoDB, виконання бізнес-логіки додатку та формування відповідей у форматі JSON. Завдяки цьому серверна частина гарантує коректну обробку даних та їх передачу клієнтській стороні.

Клієнтська частина платформи створена за допомогою Next.js, що дозволяє реалізувати серверний рендеринг сторінок для швидкого завантаження контенту і подальший перехід до односторінкової взаємодії (SPA) за допомогою React [38]. Після початкового завантаження користувач взаємодіє з динамічним інтерфейсом, який відправляє запити до серверного API для отримання або оновлення даних. Таким чином, комунікація у системі базується на уніфікованій моделі HTTP-запитів, що забезпечує надійність, гнучкість та масштабованість розробленої платформи.

Структурну схему взаємодії клієнта, серверу та бази даних наведено на рисунку 2.4.

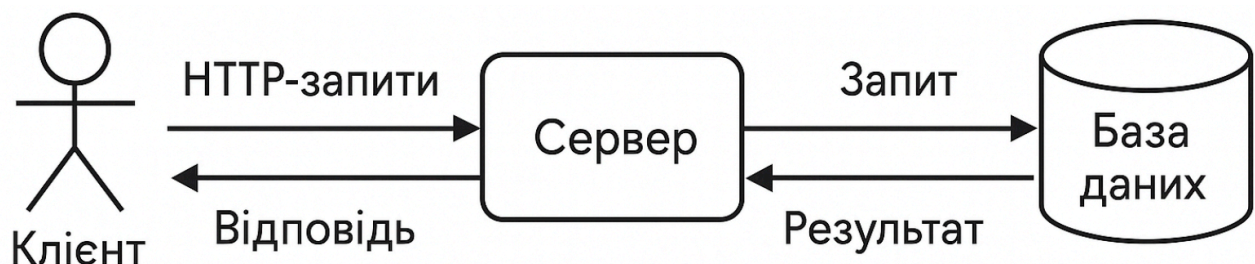


Рисунок 2.4. Структурна схема взаємодії клієнта, серверу та бази даних.

Для відображення загальної логіки роботи платформи було побудовано загальний алгоритм роботи системи (рис. 2.5), який демонструє основні потоки виконання процесів залежно від ролі користувача. У схемі відображено базові сценарії взаємодії для користувача, юриста та адміністратора, починаючи від процесу реєстрації та авторизації, продовжуючи взаємодією із чат-ботом, замовленням консультацій чи управлінням документами, і закінчуючи обробкою звернень, адмініструванням ботів та керуванням підписками. Загальний алгоритм дозволяє наочно відобразити послідовність основних дій у системі залежно від обраної ролі користувача, що спрощує розуміння логіки функціонування платформи та взаємозв'язків між її основними процесами.

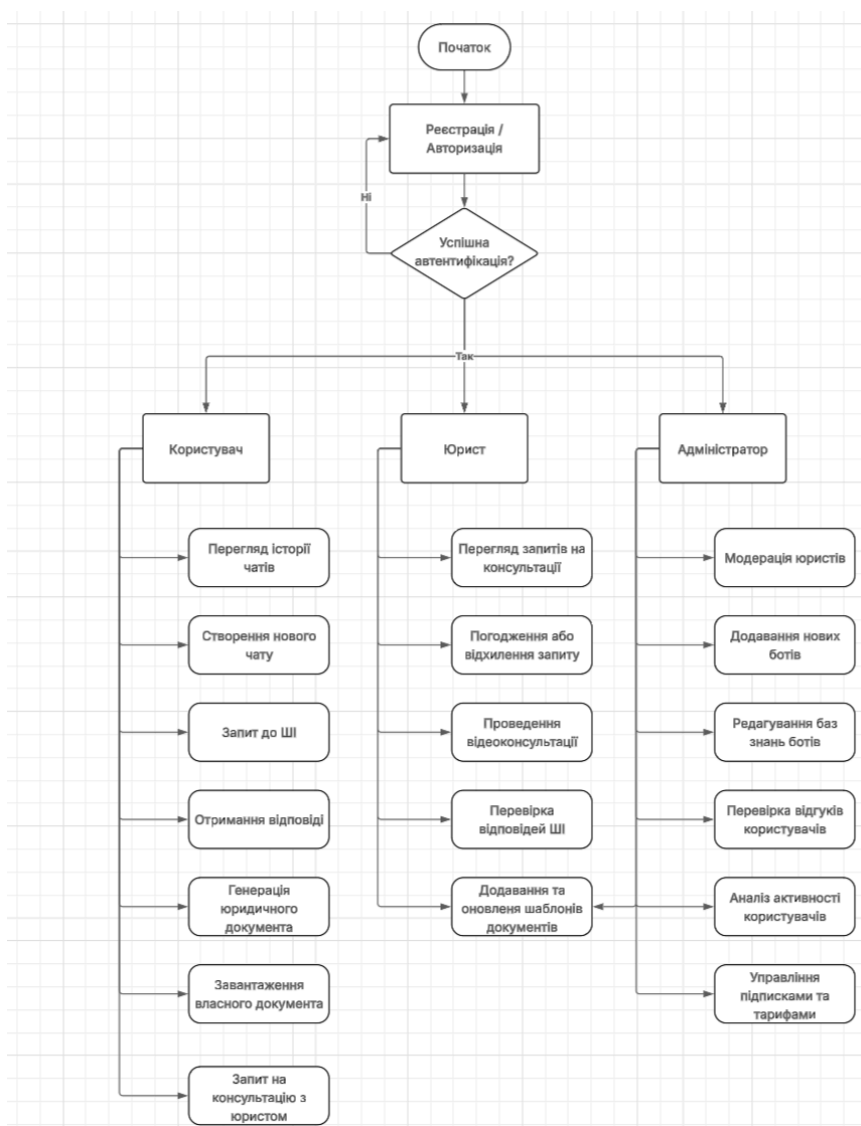


Рисунок 2.5 Загальний алгоритм роботи платформи юридичної допомоги з використанням ШІ

Склавши загальний алгоритм роботи платформи, доцільно також розробити окремі алгоритми для ключових процесів системи. Для представлення таких сценаріїв використовуються діаграми діяльності, які деталізують логіку виконання окремих дій. Такі діаграми дозволяють більш точно описати специфіку поведінки користувача та внутрішніх процесів, що значно полегшує подальшу розробку, тестування та підтримку системи. Для прикладу були побудовані діаграми для процесу «Авторизація користувача» (рис. 2.6) та «Взаємодія користувача з чат-ботом» (рис. 2.7), оскільки саме ці процеси є базовими для забезпечення доступу до функціоналу платформи та надання юридичних консультацій через штучний інтелект.

Діаграма діяльності «Авторизація користувача» (рис. 2.6) демонструє послідовність дій під час проходження автентифікації у системі. Користувач на клієнтській стороні вводить облікові дані та надсилає їх на сервер. Сервер перевіряє правильність даних, і у разі успіху генерує токен доступу, який повертається клієнту для подальшого використання. У разі помилкової авторизації сервер повертає повідомлення про помилку.

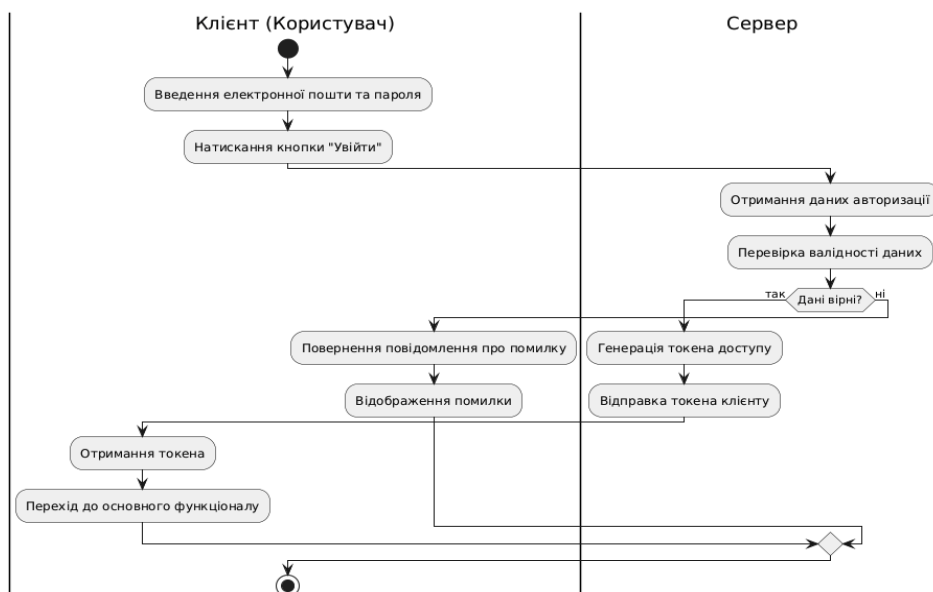


Рисунок 2.6 – Діаграма діяльності «Авторизація користувача»

Діаграма діяльності «Взаємодія користувача з чат-ботом ШІ» (рис. 2.7) описує процес обробки користувацького запиту у чаті. Після введення тексту повідомлення користувачем клієнт надсилає запит серверу. Сервер виконує валідацію запиту, здійснює пошук релевантної інформації у векторному сховищі та передає знайдені дані штучному інтелекту. На основі отриманої інформації ШІ формує відповідь, яка повертається користувачу через сервер.

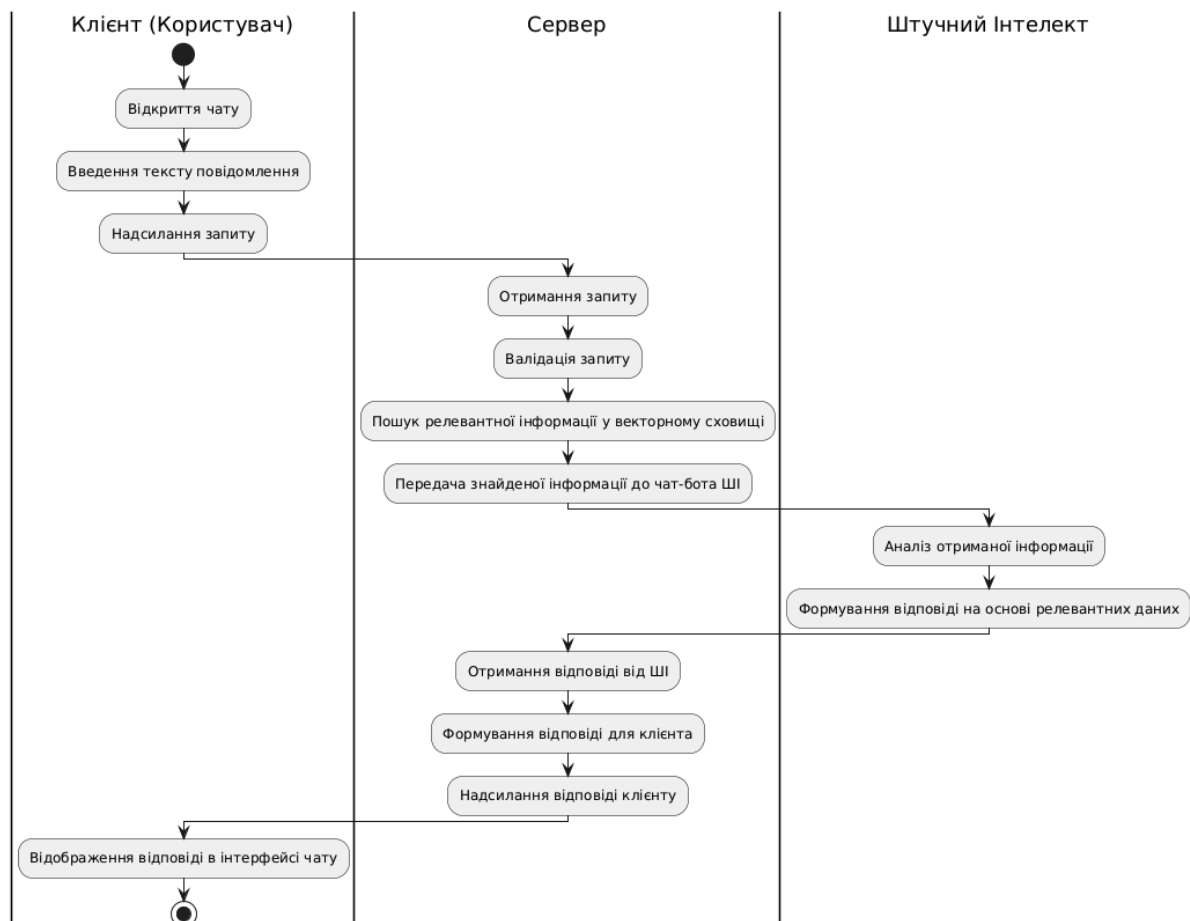


Рисунок 2.7 – Діаграма діяльності «Взаємодія користувача з чат-ботом ШІ»

2.4 Реалізація платформи для надання юридичної допомоги з штучним інтелектом

Реалізація платформи включає розробку кількох взаємопов'язаних частин — клієнтської, серверної та адміністративної панелі. Кожна з них створюється у вигляді окремого застосунку, що взаємодіють між собою через визначений інтерфейс API. Серверна частина платформи була реалізована із використанням фреймворку NestJS, який забезпечує побудову масштабованої та структурованої

серверної архітектури. Основною точкою входу у серверний застосунок є файл `main.ts`, у якому ініціалізується сервер за допомогою функції `bootstrap()`. На цьому етапі виконується налаштування основних сервісів, таких як глобальні валідаційні пайпи, префікс API, CORS-політика та конфігурація Swagger-документації для API.

Основна частина коду функції `bootstrap()` виглядає наступним чином:

```
const app = await NestFactory.create(AppModule);
app.setGlobalPrefix('api');
app.useGlobalPipes(new ValidationPipe());
app.enableCors();
const documentFactory = () => SwaggerModule.createDocument(app, config);
SwaggerModule.setup('api/docs', app, documentFactory);
const configService = app.get(ConfigService);
const port = configService.get<number>('PORT');
await app.listen(port || 3000);
```

Повний код реалізації ініціалізації серверного застосунку наведено у додатку А.

Крім того, важливою складовою конфігурації додатку є файл `configuration.ts`, який відповідає за зчитування і передачу основних налаштувань середовища, таких як параметри бази даних, секрети токенів авторизації та інші ключові змінні. Основний код файлу `configuration.ts` виглядає наступним чином:

```
export default () => ({
  port: parseInt(process.env.PORT, 10) || 3000,
  databaseUrl: process.env.MONGODB_URI,
  jwtSecret: process.env.JWT_ACCESS_SECRET,
  jwtExpiration: process.env.JWT_ACCESS_EXPIRATION,
  jwtRefreshSecret: process.env.JWT_REFRESH_SECRET,
  jwtRefreshExpiration: process.env.JWT_REFRESH_EXPIRATION,
  saltRounds: parseInt(process.env.SALT_ROUNDS, 10) || 10,
});
```

Ключовим елементом структури серверного застосунку є файл `app.module.ts`, який відповідає за імпорт основних модулів, налаштування бази даних, підключення конфігурації та реєстрацію контролерів і сервісів.

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				54
Змн.	Арк.	№ докум.	Підпис	Дата		

```

@Module({
  imports: [
    ConfigModule.forRoot({
      isGlobal: true,
      load: [configuration],
    }),
    MongooseModule.forRootAsync({
      imports: [ConfigModule],
      useFactory: async (configService: ConfigService) => ({
        uri: configService.get<string>('databaseUrl'),
      }),
      inject: [ConfigService],
    }),
    UserModule,
    AuthModule,
    MailModule,
    ConversationModule,
    VectorStoreModule,
    LlmModule,
    MessageModule,
    SubscriptionModule,
    SubscriptionPlanModule,
    BotsModule,
    VideoConsultationModule,
    LawyerProfileModule,
    TemplateModule,
    FeedbackModule,
  ],
  controllers: [AppController],
  providers: [AppService, VideoGateway],
})
export class AppModule {}

```

Наступним кроком розглянемо реалізацію методу взаємодії з мовною моделлю в сервісі LlmService. Він відповідає за динамічне генерувати відповіді на юридичні запити з урахуванням контексту всієї попередньої розмови та релевантних матеріалів із векторного сховища. У процесі виклику методу до мовної моделі надсилається сформований запит, який містить як історію діалогу користувача, так і додаткову контекстну інформацію, таку як знайдені документи або прикріплені файли. Запит структурується так, щоб модель отримувала максимально повний контекст для побудови осмисленої й релевантної відповіді

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				55
Змн.	Арк.	№ докум.	Підпис	Дата		

і після цього відповідь генерується у стрімінговому режимі, що дозволяє отримувати її частинами у реальному часі та забезпечує динамічну й контекстну консультацію[]]. Повний код сервісу LlmService наведено у додатку А.

```

async *generateStream( chatId: Types.ObjectId, prompt: string collectionName = 'documents-test', files?:
FilePartItem[]) {
    const conversationMessages = await this.messageService.getMessagesByChat(chatId.toString());
    const conversationContext = conversationMessages.map((msg) => `${msg.role}: ${msg.content}`).join("\n");
    const messages: any[] = [];
    if (files && files.length > 0) {
        messages.push(new HumanMessage({content: files}), );
        messages.push(
            new HumanMessage({content: `Conversation context: ${conversationContext} Question: ${prompt} Answer: `});
    } else {
        const vectorResults = await this.vectorStoreService.similaritySearch(prompt,8,collectionName);
        const vectorContext = vectorResults.map((doc) => doc.pageContent).join("\n\n");
        const augmentedPrompt = `Conversation context: ${conversationContext} Document context: ${vectorContext}
Question: ${prompt} Answer: `;
        messages.push(new HumanMessage({ content: augmentedPrompt }));
    }
    const streamIter = await this.agent.stream({ messages }, { streamMode: 'messages',configurable: { thread_id:
chatId.toString() }, },);
    for await (const [msg] of streamIter) {
        if (!isAIMessageChunk(msg)) continue;
        if (typeof msg.content === 'string') {
            yield msg.content;
        } else {
            for (const chunk of msg.content as MessageContentComplex[]) {
                if (chunk.type === 'text') {
                    yield chunk.text;
                }
            }
        }
    }
}

```

Для отримання якісних векторних представлень тексту у платформі реалізовано спеціальний клас OpenAIEmbeddingFunction. Він виступає адаптером між системою та моделлю ембедінгів від OpenAI, забезпечуючи

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				56
Змн.	Арк.	№ докум.	Підпис	Дата		

генерацію векторів для будь-яких текстових даних за допомогою моделі text-embedding-ada-002. Такий підхід дозволяє швидко й ефективно конвертувати текст у формат, придатний для зберігання й пошуку у векторній базі даних.

```
class OpenAIEmbeddingFunction implements IEmbeddingFunction {
  private embeddings: OpenAIEmbeddings;

  constructor(apiKey: string) {
    this.embeddings = new OpenAIEmbeddings({
      openAIApiKey: apiKey,
      modelName: 'text-embedding-ada-002',
    });
  }

  async generate(texts: string[]): Promise<number[][]> {
    return this.embeddings.embedDocuments(texts);
  }
}
```

Оскільки для роботи мовної моделі у системі використовується додатковий контекст із документів, наступним кроком стала реалізація сервісу для роботи з векторним сховищем. Сервіс VectorStoreService забезпечує збереження, пошук та обробку текстових документів у векторному форматі за допомогою ChromaDB та моделей ембедінгу OpenAI. Він дозволяє створювати колекції, додавати до них PDF-документи, а також здійснювати пошук релевантної інформації за запитами користувачів. Саме тому для ілюстрації ключової логіки нижче розглядається метод додавання файлу у колекцію, тоді як повна реалізація цього сервісу наведена у додатку А.

```
async addPdfToCollection(pdfBuffer: Buffer, fileName: string, collectionName = 'documents-test') {
  try {
    const data = await pdfParse(pdfBuffer);
    const text = data.text;
    const textSplitter = new RecursiveCharacterTextSplitter({chunkSize: 1000, chunkOverlap: 200,});
    const docs = await textSplitter.createDocuments([text]);

    const collection = await this.getCollectionByName(collectionName);
    const ids = docs.map(() => uuidv4());
    const nowIso = new Date().toISOString();
    const documents = docs.map((d) => d.pageContent);
```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				57
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const metadatas = docs.map(() => ( {source: fileName,addedAt: nowIso,updatedAt: nowIso,}));
await collection.add({ ids, documents, metadatas });

const now = new Date();
const updResult = await this.vectorCollectionModel.updateOne({ name: collectionName, 'files.source': fileName },
  { $set: { 'files.$.lastUpdated': now } },);
if (updResult.matchedCount === 0) {
  await this.vectorCollectionModel.updateOne(
    { name: collectionName },
    {
      $push: {
        files: {
          source: fileName,
          firstAdded: now,
          lastUpdated: now,
        },
      },
    },
  );
}

return { success: true, message: 'PDF added to collection successfully' };
} catch (error) {
  throw new Error(`Failed to add PDF to collection: ${error.message}`);
}
}

```

Цей метод додає PDF-документ до векторного сховища, де спочатку парсить файл і витягує текст, розбиває його на фрагменти для ефективної векторизації, а потім зберігає ці фрагменти разом із метаданими у ChromaDB. Після цього інформація про файл оновлюється або створюється у MongoDB, що дозволяє повністю автоматизувати процес обробки та збереження PDF-документів у системі.

Після розгляду сервісів для взаємодії з мовною моделлю та векторним сховищем, необхідно також розглянути сервіс, що відповідає за управління чатами користувачів. Сервіс ConversationService реалізує логіку створення нових чатів, додавання повідомлень, пошуку розмов за змістом повідомлень та видалення чатів разом із пов'язаними повідомленнями. Він забезпечує збереження історії комунікації користувача у системі та підтримує базову

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				58
Змн.	Арк.	№ докум.	Підпис	Дата		

навігацію між розмовами. Повна реалізація сервісу ConversationService наведена у додатку А.

