Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Магістр
(назва освітнього ступеня)

на тему: Розробка сервісу для читання та автоматичного перекладу книг з використанням технологій NuxtJS та Django.

Виконав(ла): студент(ка) 6 курсу, групи СПм-61
спеціальності 121 «Інженерія програмного Забезпечення»
(шифр і назва спеціальності)

	(підпис)	Грицишин П. А. (прізвище та ініціали)
Керівник	(підпис)	Стоянов Ю. М. (прізвище та ініціали)
Нормоконтроль	(підпис)	Стоянов Ю. М. (прізвище та ініціали)
Завідувач кафедри	(підпис)	Петрик М.Р. (прізвище та ініціали)
Рецензент	(підпис)	(прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М. Р.
(підпис) (прізвище та ініціали)
« » 2025 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Грицишину Павлу Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка сервісу для читання та автоматичного перекладу книг з використанням технологій NuxtJS та Django.

Керівник роботи Стоянов Юрій Миколайович, кандидат технічних наук, доцент кафедри ПІ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____

2. Термін подання студентом завершеної роботи _____

3. Вихідні дані до роботи Методичні вказівки; Технічна документація для: JavaScript, Vue, Nuxt Python, Django, Django REST Framework

4. Зміст роботи (перелік питань, які потрібно розробити)
Огляд конкурентів і порівняльний аналіз, обґрунтування вибору напрямку дослідження, вибір методології розробки, формування вимог до системи, проєктування відношень між акторами та прецедентами, визначення архітектури системи, проєктування головних класів системи, проєктування структури веб-застосунку, конструювання серверної частини, конструювання клієнтської частини, тестування та верифікація вимог, долікарська допомога при уражен ні струмом, питання безпеки при експлуатації електроустановок високих, надвисоких та ультрависоких частот

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Діаграма варіантів використання, діаграма класів, діаграма структури веб-застосунку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Осухівська Г. М., к.т.н., доц., зав. каф.		
	комп. систем та мереж		
Безпека в надзвичайних	Стручок В. С., с. в. каф. обладнання		
ситуаціях	харчових технологій		
Нормоконтроль	Стоянов Ю. М., к.т.н.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	**Примітка**
1.	Вибір та затвердження теми		**Виконано**
2.	Аналіз предметної області та технічного завдання		**Виконано**
3.	Вибір методології розробки		**Виконано**
4.	Формування вимог до системи та проєктування		**Виконано**
	відношень.		
5.	Визначення архітектури системи, проєктування		**Виконано**
	головних класів та структури веб-застосунку		
6.	Конструювання серверної частини		**Виконано**
7.	Конструювання клієнтської частини		**Виконано**
8.	Тестування та верифікація вимог		**Виконано**
9.	Безпека життєдіяльності, основи охорони праці		**Виконано**
10.	Підготовка презентації та доповіді		**Виконано**
11.	Нормоконтроль		**Виконано**
12.	Перевірка на плагіат		**Виконано**
13.	Попередній захист кваліфікаційної роботи		**Виконано**
14.	Захист кваліфікаційної роботи		

Студент _____ *Грицишин П. А.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Стоянов Ю. М .*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота магістра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2025, Сторінок 97, таблиць 5, рисунків 23, презентація.

Тема: Розробка сервісу для читання та автоматичного перекладу книг з використанням технологій NuxtJS та Django.

У межах виконання кваліфікаційної роботи магістра здійснено комплексне проєктування та програмну реалізацію веб-орієнтованої інформаційної системи, що забезпечує інтеграцію процесу читання автентичних іншомовних текстів із лінгвістичним навчанням. Основна увага при розробці приділялася вирішенню проблеми фрагментації уваги користувача, яка виникає внаслідок необхідності використання сторонніх інструментів перекладу, що негативно впливає на ефективність засвоєння матеріалу. Функціональна архітектура сервісу підтримує роботу з популярними форматами електронних видань, такими як PDF та EPUB, та передбачає використання технологій штучного інтелекту. Останні забезпечують реалізацію механізмів контекстного перекладу та функціонування інтелектуального чат-бота, який виступає в ролі цифрового асистента для пояснення складних мовних конструкцій. Технічна реалізація клієнтської складової виконана на основі фреймворку NuxtJS із дотриманням стандартів прогресивних веб-додатків (PWA), що гарантує адаптивність та високу швидкість відгуку інтерфейсу. Серверна логіка побудована на базі Django, що забезпечує надійну обробку даних та взаємодію з алгоритмами машинного навчання. Результати роботи підтверджують доцільність обраного архітектурного підходу для оптимізації процесу вивчення іноземних мов через читання.

Ключові слова: веб-сервіс, читання книг, автоматичний переклад, NuxtJS, Django, PWA, машинне навчання, Python.

ABSTRACT

Master's qualification work. Ternopil National Technical University named after Ivan Puluj, Department of Software Engineering, specialty 121 "Software Engineering". TNTU, 2025, Pages 97, tables 5, images 23, presentation.

Topic: Development of a service for reading and automatically translating books using NuxtJS and Django technologies.

This Master's thesis presents the comprehensive design and software implementation of a web-based system that integrates the reading of authentic foreign texts with language learning. The primary objective was to mitigate user attention fragmentation caused by the need for third-party translation tools, a factor that negatively impacts material assimilation. The platform supports standard electronic formats, such as PDF and EPUB, and leverages artificial intelligence to provide contextual translations and an intelligent chatbot assistant for explaining complex linguistic structures. The client-side is built on the NuxtJS framework, adhering to Progressive Web Application (PWA) standards to ensure high responsiveness and adaptability. The server-side logic, powered by Django, handles reliable data processing and machine learning algorithm integration. The results validate the effectiveness of this architectural approach in optimizing language acquisition through reading.

Keywords: web service, book reading, automatic translation, NuxtJS, Django, PWA, machine learning, Python.

ЗМІСТ

ВСТУП..... 8

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОЇ ОБЛАСТІ.....10

1.1 Аналіз предметної області..... 10

1.2 Постановка завдання та цілей..... 14

1.3 Пошук акторів та варіантів використання.....17

1.4 Опис ключових варіантів використання.....22

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА КОНСТРУЮВАННЯ СИСТЕМИ.26

2.1 Вибір процесу розробки.....26

2.2 Проектування архітектури системи..... 29

2.3 Побудова схеми бази даних..... 32

2.4 Побудова UML діаграми класів.....34

2.5 Вибір мови та середовища розробки..... 40

2.6 Реалізація основних класів системи..... 48

2.7 Розробка інтерфейсу користувача..... 52

3 ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ПІДТРИМКА..... 61

3.1 Тестування програмної системи.....61

3.1.1 Види та план тестування.....62

3.1.2 Розробка модульних тестів для backend.....63

3.1.3 Ручне тестування інтерфейсу користувача..... 68

3.2 Розгортання програмної системи та системні вимоги..... 74

3.3 Верифікація програмної системи..... 77

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ.....80

4.1 Охорона праці..... 80

4.2 Вплив виробничого середовища на працездатність та здоров’я користувачів комп’ютерів..... 83

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....89

ДОДАТКИ.....93

ДОДАТОК А.....94

ДОДАТОК Б.....95

ДОДАТОК В.....97

ВСТУП

Володіння іноземною мовою у сучасному науково-технічному та соціокультурному просторі розглядається не лише як професійна компетенція, але і як важливий інструмент інтелектуального розвитку та доступу до глобальних інформаційних ресурсів. Здатність опрацьовувати літературні джерела мовою оригіналу є ефективним методом самоосвіти, однак значний розрив між наявним рівнем мовної підготовки суб'єкта та лексико-граматичною складністю автентичних текстів створює суттєві перешкоди для ефективного навчання. Читання є критично важливим компонентом процесу засвоєння мови, оскільки воно забезпечує інтерналізацію граматичних структур та розширення словникового запасу в природному контексті. Водночас дефіцит лексичних знань та нерозуміння ідіоматичних зворотів призводять до необхідності частого звернення до допоміжних засобів перекладу, що фрагментує когнітивний процес та знижує ефективність сприйняття інформації. Прогрес у вивченні мови значною мірою залежить від опрацювання зрозумілого вхідного матеріалу, складність якого лише незначно перевищує поточні можливості учня, що актуалізує потребу в інструментах для адаптації тексту без порушення цілісності читання. Оптимізація взаємодії з іншомовним контентом за допомогою спеціалізованих програмних засобів дозволяє нівелювати бар'єри сприйняття та забезпечити доступ до складної літератури. Використання інтегрованих інструментів також сприяє зниженню психологічної напруги, пов'язаної з нерозумінням тексту, що позитивно впливає на мотивацію та сталість навчального процесу. Проблема перемикання контексту, яка виникає при переході між додатком для читання та зовнішнім словником, призводить до втрати концентрації та зниження продуктивності навчання. Відсутність адекватного програмного забезпечення, яке виконувало б функцію когнітивної підтримки, є основною причиною відмови користувачів від читання автентичних творів. Розроблена інформаційна система спрямована на вирішення зазначених проблем шляхом створення єдиного середовища, що поєднує процес читання з активним навчанням. Архітектура платформи

передбачає підтримку поширених форматів електронних документів, таких як PDF та EPUB, та інтеграцію модулів штучного інтелекту для надання контекстно-залежної допомоги. Функціонал системи включає механізми миттєвого перекладу та інтерактивний чат-бот, який виконує роль віртуального тьютора для роз’яснення лінгвістичних та культурних нюансів тексту. Програмна реалізація клієнтської частини базується на фреймворку NuxtJS із застосуванням методології прогресивних вебзастосунків (PWA), що забезпечує кросплатформність та високу швидкодію, наближену до нативних програмних рішень. Серверна частина, розроблена на базі фреймворку Django, забезпечує надійність обробки даних та інтеграцію з алгоритмами машинного навчання. У цій роботі розглядається ефективність застосування зазначеного технологічного стека для підвищення доступності іншомовної літератури та оптимізації процесу вивчення мови.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОЇ ОБЛАСТІ

Розробка сучасних інформаційних систем вимагає ретельного передпроектного дослідження, яке слугує фундаментом для прийняття обґрунтованих архітектурних та технологічних рішень. Перший розділ кваліфікаційної роботи присвячено комплексному аналізу вимог до програмної системи, що проектується. У ньому розглядається специфіка предметної області, пов’язаної з інтеграцією процесів читання електронних книг та їх автоматичного перекладу, а також виявляються основні проблеми існуючих на ринку аналогів. Метою цього етапу є чітка формалізація завдання та визначення цілей розробки, що дозволить створити затребуваний та конкурентоспроможний веб-сервіс. Особлива увага приділяється моделюванню поведінки майбутньої системи: ідентифікації ключових акторів, визначенню їхніх ролей та деталізації сценаріїв використання (Use Cases). Запорукою створення програмного забезпечення, що відповідає очікуванням клієнтів, є детальне опрацювання вимог на старті. Використовуючи цей підхід, можна не лише знизити ризики, але й значно спростити процес інтеграції інтерфейсу на NuxtJS із серверною частиною на Django.

1.1 Аналіз предметної області

Поточний період становлення цифрової ери вирізняється активним впровадженням новітніх систем у всі аспекти буття, причому центральну позицію тут посідають саме процеси набуття знань та індивідуальне зростання. В умовах глобалізації та посилення міжкультурних комунікацій володіння іноземними мовами перетворюється з додаткової навички на необхідну умову професійної конкурентоспроможності та повноцінної участі в світовому інформаційному обміні. Аналіз предметної області розробки сервісу для читання та автоматичного

перекладу книг дозволяє виявити ключові тенденції, потреби користувачів та технологічні виклики, що супроводжують процес вивчення мов через читання автентичних текстів. Читання іноземної літератури в оригіналі є надзвичайно дієвим засобом для розширення лексичного запасу та розуміння граматичних конструкцій у контексті, проте цей процес часто супроводжується значними труднощами для осіб, чий рівень володіння мовою є недостатнім для вільного сприйняття тексту. Основною проблемою, з якою стикаються користувачі, є необхідність постійного звернення до словників або сторонніх перекладачів, що призводить до фрагментації уваги, зниження темпу читання та втрати інтересу до твору. Традиційні методи роботи з текстом, такі як використання паперових словників або перемикання між вікнами програмного забезпечення на комп'ютері чи мобільному пристрої, створюють когнітивне навантаження, яке перешкоджає зануренню в сюжет та засвоєнню мовного матеріалу.

Еволюція мережевих рішень разом із штучним розумом надає унікальні перспективи для вдосконалення зазначеної процедури. Доцільність розробки профільних платформ продиктована посиленим запитом на рішення, котрі б інтегрували можливості цифрових книгосховищ із механізмами автоматизованого тлумачення. Сфера наукових інтересів включає технології аналізу людського мовлення, засади архітектури інтернет-систем, а також дидактичні й психологічні особливості опанування інших мов. Важливим аспектом аналізу є розуміння різниці між інтенсивним та екстенсивним читанням. Екстенсивне читання, яке передбачає опрацювання великих обсягів тексту для загального розуміння та задоволення, вимагає мінімізації зусиль на пошук невідомих слів. Існуючі на ринку рішення часто не забезпечують достатньої ергономічності: програми-рідери зазвичай мають обмежені функції перекладу, а онлайн-перекладачі не пристосовані для читання довгих текстів зі збереженням форматування та навігації. Відтак виникає потреба у створенні інтегрованого середовища, де користувач отримує миттєвий доступ до перекладу окремих слів, речень або абзаців безпосередньо в інтерфейсі читання. Наукові дослідження у сфері комп'ютерної лінгводидактики підтверджують, що використання інтерактивних

підказок та паралельних перекладів значно підвищує ефективність засвоєння нових лексичних одиниць, оскільки дозволяє встановити асоціативний зв’язок між словом та його значенням без відриву від контексту [1].

Технологічний аспект предметної області включає аналіз засобів реалізації клієнтської та серверної частин веб-додатків. Використання сучасних фреймворків, таких як NuxtJS для фронтенду та Django для бекенду, дозволяє створювати високопродуктивні односторінкові додатки (SPA) з підтримкою рендерингу на стороні сервера (SSR), що є критично важливим для забезпечення швидкодії та SEO-оптимізації. Django, як потужний інструмент на базі мови Python, пропонує потужний інструментарій для керування даними й обробки текстових файлів різних форматів та інтеграції з API сервісів машинного перекладу. Аналіз форматів електронних книг показує, що найбільш поширеними є EPUB, FB2 та PDF, кожен з яких має свою специфіку внутрішньої структури. Розробка універсального парсера, здатного коректно відображати структуру книги та розбивати її на логічні сегменти для перекладу, є одним із ключових технічних завдань у даній предметній області. Окремої уваги варта проблема фіксації прочитаного матеріалу й обміну відомостями між гаджетами клієнта, що диктує необхідність у стійкій структурі сховища й протоколах верифікації доступу.

Окремої уваги заслуговує аналіз якості автоматичного перекладу. За останні роки технології нейронного машинного перекладу (NMT) досягли значного прогресу, забезпечуючи високу точність передачі змісту навіть для складних художніх текстів. На відміну від статистичних методів, нейромережеві моделі враховують контекст усього речення, що дозволяє уникати грубих помилок та зберігати стилістичні особливості оригіналу. Інтеграція таких сервісів у платформу для читання дозволяє користувачеві отримувати переклад, який є достатнім для розуміння сюжету, хоча і може поступатися професійному літературному перекладу. Проте для цілей вивчення мови саме можливість миттєвого порівняння оригіналу з автоматичним перекладом є цінним інструментом, що розвиває навички мовної інтуїції. Дослідники зазначають, що доступність та зручність використання інструментів перекладу є визначальними

факторами, що впливають на мотивацію учнів продовжувати навчання та роботу зі складними текстами [2].

Ринок програмного забезпечення для вивчення мов є висококонкурентним, проте ніша спеціалізованих сервісів для читання з «розумним» перекладом залишається недостатньо заповненою. Більшість існуючих рішень орієнтовані або на короткі вправи (гейміфікація процесу навчання), або на професійний переклад документів. Сервіси, що пропонують доступ до адаптованих книг або книг з паралельним перекладом, часто мають обмежену бібліотеку та не дозволяють користувачам завантажувати власні файли. Це створює попит на гнучкі платформи, де користувач може самостійно обирати контент та налаштовувати параметри відображення. Соціальний аспект предметної області також включає можливість обміну досвідом, створення власних словників та обговорення прочитаного.

Важливим фактором при проектуванні подібних систем є забезпечення кросплатформності. Сучасний користувач очікує, що сервіс буде доступним як на настільному комп’ютері, так і на смартфоні чи планшеті без втрати функціональності. Використання адаптивної верстки та прогресивних веб-технологій (PWA) дозволяє досягти цього результату без необхідності розробки окремих нативних додатків для кожної операційної системи. Таким чином можна вагомо знизити вартість розробки та підтримки продукту, розширюючи потенційну аудиторію.

Економічна доцільність розробки такого сервісу підтверджується зростанням ринку онлайн-освіти та готовністю користувачів платити за якісні інструменти, що економлять час. Монетизація може здійснюватися через модель підписки, надання доступу до преміум-функцій перекладу або розширеного хмарного сховища для особистої бібліотеки. Однак, вагома частка успіху припадає саме на якість реалізації технічних рішень: швидкості завантаження сторінок, плавності анімацій, точності розпізнавання структури книги та релевантності перекладу.

Таким чином, аналіз предметної області демонструє, що розробка веб-сервісу для читання та автоматичного перекладу книг на базі NuxtJS та Django є актуальним науково-технічним завданням. Вона відповідає сучасним потребам суспільства у безперервній освіті та ефективному засвоєнні іноземних мов. Поєднання передових веб-технологій з можливостями машинного перекладу дозволяє створити інноваційний інструмент, що вирішує проблему мовного бар'єра при читанні автентичної літератури, забезпечуючи користувачам комфортне середовище для інтелектуального розвитку. Всебічна стратегія подолання наявних труднощів, яка поєднує вдосконалення візуальної взаємодії, використання потужних алгоритмів обробки даних на сервері та інтеграцію з сучасними API перекладу, створює передумови для створення конкурентоспроможного програмного продукту з високим потенціалом практичного застосування.

1.2 Постановка завдання та цілей

На основі проведеного аналізу предметної області та виявлених недоліків існуючих програмних рішень формулюється постановка завдання кваліфікаційної роботи, яка полягає у проектуванні та програмній реалізації веб-сервісу для читання електронних книг з інтегрованою функцією автоматичного перекладу. Основна проблема, яку покликана вирішити розробка, полягає у відсутності зручного інструментарію, що поєднував би ергономіку сучасних рідерів із можливостями миттєвого отримання контекстного перекладу без необхідності перемикання між різними застосунками. Це вимагає створення комплексної інформаційної системи, яка забезпечує повний цикл взаємодії користувача з текстовим контентом: від завантаження файлів у різних форматах до відображення тексту з інтерактивними елементами управління та збереженням прогресу читання.

Метою магістерської роботи є підвищення ефективності процесу читання іншомовної літератури шляхом розробки спеціалізованого веб-сервісу на базі технологій NuxtJS та Django, який автоматизує процеси обробки текстових файлів та взаємодії із зовнішніми системами машинного перекладу. Втілення задуму вимагає розв'язання групи суміжних технологічних питань, що стосуються одночасно програмного бекенду та візуальної частини майбутнього рішення.

Об’єктом дослідження є процес створення веб-орієнтованих інформаційних систем для обробки та візуалізації текстових даних. Предметом дослідження виступають методи та засоби розробки веб-додатків із використанням фреймворків NuxtJS та Django, алгоритми розбору форматів електронних книг та технології інтеграції сторонніх API для забезпечення функціональності перекладу.

Для реалізації визначеної мети необхідно виконати декомпозицію загального завдання на підзадачі. Першочергово потрібно спроектувати архітектуру, яка базуватиметься на засадах RESTful API, де серверна частина відповідає за логіку обробки даних, а клієнтська — за відображення інтерфейсу та взаємодію з користувачем. Вкрай важливо спроектувати архітектуру сховища даних, здатну надійно акумулювати відомості про клієнтів, їхню бібліотеку, службові атрибути, нотатки й архів запитів. Вирішальним кроком стане написання коду для обробки поширених типів файлів, насамперед FB2 та EPUB. Платформа мусить безпомилково ідентифікувати композицію тексту, від глав до верстки, і конвертувати все це у стандартизований вигляд для подальшої трансляції на фронтенд-інтерфейс.

Наступним завданням є розробка серверної частини додатку з використанням фреймворку Django та Django REST Framework. Це включає налаштування механізмів автентифікації та авторизації користувачів, створення кінцевих точок API для завантаження файлів, отримання списку книг та збереження прогресу читання. Особливу увагу слід приділити інтеграції із зовнішніми сервісами, які надають послуги перекладу тексту(наприклад, Google Translate API, DeepL або їх аналогами). Необхідно реалізувати механізм, який

дозволяє відправляти запити на переклад виділеного тексту та швидко отримувати відповідь, мінімізуючи затримки для користувача.

Паралельно з розробкою бекенду постає завдання реалізації клієнтської частини на базі NuxtJS. Вибір цього фреймворку зумовлений необхідністю створення швидкодіючого односторінкового додатку (SPA) з можливістю серверного рендерингу (SSR) для покращення продуктивності та SEO-оптимізації. Завдання включає розробку адаптивного інтерфейсу користувача, який забезпечує комфортне читання на пристроях з різним розширенням екрана. Важливим аспектом є реалізація інтерактивної взаємодії з текстом: виділення слів або речень, виклик спливаючих вікон з перекладом, налаштування шрифтів та колірних тем. Також необхідно забезпечити синхронізацію стану додатку між клієнтом і сервером, щоб користувач міг продовжити читання з того самого місця на іншому пристрої.

Додатковим завданням є тестування розробленого сервісу, яке включає перевірку коректності обробки файлів, стабільності роботи API, точності відображення верстки книг та швидкодії інтерфейсу. Важливо оцінити навантаження на систему при одночасній роботі кількох користувачів та оптимізувати запити до бази даних. Завершальним етапом роботи є розгортання веб-сервісу на хостингу та налаштування середовища для його функціонування в мережі Інтернет.

Отже, даний проект має на меті розробку повноцінної платформи, здатної ефективно нівелювати перешкоди в розумінні іноземної лексики, що виникають під час читання книг, демонструючи переваги використання сучасного стеку веб-технологій. Успішне виконання поставлених завдань дозволить отримати інструмент, що має як практичну цінність для кінцевих користувачів, так і теоретичне значення в контексті дослідження методів побудови спеціалізованих веб-систем.

1.3 Пошук акторів та варіантів використання

Процес проектування програмного забезпечення вимагає чіткого визначення функціональних вимог, що базуються на розумінні ролей користувачів та їхньої взаємодії з системою. Для формалізації цих вимог у рамках об’єктно-орієнтованого підходу традиційно застосовується уніфікована мова моделювання (UML), зокрема діаграми прецедентів (Use Case Diagrams). Пошук акторів та визначення варіантів використання є критичним для побудови архітектури веб-сервісу, оскільки він дозволяє окреслити межі системи та ідентифікувати зовнішні сутності, що ініціюють процеси обробки даних. Актор у контексті розроблюваної системи — це абстракція, що уособлює роль, яку виконує користувач або інша програмна система під час взаємодії з розроблюваним продуктом. Аналіз предметної області та технічного завдання дозволив виділити основних акторів, чиї потреби визначають функціонал клієнтської частини на NuxtJS та серверної логіки на Django.

Першим і ключовим актором системи є «Зареєстрований користувач». Це головна дійова особа, на задоволення потреб якої спрямована основна бізнес-логіка додатку. До повноважень цього актора належить управління особистою бібліотекою, завантаження файлів електронних книг, безпосереднє читання тексту та ініціювання запитів на переклад. Взаємодія цього актора з системою передбачає високий рівень інтерактивності, що забезпечується засобами фреймворку NuxtJS. Користувач очікує миттєвої реакції інтерфейсу на свої дії, такі як виділення тексту для перекладу або зміна налаштувань відображення сторінки. Збереження стану цього актора (прогрес читання, історія перекладів) покладається на серверну частину та базу даних, доступ до яких здійснюється через захищені API-ендпоінти. Дослідники зазначають, що коректна ідентифікація потреб основного користувача на ранніх етапах проектування дозволяє уникнути надмірної складності інтерфейсу та зосередитися на реалізації найбільш затребуваних функцій [3].

Другим актором виступає «Гість» (неавторизований користувач). Його можливості в системі суттєво обмежені з метою безпеки та стимулювання реєстрації. Гість має доступ лише до публічних сторінок веб-сервісу, які містять загальну інформацію про продукт, інструкції з використання та, можливо, демонстраційний фрагмент книги. Варіанти використання для цього актора зводяться до перегляду контенту та проходження процедури реєстрації або автентифікації. Незважаючи на обмежений функціонал, роль гостя є важливою для залучення аудиторії та первинного ознайомлення з можливостями платформи.

Третім актором є «Адміністратор». Це привілейований користувач, який відповідає за технічну підтримку та модерацію контенту. У контексті використання Django, роль адміністратора часто реалізується через вбудований адміністративний інтерфейс (Django Admin), що дозволяє керувати обліковими записами користувачів, переглядати статистику завантажень та системні логи. Адміністратор має повноваження блокувати доступ до системи у разі порушення правил використання, а також керувати глобальними налаштуваннями сервісу, наприклад, конфігурацією ключів доступу до зовнішніх API перекладу.

Окрім людей-користувачів, важливо виділити вторинного актора, яким є «Зовнішній сервіс перекладу» (API перекладача). Хоча ця сутність не є ініціатором взаємодії, вона відіграє критичну роль у реалізації основного функціоналу системи. Веб-сервіс надсилає запити до цього актора та отримує текстові дані, які потім відображаються користувачеві. Задоволення аудиторії напряму залежить від оперативності й стабільності сторонніх систем. Аналіз цієї взаємодії на етапі планування дає змогу впровадити алгоритми реагування на збої, зокрема ситуації відсутності зв'язку з провайдером послуг чи вичерпання наданих квот. Системний підхід до моделювання прецедентів вимагає чіткого розмежування відповідальності між внутрішніми компонентами системи та зовнішніми сервісами для забезпечення стабільності роботи програмного продукту [4].

Ідентифікація сценаріїв використання уможливлює чіткий опис реакцій програмного комплексу на команди операторів. Кожен випадок регламентує

порядок кроків задля досягнення важливого фіналу. Для цієї системи пріоритетними є процедури «Імпорт видання», що стартує серверний розбір контенту, та «Запит тлумачення», яка запускає наскрізну комунікацію між інтерфейсом користувача, базовою логікою та зовнішнім шлюзом обробки запитів. Важливо зазначити, що реалізація цих сценаріїв вимагає узгодженої роботи фронтенду та бекенду. Наприклад, при сценарії «Читання книги» система повинна не лише відобразити текст, але й асинхронно завантажувати наступні розділи для забезпечення плавності скролінгу, що є характерною вимогою для сучасних SPA-додатків [5].

Узагальнення виявлених акторів та їхніх функціональних можливостей дозволяє структурувати вимоги до системи у вигляді таблиці варіантів використання. Це слугує основою для подальшого проектування діаграм класів та розробки програмного коду.

Таблиця 1.3.1 - Варіанти використання системи

ID	**Назва варіанта використання**	**Актор**	**Опис**
UC-01	Реєстрація користувача	Гість	Створення нового облікового запису шляхом введення персональних даних (email, пароль).
UC-02	Автентифікація	Гість	Вхід у систему за допомогою облікових даних для отримання доступу до особистого кабінету.
UC-03	Завантаження книги	Зареєстрований користувач	Передача файлу електронної книги (EPUB, FB2) на сервер для обробки та додавання до бібліотеки.
UC-04	Читання книги	Зареєстрований користувач	Відкриття тексту книги в інтерфейсі рідера з можливістю навігації та налаштування відображення.

Продовження таблиці 1.3.1.

ID	Назва варіанта використання	Актор	Опис
UC-05	Переклад фрагмента тексту	Зареєстрований користувач	Виділення слова або речення та отримання автоматичного перекладу через взаємодію із зовнішнім API.
UC-06	Переклад всієї сторінки	Зареєстрований користувач	Переклад цілого фрагменту сторінки через взаємодію із зовнішнім API
UC-07	Спілкування з чат-ботом	Зареєстрований користувач	Спілкування з чат-ботом для в контексті відкритого документу
UC-08	Керування бібліотекою	Зареєстрований користувач	Перегляд списку завантажених книг, їх видалення або сортування за критеріями.
UC-09	Збереження прогресу	Система (автоматично) / Користувач	Фіксація поточної позиції читання в базі даних для синхронізації між пристроями.
UC-10	Керування користувачами	Адміністратор	Перегляд списку користувачів, блокування або видалення акаунтів через панель адміністрування.
UC-11	Налаштування профілю	Зареєстрований користувач	Зміна особистих даних, пароля та інтерфейсних налаштувань (мова, тема).
UC-12	Налаштування профілю	Зареєстрований користувач	Зміна особистих даних, пароля та інтерфейсних налаштувань (мова, тема).

На основі проведеного аналізу функціональних вимог, ідентифікованих ролей та визначеного переліку сценаріїв взаємодії розроблено діаграму варіантів використання. Ця графічна модель візуалізує межі проектованої інформаційної системи, демонструючи зв’язки між активними сутностями та функціональними процесами, які вони ініціюють. Діаграма відображає ієрархію акторів, де зареєстрований користувач володіє розширеним набором повноважень порівняно з гостем, а також ілюструє залежність функціоналу перекладу від зовнішнього

програмного інтерфейсу. Представлена схема слугує концептуальною основою для подальшого проектування архітектури додатку та деталізації технічних завдань.

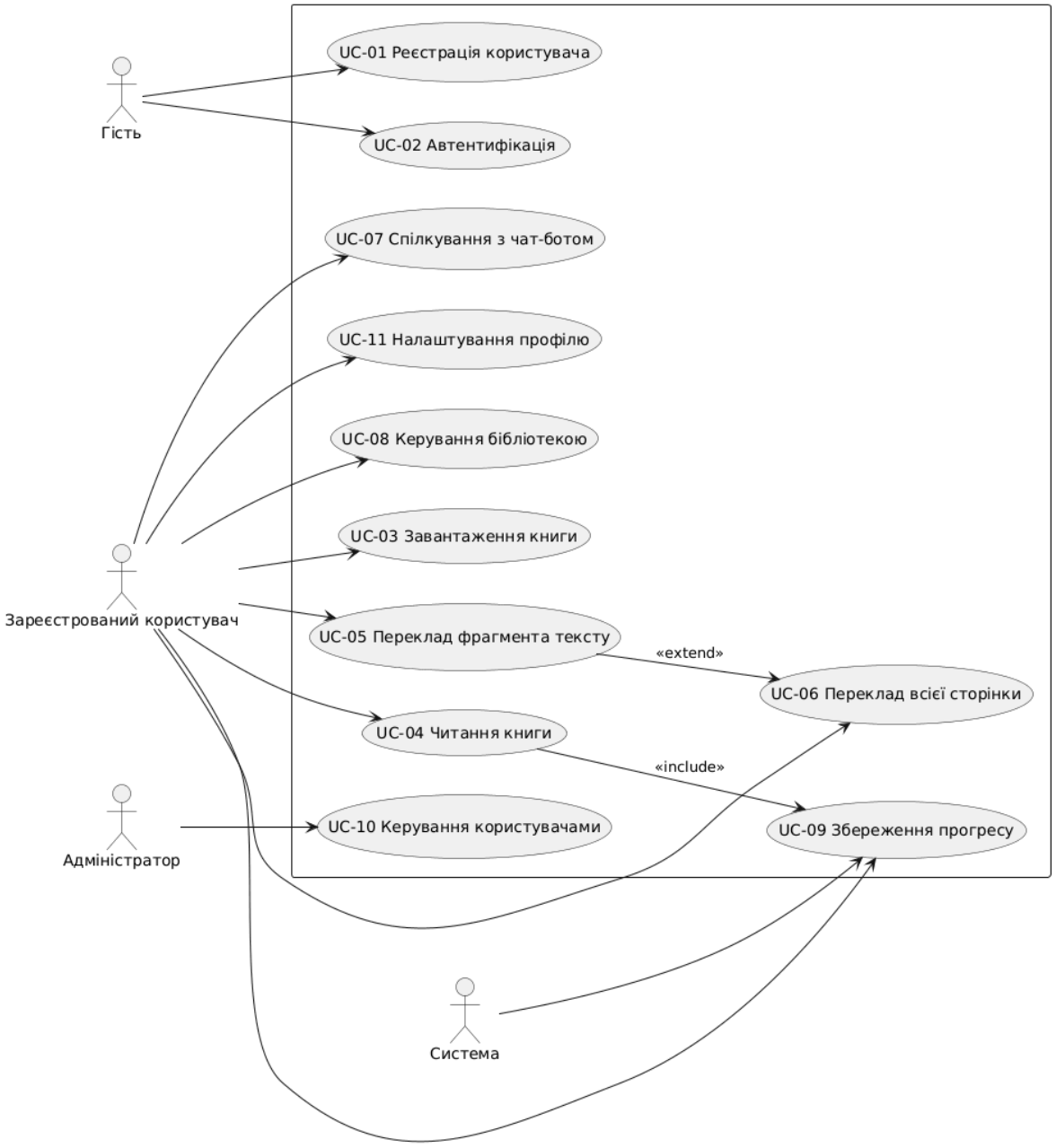

Рисунок 1.3.1 - Діаграма варіантів використання.

1.4 Опис ключових варіантів використання

Уточнення сценаріїв роботи виступає обов'язковим кроком при створенні програм, забезпечуючи трансформацію широких комерційних очікувань у конкретний опис технічних реакцій самого комплексу. Опис ключових сценаріїв взаємодії визначає послідовність дій акторів та реакцій системи, формуючи основу для подальшої розробки алгоритмів, структури бази даних та тестових наборів. З огляду на архітектурні особливості проекту, що базується на відокремленні клієнтської частини (NuxtJS) та серверної логіки (Django), критично важливо описати потоки даних та процеси обробки інформації для найбільш значущих прецедентів. До таких належать процеси, що забезпечують основну цінність продукту: завантаження та обробка контенту, безпосереднє читання, отримання перекладу та інтелектуальна взаємодія з текстом, а також синхронізація прогресу користувача. Нижче наведено детальний опис логіки виконання обраних варіантів використання.

Варіант використання UC-03 «Завантаження книги» є фундаментальним для наповнення системи контентом і ініціюється актором «Зареєстрований користувач». Для початку роботи вимагається активна сесія користувача. Процедура стартує із завантаження через інтерфейс документа у розширенні PDF або EPUB. Клієнтська частина на базі NuxtJS здійснює попередню фільтрацію, аналізуючи тип та "вагу" файлу, що запобігає надмірному використанню каналу та відсіює некоректні дані ще до моменту відправки. Після успішної перевірки клієнт формує об’єкт FormData та надсилає асинхронний HTTP-запит (POST) до відповідної кінцевої точки API серверу на базі Django. Серверна частина приймає запит та виконує вторинну валідацію, перевіряючи цілісність файлу та відповідність MIME-типу. Ключовим етапом є запуск модуля парсингу, який розбирає структуру електронної книги. Алгоритм зчитує метадані (назва, автор, обкладинка), виділяє зміст та розбиває текст на логічні частини (розділи, параграфи), одночасно зберігаючи початкові стилі оформлення книги. Отримані дані нормалізуються та зберігаються у реляційній базі даних: створюється запис у

таблиці книг, а текстовий вміст розподіляється по зв’язаних таблицях розділів. У випадку успішного збереження сервер повертає статус 201 Created та ID нової книги, після чого клієнтський додаток оновлює список бібліотеки користувача. Альтернативний потік виникає у разі пошкодження файлу або непідтримуваного формату кодування, при цьому сервер повертає повідомлення про помилку, яке відображається користувачеві.

Ключова процедура роботи з сервісом зафіксована у прецеденті використання UC-04 «Читання книги». Сценарій активується при виборі конкретної книги з бібліотеки. Система виконує запит до серверу для отримання метаданих книги та змісту першого розділу (або того, на якому користувач зупинився). Для забезпечення високої продуктивності використовується механізм пагінації або «лінивого завантаження» (lazy loading), коли текст завантажується порціями. Сервер Django серіалізує текстові дані у формат JSON і передає їх клієнту. Фреймворк NuxtJS відповідає за рендеринг отриманих даних, трансформуючи їх у HTML-структуру. Важливим аспектом є динамічне застосування налаштувань користувача, таких як розмір шрифту, міжрядковий інтервал та колірна тема (світла/темна), що реалізується через реактивні змінні стану та CSS-класи. Система також повинна забезпечити навігацію між розділами, використовуючи зміст книги. Постумова виконання сценарію полягає у відображенні тексту на екрані пристрою в зручному для читання форматі з можливістю подальшої інтеракції.

Варіант використання UC-05 «Переклад фрагмента тексту» є ключовою функціональною особливістю сервісу, що відрізняє його від стандартних рідеров. Сценарій розпочинається, коли користувач виділяє слово або частину речення в інтерфейсі читання. Клієнтський додаток перехоплює подію виділення (selection event) та відображає контекстне меню з кнопкою перекладу. Натиск активує відправку запиту засобами NuxtJS до локального API, передаючи виділений текст та мовні параметри. Сервер Django виступає в ролі посередника (proxy), що дозволяє приховати ключі доступу до зовнішніх сервісів та обійти обмеження політики CORS. Сервер перевіряє наявність кешованого перекладу для даного

фрагмента в базі даних. Якщо переклад відсутній, сервер надсилає запит до зовнішнього API (наприклад, Google Translate або DeepL). Отримавши відповідь, система зберігає її в кеші для оптимізації майбутніх запитів та повертає результат клієнту. На стороні інтерфейсу переклад відображається у вигляді спливаючого вікна (popover) або модального вікна поруч із виділеним текстом. Цей процес має відбуватися з мінімальною затримкою, щоб не порушувати концентрацію читача.

Варіант використання UC-07 «Спілкування з чат-ботом» представляє розширені можливості аналізу тексту з використанням штучного інтелекту. Цей сценарій дозволяє користувачеві отримати пояснення складних моментів у контексті книги. Дії розпочинаються з відкриття панелі чату поруч із текстом книги. Коли користувач вводить запитання, система автоматично захоплює контекст — поточний видимий параграф або сторінку книги. Клієнтський додаток надсилає на сервер запит, що містить запитання користувача та ідентифікатор фрагмента тексту. Сервер формує складний промпт для LLM (Large Language Model), додаючи системні інструкції (наприклад, «поясни значення фрази в контексті наданого уривку») та сам текст книги. Отриманий від AI-сервісу результат обробляється та надсилається назад користувачеві. Важливою технічною деталлю є підтримка історії діалогу в межах сесії читання, що дозволяє користувачеві ставити уточнюючі запитання. Цей варіант використання вимагає ретельної обробки помилок, пов’язаних з лімітами токенів та часом очікування відповіді від зовнішнього API.

Сценарій UC-09 «Збереження прогресу» гарантує безперервність роботи з платформою незалежно від гаджета, реалізуючи повноцінну кросплатформність. Цей процес може ініціюватися як автоматично (періодично або при переході на нову сторінку), так і вручну (при закритті книги). Технічна реалізація передбачає відстеження позиції скролу або ідентифікатора поточного параграфа. Для запобігання надмірній кількості запитів до серверу використовується техніка «debouncing» — відправка даних лише після того, як користувач припинив скролінг на певний час. Дані про прогрес (ID книги, номер розділу, відсоток прочитаного або конкретний селектор елемента) надсилаються на сервер через

захищений канал. Бекенд виконує модифікацію даних у сховищі, які закріплені за профілем поточного клієнта. Постумова цього сценарію гарантує, що при наступному відкритті цієї книги (UC-04) на будь-якому пристрої система автоматично відновить відображення тексту з останньої зафіксованої позиції. Важливим аспектом є вирішення конфліктів синхронізації, коли читання відбувається одночасно на декількох пристроях, для чого зазвичай використовується мітка часу останнього оновлення.

Таким чином, детальний опис наведених варіантів використання демонструє тісний взаємозв’язок між клієнтською та серверною частинами системи. Реалізація цих сценаріїв вимагає побудови надійної архітектури, здатної ефективно обробляти текстові дані, керувати станом додатку та інтегруватися із зовнішніми сервісами. Детальний опис алгоритмів функціонування дає змогу розмежувати сфери контролю клієнтської й серверної сторін, стаючи при цьому надійним путівником у процесі технічного втілення проекту.

У першому розділі магістерської роботи проведено комплексний аналіз предметної області, в результаті якого обґрунтовано актуальність створення веб-сервісу для читання та автоматичного перекладу книг. Визначено, що інтеграція інструментів машинного перекладу безпосередньо в інтерфейс рідера здатна суттєво підвищити ефективність вивчення іноземних мов та покращити користувацький досвід. На основі виявлених потреб сформульовано технічне завдання та визначено технологічний стек, що включає фреймворки NuxtJS та Django. Розроблено моделі варіантів використання, ідентифіковано основних акторів системи та детально описано ключові сценарії взаємодії, такі як завантаження контенту, контекстний переклад та використання інтелектуальних чат-ботів. Результати аналізу вимог, представлені у вигляді діаграм та специфікацій, є необхідною базою для переходу до етапу системного проектування, розробки архітектури бази даних та програмної реалізації компонентів сервісу.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА КОНСТРУЮВАННЯ СИСТЕМИ

Ґрунтуючись на результатах, отриманих в процесі виконання минулого розділу, наступним кроком є стадія безпосереднього конструювання й технічного створення платформи. Другий розділ роботи присвячується комплексному опрацюванню питань побудови архітектурного каркасу та кодової бази веб-сервісу, орієнтованого на читання й переклад текстів. Ключове завдання — втілення теоретичних моделей і користувацьких сценаріїв у дієвий програмний засіб, який задовольняє чинні критерії якості та норми безпеки. У розділі послідовно розглядаються методологічні аспекти організації процесу розробки, обґрунтовується вибір інструментальних засобів та середовища програмування. Ключову роль відведено проектуванню інформаційної структури, що включає розробку схем бази даних та об’єктно-орієнтоване моделювання класів системи, що є необхідною умовою для забезпечення цілісності даних та гнучкості архітектури. Окрему увагу приділено опису практичної реалізації серверної бізнес-логіки на базі фреймворку Django та створенню інтерактивного клієнтського інтерфейсу засобами NuxtJS, що в сукупності формує завершене рішення поставленого науково-технічного завдання.

2.1 Вибір процесу розробки

Ефективність створення сучасних програмних продуктів значною мірою залежить від обраної методології управління життєвим циклом програмного забезпечення (SDLC). Вибір процесу розробки є стратегічним рішенням, яке визначає порядок виконання етапів проектування, написання коду, тестування та розгортання, а також механізми контролю якості та адаптації до змін у вимогах. У контексті магістерської кваліфікаційної роботи, об’єктом якої є веб-сервіс зі

складною клієнт-серверною архітектурою на базі NuxtJS та Django, процес розробки повинен забезпечувати гнучкість, можливість ітеративного нарощування функціоналу та ефективне виявлення помилок на ранніх стадіях. Традиційні каскадні моделі (Waterfall), що передбачають сувору послідовність етапів без можливості повернення до попередніх фаз, є малоефективними для веб-розробки, де вимоги можуть уточнюватися в процесі реалізації, а технологічні рішення потребують перевірки на практиці. Як зазначає Іан Соммервіль у фундаментальній праці з інженерії програмного забезпечення, ітеративні підходи дозволяють значно знизити ризики провалу проекту шляхом розбиття його на керовані частини, кожна з яких завершується створенням робочого прототипу або інкременту продукту [6].

З огляду на специфіку проекту, для розробки сервісу читання та перекладу книг обрано гнучку методологію (Agile Software Development). Цей підхід базується на принципах ітеративної та інкрементної розробки, де створення системи відбувається циклічно, невеликими кроками, що дозволяє постійно оцінювати проміжні результати та вносити корективи. Використання Agile є особливо доцільним при роботі з сучасними JavaScript-фреймворками та RESTful API, оскільки це дозволяє розробляти клієнтську та серверну частини паралельно, узгоджуючи інтерфейси взаємодії в межах кожної ітерації. Ключовою перевагою гнучкого підходу є орієнтація на створення працюючого програмного забезпечення як основного показника прогресу, що відповідає цілям кваліфікаційної роботи — отримати готовий до використання продукт. Згідно з маніфестом Agile, реагування на зміни є важливішим за слідування початковому плану, що є критичним фактором успіху в умовах дослідження нових технологій перекладу та обробки природної мови [7].

В рамках обраної методології процес розробки буде організовано за адаптованою моделлю Scrum, яка передбачає поділ часу на фіксовані інтервали — спринти. Для даного проекту оптимальною тривалістю спринту визначено два тижні. Кожен спринт розпочинається з планування, під час якого з загального переліку вимог (Product Backlog) обираються пріоритетні завдання для реалізації.