```
@Injectable()
export class ConversationService {
  constructor(
    @InjectModel(Conversation.name)
    private conversationModel: Model<ConversationDocument>,
    @InjectModel(Message.name)
    private readonly messageModel: Model<MessageDocument>,
  ) {}

  async createConversation(userId: string): Promise<ConversationDocument> { ... }
  async getConversationById(conversationId: string, userId: string) { ... }
  async addMessages(conversationId: Types.ObjectId, messageIds: Types.ObjectId[]) { ... }
  async getUserConversations(userId: string, page = 1, limit = 10) { ... }
  async searchConversationByMessageContent(userId: string, searchText: string, page = 1, limit = 10) { ... }
  async deleteConversation(conversationId: string, userId: string) { ... }
}
```

Після розгляду основних сервісів платформи можемо перейти до аналізу контролера, який забезпечує взаємодію користувача із платформою через HTTP-запити. Оскільки контролер має доволі велику кількість маршрутів і функціональності, у межах даного опису буде розглянуто лише частину, що відповідає за обробку повідомлень у чатах.

Основна структура ConversationController реалізація маршруту для обробки повідомлень виглядає наступним чином: }

```
@Controller('conversation')
export class ConversationController {
  constructor(private readonly llmService, private readonly messageService, private readonly conversationService,
    private readonly botService,
  ) {}

  @Get('search')
  async searchConversations( @Query('query') query,@Query('page') page,@Query('limit') limit, ) {...}

  @Get('/:conversationId')
  async getConversation(@Param('conversationId') conversationId,) {...}

  @Post()
  async stream(@Body() body,@Res() res,) { ... }

  @Get()
```

```

async getUserConversations(@Query('page') page,@Query('limit') limit,) {...}
@Delete('/:conversationId')
async deleteConversation(@Param('conversationId') conversationId,) {...}
}

```

Повна реалізація всіх маршрутів ConversationController наведена у додатку А. Найголовнішим маршрутом цього контролера є POST-запит /conversation. Він відповідає за прийом запиту користувача та надсилає запиту до ШІ після чого відповідь надсилає у вигляді потоку даних (SSE) та зберігає відповідь як повідомлення у базі даних. Таким чином, реалізується процес ведення діалогу користувача з чат-ботом у реальному часі.

Клієнтська частина платформи реалізована на основі фреймворку Next.js 15, який забезпечує підтримку серверного та клієнтського рендерингу. Структура проекту організована у папці src, де зберігаються основні модулі додатку такі як app для побудови маршрутів і сторінок, components для створення інтерфейсних елементів, api для організації запитів до серверної частини, hooks для зберігання користувацьких хуків та libs для допоміжних бібліотек.

У кореневій частині src розташовано файл middleware.ts, який забезпечує перевірку авторизації при доступі до певних маршрутів додатку. Базові налаштування середовища проекту зосереджені у файлах next.config.ts, tsconfig.json, package.json та інших конфігураційних файлах.

Файл middleware.ts відповідає за перенаправлення користувачів залежно від стану їх автентифікації, забезпечуючи доступ лише до відповідних сторінок (/sign-in та /chat). Реалізація виглядає наступним чином:

```

import { auth } from './libs/auth';
export default auth(req => {
  if (req.auth && !req.auth.isEmailVerified && req.nextUrl.pathname !== '/verify-email') {
    return Response.redirect(new URL('/check-email', req.nextUrl.origin));
  }
  if (req.auth && req.nextUrl.pathname === '/sign-in') {
    return Response.redirect(new URL('/chat', req.nextUrl.origin));
  }

  if (!req.auth && req.nextUrl.pathname !== '/sign-in') {

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				60
Змн.	Арк.	№ докум.	Підпис	Дата		

```
const newUrl = new URL('/sign-in', req.nextUrl.origin);
return Response.redirect(newUrl);
}
});
```

```
export const config = {
  matcher: ['/sign-in', '/chat/:path*', '/profile'],
};
```

Для організації автентифікації користувачів у клієнтській частині використовується бібліотека NextAuth версії 5. Файл `libs/auth.ts` містить основну конфігурацію авторизації, де застосовується `CredentialsProvider` для входу через електронну пошту та пароль. Система підтримує оновлення токенів доступу через функцію `refreshAccessToken`, що дозволяє автоматично отримувати новий токен після закінчення терміну дії попереднього. Реалізовані колбеки `jwt` та `session` забезпечують синхронізацію даних користувача між токеном і сесією. Основні фрагменти реалізації наведено нижче, а повна реалізація знаходиться в додатку А.

```
import NextAuth, { NextAuthConfig } from 'next-auth';
import CredentialsProvider from 'next-auth/providers/credentials';
import { AuthService } from '@api/services/AuthService';
import { jwtDecode } from 'jwt-decode';

async function refreshAccessToken(token) { ... }

export const config = {
  providers: [
    CredentialsProvider({
      name: 'Credentials',
      credentials: { email: { label: 'Email' }, password: { label: 'Password' } },
      async authorize(credentials) { ... }
    }),
  ],
  pages: {
    signIn: '/sign-up',
    error: '/sign-up',
  },
  callbacks: {
```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				61
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    jwt: async ({ token, account, user }) => { ... },
    session: async ({ session, token }) => { ... },
  },
  secret: process.env.AUTH_SECRET,
} satisfies NextAuthConfig;

export const { handlers, auth, signIn, signOut } = NextAuth(config);

```

Для ефективної взаємодії клієнтської частини із серверною у платформі використовується бібліотека TanStack Query. Вона забезпечує зручне отримання, кешування, оновлення та синхронізацію даних між сервером і клієнтом без необхідності ручного керування станом запитів. Для реалізації запитів та мутацій створені окремі користувацькі хуки. Наприклад:

Хук `useConversations` використовується для отримання списку чатів користувача із серверу з підтримкою кешування та автоматичного оновлення даних:

```

import { ConversationService } from '@api/services/ConversationService';
import { Conversations } from '@api/types/conversation';
import { useQuery } from '@tanstack/react-query';

export const useConversations = (page?: number, limit?: number) => {
  return useQuery<Conversations>({
    queryKey: ['conversations', page, limit],
    queryFn: () => ConversationService.getConversations(page, limit),
    staleTime: 5 * 60 * 1000,
  });
};

```

Хук `useVerifyEmail` використовується для виконання мутації, пов'язаної з підтвердженням електронної пошти користувача:

```

import { useMutation, UseMutationOptions } from '@tanstack/react-query';
import { AuthService } from '@api/services/AuthService';

export const useVerifyEmail = (options?: UseMutationOptions<unknown, unknown, string>) => {
  return useMutation({
    mutationFn: (token: string) => AuthService.VerifyEmail(token),
    ...options,
  });
};

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				62
Змн.	Арк.	№ докум.	Підпис	Дата		

};

Використання TanStack Query дозволяє мінімізувати навантаження на сервер за рахунок розумного кешування результатів запитів, автоматичного оновлення даних після змін, а також спрощує обробку помилок та станів завантаження у клієнтському додатку.

Висновки до розділу 2

У другому розділі було детально розглянуто етапи проєктування та побудови архітектури платформи для надання юридичної допомоги з використанням сучасних інструментів і технологій. В результаті аналізу вимог сформовано логічну структуру системи, яка забезпечує чітку взаємодію між клієнтською та серверною частинами, а також зовнішніми сервісами на базі штучного інтелекту.

Особливу увагу приділено розробці моделі бази даних, яка оптимізована для зберігання інформації про користувачів, консультації, документи та історію взаємодії в системі. Проєктування структури даних та її візуалізація у вигляді діаграм дозволили визначити основні сутності платформи та відобразити їхні взаємозв'язки.

У розділі також описано ключові програмні компоненти, алгоритми обробки запитів користувачів і принципи інтеграції зі сторонніми AI-сервісами, зокрема з використанням RAG-підходу для підвищення релевантності відповідей. Деталізовано процеси додавання документів у векторне сховище, пошуку інформації, організації консультацій і захисту даних користувачів.

Впровадження обраного технологічного стеку та методів автоматизації дозволило реалізувати ефективну, масштабовану й безпечну платформу, яка відповідає заявленим вимогам і забезпечує високий рівень сервісу для кінцевих користувачів.

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				63
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПЛАТФОРМОЮ ДЛЯ НАДАННЯ ЮРИДИЧНОЇ ДОПОМОГИ З ШТУЧНИМ ІНТЕЛЕКТОМ

3.1 Порядок встановлення та налаштування параметрів додатку

Платформа для надання юридичної допомоги побудована із декількох основних компонентів, кожен з яких має окрему логіку роботи та вимагає індивідуального налаштування й розгортання. Для наочного представлення структури системи, а також візуалізації процесу її впровадження, була створена відповідна архітектурна діаграма. Такий підхід дозволяє чітко зрозуміти взаємозв’язки між складовими платформи, забезпечити гнучкість масштабування й спростити подальшу підтримку та розвиток проєкту (рис. 3.1).

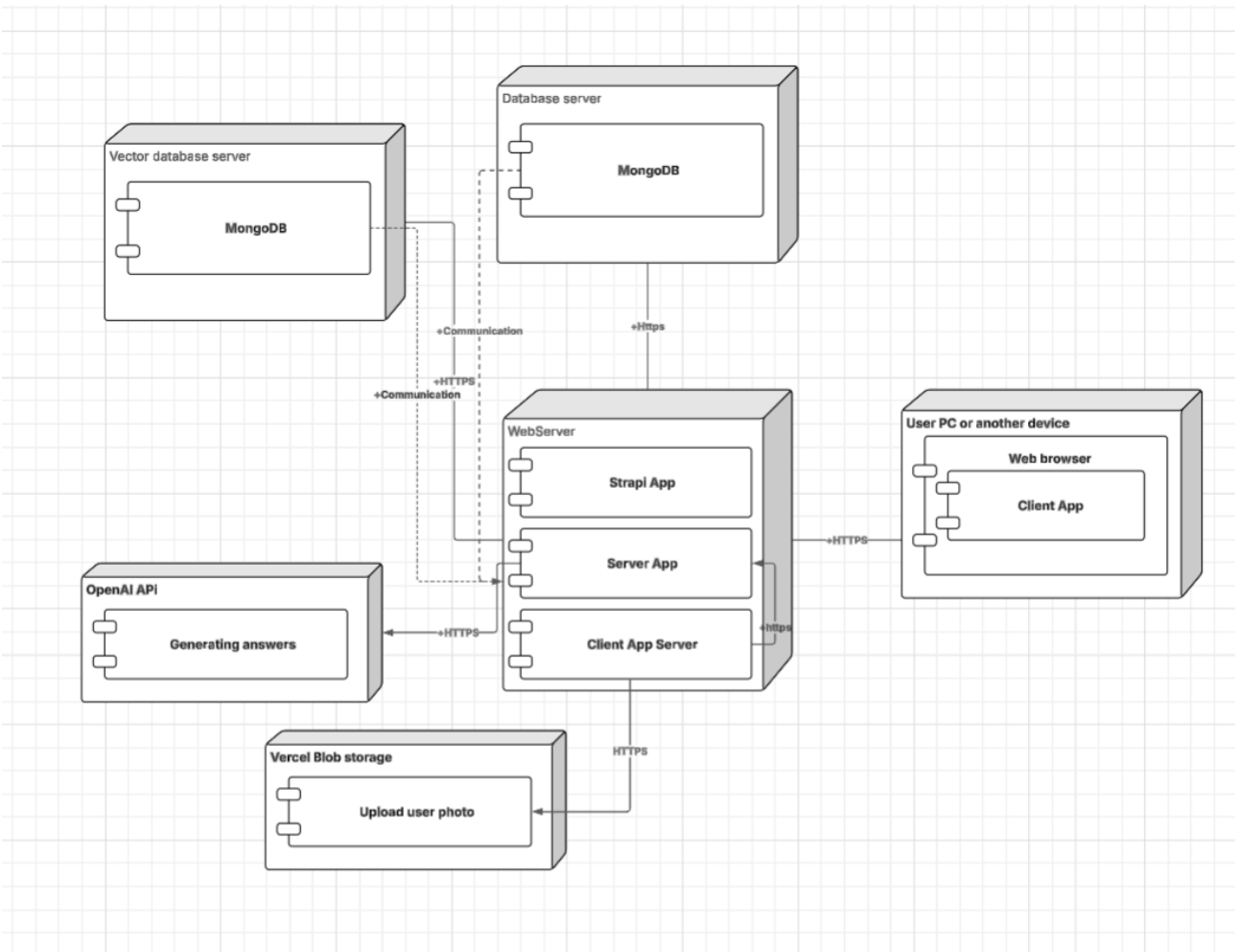


Рисунок 3.1 – Діаграма розгортання

На діаграмі розгортання представлена архітектура платформи для надання юридичних консультацій із використанням штучного інтелекту, де ключовими

компонентами є клієнтсько-серверна взаємодія, велика мовна модель від OpenAI, векторне сховище, а також зовнішні сервіси для оплати та зберігання даних користувача.

Сервер бази даних побудований на базі MongoDB і відповідає за зберігання основної інформації платформи, такої як даних користувачів, історії консультацій, підписок і тощо. Взаємодія між сервером бази даних і вебсервером здійснюється через захищений протокол, що гарантує цілісність і безпеку даних.

Вебсервер виконує центральну роль у роботі платформи. На ньому розгорнуті три основні компоненти. Strapi App [40], який забезпечує керування підписками та обробку платежів, Server App, що реалізує основну бізнес-логіку та інтеграцію із зовнішніми сервісами, та Client App Server, який відповідає за клієнтську частину та її взаємодію з користувачами. Всі ці компоненти об'єднані єдиною інфраструктурою і спілкуються між собою через протокол HTTPS.

Особливу роль у системі відіграє векторне сховище ChromaDB, що використовується для зберігання векторизованих представлень документів [41]. Завдяки цьому компоненту реалізується підхід Retrieval-Augmented Generation, де мовна модель отримує доступ до релевантних фрагментів інформації, що дозволяє давати користувачам точні та обґрунтовані відповіді з урахуванням внутрішньої бази знань платформи.

Для генерації відповідей використовується OpenAI API у поєднанні з бібліотекою langchain. Серверна частина платформи за допомогою langchain організовує логіку формування розширеного запиту до великої мовної моделі, де спочатку здійснюється пошук релевантної інформації у векторному сховищі ChromaDB, після чого знайдені фрагменти інтегруються у запит до OpenAI. Завдяки цьому, відповідь, яку отримує користувач, не лише базується на загальних знаннях моделі, а й враховує специфічні дані, що містяться у внутрішній базі знань платформи.

Зберігання фото користувача, яке завантажує користувач, реалізовано за допомогою хмарного сховища Vercel Blob Storage [42]. Цей сервіс дозволяє

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				65
Змн.	Арк.	№ докум.	Підпис	Дата		

безпечно передавати та зберігати медіафайли незалежно від навантаження на основний сервер.

Кінцевий користувач взаємодіє з платформою через веббраузер або мобільний додаток, підключаючись до клієнтської частини застосунку через протокол HTTPS. Такий підхід гарантує захищеність персональних даних та комфорт роботи для користувачів.

Завдяки такій структурі платформа забезпечує надійність, масштабованість і гнучкість, даючи змогу легко інтегрувати нові сервіси, підвищувати рівень автоматизації та ефективно використовувати можливості штучного інтелекту у сфері юридичних консультацій.

Для коректної роботи платформи потрібно встановити MongoDB як сервер бази даних і налаштувати підключення до неї. Окремо необхідно розгорнути ChromaDB на сервері або в контейнері, бо саме цк векторне сховище забезпечує швидкий пошук релевантної інформації для інтеграції з мовною моделлю. На вебсервері рекомендується використовувати операційну систему Linux, після чого встановлюється Node.js [43] версії 20 або вище. Далі розгортається основний бекенд на NestJS і клієнтська частина на Next.js. Для захищеної взаємодії між усіма модулями потрібно налаштувати HTTPS і підключити доменне ім'я.

Всі ключові параметр налаштувань середовищ для серверної та клієнтської частин зберігається у env фалах .