Наприклад, перший спринт може бути присвячений проектуванню бази даних та реалізації базової автентифікації на Django, тоді як наступні фокусуються на розробці інтерфейсу рідера на NuxtJS та інтеграції зовнішніх API. Завершення кожного спринту супроводжується тестуванням реалізованого функціоналу та інтеграцією його в основну гілку проекту (master/main branch). Такий підхід дозволяє дотримуватися принципу безперервної інтеграції (CI), що забезпечує стабільність кодової бази та спрощує пошук дефектів.

Важливим аспектом обраного процесу є управління вимогами через користувацькі історії (User Stories), окреслені під час опрацювання сценаріїв використання. Кожна історія оцінюється за складністю, що дозволяє прогнозувати терміни виконання робіт. Використання принципів Agile також сприяє кращій організації коду: необхідність частого внесення змін стимулює дотримання принципів SOLID та написання модульних тестів. Роберт Мартін, один із провідних експертів у галузі гнучкої розробки, підкреслює, що якісна архітектура та чистий код є невід'ємними складовими Agile-процесів, оскільки вони забезпечують низьку вартість змін у майбутньому [8]. Для даного проекту це означає створення слабкозв'язаних компонентів Vue.js на фронтенді та незалежних додатків (apps) в межах проекту Django на бекенді.

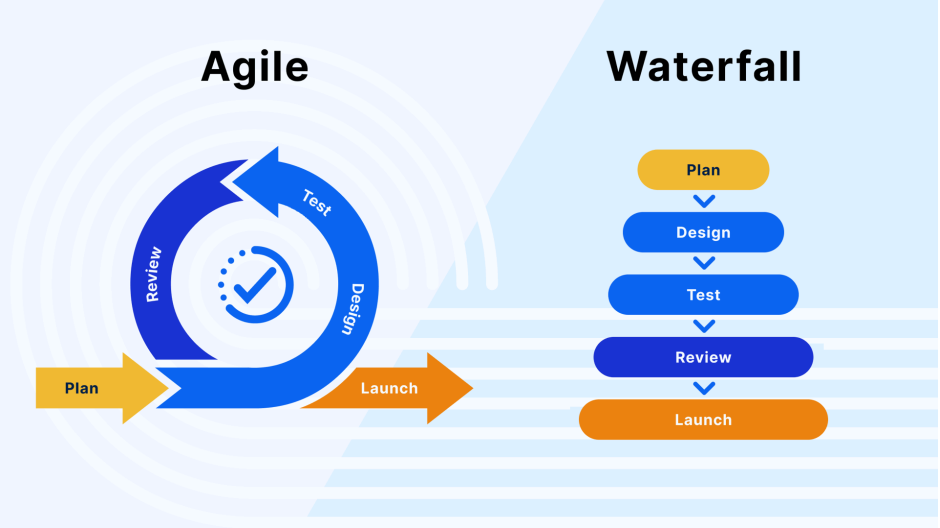

Рисунок 2.1.1 - Візуалізація методологій Aglile та Waterfall[9].

Організація процесу розробки також включає використання сучасних інструментів контролю версій (Git) та систем управління завданнями. Завдяки цьому вдається систематизувати діяльність, моніторити виконання задач і фіксувати ухвалені інженерні рішення. Ця стратегія гарантує прозорість створення продукту й дає змогу оперативно реагувати на перешкоди (скажімо, ліміти зовнішніх API чи нюанси розбору форматів), забезпечуючи поступовий рух до фінішу. Тож використання гнучких підходів є цілком виправданим і адекватним щодо складності й специфіки цілей, поставлених у даному магістерському дослідженні.

2.2 Проектування архітектури системи

Проектування архітектури програмного забезпечення є фундаментальним етапом розробки, який визначає структуру майбутньої системи, її компоненти, взаємозв’язки між ними, а також принципи та настанови, що керують їхнім проектуванням та еволюцією. Якісна архітектура забезпечує виконання нефункціональних вимог, таких як надійність, масштабованість, безпека та зручність супроводу, що є критично важливим для веб-сервісів із складною бізнес-логікою. У контексті розробки сервісу для читання та перекладу книг обрано багаторівневу клієнт-серверну архітектуру (Client-Server Architecture), реалізовану за принципом роз’єднаного фронтенду та бекенду (Decoupled Architecture). Подібна модель диктує суворий розподіл ролей між UI-частиною та бекенд-процесами, завдяки чому досягається автономність у розробці та тюнінгу кожного модуля. Зв'язок компонентів реалізовано через універсальний інтерфейс REST API, який спирається на стандарти HTTP та використовує JSON-об'єкти для обміну інформаційними потоками. Лен Басс та інші дослідники у галузі програмної інженерії зазначають, що такий розподіл є стандартом для сучасних

веб-додатків, оскільки він підвищує гнучкість системи та спрощує інтеграцію з мобільними платформами у майбутньому [10].

Клієнтська частина системи (Frontend) реалізується на базі фреймворку NuxtJS, який є надбудовою над бібліотекою Vue.js. Архітектурно цей рівень відповідає за презентацію даних та безпосередню взаємодію з користувачем. Використання NuxtJS дозволяє реалізувати гібридну модель рендерингу, поєднуючи переваги односторінкових додатків (SPA) з можливістю серверного рендерингу (SSR). Це забезпечує швидке завантаження першої сторінки, що є важливим для утримання уваги користувача, та коректну індексацію контенту пошуковими системами. Внутрішня структура клієнтського додатку базується на компонентному підході: архітектура фронтенду базується на атомарних компонентах, що допускають реюз, наприклад: навігаційний блок, зона читання та віджет тлумачення. Менеджмент глобального стану, що охоплює активний текст, конфігурацію та статус доступу, реалізовано централізовано через інструментарій Pinia. Така організація коду сприяє його читабельності та спрощує процес тестування окремих елементів інтерфейсу [11].

Серверна частина (Backend) побудована на базі високорівневого фреймворку Django та розширення Django REST Framework (DRF). Архітектура бекенду слідує патерну MVT (Model-View-Template), однак у контексті створення API роль шаблонів (Templates) мінімізована, а основний акцент зміщено на серіалізатори (Serializers) та представлення (Views). Сервер виконує роль центрального вузла обробки бізнес-логіки: він забезпечує автентифікацію користувачів через механізм JWT (JSON Web Tokens), валідацію вхідних даних, обробку файлів електронних книг та модифікацію даних в БД. Особливістю архітектури є реалізація сервісного шару (Service Layer), який інкапсулює логіку взаємодії із зовнішніми системами машинного перекладу. Це дозволяє абстрагувати контролери від деталей реалізації конкретних API (Google, DeepL тощо) та легко змінювати постачальника послуг перекладу без необхідності переписування основного коду. Річардсон та Рубі підкреслюють, що використання REST-архітектури дозволяє досягти високої

масштабованості системи завдяки відсутності збереження стану клієнта на сервері між запитами (statelessness) [12].

Рівень даних представлений реляційною системою управління базами даних (СКБД), яка взаємодіє з сервером через механізм ORM (Object-Relational Mapping), вбудований у Django. Це дозволяє працювати з даними як з об’єктами мови Python, абстрагуючись від SQL-запитів, що підвищує безпеку системи та захищає від атак типу SQL-injection. База даних зберігає дані юзерів, метадані книг, структуру розібраних файлів (розділи, параграфи) та історію перекладів. Для забезпечення швидкодії системи, особливо при повторних запитах на переклад одних і тих самих слів, доцільним є впровадження механізму кешування (наприклад, з використанням Redis), який розташовується між сервером та зовнішніми API.

Інфраструктурний аспект архітектури передбачає використання контейнеризації за допомогою Docker. Для кожного елемента системи створюється окремий та ізольований Docker контейнер, що гарантує ідентичність середовища розробки та середовища розгортання (production). Вхідною точкою в систему слугує веб-сервер Nginx, який працює як зворотний проксі-сервер (Reverse Proxy), розподіляючи запити між статичними файлами фронтенду та API-запитами до бекенду. Така архітектура забезпечує високу стійкість до навантажень та спрощує процес горизонтального масштабування. Сучасні дослідження підтверджують, що контейнеризована мікросервісна або модульна монолітна архітектура є найбільш ефективним рішенням для веб-платформ, орієнтованих на швидкий цикл оновлень та стабільність роботи [13].

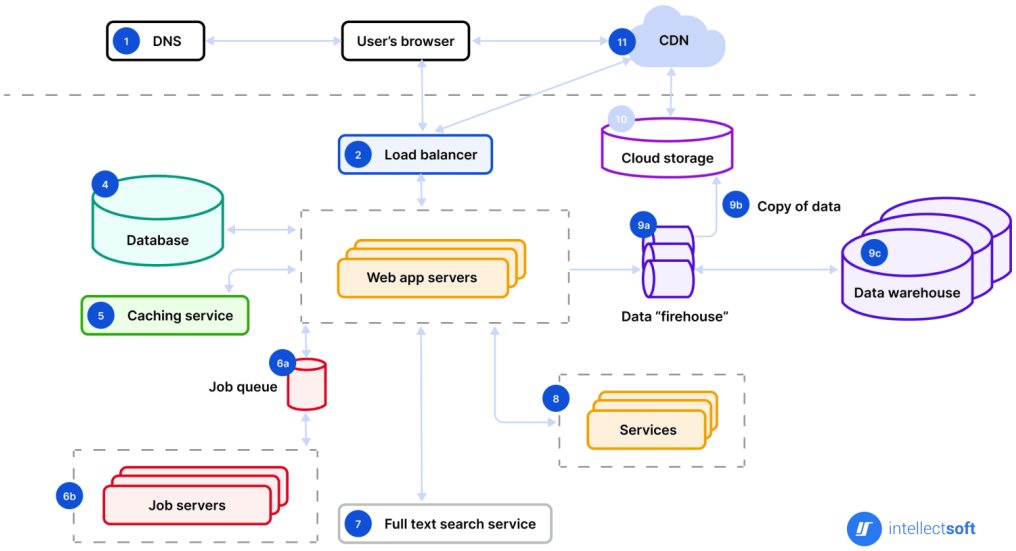

Рисунок 2.2.1 - Структурна схема архітектури веб-сервісу[14].

Таким чином, спроектована архітектура системи являє собою збалансоване поєднання перевірених часом патернів проектування та сучасних технологічних рішень. Розділення клієнтської та серверної частин, використання REST API для комунікації та застосування компонентного підходу на фронтенді створюють надійну основу для реалізації функціоналу читання та перекладу книг.

2.3 Побудова схеми бази даних

Проектування структури бази даних є важливим кроком для забезпечення ефективності, масштабованості та цілісності даних веб-сервісу. Для проекту Neiread обрано реляційну модель даних на основі СУБД PostgreSQL, що дозволяє гнучко структурувати інформацію та підтримувати складні зв'язки між сутностями. База даних організована за модульним принципом і складається з трьох основних компонентів: userprofile для управління обліковими записами, books для зберігання контенту та chat для реалізації інтелектуальної взаємодії.

Такий підхід сприяє логічній ізоляції функціональних блоків та полегшує подальший супровід системи.

Ядром системи є кастомна модель User (у модулі userprofile), яка розширює стандартний клас AbstractUser фреймворку Django. Вона зберігає ідентифікаційні дані користувача, де ключовим полем для автентифікації виступає email (замість стандартного імені користувача), а також службову інформацію: статус активності, дату реєстрації та права доступу. Ця сутність є фундаментом для всіх подальших зв'язків, забезпечуючи розмежування доступу до даних.

Модуль books містить таблицю Book, яка зберігає інформацію про завантажені твори. Реалізовано зв’язок «один-до-багатьох» через поле user_fk, що дозволяє кожному користувачеві формувати власну бібліотеку. Окрім стандартних метаданих (назва, автор, посилання на обкладинку), модель містить поля origin_file та markdown_file. Вони зберігають шляхи до вихідного файлу та його конвертованої версії у файловій системі, що дозволяє уникнути перевантаження бази даних бінарними даними. Для реалізації функції збереження прогресу читання (UC-09) передбачено поля pages (загальна кількість сторінок) та current_page (поточна позиція), що дозволяє синхронізувати стан читання між сесіями.

Модуль chat відповідає за збереження історії взаємодії користувача з AI-асистентом. Таблиця ChatSession представляє окрему сесію діалогу, яка має подвійну прив’язку: до користувача (user) та до конкретної книги (book). Це забезпечує контекстуальність діалогу, дозволяючи асистенту «розуміти», про який твір йде мова. Самі повідомлення зберігаються у таблиці ChatMessage, зв’язаній із сесією відношенням «один-до-багатьох». Поле role розрізняє репліки користувача та відповіді системи, а часова мітка created_at забезпечує правильну хронологію діалогу.

Для наочної демонстрації спроектованої структури та взаємозв’язків між описаними сутностями нижче наведено діаграму бази даних.

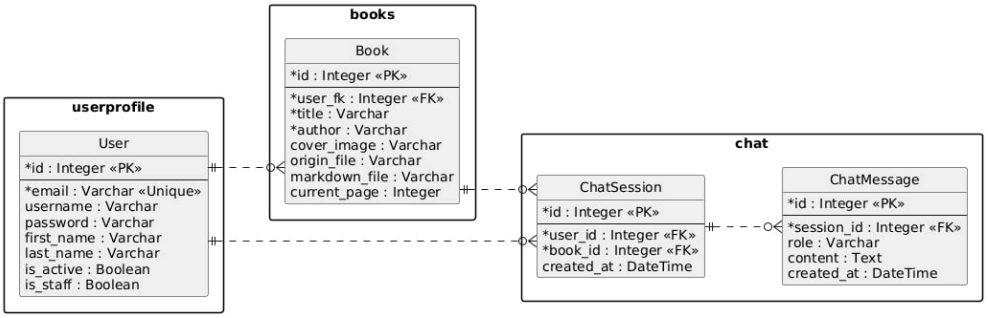

Рисунок 2.3.1 — Зв’язки між сутностями бази даних Neiread.

Важливим аспектом проектування є забезпечення цілісності даних через механізм каскадного видалення (CASCADE). При видаленні облікового запису користувача система автоматично видаляє всі пов’язані з ним книги та історії чатів. Аналогічно, видалення книги призводить до очищення відповідних сесій спілкування з асистентом та фізичного видалення файлів з диска. Продуктивність запитів забезпечується автоматичною індексацією первинних та зовнішніх ключів засобами Django, а керування змінами структури даних здійснюється через систему міграцій, файли яких розподілені за відповідними додатками проекту.

2.4 Побудова UML діаграми класів

Об’єктно-орієнтоване проектування є невід’ємною складовою створення комплексних програмних рішень, що дозволяє структурувати програмний код, забезпечити його гнучкість та придатність до масштабування. Центральним елементом статичного моделювання системи є діаграма класів (Class Diagram) уніфікованої мови моделювання (UML). Вона відображає архітектуру додатку у вигляді сукупності елементів з власними атрибутами та методами для демонстрації зв’язків між ними, таких як асоціація, агрегація, композиція та узагальнення (спадкування). Побудова діаграми класів для серверної частини (Backend) веб-сервісу Neiread дозволяє формалізувати внутрішню логіку роботи системи, чітко розмежувати зони відповідальності між компонентами та слугує

основою для програмної реалізації. Як зазначають розробники методології UML Ґрейді Буч, Джеймс Рамбо та Івар Якобсон, статичні структурні діаграми є «скелетом» системи, на який в подальшому нарощується функціональна «плоть» динамічної поведінки [16].

У контексті використання фреймворку Django та інструментарію Django REST Framework (DRF), архітектура класів проекту підпорядковується патерну MVT (Model-View-Template), трансформованому для потреб побудови REST API. Це зумовлює поділ класів на три логічні шари:

1. Моделі даних (Models) — класи, що відображають структуру бази даних (ORM) та містять методи бізнес-логіки.
2. Серіалізатори (Serializers) — класи, що відповідають за конвертацію складних типів даних у формат JSON та валідацію вхідних даних.
3. Представлення (Views) — класи контролерів, що обробляють запити, оркеструють взаємодію між моделями та серіалізаторами і формують HTTP-відповіді.

Структурно проект поділено на три функціональні модулі (apps): userprofile (керування користувачами), books (керування бібліотекою) та chat (інтелектуальна взаємодія). Нижче наведено детальний опис спроектованої ієрархії класів.

Рівень моделей є фундаментом системи, оскільки він визначає схему зберігання інформації. Усі класи цього рівня успадковуються від базового класу django.db.models.Model, за винятком моделі користувача.

Клас User (пакет apps.userprofile.models) виступає ключовою сутністю системи безпеки. Він реалізований через успадкування від AbstractUser, що дозволяє зберегти стандартні механізми Django (хешування паролів, права доступу), але адаптувати їх під вимоги проекту.

- Атрибути: Ключовою відмінністю є поле email, яке визначено як унікальний ідентифікатор (USERNAME_FIELD), що замінює стандартне ім'я користувача. Також присутні поля first_name, last_name, date_joined та службові прапорці is_active, is_staff.

- Залежності: Клас використовує кастомний менеджер UserManager, в якому перевизначено методи create_user та create_superuser для коректної обробки email як основного ідентифікатора.

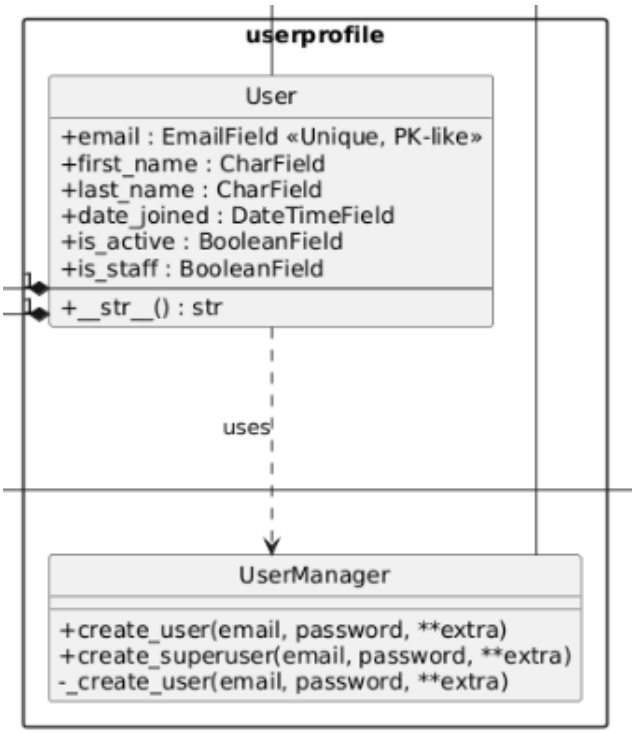

Рисунок 2.4.1 - Класи модуля userprofile.

Клас Book (пакет apps.books.models) інкапсулює дані про електронні книги.

- Атрибути: Окрім метаданих (title, author, cover_image), клас містить критично важливі для бізнес-логіки поля: origin_file та markdown_file, які зберігають шляхи до фізичних файлів у системі. Поля pages та current_page забезпечують збереження стану читання (UC-09).
- Методи: Реалізовано властивість (property) full_origin_file_path для зручного отримання абсолютного шляху до файлу. Особливої уваги заслуговує перевизначений метод delete(). Він реалізує важливе бізнес-правило: при видаленні запису про книгу з бази даних автоматично видаляються пов’язані файли з дискового сховища, що забезпечує чистоту файлової системи.
- Відношення: Асоціація з User типу «композиція» (через ForeignKey з on_delete=CASCADE), що означає, що книга не може існувати без власника.

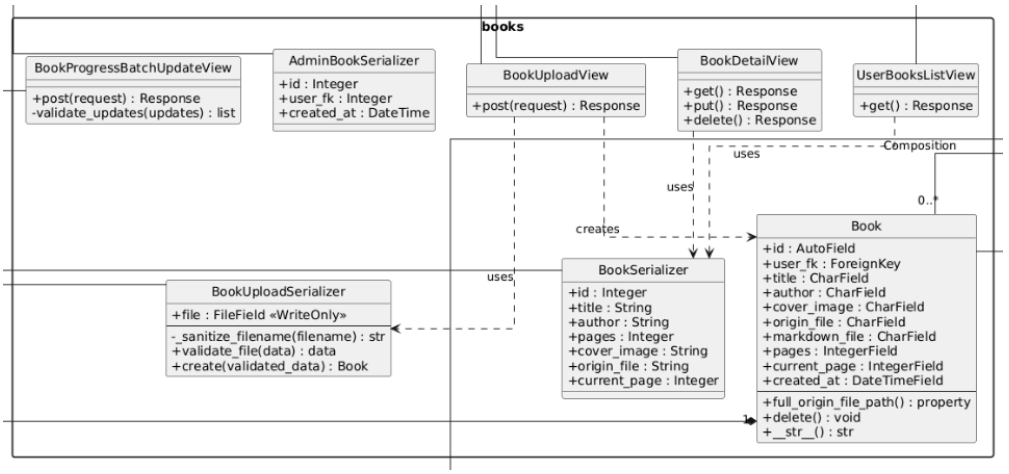

Рисунок 2.4.2 - Класи модуля Books.

Класи модуля chat — ChatSession та ChatMessage — реалізують структуру діалогів.

- ChatSession: Слугує контейнером для повідомлень. Вона має подвійну прив'язку: до User та до Book, що створює контекст для роботи AI-асистента. Метод get_history_for_gemini() виконує роль адаптера, перетворюючи внутрішнє представлення історії чату у формат, зрозумілий зовнішньому API (Google Gemini).
- ChatMessage: Містить власне контент переписки. Поле role використовує обмежений набір значень (константа ROLE_CHOICES), розрізняючи репліки користувача («user») та асистента («assistant»).

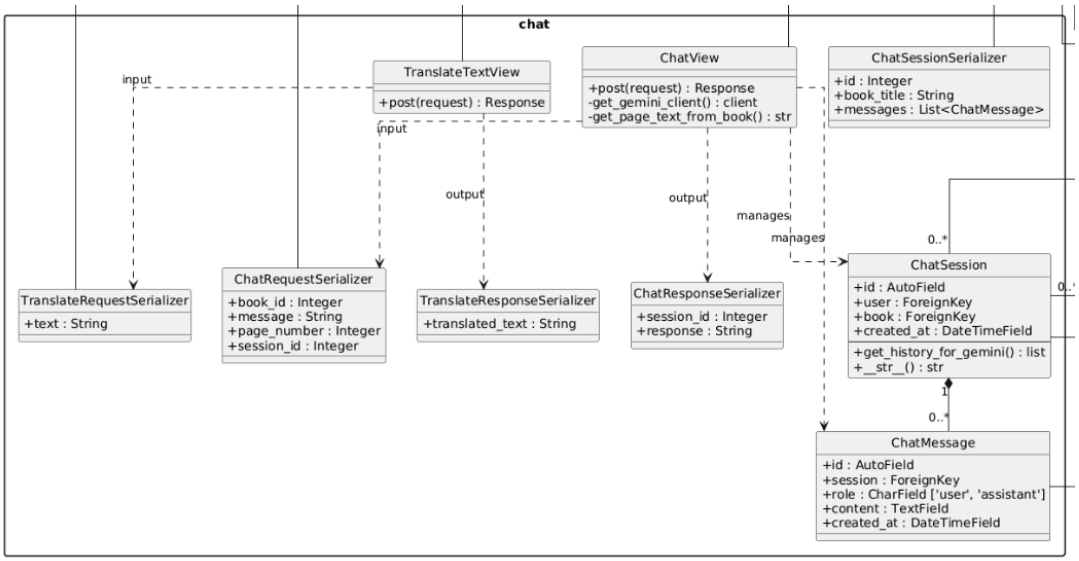

Рисунок 2.4.3 - Класи модуля chat

Рівень класів серіалізації та трансформації даних відповідає за перетворення даних (Data Transfer Objects). У проекті використано два типи серіалізаторів: ті, що базуються на моделях (ModelSerializer), та ті, що описують структуру окремих запитів (Serializer).

BookSerializer та AdminBookSerializer (пакет apps.books.serializers) забезпечують представлення даних книги для звичайного користувача та адміністратора відповідно. Вони визначають набір полів (fields) та права на запис (read_only_fields), гарантуючи, що клієнт не зможе змінити системні поля, такі як created_at або user_fk.

Спеціалізований клас BookUploadSerializer демонструє розширену логіку обробки вхідних даних. Він містить поле file (Write Only), яке не зберігається в БД напряму, а обробляється у методі create(). Цей метод реалізує складний алгоритм: санітизацію імені файлу (_sanitize_filename), збереження оригіналу та ініціацію процесу конвертації PDF у Markdown.

Група серіалізаторів чату (ChatRequestSerializer, TranslateRequestSerializer тощо) не прив’язана до моделей напряму, а діє як інтерфейс контракту API. Наприклад, ChatRequestSerializer валідує наявність обов'язкових параметрів book_id, message та опціонального page_number, забезпечуючи коректність даних перед передачею їх у контролер.

Класи представлень (Views) є точками входу для HTTP-запитів. Вони успадковуються від класів DRF та використовують інші компоненти системи.

BookUploadView (спадкоємець CreateAPIView) реалізує логіку завантаження книг. Він використовує BookUploadSerializer для валідації та створення об'єкта, а також перевіряє права доступу через клас дозволів IsAuthenticated.

ChatView (спадкоємець базового APIView) є прикладом "товстого" контролера, який інкапсулює складну логіку взаємодії. Його метод post() оркеструє процес:

1. Отримує дані через ChatRequestSerializer.
2. Витягує контекст книги (текст поточної сторінки) за допомогою допоміжної функції get_page_text_from_book.

3. Формує запит до зовнішнього сервісу через клієнт get_gemini_client.
4. Зберігає історію в ChatSession та ChatMessage.
5. Повертає відповідь через ChatResponseSerializer.

BookProgressBatchUpdateView оптимізує навантаження на базу даних, дозволяючи оновлювати прогрес читання пакетно, що є критичним для продуктивності SPA-додатку.

Аналіз побудованої діаграми класів демонструє дотримання принципів SOLID та GRASP.

- Інкапсуляція: Внутрішня структура даних (наприклад, формат зберігання файлів у Book) прихована від клієнта за серіалізаторами.
- Слабка зв'язаність (Low Coupling): Контролери не залежать від деталей реалізації моделей напряму, а взаємодіють через інтерфейси менеджерів та серіалізаторів.
- Висока згуртованість (High Cohesion): Логіка роботи з файлами зосереджена в моделі Book, логіка автентифікації — в User, а логіка чату — у відповідному модулі.

Між класами встановлено чіткі типи відносин:

1. Спадкування (Inheritance): Використовується для розширення функціоналу базових класів фреймворку (наприклад, User <|-- AbstractUser, BookUploadSerializer <|-- ModelSerializer).
2. Асоціація (Association): Зв’язки між моделями через зовнішні ключі (ForeignKey). Наприклад, User "1" --> "0..*" Book.
3. Залежність (Dependency): Відносини між View та Serializer, де зміна в серіалізаторі може вплинути на роботу контролера.

На рисунку 2.4 наведено повну UML-діаграму класів, що об’єднує всі рівні системи та демонструє їхню інтеграцію.

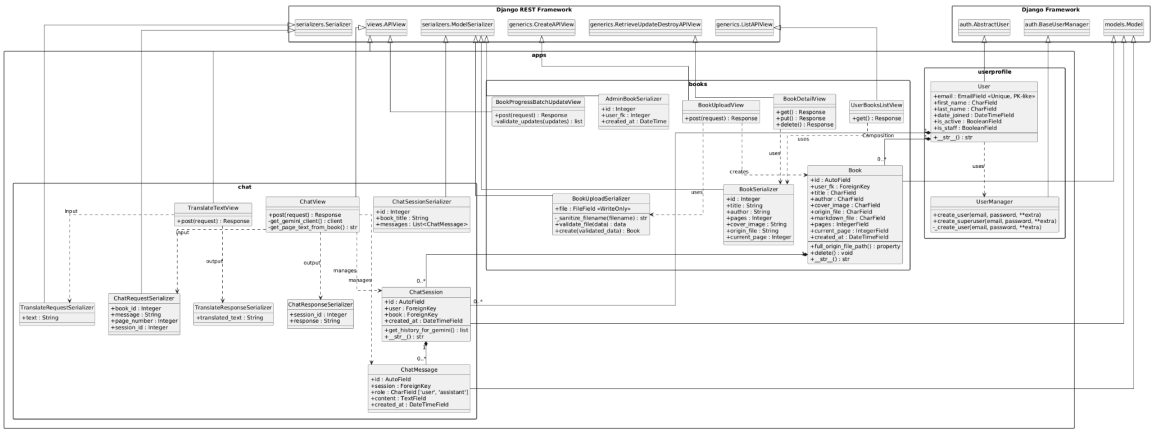

Рисунок 2.4.4 — Повна UML-діаграма класів програмної системи.

Підсумовуючи, розроблена структура класів забезпечує необхідну функціональність для реалізації сервісу читання та перекладу. Вона є модульною, розширюваною та відповідає сучасним стандартам розробки веб-додатків на Python. Виділення окремих класів для різних аспектів роботи з даними (збереження, валідація, презентація) дозволяє ефективно керувати складністю проекту та спрощує масштабування читалки: нові методи, формати книг або інтеграція з іншими AI-сервісами.

2.5 Вибір мови та середовища розробки

Обґрунтований вибір технологічного стеку та інструментальних засобів розробки є ще одним важливим кроком проєктування програмного забезпечення, що визначає не лише швидкість та зручність написання коду, але й подальшу продуктивність, масштабованість та надійність кінцевого продукту. Для реалізації веб-сервісу читання та перекладу книг Neiread обрано сучасний набір технологій, що поєднує високу продуктивність серверної частини, інтерактивність клієнтського інтерфейсу та надійність зберігання даних. Архітектурне рішення базується на розділенні логіки на бекенд (Python/Django) та фронтенд (TypeScript/NuxtJS), які взаємодіють через REST API, а як сховище даних

використовується СУБД PostgreSQL. Процес розробки здійснюється в інтелектуальному середовищі Cursor. Нижче наведено детальний аналіз кожного компонента обраного стеку.

У якості основної мови програмування серверної частини обрано Python. Це високорівнева інтерпретована мова загального призначення, яка характеризується суворою динамічною типізацією та автоматичним управлінням пам’яттю. Вибір Python обумовлений його домінуючим положенням у сферах веб-розробки та штучного інтелекту. Оскільки проектований сервіс передбачає інтеграцію з системами машинного перекладу, використання Python дозволяє створити єдине екосистемне середовище без необхідності залучення додаткових мов для написання скриптів обробки тексту.

Важливою перевагою Python є його лаконічний синтаксис, що сприяє написанню читабельного та легко підтримуваного коду. Стандартна бібліотека мови містить широкий набір модулів для роботи з мережевими протоколами, файловою системою та різними форматами даних, що спрощує реалізацію базового функціоналу. Крім того, наявність розвиненої спільноти та пакетного менеджера pip дозволяє користуватися безліччю сторонніх бібліотек, що прискорює розробку. У контексті веб-сервісів Python демонструє високу стабільність та достатню продуктивність, особливо при використанні асинхронних можливостей (asyncio), введених у нових версіях мови. Це дозволяє ефективно обробляти велику кількість одночасних з’єднань, що є актуальним для сервісу з потенційно високим навантаженням [17].

Рисунок 2.5.1 - Логотип мови Python [17].

Комунікація клієнта із сервером спирається на архітектуру REST (Representational State Transfer). Створення Python REST API вимагає побудови стандартизованих точок входу (endpoints) для обробки HTTP-викликів і повернення JSON-об'єктів. Це мінімізує залежність модулів (loose coupling), дозволяючи автономно вдосконалювати фронтенд і бекенд.

Для реалізації серверної логіки обрано веб-фреймворк Django. Це потужний інструмент високого рівня, побудований за принципом «batteries included» (все включено), що означає наявність «з коробки» більшості необхідних компонентів: системи автентифікації, адміністративної панелі, ORM (Object-Relational Mapping), механізмів міграції бази даних та засобів безпеки. Архітектура Django базується на паттерні MVT (Model-View-Template), який забезпечує чітке розмежування логіки роботи з даними, їх обробки та представлення.

Вирішальним чинником вибору Django стала його орієнтація на захищеність. Платформа автоматично нейтралізує популярні вектори атак, зокрема SQL-ін'єкції, міжсайтовий скриптинг (XSS) і CSRF-маніпуляції. Враховуючи роботу системи з приватними каталогами клієнтів, дана функціональність є життєво необхідною.

Оскільки система проектується як SPA (Single Page Application), стандартний шаблонізатор Django використовується мінімально. Натомість, основну роль відіграє бібліотека Django REST Framework (DRF). DRF надає потужний інструментарій для побудови Web API, включаючи серіалізатори для конвертації складних типів даних (моделей Django) у JSON, класи представлень (ViewSets) для обробки CRUD-операцій, а також гнучку систему дозволів (Permissions) та автентифікації (включаючи підтримку JWT). Використання DRF дозволяє стандартизувати структуру API, забезпечити його самодокументованість та спростити інтеграцію з фронтендом [18].

Рисунок 2.5.2 - Логотип Django [18].

Для зберігання інформації вирішено застосувати об'єктно-реляційну базу даних PostgreSQL. Ця система з відкритим кодом вирізняється стабільністю й багатим функціоналом, гарантуючи повну підтримку SQL-стандартів і відповідність критеріям ACID (Atomicity, Consistency, Isolation, Durability). Використання PostgreSQL у сервісі Neiread продиктоване її ефективністю під час обробки масивних обсягів інформації та нетривіальних запитів.

На противагу базовим аналогам на зразок SQLite, PostgreSQL послуговується технологією MVCC (Multi-Version Concurrency Control). Завдяки цьому паралельні операції читання й запису виконуються без блокування таблиць, що необхідно для багатокористувацьких середовищ. Також система пропонує широкий спектр типів даних, зокрема повноцінну інтеграцію формату JSONB. Це дозволяє зберігати неструктуровані дані (наприклад, метадані книг або історію складних діалогів з чат-ботом) безпосередньо в реляційній базі, поєднуючи переваги SQL та NoSQL підходів.

Інтеграція PostgreSQL з Django є «рідною» та найбільш оптимізованою серед усіх підтримуваних баз даних. Django ORM дозволяє використовувати специфічні функції PostgreSQL, такі як повнотекстовий пошук, індекси GIN/GiST та масиви, що значно розширює можливості додатку без необхідності написання "сирих" SQL-запитів. Надійність транзакцій та механізми відновлення після збоїв (Write-Ahead Logging) гарантують збереження користувацьких даних [19].

Рисунок 2.5.3 - Логотип PostgreSQL [19].

Фронтенд застосунку розробляється з використанням мови TypeScript. Це мова програмування, розроблена компанією Microsoft, яка є надбудовою (superset) над JavaScript. Основною особливістю TypeScript є статична типізація, яка дозволяє виявляти помилки ще на етапі компіляції коду, а не під час виконання програми. Для складних веб-додатків, де логіка взаємодії з користувачем є розгалуженою, а структури даних — складними, використання TypeScript є необхідною умовою забезпечення якості коду.

TypeScript дозволяє описувати інтерфейси об'єктів, типи функцій та параметрів, що робить код самодокументованим. Це значно спрощує командну розробку та підтримку проекту в майбутньому, оскільки розробник завжди бачить, які дані очікує отримати компонент або функція. Крім того, TypeScript підтримує сучасні можливості стандарту ECMAScript, такі як модулі, класи, стрілочні функції, async/await, та транспілює їх у JavaScript, сумісний з будь-якими браузерами.

Інтеграція TypeScript з сучасними IDE забезпечує потужний автокомпліт та рефакторинг коду, що підвищує продуктивність розробника. У проекті Neiread

TypeScript використовується для опису компонентів Vue, стору стану (Pinia) та інтерфейсів відповідей API, що гарантує узгодженість даних на всіх рівнях клієнтського додатку [20].

Рисунок 2.5.4 - Логотип TypeScript [20].

Фундаментом клієнтської частини став NuxtJS — надбудова над Vue.js. Цей інструмент усуває типові вади звичайних SPA-рішень: проблеми з пошуковим просуванням (SEO) та затримки під час первинного завантаження сторінки. Завдяки механізму серверного рендерингу (SSR - Server Side Rendering) або статичної генерації (SSG), Nuxt формує HTML-сторінку на сервері і відправляє її браузеру вже готовою до відображення. Після цього відбувається процес «гідратації», і сторінка стає повноцінним інтерактивним додатком. Для сервісу книг це критично важливо, оскільки контент повинен індексуватися пошуковими системами.

NuxtJS пропонує сувору структуру проекту та конвенції (Convention over Configuration), що прискорює розробку. Наприклад, файлова маршрутизація автоматично створює роути на основі структури папок у директорії pages, а система автоімпорту дозволяє використовувати компоненти та утиліти без явного імпорту в кожному файлі. Фреймворк також включає в себе потужний серверний двигун Nitro, який забезпечує високу продуктивність та можливість розгортання в безсерверному середовищі (serverless).

Важливою особливістю є модульна архітектура Nuxt. Для проекту використовуються такі модулі як @pinia/nuxt для управління станом, @nuxtjs/i18n для інтернаціоналізації інтерфейсу та @nuxt/ui або TailwindCSS для стилізації. Це

дозволяє зосередитися на бізнес-логіці, використовуючи перевірені рішення для типових задач [21].

Рисунок 2.5.5 - Логотип Nuxt [21].

Розробка програмного забезпечення ведеться в інтегрованому середовищі (IDE) Cursor. Йдеться про передовий редактор, заснований на екосистемі VS Code, однак із впровадженням ШІ-алгоритмів безпосередньо в ядро системи. Якщо стандартні розширення автозаповнення, на кшталт GitHub Copilot, мають обмеження, то Cursor оперує повним контекстом задачі, аналізуючи ієрархію файлів, залежності та супровідну документацію.

Ключовою функцією Cursor є вбудований AI-чат, який дозволяє генерувати код, пояснювати логіку роботи складних алгоритмів, знаходити помилки та пропонувати шляхи рефакторингу, враховуючи специфіку поточної кодової бази. Функція "Command K" дозволяє редагувати існуючий код за допомогою інструкцій природною мовою безпосередньо в редакторі. Крім того, Cursor підтримує індексацію локальних файлів проекту (Codebase Indexing), що дозволяє моделі надавати релевантні відповіді, базуючись на визначених у проекті класах та функціях.

Використання Cursor значно підвищує ефективність розробки, особливо при написанні рутинного коду (boilerplate), створенні тестів та роботі з новими бібліотеками. Оскільки Cursor є форком VS Code, він підтримує всі розширення з маркетплейсу Microsoft, що дозволяє використовувати звичні інструменти для роботи з Python, Vue, Git та Docker, доповнюючи їх можливостями штучного інтелекту [22].

Рисунок 2.5.6 - Логотип IDE Cursor[22].

Для реалізації інтелектуальних функцій системи, зокрема контекстного перекладу та генерації пояснень, обрано новітню велику мовну модель (LLM) Gemini 2.5 Flash від Google DeepMind. Ключовим фактором, що визначив цей вибір, стала архітектурна оптимізація моделі, спрямована на забезпечення мінімальної затримки (low latency) при генерації відповідей, що є критично важливим для збереження динаміки читання та позитивного користувацького досвіду (UX) в режимі реального часу. На відміну від більш потужних, але ресурсомістких версій (Pro або Ultra), редакція Flash пропонує оптимальний баланс між швидкістю інференсу та вартістю використання API, водночас підтримуючи розширене контекстне вікно, достатнє для аналізу цілих розділів книги без втрати семантичних зв'язків. Згідно з технічними звітами розробників, алгоритми дистиляції знань, застосовані у версії Flash, дозволяють досягати точності перекладу, співмірної з флагманськими моделями, при значно менших обчислювальних витратах, що робить її економічно доцільним рішенням для масштабованих веб-сервісів [23].

Рисунок 2.5.7 - Логотип Gemini [23].

Підсумовуючи, обраний стек технологій є збалансованим та відповідає сучасним індустріальним стандартам. Поєднання Python та Django забезпечує надійну серверну архітектуру, а інтеграція моделі Gemini 2.5 Flash наділяє систему

передовими інтелектуальними можливостями для роботи з текстом. PostgreSQL гарантує цілісність та швидкість обробки даних, що є критичним для зберігання контенту та історії діалогів. Використання TypeScript та NuxtJS дозволяє реалізувати високопродуктивний та адаптивний клієнтський інтерфейс, тоді як застосування інтелектуального середовища Cursor суттєво оптимізує процес розробки. Така комплексна комбінація інструментів створює міцний технологічний фундамент для успішної реалізації проекту Neiread та його подальшого масштабування.

2.6 Реалізація основних класів системи

Програмна реалізація інформаційної системи є кульмінаційним етапом інженерного циклу, під час якого теоретичні архітектурні моделі та спроектовані діаграми класів трансформуються у функціонуючий програмний код. У межах розробки серверної частини сервісу Neiread використовувався фреймворк Django, який нав’язує певну структурну парадигму, проте залишає достатньо простору для реалізації складної кастомної бізнес-логіки. Реалізація класів здійснювалася з дотриманням принципів об’єктно-орієнтованого програмування, таких як інкапсуляція (приховування внутрішньої реалізації методів), спадкування (розширення базових класів фреймворку) та поліморфізм. Нижче наведено поглиблений опис програмної реалізації ключових компонентів системи, що забезпечують автентифікацію, управління контентом та інтелектуальну взаємодію.

Основою безпеки та персоналізації будь-якого веб-сервісу є надійна модель користувача. У стандартній конфігурації Django автентифікація базується на полі username, що є застарілим підходом для сучасних UX-стандартів, де користувачі звикли використовувати електронну пошту як єдиний ідентифікатор. Для адаптації системи до цих вимог було розроблено клас UserManager. Цей клас

успадковує функціональність від BaseUserManager і містить низькорівневу логіку створення облікових записів.

Критично важливою частиною реалізації є метод _create_user. На відміну від стандартного методу, він виконує обов’язкову нормалізацію адреси електронної пошти через виклик self.normalize_email(email). Ця процедура приводить доменну частину email-адреси до нижнього регістру, що дозволяє уникнути дублювання облікових записів (наприклад, User@Example.com та user@example.com трактуються як одна адреса). Також метод інкапсулює логіку хешування пароля: пряме збереження пароля у відкритому вигляді є неприпустимим з точки зору безпеки, тому використовується метод set_password(), який застосовує алгоритм PBKDF2 за замовчуванням. Реалізація методів create_user та create_superuser демонструє патерн «Фабричний метод», дозволяючи створювати об’єкти користувачів з різними рівнями доступу (звичайний користувач або адміністратор) через єдиний інтерфейс, автоматично встановлюючи прапорці is_staff та is_superuser.

Безпосередньо модель User реалізована шляхом розширення абстрактного класу AbstractUser. Це стратегічне рішення дозволило зберегти повну сумісність з підсистемою дозволів (permissions) та групами Django, але повністю змінити схему ідентифікації. Поле username було явно виключено зі схеми бази даних (username = None), а поле email отримало атрибут unique=True, ставши первинним ключем для логіки входу.

Особливу увагу приділено полям first_name та last_name. Вони визначені як необов'язкові (blank=True), що дозволяє реалізувати стратегію «швидкої реєстрації», коли від користувача вимагається мінімум даних на старті, а профіль заповнюється пізніше. Поля is_active та date_joined відіграють важливу роль в адмініструванні: перше дозволяє блокувати доступ до системи без видалення даних (soft delete), а друге використовується для аналітики активності аудиторії.

Клас Book є центральним компонентом бізнес-логіки, оскільки він зв’язує користувача з контентом. При проектуванні цього класу було прийнято рішення розділити метадані (назва, автор, прогрес) та бінарні дані (файли книг). Зберігання

файлів безпосередньо в базі даних (BLOB) часто призводить до деградації продуктивності при резервному копіюванні та масштабуванні. Тому клас містить поля origin_file та markdown_file, які є текстовими посиланнями на розташування файлів у файловій системі сервера.

Важливим аспектом реалізації є використання декоратора @property для методу full_origin_file_path. Це забезпечує інкапсуляцію логіки формування повного шляху до файлу, приховуючи від зовнішніх компонентів деталі конфігурації MEDIA_ROOT. Також критично важливою є реалізація методу delete. У веб-додатках, що активно працюють з файлами, існує ризик накопичення «сміття» — файлів, записи про які вже видалені з БД. Перевизначений метод delete спочатку перевіряє фізичну наявність файлів за вказаними шляхами і видаляє їх засобами модуля os, і лише після цього викликає метод суперкласу для видалення запису з таблиці. Це гарантує цілісність даних на рівні файлової системи.

Логіка обробки завантаження реалізована в BookUploadSerializer. Цей клас виконує роль не просто валідатора, а повноцінного сервісного шару. Поле file позначено як write_only=True, що означає, що воно приймається API, але не повертається у відповіді. Метод _sanitize_filename реалізує захист від атак типу Path Traversal, видаляючи небезпечні символи з назви файлу. Основна магія відбувається в методі create: тут відбувається збереження PDF, вилучення метаданих та конвертація у Markdown. Вибір формату Markdown не є випадковим: це легкий текстовий формат, який ідеально підходить для передачі контексту в LLM (Large Language Models), оскільки він зберігає структуру заголовків та абзаців, але не містить складного бінарного коду, як PDF.