Файл .env для клієнта:

- NEXT_PUBLIC_BACKEND_URL — адреса серверної частини;
- NEXT_PUBLIC_VIDEO_CONSULTATION_URL — URL для відеоконсультацій;
- AUTH_SECRET, AUTH_SESSION_MAX_AGE — параметри для авторизації;

Файл .env для бекенду:

- PORT — порт запуску серверу;
- MONGODB_URI — підключення до бази даних MongoDB;

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				66
Змн.	Арк.	№ докум.	Підпис	Дата		

- `JWT_ACCESS_SECRET`, `JWT_REFRESH_SECRET`, `JWT_ACCESS_EXPIRATION`, `JWT_REFRESH_EXPIRATION` — параметри для JWT-аутентифікації;
- `OPENAI_API_KEY` — ключ для доступу до OpenAI;
- `CHROMA_DB_URL`, `CHROMA_DB_COLLECTION` — налаштування для підключення до ChromaDB;
- `SMTP_HOST`, `SMTP_PORT`, `SMTP_SECURE`, `SMTP_USER`, `SMTP_PASS` — параметри для надсилання електронної пошти;
- `BLOB_READ_WRITE_TOKEN` — токен для роботи з Vercel Blob Storage;
- `STRIPE_SECRET_KEY`, `STRIPE_WEBHOOK_SECRET` — ключі для інтеграції Stripe;

Також на платформі налаштована автоматизація процесу розгортання реалізована через CI/CD-процес на GitHub Actions [44]. Під час кожного оновлення коду автоматично виконуються linting через eslint та prettier [45-46], перевірка типів, запуск тестів, а після успішного проходження всіх перевірок відбувається деплой на хостинг. На рисунку 3.2 показано приклад CI/CD workflow, де відображені всі основні етапи автоматичної перевірки та розгортання коду.

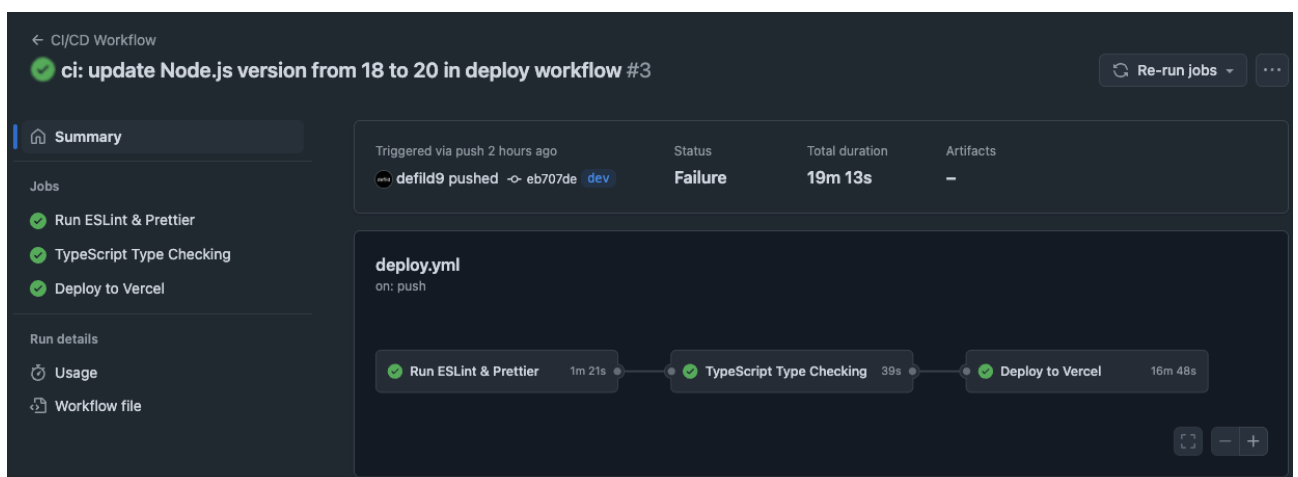


Рисунок 3.2 – Приклад роботи CI/CD-процесу на GitHub Actions для деплою

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				67
Змн.	Арк.	№ докум.	Підпис	Дата		

3.2 Структура інтерфейсу та порядок взаємодії з платформою для надання юридичної допомоги з штучним інтелектом

Після переходу на платформу користувач одразу потрапляє на головну сторінку, яка є стартовою точкою взаємодії з сервісом. У верхній частині розміщена навігаційна панель із логотипом платформи, посиланнями на основні розділи сайту та кнопками для входу чи реєстрації. Незалежно від того, чи авторизований користувач, інтерфейс забезпечує легкий доступ до основних можливостей — від перегляду інформації про послуги до швидкого старту використання сервісу. Головна сторінка також містить коротке пояснення переваг платформи та закликає користувача розпочати взаємодію (рис. 3.3).

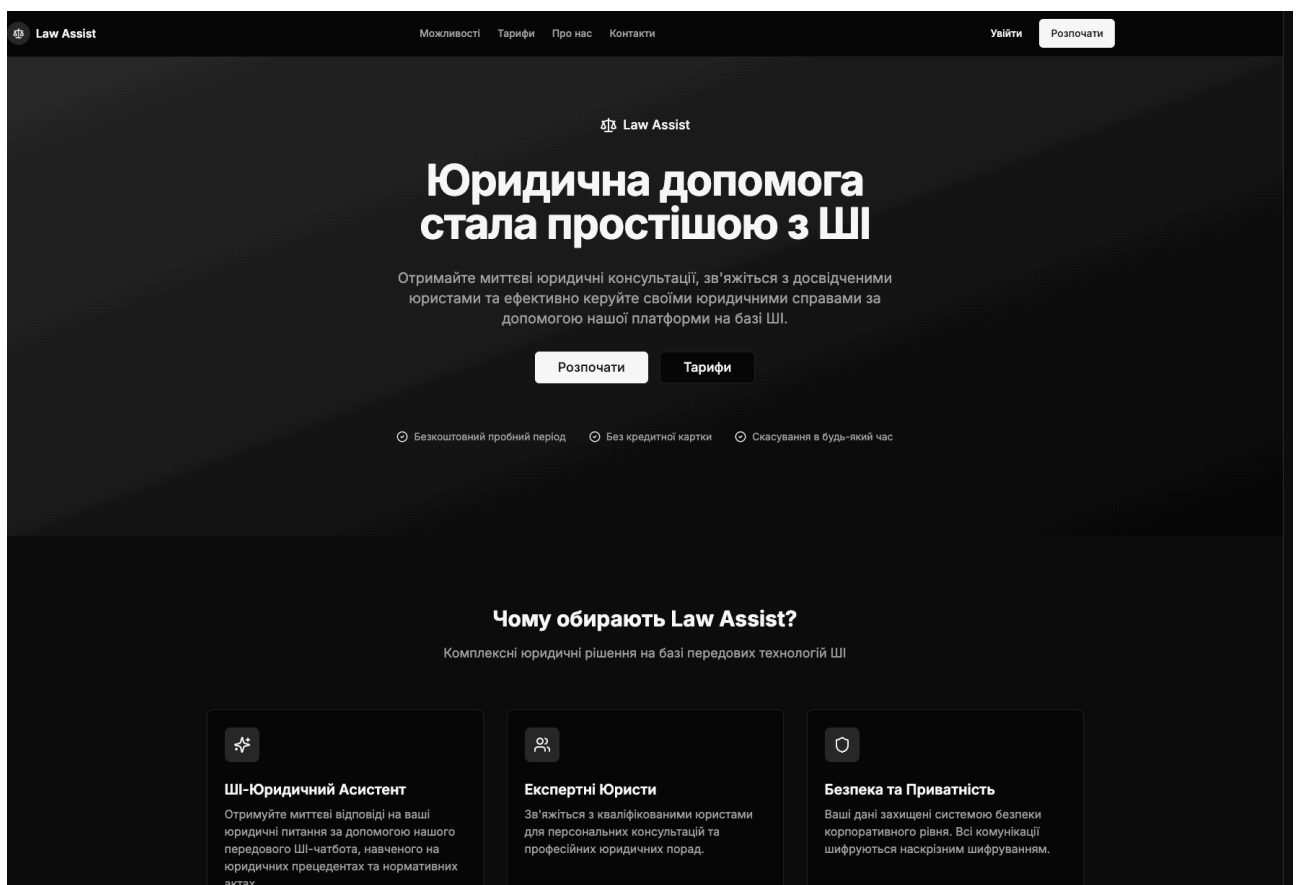


Рисунок 3.3 – Головна сторінка платформи для надання юридичної допомоги з використанням штучного інтелекту

Для того щоб користувач міг повноцінно взаємодіяти з платформою, необхідно пройти авторизацію. У випадку, якщо акаунт відсутній, можна швидко зареєструватися безпосередньо з відповідної сторінки. На рисунках 3.4 та 3.5 показано процес створення нового облікового запису, де потрібно ввести

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				68
Змн.	Арк.	№ докум.	Підпис	Дата		

електронну адресу та пароль, після чого система дозволяє увійти до особистого кабінету.

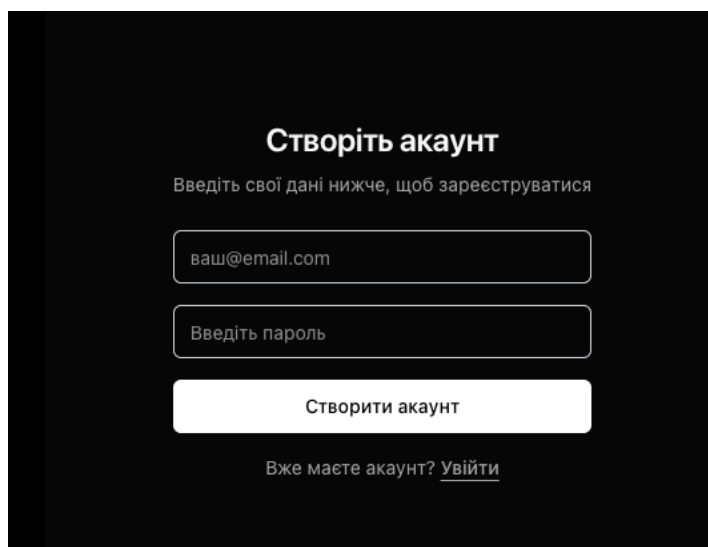


Рисунок 3.4 – Сторінка реєстрації нового користувача.

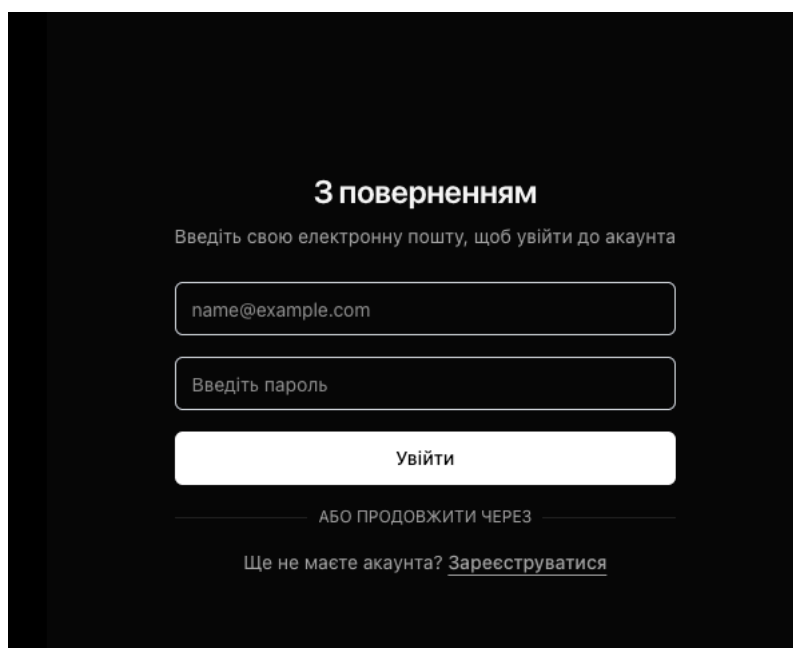


Рисунок 3.5 – Сторінка авторизація користувача.

Після авторизації користувач потрапляє на сторінку з чат-ботом, де може поставити будь-яке юридичне питання штучному інтелекту. Після надсилання питання користувач отримує відповідь від ШІ, яку за потреби можна скопіювати або залишити відгук щодо якості отриманої консультації (рисунки 3.6, 3.7).

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				69
Змн.	Арк.	№ докум.	Підпис	Дата		

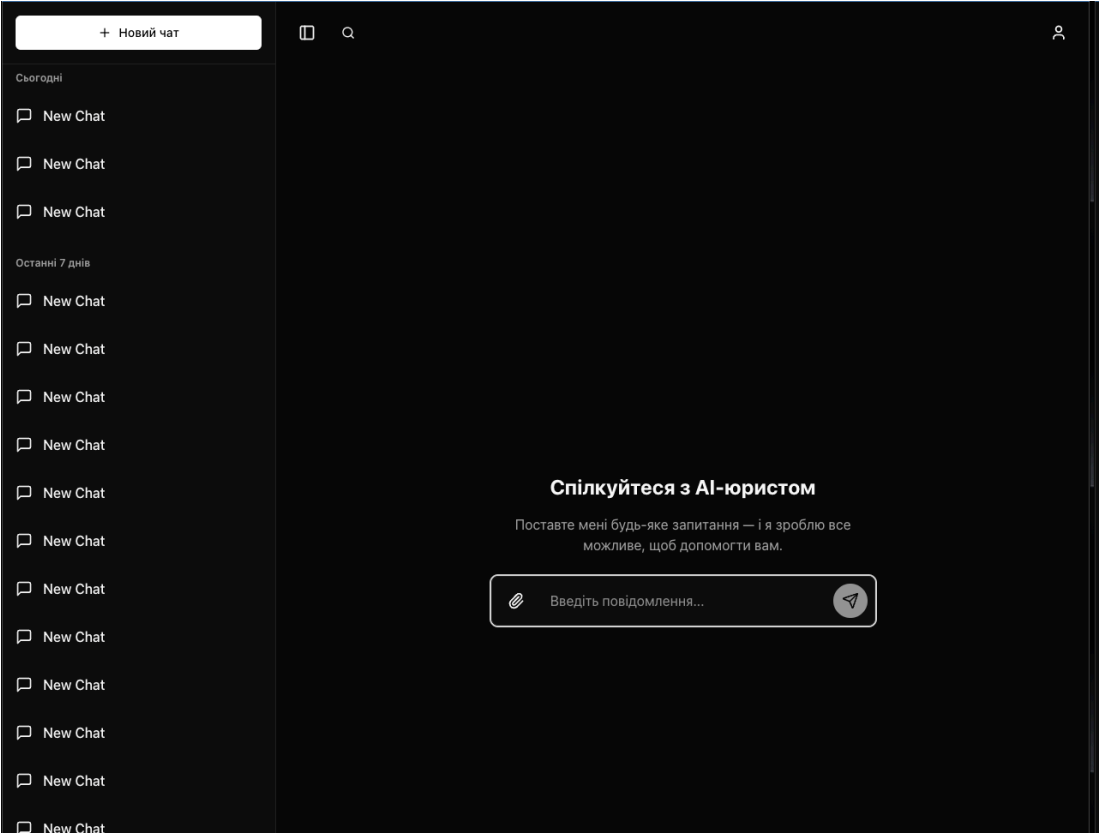


Рисунок 3.6 – Сторінка чат-бота.

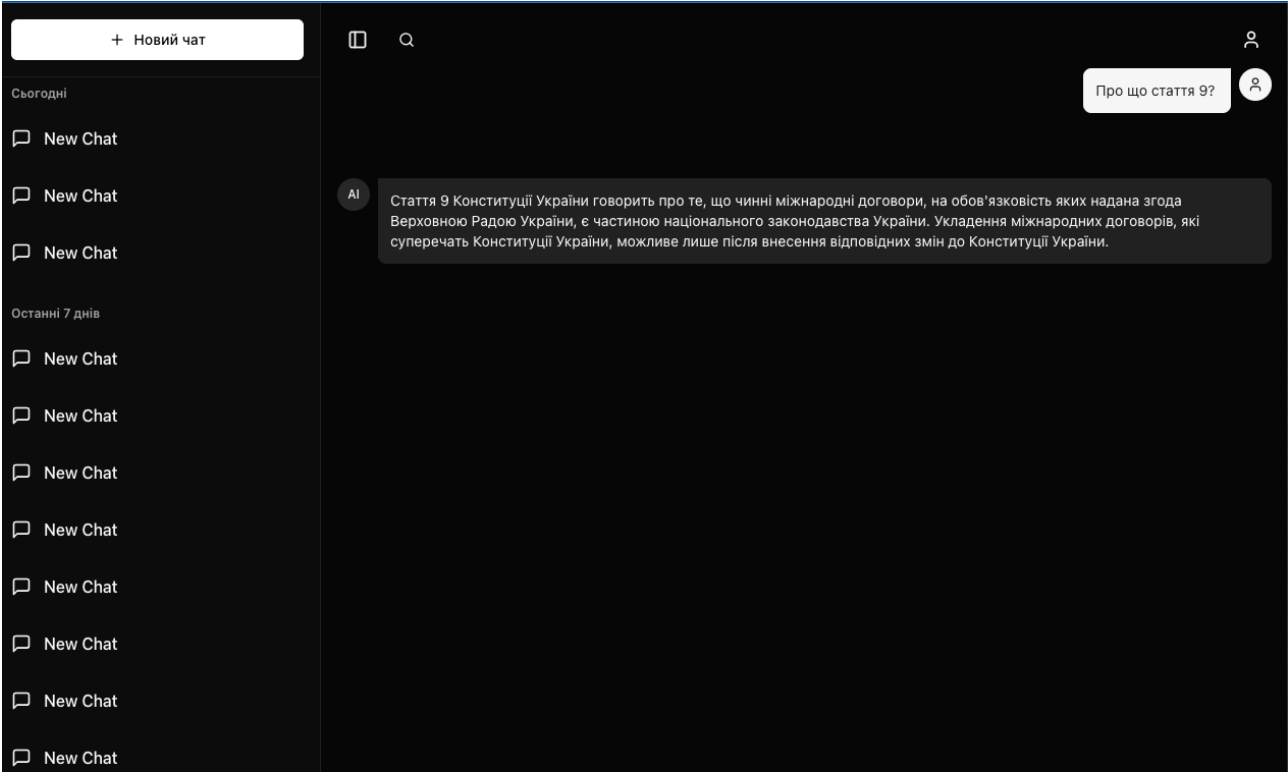


Рисунок 3.7 – Відображення відповіді ШІ на юридичне питання у чаті

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

Також користувач має можливість завантажити текстовий документ або фото документа для аналізу. Система розпізнає вміст файлу та надає консультацію на його основі (рис 3.8).