Підсистема чату спроектована для забезпечення контекстно-залежної взаємодії. Клас ChatSession вирішує задачу групування повідомлень. Він має зовнішні ключі на User та Book, що дозволяє системі чітко розрізняти контексти: обговорення книги "А" не повинно перетинатися з обговоренням книги "Б". Метод get_history_for_gemini реалізує патерн «Адаптер». Справа в тому, що внутрішня структура даних (моделі Django) відрізняється від формату JSON, який очікує API

Google Gemini (role: "user", parts: ["text"]). Цей метод виконує трансформацію даних "на льоту", звільняючи контролер від необхідності займатися форматуванням даних.

Клас ChatMessage є атомарною одиницею діалогу. Поле role використовує жорстко заданий набір варіантів (choices), що запобігає запису некоректних даних. Використання TextField для контенту дозволяє зберігати повідомлення довільної довжини, що важливо для розгорнутих відповідей AI.

Клас ChatView (контролер) об’єднує всі компоненти системи. Це точка входу для запитів на генерацію відповіді. Його метод post реалізує складний сценарій RAG (Retrieval-Augmented Generation).

1. Спочатку відбувається пошук поточної сесії або створення нової.
2. Далі виконується читання контексту: з Markdown-файлу книги витягується текст, що відповідає поточній сторінці користувача. Це критично важливо, адже без цього AI не знатиме, про що запитує користувач.
3. Формується "Системний промпт" (System Prompt), який інструктує модель: "Ти — помічник для читання. Ось текст книги: [Текст]. Відповідай на запитання користувача: [Запитання]".
4. Здійснюється виклик зовнішнього API (Gemini).
5. Забезпечується транзакційність: і запит користувача, і відповідь AI зберігаються в базі даних.

Використання get_object_or_404 забезпечує коректну обробку помилок (повернення 404, якщо книга не знайдена), а блок try-except навколо виклику AI захищає сервіс від падіння при проблемах зі з'єднанням або лімітами API.

Реалізація наведених класів демонструє комплексний підхід до побудови бекенду: від низькорівневої роботи з файловою системою та потоками даних до високорівневої оркестрації запитів до зовнішніх інтелектуальних систем. Використання патернів проектування та вбудованих можливостей Django дозволило створити код, який є не лише функціональним, але й придатним до подальшого масштабування та супроводу.

2.7 Розробка інтерфейсу користувача

Розробка клієнтської частини (Frontend) веб-сервісу є критично важливим етапом, оскільки саме інтерфейс забезпечує взаємодію користувача з функціоналом системи. Для реалізації клієнтського додатку було обрано фреймворк Nuxt.js 3, який базується на бібліотеці Vue.js. Цей вибір обумовлений необхідністю створення високопродуктивного односторінкового додатку (SPA) з можливостями серверного рендерингу (SSR), що покращує SEO-оптимізацію та прискорює час першого відображення контенту (First Contentful Paint). Процес розробки інтерфейсу базувався на компонентному підході, що дозволяє інкапсулювати логіку та стилі окремих елементів, забезпечуючи їх повторне використання та спрощуючи підтримку кодової бази. Для управління станом додатку використано бібліотеку Pinia, а для стилізації інтерфейсу — утилітарний CSS-фреймворк TailwindCSS, інтегрований через модуль @nuxt/ui.

Проектування почалося зі створення базового макета (Layout), який визначає загальну структуру сторінок. У Nuxt.js лейаути дозволяють визначити спільні елементи інтерфейсу, такі як шапка сайту (Header), навігаційна панель та підвал (Footer), які залишаються незмінними при переході між маршрутами. Базовий лейаут default.vue містить слот <slot />, куди динамічно підставляється контент конкретної сторінки. Навігаційна панель реалізує адаптивну поведінку: на десктопних пристроях відображається повне меню, а на мобільних — бургер-меню. В лейауті також реалізовано логіку перевірки стану автентифікації користувача: якщо токен доступу відсутній або недійсний, навігаційні посилання на захищені розділи (бібліотека, профіль) приховуються, а замість них відображаються кнопки входу та реєстрації.

Лістинг 2.7.1 - Базовий макет.

```
<script setup lang="ts">
import { useAuthStore } from '~/stores/auth'
```

```
const authStore = useAuthStore()
const links = computed(() => {
  if (authStore.isAuthenticated) {
    return [
      { label: 'Бібліотека', to: '/books' },
      { label: 'Профіль', to: '/profile' }
    ]
  }
  return []
})

function handleLogout() {
  authStore.logout()
  navigateTo('/auth/login')
}
</script>

<template>
      <div class="min-h-screen flex flex-col bg-gray-50 dark:bg-gray-900">
        <header class="border-b border-gray-200 dark:border-gray-800 bg-white dark:bg-gray-900 sticky top-0 z-50">
          <div class="container mx-auto px-4 h-16 flex items-center justify-between">
            <NuxtLink to="/" class="text-2xl font-bold text-primary-500">
              Neiread
            </NuxtLink>

            <nav class="hidden md:flex items-center gap-6">
              <NuxtLink
                v-for="link in links"
                :key="link.to"
                :to="link.to"
                class="text-gray-600 hover:text-primary-500 transition-colors"
              >
                {{ link.label }}
              </NuxtLink>
            </nav>

            <div class="flex items-center gap-2">
              <template v-if="!authStore.isAuthenticated">
                <UButton to="/auth/login" variant="ghost">Вхід</UButton>
                <UButton to="/auth/register" color="primary">Реєстрація</UButton>
              </template>
              <template v-else>
                <UButton icon="i-heroicons-arrow-right-on-rectangle" color="gray" variant="ghost" @click="handleLogout" />
```

```
              </template>
            </div>
          </div>
        </header>

        <main class="flex-grow container mx-auto px-4 py-8">
          <slot />
        </main>

                        <footer    class="border-t    border-gray-200
dark:border-gray-800 py-6 text-center text-gray-500 text-sm">
          © 2025 Neiread Project. All rights reserved.
        </footer>
      </div>
    </template>
```

Наступним етапом стала розробка модуля авторизації, що включає сторінки входу (login.vue), реєстрації (register.vue) та відновлення пароля (forgot-password.vue). Для забезпечення зручності користувача (UX) використано бібліотеку валідації форм Vuelidate або вбудовані можливості FormKit. Сторінка реєстрації містить форму з полями електронної пошти, пароля та його підтвердження. При відправці форми здійснюється асинхронний запит до API Django. У разі успішної відповіді отриманий JWT-токен зберігається в localStorage або cookie (за допомогою композибл useCookie в Nuxt), а стан глобального сховища Pinia оновлюється, надаючи доступ до захищених маршрутів. Сторінка входу реалізована аналогічно, з додаванням обробки помилок (наприклад, «Невірний логін або пароль»), які відображаються через спливаючі повідомлення (Toasts). Сторінка скидання пароля реалізує двоетапний процес: введення email для отримання посилання та, власне, встановлення нового пароля за токеном.

Головна сторінка (index.vue) виконує роль презентаційної вітрини (Landing Page) для незареєстрованих користувачів та панелі керування (Dashboard) для авторизованих. Каталог книг користувача (pages/books/index.vue) відображає список завантаженої літератури у вигляді адаптивної сітки карток. Кожна картка містить обкладинку, назву, автора та прогрес читання у відсотках. Для оптимізації продуктивності при великій кількості книг реалізовано «ліниве завантаження» зображень та пагінацію. Ключовим елементом цієї сторінки є модальне вікно завантаження нової книги (UploadModal), яке використовує HTML5 Drag-and-Drop

API. При виборі файлу відбувається перевірка його формату (тільки PDF) та розміру на стороні клієнта перед відправкою на сервер [24].

Лістинг 2.7.2 - Компонент BookCard.

```
<script setup lang="ts">
import type { Book } from '~/types'

defineProps<{
  book: Book
}>()

const progressPercentage = computed(() => {
  if (!props.book.pages) return 0
            return     Math.round((props.book.current_page     /
props.book.pages) * 100)
})
</script>

<template>
         <UCard    class="hover:ring-2    hover:ring-primary-500
transition-all cursor-pointer group">
      <template #header>
              <div  class="aspect-[2/3]  relative  overflow-hidden
rounded-md bg-gray-100 mb-2">
          <img
            :src="book.cover_image || '/placeholder-book.png'"
            :alt="book.title"
                              class="object-cover   w-full   h-full
group-hover:scale-105 transition-transform duration-300"
          />
        </div>
      </template>

       <h3 class="font-bold text-gray-900 truncate">{{ book.title
}}</h3>
      <p class="text-sm text-gray-500 mb-4">{{ book.author }}</p>

           <div  class="w-full  bg-gray-200  rounded-full  h-2.5
dark:bg-gray-700">
         <div class="bg-primary-600 h-2.5 rounded-full" :style="{
width: `${progressPercentage}%` }"></div>
      </div>
          <div  class="text-xs  text-right  mt-1  text-gray-500">{{
progressPercentage }}% прочитано</div>

      <template #footer>
        <UButton :to="`/books/${book.id}`" block>Читати</UButton>
      </template>
    </UCard>
```

```
</template>
```

Найскладнішим елементом інтерфейсу є сторінка читання книги (pages/books/[id].vue). Для рендерингу PDF-файлів було використано бібліотеку vue-pdf-viewer. Цей компонент забезпечує відображення сторінок документа з високою точністю, підтримуючи масштабування та навігацію. Інтеграція компонента вимагала налаштування передачі шляху до файлу, який отримується з API бекенду. Оскільки браузери мають політики безпеки CORS, файл завантажується через проксі-метод або надається через підписаний URL. Важливою частиною функціоналу читалки є синхронізація прогресу. Компонент відстежує подію зміни сторінки та з використанням техніки debouncing (затримка відправки запиту) надсилає на сервер номер поточної сторінки, щоб зберегти прогрес у базі даних [25].

Лістинг 2.7.3 - Сторінка книги.

```
<script setup lang="ts">
import { VPdfViewer, useLicense } from '@vue-pdf-viewer/viewer';
import { ref, onMounted, onUnmounted } from 'vue';
import { useRoute } from 'vue-router';

const route = useRoute();
const bookId = route.params.id;
const book = ref(null);
const pdfSource = ref('');

// Отримання даних книги
const { data, error } = await useFetch(`/api/v1/books/${bookId}/`);
if (data.value) {
  book.value = data.value;
  pdfSource.value = data.value.origin_file; // URL до PDF файлу
}

// Обробка зміни сторінки для збереження прогресу
const handlePageChange = (pageNumber: number) => {
  updateReadingProgress(bookId, pageNumber);
};

// Debounce функція для API запитів
const updateReadingProgress = useDebounce((id, page) => {
```

```
  $fetch(`/api/v1/books/${id}/progress/`, {
    method: 'PATCH',
    body: { current_page: page }
  });
}, 1000);
</script>

<template>
  <div class="h-[calc(100vh-64px)] w-full flex">
    <div class="flex-grow relative">
       <ClientOnly>
          <div v-if="pdfSource" class="w-full h-full">
            <VPdfViewer
              :src="pdfSource"
              :initial-page="book?.current_page"
              @page-change="handlePageChange"
            />
          </div>
       </ClientOnly>

       <!-- Компонент тултіпа вставляється тут -->
       <TextSelectionTooltip :book-id="bookId" />
    </div>

    <!-- Сайдбар чату -->
    <ChatSidebar :book-id="bookId" />
  </div>
</template>
```

Для реалізації інтелектуальних функцій (переклад та чат з AI) було розроблено механізм взаємодії з виділеним текстом. Компонент TextSelectionTooltip відстежує подію mouseup на контейнері з текстом. Використовуючи API браузера window.getSelection(), скрипт отримує об’єкт виділення. Якщо виділення не порожнє, обчислюються координати прямокутника виділення (getRangeAt(0).getBoundingClientRect()), і над текстом відображається плаваюче меню з кнопками «Перекласти» та «Запитати AI». Натискання кнопки ініціює запит до бекенду, передаючи виділений фрагмент. Отриманий результат відображається у тому ж спливаючому вікні або відкриває сайдбар чату. Сайдбар (ChatSidebar) містить історію повідомлень поточної сесії та поле для введення нових запитів. Стан чату (повідомлення, статус завантаження) також керується через Pinia store, що дозволяє зберігати контекст діалогу при перемиканні між сторінками [26, 27].

Лістинг 2.7.4 - Компонент тултіп.

```
<script setup lang="ts">
import { ref, onMounted, onUnmounted } from 'vue'

const props = defineProps<{ bookId: number }>()
const isVisible = ref(false)
const position = ref({ top: 0, left: 0 })
const selectedText = ref('')
const translation = ref('')

const handleSelection = () => {
  const selection = window.getSelection()
  if (!selection || selection.isCollapsed) {
    isVisible.value = false
    return
  }

  const text = selection.toString().trim()
  if (text.length > 0) {
    const range = selection.getRangeAt(0)
    const rect = range.getBoundingClientRect()

      // Розрахунок позиції відносно viewport з урахуванням
скролу
    position.value = {
      top: rect.top + window.scrollY - 40, // 40px над текстом
      left: rect.left + window.scrollX + (rect.width / 2)
    }
    selectedText.value = text
    isVisible.value = true
    translation.value = '' // Скидання попереднього перекладу
  }
}

const translateText = async () => {
  try {
    const response = await $fetch('/api/v1/translate/', {
      method: 'POST',
      body: { text: selectedText.value }
    })
    translation.value = response.translated_text
  } catch (e) {
    console.error('Translation failed', e)
  }
}

// Додаємо слухач подій на весь документ або контейнер рідера
onMounted(() => document.addEventListener('mouseup',
handleSelection))
```

```
    onUnmounted(() => document.removeEventListener('mouseup',
handleSelection))
    </script>

    <template>
      <div
        v-if="isVisible"
            class="fixed z-50 transform -translate-x-1/2 bg-white
dark:bg-gray-800 shadow-xl rounded-lg p-2 border border-gray-200
dark:border-gray-700"
                    :style="{ top: `${position.top}px`, left:
`${position.left}px` }"
      >
        <div v-if="!translation" class="flex gap-2">
                            <UButton size="xs" color="primary"
@click="translateText">Перекласти</UButton>
             <UButton size="xs" color="gray" @click="$emit('ask-ai',
selectedText)">Запитати AI</UButton>
        </div>
        <div v-else class="max-w-xs text-sm p-1">
                           <p class="font-medium text-gray-900
dark:text-gray-100">{{ translation }}</p>
        </div>
      </div>
    </template>
```

Сторінка особистого кабінету користувача (pages/profile.vue) надає інтерфейс для перегляду та редагування персональних даних. Тут реалізовано форми зміни імені та пароля, а також відображається статистика активності (кількість прочитаних книг, час у читанні). Взаємодія з сервером відбувається через захищені PUT/PATCH запити.

У підсумку, використання Nuxt.js у поєднанні з сучасними бібліотеками дозволило створити реактивний, швидкий та зручний інтерфейс користувача. Компонентна архітектура забезпечила модульність коду, а використання TypeScript гарантувало типізацію даних при обміні між клієнтом та сервером, мінімізуючи кількість помилок під час розробки [28].

У другій частині дослідження виконано системну розробку й написання коду веб-платформи, призначеної для перегляду та перекладу літератури. Обґрунтовано вибір гнучкої методології розробки, що дозволило ефективно організувати процес створення програмного забезпечення в умовах ітеративного уточнення вимог. Спроектовано архітектуру системи, яка базується на чіткому

розмежуванні серверної частини (Django REST Framework) та клієнтського інтерфейсу (NuxtJS), що забезпечує масштабованість та гнучкість рішення. Розроблено та реалізовано реляційну модель бази даних PostgreSQL, яка гарантує цілісність зберігання інформації про користувачів, бібліотечний фонд та історію інтелектуальних діалогів.

На основі побудованих UML-діаграм класів створено програмний код серверних модулів, реалізовано механізми автентифікації, безпечної обробки файлів та інтеграції з моделлю штучного інтелекту Gemini 2.5 Flash, що забезпечило функціональність контекстного перекладу та асистування при читанні. Паралельно розроблено сучасний інтерфейс користувача із застосуванням компонентного підходу, що дозволило реалізувати зручний перегляд PDF-документів та інтерактивну взаємодію з текстом. Результатом виконаних робіт є дієздатний програмний продукт, який відповідає поставленим технічним завданням та готовий до етапу тестування.

3 ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ПІДТРИМКА

Завершення етапу програмної реалізації веб-сервісу актуалізує питання забезпечення його якості, надійності та відповідності технічному завданню. Третій розділ кваліфікаційної роботи присвячено комплексному аналізу працездатності розробленої системи, що охоплює процеси тестування, розгортання в експлуатаційному середовищі та верифікації отриманих результатів. Метою цього етапу є виявлення та усунення потенційних дефектів, перевірка коректності виконання бізнес-логіки та підтвердження стабільності взаємодії клієнтської частини з серверною. У розділі послідовно розглядаються обрані стратегії та види тестування, формуються тестові сценарії для критичних модулів, таких як обробка файлів та інтеграція з нейромережею. Окрему увагу приділено визначенню системних вимог до апаратного та програмного забезпечення, а також опису процедури розгортання продукту з використанням сучасних інструментів контейнеризації, що є необхідною умовою для забезпечення життєздатності та доступності сервісу для кінцевих користувачів.

3.1 Тестування програмної системи

Гарантування якості ПЗ виступає обов'язковим елементом циклу створення продукту, маючи на меті фіксацію невідповідностей між запланованою роботою системи та її фактичним станом. Для веб-сервісу зі складною архітектурою, що включає клієнт-серверну взаємодію, асинхронну обробку файлів та інтеграцію з зовнішніми API штучного інтелекту, етап тестування набуває критичного значення для гарантування стабільності роботи та безпеки даних користувачів. Завдяки перевірці вдається знизити ризик неполадок у робочому режимі та переконатися, що створений функціонал узгоджується із затвердженими проектними вимогами. Згідно з класичним визначенням Гленфорда Майєрса, тестування розглядається не

як процес демонстрації коректності роботи програми, а як деструктивний процес виконання програми з метою виявлення в ній помилок, дефектів чи недоліків [29]. У даному підрозділі визначено стратегію контролю якості, обґрунтовано вибір рівнів та видів тестування, а також описано підходи до перевірки ключових функціональних модулів розробленого сервісу.

3.1.1 Види та план тестування

Вибір стратегії тестування для веб-сервісу Neiread базується на принципах необхідності забезпечення повноти покриття функціональних вимог при оптимальному розподілі часових ресурсів. Враховуючи клієнт-серверну архітектуру додатку, використання зовнішніх API для роботи зі штучним інтелектом та наявність складної логіки обробки файлів на бекенді, було прийнято рішення застосувати гібридну модель контролю якості. Вона поєднує автоматизоване модульне тестування (Unit Testing) для перевірки серверної логіки та ручне функціональне тестування (Manual Testing) для валідації інтерфейсу користувача та інтеграційних сценаріїв. Такий підхід дозволяє нівелювати недоліки кожного з методів окремо: автоматичні тести гарантують стабільність коду при внесенні змін, а ручне тестування забезпечує оцінку зручності використання (usability) та коректності візуального відображення даних, що є складним для автоматизації.

Першим етапом плану тестування визначено проведення модульного тестування серверної частини. Цей вид тестування передбачає перевірку найменших атомарних частин програмного коду — класів, методів та функцій — в ізоляції від інших компонентів системи. У контексті розробки на Django об'єктами модульного тестування виступають методи моделей (наприклад, коректність створення користувача менеджером UserManager або робота методу delete у моделі Book), логіка валідації вхідних даних у серіалізаторах та правильність

формування HTTP-відповідей у контролерах (Views). Використання модульних тестів дозволяє виявити логічні помилки на ранніх стадіях розробки, ще до інтеграції з клієнтською частиною. Як зазначає Володимир Хоріков у своїй праці про принципи модульного тестування, якісні unit-тести мають бути швидкими, ізольованими та стійкими до рефакторингу, що дозволяє використовувати їх як «живу документацію» коду та засіб запобігання регресії функціоналу [30]. Для реалізації цього етапу використовується вбудований у Django фреймворк тестування, що базується на бібліотеці unittest.

Другим етапом є ручне тестування послуговуючись методом Black-box testing. Цей підхід ігнорує внутрішню будову системи та зосереджується на перевірці відповідності поведінки програми зовнішнім специфікаціям та очікуванням користувача. Ручне тестування є критично важливим для перевірки клієнтської частини на NuxtJS, зокрема таких функцій, як рендеринг PDF-файлів, робота Drag-and-Drop інтерфейсу завантаження, коректність відображення спливаючих вікон (tooltips) при виділенні тексту та адаптивність верстки на різних пристроях. Для систематизації цього процесу розробляються тест-кейси (test cases) — детальні інструкції, що описують передумови, кроки виконання та очікуваний результат для кожного сценарію використання. Рон Паттон наголошує, що документування тестових випадків є обов'язковою умовою для забезпечення повторюваності тестів та об’єктивної оцінки готовності продукту до релізу [31].

3.1.2 Розробка модульних тестів для backend.

Основою валідації серверного коду виступає автоматизоване юніт-тестування, ціль якого — підтвердити справність окремих елементів архітектури в ізольованих умовах. У межах сервісу Neiread ці перевірки побудовано на базі стандартного пакету Python unittest, що працює в тандемі з утилітами бібліотек django.test і rest_framework.test. Останній надає

спеціалізований клас APITestCase, який розширює стандартні можливості тестування, дозволяючи емулювати HTTP-запити до REST API, перевіряти коди відповідей, структуру JSON-даних та стан бази даних після виконання транзакцій.

Ключовим принципом розроблених тестів є ізоляція. Тести виконуються на окремій тестовій базі даних, яка створюється автоматично на початку прогону та знищується після його завершення, що гарантує відсутність впливу на реальні дані. Для емуляції взаємодії із зовнішніми сервісами (Google Gemini API, файлова система) застосовано механізм «мокінг» (mocking) з бібліотеки unittest.mock, що дозволяє замінити реальні виклики зовнішніх функцій на фіктивні об'єкти з прогнозованою поведінкою.

Першочерговим завданням було тестування кастомної моделі користувача та менеджера UserManager, оскільки від них залежить коректність авторизації. Розроблені тести перевіряють сценарії створення звичайного користувача та суперкористувача, акцентуючи увагу на правильності нормалізації електронної пошти та обробці обов’язкових полів. Нижче наведено лістинг тестів для менеджера користувачів. Код тесту показано в лістингу 3.1.2.1.

Лістинг 3.1.2.1 - Модульні тести для UserManager

```
from django.test import TestCase
from django.contrib.auth import get_user_model

class UserManagerTests(TestCase):
    def test_create_user(self):
        User = get_user_model()
                                                    user      =
User.objects.create_user(email='Normal@User.com', password='foo')

        # Перевірка нормалізації email
        self.assertEqual(user.email, 'normal@user.com')
        # Перевірка встановлення базових прав
             self.assertFalse(user.is_active)    # Залежить від
налаштувань, за замовчуванням може бути True/False
        self.assertFalse(user.is_staff)
        self.assertFalse(user.is_superuser)

        # Перевірка відсутності username
        self.assertIsNone(user.username)
```

```
            # Перевірка обов'язковості email
            with self.assertRaises(ValueError):
                User.objects.create_user(email='', password='foo')

        def test_create_superuser(self):
            User = get_user_model()
                                                    admin_user      =
User.objects.create_superuser(email='super@user.com',
password='foo')

            self.assertEqual(admin_user.email, 'super@user.com')
            self.assertTrue(admin_user.is_active)
            self.assertTrue(admin_user.is_staff)
            self.assertTrue(admin_user.is_superuser)
```

Для модуля книг критично важливим є перевірка API завантаження файлів та логіки розмежування доступу (користувач повинен бачити лише власні книги). Тестовий клас BookApiTests використовує APIClient для відправки POST-запитів з тестовим PDF-файлом. Для уникнення реального збереження файлів на диск під час тестів використовується тимчасова файлова система або мокінг сховища. Код тесту показано в лістингу 3.1.2.2.

Лістинг 3.1.2.2 - Інтеграційні тести API книг

```
from django.urls import reverse
from rest_framework import status
from rest_framework.test import APITestCase
from django.contrib.auth import get_user_model
from django.core.files.uploadedfile import SimpleUploadedFile
from unittest.mock import patch

User = get_user_model()

class BookApiTests(APITestCase):
    def setUp(self):
          # Створення користувача та отримання токена (емуляція
авторизації)
                                                    self.user      =
User.objects.create_user(email='test@example.com',
password='password123')
            self.client.force_authenticate(user=self.user)
             self.url = reverse('book-upload') # Припускаємо, що URL
має назву 'book-upload'

       @patch('apps.books.serializers.pymupdf4llm.to_markdown')  #
Мокаємо конвертер
```

```
        def test_upload_book(self, mock_converter):
            # Налаштування мока
            mock_converter.return_value = "# Test Content"

            # Створення фіктивного PDF файлу
            pdf_content = b'%PDF-1.4 test content'
                    pdf_file = SimpleUploadedFile("test_book.pdf",
pdf_content, content_type="application/pdf")

            data = {'file': pdf_file}
                     response = self.client.post(self.url, data,
format='multipart')

            # Перевірки
                                self.assertEqual(response.status_code,
status.HTTP_201_CREATED)
            self.assertEqual(response.data['title'], 'test_book')

            # Перевірка прив'язки до користувача
            self.assertEqual(self.user.books.count(), 1)
                     self.assertEqual(self.user.books.first().title,
'test_book')

        def test_upload_invalid_file_format(self):
              txt_file = SimpleUploadedFile("test.txt", b"content",
content_type="text/plain")
                    response = self.client.post(self.url, {'file':
txt_file}, format='multipart')
                                self.assertEqual(response.status_code,
status.HTTP_400_BAD_REQUEST)
```

Найбільш складним аспектом тестування є модуль чату, оскільки він залежить від зовнішнього API Google Gemini. Щоб зробити тести детермінованими та незалежними від мережі, використано декоратор @patch для підміни клієнта genai. Тест перевіряє, чи коректно система формує контекст, чи зберігаються повідомлення в базі даних та чи повертається правильна відповідь. Код тесту показано в лістингу 3.1.2.3.

Лістинг 3.1.2.3 - Тестування Chat API з мокінгом

```
from rest_framework.test import APITestCase
from django.contrib.auth import get_user_model
from apps.books.models import Book
from apps.chat.models import ChatSession, ChatMessage
from unittest.mock import MagicMock, patch
```

```
    User = get_user_model()

    class ChatApiTests(APITestCase):
        def setUp(self):
                                                    self.user      =
User.objects.create_user(email='chat@user.com', password='pass')
                 self.book  =  Book.objects.create(user_fk=self.user,
title="Test Book", origin_file="path/to.pdf")
            self.client.force_authenticate(user=self.user)
            self.url = '/api/v1/chat/'

        @patch('apps.chat.views.ChatView._get_gemini_client')
        @patch('apps.chat.views.ChatView._get_page_text')
                def   test_send_message_to_ai(self,   mock_get_text,
mock_get_client):
            # 1. Налаштування моків
            mock_get_text.return_value = "Context from book page"

            # Створення мок-об'єкта для відповіді AI
            mock_response = MagicMock()
            mock_response.text = "This is an AI response"

            # Налаштування ланцюжка викликів клієнта
            mock_model = MagicMock()
            mock_chat = MagicMock()
            mock_chat.send_message.return_value = mock_response
            mock_model.start_chat.return_value = mock_chat
            mock_get_client.return_value = mock_model

            # 2. Виконання запиту
            payload = {
                'book_id': self.book.id,
                'message': 'Explain this text',
                'page_number': 1
            }
                   response  =  self.client.post(self.url,  payload,
format='json')

            # 3. Асерти (Перевірки)
            self.assertEqual(response.status_code, 200)
             self.assertEqual(response.data['response'], "This is an
AI response")

            # Перевірка створення сесії та повідомлень в БД
            self.assertEqual(ChatSession.objects.count(), 1)
             self.assertEqual(ChatMessage.objects.count(), 2) # User
query + AI response

                # Перевірка, що AI отримав правильний контекст у
промпті
            args, _ = mock_chat.send_message.call_args
            self.assertIn("Context from book page", args[0])
```

Виконання тестів здійснюється за допомогою стандартної команди Django python manage.py test. На основі результатів виконання написаних тестів видно, що всі реалізовані перевірки проходять успішно, що свідчить про коректність роботи серверної логіки, валідації даних та обробки помилок. Автоматизація цього процесу дозволяє гарантувати, що подальші зміни в коді (рефакторинг або додавання нових функцій) не порушать існуючий функціонал. На рисунку 3.1.2.1 наведено результат виконання тестів.

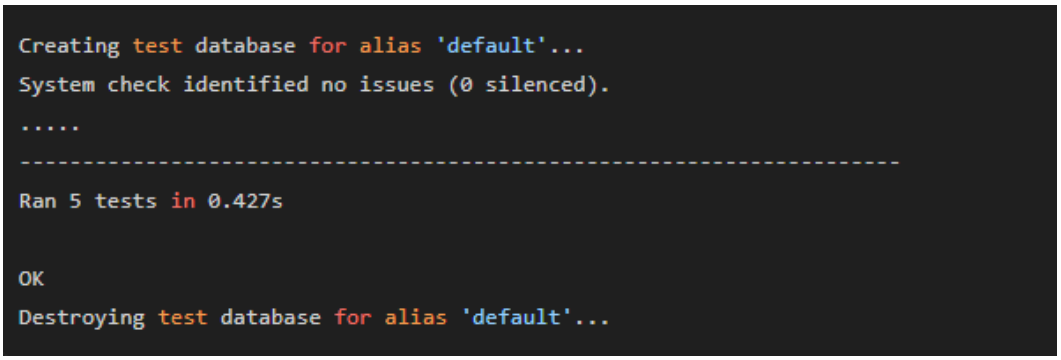

Рисунок 3.1.2.1 - Результат виконання тестів.

Як видно з рисунку, усі тести системи пройшли успішно.

3.1.3 Ручне тестування інтерфейсу користувача

Ручне тестування користувацького інтерфейсу є завершальним етапом верифікації системи перед її здачею в експлуатацію. Цей процес спрямований на перевірку відповідності візуальної складової та інтерактивної поведінки веб-додатку вимогам, сформульованим у технічному завданні. На відміну від автоматизованих тестів, які перевіряють програмний код, ручне тестування оцінює систему з точки зору людини, яка користуватиметься системою, дозволяючи виявити проблеми верстки, неочевидність навігації, некоректну обробку помилок на клієнті або затримки у відгуку інтерфейсу.

Тестування відбувалися в середовищі Google Chrome (від версії 120) із залученням панелі DevTools задля контролю мережевого трафіку й поточного стану Pinia. Щоб структурувати роботу, підготовлено комплекс тест-кейсів, спрямованих на перевірку ключових етапів взаємодії клієнта з сервісом Neiread. Кожен тест-кейс містить передумови, чітку послідовність дій та очікуваний результат.

Першим критичним модулем є система доступу. Тестування спрямоване на перевірку коректності роботи форм реєстрації та входу, валідації вхідних даних на стороні клієнта (NuxtJS) та обробки токенів доступу (JWT). Важливим аспектом є перевірка реакції інтерфейсу на некоректні дані та перенаправлення користувача після успішної авторизації.

Таблиця 3.1.3.1 - Тест-кейс TC-UI-01: Реєстрація та авторизація користувача.

Параметр	**Опис**
ID	TC-UI-01
Назва	Перевірка повного циклу реєстрації та входу в систему
Передумови	Користувач не авторизований. Відкрито сторінку /auth/register.
Кроки виконання	1. Ввести валідний email (наприклад, testuser@test.com) та пароль. 2. Натиснути кнопку «Зареєструватися». 3. Перевірити перенаправлення на сторінку входу або головну сторінку. 4. Виконати вихід із системи (Logout). 5. Перейти на сторінку /auth/login. 6. Ввести зареєстровані дані та натиснути «Увійти».

Продовження таблиці 3.1.3.1

Параметр	**Опис**
Очікуваний результат	1. Форма проходить валідацію. 2. Після реєстрації отримано повідомлення про успіх. 3. Після входу користувач перенаправляється на Дашборд. 4. У Cookies або LocalStorage з'являється токен доступу. 5. У шапці сайту відображається меню профілю замість кнопок входу.
Фактичний результат	Успішно. Токен збережено, інтерфейс оновився миттєво.

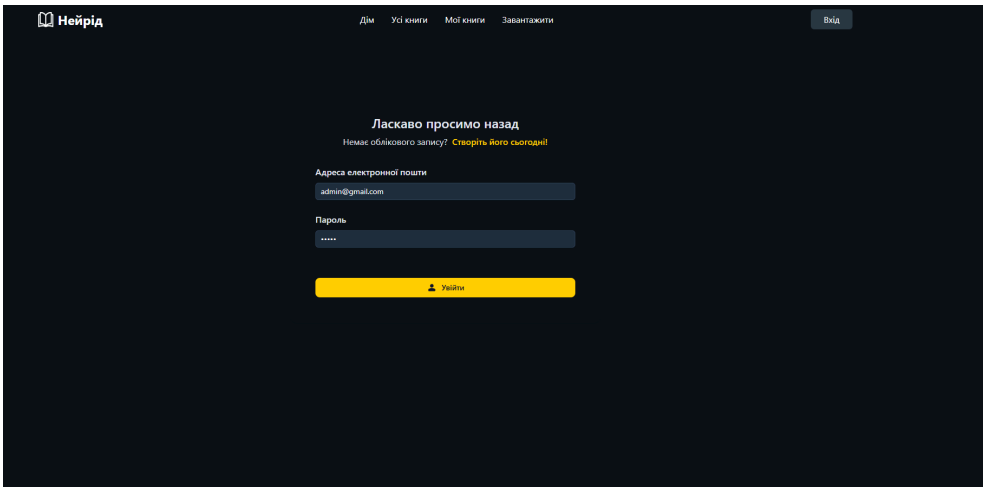

Рисунок 3.1.3.1 - Сторінка авторизації.

Далі перевіряємо взаємодію клієнта з файловою системою через API. Тестується робота компонента завантаження (Drag-and-Drop), візуалізація прогресу завантаження та коректність оновлення списку книг без перезавантаження сторінки (реактивність Vue). Також перевіряється валідація форматів файлів на клієнті.

Таблиця 3.1.3.2 - Тест-кейс TC-UI-02: Завантаження електронної книги.

Параметр	**Опис**
ID	TC-UI-02
Назва	Завантаження файлу та його відображення в бібліотеці.

Продовження таблиці 3.1.3.2

Параметр	**Опис**
Передумови	Користувач авторизований. Відкрито сторінку «Мої книги». Підготовлено тестовий файл book.pdf.
Кроки виконання	1. Натиснути кнопку «Додати книгу». 2. У модальному вікні перетягнути файл book.pdf в зону завантаження. 3. Дочекатися завершення індикатора завантаження. 4. Перевірити появу нової картки книги у списку.
Очікуваний результат	1. Файл успішно передається на сервер (код 201 Created). 2. Модальне вікно закривається автоматично. 3. У списку з'являється нова книга з коректною назвою. 4. При спробі завантажити .txt файл з'являється повідомлення про помилку формату..
Фактичний результат	Успішно. Книга додається до списку, обкладинка генерується (або ставиться заглушка).

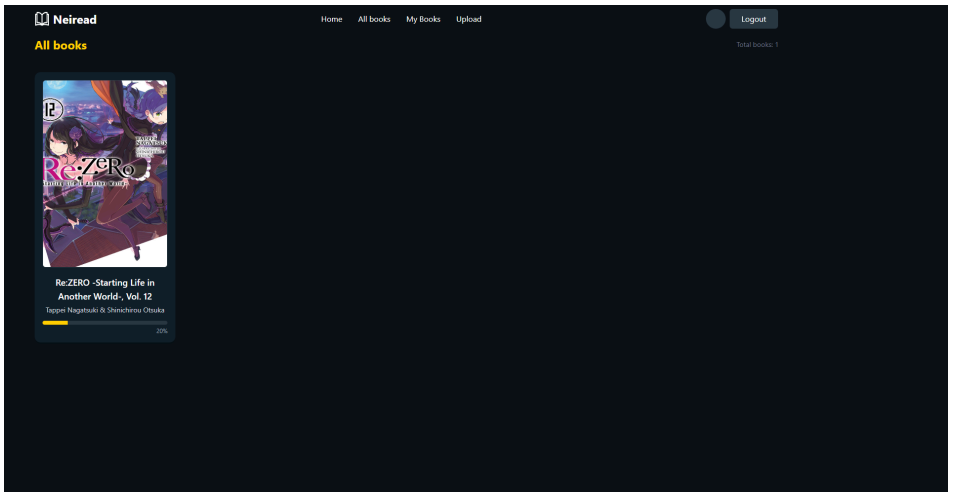

Рисунок 3.1.3.2 - Сторінка з книгами.

Основний функціонал системи — читання. Тестується інтеграція бібліотеки vue-pdf-viewer, коректність рендерингу сторінок, масштабування та навігація. Критичним моментом є перевірка збереження прогресу: система повинна запам'ятовувати останню відкриту сторінку і відновлювати її при повторному вході.

Таблиця 3.1.3.3 - Тест-кейс TC-UI-03: Робота рідера та збереження прогресу.

Параметр	**Опис**
ID	TC-UI-03
Назва	Читання книги та синхронізація позиції
Передумови	У бібліотеці є книга. Відкрито цю книгу.
Кроки виконання	1. Відкрити книгу, переконатися, що текст відображається. 2. Перегорнути книгу до 5-ї сторінки. 3. Зачекати 1-2 секунди (для спрацювання debounce-збереження). 4. Повернутися до списку книг. 5. Знову відкрити ту саму книгу.
Очікуваний результат	1. PDF-файл рендериться чітко, без артефактів. 2. Навігація працює плавно. 3. При повторному відкритті книга завантажується одразу на 5-й сторінці. 4. У консолі DevTools фіксується PATCH-запит на оновлення current_page.
Фактичний результат	Успішно. Прогрес відновлено коректно.

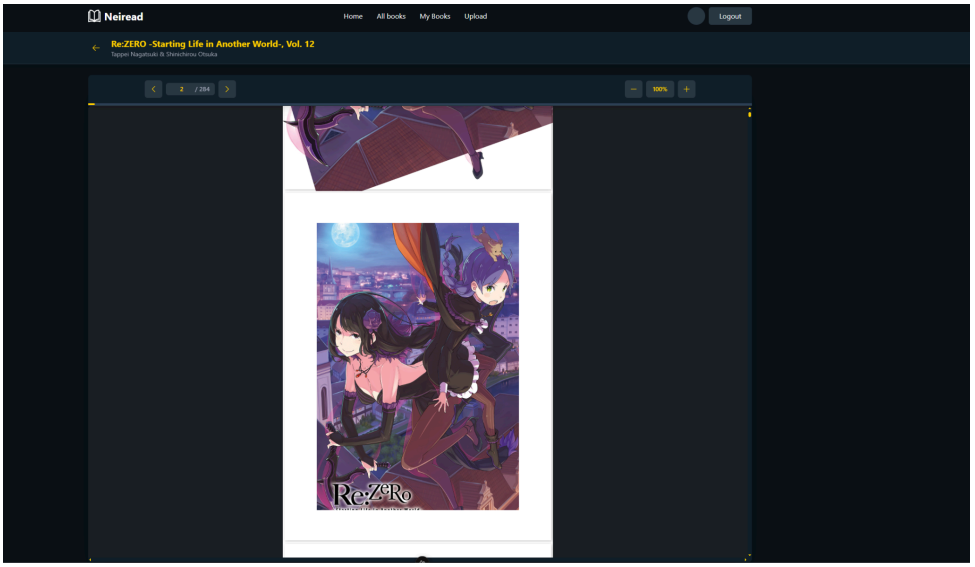

Рисунок 3.1.3.3 - Сторінка читання книги.

Наступний сценарій перевіряє взаємодію трьох компонентів: виділення тексту на клієнті, API бекенду та зовнішнього сервісу Gemini. Тестується поява

контекстного меню (тултіпа), коректність передачі виділеного тексту та відображення відповіді.

Таблиця 3.1.3.4 - Тест-кейс TC-UI-04: Контекстний переклад AI.

Параметр	**Опис**
ID	TC-UI-04
Назва	Використання AI-асистента для пояснення тексту
Передумови	Відкрито книгу. Працює підключення до Інтернету.
Кроки виконання	1. Виділити мишкою складне речення або абзац у тексті. 2. Перевірити появу спливаючого меню. 3. Натиснути кнопку Перекласти в AI». 4. Дочекатися відповіді від асистента.
Очікуваний результат	1. Тултіп з'являється точно над виділенням. 2. Протягом 2 секунд з'являється відповідь від AI, що відповідає контексту книги..
Фактичний результат	Успішно. Асистент надав правильний переклад відносно контексту.

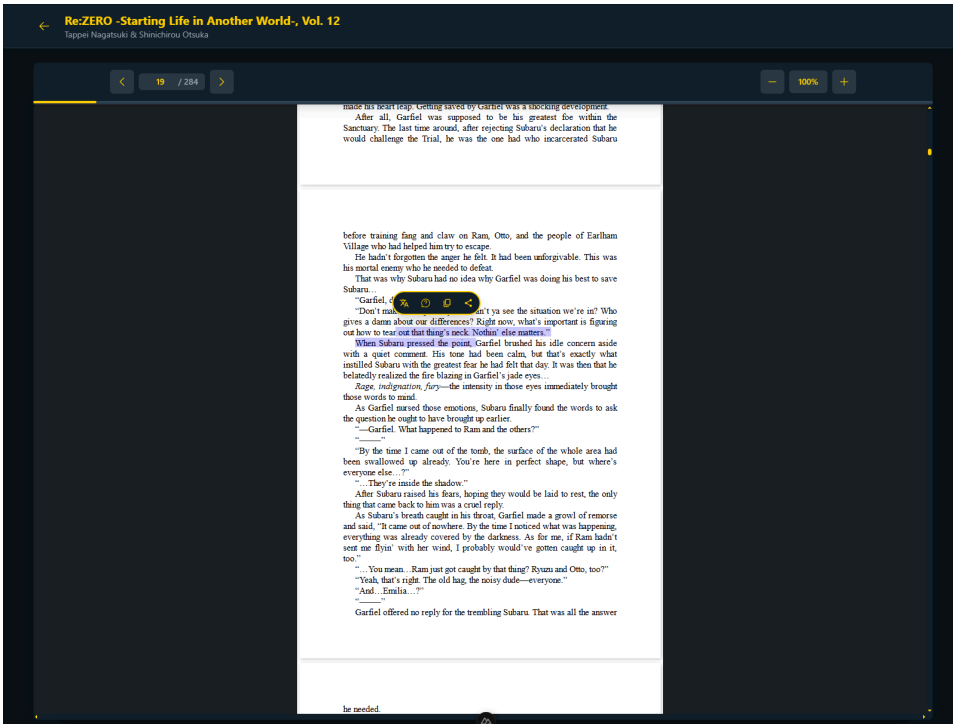

Рисунок 3.1.3.4 - Тутліп з діями.

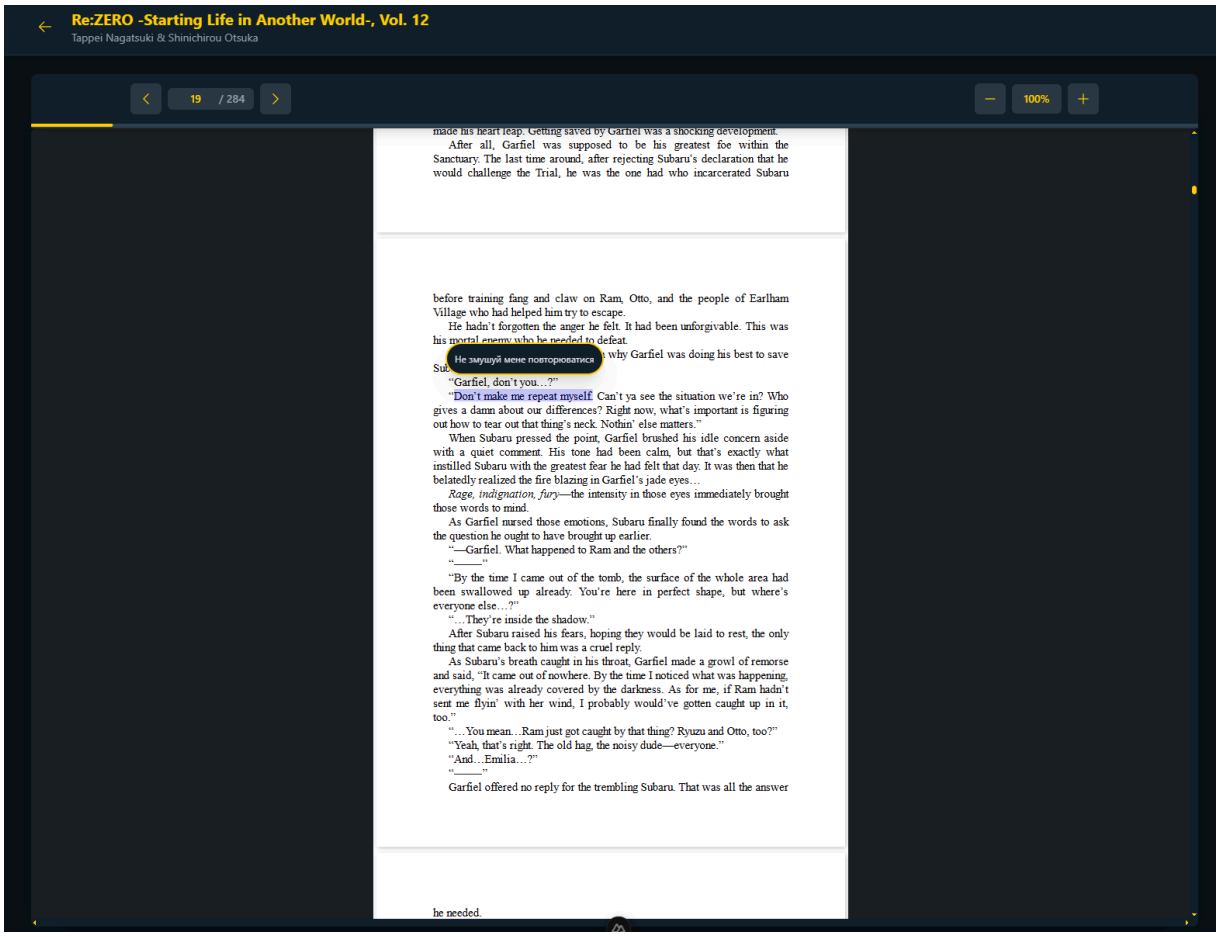

Рисунок 3.1.3.5 - Переклад речення.