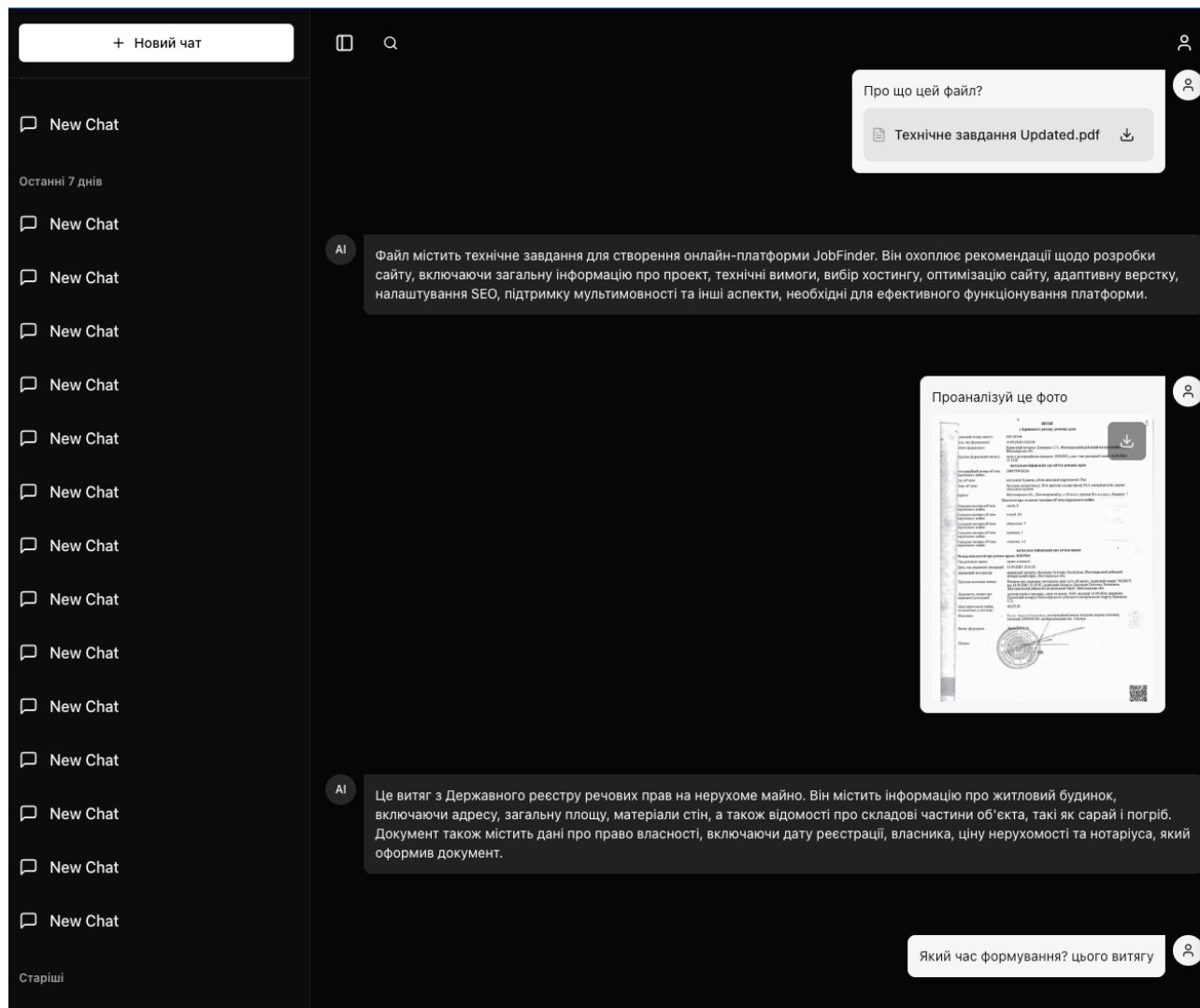


Рисунок 3.8 – Аналіз текстового файлу та фото документа в чаті з ШІ

У чат-боті також реалізована можливість генерації юридичних документів. Для цього користувач повинен надіслати відповідний запит на створення, наприклад, договору купівлі-продажу. Якщо деякі обов'язкові дані для формування документа не були вказані у запиті, штучний інтелект автоматично запитує в користувача необхідну інформацію, надаючи перелік полів, які потрібно заповнити. Після того як користувач надає усі необхідні відомості, чат-бот формує і відправляє готовий документ, який можна використовувати за призначенням (рис 3.9).

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				71
Змн.	Арк.	№ докум.	Підпис	Дата		

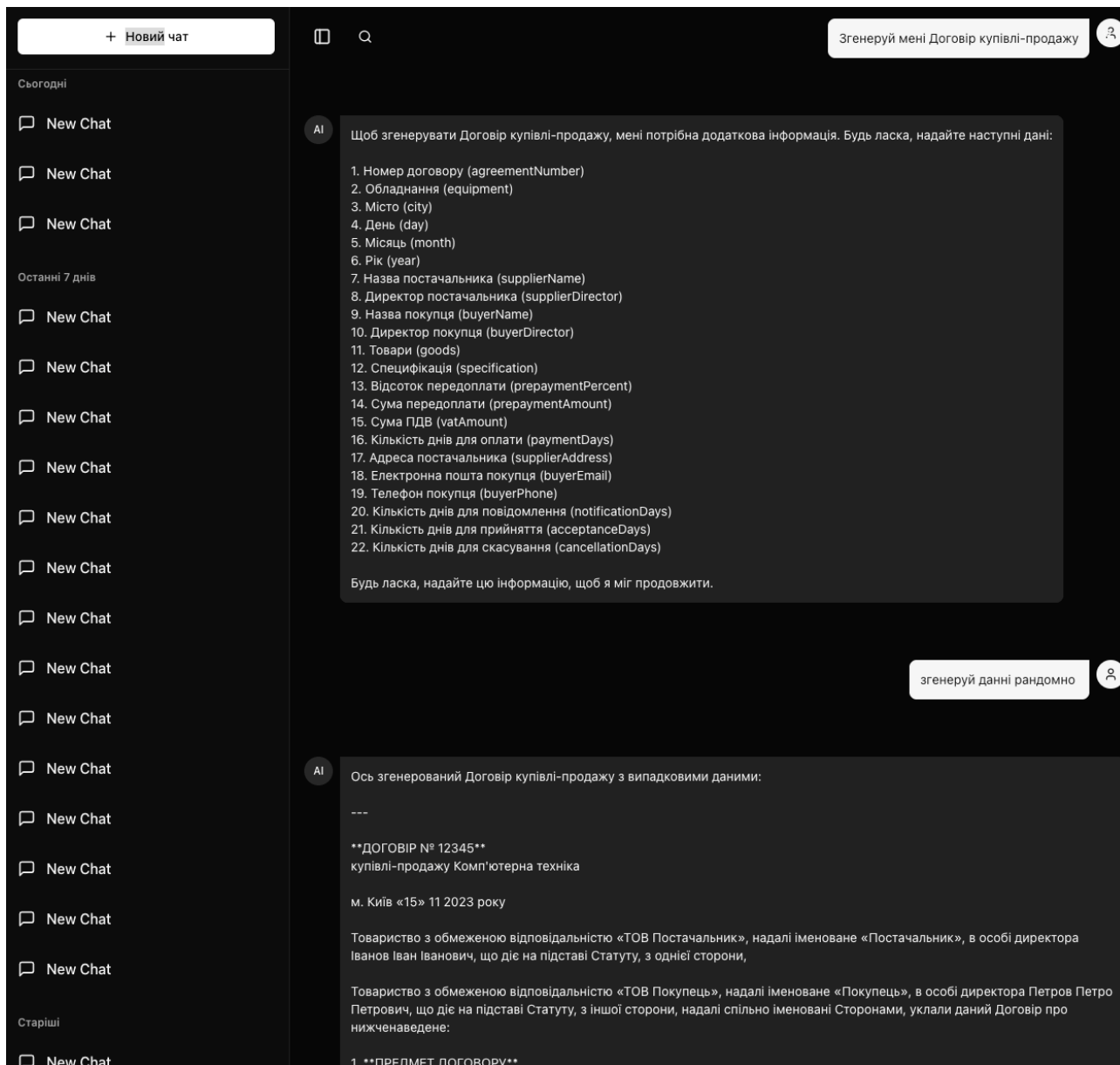


Рисунок 3.9 – Інтерфейс генерації юридичних документів у чат-боті

Крім основного функціоналу, платформа дозволяє користувачеві самостійно керувати своїми чатами. За необхідності користувач може видаляти окремі чати, наприклад, у разі завершення консультації або якщо інформація стала неактуальною (рис 3.10). Перед остаточним видаленням система відображає підтверджуюче діалогове вікно, у якому необхідно підтвердити або скасувати дію. Це допомагає уникнути випадкового видалення важливої переписки, оскільки дія є незворотною.

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				72
Змн.	Арк.	№ докум.	Підпис	Дата		

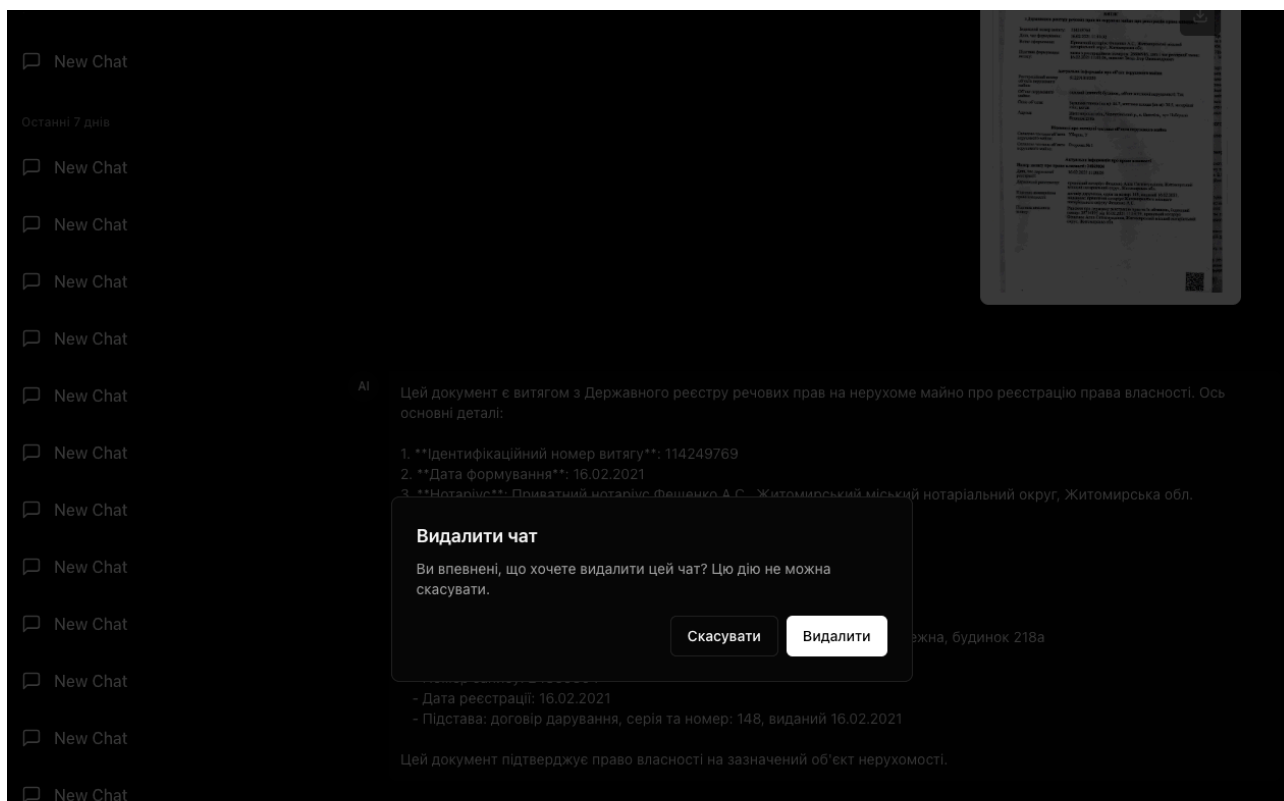


Рисунок 3.10 – Діалогове вікно підтвердження видалення чату

Ще однією важливою функцією платформи є можливість запису на консультацію з реальним юристом для користувачів, які мають активну підписку. Після натискання на кнопку «Записатися на консультацію» відкривається форма, де користувач обирає юриста, дату, час та вказує деталі свого звернення (рис 3.11 та 3.12). Після підтвердження запису система перенаправляє користувача на сторінку профілю, де у вкладці «Консультації» відображається перелік усіх майбутніх консультацій (рис 3.13). Це дозволяє зручно керувати своїми запланованими зустрічами з юристом.

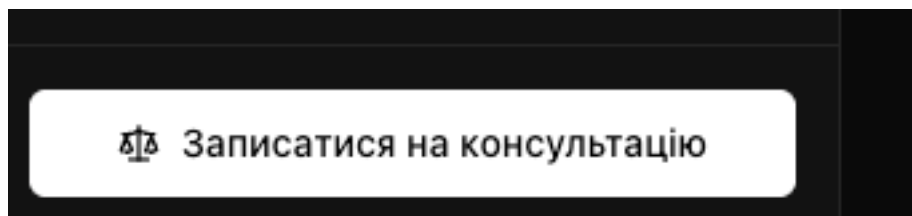


Рисунок 3.11 – Кнопка для запису на юридичну консультацію

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				73
Змн.	Арк.	№ докум.	Підпис	Дата		

Law Assist

Запис на юридичну консультацію

Оберіть дату та час відеоконсультації з юристом

Виберіть юриста

Анатолій Ковальчук | Цивільне право
Практикуючий юрист з 5-річним досвідом у цивільних справах.

Степан Ретривих | Конституційне право

Миколай Ярмоленко | Господарське право
Практикуючий юрист з 4-річним досвідом у господарських справах.

Виберіть дату та час

<

May 2025

>

Su	Mo	Tu	We	Th	Fr	Sa
27	28	29	30	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Доступний час

11:00

Додаткові деталі

У мене проблема з моїм партнером

Підсумок бронювання

Юрист

kostolev228800@gmail.com

Дата

30 May 2025

Час

11:00

Підтвердити запис

Рисунок 3.12 – Форма для запису на консультацію з юристом

Налаштування профілю

Профіль
Консультації

Upcoming Consultations

View and manage your scheduled consultations

Анатолій Ковальчук

May 30, 2025

11:00 AM

Pending

Рисунок 3.13 – Перелік запланованих консультацій у профілі користувача

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				74
Змн.	Арк.	№ докум.	Підпис	Дата		

Також користувач з підпискою може вибрати одного з доступних ботів із певними знаннями в галузі юриспруденції, залежно від своєї потреби (рис 3.14). Для цього реалізовано випадаючий список моделей, серед яких можна обрати, наприклад, універсального або спеціалізованого бота для окремих правових питань. Така можливість забезпечує більш релевантні та точні відповіді на запити користувачів.

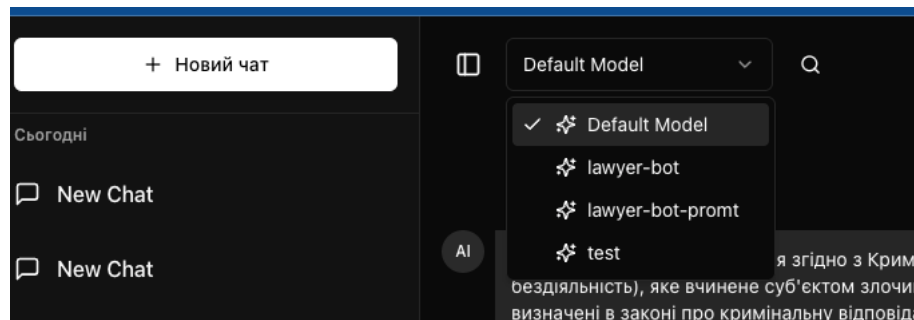


Рисунок 3.14 – Вибір спеціалізованого бота для консультування

Для того щоб оформити підписку, користувач переходить на сторінку з тарифними планами, де може ознайомитися з доступними варіантами та їх умовами (рис 3.15). Після вибору потрібної підписки і натискання кнопки оформлення, користувач перенаправляється на сторінку оплати, де заповнює платіжні дані (рис 3.16). У разі успішної оплати система повідомляє про активацію підписки, після чого користувач отримує повний доступ до всіх функцій платформи (рис 3.17).

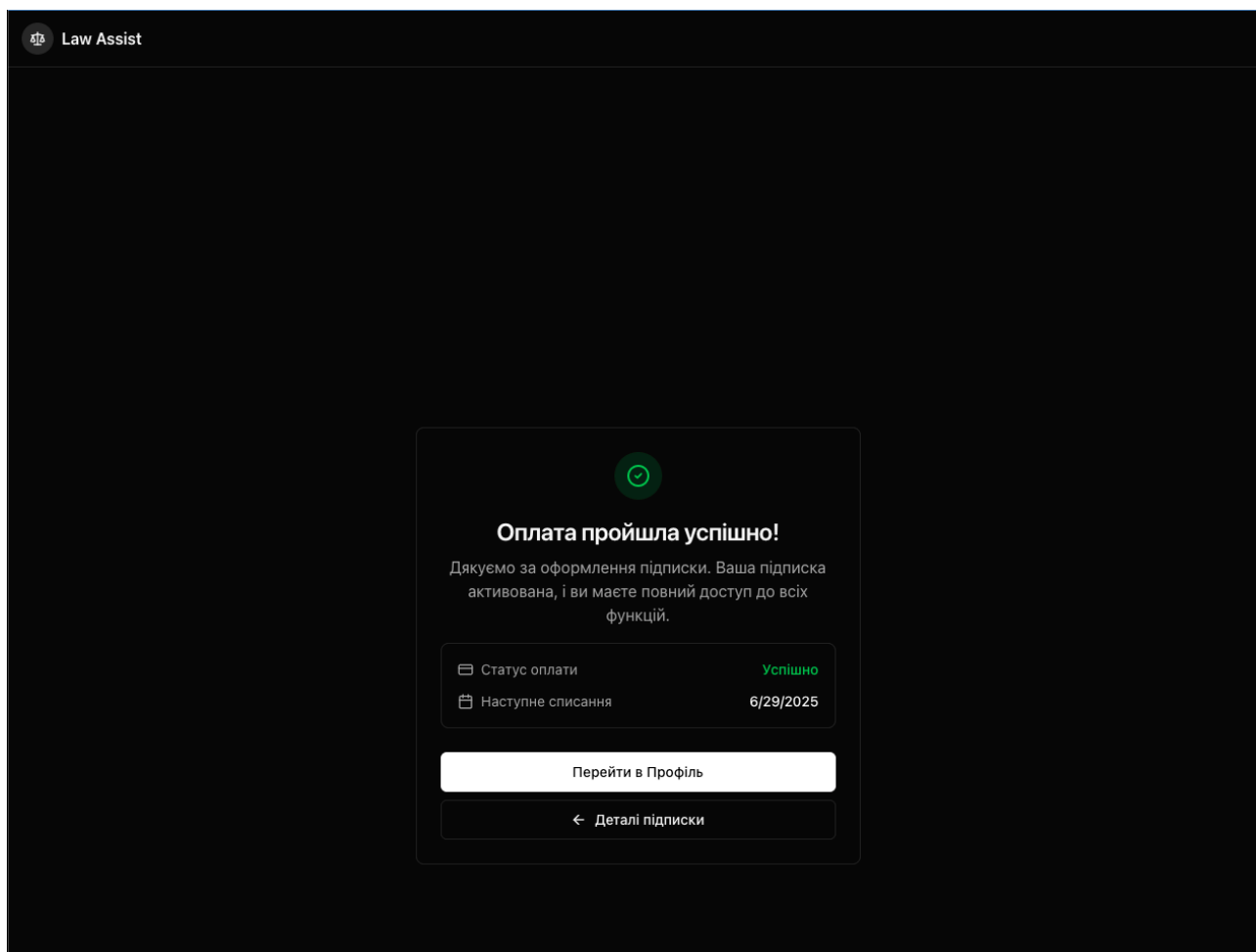


Рисунок 3.17 – Повідомлення про успішне оформлення підписки

З боку юриста реалізовано інтерфейс для адміністрування відеоконсультацій, що дозволяє переглядати всі заплановані консультації користувачів у профілі (рис 3.18). Юрист має можливість змінювати статус консультації, а також редагувати дату й час проведення зустрічі через спеціальну форму (рис 3.19). Такий підхід забезпечує гнучке управління розкладом і статусами консультацій, що підвищує якість обслуговування клієнтів.

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				77
Змн.	Арк.	№ докум.	Підпис	Дата		

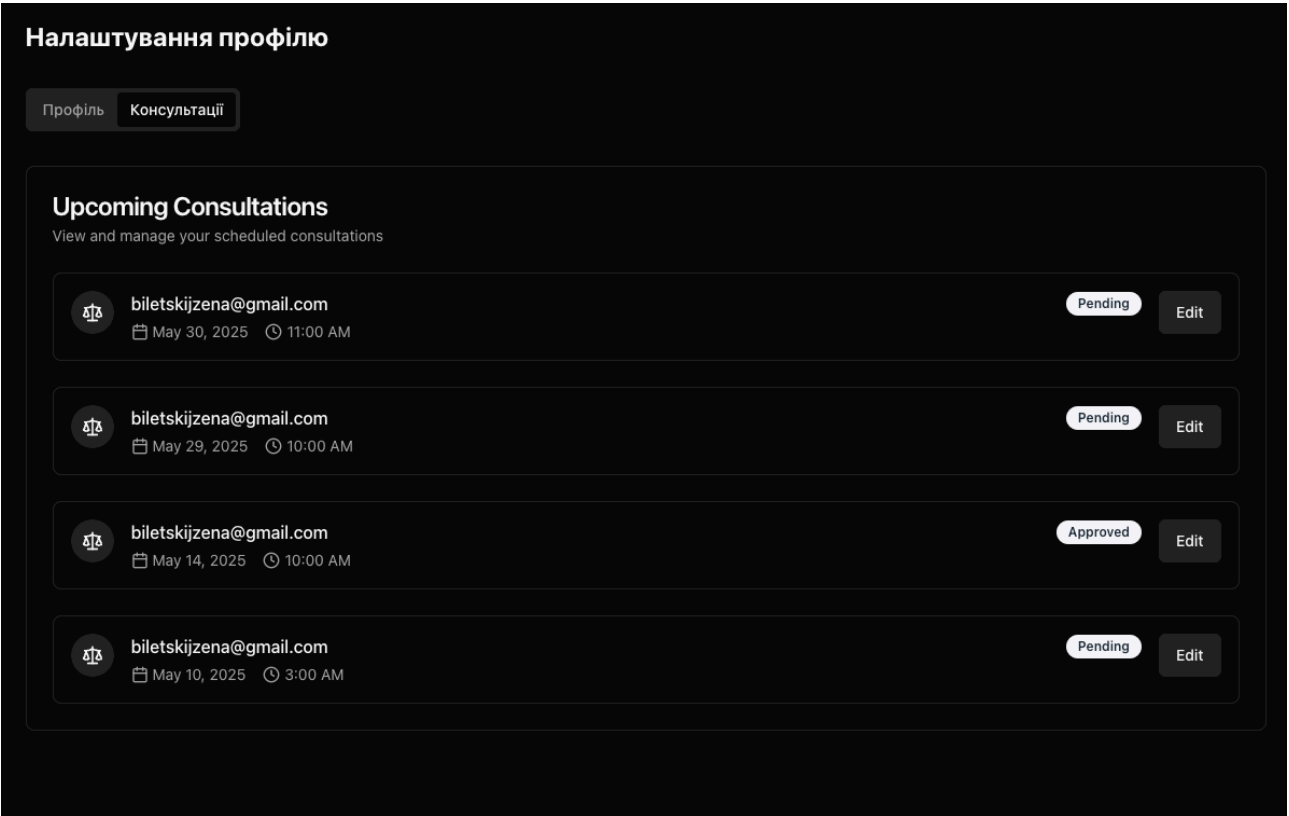


Рисунок 3.18 – Перелік запланованих консультацій для юриста

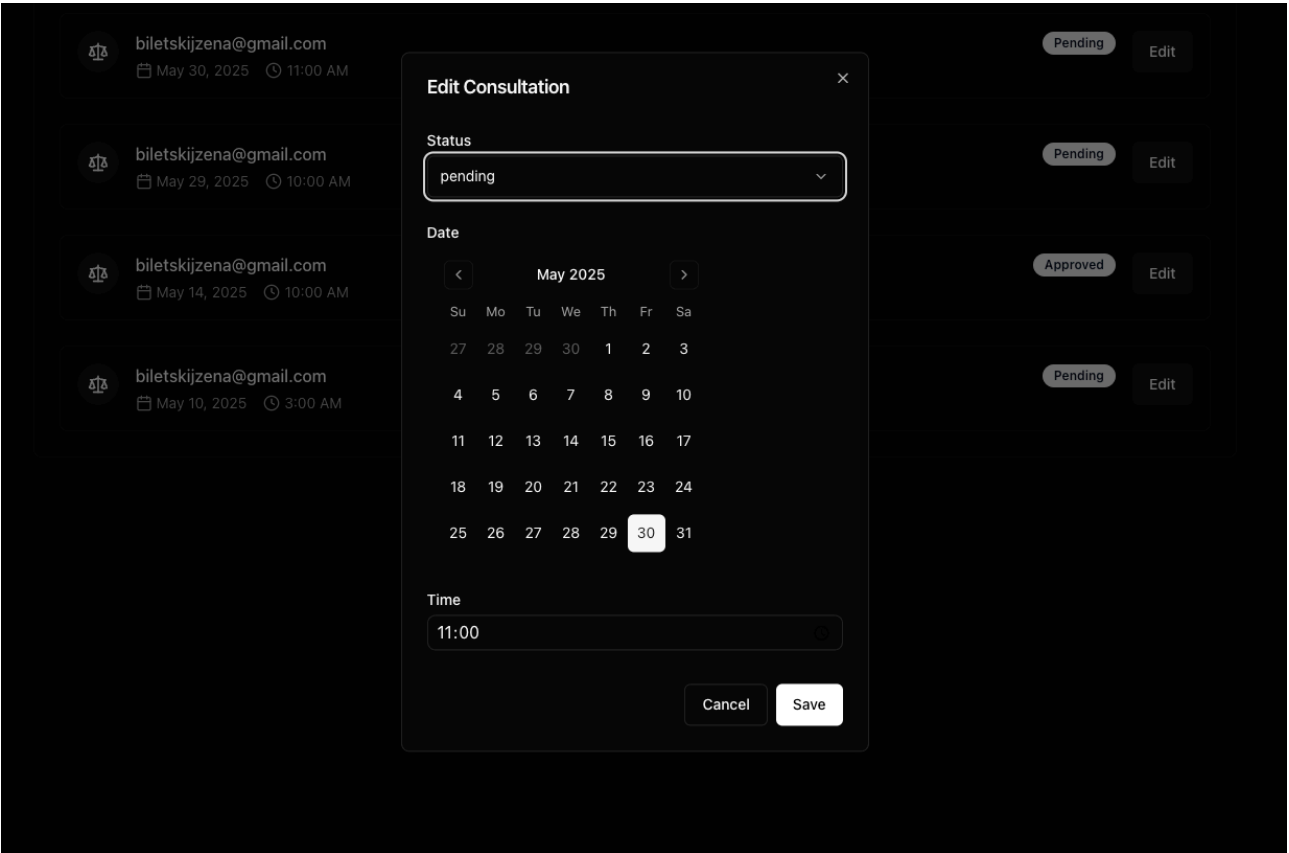


Рисунок 3.19 – Форма редагування статусу та часу консультації

Після того як юрист підтвердив відеоконсультацію, у визначений час стає доступною спеціальна кімната для проведення відеозустрічі. Користувач і юрист можуть приєднатися до консультації, обравши камеру й мікрофон, а також переглянути коротку інформацію одне про одного перед початком розмови. Інтерфейс забезпечує зручний доступ до відеозв'язку у форматі "обличчя до обличчя" (рис 3.20-3.21).

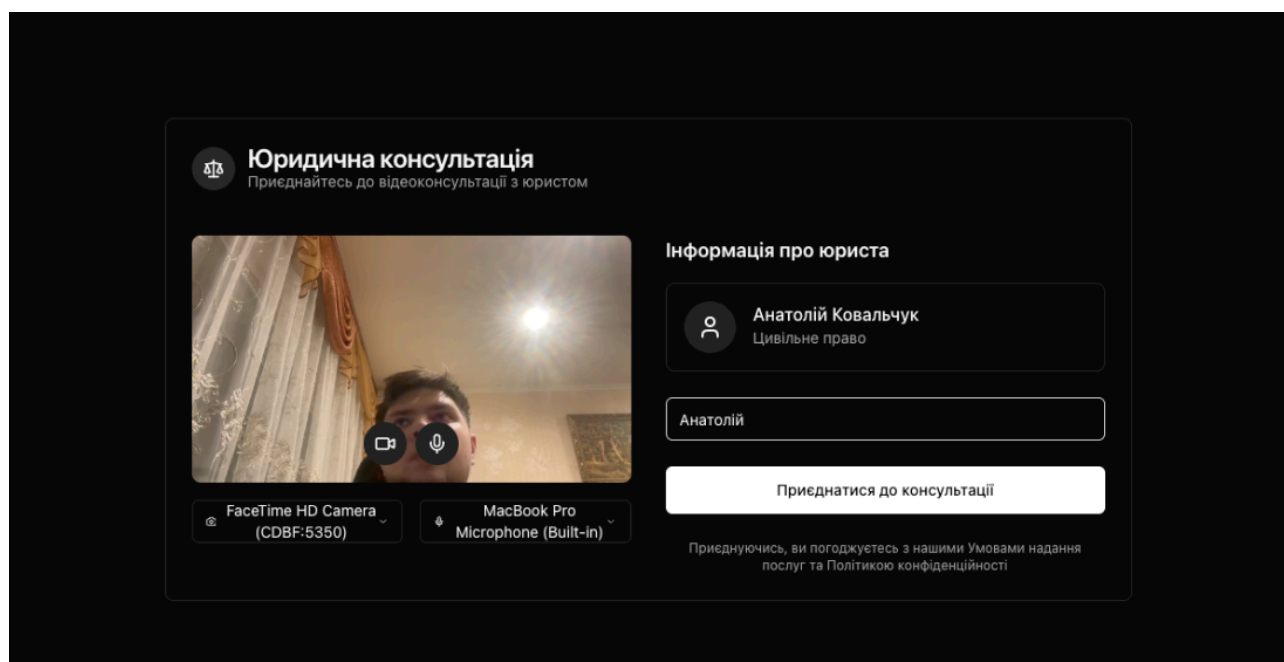


Рисунок 3.20 – Інтерфейс приєднання до відеоконсультації для користувача

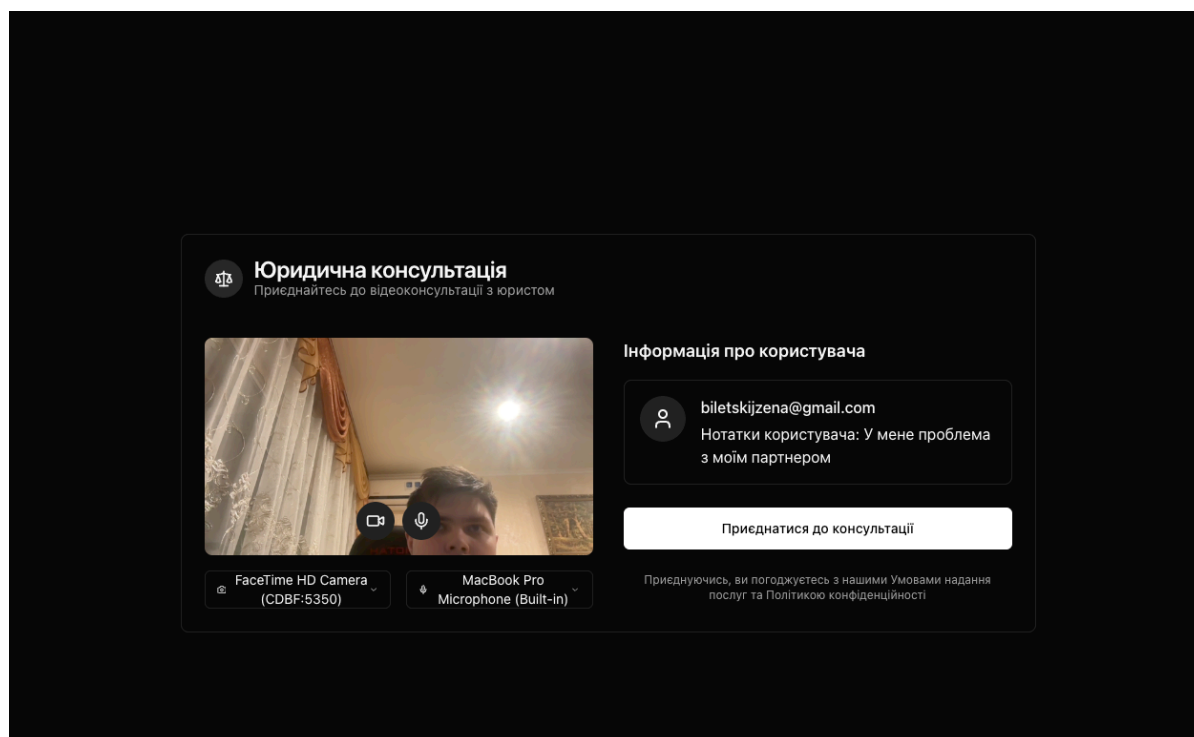


Рисунок 3.21 – Інтерфейс приєднання до відеоконсультації для юриста

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				79
Змн.	Арк.	№ докум.	Підпис	Дата		

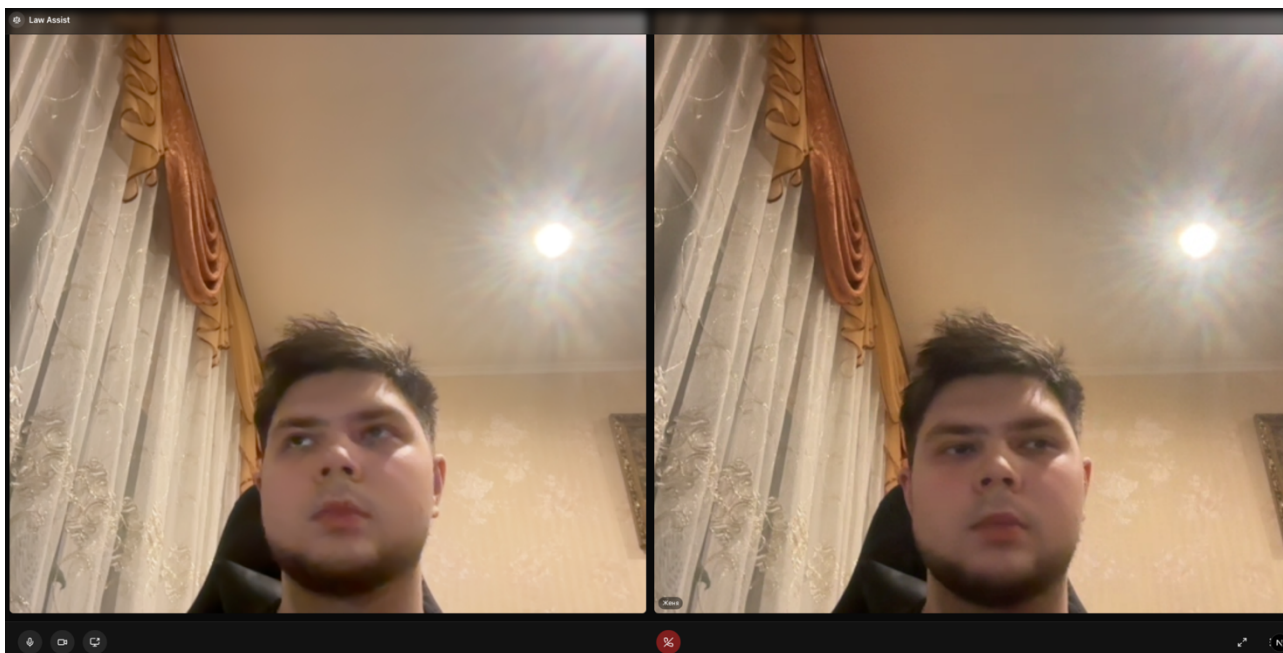


Рисунок 3.22 – Вікно відеоконсультації з двома учасниками

У платформі реалізовано окремий дашборд для адміністрування, до якого мають доступ юристи. Тут можна переглядати, створювати та редагувати AI-ботів і колекції знань, керувати шаблонами документів, а також аналізувати відгуки користувачів на відповіді штучного інтелекту (рис 3.23). Окрім цього, дашборд містить статистику відгуків і забезпечує повний контроль за актуальністю та якістю баз знань, доступних у системі (рис 3.24-3.25).