3.2 Розгортання програмної системи та системні вимоги

Забезпечення стабільного функціонування веб-сервісу в промисловому середовищі вимагає чіткого визначення вимог до апаратного та програмного забезпечення, а також розробки надійної стратегії розгортання (deployment). Процес впровадження системи Neiread базується на принципах контейнеризації, що дозволяє ізолювати залежності компонентів, спростити масштабування та гарантувати ідентичність поведінки додатку в середовищах розробки, тестування та експлуатації. У цьому підрозділі описано системні вимоги до клієнтської та серверної сторін, а також деталізовано архітектуру розгортання з використанням технологій Docker та Nginx.

Вимоги до програмно-апаратного забезпечення поділяються на дві категорії: клієнтські (вимоги до пристроїв кінцевих користувачів) та серверні (вимоги до хостингу або хмарної інфраструктури).

Клієнтська частина, реалізована як односторінковий додаток (SPA) на NuxtJS, виконується безпосередньо у браузері користувача. Оскільки основне навантаження з рендерингу інтерфейсу та обробки PDF-файлів лягає на клієнтський пристрій, вимоги до нього є наступними:

- ОС: Windows 11, macOS 10.15+, Linux (сучасні дистрибутиви), Android 8.0+, iOS 14+.
- Браузер: Google Chrome 90+, Mozilla Firefox 88+, Safari 14+, Microsoft Edge 90+. Підтримка JavaScript та HTML5 є обов’язковою.
- Апаратне забезпечення: Процесор з тактовою частотою не менше 1.5 ГГц, обсяг ОЗУ від 2 ГБ (для комфортного перегляду великих PDF-файлів рекомендовано 4 ГБ).
- Мережа: Стабільне підключення до Інтернету не менше 1 Мбіт/с для завантаження книг та взаємодії з API перекладу.

Серверна частина, що включає веб-сервер, сервіс додатку (Django), сервіс фронтенду (Node.js/Nuxt) та базу даних (PostgreSQL), потребує більш значних обчислювальних ресурсів. Для забезпечення стабільної роботи системи при середньому навантаженні (до 100 активних користувачів) сформульовано наступні мінімальні вимоги до сервера:

- Процесор (CPU): Мінімум 2 віртуальних ядра (vCPU). Це необхідно для паралельної обробки HTTP-запитів та виконання ресурсомістких операцій конвертації файлів.
- Оперативна пам'ять (RAM): Мінімум 4 ГБ. Розподіл пам'яті: PostgreSQL (~1 ГБ), Django/Gunicorn (~1 ГБ), Nuxt/Nitro (~512 МБ), операційна система та кеш (~1.5 ГБ).
- Дисковий простір: SSD накопичувач об'ємом від 20 ГБ. Основний простір буде використовуватися для зберігання Docker-образів, бази даних та завантажених файлів книг (Media files).

- Програмне забезпечення: ОС Linux, встановлений Docker Engine версії 20.10+ та Docker Compose версії 2.0+.

Для організації процесу розгортання обрано архітектуру на основі мікросервісів у контейнерах Docker. Це дозволяє запакувати кожен компонент системи разом з його залежностями в окремий образ. Управління життєвим циклом контейнерів здійснюється за допомогою інструменту Docker Compose. Схема розгортання включає чотири основні сервіси:

- db: Контейнер з СУБД PostgreSQL. Використовує офіційний образ postgres:15-alpine. Дані бази зберігаються у іменованому томі (docker volume) для забезпечення персистентності.
- backend: Контейнер з Django-додатком. Збирається на основі образу python:3.11-slim. Як сервер додатків використовується Gunicorn, який запускає Django через WSGI-інтерфейс у кілька робочих процесів (workers).
- frontend: Контейнер з NuxtJS-додатком. Збирається на основі node:18-alpine. Додаток працює в режимі SSR (Server-Side Rendering), що вимагає запуску Node.js сервера (Nitro engine) для генерації HTML-сторінок.
- nginx: Веб-сервер, що виступає в ролі зворотного проксі (Reverse Proxy). Він є єдиною точкою входу для зовнішніх запитів (порти 80/443). Nginx обслуговує статичні файли (CSS, JS, зображення), перенаправляє запити на API до контейнера backend та запити на сторінки до контейнера frontend. Також на цьому рівні налаштовується SSL-шифрування (HTTPS).

Процедура розгортання системи на виробничому сервері складається з наступних кроків:

1. Підготовка оточення: На сервері встановлюється необхідне ПЗ (Docker, Git). Клонується репозиторій проекту з системи контролю версій.
2. Конфігурація змінних середовища: Створюється файл .env, в якому прописуються конфіденційні дані, що не зберігаються в репозиторії: паролі до бази даних, секретний ключ Django (SECRET_KEY), API-ключі зовнішніх сервісів (Google Gemini API), налаштування хоста (ALLOWED_HOSTS).

3. Складання (Build) образів: За допомогою команди docker-compose build створюються Docker-образи для проєктів Nuxt та Django. На цьому етапі для бекенду встановлюються Python-залежності з requirements.txt, а для фронтенду виконується збірка проекту (npm run build).
4. Запуск контейнерів: Команда docker-compose up -d запускає сервіси у фоновому режимі. Docker автоматично створює внутрішню віртуальну мережу, що дозволяє контейнерам взаємодіяти між собою за назвами сервісів.
5. Ініціалізація бази даних: У контейнері бекенду виконуються міграції бази даних для створення необхідних таблиць.
6. Збір статичних файлів: Виконується команда python manage.py collectstatic, яка збирає всі статичні файли Django в одну директорію, доступну для Nginx.

Такий підхід до розгортання забезпечує високу надійність системи. У разі виникнення помилки в одному з контейнерів, його можна перезапустити без зупинки інших компонентів. Крім того, використання Docker дозволяє легко переносити систему між різними хмарними провайдерами (AWS, DigitalOcean, Google Cloud) без необхідності переналаштування середовища.

3.3 Верифікація програмної системи

Верифікація ПЗ стане завершальним кроком інженерного циклу, метою якого є об’єктивне підтвердження того, що розроблений веб-сервіс повністю відповідає вимогам, специфікаціям та архітектурним рішенням, визначеним на етапах аналізу та проектування. На відміну від тестування, яке фокусується на пошуку помилок, процедура верифікації спрямована на доведення коректності реалізації бізнес-логіки, оцінку продуктивності та надійності системи в умовах, наближених до реальної експлуатації. У рамках даної роботи верифікація

проводилася за трьома основними напрямами: перевірка відповідності функціональним вимогам, оцінка продуктивності клієнтської частини та аудит безпеки.

Першим кроком верифікації стало створення матриці відповідності (Traceability Matrix), яка дозволяє простежити зв’язок між сформульованими у першому розділі варіантами використання та фактично реалізованими функціями. Аналіз показав, що всі критичні вимоги, включаючи реєстрацію користувачів, завантаження та парсинг книг форматів PDF, відображення тексту з підтримкою пагінації та збереження прогресу читання, реалізовані у повному обсязі. Особливу увагу було приділено верифікації інтеграційних модулів: підтверджено, що система коректно взаємодіє з API Google Gemini, забезпечуючи отримання контекстного перекладу та відповідей на запити користувача протягом прийнятного часу очікування. Сценарії, що передбачали обробку помилок (наприклад, завантаження пошкодженого файлу або розрив з’єднання під час діалогу з чат-ботом), також пройшли перевірку, продемонструвавши здатність системи до відновлення працездатності без втрати даних.

Другим важливим етапом стала верифікація продуктивності клієнтської частини додатку, розробленої на NuxtJS. Для оцінки якості веб-інтерфейсу використовувався інструмент Google Lighthouse, який аналізує сторінки за показниками продуктивності, доступності та SEO-оптимізації. Результати аудиту головної сторінки та інтерфейсу рідера показали високі бали (понад 90 зі 100) за метрикою Performance. Час першого малювання контенту (FCP) склав менше 1.5 секунд, що є відмінним показником для SPA-додатку. Цього вдалося досягти завдяки використанню серверного рендерингу (SSR) та оптимізації розміру бандлів JavaScript. Верифікація швидкодії серверної частини проводилася шляхом вимірювання часу відгуку API. Середній час обробки запитів на отримання списку книг та метаданих не перевищує 200 мілісекунд. Операції, що вимагають взаємодії з зовнішніми нейромережами, демонструють затримку в межах 2–5 секунд, що є технологічним обмеженням API провайдера, проте реалізована

асинхронна модель взаємодії та візуалізація стану завантаження (скелетони, спінери) нівелюють негативний вплив на користувацький досвід.

Третім напрямом стала верифікація безпеки системи. Аудит підтвердив надійність механізму автентифікації на основі JWT-токенів: спроби доступу до захищених маршрутів без валідного токена успішно блокуються як на рівні клієнтського роутингу, так і на рівні серверних дозволів (Permissions). Перевірка механізму завантаження файлів засвідчила ефективність реалізованої санітизації імен файлів та валідації MIME-типів, що унеможливлює завантаження шкідливого виконуваного коду під виглядом електронних книг. Також верифіковано налаштування CORS (Cross-Origin Resource Sharing), які дозволяють приймати запити лише з довірених доменів, запобігаючи атакам типу CSRF.

Таким чином, комплексна верифікація програмної системи Neiread підтвердила її працездатність, стабільність та повну відповідність технічному завданню. Реалізований продукт забезпечує виконання всіх запланованих функцій, демонструє високу продуктивність та дотримується сучасних стандартів безпеки веб-застосунків, що дозволяє рекомендувати його до впровадження в промислову експлуатацію.

У третьому розділі проведено комплекс заходів щодо забезпечення якості та підготовки до експлуатації розробленого веб-сервісу. Визначено стратегію тестування. Розроблено та виконано набори тестових сценаріїв, що покривають критичні функціональні блоки. Результати тестування підтвердили коректність роботи системи та відсутність критичних дефектів. Встановлено критерії для програмно-апаратного середовища й подано опис механізму розгортання розробленого комплексу. Проведена верифікація підтвердила відповідність продукту функціональним вимогам, високі показники продуктивності та надійність захисту даних, що свідчить про успішне досягнення мети кваліфікаційної роботи.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

Забезпечення безпечних та нешкідливих умов праці є важливим аспектом організації виробничого процесу в сфері інформаційних технологій, де професійна діяльність пов’язана з тривалим використанням комп’ютерної техніки та високим нервово-емоційним навантаженням. Четвертий розділ кваліфікаційної роботи присвячено комплексному аналізу потенційних небезпек та шкідливих виробничих факторів, що супроводжують етапи розробки, тестування та експлуатації веб-сервісу. Метою розділу є обґрунтування та розробка системи інженерно-технічних та організаційних заходів, спрямованих на створення нормативних санітарно-гігієнічних умов на робочому місці програміста, забезпечення електробезпеки та пожежної безпеки. Окрему увагу приділено питанням цивільного захисту та плануванню дій персоналу у разі виникнення надзвичайних ситуацій техногенного або природного характеру, що дозволяє мінімізувати ризики для життя і здоров’я людей та забезпечити збереження матеріальних цінностей.

4.1 Охорона праці

Організація процесу розробки веб-сервісу Neiread, призначеного для читання та автоматичного перекладу книг, вимагала створення безпечного виробничого середовища, що відповідає сучасним стандартам ергономіки та гігієни праці. Специфіка роботи інженера-програміста в рамках даного проекту характеризувалася тривалою взаємодією з візуальними терміналами, високим рівнем концентрації уваги при написанні коду мовами Python і TypeScript, а також значним інтелектуальним навантаженням під час проектування архітектури системи. З огляду на це, пріоритетним завданням стало забезпечення умов праці,

які мінімізують вплив шкідливих виробничих факторів та запобігають розвитку професійних захворювань.

Основним нормативним документом, що регламентує безпеку праці та ергономіку робочого місця при розробці програмного забезпечення, є «Вимоги до безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», затверджені наказом Міністерства соціальної політики України. Відповідно до положень цього документу, робоче місце розробника було спроектовано так, щоб робочий стіл та крісло дозволяли займати зручну робочу позу та змінювати її протягом робочого дня, а сидіння стільця регулювалося по висоті [32]. Для роботи використовувався рідкокристалічний монітор з діагоналлю 27 дюймів, поверхня якого не створювала відблисків, а зображення було стабільним і не мало мерехтінь, що є обов’язковою умовою згідно з вимогами до екрана [32]. Екран розташовувався на відстані, що становила від 600 до 700 міліметрів від очей працівника, що відповідає нормативним вимогам для запобігання перенапруженню зорового аналізатора, а клавіатура розміщувалася так, щоб у користувача була опора для рук і передпліч [32].

Загальна організація безпеки праці на проекті базувалася на положеннях Закону України «Про охорону праці» [33]. Згідно зі статтею 13 цього Закону, роботодавець зобов’язаний створити на робочому місці умови праці відповідно до нормативно-правових актів, а також забезпечити додержання вимог законодавства щодо прав працівників у галузі охорони праці. Тому перед початком робіт було проведено вступний інструктаж з питань охорони праці, а саме робоче місце було організовано з урахуванням вимог електробезпеки: всі струмопровідні частини обладнання були надійно ізольовані, а корпуси комп'ютерної техніки — заземлені, що гарантувало захист від ураження електричним струмом під час експлуатації технічних засобів [33].

Ключовим фактором, що визначає рівень зорового комфорту при роботі з текстовим кодом та графічними інтерфейсами, є освітлення. Параметри світлового середовища на робочому місці були приведені у відповідність до державних будівельних норм ДБН В.2.5-28:2018 «Природне і штучне освітлення». У

приміщенні застосовувалася система комбінованого освітлення, де рівень освітленості робочої поверхні підтримувався в межах 500 люкс, що є нормативним показником для робіт високої точності, до яких належить робота з комп’ютером [34]. Для обмеження прямої блискучості від джерел світла світильники були розташовані так, щоб їхні світлові потоки не потрапляли безпосередньо в очі працівника. Крім того, світлодіодні джерела світла мали індекс кольоропередачі CRI понад 80, що забезпечувало коректне розрізнення кольорів, як того вимагають норми для приміщень з тривалим перебуванням людей [34].

Мікроклімат у приміщенні суттєво впливає на теплообмін організму та загальне самопочуття. Під час виконання роботи параметри мікроклімату, такі як температура, вологість та швидкість руху повітря, підтримувалися на рівні комфортних значень. Відповідно до «Вимог до безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», на робочих місцях було забезпечено дотримання значень параметрів мікроклімату згідно з чинними санітарними нормами, а надлишкове тепло, що виділяється обладнанням, не спричиняло дискомфорту працівникам.

Пожежна безпека на об’єкті розробки регламентувалася Кодексом цивільного захисту України. Оскільки приміщення насичене електронним обладнанням, основним заходом профілактики було дотримання правил експлуатації електроустановок для запобігання коротким замиканням. Робоче місце було забезпечене первинними засобами пожежогасіння, а саме вуглекислотним вогнегасником, придатним для гасіння електрообладнання під напругою. Відповідно до вимог законодавства, суб’єкт господарювання забезпечив виконання заходів пожежної безпеки, спрямованих на запобігання виникненню пожеж та створення умов для швидкої евакуації людей у разі небезпеки [35].

Окремим важливим аспектом охорони праці стала організація режиму праці та відпочинку для профілактики гіподинамії та зорової втоми. Згідно з «Вимогами до безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», роботодавець зобов’язаний планувати діяльність працівника так,

щоб щоденна робота з екраном періодично переривалася перервами або іншими видами діяльності, які не передбачають контакту з екранними пристроями. Тому щогодини робилися регламентовані перерви тривалістю 10–15 хвилин, під час яких виконувалися вправи для зняття напруги з очей та розвантаження опорно-рухового апарату, що дозволило зберегти високу працездатність протягом усього періоду виконання проекту.

4.2 Вплив виробничого середовища на працездатність та здоров’я користувачів комп’ютерів

В умовах сучасної інформаційної економіки професійна діяльність розробників програмного забезпечення характеризується формуванням специфічного виробничого середовища, в якому домінуючу роль відіграє взаємодія людини з електронно-обчислювальними машинами. Процес створення веб-сервісу Neiread передбачав тривалу роботу інженера-програміста за комп’ютером, що супроводжувалося впливом комплексу шкідливих та небезпечних факторів: від психофізіологічного напруження до електромагнітного випромінювання, проте їхня сукупна дія здатна викликати стійкі функціональні зрушення в організмі. Аналіз впливу виробничого середовища на здоров’я користувача є необхідною передумовою для розробки ефективних профілактичних заходів.

Першим і найбільш вираженим фактором, що визначає специфіку праці програміста, є високе напруження зорового аналізатора. Робота з програмним кодом, налагодження інтерфейсів та читання технічної документації вимагають розрізнення дрібних об’єктів (символів, графічних примітивів) в умовах світлового випромінювання екрана. Це призводить до інтенсивної експлуатації апарату акомодації та конвергенції ока. Відповідно до «Вимоги до безпеки та захисту здоров’я працівників під час роботи з екранними пристроями»,

затверджених наказом Міністерства соціальної політики України, така діяльність пов’язана з ризиком розвитку зорового та загального стомлення. Тривала фіксація погляду на екрані монітора викликає зниження частоти мимовільного моргання, що призводить до порушення стабільності слізної плівки та пересихання рогівки (синдром «сухого ока»). Крім того, постійна необхідність перефокусування погляду між екраном, клавіатурою та паперовими носіями викликає втому циліарного м’яза. У сукупності ці явища формують симптомокомплекс, відомий як комп’ютерний зоровий синдром (Computer Vision Syndrome), що проявляється затуманенням зору, відчуттям печіння в очах, двоїнням предметів, а також головним болем. Зазначений нормативний документ наголошує, що ігнорування режимів праці та відпочинку при такій роботі може призвести до незворотних змін зору, тому організація робочого процесу повинна передбачати регулярні перерви для розвантаження зорової системи [32].

Наступним вагомим блоком факторів впливу є ергономічні аспекти організації робочого місця та вимушена гіпокінезія (зниження рухової активності). Специфіка роботи вимагала сидячої пози протягом більшої частини робочого часу. Це створює значне статичне навантаження на опорно-руховий апарат, зокрема на хребетний стовп та м’язи тулуба. Згідно з положеннями національного стандарту ДСТУ EN ISO 9241-5:2018 «Ергономіка взаємодії людина-система», невідповідність параметрів робочого місця антропометричним даним користувача є ключовим фактором ризику виникнення скелетно-м’язових розладів. Тривале сидіння призводить до перерозподілу навантаження на міжхребцеві диски, особливо у поперековому відділі, де тиск може зростати у кілька разів порівняно з положенням стоячи. Це створює передумови для розвитку остеохондрозу, протрузій та гриж дисків. Крім того, статичне напруження м’язів шиї та плечового поясу, необхідне для утримання голови перед монітором, викликає погіршення кровопостачання головного мозку, що може проявлятися запамороченням та зниженням когнітивних функцій.

Окрему загрозу становить специфічне навантаження на верхні кінцівки. Висока інтенсивність роботи з клавіатурою та маніпулятором «миша» при

написанні коду передбачає виконання тисяч стереотипних рухів пальцями та кистю. Стандарт ДСТУ EN ISO 9241-5:2018 вказує, що при неправильному розташуванні пристроїв введення виникає тривале напруження сухожиль згиначів та розгиначів пальців, а також стиснення серединного нерва в зап'ястному каналі. Це є патогенетичною основою розвитку синдрому зап'ястного каналу (тунельного синдрому), який супроводжується болем, парестезіями (онімінням) пальців та слабкістю кисті. Застійні явища у венозній системі малого тазу та нижніх кінцівок, спричинені тривалим сидінням та перетисканням судин краєм стільця, також підвищують ризик розвитку судинних патологій. Таким чином, ергономічні недоліки робочого місця безпосередньо трансформуються у зниження фізичної працездатності та погіршення загального соматичного здоров'я розробника [36].

Третя група факторів, що впливають на здоров'я користувача комп'ютера, пов'язана з фізичними характеристиками виробничого середовища, серед яких особливе місце посідають електромагнітні випромінювання. Хоча сучасні рідкокристалічні дисплеї мають низький рівень випромінювання, системні блоки, блоки живлення, кабельні комунікації та серверне обладнання генерують електромагнітні поля (ЕМП) промислової частоти (50 Гц) та радіочастотного діапазону. Відповідно до «Державних санітарних норм і правил захисту населення від впливу електромагнітних випромінювань», людина, яка працює з комп'ютерною технікою, піддається хронічному впливу цих полів. Біологічна дія ЕМП на організм є складною та багатогранною. Найбільш чутливими до електромагнітного впливу є нервова, імунна, ендокринна та репродуктивна системи. Під дією електромагнітних полів можливий головний біль, підвищена дратівливості, порушення сну, зниження пам'яті та концентрації уваги.

Крім того, робота комп'ютерної техніки впливає на аеройонний склад повітря у приміщенні. Електростатичні поля, що утворюються на корпусах обладнання та екранах, сприяють осадженню легких негативних аеройонів, які є біологічно корисними, і збільшенню концентрації важких позитивних йонів. Це явище, відоме як «електричне забруднення» повітря, може призводити до зниження оксигенації крові, погіршення легеневого газообміну та зниження

імунітету. Зазначені санітарні норми регламентують гранично допустимі рівні напруженості електричного та магнітного полів, перевищення яких є неприпустимим, оскільки це створює пряму загрозу здоров’я персоналу. Таким чином, забезпечення електромагнітної безпеки є важливою умовою збереження працездатності фахівця [37].