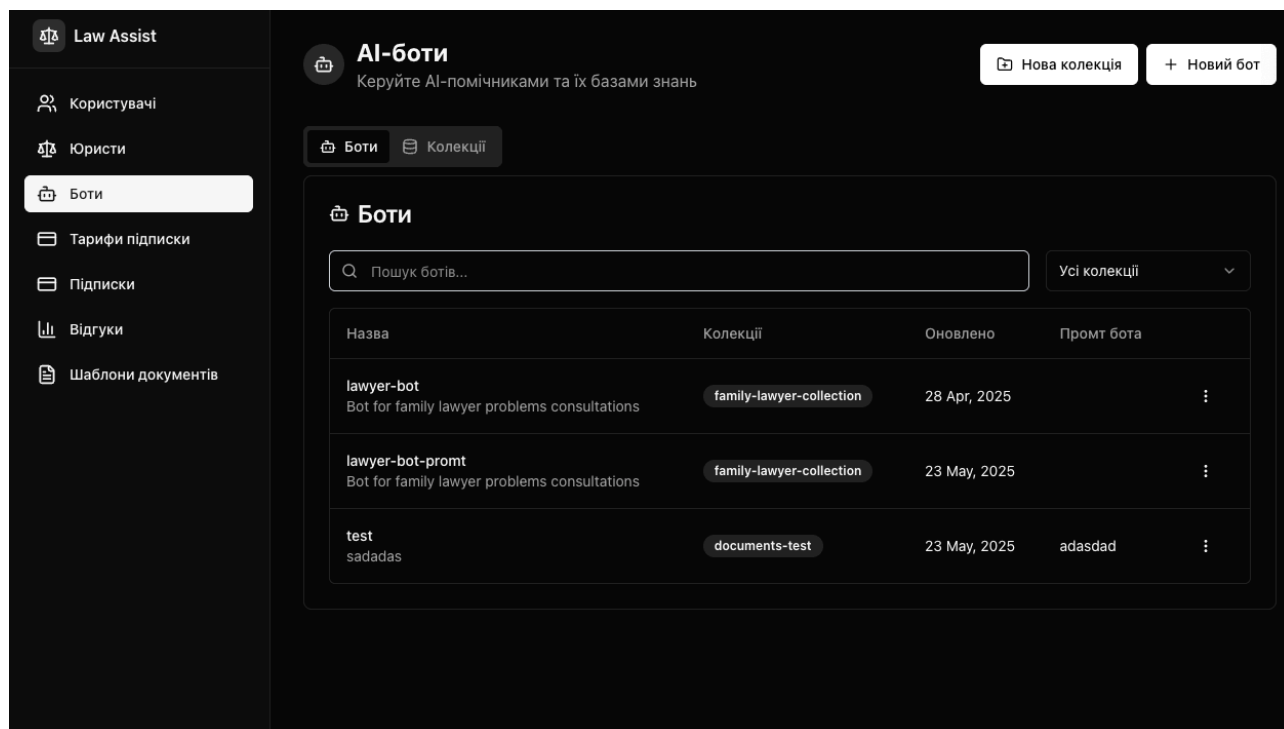


Рисунок 3.23 – Управління ШІ-ботами та колекціями знань

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				80
Змн.	Арк.	№ докум.	Підпис	Дата		

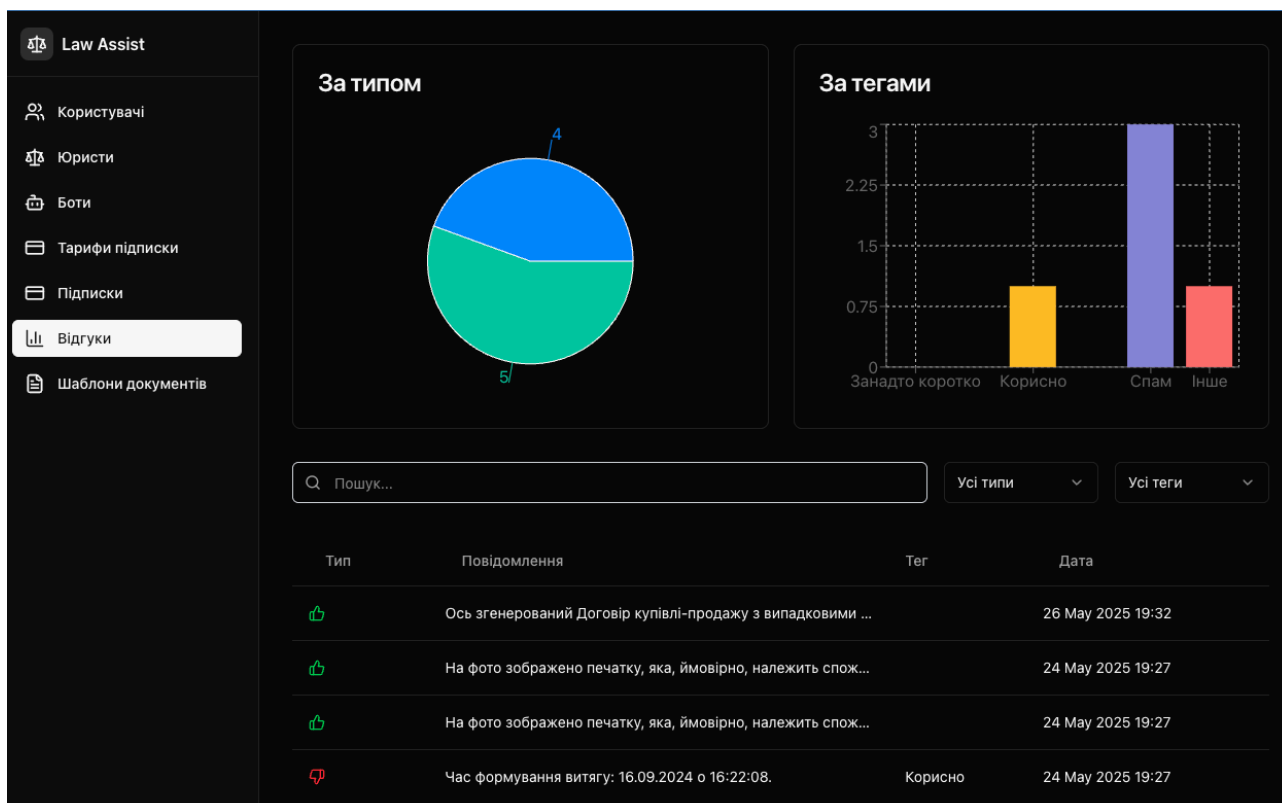


Рисунок 3.24 – Сторінка перегляду і аналітики відгуків користувачів

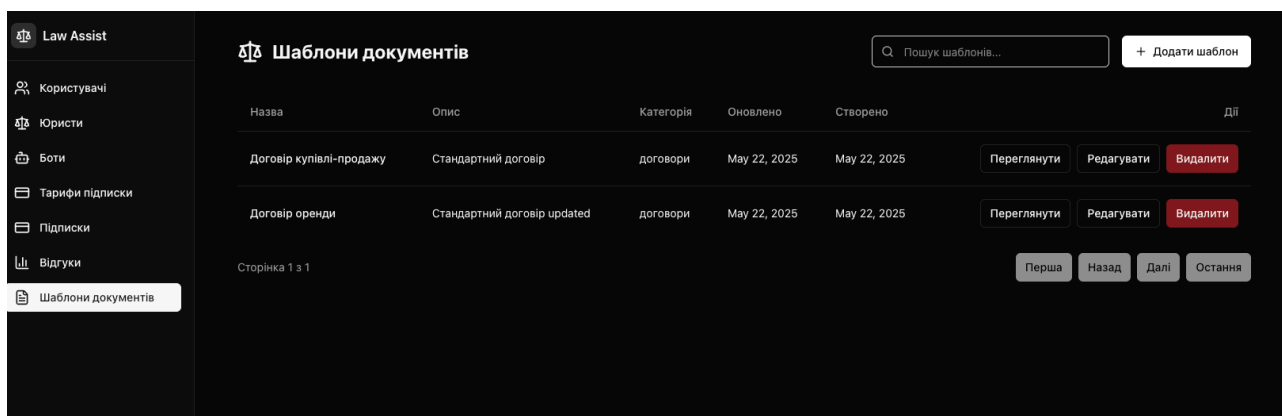


Рисунок 3.25 – Управління шаблонами документів у дашборді адміністратора

У адміністратора платформи доступні всі ті ж самі сторінки дашборду, що й у юриста, проте додатково реалізовано низку розширених можливостей. Адміністратор має окрему сторінку загальної статистики, може переглядати та змінювати ролі користувачів, створювати і редагувати тарифні плани для підписок, а також керувати купленими підписками. Окрім цього, доступна сторінка для управління юристами, де адміністратор може переглядати та редагувати профілі фахівців платформи (рис 3.25-3.30).

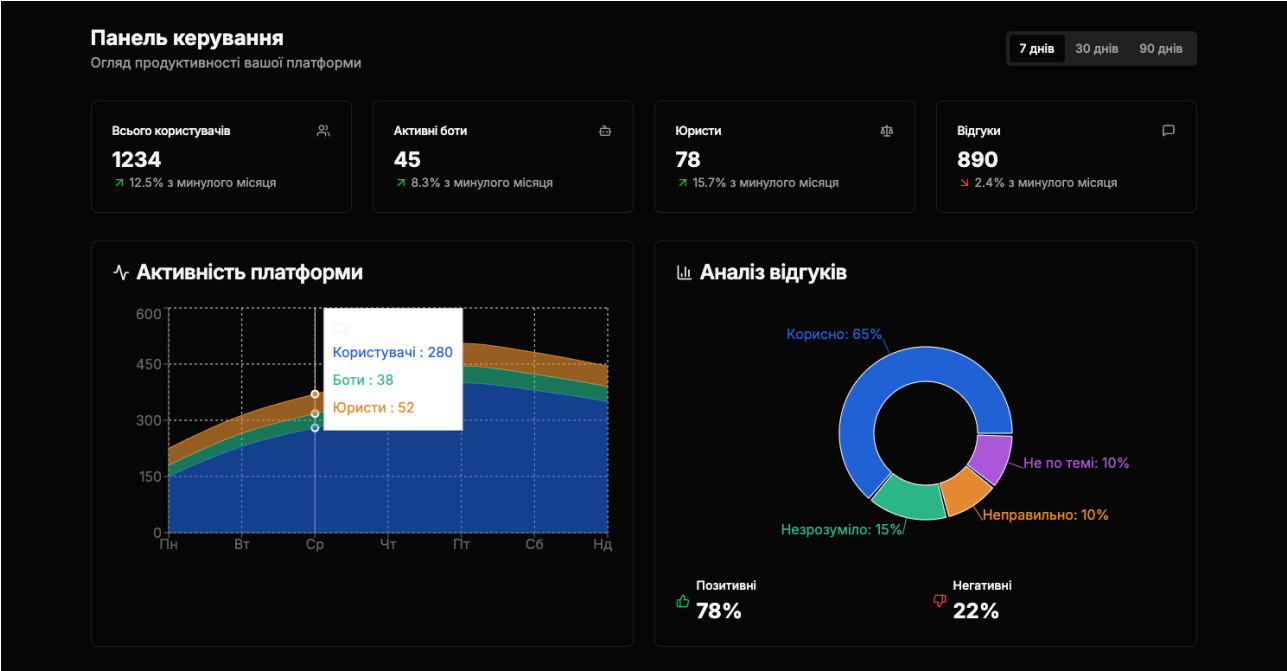


Рисунок 3.26 – Загальна статистика платформи в дашборді адміністратора

Law Assist

Користувачі

Юристи

Боти

Тарифи підписки

Підписки

Відгуки

Шаблони документів

Користувачі
Керуйте обліковими записами та правами доступу

Управління користувачами

Пошук користувачів...

10 на сторінку

Користувач	Email	Статус	Роль	Дата створення
admin	admin@gmail.com	Не підтверджено	admin	3 Jan 2025
kostolev228800	kostolev228800@gmail.com	Підтверджено	lawyer	21 Feb 2025
user1	user1@example.com	Не підтверджено	lawyer	26 Feb 2025

Рисунок 3.27 – Управління користувачами та зміна ролей

Law Assist

Користувачі

Юристи

Боти

Тарифи підписки

Підписки

Відгуки

Шаблони документів

Тарифні плани
Керуйте своїми тарифними планами та цінами

+ Новий план

Плани

Пошук планів...

10 на сторінку

Назва	Ціна	Можливості	Пробний період	Дата створення	Дії
VIP+	\$100.00	featureA featureB	14 днів	May 7, 2025	

Рисунок 3.28 – Створення та редагування тарифних планів

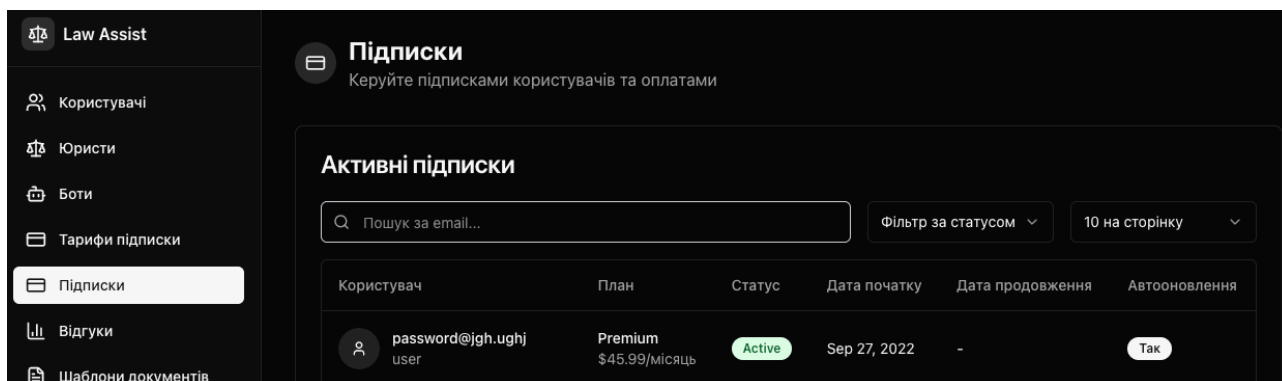


Рисунок 3.29 – Перегляд і керування купленими підписками

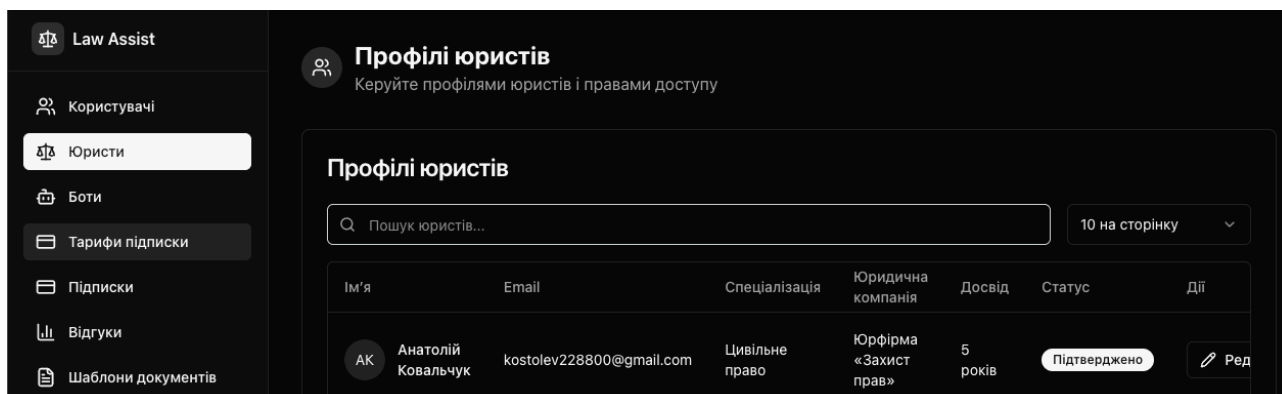


Рисунок 3.30 – Управління профілями юристів

3.3 Тестування роботи програмного продукту

Створення будь-якого програмного продукту завжди супроводжується певними помилками на етапі розробки, адже передбачити всі можливі випадки неможливо. Тестування є невід'ємною складовою процесу розробки, адже дозволяє перевірити точність вхідних і вихідних даних, релевантність функцій, обробку помилок та стійкість програми до збоїв, а головна мета — виявити і виправити недоліки до моменту впровадження системи у реальну експлуатацію.

У межах даного проєкту було застосовано комплексний підхід до тестування. Проведено модульні, інтеграційні та функціональні тести для перевірки коректної роботи основних компонентів як на серверній, так і на клієнтській частині. Зокрема, для бекенду на базі NestJS із використанням Jest [47] були протестовані основні контролери та сервіси управління підписками, тарифними планами, зворотним зв'язком і автентифікацією, включаючи

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				83
Змн.	Арк.	№ докум.	Підпис	Дата		

перевірку валідності сценаріїв доступу та обробки помилок, що підтверджується результатами запуску unit-тестів (рис. 3.31).

```

● yevheniibeletskiy@Yevheniis-MacBook-Pro law-assist-backend % yarn test
yarn run v1.22.22
$ jest --config jest.config.ts
PASS src/app.controller.spec.ts (16.408 s)
PASS src/subscription/subscription.controller.spec.ts (20.35 s)
PASS src/subscription-plan/subscription-plan.controller.spec.ts (20.654 s)
PASS src/auth/auth.controller.spec.ts (20.761 s)
● Console

    console.log
      { email: 'reset@test.com' }

      at AuthController.forgotPassword (auth/auth.controller.ts:106:13)

PASS src/feedback/feedback.controller.spec.ts (20.963 s)

Test Suites: 5 passed, 5 total
Tests:       37 passed, 37 total
Snapshots:   0 total
Time:        21.397 s
Ran all test suites.
🌟 Done in 29.02s.
○ yevheniibeletskiy@Yevheniis-MacBook-Pro law-assist-backend %

```

Рисунок 3.31. – Результати запуску unit-тестів для бекенд-частини програмного продукту.

Клієнтська частина програмного продукту була протестована на відповідність усім ключовим сценаріям роботи. Зокрема, проведено перевірку процесів реєстрації та авторизації користувачів, взаємодії із чат-ботом на основі штучного інтелекту, генерації та завантаження юридичних документів, а також завантаження власних файлів для аналізу. Окремо протестовано функціонал адміністративного дашборду та можливість проведення відеоконференцій. Узагальнені результати тестування клієнтської частини наведено у таблиці 3.1.

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				84
Змн.	Арк.	№ докум.	Підпис	Дата		

Таблиця 3.1

№	Назва тест-кейсу	Короткий опис дій	Очікуваний результат	Фактичний результат	Статус
1	Реєстрація користувача	Введення коректних даних, підтвердження реєстрації	Користувач зареєстрований, отримано лист підтвердження	Лист отримано	Успішно
2	Авторизація користувача	Введення email і пароля, натискання «Увійти»	Відкривається особистий кабінет	Відкривається кабінет	Успішно
3	Запит до ШІ	Введення питання, натискання «Надіслати»	Відображається відповідь від ШІ	Відповідь отримано	Успішно
4	Генерування юридичного документа	Вибір шаблону, заповнення полів, натискання «Згенерувати»	Згенерований документ відображається на сторінці	Документ згенеровано	Успішно
5	Завантаження власного документа для аналізу	Завантаження PDF чи фото документа, відправка на аналіз	Документ успішно завантажено, починається аналіз	Аналіз розпочато	Успішно

6	Тестування адмін-дашборду	Вхід під обліковим записом адміністратора, перегляд сторінки керування	Доступний дашборд адміністрування	Дашборд відкрито	Успішно
7	Керування користувачами у дашборді	Вибір користувача, зміна статусу/видалення	Зміни відображаються у списку	Зміни застосовано	Успішно
8	Проведення відеоконференції	Запланувати/почати відеоконференцію, підключення двох користувачів	Відеозв'язок встановлено, іде обмін відео/аудіо	Зв'язок працює	Успішно
9	Завершення відеоконференції	Завершення сесії, вихід обох користувачів	Сеанс завершено, учасники відключені	Сеанс завершено	Успішно

Таким чином, результати проведеного тестування підтверджують стабільність і надійність як серверної, так і клієнтської частин програмного продукту. Усі основні сценарії взаємодії користувачів і адміністраторів працюють коректно, а функціонал відповідає поставленим вимогам. Комплексний підхід до тестування дозволив виявити й усунути потенційні недоліки ще на етапі розробки, що значно підвищує якість та експлуатаційну готовність системи.

Висновки до третього розділу

У третьому розділі було розглянуто структуру, порядок розгортання та налаштування платформи для надання юридичної допомоги із застосуванням штучного інтелекту. Представлено архітектурну діаграму, описано основні компоненти системи, механізми їхньої взаємодії, а також особливості інтеграції зовнішніх сервісів, таких як OpenAI API, ChromaDB, Stripe та Vercel Blob Storage. Надано рекомендації щодо налаштування середовища, підключення бази даних, конфігурування серверної і клієнтської частин, а також організації автоматизованого CI/CD-процесу.

Детально проаналізовано інтерфейс користувача для різних ролей – від звичайного користувача до адміністратора та юриста. Описано ключові сценарії взаємодії: реєстрація та авторизація, робота з чат-ботом на основі ШІ, завантаження документів для аналізу, генерація юридичних документів, запис на відеоконсультації, управління підписками, а також можливості адміністрування ботів, колекцій знань і шаблонів документів. Наведено приклади організації інтерфейсу та продемонстровано, як користувачі отримують доступ до різних функцій залежно від своєї ролі.

У розділі також розглянуто комплексний підхід до тестування платформи, включаючи модульні, інтеграційні й функціональні тести для серверної та клієнтської частин. Оцінено стабільність роботи всіх основних функціональних модулів, підтверджено коректність взаємодії користувачів і адміністраторів із системою. За підсумками тестування платформа продемонструвала відповідність поставленим вимогам, високу надійність і готовність до подальшого впровадження та експлуатації.

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				87
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи було спроектовано та реалізовано сучасну веборієнтовану платформу для надання юридичної допомоги з використанням технологій штучного інтелекту. На основі аналізу предметної області та ринку, а також порівняльної характеристики провідних аналогів, визначено основні вимоги до функціоналу системи, її структури та інтерфейсу.

У ході роботи було проведено комплексне дослідження існуючих рішень, визначено їх переваги й недоліки, що дозволило сформулювати конкурентні переваги власного продукту — адаптацію до українського законодавства, інтеграцію відеоконсультацій з реальними юристами, зручне адміністрування, підтримку підписок і рольового доступу. У результаті проектування було створено базу даних, розроблено інтуїтивно зрозумілий вебінтерфейс для різних категорій користувачів, а також забезпечено належний рівень захисту персональних даних.

Детально описано основні сценарії використання системи: отримання правових консультацій через чат-бот із ШІ, генерацію юридичних документів, організацію та проведення відеоконсультацій, адміністрування контенту й статистики. Платформа успішно пройшла тестування, у ході якого були перевірені всі основні функціональні модулі, виявлені та усунуті недоліки.

Розроблена система відповідає сучасним вимогам до автоматизації процесів юридичної допомоги, підвищує доступність правових послуг, сприяє оптимізації роботи юристів і підвищенню юридичної грамотності населення. Продукт готовий до впровадження та подальшого розвитку відповідно до актуальних потреб ринку.

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				88
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Білецький Є. В., Савіцький Р. С. ВПЛИВ ШТУЧНОГО ІНТЕЛЕКТУ НА ТРАНСФОРМАЦІЮ ЮРИДИЧНИХ ПОСЛУГ. *ТЕЗИ ДОПОВІДЕЙ XV Міжнародної науково-технічної конференції Інформаційно-комп'ютерні технології*, Житомир, Україна, 28–29 берез. 2025.
2. Білецький Є. В., Савіцький Р. С. LANGCHAIN ДЛЯ РЕАЛІЗАЦІЇ RETRIEVAL-AUGMENTED GENERATION (RAG) У AI-СИСТЕМАХ. *VI Міжнародна наукова конференція «Актуальні проблеми фундаментальних наук» (АПФН-2025) присвяченої пам'яті Джордано Бруно*, Луцьк, Україна, 9–15 черв. 2025.
3. Retrieval augmented generation [Електронний ресурс] - Режим доступу до ресурсу: <https://js.langchain.com/docs/concepts/rag/>
4. AI Layew. Платформа «AI Layew» [Електронний ресурс] - Режим доступу до ресурсу: <https://ailawyer.pro/>
5. Law.co. Платформа «Law.co» [Електронний ресурс] - Режим доступу до ресурсу: <https://law.co/>
6. DocDraft. Платформа «DocDraft» [Електронний ресурс] - Режим доступу до ресурсу: <https://www.docdraft.ai/>
7. Посвістак В. С. Клієнт-серверна архітектура та її використання при розробці програмного забезпечення / В. С. Посвістак, Т. І. Демківська // Інформаційні технології в науці, виробництві та підприємстві: збірник наукових праць молодих вчених, аспірантів, магістрів кафедри комп'ютерних наук та технологій / за заг. наук. ред. В. Ю. Щербаня. – Київ: Освіта України; ФОП Маслаков, 2020. – С. 78-81.
8. OpenAI API [Електронний ресурс] – Режим доступу до ресурсу: <https://platform.openai.com/docs/api-reference>
9. Mikowski M., Powell J. Single Page Web Applications: JavaScript end-to-end. – Manning Publications, 2013. – 432 с. – ISBN 978-1617290756, DOI 0.5555/2663433.

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				89
Змн.	Арк.	№ докум.	Підпис	Дата		

10. TypeScript [Електронний ресурс] – Режим доступу до ресурсу: <https://www.typescriptlang.org/>
11. NextJS [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org>
12. NestJS [Електронний ресурс] – Режим доступу до ресурсу: <https://nestjs.com>
13. Mongoose [Електронний ресурс] – Режим доступу до ресурсу: <https://mongoosejs.com/>
14. NextJS SSR [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org/docs/pages/building-your-application/rendering/server-side-rendering>
15. NextJS SSG [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org/docs/pages/building-your-application/rendering/static-site-generation>
16. NextAuth.js [Електронний ресурс] – Режим доступу до ресурсу: <https://next-auth.js.org/>
17. Tanstack query [Електронний ресурс] – Режим доступу до ресурсу: <https://tanstack.com/query/latest/>
18. Tailwind CSS [Електронний ресурс] – Режим доступу до ресурсу: <https://tailwindcss.com/docs>
19. CSS [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/CSS>
20. RadixUI [Електронний ресурс] – Режим доступу до ресурсу: <https://www.radix-ui.com/>
21. React hook form [Електронний ресурс] – Режим доступу до ресурсу: <https://www.react-hook-form.com/>
22. Langchain [Електронний ресурс] – Режим доступу до ресурсу: <https://www.langchain.com/>
23. ChromaDB [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.trychroma.com/>
24. JWT [Електронний ресурс] – Режим доступу до ресурсу: <https://jwt.io/introduction>

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				90
Змн.	Арк.	№ докум.	Підпис	Дата		

25. OAuth [Електронний ресурс] – Режим доступу до ресурсу:
<https://oauth.net/2/>
26. Pdf-parse [Електронний ресурс] – Режим доступу до ресурсу:
<https://gitlab.com/autokent/pdf-parse>
27. Puppeteer [Електронний ресурс] – Режим доступу до ресурсу:
<https://pptr.dev/>
28. Nodemailer [Електронний ресурс] – Режим доступу до ресурсу:
<https://nodemailer.com/>
29. Langchain Text Splitters [Електронний ресурс] – Режим доступу до ресурсу:
https://js.langchain.com/docs/concepts/text_splitters/
30. MongoDB [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.mongodb.com/>
31. PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.postgresql.org/>
32. MySQL [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.mysql.com/>
33. Casandra [Електронний ресурс] – Режим доступу до ресурсу:
https://cassandra.apache.org/_/index.html
34. Pinecore [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.pinecone.io/>
35. Weavite [Електронний ресурс] – Режим доступу до ресурсу:
<https://weaviate.io>
36. FAISS [Електронний ресурс] - Режим доступу до ресурсу: <https://faiss.ai/>
37. ReactJS [Електронний ресурс] – Режим доступу до ресурсу:
<https://react.dev/reference/react>
38. Singh A. Building LLM Applications with RAG Architecture. San Diego : AI Frontier Press, 2024. 198 p. ISBN 978-1958579023.
39. Stripe [Електронний ресурс] – Режим доступу до ресурсу:
<https://docs.stripe.com/>

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				91
Змн.	Арк.	№ докум.	Підпис	Дата		

40. Codemancers. Implementing Retrieval-Augmented Generation with LangChain, Pgvector and OpenAI [Електронний ресурс]. Codemancers, 24.10.2024. Режим доступу: <https://www.codemancers.com/blog/2024-10-24-rag-with-langchain>.
41. Vercel BLOB [Електронний ресурс] – Режим доступу до ресурсу: <https://vercel.com/docs/vercel-blob>
42. Node.JS [Електронний ресурс] – Режим доступу до ресурсу: <https://nodejs.org/en>
43. Github Action [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/features/actions>
44. Eslint [Електронний ресурс] – Режим доступу до ресурсу: <https://eslint.org/docs/>
45. Prettire [Електронний ресурс] – Режим доступу до ресурсу: <https://prettier.io/>
46. Jest [Електронний ресурс] - Режим доступу до ресурсу: <https://jestjs.io/uk/>
47. Langchain Tools [Електронний ресурс] – Режим доступу до ресурсу: <https://js.langchain.com/docs/concepts/tools/>
48. Olson P. Supremacy: AI, ChatGPT and the Race That Will Change the World. London : Pan Macmillan, 2024. 352 p. ISBN 978-1529099424.
49. Suard T., Modi D. Enterprise RAG: Building Production-Ready Retrieval-Augmented Generation Systems. Shelter Island, NY : Manning Publications, 2025.
50. Ben Auffarth. Generative AI with LangChain. Birmingham : Packt Publishing, 2025. 290 p. ISBN 978-1837022007.
51. Infante R. LangChain in Action. – New York: Manning Publications, 2024. – 350 с.
52. Towards Data Science. Retrieval-Augmented Generation (RAG): From Theory to LangChain Implementation [Електронний ресурс]. Medium. Режим доступу: <https://medium.com/towards-data-science/retrieval-augmented-generation-rag-from-theory-to-langchain-implementation-4e9bd5f6a4f2>.

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				92
Змн.	Арк.	№ докум.	Підпис	Дата		

53. Tyler S., Darshil M. AI Engineering: Building Applications with Foundation Models. Manning Publications, 2025. 225 с. ISBN 9781633435476.
54. Denis R. RAG-Driven Generative AI. Packt Publishing, 2024. 334 с. ISBN 9781836200918.
55. NextJS CSR [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org/docs/pages/building-your-application/rendering/client-side-rendering>

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				93
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				94
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг файлу main.ts

```

import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { ConfigService } from '@nestjs/config';
import { ValidationPipe } from '@nestjs/common';
import { SwaggerModule, DocumentBuilder } from '@nestjs/swagger';
import * as express from 'express';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  app.setGlobalPrefix('api');
  app.use('/api/webhook', express.raw({ type: 'application/json' }));

  app.use(express.json());
  app.use(express.urlencoded({ extended: true }));
  app.useGlobalPipes(new ValidationPipe());

  const config = new DocumentBuilder()
    .setTitle('LawAssist')
    .setDescription('LawAssist API description')
    .setVersion('1.0')
    .addBearerAuth()
    .build();

  app.enableCors();

  const documentFactory = () => SwaggerModule.createDocument(app, config);
  SwaggerModule.setup('api/docs', app, documentFactory);

  const configService = app.get(ConfigService);
  const port = configService.get<number>('PORT');

  await app.listen(port || 3000);
}
bootstrap();

```

Лістинг файлу llm.service.ts

```

import { Injectable } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				95
Змн.	Арк.	№ докум.	Підпис	Дата		

```

import { ChatOpenAI } from '@langchain/openai';
import {
  HumanMessage,
  isAIMessageChunk,
  MessageContentComplex,
} from '@langchain/core/messages';
import { VectorStoreService } from 'src/vector-store/vector-store.service';
import { MessageService } from 'src/message/message.service';
import { Model, Types } from 'mongoose';
import { createReactAgent } from '@langchain/langgraph/prebuilt';
import { MemorySaver } from '@langchain/langgraph';
import { TemplateGeneratorTool } from './tools/TemplateGenerator.tool';
import { LegalTemplate } from 'src/schemas/legal-template.schema';
import { InjectModel } from '@nestjs/mongoose';
import { FilePartItem } from 'src/schemas/message.schema';

```

```

@Injectable()
export class LlmService {
  private chatModel: ChatOpenAI;
  private agent: ReturnType<typeof createReactAgent>;

  constructor(
    private readonly configService: ConfigService,
    private readonly vectorStoreService: VectorStoreService,
    private readonly messageService: MessageService,
    @InjectModel(LegalTemplate.name)
    private readonly templateModel: Model<LegalTemplate>,
  ) {
    this.initializeChatModel();
    this.initializeAgent();
  }

  private initializeChatModel() {
    this.chatModel = new ChatOpenAI({
      model: 'gpt-4o',
      openAIApiKey: this.configService.get<string>('OPENAI_API_KEY'),
      temperature: 0,
      streaming: true,
    });
  }

  private initializeAgent() {
    const memorySaver = new MemorySaver();

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				96
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    this.agent = createReactAgent({
      llm: this.chatModel,
      tools: [new TemplateGeneratorTool(this.templateModel)],
      checkpointSaver: memorySaver,
    });
  }
  async *generateStream(
    chatId: Types.ObjectId,
    prompt: string,
    collectionName = 'documents-test',
    files?: FilePartItem[],
  ) {
    const conversationMessages = await this.messageService.getMessagesByChat(
      chatId.toString(),
    );
    const conversationContext = conversationMessages
      .map((msg) => `${msg.role}: ${msg.content}`)
      .join('\n');

    const messages: any[] = [];

    if (files && files.length > 0) {
      messages.push(
        new HumanMessage({
          content: files,
        }),
      );

      messages.push(
        new HumanMessage({
          content: `
            Conversation context: ${conversationContext}

            Question: ${prompt}

            Answer: `,
        }),
      );
    } else {
      const vectorResults = await this.vectorStoreService.similaritySearch(
        prompt,
        8,
        collectionName,
      );
    }
  }

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				97
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const vectorContext = vectorResults
  .map((doc) => doc.pageContent)
  .join('\n\n');

const augmentedPrompt = `
  Conversation context: ${conversationContext}

  Document context: ${vectorContext}

  Question: ${prompt}

  Answer: `;

messages.push(new HumanMessage({ content: augmentedPrompt }));
}

const streamIter = await this.agent.stream(
  { messages },
  {
    streamMode: 'messages',
    configurable: { thread_id: chatId.toString() },
  },
);

for await (const [msg] of streamIter) {
  if (!isAIMessageChunk(msg)) continue;

  if (typeof msg.content === 'string') {
    yield msg.content;
  } else {
    for (const chunk of msg.content as MessageContentComplex[]) {
      if (chunk.type === 'text') {
        yield chunk.text;
      }
    }
  }
}
}
}

```

Лістинг файлу conversation.service.ts

```
import { Injectable, NotFoundException } from '@nestjs/common';
```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				98
Змн.	Арк.	№ докум.	Підпис	Дата		

```

import { InjectModel } from '@nestjs/mongoose';
import { console } from 'inspector';
import { isValidObjectId, Model, Types } from 'mongoose';
import {
  Conversation,
  ConversationDocument,
} from 'src/schemas/conversation.schema';
import { Message, MessageDocument } from 'src/schemas/message.schema';

@Injectable()
export class ConversationService {
  constructor(
    @InjectModel(Conversation.name)
    private conversationModel: Model<ConversationDocument>,
    @InjectModel(Message.name)
    private readonly messageModel: Model<MessageDocument>,
  ) {}

  async createConversation(userId: string): Promise<ConversationDocument> {
    const conversation = new this.conversationModel({ userId });
    return conversation.save();
  }

  async findById(
    conversationId: string,
    userId: string,
  ): Promise<ConversationDocument | null> {
    if (!isValidObjectId(conversationId) || !userId) {
      return null;
    }
    return this.conversationModel
      .findOne({ _id: new Types.ObjectId(conversationId), userId })
      .exec();
  }

  async addMessages(
    conversationId: Types.ObjectId,
    messageIds: Types.ObjectId[],
  ) {
    await this.conversationModel.updateOne(
      { _id: conversationId },
      { $push: { messages: { $each: messageIds } } },
    );
  }
}

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				99
Змн.	Арк.	№ докум.	Підпис	Дата		

```

async getConversationById(conversationId: string, userId: string) {
  try {
    if (!isValidObjectId(conversationId)) {
      throw new NotFoundException('Invalid conversation id');
    }

    const conversation = await this.conversationModel
      .findOne({ _id: conversationId, userId })
      .populate({
        path: 'messages',
        model: 'Message',
        options: { sort: { createdAt: 1 } },
      })
      .exec();

    if (!conversation) {
      throw new NotFoundException('Conversation not found');
    }

    return conversation;
  } catch (error) {
    throw error;
  }
}

```

```

async getUserConversations(
  userId: string,
  page: number = 1,
  limit: number = 10,
) {
  if (!userId) {
    throw new NotFoundException('User ID is required');
  }

  const skip = (page - 1) * limit;
  const [conversations, total] = await Promise.all([
    this.conversationModel
      .find({ userId })
      .sort({ createdAt: -1 })
      .skip(skip)
      .limit(limit)
      .exec(),
    this.conversationModel.countDocuments({ userId }),
  ]);
}

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				100
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    });

    return {
        conversations,
        total,
        page,
        totalPages: Math.ceil(total / limit),
    };
}

async searchConversationByMessageContent(
    userId: string,
    searchText: string,
    page: number = 1,
    limit: number = 10,
) {
    try {
        const skip = (page - 1) * limit;

        const results = await this.conversationModel.aggregate([
            {
                $lookup: {
                    from: 'messages',
                    localField: 'messages',
                    foreignField: '_id',
                    as: 'messages_details',
                },
            },
            {
                $match: {
                    userId: userId,
                    'messages_details.content': { $regex: searchText, $options: 'i' },
                },
            },
            {
                $facet: {
                    data: [{ $skip: skip }, { $limit: limit }],
                    totalCount: [{ $count: 'total' }],
                },
            },
        ]);

        const { data, totalCount } = results[0] || { data: [], totalCount: [] };
        const total = totalCount[0] ? totalCount[0].total : 0;

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				101
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    return {
      conversations: data,
      total,
      page,
      totalPages: Math.ceil(total / limit),
    };
  } catch (error) {
    throw error;
  }
}

async deleteConversation(conversationId: string, userId: string) {
  if (!isValidObjectId(conversationId)) {
    throw new NotFoundException('Invalid conversation id');
  }

  const conversation = await this.conversationModel
    .findOneAndDelete({ _id: conversationId, userId })
    .exec();

  if (!conversation) {
    throw new NotFoundException('Conversation not found');
  }

  await this.messageModel.deleteMany({ chatId: conversation._id });

  return {
    success: true,
    message: 'Conversation and its messages were successfully deleted',
  };
}
}

```

Лістинг файлу vector-store.service.ts

```

import { Injectable } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { Chroma } from '@langchain/community/vectorstores/chroma';
import { OpenAIEmbeddings } from '@langchain/openai';
import { ChromaClient, Collection, IEmbeddingFunction } from 'chromadb';
import pdfParse from 'pdf-parse';
import { RecursiveCharacterTextSplitter } from '@langchain/textsplitters';
import { v4 as uuidv4 } from 'uuid';
import { InjectModel } from '@nestjs/mongoose';

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				102
Змн.	Арк.	№ докум.	Підпис	Дата		