Психофізіологічний аспект впливу виробничого середовища також не можна ігнорувати. Інтелектуальна праця розробника пов’язана з переробкою великих обсягів інформації. Це призводить до значного нервово-емоційного напруження. Монотонність окремих операцій, сенсорна ізоляція (зосередженість на віртуальному об’єкті праці) та інформаційні перевантаження можуть викликати стан стресу. Хронічний стрес, у поєднанні з описаними вище фізичними факторами, виснажує адаптаційні ресурси організму, що проявляється у зниженні продуктивності праці, збільшенні кількості помилок у коді та погіршенні психологічного клімату в колективі.

Комплексний вплив вищезазначених факторів формує загальний рівень працездатності користувача комп’ютера. Дослідження показують, що при роботі в несприятливих умовах продуктивність праці програміста може суттєво знижуватися вже через кілька годин роботи. Тому створення оптимального виробничого середовища є не лише вимогою законодавства про охорону праці, але й економічно обґрунтованою необхідністю. Економічна ефективність цих робіт досягається за узгодженості із завданнями щодо забезпечення безаварійної роботи, покращенні умов праці, вдосконаленні виробничого процесу [38, 39].

Підсумовуючи вплив виробничого середовища на користувачів комп’ютерів носить комплексний характер і зачіпає практично всі системи організму. Забезпечення високої працездатності та збереження здоров’я можливе лише за умови системного підходу до охорони праці, що включає дотримання вимог щодо режиму роботи з екранними пристроями, впровадження ергономічних стандартів організації робочого місця та контроль фізичних факторів середовища відповідно до санітарних норм. Ігнорування цих аспектів неминуче призводить до зниження ефективності професійної діяльності та економічних втрат.

ВИСНОВКИ

У рамках виконання кваліфікаційної роботи магістра було спроєктовано та програмно реалізовано веб-сервіс для читання та автоматичного перекладу книг, побудований на базі технологічного стека NuxtJS та Django. Розроблена інформаційна система відповідає сучасним вимогам до веб-додатків, забезпечуючи високу продуктивність, масштабованість та зручність користування.

У ході дослідження предметної області здійснено комплексний аналіз проблем, що виникають при читанні іншомовної літератури, зокрема проблеми перемикання контексту при використанні сторонніх словників. На основі порівняльного аналізу існуючих рішень виявлено потребу у створенні інтегрованого середовища, що поєднує функціонал рідера та інтелектуального асистента. Сформульовано функціональні та нефункціональні вимоги до системи, визначено ролі акторів та побудовано діаграми варіантів використання, що стало основою для подальшого проектування.

Процес розробки базувався на гнучкій методології Agile (Scrum), що дозволило оптимізувати робочий час та забезпечити поетапне впровадження функціоналу. Спроєктовано клієнт-серверну архітектуру системи із використанням REST API для взаємодії компонентів. Розроблено реляційну схему бази даних PostgreSQL, яка забезпечує ефективне зберігання контенту, метаданих користувачів та історії діалогів. Побудовано UML-діаграми класів, що дозволило формалізувати внутрішню структуру серверної частини.

Програмна реалізація серверної частини виконана на фреймворку Django. Реалізовано механізми автентифікації користувачів через електронну пошту, логіку завантаження та конвертації файлів PDF у текстовий формат Markdown для подальшої обробки. Ключовим досягненням стала інтеграція з великою мовною моделлю Google Gemini 2.5 Flash, що дозволило реалізувати функції контекстного перекладу та інтелектуального чат-бота, здатного пояснювати складні фрагменти тексту з урахуванням контенту книги.

Клієнтську частину розроблено з використанням фреймворку NuxtJS. Створено адаптивний інтерфейс користувача, що включає особистий кабінет, бібліотеку книг та ергономічний рідер. Реалізовано інтерактивні елементи взаємодії з текстом: виділення фрагментів для миттєвого перекладу та комунікації з AI-асистентом. Забезпечено синхронізацію прогресу читання між різними сесіями.

Проведено комплексне тестування системи, що включало написання автоматизованих модульних тестів для перевірки бізнес-логіки бекенду та ручне тестування інтерфейсу користувача. Верифікація підтвердила відповідність розробленого продукту технічному завданню. Розроблено конфігурацію для розгортання системи у контейнеризованому середовищі Docker, що спрощує процес впровадження та підтримки. Також проаналізовано умови праці розробника та визначено заходи щодо забезпечення безпеки життєдіяльності.

Результати кваліфікаційної роботи демонструють створення повнофункціонального програмного продукту, що має практичну цінність для осіб, які вивчають іноземні мови. Система поєднує передові веб-технології з можливостями штучного інтелекту, вирішуючи актуальну проблему мовного бар’єра та підвищуючи ефективність самоосвіти.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Морська Л. І. Інформаційні технології у навчанні іноземних мов : навчальний посібник. Тернопіль : Астон, 2015. 256 с.

2. Карпюк О. А., Бовсунівська Т. В. Особливості використання систем машинного перекладу в процесі вивчення іноземної мови студентами технічних спеціальностей. Вісник Житомирського державного університету імені Івана Франка. Педагогічні науки. 2019. Вип. 2 (97). С. 45–50.

3. Грицюк Ю. І., Рак Т. Є. Об’єктно-орієнтоване моделювання програмних систем : навч. посіб. Львів : Вид-во Львівського держ. ун-ту безпеки життєдіяльності, 2018. 326 с.

4. Левченко О. М. Основи проектування інформаційних систем : навчальний посібник. Київ : КПІ ім. Ігоря Сікорського, 2019. 215 с.

5. Блінов І. О., Романенко В. А. Розробка корпоративних інформаційних систем : підручник. Київ : Нац. авіац. ун-т, 2020. 360 с.

6. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2016. 810 p.

7. Layton M. C., Ostermiller S. J., Kynaston D. J. Agile Project Management For Dummies. 3rd ed. Hoboken : Wiley, 2020. 432 p.

8. Martin R. C. Clean Agile: Back to Basics. Boston : Prentice Hall, 2019. 240 p.

9. Waterfall vs. Agile: What is the best approach for a software development project? [Електронний ресурс] // Easy Redmine. – Режим доступу: https://www.easyredmine.com/news/waterfall-vs-agile-what-is-the-best-approach-for-a-software-development-project. – Дата звернення: 12.11.2025.

10. Bass L., Clements P., Kazman R. Software Architecture in Practice. 3rd ed. Upper Saddle River : Addison-Wesley Professional, 2012. 624 p.

11. Freeman A. Pro Vue.js 2. New York : Apress, 2018. 835 p.

12. Richardson L., Ruby S. RESTful Web Services. Sebastopol : O’Reilly Media, 2007. 448 p.

13. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O'Reilly Media, 2021. 614 p.

14. Web Application Architecture [Електронний ресурс] // Intellectsoft. – Режим доступу: https://www.intellectsoft.net/blog/web-application-architecture/. – Дата звернення: 15.11.2025.

15. Connolly T., Begg C. Database Systems: A Practical Approach to Design, Implementation, and Management. 6th ed. Boston : Pearson, 2014. 1440 p.

16. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. 2nd ed. Upper Saddle River : Addison-Wesley Professional, 2005. 496 p.

17. Lutz M. Learning Python. 5th ed. Sebastopol : O'Reilly Media, 2013. 1600 p.

18. Vincent W. Django for APIs: Build web APIs with Python and Django. Amazon Digital Services, 2022. 254 p.

19. Schönig H. Mastering PostgreSQL 13. 4th ed. Birmingham : Packt Publishing, 2020. 498 p.

20. Bierman G., Abadi M., Torgersen M. Understanding TypeScript [Електронний ресурс] // Microsoft Research. – Режим доступу: https://www.typescriptlang.org/docs/. – Дата звернення: 17.11.2025.

21. Nuxt Documentation [Електронний ресурс] // NuxtLabs. – Режим доступу: https://nuxt.com/docs/getting-started/introduction. – Дата звернення: 17.11.2025.

22. Cursor - The AI Code Editor [Електронний ресурс] // Cursor. – Режим доступу: https://cursor.sh/. – Дата звернення: 17.11.2025.

23. Gemini 2.5 Technical Report: Capabilities and Performance on Text-Rich Tasks [Електронний ресурс] // Google DeepMind. – Режим доступу: https://deepmind.google/technologies/gemini/flash/. – Дата звернення: 17.11.2025.

24. Nuxt 3 Documentation [Електронний ресурс] // NuxtLabs. – Режим доступу: https://nuxt.com/docs/getting-started/introduction. – Дата звернення: 20.11.2025.

25. Vue PDF Viewer Documentation [Електронний ресурс] // Vue PDF Viewer. – Режим доступу: https://docs.vue-pdf-viewer.dev/. – Дата звернення: 20.11.2025.

26. Pinia: The Vue Store [Електронний ресурс] // Vue.js. – Режим доступу: https://pinia.vuejs.org/introduction.html. – Дата звернення: 24.11.2025.

27. Tailwind CSS Documentation [Електронний ресурс] // Tailwind Labs. – Режим доступу: https://tailwindcss.com/docs. – Дата звернення: 24.11.2025.

28. MDN Web Docs: Selection API [Електронний ресурс] // Mozilla. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/Selection. – Дата звернення: 24.11.2025.

29. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. Hoboken : John Wiley & Sons, 2011. 240 p.

30. Khorikov V. Unit Testing Principles, Practices, and Patterns. Shelter Island : Manning Publications, 2020. 275 p.

31. Patton R. Software Testing. 2nd ed. Indianapolis : Sams Publishing, 2005. 408 p.

32. Вимоги до безпеки та захисту здоров’я працівників під час роботи з екранними пристроями : затв. наказом М-ва соц. політики України від 14.02.2018 № 207. URL: https://zakon.rada.gov.ua/laws/show/z0508-18 (дата звернення: 01.12.2025).

33. Про охорону праці : Закон України від 14.10.1992 № 2694-XII : станом на 01 січ. 2025 р. / Верховна Рада України. URL: https://zakon.rada.gov.ua/laws/show/2694-12 (дата звернення: 01.12.2025).

34. ДБН В.2.5-28:2018. Природне і штучне освітлення. Київ : Мінрегіон України, 2018. 133 с.

35. Кодекс цивільного захисту України : Закон України від 02.10.2012 № 5403-VI : станом на 01 січ. 2025 р. / Верховна Рада України. URL: https://zakon.rada.gov.ua/laws/show/5403-17 (дата звернення: 01.12.2025).

36. ДСТУ EN ISO 9241-5:2018 (EN ISO 9241-5:1999, IDT; ISO 9241-5:1998, IDT). Ергономічні вимоги до роботи з відеотерміналами в офісі. Частина 5. Вимоги до розташування робочого місця та постави. Київ : ДП «УкрНДНЦ», 2018. 28 с.

37. Державні санітарні норми і правила захисту населення від впливу електромагнітних випромінювань : затв. наказом М-ва охорони здоров’я України від 01.08.1996 № 239 : у редакції наказу МОЗ України від 03.04.2017 № 356. URL: https://zakon.rada.gov.ua/laws/show/z0488-96 (дата звернення: 08.12.2025).

38. Стручок В.С. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання «БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ» / Тернопіль: ФОП Паляниця В. А., –156 с. Отримано з https://elartu.tntu.edu.ua/handle/lib/39196.

39. Стручок В.С. Навчальний посібник «ТЕХНОЕКОЛОГІЯ ТА ЦИВІЛЬНА БЕЗПЕКА. ЧАСТИНА «ЦИВІЛЬНА БЕЗПЕКА»» / Тернопіль: ФОП Паляниця В. А., – 156 с. Отримано з http://elartu.tntu.edu.ua/handle/lib/39424

40. Методичні вказівки до виконання кваліфікаційної роботи магістра для здобувачів спеціальності 121 – Інженерія програмного забезпечення, всіх форм навчання / укладачі Михалик Д.М., Цуприк Г.Б., Бревус В.М., Мудрик І.Я. – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. – 44 с.

ДОДАТКИ

ДОДАТОК А

Тези для XIII науково-технічна конференція «Інформаційні моделі, системи та технології»

УДК 004.41
Павло Грицишин – ст. гр. СПм-61, Юрій Стоянов к.т.н., доц. каф. ПІ
Тернопільський національний технічний університет імені Івана Пулюя

Розробка сервісу для читання та автоматичного перекладу книг з використанням технологій NuxtJS та Django

Pavlo Hrytsyshyn, Yurii Stoianov PhD
Development of a service for reading and automatically translating books using NuxtJS and Django technologies

Ключові слова: веб-розробка, LLM, реактивність, NuxtJS, література.
Keywords: web development, LLM, reactivity, NuxtJS, literature.

В умовах глобалізації ефективна робота з іншомовними матеріалами стає визначальним фактором професійного зростання. Це актуалізує потребу у створенні програмних комплексів, що поєднують функціонал електронних рідерів та систем автоматизованого перекладу. Оптимальним технологічним рішенням для розробки клієнтської частини таких сервісів є використання NuxtJS. Цей інструмент базується на екосистемі Vue.js та дозволяє створювати універсальні веб-додатки з високими показниками продуктивності [1].

При інтеграції з великими мовними моделями через API ключового значення набуває реактивність інтерфейсу. Завдяки архітектурі NuxtJS система забезпечує миттєве оновлення контенту без необхідності повного перезавантаження сторінки. Використання даного фреймворку також дозволяє реалізувати серверний рендеринг, що є критично важливим для оптимізації швидкості завантаження текстових масивів книг.

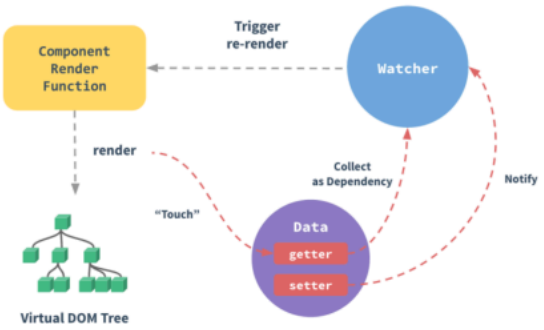

Рисунок 1 – Схема роботи реактивності[2].

Поєднання гнучкого фронтенду на Nuxt із потужним бекендом на Django формує стійку до навантажень архітектуру, яка гарантує стабільність та високу швидкодію програмного продукту.

Література

1. Introduction to Nuxt [Електронний ресурс] – Режим доступу: https://nuxt.com/docs/getting-started/introduction.
2. The Best Explanation of JavaScript Reactivity [Електронний ресурс] – Режим допступу: https://medium.com/vue-mastery/the-best-explanation-of-javascript-reactivity-fea6112dd80d

ДОДАТОК Б
ДІАГРАМИ ТА РИСУНКИ

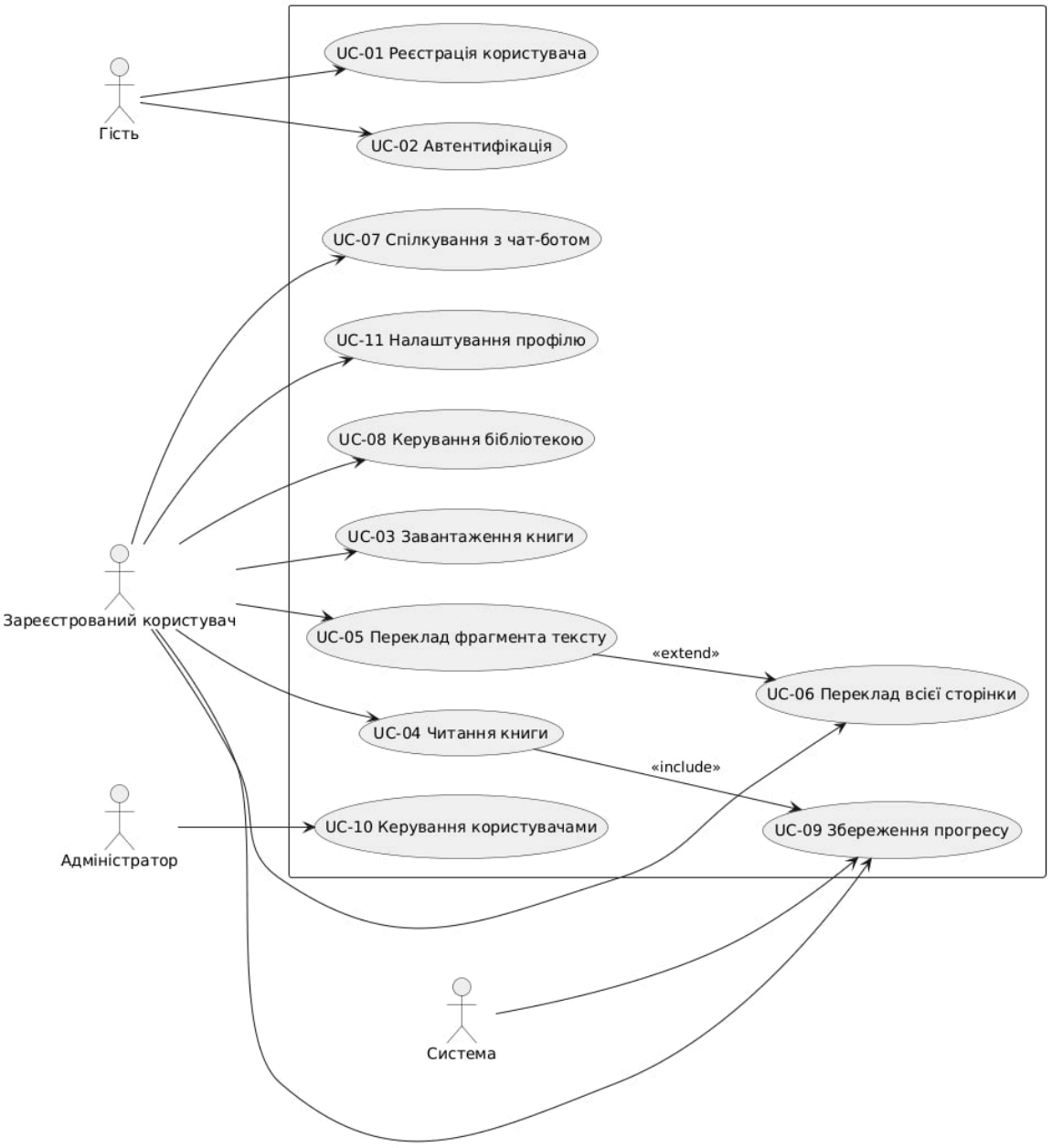

Рисунок Б.1 – Діаграмі варіантів використання

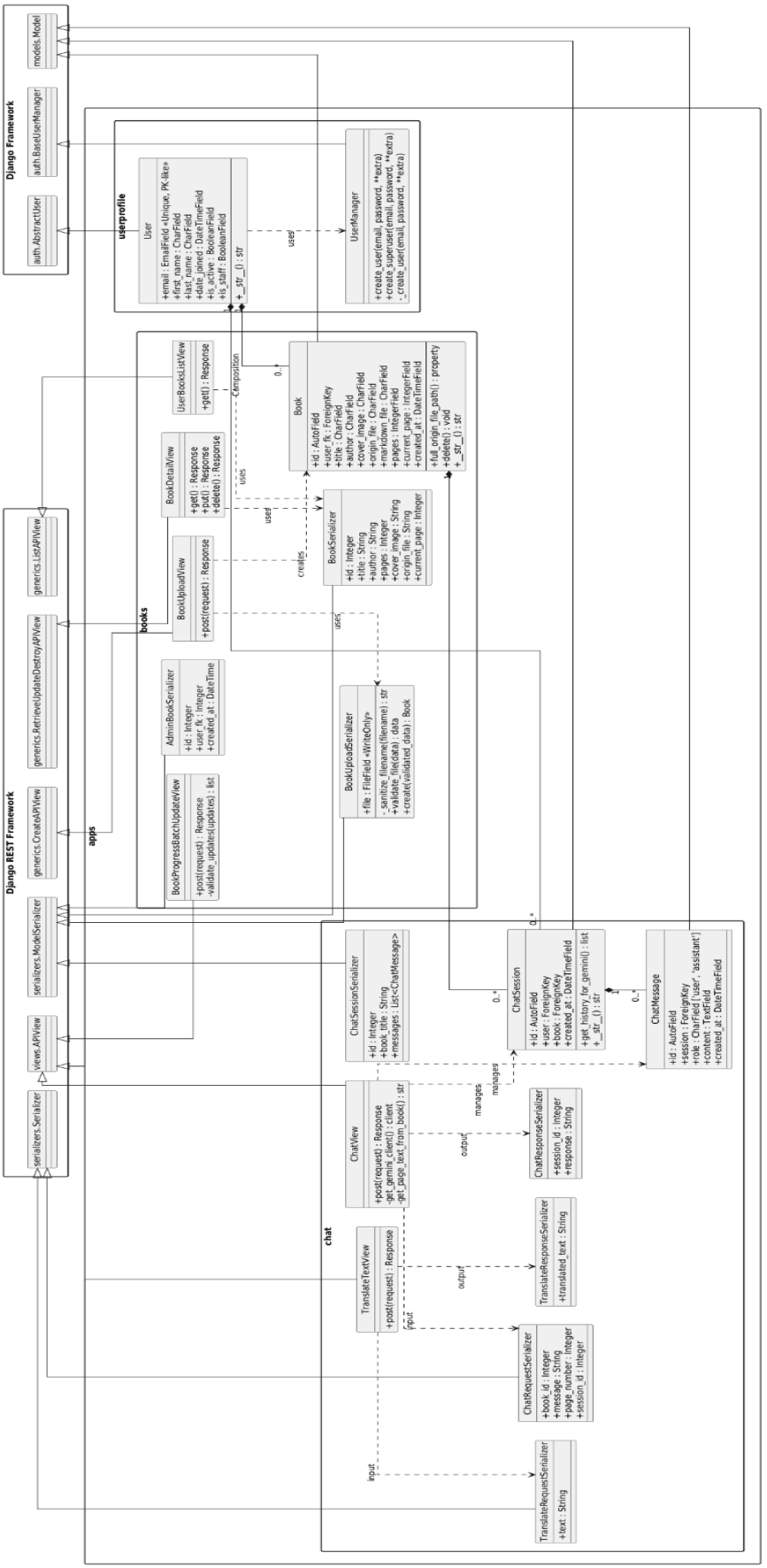

Рисунок Б.2 – Діаграма класів.

ДОДАТОК В
РЕПОЗИТОРІЙ

1. Репозиторій клієнтської частини. URL: https://github.com/zerrty/Neiread.
2. Репозиторій серверної частини. URL: https://github.com/zerrty/Neiread_backend.