```

import { Model } from 'mongoose';
import {
  VectorCollection,
  VectorCollectionDocument,
} from 'src/schemas/vector-collection.schema';

interface PaginatedResult<T> {
  data: T[];
  total: number;
  page: number;
  limit: number;
}

class OpenAIEmbeddingFunction implements IEmbeddingFunction {
  private embeddings: OpenAIEmbeddings;

  constructor(apiKey: string) {
    this.embeddings = new OpenAIEmbeddings({
      openAIApiKey: apiKey,
      modelName: 'text-embedding-ada-002',
    });
  }

  async generate(texts: string[]): Promise<number[][]> {
    return this.embeddings.embedDocuments(texts);
  }
}

@Injectable()
export class VectorStoreService {
  private readonly chromaUrl: string;
  private readonly openAIApiKey: string;
  private readonly chromaClient: ChromaClient;
  private readonly embeddingFunction: OpenAIEmbeddingFunction;

  constructor(
    private readonly configService: ConfigService,
    @InjectModel(VectorCollection.name)
    private readonly vectorCollectionModel: Model<VectorCollectionDocument>,
  ) {
    this.chromaUrl =
      this.configService.get('CHROMA_DB_URL') || 'http://localhost:8000';
    this.openAIApiKey = this.configService.get<string>('OPENAI_API_KEY');
    this.chromaClient = new ChromaClient({ path: this.chromaUrl });
  }
}

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				103
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    this.embeddingFunction = new OpenAIEmbeddingFunction(this.openAPIKey);
}

private createChromaVectorStore(collectionName: string): Chroma {
    return new Chroma(
        new OpenAIEmbeddings({
            openAPIKey: this.openAPIKey,
            modelName: 'text-embedding-ada-002',
        }),
        {
            url: this.chromaUrl,
            collectionName,
        },
    );
}

async similaritySearch(
    query: string,
    k = 3,
    collectionName = 'documents-test',
) {
    const vectorStore = this.createChromaVectorStore(collectionName);
    return await vectorStore.similaritySearch(query, k);
}

async createCollection(collectionName: string) {
    const chromaCol = await this.chromaClient.createCollection({
        name: collectionName,
        embeddingFunction: this.embeddingFunction,
    });

    await this.vectorCollectionModel.updateOne(
        { name: collectionName },
        { $setOnInsert: { name: collectionName, files: [] } },
        { upsert: true },
    );

    return chromaCol;
}

async getCollections() {
    return await this.chromaClient.listCollections();
}

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				104
Змн.	Арк.	№ докум.	Підпис	Дата		

```

async getCollectionByName(collectionName: string): Promise<Collection> {
  return await this.chromaClient.getOrCreateCollection({
    name: collectionName,
    embeddingFunction: this.embeddingFunction,
  });
}

async addPdfToCollection(
  pdfBuffer: Buffer,
  fileName: string,
  collectionName = 'documents-test',
) {
  try {
    const data = await pdfParse(pdfBuffer);
    const text = data.text;
    const textSplitter = new RecursiveCharacterTextSplitter({
      chunkSize: 1000,
      chunkOverlap: 200,
    });
    const docs = await textSplitter.createDocuments([text]);

    const collection = await this.getCollectionByName(collectionName);
    const ids = docs.map(() => uuidv4());
    const nowIso = new Date().toISOString();
    const documents = docs.map((d) => d.pageContent);
    const metadatas = docs.map(() => ({
      source: fileName,
      addedAt: nowIso,
      updatedAt: nowIso,
    }));
    await collection.add({ ids, documents, metadatas });

    const now = new Date();
    const updResult = await this.vectorCollectionModel.updateOne(
      { name: collectionName, 'files.source': fileName },
      { $set: { 'files.$.lastUpdated': now } },
    );
    if (updResult.matchedCount === 0) {
      await this.vectorCollectionModel.updateOne(
        { name: collectionName },
        {
          $push: {
            files: {
              source: fileName,

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				105
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        firstAdded: now,
        lastUpdated: now,
    },
},
},
);
}

return { success: true, message: 'PDF added to collection successfully' };
} catch (error) {
    throw new Error(`Failed to add PDF to collection: ${error.message}`);
}
}

async deleteCollection(collectionName: string) {
    try {
        await this.chromaClient.deleteCollection({ name: collectionName });

        const deleteResult = await this.vectorCollectionModel.deleteOne({
            name: collectionName,
        });

        if (deleteResult.deletedCount === 0) {
            return {
                success: false,
                message: `Collection '${collectionName}' was removed from Chroma, but
no corresponding Mongo document was found`,
            };
        }

        return {
            success: true,
            message: `Collection '${collectionName}' deleted successfully from
Chroma and Mongo`,
        };
    } catch (error) {
        throw new Error(`Failed to delete collection: ${error.message}`);
    }
}

async listFilesInCollection(collectionName = 'documents-test') {
    const collection = await this.vectorCollectionModel
        .findOne({
            name: collectionName,

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				106
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    })
    .exec();
    if (!collection) {
        return [];
    }
    return collection.files;
}

async getAllCollectionsWithFiles(
    searchQuery?: string,
    page = 1,
    limit = 10,
): Promise<PaginatedResult<VectorCollectionDocument>> {
    const filter: any = {};

    if (searchQuery) {
        const re = new RegExp(searchQuery, 'i');
        filter.$or = [{ name: re }, { 'files.source': re }];
    }

    // Вычисляем, сколько пропустить
    const skip = (page - 1) * limit;

    const [data, total] = await Promise.all([
        this.vectorCollectionModel.find(filter).skip(skip).limit(limit).exec(),
        this.vectorCollectionModel.countDocuments(filter).exec(),
    ]);

    return { data, total, page, limit };
}

async syncChromaWithMongo(): Promise<{
    created: number;
    updated: number;
    removed: number;
}> {
    const listResp = await this.chromaClient.listCollections();
    const chromaNames = (listResp as any).collections
        ? ((listResp as any).collections as string[])
        : (listResp as string[]);

    const mongoDocs = await this.vectorCollectionModel.find().exec();
    const mongoNames = mongoDocs.map((doc) => doc.name);

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				107
Змн.	Арк.	№ докум.	Підпис	Дата		

```

let created = 0;
let updated = 0;
let removed = 0;

for (const name of chromaNames) {
  const rawFiles = await this.listFilesInCollection(name);
  const files = rawFiles.map((f) => ({
    source: f.source,
    firstAdded: !isNaN(Date.parse(f.firstAdded))
      ? new Date(f.firstAdded)
      : new Date(),
    lastUpdated: !isNaN(Date.parse(f.lastUpdated))
      ? new Date(f.lastUpdated)
      : new Date(),
  }));

  if (!mongoNames.includes(name)) {
    await this.vectorCollectionModel.create({ name, files });
    created++;
  } else {
    await this.vectorCollectionModel.updateOne(
      { name },
      { $set: { files } },
    );
    updated++;
  }
}

const toRemove = mongoNames.filter((n) => !chromaNames.includes(n));
for (const name of toRemove) {
  await this.vectorCollectionModel.deleteOne({ name });
  removed++;
}

return { created, updated, removed };
}

async deleteFileFromCollection(
  collectionName: string,
  fileName: string,
): Promise<{ success: boolean; message: string }> {
  const collection = await this.getCollectionByName(collectionName);

  await collection.delete({ where: { source: fileName } });

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				108
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const updateResult = await this.vectorCollectionModel.updateOne(
  { name: collectionName },
  { $pull: { files: { source: fileName } } },
);

if (updateResult.modifiedCount === 0) {
  return {
    success: false,
    message: `File '${fileName}' not found in Mongo for collection
'${collectionName}', but removed from Chroma.`,
  };
}

return {
  success: true,
  message: `File '${fileName}' removed from Chroma and Mongo.`,
};
}
}

```

Лістинг файлу conversation.controller.ts

```

import {
  Controller,
  Post,
  Body,
  Res,
  Get,
  Param,
  Query,
  UseGuards,
  NotFoundException,
  Delete,
  DefaultValuePipe,
  ParseIntPipe,
  BadRequestException,
  UseInterceptors,
  UploadedFiles,
} from '@nestjs/common';
import { Response } from 'express';
import { LlmService } from 'src/llm/llm.service';
import { SendRequestDto } from '../dto/send-request.dto';
import {

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				109
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    ApiOperation,
    ApiTags,
    ApiResponse,
    ApiBody,
    ApiBearerAuth,
    ApiQuery,
    ApiConsumes,
} from '@nestjs/swagger';
import { MessageService } from 'src/message/message.service';
import { Types } from 'mongoose';
import { JwtAuthGuard } from 'src/auth/guards/jwt.guard';
import { User } from 'src/common/decorators/user.decorator';
import { ConversationService } from './conversation.service';
import { BotsService } from 'src/bots/bots.service';
import { FilesInterceptor } from '@nestjs/platform-express';
import { memoryStorage } from 'multer';
import { FilePartItem } from 'src/schemas/message.schema';

@ApiTags('Conversation')
@UseGuards(JwtAuthGuard)
@ApiBearerAuth()
@Controller('conversation')
export class ConversationController {
  constructor(
    private readonly llmService: LlmService,
    private readonly messageService: MessageService,
    private readonly conversationService: ConversationService,
    private readonly botService: BotsService,
  ) {}

  @Get('search')
  @ApiOperation({ summary: 'Search conversations by message content' })
  @ApiQuery({
    name: 'query',
    description: 'Search string for message content',
    required: true,
    type: String,
  })
  @ApiQuery({
    name: 'page',
    description: 'Page number for pagination (default: 1)',
    required: false,
    type: Number,
  })

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				110
Змн.	Арк.	№ докум.	Підпис	Дата		

```

@ApiQuery({
  name: 'limit',
  description: 'Number of items per page (default: 10)',
  required: false,
  type: Number,
})
async searchConversations(
  @User('userId') userId: string,
  @Query('query') query: string,
  @Query('page', new DefaultValuePipe(1), ParseIntPipe) page: number,
  @Query('limit', new DefaultValuePipe(10), ParseIntPipe) limit: number,
) {
  if (!query) {
    throw new NotFoundException('Query parameter is required');
  }

  console.log(userId);
  return this.conversationService.searchConversationByMessageContent(
    userId,
    query,
    page,
    limit,
  );
}

@Get('/:conversationId')
@ApiOperation({ summary: 'Get conversation by ID' })
@ApiResponse({ status: 200, description: 'Returns conversation data' })
@ApiResponse({ status: 404, description: 'Conversation not found' })
async getConversation(
  @User('userId') userId: string,
  @Param('conversationId') conversationId: string,
) {
  try {
    const conversation = await this.conversationService.getConversationById(
      conversationId,
      userId,
    );
    if (!conversation) {
      throw new NotFoundException('Conversation not found');
    }
    return conversation;
  } catch (error) {
    return { error: error.message };
  }
}

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				111
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }
}

@Post()
@ApiOperation({ summary: 'Send a request to LLM with custom data' })
@ApiBody({
    type: SendRequestDto,
    description:
        'Request data. The request body should include the "prompt" field
containing the query string.',
})
@ApiResponse({
    status: 200,
    description: 'Response as a data stream (Server-Sent Events)',
    content: {
        'text/event-stream': {
            schema: { type: 'string', example: 'data: Your response\n\n' },
        },
    },
})
@ApiResponse({ status: 404, description: 'Conversation not found' })
@ApiResponse({ status: 500, description: 'Internal server error' })
@ApiConsumes('multipart/form-data')
@UseInterceptors(
    FilesInterceptor('files', 5, {
        storage: memoryStorage(),
        fileFilter: (req, file, cb) => {
            const allowed = ['image/jpeg', 'image/png', 'application/pdf'];
            if (allowed.includes(file.mimetype)) cb(null, true);
            else cb(new BadRequestException('Unsupported file type'), false);
        },
        limits: { fileSize: 10 * 1024 * 1024 },
    }),
)
async stream(
    @User('userId') userId: string,
    @Body() body: SendRequestDto,
    @Res() res: Response,
    @UploadedFiles() files: Express.Multer.File[],
) {
    res.setHeader('Content-Type', 'text/event-stream');
    res.setHeader('Cache-Control', 'no-cache');
    res.setHeader('Connection', 'keep-alive');

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				112
Змн.	Арк.	№ докум.	Підпис	Дата		

```

try {
  const fileParts = Array.isArray(files)
    ? files.map((file) => {
        const b64 = file.buffer.toString('base64');
        const isImage = file.mimetype.startsWith('image/');

        return {
          type: isImage ? 'image_url' : 'file',
          ...(isImage
            ? { image_url: { url: `data:${file.mimetype};base64,${b64}` } }
            : {
                file: {
                  file_data: `data:${file.mimetype};base64,${b64}`,
                  filename: file.originalname,
                },
              }),
        } as FilePartItem;
      })
    : undefined;

  console.log(fileParts);

  const conversation = body.chatId
    ? ((await this.conversationService.findById(body.chatId, userId)) ??
      (await this.conversationService.createConversation(userId)))
    : await this.conversationService.createConversation(userId);

  const lastMessage = await this.messageService.getLastMessage(
    conversation._id as Types.ObjectId,
  );

  const userMessage = await this.messageService.create({
    chatId: conversation._id as Types.ObjectId,
    role: 'user',
    content: body.prompt,
    parentId: (lastMessage?._id as Types.ObjectId) || new Types.ObjectId(),
    files: fileParts || [],
  });

  const collectionName = body.model
    ? ((await this.botService.getBotByName(body.model))?.chromaCollection ??
      process.env.DEFAULT_COLLECTION)
    : process.env.DEFAULT_COLLECTION;

  let fullBotResponse = '';

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				113
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const stream = this.llmService.generateStream(
  conversation._id as Types.ObjectId,
  body.prompt,
  collectionName,
  fileParts,
);

for await (const chunk of stream) {
  res.write(
    `data: ${JSON.stringify({
      content: chunk,
      chatId: conversation.id,
    })}\n\n`,
  );
  fullBotResponse += chunk;
}

const assistantMessage = await this.messageService.create({
  chatId: conversation._id as Types.ObjectId,
  role: 'assistant',
  content: fullBotResponse,
  parentId: userMessage._id as Types.ObjectId,
});

await this.messageService.addChild(
  userMessage._id as Types.ObjectId,
  assistantMessage._id as Types.ObjectId,
);
await this.conversationService.addMessages(
  conversation._id as Types.ObjectId,
  [
    userMessage._id as Types.ObjectId,
    assistantMessage._id as Types.ObjectId,
  ],
);

res.end();
} catch (error) {
  res.write(`data: ${JSON.stringify({ error: error.message })}\n\n`);
  res.end();
}
}

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				114
Змн.	Арк.	№ докум.	Підпис	Дата		

```

@Get()
@ApiOperation({ summary: 'Get user conversations' })
@ApiQuery({ name: 'page', required: false, type: Number })
@ApiQuery({ name: 'limit', required: false, type: Number })
@ApiResponse({
  status: 200,
  description: 'Returns a paginated list of user conversations',
})
async getUserConversations(
  @User('userId') userId: string,
  @Query('page') page: number = 1,
  @Query('limit') limit: number = 10,
) {
  return this.conversationService.getUserConversations(userId, page, limit);
}

@Delete('/:conversationId')
@ApiOperation({ summary: 'Delete a conversation and its messages' })
@ApiResponse({
  status: 200,
  description:
    'Conversation and its messages were successfully deleted. Returns an
    object with success flag and a message.',
  schema: {
    example: {
      success: true,
      message: 'Conversation and its messages were successfully deleted',
    },
  },
})
@ApiResponse({ status: 404, description: 'Conversation not found' })
async deleteConversation(
  @User('userId') userId: string,
  @Param('conversationId') conversationId: string,
) {
  return await this.conversationService.deleteConversation(
    conversationId,
    userId,
  );
}
}

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				115
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг файлу auth.ts:

```
import NextAuth, { NextAuthConfig } from 'next-auth';
import CredentialsProvider from 'next-auth/providers/credentials';
import { AuthService } from '@api/services/AuthService';
import { jwtDecode } from 'jwt-decode';

async function refreshAccessToken(token) {
  try {
    const response = await AuthService.refreshToken(token.refreshToken);
    if (!response.accessToken) {
      throw response;
    }

    return {
      ...token,
      accessToken: response.accessToken,
      refreshToken: response.refreshToken ?? token.refreshToken,
    };
  } catch (error) {
    console.log(error);
    return {
      ...token,
      error: 'RefreshAccessTokenError',
    };
  }
}

export const config = {
  providers: [
    CredentialsProvider({
      name: 'Credentials',
      credentials: {
        email: { label: 'Email', type: 'email' },
        password: {
          label: 'Password',
          type: 'password',
        },
      },
    },
  ],
  async authorize(credentials) {
    try {
      if (!credentials.email || !credentials.password) {
        return null;
      }
    }
  }
}
```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				116
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const response = await AuthService.login({
  email: String(credentials.email),
  password: String(credentials.password),
});

if (response.accessToken) {
  return {
    accessToken: response.accessToken,
    refreshToken: response.refreshToken,
    email: response.user.email,
    role: response.user.role,
    userImage: response.user.profile_picture,
    id: response.user.id,
    isEmailVerified: response.user.isEmailVerified,
  };
}

return null;
} catch (error) {
  console.log(error);
  return null;
}
},
}),
],
pages: {
  signIn: '/sign-up',
  error: '/sign-up',
},
callbacks: {
  jwt: async ({ token, account, user, trigger, session: newSession }) => {
    if (token?.accessToken) {
      const decodedToken = jwtDecode(String(token.accessToken));
      token.accessTokenExpires = Number(decodedToken?.exp) * 1000;
    }

    if (account && user) {
      return {
        ...token,
        accessToken: user.accessToken,
        refreshToken: user.refreshToken,
        email: user.email,
        role: user.role,

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				117
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        id: user.id,
        isEmailVerified: user.isEmailVerified,
        userImage: user.userImage,
    };
}
if (trigger === 'update' && newSession) {
    if (newSession.user?.email) {
        token.email = newSession.user.email;
    }
    if (newSession.userImage) {
        token.userImage = newSession.userImage;
    }
}

if (Date.now() < Number(token.accessTokenExpires)) {
    return token;
}

return refreshAccessToken(token);
},
session: async ({ session, token, trigger, newSession }) => {
    if (token) {
        session.accessToken = token.accessToken as string;
        session.refreshToken = token.refreshToken as string;
        session.email = token.email as string;
        session.role = token.role as 'user' | 'admin';
        session.id = token.id as string;
        session.isEmailVerified = token.isEmailVerified as boolean;
        session.userImage = token.userImage as string;
    }

    if (trigger === 'update' && newSession) {
        if (newSession.user?.email) {
            session.user.email = newSession.user.email;
        }
        if (newSession.userImage) {
            session.userImage = newSession.userImage;
        }
    }

    return session;
},
secret: process.env.AUTH_SECRET,

```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				118
Змн.	Арк.	№ докум.	Підпис	Дата		

```
} satisfies NextAuthConfig;
```

```
export const { handlers, auth, signIn, signOut } = NextAuth(config);
```

		Білецький Є.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				119
Змн.	Арк.	№ докум.	Підпис	Дата		